```
Archive-name: ai-faq/neural-nets/part1
Last-modified: 2002-05-17
URL: ftp://ftp.sas.com/pub/neural/FAQ.html
Maintainer: saswss@unx.sas.com (Warren S. Sarle)
```



```
  ------------------------------------------------------------
    Additions, corrections, or improvements are always welcome.
    Anybody who is willing to contribute any information,
    please email me; if it is relevant, I will incorporate it.

    The monthly posting departs around the 28th of every month.
  ------------------------------------------------------------
```

This is the first of seven parts of a monthly posting to the Usenet newsgroup comp.ai.neural-nets (as well as comp.answers and news.answers, where it should be findable at any time). Its purpose is to provide basic information for individuals who are new to the field of neural networks or who are just beginning to read this group. It will help to avoid lengthy discussion of questions that often arise for beginners.

```
   SO, PLEASE, SEARCH THIS POSTING FIRST IF YOU HAVE A QUESTION
                           and
   DON'T POST ANSWERS TO FAQs: POINT THE ASKER TO THIS POSTING
```

The latest version of the FAQ is available as a hypertext document, readable by any WWW (World Wide Web) browser such as Netscape, under the URL: ftp://ftp.sas.com/pub/neural/FAQ.html.

If you are reading the version of the FAQ posted in comp.ai.neural-nets, be sure to view it with a monospace font such as Courier. If you view it with a proportional font, tables and formulas will be mangled. Some newsreaders or WWW news services garble plain text. If you have trouble viewing plain text, try the HTML version described above.

All seven parts of the FAQ can be downloaded from either of the following URLS:

ftp://ftp.sas.com/pub/neural/FAQ.html.zip
ftp://ftp.sas.com/pub/neural/FAQ.txt.zip

These postings are archived in the periodic posting archive on host rtfm.mit.edu (and on some other hosts as well). Look in the anonymous ftp directory "/pub/usenet/news.answers/ai-faq/neural-nets" under the file names "part1", "part2", ... "part7". If you do not have anonymous ftp access, you can access the archives by mail server as well. Send an E-mail message to mail-server@rtfm.mit.edu with "help" and "index" in the body on separate lines for more information.

For those of you who read this FAQ anywhere other than in Usenet: To read comp.ai.neural-nets (or post articles to it) you need Usenet News access. Try the commands, 'xrn', 'rn', 'nn', or 'trn' on your Unix machine, 'news' on your VMS machine, or ask a local guru. WWW browsers are often set up for Usenet access, too--try the URL news:comp.ai.neural-nets.

The FAQ posting departs to comp.ai.neural-nets around the 28th of every month. It is also sent to the groups comp.answers and news.answers where it should be available at any time (ask your news manager). The FAQ posting, like any other posting, may a take a few days to find its way over Usenet to your site. Such delays are especially common outside of North America.

All changes to the FAQ from the previous month are shown in another monthly posting having the subject `changes to "comp.ai.neural-nets FAQ" -- monthly posting', which immediately follows the FAQ posting. The `changes' post contains the full text of all changes and can also be found at ftp://ftp.sas.com/pub/neural/changes.txt . There is also a weekly post with the subject "comp.ai.neural-nets FAQ: weekly reminder" that briefly describes any major changes to the FAQ.

This FAQ is not meant to discuss any topic exhaustively. It is neither a tutorial nor a textbook, but should be viewed as a supplement to the many excellent books and online resources described in Part 4: Books, data, etc..

Disclaimer:

> This posting is provided 'as is'. No warranty whatsoever is expressed or implied, in particular, no warranty that the information contained herein is correct or useful in any way, although both are intended.

To find the answer of question "x", search for the string "Subject: x"

========== Questions ==========

Part 1: Introduction

Subject: What is this newsgroup for? How shall it be used?

The newsgroup comp.ai.neural-nets is intended as a forum for people who want to use or explore the capabilities of Artificial Neural Networks or Neural-Network-like structures.

Posts should be in plain-text format, not postscript, html, rtf, TEX, MIME, or any word-processor format.

Do not use vcards or other excessively long signatures.

Please do not post homework or take-home exam questions. Please do not post a long source-code listing and ask readers to debug it. And note that chain letters and other get-rich-quick pyramid schemes are illegal in the USA; for example, see http://www.usps.gov/websites/depart/inspect/chainlet.htm

There should be the following types of articles in this newsgroup:

1. ## Requests

   Requests are articles of the form `"I am looking for X"`, where X is something public like a book, an article, a piece of software. The most important about such a request is to be as specific as possible!

   If multiple different answers can be expected, the person making the request should prepare to make a summary of the answers he/she got and announce to do so with a phrase like `"Please reply by email, I'll summarize to the group"` at the end of the posting.

   The Subject line of the posting should then be something like `"Request: X"`

2. ## Questions

   As opposed to requests, questions ask for a larger piece of information or a more or less detailed explanation of something. To avoid lots of redundant traffic it is important that the poster provides with the question all information s/he already has about the subject asked and state the actual question as precise and narrow as possible. The poster should prepare to make a summary of the

answers s/he got and announce to do so with a phrase like `"Please reply by email, I'll summarize to the group"` at the end of the posting.

The Subject line of the posting should be something like `"Question: this-and-that"` or have the form of a question (i.e., end with a question mark)

Students: please do not ask comp.ai.neural-net readers to do your homework or take-home exams for you.

3. Answers

These are reactions to questions or requests. If an answer is too specific to be of general interest, or if a summary was announced with the question or request, the answer should be e-mailed to the poster, not posted to the newsgroup.

Most news-reader software automatically provides a subject line beginning with "Re:" followed by the subject of the article which is being followed-up. Note that sometimes longer threads of discussion evolve from an answer to a question or request. In this case posters should change the subject line suitably as soon as the topic goes too far away from the one announced in the original subject line. You can still carry along the old subject in parentheses in the form `"Re: new subject (was: old subject)"`

4. Summaries

In all cases of requests or questions the answers for which can be assumed to be of some general interest, the poster of the request or question shall summarize the answers he/she received. Such a summary should be announced in the original posting of the question or request with a phrase like `"Please answer by email, I'll summarize"`

In such a case, people who answer to a question should NOT post their answer to the newsgroup but instead mail them to the poster of the question who collects and reviews them. After about 5 to 20 days after the original posting, its poster should make the summary of answers and post it to the newsgroup.

Some care should be invested into a summary:

- simple concatenation of all the answers is not enough: instead, redundancies, irrelevancies, verbosities, and errors should be filtered out (as well as possible)
- the answers should be separated clearly
- the contributors of the individual answers should be identifiable (unless they requested to remain anonymous [yes, that happens])
- the summary should start with the "quintessence" of the answers, as seen by the original poster
- A summary should, when posted, clearly be indicated to be one by giving it a Subject line starting with "SUMMARY:"

Note that a good summary is pure gold for the rest of the newsgroup community, so summary work will be most appreciated by all of us. Good summaries are more valuable than any moderator ! :-)

5. ## Announcements

   Some articles never need any public reaction. These are called announcements (for instance for a workshop, conference or the availability of some technical report or software system).

   Announcements should be clearly indicated to be such by giving them a subject line of the form `"Announcement: this-and-that"`

6. ## Reports

   Sometimes people spontaneously want to report something to the newsgroup. This might be special experiences with some software, results of own experiments or conceptual work, or especially interesting information from somewhere else.

   Reports should be clearly indicated to be such by giving them a subject line of the form `"Report: this-and-that"`

7. ## Discussions

   An especially valuable possibility of Usenet is of course that of discussing a certain topic with hundreds of potential participants. All traffic in the newsgroup that can not be subsumed under one of the above categories should belong to a discussion.

   If somebody explicitly wants to start a discussion, he/she can do so by giving the posting a subject line of the form `"Discussion: this-and-that"`

   It is quite difficult to keep a discussion from drifting into chaos, but, unfortunately, as many many other newsgroups show there seems to be no secure way to avoid this. On the other hand, comp.ai.neural-nets has not had many problems with this effect in the past, so let's just go and hope...

8. ## Jobs Ads

   Advertisements for jobs requiring expertise in artificial neural networks are appropriate in comp.ai.neural-nets. Job ads should be clearly indicated to be such by giving them a subject line of the form `"Job: this-and-that".` It is also useful to include the country-state-city abbreviations that are conventional in misc.jobs.offered, such as: `"Job: US-NY-NYC Neural network engineer".` If an employer has more than one job opening, all such openings should be listed in a single post, not multiple posts. Job ads should not be reposted more than once per month.

Subject: Where is comp.ai.neural-nets archived?

The following archives are available for comp.ai.neural-nets:

- http://groups.google.com, formerly Deja News. Does not work very well yet.
- 94-09-14 through 97-08-16 ftp://ftp.cs.cmu.edu/user/ai/pubs/news/comp.ai.neural-nets

For more information on newsgroup archives, see
http://starbase.neosoft.com/~claird/news.lists/newsgroup_archives.html
or http://www.pitt.edu/~grouprev/Usenet/Archive-List/newsgroup_archives.html

---

Subject: What if my question is not answered in the FAQ?

If your question is not answered in the FAQ, you can try a web search. The following search engines are especially useful:
http://www.google.com/
http://search.yahoo.com/
http://www.altavista.com/
http://citeseer.nj.nec.com/cs

Another excellent web site on NNs is Donald Tveter's Backpropagator's Review at http://www.dontveter.com/bpr/bpr.html or http://gannoo.uce.ac.uk/bpr/bpr.html.

For feedforward NNs, the best reference book is:

> Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press.

If the answer isn't in Bishop, then for more theoretical questions try:

> Ripley, B.D. (1996) *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.

For more practical questions about MLP training, try:

> Masters, T. (1993). *Practical Neural Network Recipes in C++*, San Diego: Academic Press.

> Reed, R.D., and Marks, R.J, II (1999), *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*, Cambridge, MA: The MIT Press.

There are many more excellent books and web sites listed in the Neural Network FAQ, Part 4: Books, data, etc.

- -

Subject: May I copy this FAQ?

The intent in providing a FAQ is to make the information freely available to whoever needs it. You may copy all or part of the FAQ, but please be sure to include a reference to the URL of the master copy, ftp://ftp.sas.com/pub/neural/FAQ.html, and do not sell copies of the FAQ. If you want to include information from the FAQ in your own web site, it is better to include links to the master copy rather than to copy text from the FAQ to your web pages, because various answers in the FAQ are updated at unpredictable times. To cite the FAQ in an academic-style bibliography, use something along the lines of:

> Sarle, W.S., ed. (1997), *Neural Network FAQ, part 1 of 7: Introduction*, periodic posting to the Usenet newsgroup comp.ai.neural-nets, URL: ftp://ftp.sas.com/pub/neural/FAQ.html

- -

Subject: What is a neural network (NN)?

> The question 'What is a neural network?' is ill-posed.
> - Pinkus (1999)

First of all, when we are talking about a neural network, we should more properly say "artificial neural network" (ANN), because that is what we mean most of the time in comp.ai.neural-nets. Biological neural networks are much more complicated than the mathematical models we use for ANNs. But it is customary to be lazy and drop the "A" or the "artificial".

There is no universally accepted definition of an NN. But perhaps most people in the field would agree that an NN is a network of many simple processors ("units"), each possibly having a small amount of local memory. The units are connected by communication channels ("connections") which usually carry numeric (as opposed to symbolic) data, encoded by any of various means. The units operate only on their local data and on the inputs they receive via the connections. The restriction to local operations is often relaxed during training.

Some NNs are models of biological neural networks and some are not, but historically, much of the inspiration for the field of NNs came from the desire to produce artificial systems capable of sophisticated, perhaps "intelligent", computations similar to those that the human brain routinely performs, and thereby possibly to enhance our understanding of the human brain.

Most NNs have some sort of "training" rule whereby the weights of connections are adjusted on the basis of data. In other words, NNs "learn" from examples, as children learn to distinguish dogs from cats based on examples of dogs and cats. If trained carefully, NNs may exhibit some capability for generalization beyond the training data, that is, to produce approximately correct results for new cases that were not used for training.

NNs normally have great potential for parallelism, since the computations of the components are largely independent of each other. Some people regard massive parallelism and high connectivity to be defining

characteristics of NNs, but such requirements rule out various simple models, such as simple linear regression (a minimal feedforward net with only two units plus bias), which are usefully regarded as special cases of NNs.

Here is a sampling of definitions from the books on the FAQ maintainer's shelf. None will please everyone. Perhaps for that reason many NN textbooks do not explicitly define neural networks.

According to the *DARPA Neural Network Study* (1988, AFCEA International Press, p. 60):

> ... a neural network is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes.

According to Haykin (1994), p. 2:

> A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:
>
> 1. Knowledge is acquired by the network through a learning process.
> 2. Interneuron connection strengths known as synaptic weights are used to store the knowledge.

According to Nigrin (1993), p. 11:

> A neural network is a circuit composed of a very large number of simple processing elements that are neurally based. Each element operates only on local information. Furthermore each element operates asynchronously; thus there is no overall system clock.

According to Zurada (1992), p. xv:

> Artificial neural systems, or neural networks, are physical cellular systems which can acquire, store, and utilize experiential knowledge.

References:

Pinkus, A. (1999), "Approximation theory of the MLP model in neural networks," Acta Numerica, 8, 143-196.

Haykin, S. (1994), *Neural Networks: A Comprehensive Foundation*, NY: Macmillan.

Nigrin, A. (1993), *Neural Networks for Pattern Recognition*, Cambridge, MA: The MIT Press.

Zurada, J.M. (1992), *Introduction To Artificial Neural Systems*, Boston: PWS Publishing Company.

Subject: Where can I find a simple introduction to NNs?

Several excellent introductory books on NNs are listed in part 4 of the FAQ under "Books and articles about Neural Networks?" If you want a book with minimal math, see "The best introductory book for business executives."

Dr. Leslie Smith has a brief on-line introduction to NNs with examples and diagrams at http://www.cs.stir.ac.uk/~lss/NNIntro/InvSlides.html.

If you are a Java enthusiast, see Jochen Fröhlich's diploma at http://rfhs8012.fh-regensburg.de/~saj39122/jfroehl/diplom/e-index.html

For a more detailed introduction, see Donald Tveter's excellent Backpropagator's Review at http://www.dontveter.com/bpr/bpr.html or http://gannoo.uce.ac.uk/bpr/bpr.html, which contains both answers to additional FAQs and an annotated neural net bibliography emphasizing on-line articles.

StatSoft Inc. has an on-line *Electronic Statistics Textbook*, at http://www.statsoft.com/textbook/stathome.html that includes a chapter on neural nets as well as many statistical topics relevant to neural nets.

Subject: Are there any online books about NNs?

Kevin Gurney has on-line a preliminary draft of his book, *An Introduction to Neural Networks*, at http://www.shef.ac.uk/psychology/gurney/notes/index.html The book is now in print and is one of the better general-purpose introductory textbooks on NNs. Here is the table of contents from the on-line version:

1. Computers and Symbols versus Nets and Neurons
2. TLUs and vectors - simple learning rules
3. The delta rule
4. Multilayer nets and backpropagation
5. Associative memories - the Hopfield net
6. Hopfield nets (contd.)
7. Kohonen nets
8. Alternative node types
9. Cubic nodes (contd.) and Reward Penalty training
10. Drawing things together - some perspectives

Another on-line book by Ben Kröse and Patrick van der Smagt, also called *An Introduction to Neural Networks*, can be found at ftp://ftp.wins.uva.nl/pub/computer-systems/aut-sys/reports/neuro-intro/neuro-

intro.ps.gz or http://www.robotic.dlr.de/Smagt/books/neuro-intro.ps.gz. or http://www.supelec-rennes.fr/acth/net/neuro-intro.ps.gz
Here is the table of contents:

Subject: What can you do with an NN and what not?

In principle, NNs can compute any computable function, i.e., they can do everything a normal digital computer can do (Valiant, 1988; Siegelmann and Sontag, 1999; Orponen, 2000; Sima and Orponen, 2001), or perhaps even more, under some assumptions of doubtful practicality (see Siegelmann, 1998, but also Hadley, 1999).

Practical applications of NNs most often employ supervised learning. For supervised learning, you must provide training data that includes both the input and the desired result (the target value). After successful training, you can present input data alone to the NN (that is, input data without the desired result), and the NN will compute an output value that approximates the desired result. However, for training to be successful, you may need lots of training data and lots of computer time to do the training. In many applications, such as image and text processing, you will have to do a lot of work to select appropriate input data and to code the data as numeric values.

In practice, NNs are especially useful for classification and function approximation/mapping problems which are tolerant of some imprecision, which have lots of training data available, but to which hard and fast rules (such as those that might be used in an expert system) cannot easily be applied. Almost any finite-dimensional vector function on a compact set can be approximated to arbitrary precision by feedforward NNs (which are the type most often used in practical applications) if you have enough data and enough computing resources.

To be somewhat more precise, feedforward networks with a single hidden layer and trained by least-squares are statistically consistent estimators of arbitrary square-integrable regression functions under certain practically-satisfiable assumptions regarding sampling, target noise, number of hidden units, size of weights, and form of hidden-unit activation function (White, 1990). Such networks can also be trained as statistically consistent estimators of derivatives of regression functions (White and Gallant, 1992) and quantiles of the conditional noise distribution (White, 1992a). Feedforward networks with a single hidden layer using threshold or sigmoid activation functions are universally consistent estimators of binary

classifications (Faragó and Lugosi, 1993; Lugosi and Zeger 1995; Devroye, Györfi, and Lugosi, 1996) under similar assumptions. Note that these results are stronger than the universal approximation theorems that merely show the existence of weights for arbitrarily accurate approximations, without demonstrating that such weights can be obtained by learning.

Unfortunately, the above consistency results depend on one impractical assumption: that the networks are trained by an error (`L_p` error or misclassification rate) minimization technique that comes arbitrarily close to the global minimum. Such minimization is computationally intractable except in small or simple problems (Blum and Rivest, 1989; Judd, 1990). In practice, however, you can usually get good results without doing a full-blown global optimization; e.g., using multiple (say, 10 to 1000) random weight initializations is usually sufficient.

One example of a function that a typical neural net cannot learn is `Y=1/X` on the open interval (0,1). An open interval is not a compact set. With any bounded output activation function, the error will get arbitrarily large as the input approaches zero. Of course, you could make the output activation function a reciprocal function and easily get a perfect fit, but neural networks are most often used in situations where you do not have enough prior knowledge to set the activation function in such a clever way. There are also many other important problems that are so difficult that a neural network will be unable to learn them without memorizing the entire training set, such as:

- Predicting random or pseudo-random numbers.
- Factoring large integers.
- Determing whether a large integer is prime or composite.
- Decrypting anything encrypted by a good algorithm.

And it is important to understand that there are no methods for training NNs that can magically create information that is not contained in the training data.

Feedforward NNs are restricted to finite-dimensional input and output spaces. Recurrent NNs can in theory process arbitrarily long strings of numbers or symbols. But training recurrent NNs has posed much more serious practical difficulties than training feedforward networks. NNs are, at least today, difficult to apply successfully to problems that concern manipulation of symbols and rules, but much research is being done.

There have been attempts to pack recursive structures into finite-dimensional real vectors (Blair, 1997; Pollack, 1990; Chalmers, 1990; Chrisman, 1991; Plate, 1994; Hammerton, 1998). Obviously, finite precision limits how far the recursion can go (Hadley, 1999). The practicality of such methods is open to debate.

As for simulating human consciousness and emotion, that's still in the realm of science fiction. Consciousness is still one of the world's great mysteries. Artificial NNs may be useful for modeling some aspects of or prerequisites for consciousness, such as perception and cognition, but ANNs provide no insight so far into what Chalmers (1996, p. xi) calls the "hard problem":

> Many books and articles on consciousness have appeared in the past few years, and one might think we are making progress. But on a closer look, most of this work leaves the hardest problems about consciousness untouched. Often, such work addresses what might be called the "easy problems" of consciousness: How does the brain process environmental

stimulation? How does it integrate information? How do we produce reports on internal states? These are important questions, but to answer them is not to solve the hard problem: Why is all this processing accompanied by an experienced inner life?

For more information on consciousness, see the on-line journal Psyche at http://psyche.cs.monash.edu.au/index.html.

For examples of specific applications of NNs, see What are some applications of NNs?

References:

Blair, A.D. (1997), "Scaling Up RAAMs," Brandeis University Computer Science Technical Report CS-97-192, http://www.demo.cs.brandeis.edu/papers/long.html#sur97

Blum, A., and Rivest, R.L. (1989), "Training a 3-node neural network is NP-complete," in Touretzky, D.S. (ed.), *Advances in Neural Information Processing Systems 1*, San Mateo, CA: Morgan Kaufmann, 494-501.

Chalmers, D.J. (1990), "Syntactic Transformations on Distributed Representations," Connection Science, 2, 53-62, http://ling.ucsc.edu/~chalmers/papers/transformations.ps

Chalmers, D.J. (1996), *The Conscious Mind: In Search of a Fundamental Theory*, NY: Oxford University Press.

Chrisman, L. (1991), "Learning Recursive Distributed Representations for Holistic Computation", Connection Science, 3, 345-366, ftp://reports.adm.cs.cmu.edu/usr/anon/1991/CMU-CS-91-154.ps

Collier, R. (1994), "An historical overview of natural language processing systems that learn," Artificial Intelligence Review, 8(1), ??-??.

Devroye, L., Györfi, L., and Lugosi, G. (1996), *A Probabilistic Theory of Pattern Recognition*, NY: Springer.

Faragó, A. and Lugosi, G. (1993), "Strong Universal Consistency of Neural Network Classifiers," IEEE Transactions on Information Theory, 39, 1146-1151.

Hadley, R.F. (1999), "Cognition and the computational power of connectionist networks," http://www.cs.sfu.ca/~hadley/online.html

Hammerton, J.A. (1998), "Holistic Computation: Reconstructing a muddled concept," Connection Science, 10, 3-19, http://www.tardis.ed.ac.uk/~james/CNLP/holcomp.ps.gz

Judd, J.S. (1990), *Neural Network Design and the Complexity of Learning*, Cambridge, MA: The MIT Press.

Lugosi, G., and Zeger, K. (1995), "Nonparametric Estimation via Empirical Risk Minimization,"

IEEE Transactions on Information Theory, 41, 677-678.

Orponen, P. (2000), "An overview of the computational power of recurrent neural networks," Finnish AI Conference, Helsinki, http://www.math.jyu.fi/~orponen/papers/rnncomp.ps

Plate, T.A. (1994), *Distributed Representations and Nested Compositional Structure*, Ph.D. Thesis, University of Toronto, ftp://ftp.cs.utoronto.ca/pub/tap/

Pollack, J. B. (1990), "Recursive Distributed Representations," Artificial Intelligence 46, 1, 77-105, http://www.demo.cs.brandeis.edu/papers/long.html#raam

Siegelmann, H.T. (1998), *Neural Networks and Analog Computation: Beyond the Turing Limit*, Boston: Birkhauser, ISBN 0-8176-3949-7, http://iew3.technion.ac.il:8080/Home/Users/iehava/book/

Siegelmann, H.T., and Sontag, E.D. (1999), "Turing Computability with Neural Networks," Applied Mathematics Letters, 4, 77-80.

Sima, J., and Orponen, P. (2001), "Computing with continuous-time Liapunov systems," 33rd ACM STOC, http://www.math.jyu.fi/~orponen/papers/liapcomp.ps

Valiant, L. (1988), "Functionality in Neural Nets," *Learning and Knowledge Acquisition*, Proc. AAAI, 629-634.

White, H. (1990), "Connectionist Nonparametric Regression: Multilayer Feedforward Networks Can Learn Arbitrary Mappings," Neural Networks, 3, 535-550. Reprinted in White (1992b).

White, H. (1992a), "Nonparametric Estimation of Conditional Quantiles Using Neural Networks," in Page, C. and Le Page, R. (eds.), *Proceedings of the 23rd Sympsium on the Interface: Computing Science and Statistics*, Alexandria, VA: American Statistical Association, pp. 190-199. Reprinted in White (1992b).

White, H. (1992b), *Artificial Neural Networks: Approximation and Learning Theory*, Blackwell.

White, H., and Gallant, A.R. (1992), "On Learning the Derivatives of an Unknown Mapping with Multilayer Feedforward Networks," Neural Networks, 5, 129-138. Reprinted in White (1992b).

Subject: Who is concerned with NNs?

Neural Networks are interesting for quite a lot of very different people:

- Computer scientists want to find out about the properties of non-symbolic information processing with neural nets and about learning systems in general.
- Statisticians use neural nets as flexible, nonlinear regression and classification models.

- Engineers of many kinds exploit the capabilities of neural networks in many areas, such as signal processing and automatic control.
- Cognitive scientists view neural networks as a possible apparatus to describe models of thinking and consciousness (High-level brain function).
- Neuro-physiologists use neural networks to describe and explore medium-level brain function (e.g. memory, sensory system, motorics).
- Physicists use neural networks to model phenomena in statistical mechanics and for a lot of other tasks.
- Biologists use Neural Networks to interpret nucleotide sequences.
- Philosophers and some other people may also be interested in Neural Networks for various reasons.

For world-wide lists of groups doing research on NNs, see the Foundation for Neural Networks's (SNN) page at http://www.mbfys.kun.nl/snn/pointers/groups.html and see Neural Networks Research on the IEEE Neural Network Council's homepage http://www.ieee.org/nnc.

Subject: How many kinds of NNs exist?

There are many many kinds of NNs by now. Nobody knows exactly how many. New ones (or at least variations of old ones) are invented every week. Below is a collection of some of the most well known methods, not claiming to be complete.

The two main kinds of learning algorithms are supervised and unsupervised.

- In supervised learning, the correct results (target values, desired outputs) are known and are given to the NN during training so that the NN can adjust its weights to try match its outputs to the target values. After training, the NN is tested by giving it only input values, not target values, and seeing how close it comes to outputting the correct target values.
- In unsupervised learning, the NN is not provided with the correct results during training. Unsupervised NNs usually perform some kind of data compression, such as dimensionality reduction or clustering. See "What does unsupervised learning learn?"

The distinction between supervised and unsupervised methods is not always clear-cut. An unsupervised method can learn a summary of a probability distribution, then that summarized distribution can be used to make predictions. Furthermore, supervised methods come in two subvarieties: auto-associative and hetero-associative. In auto-associative learning, the target values are the same as the inputs, whereas in hetero-associative learning, the targets are generally different from the inputs. Many unsupervised methods are equivalent to auto-associative supervised methods. For more details, see "What does unsupervised learning learn?"

Two major kinds of network topology are feedforward and feedback.

- In a feedforward NN, the connections between units do not form cycles. Feedforward NNs usually produce a response to an input quickly. Most feedforward NNs can be trained using a wide variety of efficient conventional numerical methods (e.g. see "What are conjugate gradients, Levenberg-

Marquardt, etc.?") in addition to algorithms invented by NN reserachers.

- In a feedback or recurrent NN, there are cycles in the connections. In some feedback NNs, each time an input is presented, the NN must iterate for a potentially long time before it produces a response. Feedback NNs are usually more difficult to train than feedforward NNs.

Some kinds of NNs (such as those with winner-take-all units) can be implemented as either feedforward or feedback networks.

NNs also differ in the kinds of data they accept. Two major kinds of data are categorical and quantitative.

- Categorical variables take only a finite (technically, countable) number of possible values, and there are usually several or more cases falling into each category. Categorical variables may have symbolic values (e.g., "male" and "female", or "red", "green" and "blue") that must be encoded into numbers before being given to the network (see "How should categories be encoded?") Both supervised learning with categorical target values and unsupervised learning with categorical outputs are called "classification."
- Quantitative variables are numerical measurements of some attribute, such as length in meters. The measurements must be made in such a way that at least some arithmetic relations among the measurements reflect analogous relations among the attributes of the objects that are measured. For more information on measurement theory, see the Measurement Theory FAQ at ftp://ftp.sas.com/pub/neural/measurement.html. Supervised learning with quantitative target values is called "regression."

Some variables can be treated as either categorical or quantitative, such as number of children or any binary variable. Most regression algorithms can also be used for supervised classification by encoding categorical target values as 0/1 binary variables and using those binary variables as target values for the regression algorithm. The outputs of the network are posterior probabilities when any of the most common training methods are used.

Here are some well-known kinds of NNs:

1. Supervised
    1. Feedforward
        - Linear
            - Hebbian - Hebb (1949), Fausett (1994)
            - Perceptron - Rosenblatt (1958), Minsky and Papert (1969/1988), Fausett (1994)
            - Adaline - Widrow and Hoff (1960), Fausett (1994)
            - Higher Order - Bishop (1995)
            - Functional Link - Pao (1989)
        - MLP: Multilayer perceptron - Bishop (1995), Reed and Marks (1999), Fausett (1994)
            - Backprop - Rumelhart, Hinton, and Williams (1986)
            - Cascade Correlation - Fahlman and Lebiere (1990), Fausett (1994)
            - Quickprop - Fahlman (1989)
            - RPROP - Riedmiller and Braun (1993)

- RBF networks - Bishop (1995), Moody and Darken (1989), Orr (1996)
  - OLS: Orthogonal Least Squares - Chen, Cowan and Grant (1991)
- CMAC: Cerebellar Model Articulation Controller - Albus (1975), Brown and Harris (1994)
- Classification only
  - LVQ: Learning Vector Quantization - Kohonen (1988), Fausett (1994)
  - PNN: Probabilistic Neural Network - Specht (1990), Masters (1993), Hand (1982), Fausett (1994)
- Regression only
  - GNN: General Regression Neural Network - Specht (1991), Nadaraya (1964), Watson (1964)

2. Feedback - Hertz, Krogh, and Palmer (1991), Medsker and Jain (2000)

   - BAM: Bidirectional Associative Memory - Kosko (1992), Fausett (1994)
   - Boltzman Machine - Ackley et al. (1985), Fausett (1994)
   - Recurrent time series
     - Backpropagation through time - Werbos (1990)
     - Elman - Elman (1990)
     - FIR: Finite Impulse Response - Wan (1990)
     - Jordan - Jordan (1986)
     - Real-time recurrent network - Williams and Zipser (1989)
     - Recurrent backpropagation - Pineda (1989), Fausett (1994)
     - TDNN: Time Delay NN - Lang, Waibel and Hinton (1990)

3. Competitive

   - ARTMAP - Carpenter, Grossberg and Reynolds (1991)
   - Fuzzy ARTMAP - Carpenter, Grossberg, Markuzon, Reynolds and Rosen (1992), Kasuba (1993)
   - Gaussian ARTMAP - Williamson (1995)
   - Counterpropagation - Hecht-Nielsen (1987; 1988; 1990), Fausett (1994)
   - Neocognitron - Fukushima, Miyake, and Ito (1983), Fukushima, (1988), Fausett (1994)

2. Unsupervised - Hertz, Krogh, and Palmer (1991)

   1. Competitive

      - Vector Quantization
        - Grossberg - Grossberg (1976)
        - Kohonen - Kohonen (1984)
        - Conscience - Desieno (1988)
      - Self-Organizing Map
        - Kohonen - Kohonen (1995), Fausett (1994)
        - GTM: - Bishop, Svensén and Williams (1997)
        - Local Linear - Mulier and Cherkassky (1995)
      - Adaptive resonance theory

        - ART 1 - Carpenter and Grossberg (1987a), Moore (1988), Fausett (1994)
        - ART 2 - Carpenter and Grossberg (1987b), Fausett (1994)
        - ART 2-A - Carpenter, Grossberg and Rosen (1991a)
        - ART 3 - Carpenter and Grossberg (1990)
        - Fuzzy ART - Carpenter, Grossberg and Rosen (1991b)
    - DCL: Differential Competitive Learning - Kosko (1992)

2. Dimension Reduction - Diamantaras and Kung (1996)

    - Hebbian - Hebb (1949), Fausett (1994)
    - Oja - Oja (1989)
    - Sanger - Sanger (1989)
    - Differential Hebbian - Kosko (1992)

3. Autoassociation

    - Linear autoassociator - Anderson et al. (1977), Fausett (1994)
    - BSB: Brain State in a Box - Anderson et al. (1977), Fausett (1994)
    - Hopfield - Hopfield (1982), Fausett (1994)

3. Nonlearning

    1. Hopfield - Hertz, Krogh, and Palmer (1991)
    2. various networks for optimization - Cichocki and Unbehauen (1993)

References:

Ackley, D.H., Hinton, G.F., and Sejnowski, T.J. (1985), "A learning algorithm for Boltzman machines," Cognitive Science, 9, 147-169.

Albus, J.S (1975), "New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)," Transactions of the ASME Journal of Dynamic Systems, Measurement, and Control, September 1975, 220-27.

Anderson, J.A., and Rosenfeld, E., eds. (1988), *Neurocomputing: Foundatons of Research*, Cambridge, MA: The MIT Press.

Anderson, J.A., Silverstein, J.W., Ritz, S.A., and Jones, R.S. (1977) "Distinctive features, categorical perception, and probability learning: Some applications of a neural model," Psychological Rveiew, 84, 413-451. Reprinted in Anderson and Rosenfeld (1988).

Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press.

Bishop, C.M., Svensén, M., and Williams, C.K.I (1997), "GTM: A principled alternative to the self-organizing map," in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) *Advances in Neural Information Processing Systems 9*, Cambrideg, MA: The MIT Press, pp. 354-360. Also see http://www.ncrg.aston.ac.uk/GTM/

Brown, M., and Harris, C. (1994), *Neurofuzzy Adaptive Modelling and Control*, NY: Prentice Hall.

Carpenter, G.A., Grossberg, S. (1987a), "A massively parallel architecture for a self-organizing neural pattern recognition machine," Computer Vision, Graphics, and Image Processing, 37, 54-115.

Carpenter, G.A., Grossberg, S. (1987b), "ART 2: Self-organization of stable category recognition codes for analog input patterns," Applied Optics, 26, 4919-4930.

Carpenter, G.A., Grossberg, S. (1990), "ART 3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. Neural Networks, 3, 129-152.

Carpenter, G.A., Grossberg, S., Markuzon, N., Reynolds, J.H., and Rosen, D.B. (1992), "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps," IEEE Transactions on Neural Networks, 3, 698-713

Carpenter, G.A., Grossberg, S., Reynolds, J.H. (1991), "ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network," Neural Networks, 4, 565-588.

Carpenter, G.A., Grossberg, S., Rosen, D.B. (1991a), "ART 2-A: An adaptive resonance algorithm for rapid category learning and recognition," Neural Networks, 4, 493-504.

Carpenter, G.A., Grossberg, S., Rosen, D.B. (1991b), "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system," Neural Networks, 4, 759-771.

Chen, S., Cowan, C.F.N., and Grant, P.M. (1991), "Orthogonal least squares learning for radial basis function networks," IEEE Transactions on Neural Networks, 2, 302-309.

Cichocki, A. and Unbehauen, R. (1993). *Neural Networks for Optimization and Signal Processing.* NY: John Wiley & Sons, ISBN 0-471-93010-5.

Desieno, D. (1988), "Adding a conscience to competitive learning," Proc. Int. Conf. on Neural Networks, I, 117-124, IEEE Press.

Diamantaras, K.I., and Kung, S.Y. (1996) *Principal Component Neural Networks: Theory and Applications*, NY: Wiley.

Elman, J.L. (1990), "Finding structure in time," Cognitive Science, 14, 179-211.

Fahlman, S.E. (1989), "Faster-Learning Variations on Back-Propagation: An Empirical Study", in Touretzky, D., Hinton, G, and Sejnowski, T., eds., *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann, 38-51.

Fahlman, S.E., and Lebiere, C. (1990), "The Cascade-Correlation Learning Architecture", in Touretzky, D. S. (ed.), *Advances in Neural Information Processing Systems 2,*, Los Altos, CA:

Morgan Kaufmann Publishers, pp. 524-532.

Fausett, L. (1994), *Fundamentals of Neural Networks*, Englewood Cliffs, NJ: Prentice Hall.

Fukushima, K., Miyake, S., and Ito, T. (1983), "Neocognitron: A neural network model for a mechanism of visual pattern recognition," IEEE Transactions on Systems, Man, and Cybernetics, 13, 826-834.

Fukushima, K. (1988), "Neocognitron: A hierarchical neural network capable of visual pattern recognition," Neural Networks, 1, 119-130.

Grossberg, S. (1976), "Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors," Biological Cybernetics, 23, 121-134

Hand, D.J. (1982) *Kernel Discriminant Analysis*, Research Studies Press.

Hebb, D.O. (1949), *The Organization of Behavior*, NY: John Wiley & Sons.

Hecht-Nielsen, R. (1987), "Counterpropagation networks," Applied Optics, 26, 4979-4984.

Hecht-Nielsen, R. (1988), "Applications of counterpropagation networks," Neural Networks, 1, 131-139.

Hecht-Nielsen, R. (1990), *Neurocomputing*, Reading, MA: Addison-Wesley.

Hertz, J., Krogh, A., and Palmer, R. (1991). *Introduction to the Theory of Neural Computation.* Addison-Wesley: Redwood City, California.

Hopfield, J.J. (1982), "Neural networks and physical systems with emergent collective computational abilities," Proceedings of the National Academy of Sciences, 79, 2554-2558. Reprinted in Anderson and Rosenfeld (1988).

Jordan, M. I. (1986), "Attractor dynamics and parallelism in a connectionist sequential machine," In Proceedings of the Eighth Annual conference of the Cognitive Science Society, pages 531-546. Lawrence Erlbaum.

Kasuba, T. (1993), "Simplified Fuzzy ARTMAP," AI Expert, 8, 18-25.

Kohonen, T. (1984), *Self-Organization and Associative Memory*, Berlin: Springer.

Kohonen, T. (1988), "Learning Vector Quantization," Neural Networks, 1 (suppl 1), 303.

Kohonen, T. (1995/1997), *Self-Organizing Maps*, Berlin: Springer-Verlag. First edition was 1995, second edition 1997. See http://www.cis.hut.fi/nnrc/new_book.html for information on the second edition.

Kosko, B.(1992), *Neural Networks and Fuzzy Systems,* Englewood Cliffs, N.J.: Prentice-Hall.

Lang, K. J., Waibel, A. H., and Hinton, G. (1990), "A time-delay neural network architecture for isolated word recognition," Neural Networks, 3, 23-44.

Masters, T. (1993). *Practical Neural Network Recipes in C++*, San Diego: Academic Press.

Masters, T. (1995) *Advanced Algorithms for Neural Networks: A C++ Sourcebook*, NY: John Wiley and Sons, ISBN 0-471-10588-0

Medsker, L.R., and Jain, L.C., eds. (2000), *Recurrent Neural Networks: Design and Applications*, Boca Raton, FL: CRC Press, ISBN 0-8493-7181-3.

Minsky, M.L., and Papert, S.A. (1969/1988), *Perceptrons*, Cambridge, MA: The MIT Press (first edition, 1969; expanded edition, 1988).

Moody, J. and Darken, C.J. (1989), "Fast learning in networks of locally-tuned processing units," Neural Computation, 1, 281-294.

Moore, B. (1988), "ART 1 and Pattern Clustering," in Touretzky, D., Hinton, G. and Sejnowski, T., eds., *Proceedings of the 1988 Connectionist Models Summer School*, 174-185, San Mateo, CA: Morgan Kaufmann.

Mulier, F. and Cherkassky, V. (1995), "Self-Organization as an Iterative Kernel Smoothing Process," Neural Computation, 7, 1165-1177.

Nadaraya, E.A. (1964) "On estimating regression", Theory Probab. Applic. 10, 186-90.

Oja, E. (1989), "Neural networks, principal components, and subspaces," International Journal of Neural Systems, 1, 61-68.

Orr, M.J.L. (1996), "Introduction to radial basis function networks," http://www.anc.ed.ac.uk/~mjo/papers/intro.ps or http://www.anc.ed.ac.uk/~mjo/papers/intro.ps.gz

Pao, Y. H. (1989), *Adaptive Pattern Recognition and Neural Networks*, Reading, MA: Addison-Wesley Publishing Company, ISBN 0-201-12584-6.

Pineda, F.J. (1989), "Recurrent back-propagation and the dynamical approach to neural computation," Neural Computation, 1, 161-172.

Reed, R.D., and Marks, R.J, II (1999), *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*, Cambridge, MA: The MIT Press, ISBN 0-262-18190-8.

Riedmiller, M. and Braun, H. (1993), "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm", *Proceedings of the IEEE International Conference on Neural Networks* 1993, San Francisco: IEEE.

Rosenblatt, F. (1958), "The perceptron: A probabilistic model for information storage and organization in the brain., Psychological Review, 65, 386-408.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986), "Learning internal representations by error propagation", in Rumelhart, D.E. and McClelland, J. L., eds. (1986), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1, 318-362, Cambridge, MA: The MIT Press.

Sanger, T.D. (1989), "Optimal unsupervised learning in a single-layer linear feedforward neural network," Neural Networks, 2, 459-473.

Specht, D.F. (1990) "Probabilistic neural networks," Neural Networks, 3, 110-118.

Specht, D.F. (1991) "A Generalized Regression Neural Network", IEEE Transactions on Neural Networks, 2, Nov. 1991, 568-576.

Wan, E.A. (1990), "Temporal backpropagation: An efficient algorithm for finite impulse response neural networks," in *Proceedings of the 1990 Connectionist Models Summer School*, Touretzky, D.S., Elman, J.L., Sejnowski, T.J., and Hinton, G.E., eds., San Mateo, CA: Morgan Kaufmann, pp. 131-140.

Watson, G.S. (1964) "Smooth regression analysis", Sankhy{\=a}, Series A, 26, 359-72.

Werbos, P.J. (1990), "Backpropagtion through time: What it is and how to do it," Proceedings of the IEEE, 78, 1550-1560.

Widrow, B., and Hoff, M.E., Jr., (1960), "Adaptive switching circuits," IRE WESCON Convention Record. part 4, pp. 96-104. Reprinted in Anderson and Rosenfeld (1988).

Williams, R.J., and Zipser, D., (1989), "A learning algorithm for continually running fully recurrent neurla networks," Neural Computation, 1, 270-280.

Williamson, J.R. (1995), "Gaussian ARTMAP: A neural network for fast incremental learning of noisy multidimensional maps," Technical Report CAS/CNS-95-003, Boston University, Center of Adaptive Systems and Department of Cognitive and Neural Systems.

Subject: How many kinds of Kohonen networks exist? (And what is k-means?)

Teuvo Kohonen is one of the most famous and prolific researchers in neurocomputing, and he has invented a variety of networks. But many people refer to "Kohonen networks" without specifying which *kind* of Kohonen network, and this lack of precision can lead to confusion. The phrase "Kohonen network" most often refers to one of the following three types of networks:

- VQ: Vector Quantization--competitive networks that can be viewed as unsupervised density estimators or autoassociators (Kohonen, 1995/1997; Hecht-Nielsen 1990), closely related to k-means cluster analysis (MacQueen, 1967; Anderberg, 1973). Each competitive unit corresponds to a cluster, the center of which is called a "codebook vector". Kohonen's learning law is an on-line algorithm that finds the codebook vector closest to each training case and moves the "winning" codebook vector closer to the training case. The codebook vector is moved a certain proportion of the distance between it and the training case, the proportion being specified by the learning rate, that is:

```
new_codebook = old_codebook * (1-learning_rate)
             + data * learning_rate
```

  Numerous similar algorithms have been developed in the neural net and machine learning literature; see Hecht-Nielsen (1990) for a brief historical overview, and Kosko (1992) for a more technical overview of competitive learning.

  MacQueen's on-line k-means algorithm is essentially the same as Kohonen's learning law except that the learning rate is the reciprocal of the number of cases that have been assigned to the winnning cluster. Suppose that when processing a given training case, `N` cases have been previously assigned to the winning codebook vector. Then the codebook vector is updated as:

```
new_codebook = old_codebook * N/(N+1)
             + data * 1/(N+1)
```

  This reduction of the learning rate makes each codebook vector the mean of all cases assigned to its cluster and guarantees convergence of the algorithm to an optimum value of the error function (the sum of squared Euclidean distances between cases and codebook vectors) as the number of training cases goes to infinity. Kohonen's learning law with a fixed learning rate does not converge. As is well known from stochastic approximation theory, convergence requires the sum of the infinite sequence of learning rates to be infinite, while the sum of squared learning rates must be finite (Kohonen, 1995, p. 34). These requirements are satisfied by MacQueen's k-means algorithm.

  Kohonen VQ is often used for off-line learning, in which case the training data are stored and Kohonen's learning law is applied to each case in turn, cycling over the data set many times (incremental training). Convergence to a local optimum can be obtained as the training time goes to infinity if the learning rate is reduced in a suitable manner as described above. However, there are off-line k-means algorithms, both batch and incremental, that converge in a finite number of iterations (Anderberg, 1973; Hartigan, 1975; Hartigan and Wong, 1979). The batch algorithms such as Forgy's (1965; Anderberg, 1973) have the advantage for large data sets, since the incremental methods require you either to store the cluster membership of each case or to do two nearest-cluster computations as each case is processed. Forgy's algorithm is a simple alternating least-squares algorithm consisting of the following steps:

  1. Initialize the codebook vectors.
  2. Repeat the following two steps until convergence:

A. Read the data, assigning each case to the nearest (using Euclidean distance) codebook vector.
B. Replace each codebook vector with the mean of the cases that were assigned to it.

Fastest training is usually obtained if MacQueen's on-line algorithm is used for the first pass and off-line k-means algorithms are applied on subsequent passes (Bottou and Bengio, 1995). However, these training methods do not necessarily converge to a global optimum of the error function. The chance of finding a global optimum can be improved by using rational initialization (SAS Institute, 1989, pp. 824-825), multiple random initializations, or various time-consuming training methods intended for global optimization (Ismail and Kamel, 1989; Zeger, Vaisy, and Gersho, 1992).

VQ has been a popular topic in the signal processing literature, which has been largely separate from the literature on Kohonen networks and from the cluster analysis literature in statistics and taxonomy. In signal processing, on-line methods such as Kohonen's and MacQueen's are called "adaptive vector quantization" (AVQ), while off-line k-means methods go by the names of "Lloyd" or "Lloyd I" (Lloyd, 1982) and "LBG" (Linde, Buzo, and Gray, 1980). There is a recent textbook on VQ by Gersho and Gray (1992) that summarizes these algorithms as information compression methods.

Kohonen's work emphasized VQ as density estimation and hence the desirability of equiprobable clusters (Kohonen 1984; Hecht-Nielsen 1990). However, Kohonen's learning law does not produce equiprobable clusters--that is, the proportions of training cases assigned to each cluster are not usually equal. If there are `I` inputs and the number of clusters is large, the density of the codebook vectors approximates the `I/(I+2)` power of the density of the training data (Kohonen, 1995, p. 35; Ripley, 1996, p. 202; Zador, 1982), so the clusters are approximately equiprobable only if the data density is uniform or the number of inputs is large. The most popular method for obtaining equiprobability is Desieno's (1988) algorithm which adds a "conscience" value to each distance prior to the competition. The conscience value for each cluster is adjusted during training so that clusters that win more often have larger conscience values and are thus handicapped to even out the probabilities of winning in later iterations.

Kohonen's learning law is an approximation to the k-means model, which is an approximation to normal mixture estimation by maximum likelihood assuming that the mixture components (clusters) all have spherical covariance matrices and equal sampling probabilities. Hence if the population contains clusters that are not equiprobable, k-means will tend to produce sample clusters that are more nearly equiprobable than the population clusters. Corrections for this bias can be obtained by maximizing the likelihood without the assumption of equal sampling probabilities Symons (1981). Such corrections are similar to conscience but have the opposite effect.

In cluster analysis, the purpose is not to compress information but to recover the true cluster memberships. K-means differs from mixture models in that, for k-means, the cluster membership for each case is considered a separate parameter to be estimated, while mixture models estimate a posterior probability for each case based on the means, covariances, and sampling probabilities of each cluster. Balakrishnan, Cooper, Jacob, and Lewis (1994) found that k-means algorithms recovered cluster membership more accurately than Kohonen VQ.

- SOM: Self-Organizing Map--competitive networks that provide a "topological" mapping from the input space to the clusters (Kohonen, 1995). The SOM was inspired by the way in which various

human sensory impressions are neurologically mapped into the brain such that spatial or other relations among stimuli correspond to spatial relations among the neurons. In a SOM, the neurons (clusters) are organized into a grid--usually two-dimensional, but sometimes one-dimensional or (rarely) three- or more-dimensional. The grid exists in a space that is separate from the input space; any number of inputs may be used as long as the number of inputs is greater than the dimensionality of the grid space. A SOM tries to find clusters such that any two clusters that are close to each other in the grid space have codebook vectors close to each other in the input space. But the converse does not hold: codebook vectors that are close to each other in the input space do not necessarily correspond to clusters that are close to each other in the grid. Another way to look at this is that a SOM tries to embed the grid in the input space such every training case is close to some codebook vector, but the grid is bent or stretched as little as possible. Yet another way to look at it is that a SOM is a (discretely) smooth mapping between regions in the input space and points in the grid space. The best way to undestand this is to look at the pictures in Kohonen (1995) or various other NN textbooks.

The Kohonen algorithm for SOMs is very similar to the Kohonen algorithm for AVQ. Let the codebook vectors be indexed by a subscript `j`, and let the index of the codebook vector nearest to the current training case be `n`. The Kohonen SOM algorithm requires a kernel function `K(j,n)`, where `K(j,j)=1` and `K(j,n)` is usually a non-increasing function of the distance (in any metric) between codebook vectors `j` and `n` in the grid space (not the input space). Usually, `K(j,n)` is zero for codebook vectors that are far apart in the grid space. As each training case is processed, all the codebook vectors are updated as:

```
new_codebook  = old_codebook  * [1 - K(j,n) * learning_rate]
           j               j

             + data * K(j,n) * learning_rate
```

The kernel function does not necessarily remain constant during training. The neighborhood of a given codebook vector is the set of codebook vectors for which `K(j,n)>0.` To avoid poor results (akin to local minima), it is usually advisable to start with a large neighborhood, and let the neighborhood gradually shrink during training. If `K(j,n)=0` for `j` not equal to `n`, then the SOM update formula reduces to the formula for Kohonen vector quantization. In other words, if the neighborhood size (for example, the radius of the support of the kernel function `K(j,n)`) is zero, the SOM algorithm degenerates into simple VQ. Hence it is important not to let the neighborhood size shrink all the way to zero during training. Indeed, the choice of the final neighborhood size is the most important tuning parameter for SOM training, as we will see shortly.

A SOM works by smoothing the codebook vectors in a manner similar to kernel estimation methods, but the smoothing is done in neighborhoods in the grid space rather than in the input space (Mulier and Cherkassky 1995). This is easier to see in a batch algorithm for SOMs, which is similar to Forgy's algorithm for batch k-means, but incorporates an extra smoothing process:

1. Initialize the codebook vectors.
2. Repeat the following two steps until convergence or boredom:
   A. Read the data, assigning each case to the nearest (using Euclidean distance) codebook vector. While you are doing this, keep track of the mean and the number of cases for each cluster.

B. Do a nonparametric regression using `K(j,n)` as a kernel function, with the grid points as inputs, the cluster means as target values, and the number of cases in each cluster as an case weight. Replace each codebook vector with the output of the nonparametric regression function evaluated at its grid point.

If the nonparametric regression method is Nadaraya-Watson kernel regression (see What is GRNN?), the batch SOM algorithm produces essentially the same results as Kohonen's algorithm, barring local minima. The main difference is that the batch algorithm often converges. Mulier and Cherkassky (1995) note that other nonparametric regression methods can be used to provide superior SOM algorithms. In particular, local-linear smoothing eliminates the notorious "border effect", whereby the codebook vectors near the border of the grid are compressed in the input space. The border effect is especially problematic when you try to use a high degree of smoothing in a Kohonen SOM, since all the codebook vectors will collapse into the center of the input space. The SOM border effect has the same mathematical cause as the "boundary effect" in kernel regression, which causes the estimated regression function to flatten out near the edges of the regression input space. There are various cures for the edge effect in nonparametric regression, of which local-linear smoothing is the simplest (Wand and Jones, 1995). Hence, local-linear smoothing also cures the border effect in SOMs. Furthermore, local-linear smoothing is much more general and reliable than the heuristic weighting rule proposed by Kohonen (1995, p. 129).

Since nonparametric regression is used in the batch SOM algorithm, various properties of nonparametric regression extend to SOMs. In particular, it is well known that the shape of the kernel function is not a crucial matter in nonparametric regression, hence it is not crucial in SOMs. On the other hand, the amount of smoothing used for nonparametric regression is crucial, hence the choice of the final neighborhood size in a SOM is crucial. Unfortunately, I am not aware of any systematic studies of methods for choosing the final neighborhood size.

The batch SOM algorithm is very similar to the principal curve and surface algorithm proposed by Hastie and Stuetzle (1989), as pointed out by Ritter, Martinetz, and Schulten (1992) and Mulier and Cherkassky (1995). A principal curve is a nonlinear generalization of a principal component. Given the probability distribution of a population, a principal curve is defined by the following self-consistency condition:

1. If you choose any point on a principal curve,
2. then find the set of all the points in the input space that are closer to the chosen point than any other point on the curve,
3. and compute the expected value (mean) of that set of points with respect to the probability distribution, then
4. you end up with the same point on the curve that you chose originally.

See http://www.iro.umontreal.ca/~kegl/research/pcurves/ for more information about principal curves and surfaces.

In a multivariate normal distribution, the principal curves are the same as the principal components. A principal surface is the obvious generalization from a curve to a surface. In a multivariate normal distribution, the principal surfaces are the subspaces spanned by any two principal components.

A one-dimensional local-linear batch SOM can be used to estimate a principal curve by letting the

number of codebook vectors approach infinity while choosing a kernel function of appropriate smoothness. A two-dimensional local-linear batch SOM can be used to estimate a principal surface by letting the number of both rows and columns in the grid approach infinity. This connection between SOMs and principal curves and surfaces shows that the choice of the number of codebook vectors is not crucial, provided the number is fairly large.

If the final neighborhood size in a local-linear batch SOM is large, the SOM approximates a subspace spanned by principal components--usually the first principal component if the SOM is one-dimensional, the first two principal components if the SOM is two-dimensional, and so on. This result does not depend on the data having a multivariate normal distribution.

Principal component analysis is a method of data compression, not a statistical model. However, there is a related method called "common factor analysis" that is often confused with principal component analysis but is indeed a statistical model. Common factor analysis posits that the relations among the observed variables can be explained by a smaller number of unobserved, "latent" variables. Tibshirani (1992) has proposed a latent-variable variant of principal curves, and latent-variable modifications of SOMs have been introduced by Utsugi (1996, 1997) and Bishop, Svensén, and Williams (1997).

The choice of the number of codebook vectors is usually not critical as long as the number is fairly large. But results can be sensitive to the shape of the grid, e.g., square or an elongated rectangle. And the dimensionality of the grid is a crucial choice. It is difficult to guess the appropriate shape and dimensionality before analyzing the data. Determining the shape and dimensionality by trial and error can be quite tedious. Hence, a variety of methods have been tried for growing SOMs and related kinds of NNs during training. For more information on growing SOMs, see Bernd Fritzke's home page at http://pikas.inf.tu-dresden.de/~fritzke/

Using a 1-by-2 SOM is pointless. There is no "topological structure" in a 1-by-2 grid. A 1-by-2 SOM is essentially the same as VQ with two clusters, except that the SOM clusters will be closer together than the VQ clusters if the final neighborhood size for the SOM is large.

In a Kohonen SOM, as in VQ, it is necessary to reduce the learning rate during training to obtain convergence. Greg Heath has commented in this regard:

> I favor separate learning rates for each winning SOM node (or k-means cluster) in the form `1/(N_0i + N_i + 1)`, where `N_i` is the count of vectors that have caused node `i` to be a winner and `N_0i` is an initializing count that indicates the confidence in the initial weight vector assignment. The winning node expression is based on stochastic estimation convergence constraints and pseudo-Bayesian estimation of mean vectors. Kohonen derived a heuristic recursion relation for the "optimal" rate. To my surprise, when I solved the recursion relation I obtained the same above expression that I've been using for years.
>
> In addition, I have had success using the similar form `(1/n)/(N_0j + N_j + (1/n))` for the `n` nodes in the shrinking updating-neighborhood. Before the final "winners-only" stage when neighbors are no longer updated, the number of updating neighbors eventually shrinks to `n = 6` or `8` for hexagonal or rectangular

neighborhoods, respectively.

Kohonen's neighbor-update formula is more precise replacing my constant fraction `(1/n)` with a node-pair specific `h_ij (h_ij < 1).` However, as long as the initial neighborhood is sufficiently large, the shrinking rate is sufficiently slow, and the final winner-only stage is sufficiently long, the results should be relatively insensitive to exact form of `h_ij.`

Another advantage of batch SOMs is that there is no learning rate, so these complications evaporate.

Kohonen (1995, p. VII) says that SOMs are not intended for pattern recognition but for clustering, visualization, and abstraction. Kohonen has used a "supervised SOM" (1995, pp. 160-161) that is similar to counterpropagation (Hecht-Nielsen 1990), but he seems to prefer LVQ (see below) for supervised classification. Many people continue to use SOMs for classification tasks, sometimes with surprisingly (I am tempted to say "inexplicably") good results (Cho, 1997).

- LVQ: Learning Vector Quantization--competitive networks for supervised classification (Kohonen, 1988, 1995; Ripley, 1996). Each codebook vector is assigned to one of the target classes. Each class may have one or more codebook vectors. A case is classified by finding the nearest codebook vector and assigning the case to the class corresponding to the codebook vector. Hence LVQ is a kind of nearest-neighbor rule.

  Ordinary VQ methods, such as Kohonen's VQ or k-means, can easily be used for supervised classification. Simply count the number of training cases from each class assigned to each cluster, and divide by the total number of cases in the cluster to get the posterior probability. For a given case, output the class with the greatest posterior probability--i.e. the class that forms a majority in the nearest cluster. Such methods can provide universally consistent classifiers (Devroye et al., 1996) even when the codebook vectors are obtained by unsupervised algorithms. LVQ tries to improve on this approach by adapting the codebook vectors in a supervised way. There are several variants of LVQ--called LVQ1, OLVQ1, LVQ2, and LVQ3--based on heuristics. However, a smoothed version of LVQ can be trained as a feedforward network using a NRBFEQ architecture (see "How do MLPs compare with RBFs?") and optimizing any of the usual error functions; as the width of the RBFs goes to zero, the NRBFEQ network approaches an optimized LVQ network.

There are several other kinds of Kohonen networks described in Kohonen (1995), including:

- DEC--Dynamically Expanding Context
- LSM--Learning Subspace Method
- ASSOM--Adaptive Subspace SOM
- FASSOM--Feedback-controlled Adaptive Subspace SOM
- Supervised SOM
- LVQ-SOM

More information on the error functions (or absence thereof) used by Kohonen VQ and SOM is provided under "What does unsupervised learning learn?"

For more on-line information on Kohonen networks and other varieties of SOMs, see:

- The web page of The Neural Networks Research Centre, Helsinki University of Technology, at http://www.cis.hut.fi/research/
- The SOM of articles from comp.ai.neural-nets at http://websom.hut.fi/websom/comp.ai.neural-nets-new/html/root.html
- Akio Utsugi's web page on Bayesian SOMs at the National Institute of Bioscience and Human-Technology, Agency of Industrial Science and Technology, M.I.T.I., 1-1, Higashi, Tsukuba, Ibaraki, 305 Japan, at http://www.aist.go.jp/NIBH/~b0616/Lab/index-e.html
- The GTM (generative topographic mapping) home page at the Neural Computing Research Group, Aston University, Birmingham, UK, at http://www.ncrg.aston.ac.uk/GTM/
- Nenet SOM software at http://www.mbnet.fi/~phodju/nenet/nenet.html
- Bernd Fritzke's home page at http://pikas.inf.tu-dresden.de/~fritzke/ has information on growing SOMs and other related types of NNs

References:

Anderberg, M.R. (1973), *Cluster Analysis for Applications*, New York: Academic Press, Inc.

Balakrishnan, P.V., Cooper, M.C., Jacob, V.S., and Lewis, P.A. (1994) "A study of the classification capabilities of neural networks using unsupervised learning: A comparison with k-means clustering", Psychometrika, 59, 509-525.

Bishop, C.M., Svensén, M., and Williams, C.K.I (1997), "GTM: A principled alternative to the self-organizing map," in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) *Advances in Neural Information Processing Systems 9*, Cambridge, MA: The MIT Press, pp. 354-360. Also see http://www.ncrg.aston.ac.uk/GTM/

Bottou, L., and Bengio, Y. (1995), "Convergence properties of the K-Means algorithms," in Tesauro, G., Touretzky, D., and Leen, T., (eds.) *Advances in Neural Information Processing Systems 7*, Cambridge, MA: The MIT Press, pp. 585-592.

Cho, S.-B. (1997), "Self-organizing map with dynamical node-splitting: Application to handwritten digit recognition," Neural Computation, 9, 1345-1355.

Desieno, D. (1988), "Adding a conscience to competitive learning," Proc. Int. Conf. on Neural Networks, I, 117-124, IEEE Press.

Devroye, L., Györfi, L., and Lugosi, G. (1996), *A Probabilistic Theory of Pattern Recognition*, NY: Springer,

Forgy, E.W. (1965), "Cluster analysis of multivariate data: Efficiency versus interpretability," Biometric Society Meetings, Riverside, CA. Abstract in Biomatrics, 21, 768.

Gersho, A. and Gray, R.M. (1992), Vector Quantization and Signal Compression, Boston: Kluwer Academic Publishers.

Hartigan, J.A. (1975), *Clustering Algorithms*, NY: Wiley.

Hartigan, J.A., and Wong, M.A. (1979), "Algorithm AS136: A k-means clustering algorithm," Applied Statistics, 28-100-108.

Hastie, T., and Stuetzle, W. (1989), "Principal curves," Journal of the American Statistical Association, 84, 502-516.

Hecht-Nielsen, R. (1990), *Neurocomputing*, Reading, MA: Addison-Wesley.

Ismail, M.A., and Kamel, M.S. (1989), "Multidimensional data clustering utilizing hybrid search strategies," Pattern Recognition, 22, 75-89.

Kohonen, T (1984), *Self-Organization and Associative Memory*, Berlin: Springer-Verlag.

Kohonen, T (1988), "Learning Vector Quantization," Neural Networks, 1 (suppl 1), 303.

Kohonen, T. (1995/1997), *Self-Organizing Maps*, Berlin: Springer-Verlag. First edition was 1995, second edition 1997. See http://www.cis.hut.fi/nnrc/new_book.html for information on the second edition.

Kosko, B.(1992), *Neural Networks and Fuzzy Systems*, Englewood Cliffs, N.J.: Prentice-Hall.

Linde, Y., Buzo, A., and Gray, R. (1980), "An algorithm for vector quantizer design," IEEE Transactions on Communications, 28, 84-95.

Lloyd, S. (1982), "Least squares quantization in PCM," IEEE Transactions on Information Theory, 28, 129-137.

MacQueen, J.B. (1967), "Some Methods for Classification and Analysis of Multivariate Observations,"Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1, 281-297.

Max, J. (1960), "Quantizing for minimum distortion," IEEE Transactions on Information Theory, 6, 7-12.

Mulier, F. and Cherkassky, V. (1995), "Self-Organization as an iterative kernel smoothing process," Neural Computation, 7, 1165-1177.

Ripley, B.D. (1996), *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.

Ritter, H., Martinetz, T., and Schulten, K. (1992), *Neural Computation and Self-Organizing Maps: An Introduction*, Reading, MA: Addison-Wesley.

SAS Institute (1989), *SAS/STAT User's Guide*, Version 6, 4th edition, Cary, NC: SAS Institute.

Symons, M.J. (1981), "Clustering Criteria and Multivariate Normal Mixtures," Biometrics, 37, 35-43.

Tibshirani, R. (1992), "Principal curves revisited," Statistics and Computing, 2, 183-190.

Utsugi, A. (1996), "Topology selection for self-organizing maps," Network: Computation in Neural Systems, 7, 727-740, available on-line at http://www.aist.go.jp/NIBH/~b0616/Lab/index-e.html

Utsugi, A. (1997), "Hyperparameter selection for self-organizing maps," Neural Computation, 9, 623-635, available on-line at http://www.aist.go.jp/NIBH/~b0616/Lab/index-e.html

Wand, M.P., and Jones, M.C. (1995), *Kernel Smoothing*, London: Chapman & Hall.

Zador, P.L. (1982), "Asymptotic quantization error of continuous signals and the quantization dimension," IEEE Transactions on Information Theory, 28, 139-149.

Zeger, K., Vaisey, J., and Gersho, A. (1992), "Globally optimal vector quantizer design by stochastic relaxation," IEEE Transactions on Signal Procesing, 40, 310-322.

Subject: How are layers counted?

How to count layers is a matter of considerable dispute.

- Some people count layers of *units*. But of these people, some count the input layer and some don't.
- Some people count layers of *weights*. But I have no idea how they count skip-layer connections.

To avoid ambiguity, you should speak of a 2-hidden-layer network, not a 4-layer network (as some would call it) or 3-layer network (as others would call it). And if the connections follow any pattern other than fully connecting each layer to the next and to no others, you should carefully specify the connections.

Subject: What are cases and variables?

A vector of values presented at one time to all the input units of a neural network is called a "case", "example", "pattern, "sample", etc. The term "case" will be used in this FAQ because it is widely recognized, unambiguous, and requires less typing than the other terms. A case may include not only input values, but also target values and possibly other information.

A vector of values presented at different times to a single input unit is often called an "input variable" or "feature". To a statistician, it is a "predictor", "regressor", "covariate", "independent variable", "explanatory

variable", etc. A vector of target values associated with a given output unit of the network during training will be called a "target variable" in this FAQ. To a statistician, it is usually a "response" or "dependent variable".

A "data set" is a matrix containing one or (usually) more cases. In this FAQ, it will be assumed that cases are rows of the matrix, while variables are columns.

Note that the often-used term "input vector" is ambiguous; it can mean either an input case or an input variable.

Subject: What are the population, sample, training set, design set, validation set, and test set?

It is rarely useful to have a NN simply memorize a set of data, since memorization can be done much more efficiently by numerous algorithms for table look-up. Typically, you want the NN to be able to perform accurately on new data, that is, to generalize.

There seems to be no term in the NN literature for the set of all cases that you want to be able to generalize to. Statisticians call this set the "population". Tsypkin (1971) called it the "grand truth distribution," but this term has never caught on.

Neither is there a consistent term in the NN literature for the set of cases that are available for training and evaluating an NN. Statisticians call this set the "sample". The sample is usually a subset of the population.

(Neurobiologists mean something entirely different by "population," apparently some collection of neurons, but I have never found out the exact meaning. I am going to continue to use "population" in the statistical sense until NN researchers reach a consensus on some other terms for "population" and "sample"; I suspect this will never happen.)

In NN methodology, the sample is often subdivided into "training", "validation", and "test" sets. The distinctions among these subsets are crucial, but the terms "validation" and "test" sets are often confused. Bishop (1995), an indispensable reference on neural networks, provides the following explanation (p. 372):

> Since our goal is to find the network having the best performance on new data, the simplest approach to the comparison of different networks is to evaluate the error function using data which is independent of that used for training. Various networks are trained by minimization of an appropriate error function defined with respect to a *training* data set. The performance of the networks is then compared by evaluating the error function using an independent *validation* set, and the network having the smallest error with respect to the validation set is selected. This approach is called the *hold out method.* Since this procedure can itself lead to some overfitting to the validation set, the performance of the selected network should be confirmed by measuring its performance on a third independent set of data called a *test* set.

And there is no book in the NN literature more authoritative than Ripley (1996), from which the following definitions are taken (p.354):

Training set:
: A set of examples used for learning, that is to fit the parameters [i.e., weights] of the classifier.

Validation set:
: A set of examples used to tune the parameters [i.e., architecture, not weights] of a classifier, for example to choose the number of hidden units in a neural network.

Test set:
: A set of examples used only to assess the performance [generalization] of a fully-specified classifier.

The literature on machine learning often reverses the meaning of "validation" and "test" sets. This is the most blatant example of the terminological confusion that pervades artificial intelligence research.

The crucial point is that a test set, by the standard definition in the NN literature, is *never* used to choose among two or more networks, so that the error on the test set provides an unbiased estimate of the generalization error (assuming that the test set is representative of the population, etc.). Any data set that is used to choose the best of two or more networks is, by definition, a validation set, and the error of the chosen network on the validation set is optimistically biased.

There is a problem with the usual distinction between training and validation sets. Some training approaches, such as early stopping, require a validation set, so in a sense, the validation set is used for training. Other approaches, such as maximum likelihood, do not inherently require a validation set. So the "training" set for maximum likelihood might encompass both the "training" and "validation" sets for early stopping. Greg Heath has suggested the term "design" set be used for cases that are used solely to adjust the weights in a network, while "training" set be used to encompass both design and validation sets. There is considerable merit to this suggestion, but it has not yet been widely adopted.

But things can get more complicated. Suppose you want to train nets with 5 ,10, and 20 hidden units using maximum likelihood, and you want to train nets with 20 and 50 hidden units using early stopping. You also want to use a validation set to choose the best of these various networks. Should you use the same validation set for early stopping that you use for the final network choice, or should you use two separate validation sets? That is, you could divide the sample into 3 subsets, say A, B, C and proceed as follows:

- Do maximum likelihood using A.
- Do early stopping with A to adjust the weights and B to decide when to stop (this makes B a validation set).
- Choose among all 3 nets trained by maximum likelihood and the 2 nets trained by early stopping based on the error computed on B (the validation set).
- Estimate the generalization error of the chosen network using C (the test set).

Or you could divide the sample into 4 subsets, say A, B, C, and D and proceed as follows:

- Do maximum likelihood using A and B combined.
- Do early stopping with A to adjust the weights and B to decide when to stop (this makes B a validation set with respect to early stopping).

- Choose among all 3 nets trained by maximum likelihood and the 2 nets trained by early stopping based on the error computed on C (this makes C a second validation set).
- Estimate the generalization error of the chosen network using D (the test set).

Or, with the same 4 subsets, you could take a third approach:

- Do maximum likelihood using A.
- Choose among the 3 nets trained by maximum likelihood based on the error computed on B (the first validation set)
- Do early stopping with A to adjust the weights and B (the first validation set) to decide when to stop.
- Choose among the best net trained by maximum likelihood and the 2 nets trained by early stopping based on the error computed on C (the second validation set).
- Estimate the generalization error of the chosen network using D (the test set).

You could argue that the first approach is biased towards choosing a net trained by early stopping. Early stopping involves a choice among a potentially large number of networks, and therefore provides more opportunity for overfitting the validation set than does the choice among only 3 networks trained by maximum likelihood. Hence if you make the final choice of networks using the same validation set (B) that was used for early stopping, you give an unfair advantage to early stopping. If you are writing an article to compare various training methods, this bias could be a serious flaw. But if you are using NNs for some practical application, this bias might not matter at all, since you obtain an honest estimate of generalization error using C.

You could also argue that the second and third approaches are too wasteful in their use of data. This objection could be important if your sample contains 100 cases, but will probably be of little concern if your sample contains 100,000,000 cases. For small samples, there are other methods that make more efficient use of data; see "What are cross-validation and bootstrapping?"

References:

Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press.

Ripley, B.D. (1996) *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.

Tsypkin, Y. (1971), *Adaptation and Learning in Automatic Systems*, NY: Academic Press.

Subject: How are NNs related to statistical methods?

There is considerable overlap between the fields of neural networks and statistics. Statistics is concerned with data analysis. In neural network terminology, statistical inference means learning to generalize from noisy data. Some neural networks are not concerned with data analysis (e.g., those intended to model biological systems) and therefore have little to do with statistics. Some neural networks do not learn (e.g.,

Hopfield nets) and therefore have little to do with statistics. Some neural networks can learn successfully only from noise-free data (e.g., ART or the perceptron rule) and therefore would not be considered statistical methods. But most neural networks that can learn to generalize effectively from noisy data are similar or identical to statistical methods. For example:

- Feedforward nets with no hidden layer (including functional-link neural nets and higher-order neural nets) are basically generalized linear models.
- Feedforward nets with one hidden layer are closely related to projection pursuit regression.
- Probabilistic neural nets are identical to kernel discriminant analysis.
- Kohonen nets for adaptive vector quantization are very similar to k-means cluster analysis.
- Kohonen self-organizing maps are discrete approximations to principal curves and surfaces.
- Hebbian learning is closely related to principal component analysis.

Some neural network areas that appear to have no close relatives in the existing statistical literature are:

- Reinforcement learning (although this is treated in the operations research literature on Markov decision processes).
- Stopped training (the purpose and effect of stopped training are similar to shrinkage estimation, but the method is quite different).

Feedforward nets are a subset of the class of nonlinear regression and discrimination models. Statisticians have studied the properties of this general class but had not considered the specific case of feedforward neural nets before such networks were popularized in the neural network field. Still, many results from the statistical theory of nonlinear models apply directly to feedforward nets, and the methods that are commonly used for fitting nonlinear models, such as various Levenberg-Marquardt and conjugate gradient algorithms, can be used to train feedforward nets. The application of statistical theory to neural networks is explored in detail by Bishop (1995) and Ripley (1996). Several summary articles have also been published relating statistical models to neural networks, including Cheng and Titterington (1994), Kuan and White (1994), Ripley (1993, 1994), Sarle (1994), and several articles in Cherkassky, Friedman, and Wechsler (1994). Among the many statistical concepts important to neural nets is the bias/variance trade-off in nonparametric estimation, discussed by Geman, Bienenstock, and Doursat, R. (1992). Some more advanced results of statistical theory applied to neural networks are given by White (1989a, 1989b, 1990, 1992a) and White and Gallant (1992), reprinted in White (1992b).

While neural nets are often defined in terms of their algorithms or implementations, statistical methods are usually defined in terms of their results. The arithmetic mean, for example, can be computed by a (very simple) backprop net, by applying the usual formula SUM(x_i)/n, or by various other methods. What you get is still an arithmetic mean regardless of how you compute it. So a statistician would consider standard backprop, Quickprop, and Levenberg-Marquardt as different algorithms for implementing the same statistical model such as a feedforward net. On the other hand, different training criteria, such as least squares and cross entropy, are viewed by statisticians as fundamentally different estimation methods with different statistical properties.

It is sometimes claimed that neural networks, unlike statistical models, require no distributional assumptions. In fact, neural networks involve exactly the same sort of distributional assumptions as statistical models (Bishop, 1995), but statisticians study the consequences and importance of these assumptions while many neural networkers ignore them. For example, least-squares training methods are widely used by statisticians and neural networkers. Statisticians realize that least-squares training involves

implicit distributional assumptions in that least-squares estimates have certain optimality properties for noise that is normally distributed with equal variance for all training cases and that is independent between different cases. These optimality properties are consequences of the fact that least-squares estimation is maximum likelihood under those conditions. Similarly, cross-entropy is maximum likelihood for noise with a Bernoulli distribution. If you study the distributional assumptions, then you can recognize and deal with violations of the assumptions. For example, if you have normally distributed noise but some training cases have greater noise variance than others, then you may be able to use weighted least squares instead of ordinary least squares to obtain more efficient estimates.

Hundreds, perhaps thousands of people have run comparisons of neural nets with "traditional statistics" (whatever that means). Most such studies involve one or two data sets, and are of little use to anyone else unless they happen to be analyzing the same kind of data. But there is an impressive comparative study of supervised classification by Michie, Spiegelhalter, and Taylor (1994), which not only compares many classification methods on many data sets, but also provides unusually extensive analyses of the results. Another useful study on supervised classification by Lim, Loh, and Shih (1999) is available on-line. There is an excellent comparison of unsupervised Kohonen networks and k-means clustering by Balakrishnan, Cooper, Jacob, and Lewis (1994).

There are many methods in the statistical literature that can be used for flexible nonlinear modeling. These methods include:

- Polynomial regression (Eubank, 1999)
- Fourier series regression (Eubank, 1999; Haerdle, 1990)
- Wavelet smoothing (Donoho and Johnstone, 1995; Donoho, Johnstone, Kerkyacharian, and Picard, 1995)
- K-nearest neighbor regression and discriminant analysis (Haerdle, 1990; Hand, 1981, 1997; Ripley, 1996)
- Kernel regression and discriminant analysis (Eubank, 1999; Haerdle, 1990; Hand, 1981, 1982, 1997; Ripley, 1996)
- Local polynomial smoothing (Eubank, 1999; Wand and Jones, 1995; Fan and Gijbels, 1995)
- LOESS (Cleveland and Gross, 1991)
- Smoothing splines (such as thin-plate splines) (Eubank, 1999; Wahba, 1990; Green and Silverman, 1994; Haerdle, 1990)
- B-splines (Eubank, 1999)
- Tree-based models (CART, AID, etc.) (Haerdle, 1990; Lim, Loh, and Shih, 1997; Hand, 1997; Ripley, 1996)
- Multivariate adaptive regression splines (MARS) (Friedman, 1991)
- Projection pursuit (Friedman and Stuetzle, 1981; Haerdle, 1990; Ripley, 1996)
- Various Bayesian methods (Dey, 1998)
- GMDH (Farlow, 1984)

Why use neural nets rather than any of the above methods? There are many answers to that question depending on what kind of neural net you're interested in. The most popular variety of neural net, the MLP, tends to be useful in the same situations as projection pursuit regression, i.e.:

- the number of inputs is fairly large,
- many of the inputs are relevant, but
- most of the predictive information lies in a low-dimensional subspace.

The main advantage of MLPs over projection pursuit regression is that computing predicted values from MLPs is simpler and faster. Also, MLPs are better at learning moderately pathological functions than are many other methods with stronger smoothness assumptions (see ftp://ftp.sas.com/pub/neural/dojo/dojo.html) as long as the number of pathological features (such as discontinuities) in the function is not too large. For more discussion of the theoretical benefits of various types of neural nets, see How do MLPs compare with RBFs?

Communication between statisticians and neural net researchers is often hindered by the different terminology used in the two fields. There is a comparison of neural net and statistical jargon in ftp://ftp.sas.com/pub/neural/jargon

For free statistical software, see the StatLib repository at http://lib.stat.cmu.edu/ at Carnegie Mellon University.

There are zillions of introductory textbooks on statistics. One of the better ones is Moore and McCabe (1989). At an intermediate level, the books on linear regression by Weisberg (1985) and Myers (1986), on logistic regression by Hosmer and Lemeshow (1989), and on discriminant analysis by Hand (1981) can be recommended. At a more advanced level, the book on generalized linear models by McCullagh and Nelder (1989) is an essential reference, and the book on nonlinear regression by Gallant (1987) has much material relevant to neural nets.

Several introductory statistics texts are available on the web:

- David Lane, *HyperStat*, at http://www.ruf.rice.edu/~lane/hyperstat/contents.html
- Jan de Leeuw (ed.), *Statistics: The Study of Stability in Variation* , at http://www.stat.ucla.edu/textbook/
- StatSoft, Inc., *Electronic Statistics Textbook*, at http://www.statsoft.com/textbook/stathome.html
- David Stockburger, *Introductory Statistics: Concepts, Models, and Applications*, at http://www.psychstat.smsu.edu/sbk00.htm
- University of Newcastle (Australia) Statistics Department, *SurfStat Australia*, http://surfstat.newcastle.edu.au/surfstat/

A more advanced book covering many topics that are also relevant to NNs is:

- Frank Harrell, *REGRESSION MODELING STRATEGIES With Applications to Linear Models, Logistic Regression, and Survival Analysis*, at http://hesweb1.med.virginia.edu/biostat/rms/

References:

Balakrishnan, P.V., Cooper, M.C., Jacob, V.S., and Lewis, P.A. (1994) "A study of the classification capabilities of neural networks using unsupervised learning: A comparison with k-means clustering", Psychometrika, 59, 509-525.

Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press.

Cheng, B. and Titterington, D.M. (1994), "Neural Networks: A Review from a Statistical Perspective", Statistical Science, 9, 2-54.

Cherkassky, V., Friedman, J.H., and Wechsler, H., eds. (1994), *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, Berlin: Springer-Verlag.

Cleveland and Gross (1991), "Computational Methods for Local Regression," Statistics and Computing 1, 47-62.

Dey, D., ed. (1998) *Practical Nonparametric and Semiparametric Bayesian Statistics*, Springer Verlag.

Donoho, D.L., and Johnstone, I.M. (1995), "Adapting to unknown smoothness via wavelet shrinkage," J. of the American Statistical Association, 90, 1200-1224.

Donoho, D.L., Johnstone, I.M., Kerkyacharian, G., and Picard, D. (1995), "Wavelet shrinkage: asymptopia (with discussion)?" J. of the Royal Statistical Society, Series B, 57, 301-369.

Eubank, R.L. (1999), *Nonparametric Regression and Spline Smoothing*, 2nd ed., Marcel Dekker, ISBN 0-8247-9337-4.

Fan, J., and Gijbels, I. (1995), "Data-driven bandwidth selection in local polynomial: variable bandwidth and spatial adaptation," J. of the Royal Statistical Society, Series B, 57, 371-394.

Farlow, S.J. (1984), *Self-organizing Methods in Modeling: GMDH Type Algorithms*, NY: Marcel Dekker. (GMDH)

Friedman, J.H. (1991), "Multivariate adaptive regression splines", Annals of Statistics, 19, 1-141. (MARS)

Friedman, J.H. and Stuetzle, W. (1981) "Projection pursuit regression," J. of the American Statistical Association, 76, 817-823.

Gallant, A.R. (1987) *Nonlinear Statistical Models*, NY: Wiley.

Geman, S., Bienenstock, E. and Doursat, R. (1992), "Neural Networks and the Bias/Variance Dilemma", Neural Computation, 4, 1-58.

Green, P.J., and Silverman, B.W. (1994), *Nonparametric Regression and Generalized Linear Models: A Roughness Penalty Approach*, London: Chapman & Hall.

Haerdle, W. (1990), *Applied Nonparametric Regression*, Cambridge Univ. Press.

Hand, D.J. (1981) *Discrimination and Classification*, NY: Wiley.

Hand, D.J. (1982) *Kernel Discriminant Analysis*, Research Studies Press.

Hand, D.J. (1997) *Construction and Assessment of Classification Rules*, NY: Wiley.

Hill, T., Marquez, L., O'Connor, M., and Remus, W. (1994), "Artificial neural network models for forecasting and decision making," International J. of Forecasting, 10, 5-15.

Kuan, C.-M. and White, H. (1994), "Artificial Neural Networks: An Econometric Perspective", Econometric Reviews, 13, 1-91.

Kushner, H. & Clark, D. (1978), *Stochastic Approximation Methods for Constrained and Unconstrained Systems*, Springer-Verlag.

Lim, T.-S., Loh, W.-Y. and Shih, Y.-S. (1999?), "A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms," Machine Learning, forthcoming, preprint available at http://www.recursive-partitioning.com/mach1317.pdf, and appendix containing complete tables of error rates, ranks, and training times at http://www.recursive-partitioning.com/appendix.pdf

McCullagh, P. and Nelder, J.A. (1989) *Generalized Linear Models*, 2nd ed., London: Chapman & Hall.

Michie, D., Spiegelhalter, D.J. and Taylor, C.C., eds. (1994), *Machine Learning, Neural and Statistical Classification*, NY: Ellis Horwood; this book is out of print but available online at http://www.amsta.leeds.ac.uk/~charles/statlog/

Moore, D.S., and McCabe, G.P. (1989), *Introduction to the Practice of Statistics*, NY: W.H. Freeman.

Myers, R.H. (1986), *Classical and Modern Regression with Applications*, Boston: Duxbury Press.

Ripley, B.D. (1993), "Statistical Aspects of Neural Networks", in O.E. Barndorff-Nielsen, J.L. Jensen and W.S. Kendall, eds., *Networks and Chaos: Statistical and Probabilistic Aspects*, Chapman & Hall. ISBN 0 412 46530 2.

Ripley, B.D. (1994), "Neural Networks and Related Methods for Classification," Journal of the Royal Statistical Society, Series B, 56, 409-456.

Ripley, B.D. (1996) *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.

Sarle, W.S. (1994), "Neural Networks and Statistical Models," *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute, pp 1538-1550. (ftp://ftp.sas.com/pub/neural/neural1.ps)

Wahba, G. (1990), *Spline Models for Observational Data*, SIAM.

Wand, M.P., and Jones, M.C. (1995), *Kernel Smoothing*, London: Chapman & Hall.

Weisberg, S. (1985), *Applied Linear Regression*, NY: Wiley

White, H. (1989a), "Learning in Artificial Neural Networks: A Statistical Perspective," Neural Computation, 1, 425-464.

White, H. (1989b), "Some Asymptotic Results for Learning in Single Hidden Layer Feedforward Network Models", J. of the American Statistical Assoc., 84, 1008-1013.

White, H. (1990), "Connectionist Nonparametric Regression: Multilayer Feedforward Networks Can Learn Arbitrary Mappings," Neural Networks, 3, 535-550.

White, H. (1992a), "Nonparametric Estimation of Conditional Quantiles Using Neural Networks," in Page, C. and Le Page, R. (eds.), Computing Science and Statistics.

White, H., and Gallant, A.R. (1992), "On Learning the Derivatives of an Unknown Mapping with Multilayer Feedforward Networks," Neural Networks, 5, 129-138.

White, H. (1992b), *Artificial Neural Networks: Approximation and Learning Theory*, Blackwell.

- -

Next part is part 2 (of 7).

```
Archive-name: ai-faq/neural-nets/part2
Last-modified: 2002-05-20
URL: ftp://ftp.sas.com/pub/neural/FAQ2.html
Maintainer: saswss@unx.sas.com (Warren S. Sarle)
```


This is part 2 (of 7) of a monthly posting to the Usenet newsgroup comp.ai.neural-nets. See the part 1 of this posting for full information what it is all about.

========== Questions ==========

Subject: What are combination, activation, error, and objective functions?

Most neural networks involve combination, activation, error, and objective functions.

Combination functions

Each non-input unit in a neural network combines values that are fed into it via synaptic connections from other units, producing a single value called the "net input". There is no standard term in the NN literature for the function that combines values. In this FAQ, it will be called the "combination function". The combination function is a vector-to scalar function. Most NNs use either a linear combination function (as in MLPs) or a Euclidean distance combination function (as in RBF networks). There is a detailed discussion of networks using these two kinds of combination function under "How do MLPs compare with RBFs?"

Activation functions

Most units in neural networks transform their net input by using a scalar-to-scalar function called an

"activation function", yielding a value called the unit's "activation". Except possibly for output units, the activation value is fed via synpatic connections to one or more other units. The activation function is sometimes called a "transfer", and activation functions with a bounded range are often called "squashing" functions, such as the commonly used tanh (hyperbolic tangent) and logistic (1/(1+exp(-x)))) functions. If a unit does not transform its net input, it is said to have an "identity" or "linear" activation function. The reason for using non-identity activation functions is explained under "Why use activation functions?"

Error functions

Most methods for training supervised networks require a measure of the discrepancy between the networks output value and the target (desired output) value (even unsupervised networks may require such a measure of discrepancy--see "What does unsupervised learning learn?").

Let:

- `j` be an index for cases
- `X` or `X_j` be an input vector
- `W` be a collection (vector, matrix, or some more complicated structure) of weights and possibly other parameter estimates
- `y` or `y_j` be a target scalar
- `M(X,W)` be the output function computed by the network (the letter `M` is used to suggest "mean", "median", or "mode")
- `p` or `p_j = M(X_j,W)` be an output (the letter `p` is used to suggest "predicted value" or "posterior probability")
- `r` or `r_j = y_j - p_j` be a residual
- `Q(y,X,W)` be the case-wise error function written to show the dependence on the weights explicitly
- `L(y,p)` be the case-wise error function in simpler form where the weights are implicit (the letter `L` is used to suggest "loss" function)
- `D` be a list of indices designating a data set, including inputs and target values
  - `DL` designate the training (learning) set
  - `DV` designate the validation set
  - `DT` designate the test set
- `#(D)` be the number of elements (cases) in `D`
  - `NL` be the number of cases in the training (learning) set
  - `NV` be the number of cases in the validation set
  - `NT` be the number of cases in the test set
- `TQ(D,W)` be the total error function
- `AQ(D,W)` be the average error function

The difference between the target and output values for case `j, r_j = y_j - p_j,` is called the "residual" or "error". This is *NOT* the "error function"! Note that the residual can be either positive or negative, and negative residuals with large absolute values are typically considered just as bad as large positive residuals. Error functions, on the other hand, are defined so that bigger is worse.

Usually, an error function `Q(y,X,W)` is applied to each case and is defined in terms of the target and output values `Q(y,X,W) = L(y,M(X,W)) = L(y,p).` Error functions are also called "loss"

functions, especially when the two usages of the term "error" would sound silly when used together. For example, instead of the awkward phrase "squared-error error", you would typically use "squared-error loss" to mean an error function equal to the squared residual, `L(y,p) = (y - p)^2.` Another common error function is the classification loss for a binary target `y` in {0, 1}:

```
L(y,p) = 0 if |y-p| < 0.5
         1 otherwise
```

The error function for an entire data set is usually defined as the sum of the case-wise error functions for all the cases in a data set:

```
TQ(D,W) =  sum    Q(y_j,X_j,W)
          j in D
```

Thus, for squared-error loss, the total error is the sum of squared errors (i.e., residuals), abbreviated SSE. For classification loss, the total error is the number of misclassified cases.

It is often more convenient to work with the average (i.e, arithmetic mean) error:

```
AQ(D,W) = TQ(D,W)/#(D)
```

For squared-error loss, the average error is the mean or average of squared errors (i.e., residuals), abbreviated MSE or ASE (statisticians have a slightly different meaning for MSE in linear models, `TQ(D,W)/[#(D)-#(W)]`). For classification loss, the average error is the proportion of misclassified cases. The average error is also called the "empirical risk."

Using the average error instead of the total error is especially convenient when using batch backprop-type training methods where the user must supply a learning rate to multiply by the negative gradient to compute the change in the weights. If you use the gradient of the average error, the choice of learning rate will be relatively insensitive to the number of training cases. But if you use the gradient of the total error, you must use smaller learning rates for larger training sets. For example, consider any training set `DL_1` and a second training set `DL_2` created by duplicating every case in `DL_1.` For any set of weights, `DL_1` and `DL_2` have the same average error, but the total error of `DL_2` is twice that of `DL_1.` Hence the gradient of the total error of `DL_2` is twice the gradient for `DL_1`. So if you use the gradient of the total error, the learning rate for `DL_2` should be half the learning rate for `DL_1.` But if you use the gradient of the average error, you can use the same learning rate for both training sets, and you will get exactly the same results from batch training.

The term "error function" is commonly used to mean any of the functions, `Q(y,X,W), L(y,p), TQ(D,W),` or `AQ(D,W).` You can usually tell from the context which function is the intended meaning. The term "error surface" refers to `TQ(D,W)` or `AQ(D,W)` as a function of `W.`

Objective functions

The objective function is what you directly try to minimize during training.

Neural network training is often performed by trying to minimize the total error `TQ(DL,W)` or,

equivalently, the average error `AQ(DL,W)` for the training set, as a function of the weights `W`. However, as discussed in Part 3 of the FAQ, minimizing training error can lead to overfitting and poor generalization if the number of training cases is small relative to the complexity of the network. A common approach to improving generalization error is regularization, i.e., trying to minimize an objective function that is the sum of the total error function and a regularization function. The regularization function is a function of the weights `W` or of the output function `M(X,W)`. For example, in weight decay, the regularization function is the sum of squared weights. A crude form of Bayesian learning can be done using a regularization function that is the log of the prior density of the weights (weight decay is a special case of this). For more information on regularization, see Part 3 of the FAQ.

If no regularization function is used, the objective function is equal to the total or average error function (or perhaps some other monotone function thereof).

Subject: What are batch, incremental, on-line, off-line, deterministic, stochastic, adaptive, instantaneous, pattern, constructive, and sequential learning?

There are many ways to categorize learning methods. The distinctions are overlapping and can be confusing, and the terminology is used very inconsistently. This answer attempts to impose some order on the chaos, probably in vain.

Batch vs. Incremental Learning (also Instantaneous, Pattern, and Epoch)

Batch learning proceeds as follows:

```
Initialize the weights.
Repeat the following steps:
   Process all the training data.
   Update the weights.
```

Incremental learning proceeds as follows:

```
Initialize the weights.
Repeat the following steps:
   Process one training case.
   Update the weights.
```

In the above sketches, the exact meaning of "Process" and "Update" depends on the particular training algorithm and can be quite complicated for methods such as Levenberg-Marquardt (see "What are

conjugate gradients, Levenberg-Marquardt, etc.?"). Standard backprop (see What is backprop?) is quite simple, though. Batch standard backprop (without momentum) proceeds as follows:

```
Initialize the weights W.
Repeat the following steps:
   Process all the training data DL to compute the gradient
      of the average error function AQ(DL,W).
   Update the weights by subtracting the gradient times the
      learning rate.
```

Incremental standard backprop (without momentum) can be done as follows:

```
Initialize the weights W.
Repeat the following steps for j = 1 to NL:
   Process one training case (y_j,X_j) to compute the gradient
      of the error (loss) function Q(y_j,X_j,W).
   Update the weights by subtracting the gradient times the
      learning rate.
```

Alternatively, the index `j` can be chosen randomly each type the loop is executed, or `j` can be chosen from a random permutation.

The question of when to stop training is very complicated. Some of the possibilities are:

- Stop when the average error function for the training set becomes small.
- Stop when the gradient of the average error function for the training set becomes small.
- Stop when the average error function for the validation set starts to go up, and use the weights from the step that yielded the smallest validation error. For details, see "What is early stopping?"
- Stop when your boredom level is no longer tolerable.

It is very important *NOT* to use the following method--*which does not work*--but is often mistakenly used by beginners:

```
Initialize the weights W.
Repeat the following steps for j = 1 to NL:
   Repeat the following steps until Q(y_j,X_j,W) is small:
      Compute the gradient of Q(y_j,X_j,W).
      Update the weights by subtracting the gradient times the
         learning rate.
```

The reason this method does not work is that by the time you have finished processing the second case, the network will have forgotten what it learned about the first case, and when you are finished with the third case, the network will have forgotten what it learned about the first two cases, and so on. If you really need to use a method like this, see the section below on "Sequential Learning, Catastrophic Interference, and the Stability-Plasticity Dilemma".

The term "batch learning" is used quite consistently in the NN literature, but "incremental learning" is often

used for on-line, constructive, or sequential learning. The usage of "batch" and "incremental" in the NN FAQ follows Bertsekas and Tsitsiklis (1996), one of the few references that keeps the relevant concepts straight.

"Epoch learning" is synonymous with "batch learning."

"Instantaneous learning" and "pattern learning" are usually synonyms for incremental learning. "Instantaneous learning" is a misleading term because people often think it means learning instantaneously. "Pattern learning" is easily confused with "pattern recognition". Hence these terms are not used in the FAQ.

There are also intermediate methods, sometimes called mini-batch:

```
Initialize the weights.
Repeat the following steps:
   Process two or more, but not all training cases.
   Update the weights.
```

Conventional numerical optimization techniques (see "What are conjugate gradients, Levenberg-Marquardt, etc.?") are batch algorithms. Conventional stochastic approximation techniques (see below) are incremental algorithms. For a theoretical discussion comparing batch and incremental learning, see Bertsekas and Tsitsiklis (1996, Chapter 3).

For more information on incremental learning, see Saad (1998), but note that the authors in that volume do not reliably distinguish between on-line and incremental learning.

On-line vs. Off-line Learning

In off-line learning, all the data are stored and can be accessed repeatedly. Batch learning is always off-line.

In on-line learning, each case is discarded after it is processed and the weights are updated. On-line training is always incremental.

Incremental learning can be done either on-line or off-line.

With off-line learning, you can compute the objective function for any fixed set of weights, so you can see whether you are making progess in training. You can compute a minimum of the objective function to any desired precision. You can use a variety of algorithms for avoiding bad local minima, such as multiple random initializations or global optimization algorithms. You can compute the error function on a validation set and use early stopping or choose from different networks to improve generalization. You can use cross-validation and bootstrapping to estimate generalization error. You can compute prediction and confidence intervals (error bars). With on-line learning you can do none of these things because you cannot compute the objective function on the training set or the error function on the validation set for a fixed set of weights, since these data sets are not stored. Hence, on-line learning is generally more difficult and unreliable than off-line learning. Off-line incremental learning does not have all these problems of on-line learning, which is why it is important to distinguish between the concepts of on-line and incremental learning.

Some of the theoretical difficulties of on-line learning are alleviated when the assumptions of stochastic learning hold (see below) and training is assumed to proceed indefinitely.

For more information on on-line learning, see Saad (1998), but note that the authors in that volume do not reliably distinguish between on-line and incremental learning.

Deterministic, Stochastic, and Adaptive Learning

Deterministic learning is based on optimization of an objective function that can be recomputed many times and always produces the same value given the same weights. Deterministic learning is always off-line.

Stochastic methods are used when computation of the objective function is corrupted by noise. In particular, basic stochastic approximation is a form of on-line gradient descent learning in which the training cases are obtained by a stationary random process:

```
Initialize the weights.
Initialize the learning rate.
Repeat the following steps:
   Randomly select one (or possibly more) case(s)
      from the population.
   Update the weights by subtracting the gradient
      times the learning rate.
   Reduce the learning rate according to an
      appropriate schedule.
```

In stochastic on-line learning, the noise-corrupted objective function is the error function for any given case, assuming that the case-wise error function has some stationary random distribution. The learning rate must gradually decrease towards zero during training to guarantee convergence of the weights to a local minimum of the noiseless objective function. This gradual reduction of the learning rate is often called "annealing."

If the function that the network is trying to learn changes over time, the case-wise error function does not have a stationary random distribution. To allow the network to track changes over time, the learning rate must be kept strictly away from zero. Learning methods that track a changing environment are often called "adaptive" (as in adaptive vector quantization, Gersho and Gray, 1992) or "continuous" rather than "stochastic". There is a trade-off between accuracy and speed of adaptation. Adaptive learning does not converge in a stationary environment. Hence the long-run properties of stochastic learning and adaptive learning are quite different, even though the algorithms may differ only in the sequence of learning rates.

The object of adaptive learning is to forget the past when it is no longer relevant. If you want to remember the past in a changing learning environment, then you would be more interested in sequential learning (see below).

In stochastic learning with a suitably annealed learning rate, overtraining does not occur because the more you train, the more data you have, and the network converges toward a local optimum of the objective function for the entire population, not a local optimum for a finite training set. Of course, this conclusion

does not hold if you train by cycling through a finite training set instead of collecting new data on every step.

For a theoretical discussion of stochastic learning, see Bertsekas and Tsitsiklis (1996, Chapter 4). For further references on stochastic approximation, see "What is backprop?" For adaptive filtering, see Haykin (1996).

The term "adaptive learning" is sometimes used for gradient methods in which the learning rate is changed during training.

Constructive Learning (Growing networks)

Constructive learning adds units or connections to the network during training. Typically, constructive learning begins with a network with no hidden units, which is trained for a while. Then without altering the existing weights, one or more new hidden units are added to the network, training resumes, and so on. Many variations are possible, involving different patterns of connections and schemes for freezing and thawing weights. The most popular constructive algorithm is cascade correlation (Fahlman and Lebiere, 1990), of which many variations are possible (e.g., Littmann and Ritter, 1996; Prechelt, 1997). Various other constructive algorithms are summarized in Smieja (1993), Kwok and Yeung (1997; also other papers at http://info.cs.ust.hk/faculty/dyyeung/paper/cnn.html), and Reed and Marks (1999). For theory, see Baum (1989), Jones (1992), and Meir and Maiorov (1999). Lutz Prechelt has a bibliography on constructive algorithms at http://wwwipd.ira.uka.de/~prechelt/NN/construct.bib

Constructive algorithms can be highly effective for escaping bad local minima of the objective function, and are often much faster than algorithms for global optimization such as simulated annealing and genetic algorithms. A well-known example is the two-spirals problem, which is very difficult to learn with standard backprop but relatively easy to learn with cascade correlation (Fahlman and Lebiere, 1990). Some of the Donoho-Johnstone benchmarks (especially "bumps") are almost impossible to learn with standard backprop but can be learned very accurately with constructive algorithms (see ftp://ftp.sas.com/pub/neural/dojo/dojo.html.)

Constructive learning is commonly used to train multilayer perceptrons in which the activation functions are step functions. Such networks are difficult to train nonconstructively because the objective function is discontinuous and gradient-descent methods cannot be used. Several clever constructive algorithms (such as Upstart, Tiling, Marchand, etc.) have been devised whereby a multilayer perceptron is constructed by training a sequence of perceptrons, each of which is trained by some standard method such as the well-known perceptron or pocket algorithms. Most constructive algorithms of this kind are designed so that the training error goes to zero when the network gets sufficiently large. Such guarantees do not apply to the generalization error; you should guard against overfitting when you are using constructive algorithms just as with nonconstructive algorithms (see part 3 of the FAQ, especially "What is overfitting and how can I avoid it?")

Logically, you would expect "destructive" learning to start with a large network and delete units during training, but I have never seen this term used. The process of deleting units or connections is usually called "pruning" (Reed, 1993; Reed and Marks 1999). The term "selectionism" has also been used as the opposite of "constructivism" in cognitive neuroscience (Quartz and Sejnowski, 1997).

Sequential Learning, Catastrophic Interference, and the Stability-Plasticity Dilemma

"Sequential learning" sometimes means incremental learning but also refers to a very important problem in neuroscience (e.g., McClelland, McNaughton, and O'Reilly 1995). To reduce confusion, the latter usage should be preferred.

Sequential learning in its purest form operates on a sequence of training sets as follows:

```
Initialize the weights.
Repeat the following steps:
   Collect one or more training cases.
   Train the network on the current training set
      using any method.
   Discard the current training set and all other
      information related to training except the
      weights.
```

Pure sequential learning differs from on-line learning in that most on-line algorithms require storage of information in addition to the weights, such as the learning rate or aproximations to the objective function or Hessian Matrix. Such additional storage is not allowed in pure sequential learning.

Pure sequential learning is important as a model of how learning occurs in real, biological brains. Humans and other animals are often observed to learn new things without forgetting old things. But pure sequential learning tends not to work well in artificial neural networks. With a fixed architecture, distributed (rather than local) representations, and a training algorithm based on minimizing an objective function, sequential learning results in "catastrophic interference", because the minima of the objective function for one training set may be totally diferent than the minima for subsequent training sets. Hence each successive training set may cause the network to forget completely all previous training sets. This problem is also called the "stability-plasticity dilemma."

Successful sequential learning usually requires one or more of the following:

- Noise-free data.
- Constructive architectures.
- Local representations.
- Storage of extra information besides the weights (this is impure sequential learning and is not biologically plausible unless a a biological mechanism for storing the extra information is provided)

The connectionist literature on catastrophic interference seems oblivious to statistical and numerical theory, and much of the research is based on the ludicrous idea of using an autoassociative backprop network to model recognition memory. Some of the problems with this approach are explained by Sharkey and Sharkey (1995). Current research of the PDP variety is reviewed by French (1999). PDP remedies for catastrophic forgetting generally require cheating, i.e., storing information outside the network. For example, pseudorehearsal (Robins, 1995) requires storing the distribution of the inputs, although this fact is often overlooked. It appears that the only way to avoid catastrophic interference in a PDP-style network is

to combine two networks modularly: a fast-learning network to memorize data, and a slow-learning network to generalize from data memorized by the fast-learning network (McClelland, McNaughton, and O'Reilly 1995). The PDP literature virtually ignores the ART literature (See "What is ART?"), which provides a localist constructive solution to what the ART people call the "stability-plasticity dilemma." None of this research deals with sequential learning in a statistically sound manner, and many of the methods proposed for sequential learning require noise-free data. The statistical theory of sufficient statistics makes it obvious that efficient sequential learning requires the storage of additional information besides the weights in a standard feedforward network. I know of no references to this subject in the NN literature, but Bishop (1991) provided a mathematically sound treatment of a closely-related problem.

References:

Baum, E.B. (1989), "A proposal for more powerful learning algorithms," Neural Computation, 1, 201-207.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996), *Neuro-Dynamic Programming*, Belmont, MA: Athena Scientific, ISBN 1-886529-10-8.

Bishop, C. (1991), "A fast procedure for retraining the multilayer perceptron," International Journal of Neural Systems, 2, 229-236.

Fahlman, S.E., and Lebiere, C. (1990), "The Cascade-Correlation Learning Architecture", in Touretzky, D. S. (ed.), *Advances in Neural Information Processing Systems 2*,, Los Altos, CA: Morgan Kaufmann Publishers, pp. 524-532, ftp://archive.cis.ohio-state.edu/pub/neuroprose/fahlman.cascor-tr.ps.Z, http://www.rafael-ni.hpg.com.br/arquivos/fahlman-cascor.pdf

French, R.M. (1999), "Catastrophic forgetting in connectionist networks: Causes, consequences and solutions," Trends in Cognitive Sciences, 3, 128-135, http://www.fapse.ulg.ac.be/Lab/Trav/rfrench.html#TICS_cat_forget

Gersho, A. and Gray, R.M. (1992), *Vector Quantization and Signal Compression*, Boston: Kluwer Academic Publishers.

Haykin, S. (1996), *Adaptive Filter Theory*, Englewood Cliffs, NJ: Prentice-Hall.

Jones, L. (1992), "A simple lemma on greedy approximation in Hilbert space and convergence rate for projection pursuit regression and neural network training," Annals of Statistics, 20, 608-613.

Kwok, T.Y. and Yeung, D.Y. (1997), "Constructive algorithms for structure learning in feedforward neural networks for regression problems," IEEE Transactions on Neural Networks, volume 8, 630-645.

Littmann, E., and Ritter, H. (1996), "Learning and generalization in cascade network architectures," Neural Computation, 8, 1521-1539.

McClelland, J., McNaughton, B. and O'Reilly, R. (1995), "Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory," Psychological Review, 102, 419-457.

Meir, R., and Maiorov, V. (1999), "On the optimality of incremental neural network algorithms," in Kerans, M.S., Solla, S.A., amd Cohn, D.A. (eds.), *Advances in Neural Information Processing Systems 11*, Cambridge,MA: The MIT Press, pp. 295-301.

Prechelt, L. (1997), "Investigation of the CasCor Family of Learning Algorithms," Neural Networks, 10, 885-896, http://wwwipd.ira.uka.de/~prechelt/Biblio/#CasCor

Quartz, S.R., and Sejnowski, T.J. (1997), "The neural basis of cognitive development: A constructivist manifesto," Behavioral and Brain Sciences, 20, 537-596, ftp://ftp.princeton.edu/pub/harnad/BBS/bbs.quartz

Reed, R. (1993), "Pruning algorithms--A survey," IEEE Transactions on Neural Networks, 4, 740-747.

Reed, R.D., and Marks, R.J, II (1999), *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*, Cambridge, MA: The MIT Press, ISBN 0-262-18190-8.

Robins, A. (1995), "Catastrophic Forgetting, Rehearsal, and Pseudorehearsal," Connection Science, 7, 123-146.

Saad, D., ed. (1998), *On-Line Learning in Neural Networks*, Cambridge: Cambridge University Press.

Sharkey, N.E., and Sharkey, A.J.C. (1995), "An analysis of catastrophic interference," Connection Science, 7, 301-329.

Smieja, F.J. (1993), "Neural Network Constructive Algorithms: Trading Generalization for Learning Efficiency?" Circuits, Systems and Signal Processing, 12, 331-374, ftp://borneo.gmd.de/pub/as/janus/pre_6.ps

---

Subject: What is backprop?

"Backprop" is short for "backpropagation of error". The term *backpropagation* causes much confusion. Strictly speaking, *backpropagation* refers to the method for computing the gradient of the case-wise error function with respect to the weights for a feedforward network, a straightforward but elegant application of the chain rule of elementary calculus (Werbos 1974/1994). By extension, *backpropagation* or *backprop* refers to a training method that uses backpropagation to compute the gradient. By further extension, a *backprop* network is a feedforward network trained by backpropagation.

"Standard backprop" is a euphemism for the *generalized delta rule*, the training algorithm that was

popularized by Rumelhart, Hinton, and Williams in chapter 8 of Rumelhart and McClelland (1986), which remains the most widely used supervised training method for neural nets. The generalized delta rule (including momentum) is called the "heavy ball method" in the numerical analysis literature (Polyak 1964, 1987; Bertsekas 1995, 78-79).

Standard backprop can be used for both batch training (in which the weights are updated after processing the entire training set) and incremental training (in which the weights are updated after processing each case). For batch training, standard backprop usually converges (eventually) to a local minimum, if one exists. For incremental training, standard backprop does not converge to a stationary point of the error surface. To obtain convergence, the learning rate must be slowly reduced. This methodology is called "stochastic approximation" or "annealing".

The convergence properties of standard backprop, stochastic approximation, and related methods, including both batch and incremental algorithms, are discussed clearly and thoroughly by Bertsekas and Tsitsiklis (1996). For a practical discussion of backprop training in MLPs, Reed and Marks (1999) is the best reference I've seen.

For batch processing, there is no reason to suffer through the slow convergence and the tedious tuning of learning rates and momenta of standard backprop. Much of the NN research literature is devoted to attempts to speed up backprop. Most of these methods are inconsequential; two that are effective are Quickprop (Fahlman 1989) and RPROP (Riedmiller and Braun 1993). Concise descriptions of these algorithms are given by Schiffmann, Joost, and Werner (1994) and Reed and Marks (1999). But conventional methods for nonlinear optimization are usually faster and more reliable than any of the "props". See "What are conjugate gradients, Levenberg-Marquardt, etc.?".

Incremental backprop can be highly efficient for some large data sets if you select a good learning rate, but that can be difficult to do (see "What learning rate should be used for backprop?"). Also, incremental backprop is very sensitive to ill-conditioning (see ftp://ftp.sas.com/pub/neural/illcond/illcond.html).

For more on-line info on backprop, see Donald Tveter's Backpropagator's Review at http://www.dontveter.com/bpr/bpr.html or http://gannoo.uce.ac.uk/bpr/bpr.html.

References on backprop:

Bertsekas, D. P. (1995), *Nonlinear Programming*, Belmont, MA: Athena Scientific, ISBN 1-886529-14-0.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996), *Neuro-Dynamic Programming*, Belmont, MA: Athena Scientific, ISBN 1-886529-10-8.

Polyak, B.T. (1964), "Some methods of speeding up the convergence of iteration methods," Z. Vycisl. Mat. i Mat. Fiz., 4, 1-17.

Polyak, B.T. (1987), *Introduction to Optimization*, NY: Optimization Software, Inc.

Reed, R.D., and Marks, R.J, II (1999), *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*, Cambridge, MA: The MIT Press, ISBN 0-262-18190-8.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986), "Learning internal representations by error propagation", in Rumelhart, D.E. and McClelland, J. L., eds. (1986), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1, 318-362, Cambridge, MA: The MIT Press.

Werbos, P.J. (1974/1994), *The Roots of Backpropagation*, NY: John Wiley & Sons. Includes Werbos's 1974 Harvard Ph.D. thesis, *Beyond Regression.*

References on stochastic approximation:

Robbins, H. & Monro, S. (1951), "A Stochastic Approximation Method", Annals of Mathematical Statistics, 22, 400-407.

Kiefer, J. & Wolfowitz, J. (1952), "Stochastic Estimation of the Maximum of a Regression Function," Annals of Mathematical Statistics, 23, 462-466.

Kushner, H.J., and Yin, G. (1997), *Stochastic Approximation Algorithms and Applications*, NY: Springer-Verlag.

Kushner, H.J., and Clark, D. (1978), *Stochastic Approximation Methods for Constrained and Unconstrained Systems*, Springer-Verlag.

White, H. (1989), "Some Asymptotic Results for Learning in Single Hidden Layer Feedforward Network Models", J. of the American Statistical Assoc., 84, 1008-1013.

References on better props:

Fahlman, S.E. (1989), "Faster-Learning Variations on Back-Propagation: An Empirical Study", in Touretzky, D., Hinton, G, and Sejnowski, T., eds., *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann, 38-51.

Reed, R.D., and Marks, R.J, II (1999), *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*, Cambridge, MA: The MIT Press, ISBN 0-262-18190-8.

Riedmiller, M. (199?), "Advanced supervised learning in multi-layer perceptrons--from backpropagtion to adaptive learning algorithms," ftp://i11s16.ira.uka.de/pub/neuro/papers/riedml.csi94.ps.Z

Riedmiller, M. and Braun, H. (1993), "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm", *Proceedings of the IEEE International Conference on Neural Networks* 1993, San Francisco: IEEE.

Schiffmann, W., Joost, M., and Werner, R. (1994), "Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons," ftp://archive.cis.ohio-state.edu/pub/neuroprose/schiff.bp_speedup.ps.Z

- -

Subject: What learning rate should be used for backprop?

In standard backprop, too low a learning rate makes the network learn very slowly. Too high a learning rate makes the weights and objective function diverge, so there is no learning at all. If the objective function is quadratic, as in linear models, good learning rates can be computed from the Hessian matrix (Bertsekas and Tsitsiklis, 1996). If the objective function has many local and global optima, as in typical feedforward NNs with hidden units, the optimal learning rate often changes dramatically during the training process, since the Hessian also changes dramatically. Trying to train a NN using a constant learning rate is usually a tedious process requiring much trial and error. For some examples of how the choice of learning rate and momentum interact with numerical condition in some very simple networks, see ftp://ftp.sas.com/pub/neural/illcond/illcond.html

With batch training, there is no need to use a constant learning rate. In fact, there is no reason to use standard backprop at all, since vastly more efficient, reliable, and convenient batch training algorithms exist (see Quickprop and RPROP under "What is backprop?" and the numerous training algorithms mentioned under "What are conjugate gradients, Levenberg-Marquardt, etc.?").

Many other variants of backprop have been invented. Most suffer from the same theoretical flaw as standard backprop: the magnitude of the change in the weights (the step size) should NOT be a function of the magnitude of the gradient. In some regions of the weight space, the gradient is small and you need a large step size; this happens when you initialize a network with small random weights. In other regions of the weight space, the gradient is small and you need a small step size; this happens when you are close to a local minimum. Likewise, a large gradient may call for either a small step or a large step. Many algorithms try to adapt the learning rate, but any algorithm that multiplies the learning rate by the gradient to compute the change in the weights is likely to produce erratic behavior when the gradient changes abruptly. The great advantage of Quickprop and RPROP is that they do not have this excessive dependence on the magnitude of the gradient. Conventional optimization algorithms use not only the gradient but also second-order derivatives or a line search (or some combination thereof) to obtain a good step size.

With incremental training, it is much more difficult to concoct an algorithm that automatically adjusts the learning rate during training. Various proposals have appeared in the NN literature, but most of them don't work. Problems with some of these proposals are illustrated by Darken and Moody (1992), who unfortunately do not offer a solution. Some promising results are provided by by LeCun, Simard, and Pearlmutter (1993), and by Orr and Leen (1997), who adapt the momentum rather than the learning rate. There is also a variant of stochastic approximation called "iterate averaging" or "Polyak averaging" (Kushner and Yin 1997), which theoretically provides optimal convergence rates by keeping a running average of the weight values. I have no personal experience with these methods; if you have any solid evidence that these or other methods of automatically setting the learning rate and/or momentum in incremental training actually work in a wide variety of NN applications, please inform the FAQ maintainer (saswss@unx.sas.com).

References:

Bertsekas, D. P. and Tsitsiklis, J. N. (1996), *Neuro-Dynamic Programming*, Belmont, MA: Athena Scientific, ISBN 1-886529-10-8.

Darken, C. and Moody, J. (1992), "Towards faster stochastic gradient search," in Moody, J.E., Hanson, S.J., and Lippmann, R.P., eds. *Advances in Neural Information Processing Systems 4*, San Mateo, CA: Morgan Kaufmann Publishers, pp. 1009-1016.

Kushner, H.J., and Yin, G. (1997), *Stochastic Approximation Algorithms and Applications*, NY: Springer-Verlag.

LeCun, Y., Simard, P.Y., and Pearlmetter, B. (1993), "Automatic learning rate maximization by on-line estimation of the Hessian's eigenvectors," in Hanson, S.J., Cowan, J.D., and Giles, C.L. (eds.), *Advances in Neural Information Processing Systems 5*, San Mateo, CA: Morgan Kaufmann, pp. 156-163.

Orr, G.B. and Leen, T.K. (1997), "Using curvature information for fast stochastic search," in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) *Advances in Neural Information Processing Systems 9*, Cambrideg, MA: The MIT Press, pp. 606-612.

Subject: What are conjugate gradients, Levenberg-Marquardt, etc.?

Training a neural network is, in most cases, an exercise in numerical optimization of a usually nonlinear objective function ("objective function" means whatever function you are trying to optimize and is a slightly more general term than "error function" in that it may include other quantities such as penalties for weight decay; see "What are combination, activation, error, and objective functions?" for more details).

Methods of nonlinear optimization have been studied for hundreds of years, and there is a huge literature on the subject in fields such as numerical analysis, operations research, and statistical computing, e.g., Bertsekas (1995), Bertsekas and Tsitsiklis (1996), Fletcher (1987), and Gill, Murray, and Wright (1981). Masters (1995) has a good elementary discussion of conjugate gradient and Levenberg-Marquardt algorithms in the context of NNs.

There is no single best method for nonlinear optimization. You need to choose a method based on the characteristics of the problem to be solved. For objective functions with continuous second derivatives (which would include feedforward nets with the most popular differentiable activation functions and error functions), three general types of algorithms have been found to be effective for most practical purposes:

- For a small number of weights, stabilized Newton and Gauss-Newton algorithms, including various Levenberg-Marquardt and trust-region algorithms, are efficient. The memory required by these algorithms is proportional to the square of the number of weights.
- For a moderate number of weights, various quasi-Newton algorithms are efficient. The memory required by these algorithms is proportional to the square of the number of weights.

- For a large number of weights, various conjugate-gradient algorithms are efficient. The memory required by these algorithms is proportional to the number of weights.

Additional variations on the above methods, such as limited-memory quasi-Newton and double dogleg, can be found in textbooks such as Bertsekas (1995). Objective functions that are not continuously differentiable are more difficult to optimize. For continuous objective functions that lack derivatives on certain manifolds, such as ramp activation functions (which lack derivatives at the top and bottom of the ramp) and the least-absolute-value error function (which lacks derivatives for cases with zero error), subgradient methods can be used. For objective functions with discontinuities, such as threshold activation functions and the misclassification-count error function, Nelder-Mead simplex algorithm and various secant methods can be used. However, these methods may be very slow for large networks, and it is better to use continuously differentiable objective functions when possible.

All of the above methods find local optima--they are not guaranteed to find a global optimum. In practice, Levenberg-Marquardt often finds better optima for a variety of problems than do the other usual methods. I know of no theoretical explanation for this empirical finding.

For global optimization, there are also a variety of approaches. You can simply run any of the local optimization methods from numerous random starting points. Or you can try more complicated methods designed for global optimization such as simulated annealing or genetic algorithms (see Reeves 1993 and "What about Genetic Algorithms and Evolutionary Computation?"). Global optimization for neural nets is especially difficult because the number of distinct local optima can be astronomical.

In most applications, it is advisable to train several networks with different numbers of hidden units. Rather than train each network beginning with completely random weights, it is usually more efficient to use constructive learning (see "Constructive Learning (Growing networks"), where the weights that result from training smaller networks are used to initialize larger networks. Constructive learning can be done with any of the conventional optimization techniques or with the various "prop" methods, and can be very effective at finding good local optima at less expense than full-blown global optimization methods.

Another important consideration in the choice of optimization algorithms is that neural nets are often ill-conditioned (Saarinen, Bramley, and Cybenko 1993), especially when there are many hidden units. Algorithms that use only first-order information, such as steepest descent and standard backprop, are notoriously slow for ill-conditioned problems. Generally speaking, the more use an algorithm makes of second-order information, the better it will behave under ill-conditioning. The following methods are listed in order of increasing use of second-order information: steepest descent, conjugate gradients, quasi-Newton, Gauss-Newton, Newton-Raphson. Unfortunately, the methods that are better for severe ill-conditioning are the methods that are preferable for a small number of weights, and the methods that are preferable for a large number of weights are not as good at handling severe ill-conditioning. Therefore for networks with many hidden units, it is advisable to try to alleviate ill-conditioning by standardizing input and target variables, choosing initial values from a reasonable range, and using weight decay or Bayesian regularization methods. For more discussion of ill-conditioning in neural nets, see ftp://ftp.sas.com/pub/neural/illcond/illcond.html

Writing programs for conventional optimization algorithms is considerably more difficult than writing programs for standard backprop. As "Jive Dadson" said in comp.ai.neural-nets:

Writing a good conjugate gradient algorithm turned out to be a lot of work. It's not even easy to find all the technical info you need. The devil is in the details. There are a lot of details.

If you are not experienced in both programming and numerical analysis, use software written by professionals instead of trying to write your own. For a survey of optimization software, see Moré and Wright (1993).

For more on-line information on numerical optimization see:

- The kangaroos, a nontechnical description of various optimization methods, at ftp://ftp.sas.com/pub/neural/kangaroos.
- Sam Roweis's paper on Levenberg-Marquardt at http://www.gatsby.ucl.ac.uk/~roweis/notes.html
- Jonathan Shewchuk's paper on conjugate gradients, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain," at http://www.cs.cmu.edu/~jrs/jrspapers.html
- Lester Ingber's page on Adaptive Simulated Annealing (ASA), karate, etc. at http://www.ingber.com/ or http://www.alumni.caltech.edu/~ingber/
- The Netlib repository, http://www.netlib.org/, containing freely available software, documents, and databases of interest to the numerical and scientific computing community.
- The linear and nonlinear programming FAQs at http://www.mcs.anl.gov/home/otc/Guide/faq/.
- Arnold Neumaier's page on global optimization at http://solon.cma.univie.ac.at/~neum/glopt.html.
- Simon Streltsov's page on global optimization at http://cad.bu.edu/go.

References:

Bertsekas, D. P. (1995), *Nonlinear Programming*, Belmont, MA: Athena Scientific, ISBN 1-886529-14-0.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996), *Neuro-Dynamic Programming*, Belmont, MA: Athena Scientific, ISBN 1-886529-10-8.

Fletcher, R. (1987) *Practical Methods of Optimization*, NY: Wiley.

Gill, P.E., Murray, W. and Wright, M.H. (1981) *Practical Optimization*, Academic Press: London.

Levenberg, K. (1944) "A method for the solution of certain problems in least squares," Quart. Appl. Math., 2, 164-168.

Marquardt, D. (1963) "An algorithm for least-squares estimation of nonlinear parameters," SIAM J. Appl. Math., 11, 431-441. This is the third most frequently cited paper in all the mathematical sciences.

Masters, T. (1995) *Advanced Algorithms for Neural Networks: A C++ Sourcebook*, NY: John Wiley and Sons, ISBN 0-471-10588-0

Moré, J.J. (1977) "The Levenberg-Marquardt algorithm: implementation and theory," in Watson, G.A., ed., *Numerical Analysis*, Lecture Notes in Mathematics 630, Springer-Verlag, Heidelberg, 105-

116.

Moré, J.J. and Wright, S.J. (1993), *Optimization Software Guide,* Philadelphia: SIAM, ISBN 0-89871-322-6.

Reed, R.D., and Marks, R.J, II (1999), *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks,* Cambridge, MA: The MIT Press, ISBN 0-262-18190-8.

Reeves, C.R., ed. (1993) *Modern Heuristic Techniques for Combinatorial Problems,* NY: Wiley.

Rinnooy Kan, A.H.G., and Timmer, G.T., (1989) *Global Optimization: A Survey,* International Series of Numerical Mathematics, vol. 87, Basel: Birkhauser Verlag.

Saarinen, S., Bramley, R., and Cybenko, G. (1993), "Ill-conditioning in neural network training problems," Siam J. of Scientific Computing, 14, 693-714.

Subject: How does ill-conditioning affect NN training?

Numerical condition is one of the most fundamental and important concepts in numerical analysis. Numerical condition affects the speed and accuracy of most numerical algorithms. Numerical condition is especially important in the study of neural networks because ill-conditioning is a common cause of slow and inaccurate results from backprop-type algorithms. For more information, see: ftp://ftp.sas.com/pub/neural/illcond/illcond.html

---

Subject: How should categories be encoded?

First, consider unordered categories. If you want to classify cases into one of C categories (i.e. you have a categorical target variable), use 1-of-C coding. That means that you code C binary (0/1) target variables corresponding to the C categories. Statisticians call these "dummy" variables. Each dummy variable is given the value zero except for the one corresponding to the correct category, which is given the value one. Then use a softmax output activation function (see "What is a softmax activation function?") so that the net, if properly trained, will produce valid posterior probability estimates (McCullagh and Nelder, 1989; Finke and Müller, 1994). If the categories are Red, Green, and Blue, then the data would look like this:

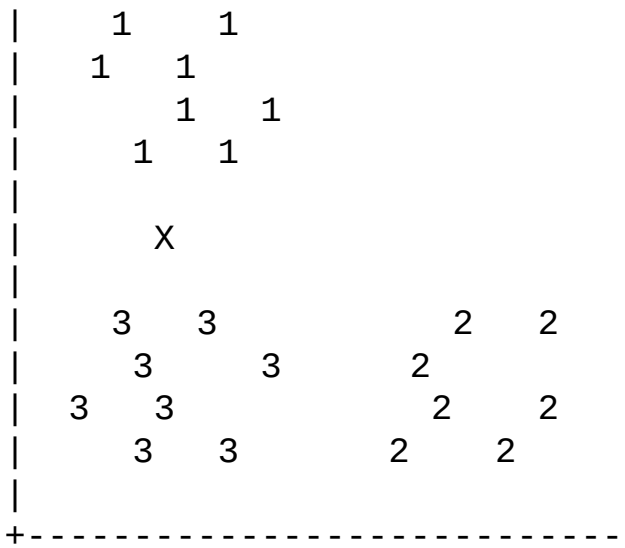
```
Category  Dummy variables
--------  ---------------
 Red        1   0   0
 Green      0   1   0
 Blue       0   0   1
```

When there are only two categories, it is simpler to use just one dummy variable with a logistic output activation function; this is equivalent to using softmax with two dummy variables.

The common practice of using target values of .1 and .9 instead of 0 and 1 prevents the outputs of the network from being directly interpretable as posterior probabilities, although it is easy to rescale the outputs to produce probabilities (Hampshire and Pearlmutter, 1991, figure 3). This practice has also been advocated on the grounds that infinite weights are required to obtain outputs of 0 or 1 from a logistic function, but in fact, weights of about 10 to 30 will produce outputs close enough to 0 and 1 for all practical purposes, assuming standardized inputs. Large weights will not cause overflow if the activation functions are coded properly; see "How to avoid overflow in the logistic function?"

Another common practice is to use a logistic activation function for each output. Thus, the outputs are not constrained to sum to one, so they are not admissible posterior probability estimates. The usual justification advanced for this procedure is that if a test case is not similar to any of the training cases, all of the outputs will be small, indicating that the case cannot be classified reliably. This claim is incorrect, since a test case that is not similar to any of the training cases will require the net to extrapolate, and extrapolation is thoroughly unreliable; such a test case may produce all small outputs, all large outputs, or any combination of large and small outputs. If you want a classification method that detects novel cases for which the classification may not be reliable, you need a method based on probability density estimation. For example, see "What is PNN?".

It is very important *not* to use a single variable for an unordered categorical target. Suppose you used a single variable with values 1, 2, and 3 for red, green, and blue, and the training data with two inputs looked like this:



Consider a test point located at the X. The correct output would be that X has about a 50-50 chance of being a 1 or a 3. But if you train with a single target variable with values of 1, 2, and 3, the output for X will be the average of 1 and 3, so the net will say that X is definitely a 2!

If you are willing to forego the simple posterior-probability interpretation of outputs, you can try more elaborate coding schemes, such as the error-correcting output codes suggested by Dietterich and Bakiri (1995).

For an input with categorical values, you can use 1-of-(C-1) coding if the network has a bias unit. This is just like 1-of-C coding, except that you omit one of the dummy variables (doesn't much matter which one). Using all C of the dummy variables creates a linear dependency on the bias unit, which is not advisable unless you are using weight decay or Bayesian learning or some such thing that requires all C weights to be treated on an equal basis. 1-of-(C-1) coding looks like this:

```
Category  Dummy variables
--------  ---------------
 Red        1   0
 Green      0   1
 Blue       0   0
```

If you use 1-of-C or 1-of-(C-1) coding, it is important to standardize the dummy inputs; see "Should I standardize the input variables?" "Why not code binary inputs as 0 and 1?" for details.

Another possible coding is called "effects" coding or "deviations from means" coding in statistics. It is like 1-of-(C-1) coding, except that when a case belongs to the category for the omitted dummy variable, all of the dummy variables are set to -1, like this:

```
Category  Dummy variables
--------  ---------------
 Red        1   0
 Green      0   1
 Blue      -1  -1
```

As long as a bias unit is used, any network with effects coding can be transformed into an equivalent network with 1-of-(C-1) coding by a linear transformation of the weights, so if you train to a global optimum, there will be no difference in the outputs for these two types of coding. One advantage of effects coding is that the dummy variables require no standardizing, since effects coding directly produces values that are approximately standardized.

If you are using weight decay, you want to make sure that shrinking the weights toward zero biases ('bias' in the statistical sense) the net in a sensible, usually smooth, way. If you use 1 of C-1 coding for an input, weight decay biases the output for the C-1 categories towards the output for the 1 omitted category, which is probably not what you want, although there might be special cases where it would make sense. If you use 1 of C coding for an input, weight decay biases the output for all C categories roughly towards the mean output for all the categories, which is smoother and usually a reasonable thing to do.

Now consider ordered categories. For inputs, some people recommend a "thermometer code" (Smith, 1996; Masters, 1993) like this:

```
Category  Dummy variables
--------  ---------------
 Red        1   1   1
 Green      0   1   1
 Blue       0   0   1
```

However, thermometer coding is equivalent to 1-of-C coding, in that for any network using 1-of-C coding, there exists a network with thermometer coding that produces identical outputs; the weights in the thermometer-encoded network are just the differences of successive weights in the 1-of-C-encoded network. To get a genuinely ordinal representation, you must constrain the weights connecting the dummy variables to the hidden units to be nonnegative (except for the first dummy variable). Another approach that makes some use of the order information is to use weight decay or Bayesian learning to encourage the the weights for all but the first dummy variable to be small.

It is often effective to represent an ordinal input as a single variable like this:

Category	Input
Red	1
Green	2
Blue	3

Although this representation involves only a single quantitative input, given enough hidden units, the net is capable of computing nonlinear transformations of that input that will produce results equivalent to any of the dummy coding schemes. But using a single quantitative input makes it easier for the net to use the order of the categories to generalize when that is appropriate.

B-splines provide a way of coding ordinal inputs into fewer than C variables while retaining information about the order of the categories. See Brown and Harris (1994) or Gifi (1990, 365-370).

Target variables with ordered categories require thermometer coding. The outputs are thus cumulative probabilities, so to obtain the posterior probability of any category except the first, you must take the difference between successive outputs. It is often useful to use a proportional-odds model, which ensures that these differences are positive. For more details on ordered categorical targets, see McCullagh and Nelder (1989, chapter 5).

References:

Brown, M., and Harris, C. (1994), *Neurofuzzy Adaptive Modelling and Control*, NY: Prentice Hall.

Dietterich, T.G. and Bakiri, G. (1995), "Error-correcting output codes: A general method for improving multiclass inductive learning programs," in Wolpert, D.H. (ed.), *The Mathematics of Generalization: The Proceedings of the SFI/CNLS Workshop on Formal Approaches to Supervised Learning*, Santa Fe Institute Studies in the Sciences of Complexity, Volume XX, Reading, MA: Addison-Wesley, pp. 395-407.

Finke, M. and Müller, K.-R. (1994), "Estimating a-posteriori probabilities using stochastic network models," in Mozer, M., Smolensky, P., Touretzky, D., Elman, J., and Weigend, A. (eds.), *Proceedings of the 1993 Connectionist Models Summer School*, Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 324-331.

Gifi, A. (1990), *Nonlinear Multivariate Analysis*, NY: John Wiley & Sons, ISBN 0-471-92620-5.

Hampshire II, J.B., and Pearlmutter, B. (1991), "Equivalence proofs for multi-layer perceptron classifiers and the Bayesian discriminant function," in Touretzky, D.S., Elman, J.L., Sejnowski, T.J., and Hinton, G.E. (eds.), *Connectionist Models: Proceedings of the 1990 Summer School*, San Mateo, CA: Morgan Kaufmann, pp.159-172.

Masters, T. (1993). *Practical Neural Network Recipes in C++*, San Diego: Academic Press.

McCullagh, P. and Nelder, J.A. (1989) *Generalized Linear Models*, 2nd ed., London: Chapman & Hall.

Smith, M. (1996). *Neural Networks for Statistical Modeling*, Boston: International Thomson Computer Press, ISBN 1-850-32842-0.

Subject: Why not code binary inputs as 0 and 1?

The importance of standardizing input variables is discussed in detail under "Should I standardize the input variables?" But for the benefit of those people who don't believe in theory, here is an example using the 5-bit parity problem. The unstandardized data are:

x1	x2	x3	x4	x5	target
0	0	0	0	0	0
1	0	0	0	0	1
0	1	0	0	0	1
1	1	0	0	0	0
0	0	1	0	0	1
1	0	1	0	0	0
0	1	1	0	0	0
1	1	1	0	0	1
0	0	0	1	0	1
1	0	0	1	0	0
0	1	0	1	0	0
1	1	0	1	0	1
0	0	1	1	0	0
1	0	1	1	0	1
0	1	1	1	0	1
1	1	1	1	0	0
0	0	0	0	1	1
1	0	0	0	1	0
0	1	0	0	1	0
1	1	0	0	1	1
0	0	1	0	1	0
1	0	1	0	1	1
0	1	1	0	1	1
1	1	1	0	1	0

```
 0     0     0     1     1       0
 1     0     0     1     1       1
 0     1     0     1     1       1
 1     1     0     1     1       0
 0     0     1     1     1       1
 1     0     1     1     1       0
 0     1     1     1     1       0
 1     1     1     1     1       1
```

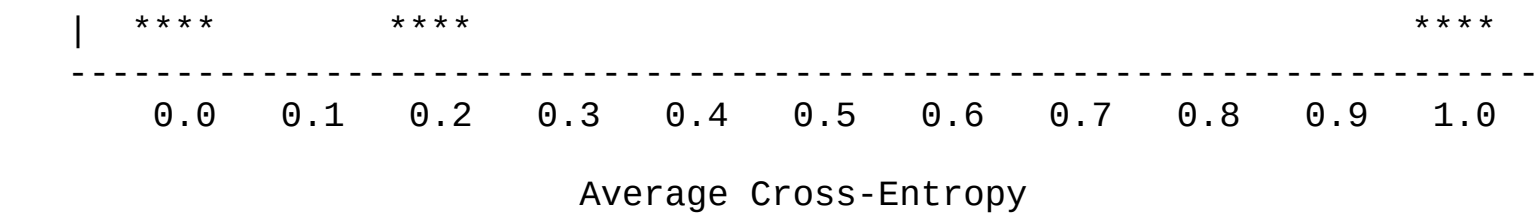
The network characteristics were:

```
Inputs:                         5
Hidden units:                   5
Outputs:                        5
Activation for hidden units:  tanh
Activation for output units:  logistic
Error function:               cross-entropy
Initial weights:              random normal with mean=0,
                                 st.dev.=1/sqrt(5) for input-to-hidden
                                        =1         for hidden-to-output
Training method:              batch standard backprop
Learning rate:                0.1
Momentum:                     0.9
Minimum training iterations:  100
Maximum training iterations:  10000
```

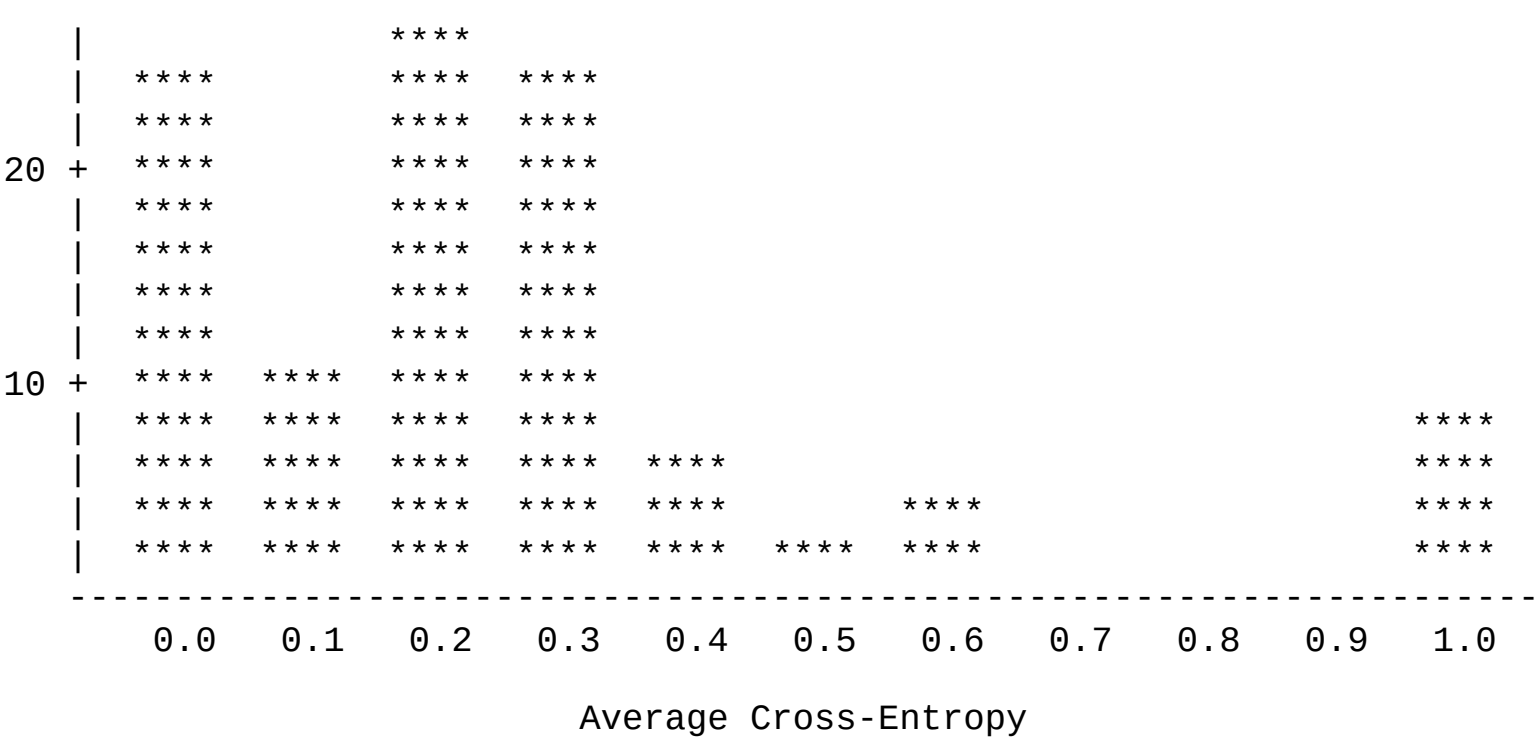
One hundred networks were trained from different random initial weights. The following bar chart shows the distribution of the average cross-entropy after training:

```
Frequency

   |                                                                  ****
   |                                                                  ****
80 +                                                                  ****
   |                                                                  ****
   |                                                                  ****
   |                                                                  ****
60 +                                                                  ****
   |                                                                  ****
   |                                                                  ****
   |                                                                  ****
40 +                                                                  ****
   |                                                                  ****
   |                                                                  ****
   |                                                                  ****
20 +                                                                  ****
   |                                                                  ****
   |                                                                  ****
```



As you can see, very few networks found a good (near zero) local optimum. Recoding the inputs from {0,1} to {-1,1} produced the following distribution of the average cross-entropy after training:



The results are dramatically better. The difference is due to simple geometry. The initial hyperplanes pass fairly near the origin. If the data are centered near the origin (as with {-1,1} coding), the initial hyperplanes will cut through the data in a variety of directions. If the data are offset from the origin (as with {0,1} coding), many of the initial hyperplanes will miss the data entirely, and those that pass through the data will provide a only a limited range of directions, making it difficult to find local optima that use hyperplanes that go in different directions. If the data are far from the origin (as with {9,10} coding), most of the initial hyperplanes will miss the data entirely, which will cause most of the hidden units to saturate and make any learning difficult. See "Should I standardize the input variables?" for more information.

Subject: Why use a bias/threshold?

Sigmoid hidden and output units usually use a "bias" or "threshold" term in computing the net input to the unit. For a linear output unit, a bias term is equivalent to an intercept in a regression model.

A bias term can be treated as a connection weight from a special unit with a constant, nonzero activation value. The term "bias" is usually used with respect to a "bias unit" with a constant value of one. The term

"threshold" is usually used with respect to a unit with a constant value of negative one. Not all authors follow this distinction. Regardless of the terminology, whether biases or thresholds are added or subtracted has no effect on the performance of the network.

The single bias unit is connected to every hidden or output unit that needs a bias term. Hence the bias terms can be learned just like other weights.

Consider a multilayer perceptron with any of the usual sigmoid activation functions. Choose any hidden unit or output unit. Let's say there are N inputs to that unit, which define an N-dimensional space. The given unit draws a hyperplane through that space, producing an "on" output on one side and an "off" output on the other. (With sigmoid units the plane will not be sharp -- there will be some gray area of intermediate values near the separating plane -- but ignore this for now.)

The weights determine where this hyperplane lies in the input space. Without a bias term, this separating hyperplane is constrained to pass through the origin of the space defined by the inputs. For some problems that's OK, but in many problems the hyperplane would be much more useful somewhere else. If you have many units in a layer, they share the same input space and without bias they would ALL be constrained to pass through the origin.

The "universal approximation" property of multilayer perceptrons with most commonly-used hidden-layer activation functions does not hold if you omit the bias terms. But Hornik (1993) shows that a sufficient condition for the universal approximation property without biases is that no derivative of the activation function vanishes at the origin, which implies that with the usual sigmoid activation functions, a fixed nonzero bias term can be used instead of a trainable bias.

Typically, every hidden and output unit has its own bias term. The main exception to this is when the activations of two or more units in one layer always sum to a nonzero constant. For example, you might scale the inputs to sum to one (see Should I standardize the input cases?), or you might use a normalized RBF function in the hidden layer (see How do MLPs compare with RBFs?). If there do exist units in one layer whose activations sum to a nonzero constant, then any subsequent layer does not need bias terms if it receives connections from the units that sum to a constant, since using bias terms in the subsequent layer would create linear dependencies.

If you have a large number of hidden units, it may happen that one or more hidden units "saturate" as a result of having large incoming weights, producing a constant activation. If this happens, then the saturated hidden units act like bias units, and the output bias terms are redundant. However, you should not rely on this phenomenon to avoid using output biases, since networks without output biases are usually ill-conditioned (see ftp://ftp.sas.com/pub/neural/illcond/illcond.html) and harder to train than networks that use output biases.

Regarding bias-like terms in RBF networks, see "How do MLPs compare with RBFs?"

Reference:

Hornik, K. (1993), "Some new results on neural network approximation," Neural Networks, 6, 1069-1072.

Subject: Why use activation functions?

Activation functions for the hidden units are needed to introduce nonlinearity into the network. Without nonlinearity, hidden units would not make nets more powerful than just plain perceptrons (which do not have any hidden units, just input and output units). The reason is that a linear function of linear functions is again a linear function. However, it is the nonlinearity (i.e, the capability to represent nonlinear functions) that makes multilayer networks so powerful. Almost any nonlinear function does the job, except for polynomials. For backpropagation learning, the activation function must be differentiable, and it helps if the function is bounded; the sigmoidal functions such as logistic and tanh and the Gaussian function are the most common choices. Functions such as tanh or arctan that produce both positive and negative values tend to yield faster training than functions that produce only positive values such as logistic, because of better numerical conditioning (see ftp://ftp.sas.com/pub/neural/illcond/illcond.html).

For hidden units, sigmoid activation functions are usually preferable to threshold activation functions. Networks with threshold units are difficult to train because the error function is stepwise constant, hence the gradient either does not exist or is zero, making it impossible to use backprop or more efficient gradient-based training methods. Even for training methods that do not use gradients--such as simulated annealing and genetic algorithms--sigmoid units are easier to train than threshold units. With sigmoid units, a small change in the weights will usually produce a change in the outputs, which makes it possible to tell whether that change in the weights is good or bad. With threshold units, a small change in the weights will often produce no change in the outputs.

For the output units, you should choose an activation function suited to the distribution of the target values:

- For binary (0/1) targets, the logistic function is an excellent choice (Jordan, 1995).
- For categorical targets using 1-of-C coding, the softmax activation function is the logical extension of the logistic function.
- For continuous-valued targets with a bounded range, the logistic and tanh functions can be used, provided you either scale the outputs to the range of the targets or scale the targets to the range of the output activation function ("scaling" means multiplying by and adding appropriate constants).
- If the target values are positive but have no known upper bound, you can use an exponential output activation function, but beware of overflow.
- For continuous-valued targets with no known bounds, use the identity or "linear" activation function (which amounts to no activation function) unless you have a very good reason to do otherwise.

There are certain natural associations between output activation functions and various noise distributions which have been studied by statisticians in the context of generalized linear models. The output activation function is the inverse of what statisticians call the "link function". See:

McCullagh, P. and Nelder, J.A. (1989) *Generalized Linear Models*, 2nd ed., London: Chapman & Hall.

Jordan, M.I. (1995), "Why the logistic function? A tutorial discussion on probabilities and neural networks", MIT Computational Cognitive Science Report 9503,

http://www.cs.berkeley.edu/~jordan/papers/uai.ps.Z.

For more information on activation functions, see Donald Tveter's Backpropagator's Review.

Subject: How to avoid overflow in the logistic function?

The formula for the logistic activation function is often written as:

```
netoutput = 1 / (1+exp(-netinput));
```

But this formula can produce floating-point overflow in the exponential function if you program it in this simple form. To avoid overflow, you can do this:

```
if (netinput < -45) netoutput = 0;
else if (netinput > 45) netoutput = 1;
else netoutput = 1 / (1+exp(-netinput));
```

The constant 45 will work for double precision on all machines that I know of, but there may be some bizarre machines where it will require some adjustment. Other activation functions can be handled similarly.

Subject: What is a softmax activation function?

If you want the outputs of a network to be interpretable as posterior probabilities for a categorical target variable, it is highly desirable for those outputs to lie between zero and one and to sum to one. The purpose of the softmax activation function is to enforce these constraints on the outputs. Let the net input to each output unit be `q_i, i=1,...,c,` where `c` is the number of categories. Then the softmax output `p_i` is:

```
         exp(q_i)
p_i = ------------
       c
      sum exp(q_j)
      j=1
```

Unless you are using weight decay or Bayesian estimation or some such thing that requires the weights to be treated on an equal basis, you can choose any one of the output units and leave it completely unconnected--just set the net input to 0. Connecting all of the output units will just give you redundant weights and will slow down training. To see this, add an arbitrary constant z to each net input and you get:

$$p_i = \frac{\exp(q_i+z)}{\sum_{j=1}^{c} \exp(q_j+z)} = \frac{\exp(q_i)\exp(z)}{\sum_{j=1}^{c} \exp(q_j)\exp(z)} = \frac{\exp(q_i)}{\sum_{j=1}^{c} \exp(q_j)}$$

so nothing changes. Hence you can always pick one of the output units, and add an appropriate constant to each net input to produce any desired net input for the selected output unit, which you can choose to be zero or whatever is convenient. You can use the same trick to make sure that none of the exponentials overflows.

Statisticians usually call softmax a "multiple logistic" function. It reduces to the simple logistic function when there are only two categories. Suppose you choose to set `q_2` to 0. Then

$$p_1 = \frac{\exp(q_1)}{\sum_{j=1}^{c} \exp(q_j)} = \frac{\exp(q_1)}{\exp(q_1) + \exp(0)} = \frac{1}{1 + \exp(-q_1)}$$

and `p_2,` of course, is `1-p_1.`

The softmax function derives naturally from log-linear models and leads to convenient interpretations of the weights in terms of odds ratios. You could, however, use a variety of other nonnegative functions on the real line in place of the exp function. Or you could constrain the net inputs to the output units to be nonnegative, and just divide by the sum--that's called the Bradley-Terry-Luce model.

The softmax function is also used in the hidden layer of normalized radial-basis-function networks; see "How do MLPs compare with RBFs?"

References:

Bridle, J.S. (1990a). Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. In: F.Fogleman Soulie and J.Herault (eds.), Neurocomputing: Algorithms, Architectures and Applications, Berlin: Springer-Verlag, pp. 227-236.

Bridle, J.S. (1990b). Training Stochastic Model Recognition Algorithms as Networks can lead to Maximum Mutual Information Estimation of Parameters. In: D.S.Touretzky (ed.), Advances in Neural Information Processing Systems 2, San Mateo: Morgan Kaufmann, pp. 211-217.

McCullagh, P. and Nelder, J.A. (1989) *Generalized Linear Models*, 2nd ed., London: Chapman & Hall. See Chapter 5.

Subject: What is the curse of dimensionality?

Answer by Janne Sinkkonen.

Curse of dimensionality (Bellman 1961) refers to the exponential growth of hypervolume as a function of dimensionality. In the field of NNs, curse of dimensionality expresses itself in two related problems:

1. Many NNs can be thought of mappings from an input space to an output space. Thus, loosely speaking, an NN needs to somehow "monitor", cover or represent every part of its input space in order to know how that part of the space should be mapped. Covering the input space takes resources, and, in the most general case, the amount of resources needed is proportional to the hypervolume of the input space. The exact formulation of "resources" and "part of the input space" depends on the type of the network and should probably be based on the concepts of information theory and differential geometry.

   As an example, think of a vector quantization (VQ). In VQ, a set of units competitively learns to represents an input space (this is like Kohonen's Self-Organizing Map but without topography for the units). Imagine a VQ trying to share its units (resources) more or less equally over hyperspherical input space. One could argue that the average distance from a random point of the space to the nearest network unit measures the goodness of the representation: the shorter the distance, the better is the represention of the data in the sphere. It is intuitively clear (and can be experimentally verified) that the total number of units required to keep the average distance constant increases exponentially with the dimensionality of the sphere (if the radius of the sphere is fixed).

   The curse of dimensionality causes networks with lots of irrelevant inputs to be behave relatively badly: the dimension of the input space is high, and the network uses almost all its resources to represent irrelevant portions of the space.

   Unsupervised learning algorithms are typically prone to this problem - as well as conventional RBFs. A partial remedy is to preprocess the input in the right way, for example by scaling the components according to their "importance".

2. Even if we have a network algorithm which is able to focus on important portions of the input space, the higher the dimensionality of the input space, the more data may be needed to find out what is important and what is not.

A priori information can help with the curse of dimensionality. Careful feature selection and scaling of the inputs fundamentally affects the severity of the problem, as well as the selection of the neural network model. For classification purposes, only the borders of the classes are important to represent accurately.

References:

Bellman, R. (1961), *Adaptive Control Processes: A Guided Tour*, Princeton University Press.

Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press, section 1.4.

Scott, D.W. (1992), *Multivariate Density Estimation*, NY: Wiley.

Subject: How do MLPs compare with RBFs?

Multilayer perceptrons (MLPs) and radial basis function (RBF) networks are the two most commonly-used types of feedforward network. They have much more in common than most of the NN literature would suggest. The only fundamental difference is the way in which hidden units combine values coming from preceding layers in the network--MLPs use inner products, while RBFs use Euclidean distance. There are also differences in the customary methods for training MLPs and RBF networks, although most methods for training MLPs can also be applied to RBF networks. Furthermore, there are crucial differences between two broad types of RBF network--ordinary RBF networks and normalized RBF networks--that are ignored in most of the NN literature. These differences have important consequences for the generalization ability of the networks, especially when the number of inputs is large.

Notation:

```
a_j     is the altitude or height of the jth hidden unit
b_j     is the bias of the jth hidden unit
f       is the fan-in of the jth hidden unit
h_j     is the activation of the jth hidden unit
s       is a common width shared by all hidden units in the layer
s_j     is the width of the jth hidden unit
w_ij    is the weight connecting the ith input to
          the jth hidden unit
w_i     is the common weight for the ith input shared by
          all hidden units in the layer
x_i     is the ith input
```

The inputs to each hidden or output unit must be combined with the weights to yield a single value called the "net input" to which the activation function is applied. There does not seem to be a standard term for the function that combines the inputs and weights; I will use the term "combination function". Thus, each hidden or output unit in a feedforward network first computes a combination function to produce the net input, and then applies an activation function to the net input yielding the activation of the unit.

A multilayer perceptron has one or more hidden layers for which the combination function is the inner product of the inputs and weights, plus a bias. The activation function is usually a `logistic` or `tanh` function. Hence the formula for the activation is typically:

```
h_j = tanh( b_j + sum[w_ij*x_i] )
```

The MLP architecture is the most popular one in practical applications. Each layer uses a linear combination function. The inputs are fully connected to the first hidden layer, each hidden layer is fully connected to the next, and the last hidden layer is fully connected to the outputs. You can also have "skip-layer" connections; direct connections from inputs to outputs are especially useful.

Consider the multidimensional space of inputs to a given hidden unit. Since an MLP uses linear combination functions, the set of all points in the space having a given value of the activation function is a hyperplane. The hyperplanes corresponding to different activation levels are parallel to each other (the hyperplanes for different units are not parallel in general). These parallel hyperplanes are the *isoactivation contours* of the hidden unit.

Radial basis function (RBF) networks usually have only one hidden layer for which the combination function is based on the Euclidean distance between the input vector and the weight vector. RBF networks do not have anything that's exactly the same as the bias term in an MLP. But some types of RBFs have a "width" associated with each hidden unit or with the the entire hidden layer; instead of adding it in the combination function like a bias, you divide the Euclidean distance by the width.

To see the similarity between RBF networks and MLPs, it is convenient to treat the combination function as the square of distance/width. Then the familiar `exp` or `softmax` activation functions produce members of the popular class of Gaussian RBF networks. It can also be useful to add another term to the combination function that determines what I will call the "altitude" of the unit. The altitude is the maximum height of the Gaussian curve above the horizontal axis. I have not seen altitudes used in the NN literature; if you know of a reference, please tell me (saswss@unx.sas.com).

The output activation function in RBF networks is usually the identity. The identity output activation function is a computational convenience in training (see Hybrid training and the curse of dimensionality) but it is possible and often desirable to use other output activation functions just as you would in an MLP.

There are many types of radial basis functions. Gaussian RBFs seem to be the most popular by far in the NN literature. In the statistical literature, thin plate splines are also used (Green and Silverman 1994). This FAQ will concentrate on Gaussian RBFs.

There are two distinct types of Gaussian RBF architectures. The first type uses the `exp` activation function, so the activation of the unit is a Gaussian "bump" as a function of the inputs. There seems to be no specific term for this type of Gaussian RBF network; I will use the term "ordinary RBF", or ORBF, network.

The second type of Gaussian RBF architecture uses the softmax activation function, so the activations of all the hidden units are normalized to sum to one. This type of network is often called a "normalized RBF", or NRBF, network. In a NRBF network, the output units should not have a bias, since the constant bias term would be linearly dependent on the constant sum of the hidden units.

While the distinction between these two types of Gaussian RBF architectures is sometimes mentioned in the NN literature, its importance has rarely been appreciated except by Tao (1993) and Werntges (1993). Shorten and Murray-Smith (1996) also compare ordinary and normalized Gaussian RBF networks.

There are several subtypes of both ORBF and NRBF architectures. Descriptions and formulas are as follows:

ORBFUN
: Ordinary radial basis function (RBF) network with unequal widths

```
h_j = exp( - s_j^-2 * sum[(w_ij-x_i)^2] )
```

ORBFEQ

Ordinary radial basis function (RBF) network with equal widths

```
h_j = exp( - s^-2 * sum[(w_ij-x_i)^2] )
```

NRBFUN

Normalized RBF network with unequal widths and heights

```
h_j = softmax(f*log(a_j) - s_j^-2 * sum[(w_ij-x_i)^2] )
```

NRBFEV

Normalized RBF network with equal volumes

```
h_j = softmax( f*log(s_j) - s_j^-2 * sum[(w_ij-x_i)^2] )
```

NRBFEH

Normalized RBF network with equal heights (and unequal widths)

```
h_j = softmax( - s_j^-2 * sum[(w_ij-x_i)^2] )
```

NRBFEW

Normalized RBF network with equal widths (and unequal heights)

```
h_j = softmax( f*log(a_j) - s^-2 * sum[(w_ij-x_i)^2] )
```

NRBFEQ

Normalized RBF network with equal widths and heights

```
h_j = softmax( - s^-2 * sum[(w_ij-x_i)^2] )
```

To illustrate various architectures, an example with two inputs and one output will be used so that the results can be shown graphically. The function being learned resembles a landscape with a Gaussian hill and a logistic plateau as shown in ftp://ftp.sas.com/pub/neural/hillplat.gif. There are 441 training cases on a regular 21-by-21 grid. The table below shows the root mean square error (RMSE) for a test data set. The test set has 1681 cases on a regular 41-by-41 grid over the same domain as the training set. If you are reading the HTML version of this document via a web browser, click on any number in the table to see a surface plot of the corresponding network output (each plot is a gif file, approximately 9K).

The MLP networks in the table have one hidden layer with a tanh activation function. All of the networks use an identity activation function for the outputs.

Hill and Plateau Data: RMSE for the Test Set

HUs	MLP	ORBFEQ	ORBFUN	NRBFEQ	NRBFEW	NRBFEV	NRBFEH	NRBFUN
2	0.218	0.247	0.247	0.230	0.230	0.230	0.230	0.230
3	0.192	0.244	0.143	0.218	0.218	0.036	0.012	0.001
4	0.174	0.216	0.096	0.193	0.193	0.036	0.007	
5	0.160	0.188	0.083	0.086	0.051	0.003		
6	0.123	0.142	0.058	0.053	0.030			
7	0.107	0.123	0.051	0.025	0.019			
8	0.093	0.105	0.043	0.020	0.008			
9	0.084	0.085	0.038	0.017				
10	0.077	0.082	0.033	0.016				
12	0.059	0.074	0.024	0.005				
15	0.042	0.060	0.019					
20	0.023	0.046	0.010					
30	0.019	0.024						
40	0.016	0.022						

50 0.010 0.014

The ORBF architectures use radial combination functions and the exp activation function. Only two of the radial combination functions are useful with ORBF architectures. For radial combination functions including an altitude, the altitude would be redundant with the hidden-to-output weights.

Radial combination functions are based on the Euclidean distance between the vector of inputs to the unit and the vector of corresponding weights. Thus, the isoactivation contours for ORBF networks are concentric hyperspheres. A variety of activation functions can be used with the radial combination function, but the exp activation function, yielding a Gaussian surface, is the most useful. Radial networks typically have only one hidden layer, but it can be useful to include a linear layer for dimensionality reduction or oblique rotation before the RBF layer.

The output of an ORBF network consists of a number of superimposed bumps, hence the output is quite bumpy unless many hidden units are used. Thus an ORBF network with only a few hidden units is incapable of fitting a wide variety of simple, smooth functions, and should rarely be used.

The NRBF architectures also use radial combination functions but the activation function is softmax, which forces the sum of the activations for the hidden layer to equal one. Thus, each output unit computes a weighted average of the hidden-to-output weights, and the output values must lie within the range of the hidden-to-output weights. Therefore, if the hidden-to-output weights are within a reasonable range (such as the range of the target values), you can be sure that the outputs will be within that same range for all possible inputs, even when the net is extrapolating. No comparably useful bound exists for the output of an ORBF network.

If you extrapolate far enough in a Gaussian ORBF network with an identity output activation function, the activation of every hidden unit will approach zero, hence the extrapolated output of the network will equal the output bias. If you extrapolate far enough in an NRBF network, one hidden unit will come to dominate the output. Hence if you want the network to extrapolate different values in a different directions, an NRBF should be used instead of an ORBF.

Radial combination functions incorporating altitudes are useful with NRBF architectures. The NRBF architectures combine some of the virtues of both the RBF and MLP architectures, as explained below. However, the isoactivation contours are considerably more complicated than for ORBF architectures.

Consider the case of an NRBF network with only two hidden units. If the hidden units have equal widths, the isoactivation contours are parallel hyperplanes; in fact, this network is equivalent to an MLP with one logistic hidden unit. If the hidden units have unequal widths, the isoactivation contours are concentric hyperspheres; such a network is almost equivalent to an ORBF network with one Gaussian hidden unit.

If there are more than two hidden units in an NRBF network, the isoactivation contours have no such simple characterization. If the RBF widths are very small, the isoactivation contours are approximately piecewise linear for RBF units with equal widths, and approximately piecewise spherical for RBF units with unequal widths. The larger the widths, the smoother the isoactivation contours where the pieces join. As Shorten and Murray-Smith (1996) point out, the activation is not necessarily a monotone function of distance from the center when unequal widths are used.

The NRBFEQ architecture is a smoothed variant of the learning vector quantization (Kohonen 1988, Ripley 1996) and counterpropagation (Hecht-Nielsen 1990), architectures. In LVQ and counterprop, the hidden units are often called "codebook vectors". LVQ amounts to nearest-neighbor classification on the codebook vectors, while counterprop is nearest-neighbor regression on the codebook vectors. The NRBFEQ architecture uses not just the single nearest neighbor, but a weighted average of near neighbors. As the width of the NRBFEQ functions approaches zero, the weights approach one for the nearest neighbor and zero for all other codebook vectors. LVQ and counterprop use ad hoc algorithms of uncertain reliability, but standard numerical optimization algorithms (not to mention backprop) can be applied with the NRBFEQ architecture.

In a NRBFEQ architecture, if each observation is taken as an RBF center, and if the weights are taken to be the target values, the outputs are simply weighted averages of the target values, and the network is identical to the well-known Nadaraya-Watson kernel regression estimator, which has been reinvented at least twice in the neural net literature (see "What is GRNN?"). A similar NRBFEQ network used for classification is equivalent to kernel discriminant analysis (see "What is PNN?").

Kernels with variable widths are also used for regression in the statistical literature. Such kernel estimators correspond to the the NRBFEV architecture, in which the kernel functions have equal volumes but different altitudes. In the neural net literature, variable-width kernels appear always to be of the NRBFEH variety, with equal altitudes but unequal volumes. The analogy with kernel regression would make the NRBFEV architecture the obvious choice, but which of the two architectures works better in practice is an open question.

Hybrid training and the curse of dimensionality

A comparison of the various architectures must separate training issues from architectural issues to avoid common sources of confusion. RBF networks are often trained by "hybrid" methods, in which the hidden weights (*centers*) are first obtained by unsupervised learning, after which the output weights are obtained by supervised learning. Unsupervised methods for choosing the centers include:

1. Distribute the centers in a regular grid over the input space.
2. Choose a random subset of the training cases to serve as centers.
3. Cluster the training cases based on the input variables, and use the mean of each cluster as a center.

Various heuristic methods are also available for choosing the RBF widths (e.g., Moody and Darken 1989; Sarle 1994b). Once the centers and widths are fixed, the output weights can be learned very efficiently, since the computation reduces to a linear or generalized linear model. The hybrid training approach can thus be much faster than the nonlinear optimization that would be required for supervised training of all of the weights in the network.

Hybrid training is not often applied to MLPs because no effective methods are known for unsupervised training of the hidden units (except when there is only one input).

Hybrid training will usually require more hidden units than supervised training. Since supervised training optimizes the locations of the centers, while hybrid training does not, supervised training will provide a better approximation to the function to be learned for a given number of hidden units. Thus, the better fit provided by supervised training will often let you use fewer hidden units for a given accuracy of

approximation than you would need with hybrid training. And if the hidden-to-output weights are learned by linear least-squares, the fact that hybrid training requires more hidden units implies that hybrid training will also require more training cases for the same accuracy of generalization (Tarassenko and Roberts 1994).

The number of hidden units required by hybrid methods becomes an increasingly serious problem as the number of inputs increases. In fact, the required number of hidden units tends to increase exponentially with the number of inputs. This drawback of hybrid methods is discussed by Minsky and Papert (1969). For example, with method (1) for RBF networks, you would need at least five elements in the grid along each dimension to detect a moderate degree of nonlinearity; so if you have `Nx` inputs, you would need at least `5^Nx` hidden units. For methods (2) and (3), the number of hidden units increases exponentially with the effective dimensionality of the input distribution. If the inputs are linearly related, the effective dimensionality is the number of nonnegligible (a deliberately vague term) eigenvalues of the covariance matrix, so the inputs must be highly correlated if the effective dimensionality is to be much less than the number of inputs.

The exponential increase in the number of hidden units required for hybrid learning is one aspect of the curse of dimensionality. The number of training cases required also increases exponentially in general. No neural network architecture--in fact no method of learning or statistical estimation--can escape the curse of dimensionality in general, hence there is no practical method of learning general functions in more than a few dimensions.

Fortunately, in many practical applications of neural networks with a large number of inputs, most of those inputs are additive, redundant, or irrelevant, and some architectures can take advantage of these properties to yield useful results. But escape from the curse of dimensionality requires fully supervised training as well as special types of data. Supervised training for RBF networks can be done by "backprop" (see "What is backprop?") or other optimization methods (see "What are conjugate gradients, Levenberg-Marquardt, etc.?"), or by subset regression "What are OLS and subset/stepwise regression?").

Additive inputs

An additive model is one in which the output is a sum of linear or nonlinear transformations of the inputs. If an additive model is appropriate, the number of weights increases linearly with the number of inputs, so high dimensionality is not a curse. Various methods of training additive models are available in the statistical literature (e.g. Hastie and Tibshirani 1990). You can also create a feedforward neural network, called a "generalized additive network" (GAN), to fit additive models (Sarle 1994a). Additive models have been proposed in the neural net literature under the name "topologically distributed encoding" (Geiger 1990).

Projection pursuit regression (PPR) provides both universal approximation and the ability to avoid the curse of dimensionality for certain common types of target functions (Friedman and Stuetzle 1981). Like MLPs, PPR computes the output as a sum of nonlinear transformations of linear combinations of the inputs. Each term in the sum is analogous to a hidden unit in an MLP. But unlike MLPs, PPR allows general, smooth nonlinear transformations rather than a specific nonlinear activation function, and allows a different transformation for each term. The nonlinear transformations in PPR are usually estimated by nonparametric regression, but you can set up a *projection pursuit network* (PPN), in which each nonlinear transformation is performed by a subnetwork. If a PPN provides an adequate fit with few terms, then the curse of

dimensionality can be avoided, and the results may even be interpretable.

If the target function can be accurately approximated by projection pursuit, then it can also be accurately approximated by an MLP with a single hidden layer. The disadvantage of the MLP is that there is little hope of interpretability. An MLP with two or more hidden layers can provide a parsimonious fit to a wider variety of target functions than can projection pursuit, but no simple characterization of these functions is known.

Redundant inputs

With proper training, all of the RBF architectures listed above, as well as MLPs, can process redundant inputs effectively. When there are redundant inputs, the training cases lie close to some (possibly nonlinear) subspace. If the same degree of redundancy applies to the test cases, the network need produce accurate outputs only near the subspace occupied by the data. Adding redundant inputs has little effect on the effective dimensionality of the data; hence the curse of dimensionality does not apply, and even hybrid methods (2) and (3) can be used. However, if the test cases do not follow the same pattern of redundancy as the training cases, generalization will require extrapolation and will rarely work well.

Irrelevant inputs

MLP architectures are good at ignoring irrelevant inputs. MLPs can also select linear subspaces of reduced dimensionality. Since the first hidden layer forms linear combinations of the inputs, it confines the networks attention to the linear subspace spanned by the weight vectors. Hence, adding irrelevant inputs to the training data does not increase the number of hidden units required, although it increases the amount of training data required.

ORBF architectures are not good at ignoring irrelevant inputs. The number of hidden units required grows exponentially with the number of inputs, regardless of how many inputs are relevant. This exponential growth is related to the fact that ORBFs have *local receptive fields*, meaning that changing the hidden-to-output weights of a given unit will affect the output of the network only in a neighborhood of the center of the hidden unit, where the size of the neighborhood is determined by the width of the hidden unit. (Of course, if the width of the unit is learned, the receptive field could grow to cover the entire training set.)

Local receptive fields are often an advantage compared to the *distributed* architecture of MLPs, since local units can adapt to local patterns in the data without having unwanted side effects in other regions. In a distributed architecture such as an MLP, adapting the network to fit a local pattern in the data can cause spurious side effects in other parts of the input space.

However, ORBF architectures often must be used with relatively small neighborhoods, so that several hidden units are required to cover the range of an input. When there are many nonredundant inputs, the hidden units must cover the entire input space, and the number of units required is essentially the same as in the hybrid case (1) where the centers are in a regular grid; hence the exponential growth in the number of hidden units with the number of inputs, regardless of whether the inputs are relevant.

You can enable an ORBF architecture to ignore irrelevant inputs by using an extra, linear hidden layer before the radial hidden layer. This type of network is sometimes called an "elliptical basis function" network. If the number of units in the linear hidden layer equals the number of inputs, the linear hidden

layer performs an oblique rotation of the input space that can suppress irrelevant directions and differentally weight relevant directions according to their importance. If you think that the presence of irrelevant inputs is highly likely, you can force a reduction of dimensionality by using fewer units in the linear hidden layer than the number of inputs.

Note that the linear and radial hidden layers must be connected in series, not in parallel, to ignore irrelevant inputs. In some applications it is useful to have linear and radial hidden layers connected in parallel, but in such cases the radial hidden layer will be sensitive to all inputs.

For even greater flexibility (at the cost of more weights to be learned), you can have a separate linear hidden layer for each RBF unit, allowing a different oblique rotation for each RBF unit.

NRBF architectures with equal widths (NRBFEW and NRBFEQ) combine the advantage of local receptive fields with the ability to ignore irrelevant inputs. The receptive field of one hidden unit extends from the center in all directions until it encounters the receptive field of another hidden unit. It is convenient to think of a "boundary" between the two receptive fields, defined as the hyperplane where the two units have equal activations, even though the effect of each unit will extend somewhat beyond the boundary. The location of the boundary depends on the heights of the hidden units. If the two units have equal heights, the boundary lies midway between the two centers. If the units have unequal heights, the boundary is farther from the higher unit.

If a hidden unit is surrounded by other hidden units, its receptive field is indeed local, curtailed by the field boundaries with other units. But if a hidden unit is not completely surrounded, its receptive field can extend infinitely in certain directions. If there are irrelevant inputs, or more generally, irrelevant directions that are linear combinations of the inputs, the centers need only be distributed in a subspace orthogonal to the irrelevant directions. In this case, the hidden units can have local receptive fields in relevant directions but infinite receptive fields in irrelevant directions.

For NRBF architectures allowing unequal widths (NRBFUN, NRBFEV, and NRBFEH), the boundaries between receptive fields are generally hyperspheres rather than hyperplanes. In order to ignore irrelevant inputs, such networks must be trained to have equal widths. Hence, if you think there is a strong possibility that some of the inputs are irrelevant, it is usually better to use an architecture with equal widths.

References:

There are few good references on RBF networks. Bishop (1995) gives one of the better surveys, but also see Tao (1993) and Werntges (1993) for the importance of normalization. Orr (1996) provides a useful introduction.

Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press.

Friedman, J.H. and Stuetzle, W. (1981), "Projection pursuit regression," J. of the American Statistical Association, 76, 817-823.

Geiger, H. (1990), "Storing and Processing Information in Connectionist Systems," in Eckmiller, R., ed., *Advanced Neural Computers*, 271-277, Amsterdam: North-Holland.

Green, P.J. and Silverman, B.W. (1994), *Nonparametric Regression and Generalized Linear Models: A roughness penalty approach*,, London: Chapman & Hall.

Hastie, T.J. and Tibshirani, R.J. (1990) *Generalized Additive Models*, London: Chapman & Hall.

Hecht-Nielsen, R. (1990), *Neurocomputing*, Reading, MA: Addison-Wesley.

Kohonen, T (1988), "Learning Vector Quantization," Neural Networks, 1 (suppl 1), 303.

Minsky, M.L. and Papert, S.A. (1969), *Perceptrons*, Cambridge, MA: MIT Press.

Moody, J. and Darken, C.J. (1989), "Fast learning in networks of locally-tuned processing units," Neural Computation, 1, 281-294.

Orr, M.J.L. (1996), "Introduction to radial basis function networks," http://www.anc.ed.ac.uk/~mjo/papers/intro.ps or http://www.anc.ed.ac.uk/~mjo/papers/intro.ps.gz

Ripley, B.D. (1996), *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.

Sarle, W.S. (1994a), "Neural Networks and Statistical Models," in SAS Institute Inc., *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc., pp 1538-1550, ftp://ftp.sas.com/pub/neural/neural1.ps.

Sarle, W.S. (1994b), "Neural Network Implementation in SAS Software," in SAS Institute Inc., *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc., pp 1551-1573, ftp://ftp.sas.com/pub/neural/neural2.ps.

Shorten, R., and Murray-Smith, R. (1996), "Side effects of normalising radial basis function networks" International Journal of Neural Systems, 7, 167-179.

Tao, K.M. (1993), "A closer look at the radial basis function (RBF) networks," *Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers* (Singh, A., ed.), vol 1, 401-405, Los Alamitos, CA: IEEE Comput. Soc. Press.

Tarassenko, L. and Roberts, S. (1994), "Supervised and unsupervised learning in radial basis function classifiers," IEE Proceedings-- Vis. Image Signal Processing, 141, 210-216.

Werntges, H.W. (1993), "Partitions of unity improve neural function approximation," Proceedings of the IEEE International Conference on Neural Networks, San Francisco, CA, vol 2, 914-918.

Subject: What are OLS and subset/stepwise regression?

If you are statistician, "OLS" means "ordinary least squares" (as opposed to weighted or generalized least squares), which is what the NN literature often calls "LMS" (least mean squares).

If you are a neural networker, "OLS" means "orthogonal least squares", which is an algorithm for forward stepwise regression proposed by Chen et al. (1991) for training RBF networks.

OLS is a variety of supervised training. But whereas backprop and other commonly-used supervised methods are forms of continuous optimization, OLS is a form of combinatorial optimization. Rather than treating the RBF centers as continuous values to be adjusted to reduce the training error, OLS starts with a large set of candidate centers and selects a subset that usually provides good training error. For small training sets, the candidates can include all of the training cases. For large training sets, it is more efficient to use a random subset of the training cases or to do a cluster analysis and use the cluster means as candidates.

Each center corresponds to a predictor variable in a linear regression model. The values of these predictor variables are computed from the RBF applied to each center. There are numerous methods for selecting a subset of predictor variables in regression (Myers 1986; Miller 1990). The ones most often used are:

- Forward selection begins with no centers in the network. At each step the center is added that most decreases the objective function.
- Backward elimination begins with all candidate centers in the network. At each step the center is removed that least increases the objective function.
- Stepwise selection begins like forward selection with no centers in the network. At each step, a center is added or removed. If there are any centers in the network, the one that contributes least to reducing the objective function is subjected to a statistical test (usually based on the F statistic) to see if it is worth retaining in the network; if the center fails the test, it is removed. If no centers are removed, then the centers that are not currently in the network are examined; the one that would contribute most to reducing the objective function is subjected to a statistical test to see if it is worth adding to the network; if the center passes the test, it is added. When all centers in the network pass the test for staying in the network, and all other centers fail the test for being added to the network, the stepwise method terminates.
- Leaps and bounds (Furnival and Wilson 1974) is an algorithm for determining the subset of centers that minimizes the objective function; this optimal subset can be found without examining all possible subsets, but the algorithm is practical only up to 30 to 50 candidate centers.

OLS is a particular algorithm for forward selection using modified Gram-Schmidt (MGS) orthogonalization. While MGS is not a bad algorithm, it is not the best algorithm for linear least-squares (Lawson and Hanson 1974). For ill-conditioned data (see ftp://ftp.sas.com/pub/neural/illcond/illcond.html), Householder and Givens methods are generally preferred, while for large, well-conditioned data sets, methods based on the normal equations require about one-third as many floating point operations and much less disk I/O than OLS. Normal equation methods based on sweeping (Goodnight 1979) or Gaussian elimination (Furnival and Wilson 1974) are especially simple to program.

While the theory of linear models is the most thoroughly developed area of statistical inference, subset selection invalidates most of the standard theory (Miller 1990; Roecker 1991; Derksen and Keselman 1992; Freedman, Pee, and Midthune 1992).

Subset selection methods usually do not generalize as well as regularization methods in linear models (Frank and Friedman 1993). Orr (1995) has proposed combining regularization with subset selection for RBF training (see also Orr 1996).

References:

Chen, S., Cowan, C.F.N., and Grant, P.M. (1991), "Orthogonal least squares learning for radial basis function networks," IEEE Transactions on Neural Networks, 2, 302-309.

Derksen, S. and Keselman, H. J. (1992) "Backward, forward and stepwise automated subset selection algorithms: Frequency of obtaining authentic and noise variables," British Journal of Mathematical and Statistical Psychology, 45, 265-282,

Frank, I.E. and Friedman, J.H. (1993) "A statistical view of some chemometrics regression tools," Technometrics, 35, 109-148.

Freedman, L.S. , Pee, D. and Midthune, D.N. (1992) "The problem of underestimating the residual error variance in forward stepwise regression", The Statistician, 41, 405-412.

Furnival, G.M. and Wilson, R.W. (1974), "Regression by Leaps and Bounds," Technometrics, 16, 499-511.

Goodnight, J.H. (1979), "A Tutorial on the SWEEP Operator," The American Statistician, 33, 149-158.

Lawson, C. L. and Hanson, R. J. (1974), *Solving Least Squares Problems*, Englewood Cliffs, NJ: Prentice-Hall, Inc. (2nd edition: 1995, Philadelphia: SIAM)

Miller, A.J. (1990), Subset Selection in Regression, Chapman & Hall.

Myers, R.H. (1986), *Classical and Modern Regression with Applications,* Boston: Duxbury Press.

Orr, M.J.L. (1995), "Regularisation in the selection of radial basis function centres," Neural Computation, 7, 606-623.

Orr, M.J.L. (1996), "Introduction to radial basis function networks,"
http://www.cns.ed.ac.uk/people/mark/intro.ps or
http://www.cns.ed.ac.uk/people/mark/intro/intro.html .

Roecker, E.B. (1991) "Prediction error and its estimation for subset-selected models," Technometrics, 33, 459-468.

- -

Subject: Should I normalize/standardize/rescale the

data?

First, some definitions. "Rescaling" a vector means to add or subtract a constant and then multiply or divide by a constant, as you would do to change the units of measurement of the data, for example, to convert a temperature from Celsius to Fahrenheit.

"Normalizing" a vector most often means dividing by a norm of the vector, for example, to make the Euclidean length of the vector equal to one. In the NN literature, "normalizing" also often refers to rescaling by the minimum and range of the vector, to make all the elements lie between 0 and 1.

"Standardizing" a vector most often means subtracting a measure of location and dividing by a measure of scale. For example, if the vector contains random values with a Gaussian distribution, you might subtract the mean and divide by the standard deviation, thereby obtaining a "standard normal" random variable with mean 0 and standard deviation 1.

However, all of the above terms are used more or less interchangeably depending on the customs within various fields. Since the FAQ maintainer is a statistician, he is going to use the term "standardize" because that is what he is accustomed to.

Now the question is, should you do any of these things to your data? The answer is, it depends.

There is a common misconception that the inputs to a multilayer perceptron must be in the interval [0,1]. There is in fact no such requirement, although there often are benefits to standardizing the inputs as discussed below. But it is better to have the input values centered around zero, so scaling the inputs to the interval [0,1] is usually a bad choice.

If your output activation function has a range of [0,1], then obviously you must ensure that the target values lie within that range. But it is generally better to choose an output activation function suited to the distribution of the targets than to force your data to conform to the output activation function. See "Why use activation functions?"

When using an output activation with a range of [0,1], some people prefer to rescale the targets to a range of [.1,.9]. I suspect that the popularity of this gimmick is due to the slowness of standard backprop. But using a target range of [.1,.9] for a classification task gives you incorrect posterior probability estimates. This gimmick is unnecessary if you use an efficient training algorithm (see "What are conjugate gradients, Levenberg-Marquardt, etc.?"), and it is also unnecessary to avoid overflow (see "How to avoid overflow in the logistic function?").

Now for some of the gory details: note that the training data form a matrix. Let's set up this matrix so that each case forms a row, and the inputs and target variables form columns. You could conceivably standardize the rows or the columns or both or various other things, and these different ways of choosing vectors to standardize will have quite different effects on training.

Standardizing either input or target variables tends to make the training process better behaved by improving the numerical condition (see ftp://ftp.sas.com/pub/neural/illcond/illcond.html) of the optimization problem and ensuring that various default values involved in initialization and termination are

appropriate. Standardizing targets can also affect the objective function.

Standardization of cases should be approached with caution because it discards information. If that information is irrelevant, then standardizing cases can be quite helpful. If that information is important, then standardizing cases can be disastrous.

Should I standardize the input variables (column vectors)?

That depends primarily on how the network combines input variables to compute the net input to the next (hidden or output) layer. If the input variables are combined via a distance function (such as Euclidean distance) in an RBF network, standardizing inputs can be crucial. The contribution of an input will depend heavily on its variability relative to other inputs. If one input has a range of 0 to 1, while another input has a range of 0 to 1,000,000, then the contribution of the first input to the distance will be swamped by the second input. So it is essential to rescale the inputs so that their variability reflects their importance, or at least is not in inverse relation to their importance. For lack of better prior information, it is common to standardize each input to the same range or the same standard deviation. If you know that some inputs are more important than others, it may help to scale the inputs such that the more important ones have larger variances and/or ranges.

If the input variables are combined linearly, as in an MLP, then it is rarely strictly necessary to standardize the inputs, at least in theory. The reason is that any rescaling of an input vector can be effectively undone by changing the corresponding weights and biases, leaving you with the exact same outputs as you had before. However, there are a variety of practical reasons why standardizing the inputs can make training faster and reduce the chances of getting stuck in local optima. Also, weight decay and Bayesian estimation can be done more conveniently with standardized inputs.

The main emphasis in the NN literature on initial values has been on the avoidance of saturation, hence the desire to use small random values. How small these random values should be depends on the scale of the inputs as well as the number of inputs and their correlations. Standardizing inputs removes the problem of scale dependence of the initial weights.

But standardizing input variables can have far more important effects on initialization of the weights than simply avoiding saturation. Assume we have an MLP with one hidden layer applied to a classification problem and are therefore interested in the hyperplanes defined by each hidden unit. Each hyperplane is the locus of points where the net-input to the hidden unit is zero and is thus the classification boundary generated by that hidden unit considered in isolation. The connection weights from the inputs to a hidden unit determine the orientation of the hyperplane. The bias determines the distance of the hyperplane from the origin. If the bias terms are all small random numbers, then all the hyperplanes will pass close to the origin. Hence, if the data are not centered at the origin, the hyperplane may fail to pass through the data cloud. If all the inputs have a small coefficient of variation, it is quite possible that all the initial hyperplanes will miss the data entirely. With such a poor initialization, local minima are very likely to occur. It is therefore important to center the inputs to get good random initializations. In particular, scaling the inputs to [-1,1] will work better than [0,1], although any scaling that sets to zero the mean or median or other measure of central tendency is likely to be as good, and robust estimators of location and scale (Iglewicz, 1983) will be even better for input variables with extreme outliers.

For example, consider an MLP with two inputs (X and Y) and 100 hidden units. The graph at lines-

10to10.gif shows the initial hyperplanes (which are lines, of course, in two dimensions) using initial weights and biases drawn from a normal distribution with a mean of zero and a standard deviation of one. The inputs X and Y are both shown over the interval [-10,10]. As you can see, most of the hyperplanes pass near the origin, and relatively few hyperplanes go near the corners. Furthermore, most of the hyperplanes that go near any given corner are at roughly the same angle. That is, the hyperplanes that pass near the upper right corner go predominantly in the direction from lower left to upper right. Hardly any hyperplanes near this corner go from upper left to lower right. If the network needs to learn such a hyperplane, it may take many random initializations before training finds a local optimum with such a hyperplane.

Now suppose the input data are distributed over a range of [-2,2] or [-1,1]. Graphs showing these regions can be seen at lines-2to2.gif and lines-1to1.gif. The initial hyperplanes cover these regions rather thoroughly, and few initial hyperplanes miss these regions entirely. It will be easy to learn a hyperplane passing through any part of these regions at any angle.

But if the input data are distributed over a range of [0,1] as shown at lines0to1.gif, the initial hyperplanes are concentrated in the lower left corner, with fewer passing near the upper right corner. Furthermore, many initial hyperplanes miss this region entirely, and since these hyperplanes will be close to saturation over most of the input space, learning will be slow. For an example using the 5-bit parity problem, see "Why not code binary inputs as 0 and 1?"

If the input data are distributed over a range of [1,2] as shown at lines1to2.gif, the situation is even worse. If the input data are distributed over a range of [9,10] as shown at lines9to10.gif, very few of the initial hyperplanes pass through the region at all, and it will be difficult to learn any but the simplest classifications or functions.

It is also bad to have the data confined to a very narraw range such as [-0.1,0.1], as shown at lines-0.1to0.1.gif, since most of the initial hyperplanes will miss such a small region.

Thus it is easy to see that you will get better initializations if the data are centered near zero and if most of the data are distributed over an interval of roughly [-1,1] or [-2,2]. If you are firmly opposed to the idea of standardizing the input variables, you can compensate by transforming the initial weights, but this is much more complicated than standardizing the input variables.

Standardizing input variables has different effects on different training algorithms for MLPs. For example:

- Steepest descent is very sensitive to scaling. The more ill-conditioned the Hessian is, the slower the convergence. Hence, scaling is an important consideration for gradient descent methods such as standard backprop.
- Quasi-Newton and conjugate gradient methods begin with a steepest descent step and therefore are scale sensitive. However, they accumulate second-order information as training proceeds and hence are less scale sensitive than pure gradient descent.
- Newton-Raphson and Gauss-Newton, if implemented correctly, are theoretically invariant under scale changes as long as none of the scaling is so extreme as to produce underflow or overflow.
- Levenberg-Marquardt is scale invariant as long as no ridging is required. There are several different ways to implement ridging; some are scale invariant and some are not. Performance under bad scaling will depend on details of the implementation.

For more information on ill-conditioning, see ftp://ftp.sas.com/pub/neural/illcond/illcond.html

Two of the most useful ways to standardize inputs are:

- Mean 0 and standard deviation 1
- Midrange 0 and range 2 (i.e., minimum -1 and maximum 1)

Note that statistics such as the mean and standard deviation are computed from the training data, not from the validation or test data. The validation and test data must be standardized using the statistics computed from the training data.

Formulas are as follows:

Notation:

X_i = value of the raw input variable X for the ith training case

S_i = standardized value corresponding to X_i

N = number of training cases

Standardize X_i to mean 0 and standard deviation 1:

$$\text{mean} = \frac{\sum_i X_i}{N}$$

$$\text{std} = \sqrt{\frac{\sum_i (X_i - \text{mean})^2}{N - 1}}$$

$$S_i = \frac{X_i - \text{mean}}{\text{std}}$$

Standardize X_i to midrange 0 and range 2:

$$S_i = \frac{X_i - \frac{\max X + \min X}{2}}{\frac{\max X - \min X}{2}}$$

$$\text{midrange} = \frac{i_i + i_i}{2}$$

$$\text{range} = \max_i X_i - \min_i X_i$$

$$S_i = \frac{X_i - \text{midrange}}{\text{range} / 2}$$

Various other pairs of location and scale estimators can be used besides the mean and standard deviation, or midrange and range. Robust estimates of location and scale are desirable if the inputs contain outliers. For example, see:

> Iglewicz, B. (1983), "Robust scale estimators and confidence intervals for location", in Hoaglin, D.C., Mosteller, M. and Tukey, J.W., eds., *Understanding Robust and Exploratory Data Analysis*, NY: Wiley.

Should I standardize the target variables (column vectors)?

Standardizing target variables is typically more a convenience for getting good initial weights than a necessity. However, if you have two or more target variables and your error function is scale-sensitive like the usual least (mean) squares error function, then the variability of each target relative to the others can effect how well the net learns that target. If one target has a range of 0 to 1, while another target has a range of 0 to 1,000,000, the net will expend most of its effort learning the second target to the possible exclusion of the first. So it is essential to rescale the targets so that their variability reflects their importance, or at least is not in inverse relation to their importance. If the targets are of equal importance, they should typically be standardized to the same range or the same standard deviation.

The scaling of the targets does not affect their importance in training if you use maximum likelihood estimation and estimate a separate scale parameter (such as a standard deviation) for each target variable. In this case, the importance of each target is inversely related to its estimated scale parameter. In other words, noisier targets will be given less importance.

For weight decay and Bayesian estimation, the scaling of the targets affects the decay values and prior distributions. Hence it is usually most convenient to work with standardized targets.

If you are standardizing targets to equalize their importance, then you should probably standardize to mean 0 and standard deviation 1, or use related robust estimators, as discussed under Should I standardize the input variables (column vectors)? If you are standardizing targets to force the values into the range of the output activation function, it is important to use lower and upper bounds for the values, rather than the minimum and maximum values in the training set. For example, if the output activation function has range [-1,1], you can use the following formulas:

$$Y_i = \text{value of the raw target variable } Y \text{ for the } i\text{th training case}$$

$$Z_i = \text{standardized value corresponding to } Y_i$$

$$\text{midrange} = \frac{\text{upper bound of } Y \; + \; \text{lower bound of } Y}{2}$$

$$\text{range} = \text{upper bound of } Y \; - \; \text{lower bound of } Y$$

$$Z_i = \frac{Y_i - \text{midrange}}{\text{range} / 2}$$

For a range of [0,1], you can use the following formula:

$$Z_i = \frac{Y_i - \text{lower bound of } Y}{\text{upper bound of } Y \; - \; \text{lower bound of } Y}$$

And of course, you apply the inverse of the standardization formula to the network outputs to restore them to the scale of the original target values.

If the target variable does not have known upper and lower bounds, it is not advisable to use an output activation function with a bounded range. You can use an identity output activation function or other unbounded output activation function instead; see Why use activation functions?

Should I standardize the variables (column vectors) for unsupervised learning?

The most commonly used methods of unsupervised learning, including various kinds of vector quantization, Kohonen networks, Hebbian learning, etc., depend on Euclidean distances or scalar-product similarity measures. The considerations are therefore the same as for standardizing inputs in RBF networks--see Should I standardize the input variables (column vectors)? above. In particular, if one input has a large variance and another a small variance, the latter will have little or no influence on the results.

If you are using unsupervised competitive learning to try to discover natural clusters in the data, rather than for data compression, simply standardizing the variables may be inadequate. For more sophisticated methods of preprocessing, see:

Art, D., Gnanadesikan, R., and Kettenring, R. (1982), "Data-based Metrics for Cluster Analysis,"

Utilitas Mathematica, 21A, 75-99.

Jannsen, P., Marron, J.S., Veraverbeke, N, and Sarle, W.S. (1995), "Scale measures for bandwidth selection", J. of Nonparametric Statistics, 5, 359-380.

Better yet for finding natural clusters, try mixture models or nonparametric density estimation. For example::

Girman, C.J. (1994), "Cluster Analysis and Classification Tree Methodology as an Aid to Improve Understanding of Benign Prostatic Hyperplasia," Ph.D. thesis, Chapel Hill, NC: Department of Biostatistics, University of North Carolina.

McLachlan, G.J. and Basford, K.E. (1988), Mixture Models, New York: Marcel Dekker, Inc.

SAS Institute Inc. (1993), SAS/STAT Software: The MODECLUS Procedure, SAS Technical Report P-256, Cary, NC: SAS Institute Inc.

Titterington, D.M., Smith, A.F.M., and Makov, U.E. (1985), Statistical Analysis of Finite Mixture Distributions, New York: John Wiley & Sons, Inc.

Wong, M.A. and Lane, T. (1983), "A kth Nearest Neighbor Clustering Procedure," Journal of the Royal Statistical Society, Series B, 45, 362-368.

Should I standardize the input cases (row vectors)?

Whereas standardizing variables is usually beneficial, the effect of standardizing cases (row vectors) depends on the particular data. Cases are typically standardized only across the input variables, since including the target variable(s) in the standardization would make prediction impossible.

There are some kinds of networks, such as simple Kohonen nets, where it is necessary to standardize the input cases to a common Euclidean length; this is a side effect of the use of the inner product as a similarity measure. If the network is modified to operate on Euclidean distances instead of inner products, it is no longer necessary to standardize the input cases.

Standardization of cases should be approached with caution because it discards information. If that information is irrelevant, then standardizing cases can be quite helpful. If that information is important, then standardizing cases can be disastrous. Issues regarding the standardization of cases must be carefully evaluated in every application. There are no rules of thumb that apply to all applications.

You may want to standardize each case if there is extraneous variability between cases. Consider the common situation in which each input variable represents a pixel in an image. If the images vary in exposure, and exposure is irrelevant to the target values, then it would usually help to subtract the mean of each case to equate the exposures of different cases. If the images vary in contrast, and contrast is irrelevant to the target values, then it would usually help to divide each case by its standard deviation to equate the contrasts of different cases. Given sufficient data, a NN could learn to ignore exposure and contrast. However, training will be easier and generalization better if you can remove the extraneous exposure and contrast information before training the network.

As another example, suppose you want to classify plant specimens according to species but the specimens are at different stages of growth. You have measurements such as stem length, leaf length, and leaf width. However, the over-all size of the specimen is determined by age or growing conditions, not by species. Given sufficient data, a NN could learn to ignore the size of the specimens and classify them by shape instead. However, training will be easier and generalization better if you can remove the extraneous size information before training the network. Size in the plant example corresponds to exposure in the image example.

If the input data are measured on an interval scale (for information on scales of measurement, see "Measurement theory: Frequently asked questions", at ftp://ftp.sas.com/pub/neural/measurement.html) you can control for size by subtracting a measure of the over-all size of each case from each datum. For example, if no other direct measure of size is available, you could subtract the mean of each row of the input matrix, producing a row-centered input matrix.

If the data are measured on a ratio scale, you can control for size by dividing each datum by a measure of over-all size. It is common to divide by the sum or by the arithmetic mean. For positive ratio data, however, the geometric mean is often a more natural measure of size than the arithmetic mean. It may also be more meaningful to analyze the logarithms of positive ratio-scaled data, in which case you can subtract the arithmetic mean after taking logarithms. You must also consider the dimensions of measurement. For example, if you have measures of both length and weight, you may need to cube the measures of length or take the cube root of the weights.

In NN aplications with ratio-level data, it is common to divide by the Euclidean length of each row. If the data are positive, dividing by the Euclidean length has properties similar to dividing by the sum or arithmetic mean, since the former projects the data points onto the surface of a hypersphere while the latter projects the points onto a hyperplane. If the dimensionality is not too high, the resulting configurations of points on the hypersphere and hyperplane are usually quite similar. If the data contain negative values, then the hypersphere and hyperplane can diverge widely.

Subject: Should I nonlinearly transform the data?

Most importantly, nonlinear transformations of the targets are important with noisy data, via their effect on the error function. Many commonly used error functions are functions solely of the difference abs(target-output). Nonlinear transformations (unlike linear transformations) change the relative sizes of these differences. With most error functions, the net will expend more effort, so to speak, trying to learn target values for which abs(target-output) is large.

For example, suppose you are trying to predict the price of a stock. If the price of the stock is 10 (in whatever currency unit) and the output of the net is 5 or 15, yielding a difference of 5, that is a huge error. If the price of the stock is 1000 and the output of the net is 995 or 1005, yielding the same difference of 5, that is a tiny error. You don't want the net to treat those two differences as equally important. By taking logarithms, you are effectively measuring errors in terms of ratios rather than differences, since a difference between two logs corresponds to the ratio of the original values. This has approximately the same effect as looking at percentage differences, abs(target-output)/target or abs(target-output)/output, rather than simple

differences.

Less importantly, smooth functions are usually easier to learn than rough functions. Generalization is also usually better for smooth functions. So nonlinear transformations (of either inputs or targets) that make the input-output function smoother are usually beneficial. For classification problems, you want the class boundaries to be smooth. When there are only a few inputs, it is often possible to transform the data to a linear relationship, in which case you can use a linear model instead of a more complex neural net, and many things (such as estimating generalization error and error bars) will become much simpler. A variety of NN architectures (RBF networks, B-spline networks, etc.) amount to using many nonlinear transformations, possibly involving multiple variables simultaneously, to try to make the input-output function approximately linear (Ripley 1996, chapter 4). There are particular applications, such as signal and image processing, in which very elaborate transformations are useful (Masters 1994).

It is usually advisable to choose an error function appropriate for the distribution of noise in your target variables (McCullagh and Nelder 1989). But if your software does not provide a sufficient variety of error functions, then you may need to transform the target so that the noise distribution conforms to whatever error function you are using. For example, if you have to use least-(mean-)squares training, you will get the best results if the noise distribution is approximately Gaussian with constant variance, since least-(mean-)squares is maximum likelihood in that case. Heavy-tailed distributions (those in which extreme values occur more often than in a Gaussian distribution, often as indicated by high kurtosis) are especially of concern, due to the loss of statistical efficiency of least-(mean-)square estimates (Huber 1981). Note that what is important is the distribution of the noise, not the distribution of the target values.

The distribution of inputs may suggest transformations, but this is by far the least important consideration among those listed here. If an input is strongly skewed, a logarithmic, square root, or other power (between -1 and 1) transformation may be worth trying. If an input has high kurtosis but low skewness, an arctan transform can reduce the influence of extreme values:

$$\arctan\left(c \, \frac{\text{input} - \text{mean}}{\text{stand. dev.}} \right)$$

where `c` is a constant that controls how far the extreme values are brought in towards the mean. Arctan usually works better than tanh, which squashes the extreme values too much. Using robust estimates of location and scale (Iglewicz 1983) instead of the mean and standard deviation will work even better for pathological distributions.

References:

Atkinson, A.C. (1985) *Plots, Transformations and Regression*, Oxford: Clarendon Press.

Carrol, R.J. and Ruppert, D. (1988) *Transformation and Weighting in Regression*, London: Chapman and Hall.

Huber, P.J. (1981), *Robust Statistics*, NY: Wiley.

Iglewicz, B. (1983), "Robust scale estimators and confidence intervals for location", in Hoaglin,

D.C., Mosteller, M. and Tukey, J.W., eds., *Understanding Robust and Exploratory Data Analysis*, NY: Wiley.

McCullagh, P. and Nelder, J.A. (1989) *Generalized Linear Models*, 2nd ed., London: Chapman and Hall.

Masters, T. (1994), *Signal and Image Processing with Neural Networks: A C++ Sourcebook*, NY: Wiley.

Ripley, B.D. (1996), *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.

Subject: How to measure importance of inputs?

The answer to this question is rather long and so is not included directly in the posted FAQ. See ftp://ftp.sas.com/pub/neural/importance.html.

Also see Pierre van de Laar's bibliography at ftp://ftp.mbfys.kun.nl/snn/pub/pierre/connectionists.html, but don't believe everything you read in those papers.

Subject: What is ART?

ART stands for "Adaptive Resonance Theory", invented by Stephen Grossberg in 1976. ART encompasses a wide variety of neural networks based explicitly on neurophysiology. ART networks are defined algorithmically in terms of detailed differential equations intended as plausible models of biological neurons. In practice, ART networks are implemented using analytical solutions or approximations to these differential equations.

ART comes in several flavors, both supervised and unsupervised. As discussed by Moore (1988), the unsupervised ARTs are basically similar to many iterative clustering algorithms in which each case is processed by:

1. finding the "nearest" cluster seed (AKA prototype or template) to that case
2. updating that cluster seed to be "closer" to the case

where "nearest" and "closer" can be defined in hundreds of different ways. In ART, the framework is modified slightly by introducing the concept of "resonance" so that each case is processed by:

1. finding the "nearest" cluster seed that "resonates" with the case
2. updating that cluster seed to be "closer" to the case

"Resonance" is just a matter of being within a certain threshold of a second similarity measure. A crucial feature of ART is that if no seed resonates with the case, a new cluster is created as in Hartigan's (1975) leader algorithm. This feature is said to solve the "stability-plasticity dilemma" (See "Sequential Learning, Catastrophic Interference, and the Stability-Plasticity Dilemma"

ART has its own jargon. For example, data are called an "arbitrary sequence of input patterns". The current training case is stored in "short term memory" and cluster seeds are "long term memory". A cluster is a "maximally compressed pattern recognition code". The two stages of finding the nearest seed to the input are performed by an "Attentional Subsystem" and an "Orienting Subsystem", the latter of which performs "hypothesis testing", which simply refers to the comparison with the vigilance threshhold, not to hypothesis testing in the statistical sense. "Stable learning" means that the algorithm converges. So the often-repeated claim that ART algorithms are "capable of rapid stable learning of recognition codes in response to arbitrary sequences of input patterns" merely means that ART algorithms are clustering algorithms that converge; it does *not* mean, as one might naively assume, that the clusters are insensitive to the sequence in which the training patterns are presented--quite the opposite is true.

There are various supervised ART algorithms that are named with the suffix "MAP", as in Fuzzy ARTMAP. These algorithms cluster both the inputs and targets and associate the two sets of clusters. The effect is somewhat similar to counterpropagation. The main disadvantage of most ARTMAP algorithms is that they have no mechanism to avoid overfitting and hence should not be used with noisy data (Williamson, 1995).

For more information, see the ART FAQ at http://www.wi.leidenuniv.nl/art/ and the "ART Headquarters" at Boston University, http://cns-web.bu.edu/. For a statistical view of ART, see Sarle (1995).

For C software, see the ART Gallery at http://cns-web.bu.edu/pub/laliden/WWW/nnet.frame.html

References:

Carpenter, G.A., Grossberg, S. (1996), "Learning, Categorization, Rule Formation, and Prediction by Fuzzy Neural Networks," in Chen, C.H., ed. (1996) *Fuzzy Logic and Neural Network Handbook*, NY: McGraw-Hill, pp. 1.3-1.45.

Hartigan, J.A. (1975), *Clustering Algorithms*, NY: Wiley.

Kasuba, T. (1993), "Simplified Fuzzy ARTMAP," AI Expert, 8, 18-25.

Moore, B. (1988), "ART 1 and Pattern Clustering," in Touretzky, D., Hinton, G. and Sejnowski, T., eds., *Proceedings of the 1988 Connectionist Models Summer School*, 174-185, San Mateo, CA: Morgan Kaufmann.

Sarle, W.S. (1995), "Why Statisticians Should Not FART," ftp://ftp.sas.com/pub/neural/fart.txt

Williamson, J.R. (1995), "Gaussian ARTMAP: A Neural Network for Fast Incremental Learning of Noisy Multidimensional Maps," Technical Report CAS/CNS-95-003, Boston University, Center of Adaptive Systems and Department of Cognitive and Neural Systems.

Subject: What is PNN?

PNN or "Probabilistic Neural Network" is Donald Specht's term for kernel discriminant analysis. (Kernels are also called "Parzen windows".) You can think of it as a normalized RBF network in which there is a hidden unit centered at every training case. These RBF units are called "kernels" and are usually probability density functions such as the Gaussian. The hidden-to-output weights are usually 1 or 0; for each hidden unit, a weight of 1 is used for the connection going to the output that the case belongs to, while all other connections are given weights of 0. Alternatively, you can adjust these weights for the prior probabilities of each class. So the only weights that need to be learned are the widths of the RBF units. These widths (often a single width is used) are called "smoothing parameters" or "bandwidths" and are usually chosen by cross-validation or by more esoteric methods that are not well-known in the neural net literature; gradient descent is *not* used.

Specht's claim that a PNN trains 100,000 times faster than backprop is at best misleading. While they are not iterative in the same sense as backprop, kernel methods require that you estimate the kernel bandwidth, and this requires accessing the data many times. Furthermore, computing a single output value with kernel methods requires either accessing the entire training data or clever programming, and either way is much slower than computing an output with a feedforward net. And there are a variety of methods for training feedforward nets that are much faster than standard backprop. So depending on what you are doing and how you do it, PNN may be either faster or slower than a feedforward net.

PNN is a universal approximator for smooth class-conditional densities, so it should be able to solve any smooth classification problem given enough data. The main drawback of PNN is that, like kernel methods in general, it suffers badly from the curse of dimensionality. PNN cannot ignore irrelevant inputs without major modifications to the basic algorithm. So PNN is not likely to be the top choice if you have more than 5 or 6 nonredundant inputs. For modified algorithms that deal with irrelevant inputs, see Masters (1995) and Lowe (1995).

But if all your inputs are relevant, PNN has the very useful ability to tell you whether a test case is similar (i.e. has a high density) to any of the training data; if not, you are extrapolating and should view the output classification with skepticism. This ability is of limited use when you have irrelevant inputs, since the similarity is measured with respect to all of the inputs, not just the relevant ones.

References:

Hand, D.J. (1982) *Kernel Discriminant Analysis*, Research Studies Press.

Lowe, D.G. (1995), "Similarity metric learning for a variable-kernel classifier," Neural Computation, 7, 72-85, http://www.cs.ubc.ca/spider/lowe/pubs.html

McLachlan, G.J. (1992) *Discriminant Analysis and Statistical Pattern Recognition*, Wiley.

Masters, T. (1993). *Practical Neural Network Recipes in C++*, San Diego: Academic Press.

Masters, T. (1995) *Advanced Algorithms for Neural Networks: A C++ Sourcebook*, NY: John Wiley and Sons, ISBN 0-471-10588-0

Michie, D., Spiegelhalter, D.J. and Taylor, C.C. (1994) *Machine Learning, Neural and Statistical Classification*, Ellis Horwood; this book is out of print but available online at http://www.amsta.leeds.ac.uk/~charles/statlog/

Scott, D.W. (1992) *Multivariate Density Estimation*, Wiley.

Specht, D.F. (1990) "Probabilistic neural networks," Neural Networks, 3, 110-118.

Subject: What is GRNN?

GRNN or "General Regression Neural Network" is Donald Specht's term for Nadaraya-Watson kernel regression, also reinvented in the NN literature by Schi\oler and Hartmann. (Kernels are also called "Parzen windows".) You can think of it as a normalized RBF network in which there is a hidden unit centered at every training case. These RBF units are called "kernels" and are usually probability density functions such as the Gaussian. The hidden-to-output weights are just the target values, so the output is simply a weighted average of the target values of training cases close to the given input case. The only weights that need to be learned are the widths of the RBF units. These widths (often a single width is used) are called "smoothing parameters" or "bandwidths" and are usually chosen by cross-validation or by more esoteric methods that are not well-known in the neural net literature; gradient descent is *not* used.

GRNN is a universal approximator for smooth functions, so it should be able to solve any smooth function-approximation problem given enough data. The main drawback of GRNN is that, like kernel methods in general, it suffers badly from the curse of dimensionality. GRNN cannot ignore irrelevant inputs without major modifications to the basic algorithm. So GRNN is not likely to be the top choice if you have more than 5 or 6 nonredundant inputs.

References:

Caudill, M. (1993), "GRNN and Bear It," AI Expert, Vol. 8, No. 5 (May), 28-33.

Haerdle, W. (1990), *Applied Nonparametric Regression*, Cambridge Univ. Press.

Masters, T. (1995) *Advanced Algorithms for Neural Networks: A C++ Sourcebook*, NY: John Wiley and Sons, ISBN 0-471-10588-0

Nadaraya, E.A. (1964) "On estimating regression", Theory Probab. Applic. 10, 186-90.

Schi\oler, H. and Hartmann, U. (1992) "Mapping Neural Network Derived from the Parzen Window Estimator", Neural Networks, 5, 903-909.

Specht, D.F. (1968) "A practical technique for estimating general regression surfaces," Lockheed

report LMSC 6-79-68-6, Defense Technical Information Center AD-672505.

Specht, D.F. (1991) "A Generalized Regression Neural Network", IEEE Transactions on Neural Networks, 2, Nov. 1991, 568-576.

Wand, M.P., and Jones, M.C. (1995), *Kernel Smoothing*, London: Chapman & Hall.

Watson, G.S. (1964) "Smooth regression analysis", Sankhy\=a, Series A, 26, 359-72.

Subject: What does unsupervised learning learn?

Unsupervised learning allegedly involves no target values. In fact, for most varieties of unsupervised learning, the targets are the same as the inputs (Sarle 1994). In other words, unsupervised learning usually performs the same task as an auto-associative network, compressing the information from the inputs (Deco and Obradovic 1996). Unsupervised learning is very useful for data visualization (Ripley 1996), although the NN literature generally ignores this application.

Unsupervised competitive learning is used in a wide variety of fields under a wide variety of names, the most common of which is "cluster analysis" (see the Classification Society of North America's web site for more information on cluster analysis, including software, at http://www.pitt.edu/~csna/.) The main form of competitive learning in the NN literature is vector quantization (VQ, also called a "Kohonen network", although Kohonen invented several other types of networks as well--see "How many kinds of Kohonen networks exist?" which provides more reference on VQ). Kosko (1992) and Hecht-Nielsen (1990) review neural approaches to VQ, while the textbook by Gersho and Gray (1992) covers the area from the perspective of signal processing. In statistics, VQ has been called "principal point analysis" (Flury, 1990, 1993; Tarpey et al., 1994) but is more frequently encountered in the guise of k-means clustering. In VQ, each of the competitive units corresponds to a cluster center (also called a codebook vector), and the error function is the sum of squared Euclidean distances between each training case and the nearest center. Often, each training case is normalized to a Euclidean length of one, which allows distances to be simplified to inner products. The more general error function based on distances is the same error function used in k-means clustering, one of the most common types of cluster analysis (Max 1960; MacQueen 1967; Anderberg 1973; Hartigan 1975; Hartigan and Wong 1979; Linde, Buzo, and Gray 1980; Lloyd 1982). The k-means model is an approximation to the normal mixture model (McLachlan and Basford 1988) assuming that the mixture components (clusters) all have spherical covariance matrices and equal sampling probabilities. Normal mixtures have found a variety of uses in neural networks (e.g., Bishop 1995). Balakrishnan, Cooper, Jacob, and Lewis (1994) found that k-means algorithms used as normal-mixture approximations recover cluster membership more accurately than Kohonen algorithms.

Hebbian learning is the other most common variety of unsupervised learning (Hertz, Krogh, and Palmer 1991). Hebbian learning minimizes the same error function as an auto-associative network with a linear hidden layer, trained by least squares, and is therefore a form of dimensionality reduction. This error function is equivalent to the sum of squared distances between each training case and a linear subspace of the input space (with distances measured perpendicularly), and is minimized by the leading principal components (Pearson 1901; Hotelling 1933; Rao 1964; Joliffe 1986; Jackson 1991; Diamantaras and Kung

1996). There are variations of Hebbian learning that explicitly produce the principal components (Hertz, Krogh, and Palmer 1991; Karhunen 1994; Deco and Obradovic 1996; Diamantaras and Kung 1996).

Perhaps the most novel form of unsupervised learning in the NN literature is Kohonen's self-organizing (feature) map (SOM, Kohonen 1995). SOMs combine competitive learning with dimensionality reduction by smoothing the clusters with respect to an a priori grid (see "How many kinds of Kohonen networks exist?") for more explanation). But Kohonen's original SOM algorithm does not optimize an "energy" function (Erwin et al., 1992; Kohonen 1995, pp. 126, 237). The SOM algorithm involves a trade-off between the accuracy of the quantization and the smoothness of the topological mapping, but there is no explicit combination of these two properties into an energy function. Hence Kohonen's SOM is not simply an information-compression method like most other unsupervised learning networks. Neither does Kohonen's SOM have a clear interpretation as a density estimation method. Convergence of Kohonen's SOM algorithm is allegedly demonstrated by Yin and Allinson (1995), but their "proof" assumes the neighborhood size becomes zero, in which case the algorithm reduces to VQ and no longer has topological ordering properties (Kohonen 1995, p. 111). The best explanation of what a Kohonen SOM learns seems to be provided by the connection between SOMs and principal curves and surfaces explained by Mulier and Cherkassky (1995) and Ritter, Martinetz, and Schulten (1992). For further explanation, see "How many kinds of Kohonen networks exist?"

A variety of energy functions for SOMs have been proposed (e.g., Luttrell, 1994), some of which show a connection between SOMs and multidimensional scaling (Goodhill and Sejnowski 1997). There are also other approaches to SOMs that have clearer theoretical justification using mixture models with Bayesian priors or constraints (Utsugi, 1996, 1997; Bishop, Svensén, and Williams, 1997).

For additional references on cluster analysis, see ftp://ftp.sas.com/pub/neural/clus_bib.txt.

References:

Anderberg, M.R. (1973), *Cluster Analysis for Applications*, New York: Academic Press, Inc.

Balakrishnan, P.V., Cooper, M.C., Jacob, V.S., and Lewis, P.A. (1994) "A study of the classification capabilities of neural networks using unsupervised learning: A comparison with k-means clustering", Psychometrika, 59, 509-525.

Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press.

Bishop, C.M., Svensén, M., and Williams, C.K.I (1997), "GTM: A principled alternative to the self-organizing map," in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) *Advances in Neural Information Processing Systems 9*, Cambrideg, MA: The MIT Press, pp. 354-360. Also see http://www.ncrg.aston.ac.uk/GTM/

Deco, G. and Obradovic, D. (1996), *An Information-Theoretic Approach to Neural Computing*, NY: Springer-Verlag.

Diamantaras, K.I., and Kung, S.Y. (1996) *Principal Component Neural Networks: Theory and Applications*, NY: Wiley.

Erwin, E., Obermayer, K., and Schulten, K. (1992), "Self-organizing maps: Ordering, convergence properties and energy functions," Biological Cybernetics, 67, 47-55.

Flury, B. (1990), "Principal points," Biometrika, 77, 33-41.

Flury, B. (1993), "Estimation of principal points," Applied Statistics, 42, 139-151.

Gersho, A. and Gray, R.M. (1992), *Vector Quantization and Signal Compression*, Boston: Kluwer Academic Publishers.

Goodhill, G.J., and Sejnowski, T.J. (1997), "A unifying objective function for topographic mappings," Neural Computation, 9, 1291-1303.

Hartigan, J.A. (1975), *Clustering Algorithms*, NY: Wiley.

Hartigan, J.A., and Wong, M.A. (1979), "Algorithm AS136: A k-means clustering algorithm," Applied Statistics, 28-100-108.

Hecht-Nielsen, R. (1990), *Neurocomputing*, Reading, MA: Addison-Wesley.

Hertz, J., Krogh, A., and Palmer, R. (1991). *Introduction to the Theory of Neural Computation.* Addison-Wesley: Redwood City, California.

Hotelling, H. (1933), "Analysis of a Complex of Statistical Variables into Principal Components," Journal of Educational Psychology, 24, 417-441, 498-520.

Ismail, M.A., and Kamel, M.S. (1989), "Multidimensional data clustering utilizing hybrid search strategies," Pattern Recognition, 22, 75-89.

Jackson, J.E. (1991), *A User's Guide to Principal Components*, NY: Wiley.

Jolliffe, I.T. (1986), *Principal Component Analysis*, Springer-Verlag.

Karhunen, J. (1994), "Stability of Oja's PCA subspace rule," Neural Computation, 6, 739-747.

Kohonen, T. (1995/1997), *Self-Organizing Maps*, Berlin: Springer-Verlag.

Kosko, B.(1992), *Neural Networks and Fuzzy Systems*, Englewood Cliffs, N.J.: Prentice-Hall.

Linde, Y., Buzo, A., and Gray, R. (1980), "An algorithm for vector quantizer design," IEEE Transactions on Communications, 28, 84-95.

Lloyd, S. (1982), "Least squares quantization in PCM," IEEE Transactions on Information Theory, 28, 129-137.

Luttrell, S.P. (1994), "A Bayesian analysis of self-organizing maps," Neural Computation, 6, 767-

794.

McLachlan, G.J. and Basford, K.E. (1988), *Mixture Models*, NY: Marcel Dekker, Inc.

MacQueen, J.B. (1967), "Some Methods for Classification and Analysis of Multivariate Observations,"Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1, 281-297.

Max, J. (1960), "Quantizing for minimum distortion," IEEE Transactions on Information Theory, 6, 7-12.

Mulier, F. and Cherkassky, V. (1995), "Self-Organization as an Iterative Kernel Smoothing Process," Neural Computation, 7, 1165-1177.

Pearson, K. (1901) "On Lines and Planes of Closest Fit to Systems of Points in Space," Phil. Mag., 2(6), 559-572.

Rao, C.R. (1964), "The Use and Interpretation of Principal Component Analysis in Applied Research," Sankya A, 26, 329-358.

Ripley, B.D. (1996) *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.

Ritter, H., Martinetz, T., and Schulten, K. (1992), *Neural Computation and Self-Organizing Maps: An Introduction*, Reading, MA: Addison-Wesley.

Sarle, W.S. (1994), "Neural Networks and Statistical Models," in SAS Institute Inc., *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc., pp 1538-1550, ftp://ftp.sas.com/pub/neural/neural1.ps.

Tarpey, T., Luning, L>, and Flury, B. (1994), "Principal points and self-consistent points of elliptical distributions," Annals of Statistics, ?.

Utsugi, A. (1996), "Topology selection for self-organizing maps," Network: Computation in Neural Systems, 7, 727-740, http://www.aist.go.jp/NIBH/~b0616/Lab/index-e.html

Utsugi, A. (1997), "Hyperparameter selection for self-organizing maps," Neural Computation, 9, 623-635, available on-line at http://www.aist.go.jp/NIBH/~b0616/Lab/index-e.html

Yin, H. and Allinson, N.M. (1995), "On the Distribution and Convergence of Feature Space in Self-Organizing Maps," Neural Computation, 7, 1178-1187.

Zeger, K., Vaisey, J., and Gersho, A. (1992), "Globally optimal vector quantizer design by stochastic relaxation," IEEE Transactions on Signal Procesing, 40, 310-322.

Subject: Help! My NN won't learn! What should I do?

The following advice is intended for inexperienced users. Experts may try more daring methods.

If you are using a multilayer perceptron (MLP):

- Check data for outliers. Transform variables or delete bad cases as appropriate to the purpose of the analysis.
- Standardize quantitative inputs as described in "Should I standardize the input variables?"
- Encode categorical inputs as described in "How should categories be encoded?"
- Make sure you have more training cases than the total number of input units. The number of training cases required depends on the amount of noise in the targets and the complexity of the function you are trying to learn, but as a starting point, it's a good idea to have at least 10 times as many training cases as input units. This may not be enough for highly complex functions. For classification problems, the number of cases in the smallest class should be at least several times the number of input units.
- If the target is:
    - quantitative, then it is usually a good idea to standardize the target variable as described in "Should I standardize the target variables?" Use an identity (usually called "linear") output activation function.
    - binary, then use 0/1 coding and a logistic output activation function.
    - categorical with 3 or more categories, then use 1-of-C encoding as described in "How should categories be encoded?" and use a softmax output activation function as described in "What is a softmax activation function?"
- Use a tanh (hyperbolic tangent) activation function for the hidden units. See "Why use activation functions?" for more information.
- Use a bias term (sometimes called a "threshold") in every hidden and output unit. See "Why use a bias/threshold?" for an explanation of why biases are important.
- When the network has hidden units, the results of training may depend critically on the random initial weights. You can set each initial weight (including biases) to a random number such as any of the following:
    - A uniform random variable between -2 and 2.
    - A uniform random variable between -0.2 and -0.2.
    - A normal random variable with a mean of 0 and a standard deviation of 1.
    - A normal random variable with a mean of 0 and a standard deviation of 0.1.

  If any layer in the network has a large number of units, you will need to adjust the initial weights (not including biases) of the connections from the large layer to subsequent layers. Generate random

initial weights as described above, but then divide each of these random weights by the square root of the number of units in the large layer. More sophisticated methods are described by Bishop (1995).

Train the network using several (anywhere from 10 to 1000) different sets of random initial weights. For the operational network, you can either use the weights that produce the smallest training error, or combine several trained networks as described in "How to combine networks?"

- If possible, use conventional numerical optimization techniques as described in "What are conjugate gradients, Levenberg-Marquardt, etc.?" If those techniques are unavailable in the software you are using, get better software. If you can't get better software, use RPROP or Quickprop as described in "What is backprop?" Only as a last resort should you use standard backprop.

- Use batch training, because there are fewer mistakes that can be made with batch training than with incremental (sometimes called "online") training. If you insist on using incremental training, present the training cases to the network in random order. For more details, see "What are batch, incremental, on-line, off-line, deterministic, stochastic, adaptive, instantaneous, pattern, epoch, constructive, and sequential learning?"

- If you have to use standard backprop, you must set the learning rate by trial and error. Experiment with different learning rates. If the weights and errors change very slowly, try higher learning rates. If the weights fluctuate wildly and the error increases during training, try lower learning rates. If you follow all the instructions given above, you could start with a learning rate of .1 for batch training or .01 for incremental training.

  Momentum is not as critical as learning rate, but to be safe, set the momentum to zero. A larger momentum requires a smaller learning rate.

  For more details, see What learning rate should be used for backprop?"

- Use a separate test set to estimate generalization error. If the test error is much higher than the training error, the network is probably overfitting. Read Part 3: Generalization of the FAQ and use one of the methods described there to improve generalization, such as early stopping, weight decay, or Bayesian learning.

- Start with one hidden layer.

  For a classification problem with many categories, start with one unit in the hidden layer; otherwise, start with zero hidden units. Train the network, add one or few hidden units, retrain the network, and repeat. When you get overfitting, stop adding hidden units. For more information on the number of hidden layers and hidden units, see "How many hidden layers should I use?" and "How many hidden units should I use?" in Part 3 of the FAQ.

  If the generalization error is still not satisfactory, you can try:

- adding a second hidden layer
- using an RBF network
- transforming the input variables
- deleting inputs that are not useful
- adding new input variables
- getting more training cases
- etc.

If you are writing your own software, the opportunities for mistakes are limitless. Perhaps the most critical thing for gradient-based algorithms such as backprop is that you compute the gradient (partial derivatives) correctly. The usual backpropagation algorithm will give you the partial derivatives of the objective function with respect to each weight in the network. You can check these partial derivatives by using finite-difference approximations (Gill, Murray, and Wright, 1981) as follows:

1. Be sure to standardize the variables as described above.

2. Initialize the weights `W` as described above. For convenience of notation, let's arrange all the weights in one long vector so we can use a single subsbcript `i` to refer to different weights `W_i.` Call the entire set of values of the initial weights `w0.` So `W` is a vector of variables, and `w0` is a vector of values of those variables.

3. Let's use the symbol `F(W)` to indicate the objective function you are trying to optimize with respect to the weights. If you are using batch training, `F(W)` is computed over the entire training set. If you are using incremental training, choose any one training case and compute `F(W)` for that single training case; use this same training case for all the following steps.

4. Pick any one weight `W_i.` Initially, `W_i = w0_i.`

5. Choose a constant called `h` with a value anywhere from .0001 to .00000001.

6. Change the value of `W_i` from `w0_i` to `w0_i + h.` Do not change any of the other weights. Compute the value of the objective function `f1 = F(W)` using this modified value of `W_i.`

7. Change the value of `W_i` to `w0_i - h.` Do not change any of the other weights. Compute another new value of the objective function `f2 = F(W).`

8. The central finite difference approximation to the partial derivative for `W_i` is `(f2-f1)/(2h).` This value should usually be within about 10% of the partial derivative computed by backpropagation, except for derivatives close to zero. If the finite difference approximation is very different from the partial derivative computed by backpropagation, try a different value of `h.` If no value of `h` provides close agreement between the finite difference approximation and the partial derivative computed by backpropagation, you probably have a bug.

9. Repeat the above computations for each weight `W_i` for `i=1, 2, 3, ...` up to the total number of weights.

References:

Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press.

Gill, P.E., Murray, W. and Wright, M.H. (1981) *Practical Optimization*, Academic Press: London.

Next part is part 3 (of 7). Previous part is part 1.

Archive-name: ai-faq/neural-nets/part3
Last-modified: 2001-05-21
URL: ftp://ftp.sas.com/pub/neural/FAQ3.html
Maintainer: saswss@unx.sas.com (Warren S. Sarle)

This is part 3 (of 7) of a monthly posting to the Usenet newsgroup comp.ai.neural-nets. See the part 1 of this posting for full information what it is all about.

========== Questions ==========

Subject: How is generalization possible?

During learning, the outputs of a supervised neural net come to approximate the target values given the inputs in the training set. This ability may be useful in itself, but more often the purpose of using a neural net is to generalize--i.e., to have the outputs of the net approximate target values given inputs that are *not* in the training

set. Generalizaton is not always possible, despite the blithe assertions of some authors. For example, Caudill and Butler, 1990, p. 8, claim that "A neural network is able to generalize", but they provide no justification for this claim, and they completely neglect the complex issues involved in getting good generalization. Anyone who reads comp.ai.neural-nets is well aware from the numerous posts pleading for help that artificial neural networks do not automatically generalize.

Generalization requires prior knowledge, as pointed out by Hume (1739/1978), Russell (1948), and Goodman (1954/1983) and rigorously proved by Wolpert (1995a, 1996a, 1996b). For any practical application, you have to know what the relevant inputs are (you can't simply include every imaginable input). You have to know a restricted class of input-output functions that contains an adequate approximation to the function you want to learn (you can't use a learning method that is capable of fitting every imaginable function). And you have to know that the cases you want to generalize to bear some resemblance to the training cases. Thus, there are three conditions that are typically necessary--although not sufficient--for good generalization:

1. The first necessary condition is that the inputs to the network contain sufficient information pertaining to the target, so that there exists a mathematical function relating correct outputs to inputs with the desired degree of accuracy. You can't expect a network to learn a nonexistent function--neural nets are not clairvoyant! For example, if you want to forecast the price of a stock, a historical record of the stock's prices is rarely sufficient input; you need detailed information on the financial state of the company as well as general economic conditions, and to avoid nasty surprises, you should also include inputs that can accurately predict wars in the Middle East and earthquakes in Japan. Finding good inputs for a net and collecting enough training data often take far more time and effort than training the network.

2. The second necessary condition is that the function you are trying to learn (that relates inputs to correct outputs) be, in some sense, smooth. In other words, a small change in the inputs should, most of the time, produce a small change in the outputs. For continuous inputs and targets, smoothness of the function implies continuity and restrictions on the first derivative over most of the input space. Some neural nets can learn discontinuities as long as the function consists of a finite number of continuous pieces. Very nonsmooth functions such as those produced by pseudo-random number generators and encryption algorithms cannot be generalized by neural nets. Often a nonlinear transformation of the input space can increase the smoothness of the function and improve generalization.

   For classification, if you do not need to estimate posterior probabilities, then smoothness is not theoretically necessary. In particular, feedforward networks with one hidden layer trained by minimizing the error rate (a very tedious training method) are universally consistent classifiers if the number of hidden units grows at a suitable rate relative to the number of training cases (Devroye, Györfi, and Lugosi, 1996). However, you are likely to get better generalization with realistic sample sizes if the classification boundaries are smoother.

   For Boolean functions, the concept of smoothness is more elusive. It seems intuitively clear that a Boolean network with a small number of hidden units and small weights will compute a "smoother" input-output function than a network with many hidden units and large weights. If you know a good reference characterizing Boolean functions for which good generalization is possible, please inform the FAQ maintainer (saswss@unx.sas.com).

3. The third necessary condition for good generalization is that the training cases be a sufficiently large and representative subset ("sample" in statistical terminology) of the set of all cases that you want to generalize to (the "population" in statistical terminology). The importance of this condition is related to the fact that there are, loosely speaking, two different types of generalization: interpolation and extrapolation. Interpolation applies to cases that are more or less surrounded by nearby training cases;

everything else is extrapolation. In particular, cases that are outside the range of the training data require extrapolation. Cases inside large "holes" in the training data may also effectively require extrapolation. Interpolation can often be done reliably, but extrapolation is notoriously unreliable. Hence it is important to have sufficient training data to avoid the need for extrapolation. Methods for selecting good training sets are discussed in numerous statistical textbooks on sample surveys and experimental design.

Thus, for an input-output function that is smooth, if you have a test case that is close to some training cases, the correct output for the test case will be close to the correct outputs for those training cases. If you have an adequate sample for your training set, every case in the population will be close to a sufficient number of training cases. Hence, under these conditions and with proper training, a neural net will be able to generalize reliably to the population.

If you have more information about the function, e.g. that the outputs should be linearly related to the inputs, you can often take advantage of this information by placing constraints on the network or by fitting a more specific model, such as a linear model, to improve generalization. Extrapolation is much more reliable in linear models than in flexible nonlinear models, although still not nearly as safe as interpolation. You can also use such information to choose the training cases more efficiently. For example, with a linear model, you should choose training cases at the outer limits of the input space instead of evenly distributing them throughout the input space.

References:

Caudill, M. and Butler, C. (1990). *Naturally Intelligent Systems.* MIT Press: Cambridge, Massachusetts.

Devroye, L., Györfi, L., and Lugosi, G. (1996), *A Probabilistic Theory of Pattern Recognition*, NY: Springer.

Goodman, N. (1954/1983), *Fact, Fiction, and Forecast*, 1st/4th ed., Cambridge, MA: Harvard University Press.

Holland, J.H., Holyoak, K.J., Nisbett, R.E., Thagard, P.R. (1986), *Induction: Processes of Inference, Learning, and Discovery*, Cambridge, MA: The MIT Press.

Howson, C. and Urbach, P. (1989), *Scientific Reasoning: The Bayesian Approach*, La Salle, IL: Open Court.

Hume, D. (1739/1978), *A Treatise of Human Nature*, Selby-Bigge, L.A., and Nidditch, P.H. (eds.), Oxford: Oxford University Press.

Plotkin, H. (1993), *Darwin Machines and the Nature of Knowledge*, Cambridge, MA: Harvard University Press.

Russell, B. (1948), *Human Knowledge: Its Scope and Limits*, London: Routledge.

Stone, C.J. (1977), "Consistent nonparametric regression," Annals of Statistics, 5, 595-645.

Stone, C.J. (1982), "Optimal global rates of convergence for nonparametric regression," Annals of Statistics, 10, 1040-1053.

Vapnik, V.N. (1995), *The Nature of Statistical Learning Theory*, NY: Springer.

Wolpert, D.H. (1995a), "The relationship between PAC, the statistical physics framework, the Bayesian framework, and the VC framework," in Wolpert (1995b), 117-214.

Wolpert, D.H. (ed.) (1995b), *The Mathematics of Generalization: The Proceedings of the SFI/CNLS Workshop on Formal Approaches to Supervised Learning*, Santa Fe Institute Studies in the Sciences of Complexity, Volume XX, Reading, MA: Addison-Wesley.

Wolpert, D.H. (1996a), "The lack of a priori distinctions between learning algorithms," Neural Computation, 8, 1341-1390.

Wolpert, D.H. (1996b), "The existence of a priori distinctions between learning algorithms," Neural Computation, 8, 1391-1420.

Subject: How does noise affect generalization?

"Statistical noise" means variation in the target values that is unpredictable from the inputs of a specific network, regardless of the architecture or weights. "Physical noise" refers to variation in the target values that is inherently unpredictable regardless of what inputs are used. Noise in the inputs usually refers to measurement error, so that if the same object or example is presented to the network more than once, the input values differ.

Noise in the actual data is never a good thing, since it limits the accuracy of generalization that can be achieved no matter how extensive the training set is. On the other hand, injecting artificial noise (jitter) into the inputs during training is one of several ways to improve generalization for smooth functions when you have a small training set.

Certain assumptions about noise are necessary for theoretical results. Usually, the noise distribution is assumed to have zero mean and finite variance. The noise in different cases is usually assumed to be independent or to follow some known stochastic model, such as an autoregressive process. The more you know about the noise distribution, the more effectively you can train the network (e.g., McCullagh and Nelder 1989).

If you have noise in the target values, what the network learns depends mainly on the error function. For example, if the noise is independent with finite variance for all training cases, a network that is well-trained using least squares will produce outputs that approximate the conditional mean of the target values (White, 1990; Bishop, 1995). Note that for a binary 0/1 variable, the mean is equal to the probability of getting a 1. Hence, the results in White (1990) immediately imply that for a categorical target with independent noise using 1-of-C coding (see "How should categories be encoded?"), a network that is well-trained using least squares will produce outputs that approximate the posterior probabilities of each class (see Rojas, 1996, if you want a simple explanation of why least-squares estimates probabilities). Posterior probabilities can also be learned using cross-entropy and various other error functions (Finke and Müller, 1994; Bishop, 1995). The conditional median can be learned by least-absolute-value training (White, 1992a). Conditional modes can be approximated by yet other error functions (e.g., Rohwer and van der Rest 1996). For noise distributions that cannot be adequately approximated by a single location estimate (such as the mean, median, or mode), a network can be trained to approximate quantiles (White, 1992a) or mixture components (Bishop, 1995; Husmeier, 1999).

If you have noise in the target values, the mean squared generalization error can never be less than the variance of the noise, no matter how much training data you have. But you can estimate the mean of the target values, conditional on a given set of input values, to any desired degree of accuracy by obtaining a sufficiently large and representative training set, assuming that the function you are trying to learn is one that can indeed be learned by the type of net you are using, and assuming that the complexity of the network is regulated appropriately (White 1990).

Noise in the target values increases the danger of overfitting (Moody 1992).

Noise in the inputs limits the accuracy of generalization, but in a more complicated way than does noise in the targets. In a region of the input space where the function being learned is fairly flat, input noise will have little effect. In regions where that function is steep, input noise can degrade generalization severely.

Furthermore, if the target function is Y=f(X), but you observe noisy inputs X+D, you cannot obtain an arbitrarily accurate estimate of f(X) given X+D no matter how large a training set you use. The net will not learn f(X), but will instead learn a convolution of f(X) with the distribution of the noise D (see "What is jitter?)"

For more details, see one of the statistically-oriented references on neural nets such as:

Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press, especially section 6.4.

Finke, M., and Müller, K.-R. (1994), "Estimating a-posteriori probabilities using stochastic network models," in Mozer, Smolensky, Touretzky, Elman, & Weigend, eds., *Proceedings of the 1993 Connectionist Models Summer School*, Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 324-331.

Geman, S., Bienenstock, E. and Doursat, R. (1992), "Neural Networks and the Bias/Variance Dilemma", Neural Computation, 4, 1-58.

Husmeier, D. (1999), *Neural Networks for Conditional Probability Estimation: Forecasting Beyond Point Predictions*, Berlin: Springer Verlag, ISBN 185233095.

McCullagh, P. and Nelder, J.A. (1989) *Generalized Linear Models*, 2nd ed., London: Chapman & Hall.

Moody, J.E. (1992), "The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems", in Moody, J.E., Hanson, S.J., and Lippmann, R.P., *Advances in Neural Information Processing Systems 4*, 847-854.

Ripley, B.D. (1996) *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.

Rohwer, R., and van der Rest, J.C. (1996), "Minimum description length, regularization, and multimodal data," Neural Computation, 8, 595-609.

Rojas, R. (1996), "A short proof of the posterior probability property of classifier neural networks," Neural Computation, 8, 41-43.

White, H. (1990), "Connectionist Nonparametric Regression: Multilayer Feedforward Networks Can Learn Arbitrary Mappings," Neural Networks, 3, 535-550. Reprinted in White (1992).

White, H. (1992a), "Nonparametric Estimation of Conditional Quantiles Using Neural Networks," in Page, C. and Le Page, R. (eds.), *Proceedings of the 23rd Sympsium on the Interface: Computing Science and Statistics*, Alexandria, VA: American Statistical Association, pp. 190-199. Reprinted in White (1992b).

White, H. (1992b), *Artificial Neural Networks: Approximation and Learning Theory*, Blackwell.

Subject: What is overfitting and how can I avoid it?

The critical issue in developing a neural network is generalization: how well will the network make predictions for cases that are not in the training set? NNs, like other flexible nonlinear estimation methods such as kernel regression and smoothing splines, can suffer from either underfitting or overfitting. A network that is not sufficiently complex can fail to detect fully the signal in a complicated data set, leading to underfitting. A network that is too complex may fit the noise, not just the signal, leading to overfitting. Overfitting is especially dangerous because it can easily lead to predictions that are far beyond the range of the training data with many of the common types of NNs. Overfitting can also produce wild predictions in multilayer perceptrons even with noise-free data.

For an elementary discussion of overfitting, see Smith (1996). For a more rigorous approach, see the article by Geman, Bienenstock, and Doursat (1992) on the bias/variance trade-off (it's not really a dilemma). We are talking about statistical bias here: the difference between the average value of an estimator and the correct value. Underfitting produces excessive bias in the outputs, whereas overfitting produces excessive variance. There are graphical examples of overfitting and underfitting in Sarle (1995, 1999).

The best way to avoid overfitting is to use *lots* of training data. If you have at least 30 times as many training cases as there are weights in the network, you are unlikely to suffer from much overfitting, although you may get some slight overfitting no matter how large the training set is. For noise-free data, 5 times as many training cases as weights may be sufficient. But you can't arbitrarily reduce the number of weights for fear of underfitting.

Given a fixed amount of training data, there are at least six approaches to avoiding underfitting and overfitting, and hence getting good generalization:

- Model selection
- Jittering
- Early stopping
- Weight decay
- Bayesian learning
- Combining networks

The first five approaches are based on well-understood theory. Methods for combining networks do not have such a sound theoretical basis but are the subject of current research. These six approaches are discussed in more detail under subsequent questions.

The complexity of a network is related to both the number of weights and the size of the weights. Model

selection is concerned with the number of weights, and hence the number of hidden units and layers. The more weights there are, relative to the number of training cases, the more overfitting amplifies noise in the targets (Moody 1992). The other approaches listed above are concerned, directly or indirectly, with the size of the weights. Reducing the size of the weights reduces the "effective" number of weights--see Moody (1992) regarding weight decay and Weigend (1994) regarding early stopping. Bartlett (1997) obtained learning-theory results in which generalization error is related to the L_1 norm of the weights instead of the VC dimension.

Overfitting is not confined to NNs with hidden units. Overfitting can occur in generalized linear models (networks with no hidden units) if either or both of the following conditions hold:

1. The number of input variables (and hence the number of weights) is large with respect to the number of training cases. Typically you would want at least 10 times as many training cases as input variables, but with noise-free targets, twice as many training cases as input variables would be more than adequate. These requirements are smaller than those stated above for networks with hidden layers, because hidden layers are prone to creating ill-conditioning and other pathologies.

2. The input variables are highly correlated with each other. This condition is called "multicollinearity" in the statistical literature. Multicollinearity can cause the weights to become extremely large because of numerical ill-conditioning--see "How does ill-conditioning affect NN training?"

Methods for dealing with these problems in the statistical literature include ridge regression (similar to weight decay), partial least squares (similar to Early stopping), and various methods with even stranger names, such as the lasso and garotte (van Houwelingen and le Cessie, ????).

References:

Bartlett, P.L. (1997), "For valid generalization, the size of the weights is more important than the size of the network," in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) *Advances in Neural Information Processing Systems 9*, Cambrideg, MA: The MIT Press, pp. 134-140.

Geman, S., Bienenstock, E. and Doursat, R. (1992), "Neural Networks and the Bias/Variance Dilemma", Neural Computation, 4, 1-58.

Moody, J.E. (1992), "The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems", in Moody, J.E., Hanson, S.J., and Lippmann, R.P., *Advances in Neural Information Processing Systems 4*, 847-854.

Sarle, W.S. (1995), "Stopped Training and Other Remedies for Overfitting," *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, 352-360, ftp://ftp.sas.com/pub/neural/inter95.ps.Z (this is a *very large* compressed postscript file, *747K*, 10 pages)

Sarle, W.S. (1999), "Donoho-Johnstone Benchmarks: Neural Net Results," ftp://ftp.sas.com/pub/neural/dojo/dojo.html

Smith, M. (1996). *Neural Networks for Statistical Modeling*, Boston: International Thomson Computer Press, ISBN 1-850-32842-0.

van Houwelingen,H.C., and le Cessie, S. (????), "Shrinkage and penalized likelihood as methods to

improve predictive accuracy," http://www.medstat.medfac.leidenuniv.nl/ms/HH/Files/shrinkage.pdf and http://www.medstat.medfac.leidenuniv.nl/ms/HH/Files/figures.pdf

Weigend, A. (1994), "On overfitting and the effective number of hidden units," *Proceedings of the 1993 Connectionist Models Summer School*, 335-342.

Subject: What is jitter? (Training with noise)

Jitter is artificial noise deliberately added to the inputs during training. Training with jitter is a form of smoothing related to kernel regression (see "What is GRNN?"). It is also closely related to regularization methods such as weight decay and ridge regression.

Training with jitter works because the functions that we want NNs to learn are mostly smooth. NNs can learn functions with discontinuities, but the functions must be piecewise continuous in a finite number of regions if our network is restricted to a finite number of hidden units.

In other words, if we have two cases with similar inputs, the desired outputs will usually be similar. That means we can take any training case and generate new training cases by adding small amounts of jitter to the inputs. As long as the amount of jitter is sufficiently small, we can assume that the desired output will not change enough to be of any consequence, so we can just use the same target value. The more training cases, the merrier, so this looks like a convenient way to improve training. But too much jitter will obviously produce garbage, while too little jitter will have little effect (Koistinen and Holmström 1992).

Consider any point in the input space, not necessarily one of the original training cases. That point could possibly arise as a jittered input as a result of jittering any of several of the original neighboring training cases. The average target value at the given input point will be a weighted average of the target values of the original training cases. For an infinite number of jittered cases, the weights will be proportional to the probability densities of the jitter distribution, located at the original training cases and evaluated at the given input point. Thus the average target values given an infinite number of jittered cases will, by definition, be the Nadaraya-Watson kernel regression estimator using the jitter density as the kernel. Hence, training with jitter is an approximation to training with the kernel regression estimator as target. Choosing the amount (variance) of jitter is equivalent to choosing the bandwidth of the kernel regression estimator (Scott 1992).

When studying nonlinear models such as feedforward NNs, it is often helpful first to consider what happens in linear models, and then to see what difference the nonlinearity makes. So let's consider training with jitter in a linear model. Notation:

```
x_ij is the value of the jth input (j=1, ..., p) for the
     ith training case (i=1, ..., n).
X={x_ij} is an n by p matrix.
y_i is the target value for the ith training case.
Y={y_i} is a column vector.
```

Without jitter, the least-squares weights are `B = inv(X'X)X'Y`, where "inv" indicates a matrix inverse and "'" indicates transposition. Note that if we replicate each training case `c` times, or equivalently stack `c` copies of the `X` and `Y` matrices on top of each other, the least-squares weights are `inv(cX'X)cX'Y =`

(1/c)inv(X'X)cX'Y = B, same as before.

With jitter, x_ij is replaced by c cases x_ij+z_ijk, k=1, ..., c, where z_ijk is produced by some random number generator, usually with a normal distribution with mean 0 and standard deviation s, and the z_ijk's are all independent. In place of the n by p matrix X, this gives us a big matrix, say Q, with cn rows and p columns. To compute the least-squares weights, we need Q'Q. Let's consider the jth diagonal element of Q'Q, which is

```
                  2           2       2
sum (x_ij+z_ijk) = sum (x_ij + z_ijk + 2 x_ij z_ijk)
i,k                i,k
```

which is approximately, for c large,

```
           2     2
c(sum x_ij  + ns )
   i
```

which is c times the corresponding diagonal element of X'X plus ns^2. Now consider the u,vth off-diagonal element of Q'Q, which is

```
sum (x_iu+z_iuk)(x_iv+z_ivk)
i,k
```

which is approximately, for c large,

```
c(sum x_iu x_iv)
   i
```

which is just c times the corresponding element of X'X. Thus, Q'Q equals c(X'X+ns^2I), where I is an identity matrix of appropriate size. Similar computations show that the crossproduct of Q with the target values is cX'Y. Hence the least-squares weights with jitter of variance s^2 are given by

```
    2                2                    2
B(ns ) = inv(c(X'X+ns I))cX'Y = inv(X'X+ns I)X'Y
```

In the statistics literature, B(ns^2) is called a ridge regression estimator with ridge value ns^2.

If we were to add jitter to the target values Y, the cross-product X'Y would not be affected for large c for the same reason that the off-diagonal elements of X'X are not afected by jitter. Hence, adding jitter to the targets will not change the optimal weights; it will just slow down training (An 1996).

The ordinary least squares training criterion is (Y-XB)'(Y-XB). Weight decay uses the training criterion (Y-XB)'(Y-XB)+d^2B'B, where d is the decay rate. Weight decay can also be implemented by inventing artificial training cases. Augment the training data with p new training cases containing the matrix dI for the inputs and a zero vector for the targets. To put this in a formula, let's use A;B to indicate the matrix A stacked on top of the matrix B, so (A;B)'(C;D)=A'C+B'D. Thus the augmented inputs are X;dI and the augmented targets are Y;0, where 0 indicates the zero vector of the appropriate size. The squared error for the augmented training data is:

```
(Y;0-(X;dI)B)'(Y;0-(X;dI)B)
= (Y;0)'(Y;0) - 2(Y;0)'(X;dI)B + B'(X;dI)'(X;dI)B
= Y'Y - 2Y'XB + B'(X'X+d^2I)B
= Y'Y - 2Y'XB + B'X'XB + B'(d^2I)B
= (Y-XB)'(Y-XB)+d^2B'B
```

which is the weight-decay training criterion. Thus the weight-decay estimator is:

```
inv[(X;dI)'(X;dI)](X;dI)'(Y;0) = inv(X'X+d^2I)X'Y
```

which is the same as the jitter estimator `B(d^2),` i.e. jitter with variance `d^2/n.` The equivalence between the weight-decay estimator and the jitter estimator does *not* hold for nonlinear models unless the jitter variance is small relative to the curvature of the nonlinear function (An 1996). However, the equivalence of the two estimators for linear models suggests that they will often produce similar results even for nonlinear models. Details for nonlinear models, including classification problems, are given in An (1996).

`B(0)` is obviously the ordinary least-squares estimator. It can be shown that as `s^2` increases, the Euclidean norm of `B(ns^2)` decreases; in other words, adding jitter causes the weights to shrink. It can also be shown that under the usual statistical assumptions, there always exists some value of `ns^2 > 0` such that `B(ns^2)` provides better expected generalization than `B(0).` Unfortunately, there is no way to calculate a value of `ns^2` from the training data that is guaranteed to improve generalization. There are other types of shrinkage estimators called Stein estimators that *do* guarantee better generalization than `B(0),` but I'm not aware of a nonlinear generalization of Stein estimators applicable to neural networks.

The statistics literature describes numerous methods for choosing the ridge value. The most obvious way is to estimate the generalization error by cross-validation, generalized cross-validation, or bootstrapping, and to choose the ridge value that yields the smallest such estimate. There are also quicker methods based on empirical Bayes estimation, one of which yields the following formula, useful as a first guess:

$$s_1^2 = \frac{p(Y-XB(0))'(Y-XB(0))}{n(n-p)B(0)'B(0)}$$

You can iterate this a few times:

$$s_{l+1}^2 = \frac{p(Y-XB(0))'(Y-XB(0))}{n(n-p)B(s_l)'B(s_l)}$$

Note that the more training cases you have, the less noise you need.

References:

An, G. (1996), "The effects of adding noise during backpropagation training on a generalization performance," Neural Computation, 8, 643-674.

Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press.

Holmström, L. and Koistinen, P. (1992) "Using additive noise in back-propagation training", IEEE Transaction on Neural Networks, 3, 24-38.

Koistinen, P. and Holmström, L. (1992) "Kernel regression and backpropagation training with noise," NIPS4, 1033-1039.

Reed, R.D., and Marks, R.J, II (1999), *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*, Cambridge, MA: The MIT Press, ISBN 0-262-18190-8.

Scott, D.W. (1992) *Multivariate Density Estimation*, Wiley.

Vinod, H.D. and Ullah, A. (1981) *Recent Advances in Regression Methods*, NY: Marcel-Dekker.

Subject: What is early stopping?

NN practitioners often use nets with many times as many parameters as training cases. E.g., Nelson and Illingworth (1991, p. 165) discuss training a network with 16,219 parameters with only 50 training cases! The method used is called "early stopping" or "stopped training" and proceeds as follows:

1. Divide the available data into training and validation sets.
2. Use a large number of hidden units.
3. Use very small random initial values.
4. Use a slow learning rate.
5. Compute the validation error rate periodically during training.
6. Stop training when the validation error rate "starts to go up".

It is crucial to realize that the validation error is *not* a good estimate of the generalization error. One method for getting an unbiased estimate of the generalization error is to run the net on a third set of data, the test set, that is not used at all during the training process. For other methods, see "How can generalization error be estimated?"

Early stopping has several advantages:

- It is fast.
- It can be applied successfully to networks in which the number of weights far exceeds the sample size.
- It requires only one major decision by the user: what proportion of validation cases to use.

But there are several unresolved practical issues in early stopping:

- How many cases do you assign to the training and validation sets? Rules of thumb abound, but appear to be no more than folklore. The only systematic results known to the FAQ maintainer are in Sarle (1995), which deals only with the case of a single input. Amari et al. (1995) attempts a theoretical approach but contains serious errors that completely invalidate the results, especially the incorrect assumption that the direction of approach to the optimum is distributed isotropically.
- Do you split the data into training and validation sets randomly or by some systematic algorithm?

- How do you tell when the validation error rate "starts to go up"? It may go up and down numerous times during training. The safest approach is to train to convergence, then go back and see which iteration had the lowest validation error. For more elaborate algorithms, see Prechelt (1994, 1998).

Statisticians tend to be skeptical of stopped training because it appears to be statistically inefficient due to the use of the split-sample technique; i.e., neither training nor validation makes use of the entire sample, and because the usual statistical theory does not apply. However, there has been recent progress addressing both of the above concerns (Wang 1994).

Early stopping is closely related to ridge regression. If the learning rate is sufficiently small, the sequence of weight vectors on each iteration will approximate the path of continuous steepest descent down the error surface. Early stopping chooses a point along this path that optimizes an estimate of the generalization error computed from the validation set. Ridge regression also defines a path of weight vectors by varying the ridge value. The ridge value is often chosen by optimizing an estimate of the generalization error computed by cross-validation, generalized cross-validation, or bootstrapping (see "What are cross-validation and bootstrapping?") There always exists a positive ridge value that will improve the expected generalization error in a linear model. A similar result has been obtained for early stopping in linear models (Wang, Venkatesh, and Judd 1994). In linear models, the ridge path lies close to, but does not coincide with, the path of continuous steepest descent; in nonlinear models, the two paths can diverge widely. The relationship is explored in more detail by Sjöberg and Ljung (1992).

References:

S. Amari, N.Murata, K.-R. Muller, M. Finke, H. Yang. Asymptotic Statistical Theory of Overtraining and Cross-Validation. METR 95-06, 1995, Department of Mathematical Engineering and Information Physics, University of Tokyo, Hongo 7-3-1, Bunkyo-ku, Tokyo 113, Japan.

Finnof, W., Hergert, F., and Zimmermann, H.G. (1993), "Improving model selection by nonconvergent methods," Neural Networks, 6, 771-783.

Nelson, M.C. and Illingworth, W.T. (1991), *A Practical Guide to Neural Nets*, Reading, MA: Addison-Wesley.

Orr, G.B., and Mueller, K.-R., eds. (1998), *Neural Networks: Tricks of the Trade*, Berlin: Springer, ISBN 3-540-65311-2.

Prechelt, L. (1998), "Early stopping--But when?" in Orr and Mueller (1998), 55-69.

Prechelt, L. (1994), "PROBEN1--A set of neural network benchmark problems and benchmarking rules," Technical Report 21/94, Universitat Karlsruhe, 76128 Karlsruhe, Germany, ftp://ftp.ira.uka.de/pub/papers/techreports/1994/1994-21.ps.gz.

Sarle, W.S. (1995), "Stopped Training and Other Remedies for Overfitting," *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, 352-360, ftp://ftp.sas.com/pub/neural/inter95.ps.Z (this is a *very large* compressed postscript file, *747K*, 10 pages)

Sjöberg, J. and Ljung, L. (1992), "Overtraining, Regularization, and Searching for Minimum in Neural Networks," Technical Report LiTH-ISY-I-1297, Department of Electrical Engineering, Linkoping University, S-581 83 Linkoping, Sweden, http://www.control.isy.liu.se .

Wang, C. (1994), *A Theory of Generalisation in Learning Machines with Neural Network Application*, Ph.D. thesis, University of Pennsylvania.

Wang, C., Venkatesh, S.S., and Judd, J.S. (1994), "Optimal Stopping and Effective Machine Complexity in Learning," NIPS6, 303-310.

Weigend, A. (1994), "On overfitting and the effective number of hidden units," *Proceedings of the 1993 Connectionist Models Summer School*, 335-342.

Subject: What is weight decay?

Weight decay adds a penalty term to the error function. The usual penalty is the sum of squared weights times a decay constant. In a linear model, this form of weight decay is equivalent to ridge regression. See "What is jitter?" for more explanation of ridge regression.

Weight decay is a subset of regularization methods. The penalty term in weight decay, by definition, penalizes large weights. Other regularization methods may involve not only the weights but various derivatives of the output function (Bishop 1995).

The weight decay penalty term causes the weights to converge to smaller absolute values than they otherwise would. Large weights can hurt generalization in two different ways. Excessively large weights leading to hidden units can cause the output function to be too rough, possibly with near discontinuities. Excessively large weights leading to output units can cause wild outputs far beyond the range of the data if the output activation function is not bounded to the same range as the data. To put it another way, large weights can cause excessive variance of the output (Geman, Bienenstock, and Doursat 1992). According to Bartlett (1997), the size (L_1 norm) of the weights is more important than the number of weights in determining generalization.

Other penalty terms besides the sum of squared weights are sometimes used. *Weight elimination* (Weigend, Rumelhart, and Huberman 1991) uses:

```
          (w_i)^2
sum  -------------
 i   (w_i)^2 + c^2
```

where `w_i` is the `ith` weight and `c` is a user-specified constant. Whereas decay using the sum of squared weights tends to shrink the large coefficients more than the small ones, weight elimination tends to shrink the small coefficients more, and is therefore more useful for suggesting subset models (pruning).

The generalization ability of the network can depend crucially on the decay constant, especially with small training sets. One approach to choosing the decay constant is to train several networks with different amounts of decay and estimate the generalization error for each; then choose the decay constant that minimizes the estimated generalization error. Weigend, Rumelhart, and Huberman (1991) iteratively update the decay constant during training.

There are other important considerations for getting good results from weight decay. You must either

standardize the inputs and targets, or adjust the penalty term for the standard deviations of all the inputs and targets. It is usually a good idea to omit the biases from the penalty term.

A fundamental problem with weight decay is that different types of weights in the network will usually require different decay constants for good generalization. At the very least, you need three different decay constants for input-to-hidden, hidden-to-hidden, and hidden-to-output weights. Adjusting all these decay constants to produce the best estimated generalization error often requires vast amounts of computation.

Fortunately, there is a superior alternative to weight decay: hierarchical Bayesian learning. Bayesian learning makes it possible to estimate efficiently numerous decay constants.

References:

Bartlett, P.L. (1997), "For valid generalization, the size of the weights is more important than the size of the network," in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) *Advances in Neural Information Processing Systems 9*, Cambrideg, MA: The MIT Press, pp. 134-140.

Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press.

Geman, S., Bienenstock, E. and Doursat, R. (1992), "Neural Networks and the Bias/Variance Dilemma", Neural Computation, 4, 1-58.

Ripley, B.D. (1996) *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.

Weigend, A. S., Rumelhart, D. E., & Huberman, B. A. (1991). Generalization by weight-elimination with application to forecasting. In: R. P. Lippmann, J. Moody, & D. S. Touretzky (eds.), Advances in Neural Information Processing Systems 3, San Mateo, CA: Morgan Kaufmann.

- -

Subject: What is Bayesian Learning?

By Radford Neal.

Conventional training methods for multilayer perceptrons ("backprop" nets) can be interpreted in statistical terms as variations on maximum likelihood estimation. The idea is to find a *single* set of weights for the network that maximize the fit to the training data, perhaps modified by some sort of weight penalty to prevent overfitting.

The Bayesian school of statistics is based on a different view of what it means to learn from data, in which probability is used to represent uncertainty about the relationship being learned (a use that is shunned in conventional--i.e., frequentist--statistics). Before we have seen any data, our *prior* opinions about what the true relationship might be can be expresssed in a probability distribution over the network weights that define this relationship. After we look at the data (or after our program looks at the data), our revised opinions are captured by a *posterior* distribution over network weights. Network weights that seemed plausible before, but which don't match the data very well, will now be seen as being much less likely, while the probability for values of the weights that do fit the data well will have increased.

Typically, the purpose of training is to make predictions for future cases in which only the inputs to the network are known. The result of conventional network training is a single set of weights that can be used to make such predictions. In contrast, the result of Bayesian training is a posterior *distribution* over network weights. If the inputs of the network are set to the values for some new case, the posterior distribution over network weights will give rise to a distribution over the outputs of the network, which is known as the *predictive distribution* for this new case. If a single-valued prediction is needed, one might use the mean of the predictive distribution, but the full predictive distribution also tells you how uncertain this prediction is.

Why bother with all this? The hope is that Bayesian methods will provide solutions to such fundamental problems as:

- How to judge the uncertainty of predictions. This can be solved by looking at the predictive distribution, as described above.
- How to choose an appropriate network architecture (eg, the number hidden layers, the number of hidden units in each layer).
- How to adapt to the characteristics of the data (eg, the smoothness of the function, the degree to which different inputs are relevant).

Good solutions to these problems, especially the last two, depend on using the right prior distribution, one that properly represents the uncertainty that you probably have about which inputs are relevant, how smooth the function is, how much noise there is in the observations, etc. Such carefully vague prior distributions are usually defined in a hierarchical fashion, using *hyperparameters*, some of which are analogous to the weight decay constants of more conventional training procedures. The use of hyperparameters is discussed by Mackay (1992a, 1992b, 1995) and Neal (1993a, 1996), who in particular use an "Automatic Relevance Determination" scheme that aims to allow many possibly-relevant inputs to be included without damaging effects.

Selection of an appropriate network architecture is another place where prior knowledge plays a role. One approach is to use a very general architecture, with lots of hidden units, maybe in several layers or groups, controlled using hyperparameters. This approach is emphasized by Neal (1996), who argues that there is no statistical need to limit the complexity of the network architecture when using well-designed Bayesian methods. It is also possible to choose between architectures in a Bayesian fashion, using the "evidence" for an architecture, as discussed by Mackay (1992a, 1992b).

Implementing all this is one of the biggest problems with Bayesian methods. Dealing with a distribution over weights (and perhaps hyperparameters) is not as simple as finding a single "best" value for the weights. Exact analytical methods for models as complex as neural networks are out of the question. Two approaches have been tried:

1. Find the weights/hyperparameters that are most probable, using methods similar to conventional training (with regularization), and then approximate the distribution over weights using information available at this maximum.
2. Use a Monte Carlo method to sample from the distribution over weights. The most efficient implementations of this use dynamical Monte Carlo methods whose operation resembles that of backprop with momentum.

The first method comes in two flavours. Buntine and Weigend (1991) describe a procedure in which the hyperparameters are first integrated out analytically, and numerical methods are then used to find the most probable weights. MacKay (1992a, 1992b) instead finds the values for the hyperparameters that are most likely, integrating over the weights (using an approximation around the most probable weights, conditional on the

hyperparameter values). There has been some controversy regarding the merits of these two procedures, with Wolpert (1993) claiming that analytically integrating over the hyperparameters is preferable because it is "exact". This criticism has been rebutted by Mackay (1993). It would be inappropriate to get into the details of this controversy here, but it is important to realize that the procedures based on analytical integration over the hyperparameters do *not* provide exact solutions to any of the problems of practical interest. The discussion of an analogous situation in a different statistical context by O'Hagan (1985) may be illuminating.

Monte Carlo methods for Bayesian neural networks have been developed by Neal (1993a, 1996). In this approach, the posterior distribution is represented by a sample of perhaps a few dozen sets of network weights. The sample is obtained by simulating a Markov chain whose equilibrium distribution is the posterior distribution for weights and hyperparameters. This technique is known as "Markov chain Monte Carlo (MCMC)"; see Neal (1993b) for a review. The method is exact in the limit as the size of the sample and the length of time for which the Markov chain is run increase, but convergence can sometimes be slow in practice, as for any network training method.

Work on Bayesian neural network learning has so far concentrated on multilayer perceptron networks, but Bayesian methods can in principal be applied to other network models, as long as they can be interpreted in statistical terms. For some models (eg, RBF networks), this should be a fairly simple matter; for others (eg, Boltzmann Machines), substantial computational problems would need to be solved.

Software implementing Bayesian neural network models (intended for research use) is available from the home pages of David MacKay and Radford Neal.

There are many books that discuss the general concepts of Bayesian inference, though they mostly deal with models that are simpler than neural networks. Here are some recent ones:

Bernardo, J. M. and Smith, A. F. M. (1994) *Bayesian Theory*, New York: John Wiley.

Gelman, A., Carlin, J.B., Stern, H.S., and Rubin, D.B. (1995) *Bayesian Data Analysis*, London: Chapman & Hall, ISBN 0-412-03991-5.

O'Hagan, A. (1994) *Bayesian Inference* (Volume 2B in Kendall's Advanced Theory of Statistics), ISBN 0-340-52922-9.

Robert, C. P. (1995) *The Bayesian Choice*, New York: Springer-Verlag.

The following books and papers have tutorial material on Bayesian learning as applied to neural network models:

Bishop, C. M. (1995) *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press.

Lee, H.K.H (1999), *Model Selection and Model Averaging for Neural Networks*, Doctoral dissertation, Carnegie Mellon University, Pittsburgh, USA, http://lib.stat.cmu.edu/~herbie/thesis.html

MacKay, D. J. C. (1995) "Probable networks and plausible predictions - a review of practical Bayesian methods for supervised neural networks", available at ftp://wol.ra.phy.cam.ac.uk/pub/www/mackay/network.ps.gz.

Mueller, P. and Insua, D.R. (1995) "Issues in Bayesian Analysis of Neural Network Models," Neural Computation, 10, 571-592, (also Institute of Statistics and Decision Sciences Working Paper 95-31), ftp://ftp.isds.duke.edu/pub/WorkingPapers/95-31.ps

Neal, R. M. (1996) *Bayesian Learning for Neural Networks*, New York: Springer-Verlag, ISBN 0-387-94724-8.

Ripley, B. D. (1996) *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.

Thodberg, H. H. (1996) "A review of Bayesian neural networks with an application to near infrared spectroscopy", *IEEE Transactions on Neural Networks*, 7, 56-72.

Some other references:

Bernardo, J.M., DeGroot, M.H., Lindley, D.V. and Smith, A.F.M., eds., (1985), *Bayesian Statistics 2*, Amsterdam: Elsevier Science Publishers B.V. (North-Holland).

Buntine, W. L. and Weigend, A. S. (1991) "Bayesian back-propagation", *Complex Systems*, 5, 603-643.

MacKay, D. J. C. (1992a) "Bayesian interpolation", *Neural Computation*, 4, 415-447.

MacKay, D. J. C. (1992b) "A practical Bayesian framework for backpropagation networks," *Neural Computation*, 4, 448-472.

MacKay, D. J. C. (1993) "Hyperparameters: Optimize or Integrate Out?", available at ftp://wol.ra.phy.cam.ac.uk/pub/www/mackay/alpha.ps.gz.

Neal, R. M. (1993a) "Bayesian learning via stochastic dynamics", in C. L. Giles, S. J. Hanson, and J. D. Cowan (editors), *Advances in Neural Information Processing Systems 5*, San Mateo, California: Morgan Kaufmann, 475-482.

Neal, R. M. (1993b) *Probabilistic Inference Using Markov Chain Monte Carlo Methods*, available at ftp://ftp.cs.utoronto.ca/pub/radford/review.ps.Z.

O'Hagan, A. (1985) "Shoulders in hierarchical models", in J. M. Bernardo, M. H. DeGroot, D. V. Lindley, and A. F. M. Smith (editors), *Bayesian Statistics 2*, Amsterdam: Elsevier Science Publishers B.V. (North-Holland), 697-710.

Sarle, W. S. (1995) "Stopped Training and Other Remedies for Overfitting," *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, 352-360, ftp://ftp.sas.com/pub/neural/inter95.ps.Z (this is a *very large* compressed postscript file, *747K*, 10 pages)

Wolpert, D. H. (1993) "On the use of evidence in neural networks", in C. L. Giles, S. J. Hanson, and J. D. Cowan (editors), *Advances in Neural Information Processing Systems 5*, San Mateo, California: Morgan Kaufmann, 539-546.

Finally, David MacKay maintains a FAQ about Bayesian methods for neural networks, at

http://wol.ra.phy.cam.ac.uk/mackay/Bayes_FAQ.html .

Comments on Bayesian learning

By Warren Sarle.

Bayesian purists may argue over the proper way to do a Bayesian analysis, but even the crudest Bayesian computation (maximizing over both parameters and hyperparameters) is shown by Sarle (1995) to generalize better than early stopping when learning nonlinear functions. This approach requires the use of slightly informative hyperpriors and at least twice as many training cases as weights in the network. A full Bayesian analysis by MCMC can be expected to work even better under even broader conditions. Bayesian learning works well by frequentist standards--what MacKay calls the "evidence framework" is used by frequentist statisticians under the name "empirical Bayes." Although considerable research remains to be done, Bayesian learning seems to be the most promising approach to training neural networks.

Bayesian learning should not be confused with the "Bayes classifier." In the latter, the distribution of the inputs given the target class is assumed to be known exactly, and the prior probabilities of the classes are assumed known, so that the posterior probabilities can be computed by a (theoretically) simple application of Bayes' theorem. The Bayes classifier involves no learning--you must already know everything that needs to be known! The Bayes classifier is a gold standard that can almost never be used in real life but is useful in theoretical work and in simulation studies that compare classification methods. The term "Bayes rule" is also used to mean any classification rule that gives results identical to those of a Bayes classifier.

Bayesian learning also should not be confused with the "naive" or "idiot's" Bayes classifier (Warner et al. 1961; Ripley, 1996), which assumes that the inputs are conditionally independent given the target class. The naive Bayes classifier is usually applied with categorical inputs, and the distribution of each input is estimated by the proportions in the training set; hence the naive Bayes classifier is a frequentist method.

The term "Bayesian network" often refers not to a neural network but to a belief network (also called a causal net, influence diagram, constraint network, qualitative Markov network, or gallery). Belief networks are more closely related to expert systems than to neural networks, and do not necessarily involve learning (Pearl, 1988; Ripley, 1996). Here are some URLs on Bayesian belief networks:

- http://bayes.stat.washington.edu/almond/belief.html
- http://www.cs.orst.edu/~dambrosi/bayesian/frame.html
- http://www2.sis.pitt.edu/~genie
- http://www.research.microsoft.com/dtg/msbn

References for comments:

Pearl, J. (1988) *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, San Mateo, CA: Morgan Kaufmann.

Ripley, B. D. (1996) *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.

Warner, H.R., Toronto, A.F., Veasy, L.R., and Stephenson, R. (1961), "A mathematical model for medical diagnosis--application to congenital heart disease," J. of the American Medical Association,

177, 177-184.

- -

Subject: How to combine networks?

Methods for combining networks are a subject of active research. Many different methods with different purposes have been proposed. The properties and relationships of these methods are just beginning to be understood. Some methods, such as boosting, are remedies for underfitting. Other methods, such as bagging, are mainly remedies for overfitting or instability. Bayesian learning naturally leads to model averaging (Hoeting et al., 1999). A good general reference is the book edited by Sharkey (1999), especially the article by Drucker (1999). Regarding the effects of bagging and weight decay used together, see Taniguchi and Tresp (1997).

Here is a list of terms used for various methods of combining models, mostly taken from Christoph M. Friedrich's web page (see below):

- Adaboost
- ADDEMUP
- arcing: adaptive recombination of classifiers
- bagging: bootstrap aggregation
- bag-stacking: bagging plus stacking
- boosting
- cascading
- combination of classifiers
- committees of networks
- consensus theory
- cragging: cross aggregation (like k-fold cross validation)
- dagging: disjoint-sample aggregation
- dag-stacking: dagging plus stacking
- divide and conquer classifiers
- ensembles
- hagging: half-sample aggregation
- mixture of experts
- multiple classifier systems:
- multi-stage and multi-level classifiers
- OLC: optimal linear combination
- pandemonium of reflective agents
- sieving algorithms
- stacking: feeding outputs of several models (and possibly the the original inputs) into a second-level model
- voting

URLs:

- Christoph M. Friedrich's web page, "Combinations of Classifiers and Regressors Bibliography and Guide to Internet Resources" at http://www.tussy.uni-wh.de/~chris/ensemble/ensemble.html

- Tirthankar RayChaudhuri's web page on combining estimators at http://www-

comp.mpce.mq.edu.au/~tirthank/combest.html

- Robert E. Schapire's boosting page at http://www.research.att.com/~schapire/boost.html
- http://www.boosting.org/

References:

Clemen, Robert T. (1989), "Combining forecasts: A review and annotated bibliography", International Journal of Forecasting, Vol 5, pp 559-584.

Drucker, H. (1999), "Boosting using neural networks," in Sharkey (1999), pp. 51-78.

Hoeting, J. A., Madigan, D., Raftery, A.E., and Volinsky, C.T. (1999) "Bayesian Model Averaging: A Tutorial (with discussion)," Statistical Science, 14:4, 382-417. Corrected version available at http://www.stat.washington.edu/www/research/online/hoeting1999.pdf

Sharkey, A.J.C. (1999), *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*, London: Springer.

Taniguchi, M., and Tresp, V. (1997), "Averaging regularized estimators," Neural Computation, 9, 1163-1178.

Subject: How many hidden layers should I use?

You may not need any hidden layers at all. Linear and generalized linear models are useful in a wide variety of applications (McCullagh and Nelder 1989). And even if the function you want to learn is mildly nonlinear, you may get better generalization with a simple linear model than with a complicated nonlinear model if there is too little data or too much noise to estimate the nonlinearities accurately.

In MLPs with step/threshold/Heaviside activation functions, you need two hidden layers for full generality (Sontag 1992). For further discussion, see Bishop (1995, 121-126).

In MLPs with any of a wide variety of continuous nonlinear hidden-layer activation functions, one hidden layer with an arbitrarily large number of units suffices for the "universal approximation" property (e.g., Hornik, Stinchcombe and White 1989; Hornik 1993; for more references, see Bishop 1995, 130, and Ripley, 1996, 173-180). But there is no theory yet to tell you how many hidden units are needed to approximate any given function.

If you have only one input, there seems to be no advantage to using more than one hidden layer. But things get much more complicated when there are two or more inputs. To illustrate, examples with two inputs and one output will be used so that the results can be shown graphically. In each example there are 441 training cases on a regular 21-by-21 grid. The test sets have 1681 cases on a regular 41-by-41 grid over the same domain as the training set. If you are reading the HTML version of this document via a web browser, you can see surface plots based on the test set by clicking on the file names mentioned in the folowing text. Each plot is a gif file, approximately 9K in size.

Consider a target function of two inputs, consisting of a Gaussian hill in the middle of a plane (hill.gif). An MLP with an identity output activation function can easily fit the hill by surrounding it with a few sigmoid (logistic, tanh, arctan, etc.) hidden units, but there will be spurious ridges and valleys where the plane should be flat (h_mlp_6.gif). It takes dozens of hidden units to flatten out the plane accurately (h_mlp_30.gif).

Now suppose you use a logistic output activation function. As the input to a logistic function goes to negative infinity, the output approaches zero. The plane in the Gaussian target function also has a value of zero. If the weights and bias for the output layer yield large negative values outside the base of the hill, the logistic function will flatten out any spurious ridges and valleys. So fitting the flat part of the target function is easy (h_mlpt_3_unsq.gif and h_mlpt_3.gif). But the logistic function also tends to lower the top of the hill.

If instead of a rounded hill, the target function was a mesa with a large, flat top with a value of one, the logistic output activation function would be able to smooth out the top of the mesa just like it smooths out the plane below. Target functions like this, with large flat areas with values of either zero or one, are just what you have in many noise-free classificaton problems. In such cases, a single hidden layer is likely to work well.

When using a logistic output activation function, it is common practice to scale the target values to a range of .1 to .9. Such scaling is bad in a noise-free classificaton problem, because it prevents the logistic function from smoothing out the flat areas (h_mlpt1-9_3.gif).

For the Gaussian target function, [.1,.9] scaling would make it easier to fit the top of the hill, but would reintroduce undulations in the plane. It would be better for the Gaussian target function to scale the target values to a range of 0 to .9. But for a more realistic and complicated target function, how would you know the best way to scale the target values?

By introducing a second hidden layer with one sigmoid activation function and returning to an identity output activation function, you can let the net figure out the best scaling (h_mlp1_3.gif). Actually, the bias and weight for the output layer scale the output rather than the target values, and you can use whatever range of target values is convenient.

For more complicated target functions, especially those with several hills or valleys, it is useful to have several units in the second hidden layer. Each unit in the second hidden layer enables the net to fit a separate hill or valley. So an MLP with two hidden layers can often yield an accurate approximation with fewer weights than an MLP with one hidden layer. (Chester 1990).

To illustrate the use of multiple units in the second hidden layer, the next example resembles a landscape with a Gaussian hill and a Gaussian valley, both elliptical (hillanvale.gif). The table below gives the RMSE (root mean squared error) for the test set with various architectures. If you are reading the HTML version of this document via a web browser, click on any number in the table to see a surface plot of the corresponding network output.

The MLP networks in the table have one or two hidden layers with a tanh activation function. The output activation function is the identity. Using a squashing function on the output layer is of no benefit for this function, since the only flat area in the function has a target value near the middle of the target range.

Hill and Valley Data: RMSE for the Test Set
(Number of weights in parentheses)
MLP Networks

HUs in First Layer	HUs in Second Layer				
	0	1	2	3	4
1	0.204(5)	0.204(7)	0.189(10)	0.187(13)	0.185(16)
2	0.183(9)	0.163(11)	0.147(15)	0.094(19)	0.096(23)
3	0.159(13)	0.095(15)	0.054(20)	0.033(25)	0.045(30)
4	0.137(17)	0.093(19)	0.009(25)	0.021(31)	0.016(37)
5	0.121(21)	0.092(23)		0.010(37)	0.011(44)
6	0.100(25)	0.092(27)		0.007(43)	0.005(51)
7	0.086(29)	0.077(31)			
8	0.079(33)	0.062(35)			
9	0.072(37)	0.055(39)			
10	0.059(41)	0.047(43)			
12	0.047(49)	0.042(51)			
15	0.039(61)	0.032(63)			
20	0.025(81)	0.018(83)			
25	0.021(101)	0.016(103)			
30	0.018(121)	0.015(123)			
40	0.012(161)	0.015(163)			
50	0.008(201)	0.014(203)			

For an MLP with only one hidden layer (column 0), Gaussian hills and valleys require a large number of hidden units to approximate well. When there is one unit in the second hidden layer, the network can represent one hill or valley easily, which is what happens with three to six units in the first hidden layer. But having only one unit in the second hidden layer is of little benefit for learning two hills or valleys. Using two units in the second hidden layer enables the network to approximate two hills or valleys easily; in this example, only four units are required in the first hidden layer to get an excellent fit. Each additional unit in the second hidden layer enables the network to learn another hill or valley with a relatively small number of units in the first hidden layer, as explained by Chester (1990). In this example, having three or four units in the second hidden layer helps little, and actually produces a worse approximation when there are four units in the first hidden layer due to problems with local minima.

Unfortunately, using two hidden layers exacerbates the problem of local minima, and it is important to use lots of random initializations or other methods for global optimization. Local minima with two hidden layers can have extreme spikes or blades even when the number of weights is much smaller than the number of training cases. One of the few advantages of standard backprop is that it is so slow that spikes and blades will not become very sharp for practical training times.

More than two hidden layers can be useful in certain architectures such as cascade correlation (Fahlman and Lebiere 1990) and in special applications, such as the two-spirals problem (Lang and Witbrock 1988) and ZIP code recognition (Le Cun et al. 1989).

RBF networks are most often used with a single hidden layer. But an extra, linear hidden layer before the radial hidden layer enables the network to ignore irrelevant inputs (see How do MLPs compare with RBFs?") The linear hidden layer allows the RBFs to take elliptical, rather than radial (circular), shapes in the space of the inputs. Hence the linear layer gives you an elliptical basis function (EBF) network. In the hill and valley example, an ORBFUN network requires nine hidden units (37 weights) to get the test RMSE below .01, but by

adding a linear hidden layer, you can get an essentially perfect fit with three linear units followed by two radial units (20 weights).

References:

Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press.

Chester, D.L. (1990), "Why Two Hidden Layers are Better than One," IJCNN-90-WASH-DC, Lawrence Erlbaum, 1990, volume 1, 265-268.

Fahlman, S.E. and Lebiere, C. (1990), "The Cascade Correlation Learning Architecture," NIPS2, 524-532, ftp://archive.cis.ohio-state.edu/pub/neuroprose/fahlman.cascor-tr.ps.Z.

Hornik, K., Stinchcombe, M. and White, H. (1989), "Multilayer feedforward networks are universal approximators," Neural Networks, 2, 359-366.

Hornik, K. (1993), "Some new results on neural network approximation," Neural Networks, 6, 1069-1072.

Lang, K.J. and Witbrock, M.J. (1988), "Learning to tell two spirals apart," in Touretzky, D., Hinton, G., and Sejnowski, T., eds., *Procedings of the 1988 Connectionist Models Summer School*, San Mateo, CA: Morgan Kaufmann.

Le Cun, Y., Boser, B., Denker, J.s., Henderson, D., Howard, R.E., Hubbard, W., and Jackel, L.D. (1989), "Backpropagation applied to handwritten ZIP code recognition", Neural Computation, 1, 541-551.

McCullagh, P. and Nelder, J.A. (1989) *Generalized Linear Models*, 2nd ed., London: Chapman & Hall.

Ripley, B.D. (1996) *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.

Sontag, E.D. (1992), "Feedback stabilization using two-hidden-layer nets", IEEE Transactions on Neural Networks, 3, 981-990.

Subject: How many hidden units should I use?

The best number of hidden units depends in a complex way on:

- the numbers of input and output units
- the number of training cases
- the amount of noise in the targets
- the complexity of the function or classification to be learned
- the architecture
- the type of hidden unit activation function
- the training algorithm
- regularization

In most situations, there is no way to determine the best number of hidden units without training several networks and estimating the generalization error of each. If you have too few hidden units, you will get high training error and high generalization error due to underfitting and high statistical bias. If you have too many hidden units, you may get low training error but still have high generalization error due to overfitting and high variance. Geman, Bienenstock, and Doursat (1992) discuss how the number of hidden units affects the bias/variance trade-off.

Some books and articles offer "rules of thumb" for choosing an architecture; for example:

- "A rule of thumb is for the size of this [hidden] layer to be somewhere between the input layer size ... and the output layer size ..." (Blum, 1992, p. 60).
- "To calculate the number of hidden nodes we use a general rule of: (Number of inputs + outputs) * (2/3)" (from the FAQ for a commercial neural network software company).
- "you will never require more than twice the number of hidden units as you have inputs" in an MLP with one hidden layer (Swingler, 1996, p. 53). See the section in Part 4 of the FAQ on The Worst books for the source of this myth.)
- "How large should the hidden layer be? One rule of thumb is that it should never be more than twice as large as the input layer." (Berry and Linoff, 1997, p. 323).
- "Typically, we specify as many hidden nodes as dimensions [principal components] needed to capture 70-90% of the variance of the input data set." (Boger and Guterman, 1997)

These rules of thumb are nonsense because they ignore the number of training cases, the amount of noise in the targets, and the complexity of the function. Even if you restrict consideration to minimizing training error on data with lots of training cases and no noise, it is easy to construct counterexamples that disprove these rules of thumb. For example:

- There are 100 Boolean inputs and 100 Boolean targets. Each target is a conjunction of some subset of inputs. No hidden units are needed.

- There are two continuous inputs X and Y which take values uniformly distributed on a square [0,8] by [0,8]. Think of the input space as a chessboard, and number the squares 1 to 64. The categorical target variable C is the square number, so there are 64 output units. For example, you could generate the data as follows (this is the SAS programming language, but it should be easy to translate into any other language):

```
data chess;
   step = 1/4;
   do x = step/2 to 8-step/2 by step;
      do y = step/2 to 8-step/2 by step;
         c = 8*floor(x) + floor(y) + 1;
         output;
      end;
   end;
run;
```

  No hidden units are needed.

- The classic two-spirals problem has two continuous inputs and a Boolean classification target. The data can be generated as follows:

```
data spirals;
   pi = arcos(-1);
   do i = 0 to 96;
      angle = i*pi/16.0;
      radius = 6.5*(104-i)/104;
      x = radius*cos(angle);
      y = radius*sin(angle);
      c = 1;
      output;
      x = -x;
      y = -y;
      c = 0;
      output;
   end;
run;
```

With one hidden layer, about 50 tanh hidden units are needed. Many NN programs may actually need closer to 100 hidden units to get zero training error.

- There is one continuous input X that takes values on [0,100]. There is one continuous target Y = sin(X). Getting a good approximation to Y requires about 20 to 25 tanh hidden units. Of course, 1 sine hidden unit would do the job.

Some rules of thumb relate the total number of trainable weights in the network to the number of training cases. A typical recommendation is that the number of weights should be no more than 1/30 of the number of training cases. Such rules are only concerned with overfitting and are at best crude approximations. Also, these rules do not apply when regularization is used. It is true that without regularization, if the number of training cases is *much* larger (but no one knows exactly *how* much larger) than the number of weights, you are unlikely to get overfitting, but you may suffer from underfitting. For a noise-free quantitative target variable, twice as many training cases as weights may be more than enough to avoid overfitting. For a very noisy categorical target variable, 30 times as many training cases as weights may not be enough to avoid overfitting.

An intelligent choice of the number of hidden units depends on whether you are using early stopping or some other form of regularization. If not, you must simply try many networks with different numbers of hidden units, estimate the generalization error for each one, and choose the network with the minimum estimated generalization error. For examples using statistical criteria to choose the number of hidden units, see ftp://ftp.sas.com/pub/neural/dojo/dojo.html.

Using conventional optimization algorithms (see "What are conjugate gradients, Levenberg-Marquardt, etc.?"), there is little point in trying a network with more weights than training cases, since such a large network is likely to overfit.

Using standard online backprop, however, Lawrence, Giles, and Tsoi (1996, 1997) have shown that it can be difficult to reduce training error to a level near the globally optimal value, even when using more weights than training cases. But increasing the number of weights makes it easier for standard backprop to find a good local optimum, so using "oversize" networks can reduce both training error and generalization error.

If you are using early stopping, it is essential to use *lots* of hidden units to avoid bad local optima (Sarle 1995).

There seems to be no upper limit on the number of hidden units, other than that imposed by computer time and memory requirements. Weigend (1994) makes this assertion, but provides only one example as evidence. Tetko, Livingstone, and Luik (1995) provide simulation studies that are more convincing. Similar results were obtained in conjunction with the simulations in Sarle (1995), but those results are not reported in the paper for lack of space. On the other hand, there seems to be no advantage to using more hidden units than you have training cases, since bad local minima do not occur with so many hidden units.

If you are using weight decay or Bayesian estimation, you can also use lots of hidden units (Neal 1996). However, it is not strictly necessary to do so, because other methods are available to avoid local minima, such as multiple random starts and simulated annealing (such methods are not safe to use with early stopping). You can use one network with lots of hidden units, or you can try different networks with different numbers of hidden units, and choose on the basis of estimated generalization error. With weight decay or MAP Bayesian estimation, it is prudent to keep the number of weights less than half the number of training cases.

Bear in mind that with two or more inputs, an MLP with one hidden layer containing only a few units can fit only a limited variety of target functions. Even simple, smooth surfaces such as a Gaussian bump in two dimensions may require 20 to 50 hidden units for a close approximation. Networks with a smaller number of hidden units often produce spurious ridges and valleys in the output surface (see Chester 1990 and "How do MLPs compare with RBFs?") Training a network with 20 hidden units will typically require anywhere from 150 to 2500 training cases if you do not use early stopping or regularization. Hence, if you have a smaller training set than that, it is usually advisable to use early stopping or regularization rather than to restrict the net to a small number of hidden units.

Ordinary RBF networks containing only a few hidden units also produce peculiar, bumpy output functions. Normalized RBF networks are better at approximating simple smooth surfaces with a small number of hidden units (see How do MLPs compare with RBFs?).

There are various theoretical results on how fast approximation error decreases as the number of hidden units increases, but the conclusions are quite sensitive to the assumptions regarding the function you are trying to approximate. See p. 178 in Ripley (1996) for a summary. According to a well-known result by Barron (1993), in a network with `I` inputs and `H` units in a single hidden layer, the root integrated squared error (RISE) will decrease at least as fast as `H^(-1/2)` under some quite complicated smoothness assumptions. Ripley cites another paper by DeVore et al. (1989) that says if the function has only `R` bounded derivatives, RISE may decrease as slowly as `H^(-R/I).` For some examples with from 1 to 4 hidden layers see How many hidden layers should I use?" and "How do MLPs compare with RBFs?"

For learning a finite training set exactly, bounds for the number of hidden units required are provided by Elisseeff and Paugam-Moisy (1997).

References:

Barron, A.R. (1993), "Universal approximation bounds for superpositions of a sigmoid function," IEEE Transactions on Information Theory, 39, 930-945.

Berry, M.J.A., and Linoff, G. (1997), *Data Mining Techniques*, NY: John Wiley & Sons.

Blum, A. (1992), *Neural Networks in C++*, NY: Wiley.

Boger, Z., and Guterman, H. (1997), "Knowledge extraction from artificial neural network models," IEEE Systems, Man, and Cybernetics Conference, Orlando, FL.

Chester, D.L. (1990), "Why Two Hidden Layers are Better than One," IJCNN-90-WASH-DC, Lawrence Erlbaum, 1990, volume 1, 265-268.

DeVore, R.A., Howard, R., and Micchelli, C.A. (1989), "Optimal nonlinear approximation," Manuscripta Mathematica, 63, 469-478.

Elisseeff, A., and Paugam-Moisy, H. (1997), "Size of multilayer networks for exact learning: analytic approach," in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) *Advances in Neural Information Processing Systems 9*, Cambrideg, MA: The MIT Press, pp.162-168.

Geman, S., Bienenstock, E. and Doursat, R. (1992), "Neural Networks and the Bias/Variance Dilemma", Neural Computation, 4, 1-58.

Lawrence, S., Giles, C.L., and Tsoi, A.C. (1996), "What size neural network gives optimal generalization? Convergence properties of backpropagation," Technical Report UMIACS-TR-96-22 and CS-TR-3617, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, http://www.neci.nj.nec.com/homepages/lawrence/papers/minima-tr96/minima-tr96.html

Lawrence, S., Giles, C.L., and Tsoi, A.C. (1997), "Lessons in Neural Network Training: Overfitting May be Harder than Expected," Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI-97, AAAI Press, Menlo Park, California, pp. 540-545, http://www.neci.nj.nec.com/homepages/lawrence/papers/overfitting-aaai97/overfitting-aaai97-bib.html

Neal, R. M. (1996) *Bayesian Learning for Neural Networks*, New York: Springer-Verlag, ISBN 0-387-94724-8.

Ripley, B.D. (1996) *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press,

Sarle, W.S. (1995), "Stopped Training and Other Remedies for Overfitting," *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, 352-360, ftp://ftp.sas.com/pub/neural/inter95.ps.Z (this is a *very large* compressed postscript file, *747K*, 10 pages)

Swingler, K. (1996), *Applying Neural Networks: A Practical Guide*, London: Academic Press.

Tetko, I.V., Livingstone, D.J., and Luik, A.I. (1995), "Neural Network Studies. 1. Comparison of Overfitting and Overtraining," J. Chem. Info. Comp. Sci., 35, 826-833.

Weigend, A. (1994), "On overfitting and the effective number of hidden units," *Proceedings of the 1993 Connectionist Models Summer School*, 335-342.

- -

Subject: How can generalization error be estimated?

There are many methods for estimating generalization error.

Single-sample statistics: AIC, SBC, MDL, FPE, Mallows' C_p, etc.

In linear models, statistical theory provides several simple estimators of the generalization error under various sampling assumptions (Darlington 1968; Efron and Tibshirani 1993; Miller 1990). These estimators adjust the training error for the number of weights being estimated, and in some cases for the noise variance if that is known.

These statistics can also be used as crude estimates of the generalization error in nonlinear models when you have a "large" training set. Correcting these statistics for nonlinearity requires substantially more computation (Moody 1992), and the theory does not always hold for neural networks due to violations of the regularity conditions.

Among the simple generalization estimators that do not require the noise variance to be known, Schwarz's Bayesian Criterion (known as both SBC and BIC; Schwarz 1978; Judge et al. 1980; Raftery 1995) often works well for NNs (Sarle 1995, 1999). AIC and FPE tend to overfit with NNs. Rissanen's Minimum Description Length principle (MDL; Rissanen 1978, 1987, 1999) is closely related to SBC. A special issue of *Computer Journal* contains several articles on MDL, which can be found online at http://www3.oup.co.uk/computer_journal/hdb/Volume_42/Issue_04/
Several articles on SBC/BIC are available at the University of Washigton's web site at http://www.stat.washington.edu/tech.reports

For classification problems, the formulas are not as simple as for regression with normal noise. See Efron (1986) regarding logistic regression.

Split-sample or hold-out validation.

The most commonly used method for estimating generalization error in neural networks is to reserve part of the data as a "test" set, which must not be used in *any* way during training. The test set must be a representative sample of the cases that you want to generalize to. After training, run the network on the test set, and the error on the test set provides an unbiased estimate of the generalization error, provided that the test set was chosen randomly. The disadvantage of split-sample validation is that it reduces the amount of data available for both training and validation. See Weiss and Kulikowski (1991).

Cross-validation (e.g., leave one out).

Cross-validation is an improvement on split-sample validation that allows you to use all of the data for training. The disadvantage of cross-validation is that you have to retrain the net many times. See "What are cross-validation and bootstrapping?".

Bootstrapping.

Bootstrapping is an improvement on cross-validation that often provides better estimates of generalization error at the cost of even more computing time. See "What are cross-validation and bootstrapping?".

If you use any of the above methods to choose which of several different networks to use for prediction purposes, the estimate of the generalization error of the best network will be optimistic. For example, if you train several networks using one data set, and use a second (validation set) data set to decide which network is best, you must use a third (test set) data set to obtain an unbiased estimate of the generalization error of the chosen network. Hjorth (1994) explains how this principle extends to cross-validation and bootstrapping.

References:

Darlington, R.B. (1968), "Multiple Regression in Psychological Research and Practice," Psychological Bulletin, 69, 161-182.

Efron, B. (1986), "How biased is the apparent error rate of a prediction rule?" J. of the American Statistical Association, 81, 461-470.

Efron, B. and Tibshirani, R.J. (1993), *An Introduction to the Bootstrap*, London: Chapman & Hall.

Hjorth, J.S.U. (1994), *Computer Intensive Statistical Methods: Validation, Model Selection, and Bootstrap*, London: Chapman & Hall.

Miller, A.J. (1990), *Subset Selection in Regression*, London: Chapman & Hall.

Moody, J.E. (1992), "The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems", in Moody, J.E., Hanson, S.J., and Lippmann, R.P., *Advances in Neural Information Processing Systems 4*, 847-854.

Raftery, A.E. (1995), "Bayesian Model Selection in Social Research," in Marsden, P.V. (ed.), *Sociological Methodology 1995*, Cambridge, MA: Blackwell, ftp://ftp.stat.washington.edu/pub/tech.reports/bic.ps.z or http://www.stat.washington.edu/tech.reports/bic.ps

Rissanen, J. (1978), "Modelling by shortest data description," Automatica, 14, 465-471.

Rissanen, J. (1987), "Stochastic complexity" (with discussion), J. of the Royal Statistical Society, Series B, 49, 223-239.

Rissanen, J. (1999), "Hypothesis Selection and Testing by the MDL Principle," Computer Journal, 42, 260-269, http://www3.oup.co.uk/computer_journal/hdb/Volume_42/Issue_04/

Sarle, W.S. (1995), "Stopped Training and Other Remedies for Overfitting," *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, 352-360, ftp://ftp.sas.com/pub/neural/inter95.ps.Z (this is a *very large* compressed postscript file, *747K*, 10 pages)

Sarle, W.S. (1999), "Donoho-Johnstone Benchmarks: Neural Net Results," ftp://ftp.sas.com/pub/neural/dojo/dojo.html

Weiss, S.M. & Kulikowski, C.A. (1991), *Computer Systems That Learn*, Morgan Kaufmann.

Subject: What are cross-validation and bootstrapping?

Cross-validation and bootstrapping are both methods for estimating generalization error based on "resampling" (Weiss and Kulikowski 1991; Efron and Tibshirani 1993; Hjorth 1994; Plutowski, Sakata, and White 1994;

Shao and Tu 1995). The resulting estimates of generalization error are often used for choosing among various models, such as different network architectures.

Cross-validation

In k-fold cross-validation, you divide the data into k subsets of (approximately) equal size. You train the net k times, each time leaving out one of the subsets from training, but using only the omitted subset to compute whatever error criterion interests you. If k equals the sample size, this is called "leave-one-out" cross-validation. "Leave-v-out" is a more elaborate and expensive version of cross-validation that involves leaving out all possible subsets of v cases.

Note that cross-validation is quite different from the "split-sample" or "hold-out" method that is commonly used for early stopping in NNs. In the split-sample method, only a single subset (the validation set) is used to estimate the generalization error, instead of k different subsets; i.e., there is no "crossing". While various people have suggested that cross-validation be applied to early stopping, the proper way of doing so is not obvious.

The distinction between cross-validation and split-sample validation is extremely important because cross-validation is markedly superior for small data sets; this fact is demonstrated dramatically by Goutte (1997) in a reply to Zhu and Rohwer (1996). For an insightful discussion of the limitations of cross-validatory choice among several learning methods, see Stone (1977).

Jackknifing

Leave-one-out cross-validation is also easily confused with jackknifing. Both involve omitting each training case in turn and retraining the network on the remaining subset. But cross-validation is used to estimate generalization error, while the jackknife is used to estimate the bias of a statistic. In the jackknife, you compute some statistic of interest in each subset of the data. The average of these subset statistics is compared with the corresponding statistic computed from the entire sample in order to estimate the bias of the latter. You can also get a jackknife estimate of the standard error of a statistic. Jackknifing can be used to estimate the bias of the training error and hence to estimate the generalization error, but this process is more complicated than leave-one-out cross-validation (Efron, 1982; Ripley, 1996, p. 73).

Choice of cross-validation method

Cross-validation can be used simply to estimate the generalization error of a given model, or it can be used for model selection by choosing one of several models that has the smallest estimated generalization error. For example, you might use cross-validation to choose the number of hidden units, or you could use cross-validation to choose a subset of the inputs (subset selection). A subset that contains all relevant inputs will be called a "good" subsets, while the subset that contains all relevant inputs but no others will be called the "best" subset. Note that subsets are "good" and "best" in an asymptotic sense (as the number of training cases goes to infinity). With a small training set, it is possible that a subset that is smaller than the "best" subset may provide better generalization error.

Leave-one-out cross-validation often works well for estimating generalization error for continuous error functions such as the mean squared error, but it may perform poorly for discontinuous error functions such as the number of misclassified cases. In the latter case, k-fold cross-validation is preferred. But if k gets too small, the error estimate is pessimistically biased because of the difference in training-set size between the full-sample analysis and the cross-validation analyses. (For model-selection purposes, this bias can actually help; see the

discussion below of Shao, 1993.) A value of 10 for k is popular for estimating generalization error.

Leave-one-out cross-validation can also run into trouble with various model-selection methods. Again, one problem is lack of continuity--a small change in the data can cause a large change in the model selected (Breiman, 1996). For choosing subsets of inputs in linear regression, Breiman and Spector (1992) found 10-fold and 5-fold cross-validation to work better than leave-one-out. Kohavi (1995) also obtained good results for 10-fold cross-validation with empirical decision trees (C4.5). Values of k as small as 5 or even 2 may work even better if you analyze several different random k-way splits of the data to reduce the variability of the cross-validation estimate.

Leave-one-out cross-validation also has more subtle deficiencies for model selection. Shao (1995) showed that in linear models, leave-one-out cross-validation is asymptotically equivalent to AIC (and Mallows' C_p), but leave-v-out cross-validation is asymptotically equivalent to Schwarz's Bayesian criterion (called SBC or BIC) when v = n[1-1/(log(n)-1)], where n is the number of training cases. SBC provides consistent subset-selection, while AIC does not. That is, SBC will choose the "best" subset with probability approaching one as the size of the training set goes to infinity. AIC has an asymptotic probability of one of choosing a "good" subset, but less than one of choosing the "best" subset (Stone, 1979). Many simulation studies have also found that AIC overfits badly in small samples, and that SBC works well (e.g., Hurvich and Tsai, 1989; Shao and Tu, 1995). Hence, these results suggest that leave-one-out cross-validation should overfit in small samples, but leave-v-out cross-validation with appropriate v should do better. However, when true models have an infinite number of parameters, SBC is not efficient, and other criteria that are asymptotically efficient but not consistent for model selection may produce better generalization (Hurvich and Tsai, 1989).

Shao (1993) obtained the surprising result that for selecting subsets of inputs in a linear regression, the probability of selecting the "best" does not converge to 1 (as the sample size n goes to infinity) for leave-v-out cross-validation unless the proportion v/n approaches 1. At first glance, Shao's result seems inconsistent with the analysis by Kearns (1997) of split-sample validation, which shows that the best generalization is obtained with v/n strictly between 0 and 1, with little sensitivity to the precise value of v/n for large data sets. But the apparent conflict is due to the fundamentally different properties of cross-validation and split-sample validation.

To obtain an intuitive understanding of Shao (1993), let's review some background material on generalization error. Generalization error can be broken down into three additive parts, noise variance + estimation variance + squared estimation bias. Noise variance is the same for all subsets of inputs. Bias is nonzero for subsets that are not "good", but it's zero for all "good" subsets, since we are assuming that the function to be learned is linear. Hence the generalization error of "good" subsets will differ only in the estimation variance. The estimation variance is (2p/t)s^2 where p is the number of inputs in the subset, t is the training set size, and s^2 is the noise variance. The "best" subset is better than other "good" subsets only because the "best" subset has (by definition) the smallest value of p. But the t in the denominator means that differences in generalization error among the "good" subsets will all go to zero as t goes to infinity. Therefore it is difficult to guess which subset is "best" based on the generalization error even when t is very large. It is well known that unbiased estimates of the generalization error, such as those based on AIC, FPE, and C_p, do not produce consistent estimates of the "best" subset (e.g., see Stone, 1979).

In leave-v-out cross-validation, t=n-v. The differences of the cross-validation estimates of generalization error among the "good" subsets contain a factor 1/t, not 1/n. Therefore by making t small enough (and thereby making each regression based on t cases bad enough), we can make the differences of the cross-validation estimates large enough to detect. It turns out that to make t small enough to guess the "best" subset consistently, we have to have t/n go to 0 as n goes to infinity.

The crucial distinction between cross-validation and split-sample validation is that with cross-validation, after

guessing the "best" subset, we train the linear regression model for that subset using all n cases, but with split-sample validation, only t cases are ever used for training. If our main purpose were really to choose the "best" subset, I suspect we would still have to have t/n go to 0 even for split-sample validation. But choosing the "best" subset is not the same thing as getting the best generalization. If we are more interested in getting good generalization than in choosing the "best" subset, we do not want to make our regression estimate based on only t cases as bad as we do in cross-validation, because in split-sample validation that bad regression estimate is what we're stuck with. So there is no conflict between Shao and Kearns, but there is a conflict between the two goals of choosing the "best" subset and getting the best generalization in split-sample validation.

Bootstrapping

Bootstrapping seems to work better than cross-validation in many cases (Efron, 1983). In the simplest form of bootstrapping, instead of repeatedly analyzing subsets of the data, you repeatedly analyze subsamples of the data. Each subsample is a random sample with replacement from the full sample. Depending on what you want to do, anywhere from 50 to 2000 subsamples might be used. There are many more sophisticated bootstrap methods that can be used not only for estimating generalization error but also for estimating confidence bounds for network outputs (Efron and Tibshirani 1993). For estimating generalization error in classification problems, the .632+ bootstrap (an improvement on the popular .632 bootstrap) is one of the currently favored methods that has the advantage of performing well even when there is severe overfitting. Use of bootstrapping for NNs is described in Baxt and White (1995), Tibshirani (1996), and Masters (1995). However, the results obtained so far are not very thorough, and it is known that bootstrapping does not work well for some other methodologies such as empirical decision trees (Breiman, Friedman, Olshen, and Stone, 1984; Kohavi, 1995), for which it can be excessively optimistic.

For further information

Cross-validation and bootstrapping become considerably more complicated for time series data; see Hjorth (1994) and Snijders (1988).

More information on jackknife and bootstrap confidence intervals is available at ftp://ftp.sas.com/pub/neural/jackboot.sas (this is a plain-text file).

References:

Baxt, W.G. and White, H. (1995) "Bootstrapping confidence intervals for clinical input variable effects in a network trained to identify the presence of acute myocardial infarction", Neural Computation, 7, 624-638.

Breiman, L. (1996), "Heuristics of instability and stabilization in model selection," Annals of Statistics, 24, 2350-2383.

Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J. (1984), *Classification and Regression Trees*, Belmont, CA: Wadsworth.

Breiman, L., and Spector, P. (1992), "Submodel selection and evaluation in regression: The X-random case," International Statistical Review, 60, 291-319.

Dijkstra, T.K., ed. (1988), *On Model Uncertainty and Its Statistical Implications*, Proceedings of a

workshop held in Groningen, The Netherlands, September 25-26, 1986, Berlin: Springer-Verlag.

Efron, B. (1982) *The Jackknife, the Bootstrap and Other Resampling Plans*, Philadelphia: SIAM.

Efron, B. (1983), "Estimating the error rate of a prediction rule: Improvement on cross-validation," J. of the American Statistical Association, 78, 316-331.

Efron, B. and Tibshirani, R.J. (1993), *An Introduction to the Bootstrap*, London: Chapman & Hall.

Efron, B. and Tibshirani, R.J. (1997), "Improvements on cross-validation: The .632+ bootstrap method," J. of the American Statistical Association, 92, 548-560.

Goutte, C. (1997), "Note on free lunches and cross-validation," Neural Computation, 9, 1211-1215, ftp://eivind.imm.dtu.dk/dist/1997/goutte.nflcv.ps.gz.

Hjorth, J.S.U. (1994), *Computer Intensive Statistical Methods Validation, Model Selection, and Bootstrap*, London: Chapman & Hall.

Hurvich, C.M., and Tsai, C.-L. (1989), "Regression and time series model selection in small samples," Biometrika, 76, 297-307.

Kearns, M. (1997), "A bound on the error of cross validation using the approximation and estimation rates, with consequences for the training-test split," Neural Computation, 9, 1143-1161.

Kohavi, R. (1995), "A study of cross-validation and bootstrap for accuracy estimation and model selection," International Joint Conference on Artificial Intelligence (IJCAI), pp. ?, http://robotics.stanford.edu/users/ronnyk/

Masters, T. (1995) *Advanced Algorithms for Neural Networks: A C++ Sourcebook*, NY: John Wiley and Sons, ISBN 0-471-10588-0

Plutowski, M., Sakata, S., and White, H. (1994), "Cross-validation estimates IMSE," in Cowan, J.D., Tesauro, G., and Alspector, J. (eds.) *Advances in Neural Information Processing Systems 6*, San Mateo, CA: Morgan Kaufman, pp. 391-398.

Ripley, B.D. (1996) *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.

Shao, J. (1993), "Linear model selection by cross-validation," J. of the American Statistical Association, 88, 486-494.

Shao, J. (1995), "An asymptotic theory for linear model selection," Statistica Sinica ?.

Shao, J. and Tu, D. (1995), *The Jackknife and Bootstrap*, New York: Springer-Verlag.

Snijders, T.A.B. (1988), "On cross-validation for predictor evaluation in time series," in Dijkstra (1988), pp. 56-69.

Stone, M. (1977), "Asymptotics for and against cross-validation," Biometrika, 64, 29-35.

Stone, M. (1979), "Comments on model selection criteria of Akaike and Schwarz," J. of the Royal Statistical Society, Series B, 41, 276-278.

Tibshirani, R. (1996), "A comparison of some error estimates for neural network models," Neural Computation, 8, 152-163.

Weiss, S.M. and Kulikowski, C.A. (1991), *Computer Systems That Learn*, Morgan Kaufmann.

Zhu, H., and Rohwer, R. (1996), "No free lunch for cross-validation," Neural Computation, 8, 1421-1426.

Subject: How to compute prediction and confidence intervals (error bars)?

(This answer is only about half finished. I will get around to the other half eventually.)

In addition to estimating over-all generalization error, it is often useful to be able to estimate the accuracy of the network's predictions for individual cases.

Let:

```
Y      = the target variable
y_i    = the value of Y for the ith case
X      = the vector of input variables
x_i    = the value of X for the ith case
N      = the noise in the target variable
n_i    = the value of N for the ith case
m(X)   = E(Y|X) = the conditional mean of Y given X
w      = a vector of weights for a neural network
w^     = the weight obtained via training the network
p(X,w) = the output of a neural network given input X and weights w
p_i    = p(x_i,w)
L      = the number of training (learning) cases, (y_i,x_i), i=1, ..., L
Q(w)   = the objective function
```

Assume the data are generated by the model:

```
Y = m(X) + N
E(N|X) = 0
N and X are independent
```

The network is trained by attempting to minimize the objective function `Q(w),` which, for example, could be the sum of squared errors or the negative log likelihood based on an assumed family of noise distributions.

Given a test input `x_0`, a `100c%` prediction interval for `y_0` is an interval `[LPB_0,UPB_0]` such that

`Pr(LPB_0 <= y_0 <= UPB_0) = c,` where `c` is typically .95 or .99, and the probability is computed over repeated random selection of the training set and repeated observation of `Y` given the test input `x_0.` A `100c%` confidence interval for `p_0` is an interval `[LCB_0,UCB_0]` such that `Pr(LCB_0 <= p_0 <= UCB_0) = c,` where again the probability is computed over repeated random selection of the training set. Note that `p_0` is a nonrandom quantity, since `x_0` is given. A confidence interval is narrower than the corresponding prediction interval, since the prediction interval must include variation due to noise in `y_0`, while the confidence interval does not. Both intervals include variation due to sampling of the training set and possible variation in the training process due, for example, to random initial weights and local minima of the objective function.

Traditional statistical methods for nonlinear models depend on several assumptions (Gallant, 1987):

1. The inputs for the training cases are either fixed or obtained by simple random sampling or some similarly well-behaved process.
2. `Q(w)` has continuous first and second partial derivatives with respect to `w` over some convex, bounded subset `S_W` of the weight space.
3. `Q(w)` has a unique global minimum at `w^,` which is an interior point of `S_W.`
4. The model is well-specified, which requires (a) that there exist weights `w$` in the interior of `S_W` such that `m(x) = p(x,w$),` and (b) that the assumptions about the noise distribution are correct. (Sorry about the `w$` notation, but I'm running out of plain text symbols.)

These traditional methods are based on a linear approximation to `p(x,w)` in a neighborhood of `w$,` yielding a quadratic approximation to `Q(w).` Hence the Hessian of `Q(w)` (the square matrix of second-order partial derivatives with respect to `w`) frequently appears in these methods.

Assumption (3) is not satisfied for neural nets, because networks with hidden units always have multiple global minima, and the global minima are often improper. Hence, confidence intervals for the weights cannot be obtained using standard Hessian-based methods. However, Hwang and Ding (1997) have shown that confidence intervals for predicted values can be obtained because the predicted values are statistically identified even though the weights are not.

Cardell, Joerding, and Li (1994) describe a more serious violation of assumption (3), namely that that for some `m(x)`, no finite global minimum exists. In such situations, it may be possible to use regularization methods such as weight decay to obtain valid confidence intervals (De Veaux, Schumi, Schweinsberg, and Ungar, 1998), but more research is required on this subject, since the derivation in the cited paper assumes a finite global minimum.

For large samples, the sampling variability in `w^` can be approximated in various ways:

- Fisher's information matrix, which is the expected value of the Hessian of `Q(w)` divided by `L,` can be used when `Q(w)` is the negative log likelihood (Spall, 1998).
- The delta method, based on the Hessian of `Q(w)` or the Gauss-Newton approximation using the cross-product Jacobian of `Q(w),` can also be used when `Q(w)` is the negative log likelihood (Tibshirani, 1996; Hwang and Ding, 1997; De Veaux, Schumi, Schweinsberg, and Ungar, 1998).
- The sandwich estimator, a more elaborate Hessian-based method, relaxes assumption (4) (Gallant, 1987; White, 1989; Tibshirani, 1996).
- Bootstrapping can be used without knowing the form of the noise distribution and takes into account variability introduced by local minima in training, but requires training the network many times on different resamples of the training set (Tibshirani, 1996; Heskes 1997).

References:

Cardell, N.S., Joerding, W., and Li, Y. (1994), "Why some feedforward networks cannot learn some polynomials," Neural Computation, 6, 761-766.

De Veaux,R.D., Schumi, J., Schweinsberg, J., and Ungar, L.H. (1998), "Prediction intervals for neural networks via nonlinear regression," Technometrics, 40, 273-282.

Gallant, A.R. (1987) *Nonlinear Statistical Models*, NY: Wiley.

Heskes, T. (1997), "Practical confidence and prediction intervals," in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) *Advances in Neural Information Processing Systems 9*, Cambrideg, MA: The MIT Press, pp. 176-182.

Hwang, J.T.G., and Ding, A.A. (1997), "Prediction intervals for artificial neural networks," J. of the American Statistical Association, 92, 748-757.

Nix, D.A., and Weigend, A.S. (1995), "Learning local error bars for nonlinear regression," in Tesauro, G., Touretzky, D., and Leen, T., (eds.) *Advances in Neural Information Processing Systems 7*, Cambridge, MA: The MIT Press, pp. 489-496.

Spall, J.C. (1998), "Resampling-based calculation of the information matrix in nonlinear statistical models," Proceedings of the 4th Joint Conference on Information Sciences, October 23-28, Research Triangle PArk, NC, USA, Vol 4, pp. 35-39.

Tibshirani, R. (1996), "A comparison of some error estimates for neural network models," Neural Computation, 8, 152-163.

White, H. (1989), "Some Asymptotic Results for Learning in Single Hidden Layer Feedforward Network Models", J. of the American Statistical Assoc., 84, 1008-1013.

- -

Next part is part 4 (of 7). Previous part is part 2.

Archive-name: ai-faq/neural-nets/part4
Last-modified: 2002-06-09
URL: ftp://ftp.sas.com/pub/neural/FAQ4.html
Maintainer: saswss@unx.sas.com (Warren S. Sarle)

This is part 4 (of 7) of a monthly posting to the Usenet newsgroup comp.ai.neural-nets. See the part 1 of this posting for full information what it is all about.

========== Questions ==========

Part 6: Commercial software
Part 7: Hardware and miscellaneous

Subject: Books and articles about Neural Networks?

The following search engines will search many bookstores for new and used books and return information on availability, price, and shipping charges:

AddAll: http://www.addall.com/
Bookfinder: http://www.bookfinder.com/

Clicking on the author and title of most of the books listed in the "Best" and "Notable" sections will do a search using AddAll.

There are many on-line bookstores, such as:

Amazon: http://www.amazon.com/
Amazon, UK: http://www.amazon.co.uk/
Amazon, Germany: http://www.amazon.de/
Barnes & Noble: http://www.bn.com/
Bookpool: http://www.bookpool.com/
Borders: http://www.borders.com/
Fatbrain: http://www.fatbrain.com/

The neural networks reading group at the University of Illinois at Urbana-Champaign, the Artifical Neural Networks and Computational Brain Theory (ANNCBT) forum, has compiled a large number of book and paper reviews at http://anncbt.ai.uiuc.edu/, with an emphasis more on cognitive science rather than practical applications of NNs.

The Best

The best of the best

Bishop (1995) is clearly the single best book on artificial NNs. This book excels in organization and choice of material, and is a close runner-up to Ripley (1996) for accuracy. If you are new to the field, read it from cover to cover. If you have lots of experience with NNs, it's an excellent reference. If you don't know calculus, take a class. I hope a second edition comes out soon! For more information, see The best intermediate textbooks on NNs below.

If you have questions on feedforward nets that aren't answered by Bishop, try Masters (1993) or Reed and Marks (1999) for practical issues or Ripley (1996) for theortical issues, all of which are reviewed below.

The best popular introduction to NNs

Hinton, G.E. (1992), "How Neural Networks Learn from Experience", Scientific American, 267 (September), 144-151.

Author's Webpage: http://www.cs.utoronto.ca/DCS/People/Faculty/hinton.html (official)
and http://www.cs.toronto.edu/~hinton (private)
Journal Webpage: http://www.sciam.com/
Additional Information: Unfortunately that article is not available there.

The best introductory book for business executives

Bigus, J.P. (1996), *Data Mining with Neural Networks: Solving Business Problems--from Application Development to Decision Support,* NY: McGraw-Hill, ISBN 0-07-005779-6, xvii+221 pages.
The stereotypical business executive (SBE) does not want to know how or why NNs work--he (SBEs are usually male) just wants to make money. The SBE may know what an average or percentage is, but he is deathly afraid of "statistics". He understands profit and loss but does not want to waste his time learning things involving complicated math, such as high-school algebra. For further information on the SBE, see the "Dilbert" comic strip.

Bigus has written an excellent introduction to NNs for the SBE. Bigus says (p. xv), "For business executives, managers, or computer professionals, this book provides a thorough introduction to neural network technology and the issues related to its application without getting bogged down in complex math or needless details. The reader will be able to identify common business problems that are amenable to the neural netwrk approach and will be sensitized to the issues that can affect successful completion of such applications." Bigus succeeds in explaining NNs at a practical, intuitive, and necessarily shallow level without formulas--just what the SBE needs. This book is far better than Caudill and Butler (1990), a popular but disastrous attempt to explain NNs without formulas.

Chapter 1 introduces data mining and data warehousing, and sketches some applications thereof. Chapter 2 is the semi-obligatory philosophico-historical discussion of AI and NNs and is well-written, although the SBE in a hurry may want to skip it. Chapter 3 is a very useful discussion of data preparation. Chapter 4 describes a variety of NNs and what they are good for. Chapter 5 goes into practical issues of training and testing NNs. Chapters 6 and 7 explain how to use the results from NNs. Chapter 8 discusses intelligent agents. Chapters 9 through 12 contain case histories of NN applications, including market segmentation, real-estate pricing, customer ranking, and sales forecasting.

Bigus provides generally sound advice. He briefly discusses overfitting and overtraining without going into much detail, although I think his advice on p. 57 to have at least two training cases for each connection is somewhat lenient, even for noise-free data. I do not understand his claim on pp. 73 and 170 that RBF networks have advantages over backprop networks for nonstationary inputs--perhaps he is using the word "nonstationary" in a sense different from the statistical meaning of the term. There are other things in the book that I would quibble with, but I did not find any of the flagrant errors that are common in other books on NN applications such as Swingler (1996).

The one serious drawback of this book is that it is more than one page long and may therefore tax the attention span of the SBE. But any SBE who succeeds in reading the entire book should learn enough to be able to hire a good NN expert to do the real work.

The best elementary textbooks

Fausett, L. (1994), *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications,* Englewood Cliffs, NJ: Prentice Hall, ISBN 0-13-334186-0. Also published as a Prentice Hall International Edition, ISBN 0-13-042250-9. Sample software (source code listings in C and Fortran) is included in an Instructor's Manual.
Book Webpage (Publisher): http://www.prenhall.com/books/esm_0133341860.html
Additional Information: The mentioned programs / additional support is not available. Contents:
Ch. 1 Introduction, 1.1 Why Neural Networks and Why Now?, 1.2 What Is a Neural Net?, 1.3 Where Are Neural Nets Being Used?, 1.4 How Are Neural Networks Used?, 1.5 Who Is Developing Neural Networks?, 1.6 When Neural Nets Began: the McCulloch-Pitts Neuron;
Ch. 2 Simple Neural Nets for Pattern Classification, 2.1 General Discussion, 2.2 Hebb Net, 2.3 Perceptron, 2.4 Adaline;
Ch. 3 Pattern Association, 3.1 Training Algorithms for Pattern Association, 3.2 Heteroassociative Memory Neural Network, 3.3 Autoassociative Net, 3.4 Iterative Autoassociative Net, 3.5 Bidirectional Associative Memory (BAM);

Ch. 4 Neural Networks Based on Competition, 4.1 Fixed-Weight Competitive Nets, 4.2 Kohonen Self-Organizing Maps, 4.3 Learning Vector Quantization, 4.4 Counterpropagation;
Ch. 5 Adaptive Resonance Theory, 5.1 Introduction, 5.2 Art1, 5.3 Art2;
Ch. 6 Backpropagation Neural Net, 6.1 Standard Backpropagation, 6.2 Variations, 6.3 Theoretical Results;
Ch. 7 A Sampler of Other Neural Nets, 7.1 Fixed Weight Nets for Constrained Optimization, 7.2 A Few More Nets that Learn, 7.3 Adaptive Architectures, 7.4 Neocognitron; Glossary.

Review by Ian Cresswell:

> What a relief! As a broad introductory text this is without any doubt the best currently available in its area. It doesn't include source code of any kind (normally this is badly written and compiler specific). The algorithms for many different kinds of simple neural nets are presented in a clear step by step manner in plain English.
>
> Equally, the mathematics is introduced in a relatively gentle manner. There are no unnecessary complications or diversions from the main theme.
>
> The examples that are used to demonstrate the various algorithms are detailed but (perhaps necessarily) simple.
>
> There are bad things that can be said about most books. There are only a small number of minor criticisms that can be made about this one. More space should have been given to backprop and its variants because of the practical importance of such methods. And while the author discusses early stopping in one paragraph, the treatment of generalization is skimpy compared to the books by Weiss and Kulikowski or Smith listed above.
>
> If you're new to neural nets and you don't want to be swamped by bogus ideas, huge amounts of intimidating looking mathematics, a programming language that you don't know etc. etc. then this is the book for you.
>
> In summary, this is the best starting point for the outsider and/or beginner... a truly excellent text.

Smith, M. (1996). *Neural Networks for Statistical Modeling,* NY: Van Nostrand Reinhold, ISBN 0-442-01310-8.
Apparently there is a new edition I haven't seen yet:
Smith, M. (1996). *Neural Networks for Statistical Modeling,* Boston: International Thomson Computer Press, ISBN 1-850-32842-0.
Book Webpage (Publisher): http://www.thompson.com/
Publisher's address: 20 Park Plaza, Suite 1001, Boston, MA 02116, USA.
Smith is not a statistician, but he has made an impressive effort to convey statistical fundamentals applied to neural networks. The book has entire brief chapters on overfitting and validation (early stopping and split-sample validation, which he incorrectly calls cross-validation), putting it a rung above most other introductions to NNs. There are also brief chapters on data preparation and diagnostic plots, topics usually ignored in elementary NN books. Only feedforward nets are covered in any detail.
Chapter headings: Mapping Functions; Basic Concepts; Error Derivatives; Learning Laws; Weight Initialization; The Course of Learning: An Example; Overfitting; Cross Validation; Preparing the Data; Representing Variables; Using the Model.

Weiss, S.M. and Kulikowski, C.A. (1991), *Computer Systems That Learn,* Morgan Kaufmann. ISBN 1-55860-065-5.
Author's Webpage: Kulikowski: http://ruccs.rutgers.edu/faculty/kulikowski.html
Book Webpage (Publisher): http://www.mkp.com/books_catalog/1-55860-065-5.asp
Additional Information: Information of Weiss, S.M. are not available.
Briefly covers at a very elementary level feedforward nets, linear and nearest-neighbor discriminant analysis, trees, and expert sytems, emphasizing practical applications. For a book at this level, it has an unusually good chapter on estimating generalization error, including bootstrapping.

```
1 Overview of Learning Systems
    1.1 What is a Learning System?
```

Reed, R.D., and Marks, R.J, II (1999), *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks,* Cambridge, MA: The MIT Press, ISBN 0-262-18190-8.
Author's Webpage: Marks: http://cialab.ee.washington.edu/Marks.html
Book Webpage (Publisher): http://mitpress.mit.edu/book-home.tcl?isbn=0262181908

After you have read Smith (1996) or Weiss and Kulikowski (1991), consult Reed and Marks for practical details on training MLPs (other types of neural nets such as RBF networks are barely even mentioned). They provide extensive coverage of backprop and its variants, and they also survey conventional optimization algorithms. Their coverage of initialization methods, constructive networks, pruning, and regularization methods is unusually thorough. Unlike the vast majority of books on neural nets, this one has lots of really informative graphs. The chapter on generalization assessment is slightly weak, which is why you should read Smith (1996) or Weiss and Kulikowski (1991) first. Also, there is little information on data preparation, for which Smith (1996) and Masters (1993; see below) should be consulted. There is some elementary calculus, but not enough that it should scare off anybody. Many second-rate books treat neural nets as mysterious black boxes, but Reed and Marks open up the box and provide genuine insight into the way neural nets work.

One problem with the book is that the terms "validation set" and "test set" are used inconsistently.

Chapter headings: Supervised Learning; Single-Layer Networks; MLP Representational Capabilities; Back-Propagation; Learning Rate and Momentum; Weight-Initialization Techniques; The Error Surface; Faster Variations of Back-Propagation; Classical Optimization Techniques; Genetic Algorithms and Neural Networks; Constructive Methods; Pruning Algorithms; Factors Influencing Generalization; Generalization Prediction and Assessment; Heuristics for Improving Generalization; Effects of Training with Noisy Inputs; Linear Regression; Principal Components Analysis; Jitter Calculations; Sigmoid-like Nonlinear Functions

The best books on using and programming NNs

Masters, T. (1993), *Practical Neural Network Recipes in C++,* Academic Press, ISBN 0-12-479040-2, US $45 incl. disks.
Book Webpage (Publisher): http://www.apcatalog.com/cgi-bin/AP?ISBN=0124790402&LOCATION=US&FORM=FORM2
Masters has written three exceptionally good books on NNs (the two others are listed below). He combines generally sound practical advice with some basic statistical knowledge to produce a programming text that is far superior to the competition (see "The Worst" below). Not everyone likes his C++ code (the usual complaint is that the code is not sufficiently OO) but, unlike the code in some other books, Masters's code has been successfully compiled and run by some readers of comp.ai.neural-nets. Masters's books are well worth reading even for people who have no interest in programming.
Chapter headings: Foundations; Classification; Autoassociation; Time-Series Prediction; Function Approximation; Multilayer Feedforward Networks; Eluding Local Minima I: Simulated Annealing; Eluding Local Minima II: Genetic Optimization; Regression and Neural Networks; Designing Feedforward Network Architectures; Interpreting Weights: How Does This Thing Work; Probabilistic Neural Networks; Functional Link Networks; Hybrid Networks; Designing the Training Set; Preparing Input Data; Fuzzy Data and Processing; Unsupervised Training; Evaluating Performance of Neural Networks; Confidence Measures; Optimizing the Decision Threshold; Using the NEURAL Program.

Masters, T. (1995) *Advanced Algorithms for Neural Networks: A C++ Sourcebook,* NY: John Wiley and Sons, ISBN 0-471-10588-0
Book Webpage (Publisher): http://www.wiley.com/
Additional Information: One has to search.
Clear explanations of conjugate gradient and Levenberg-Marquardt optimization algorithms, simulated annealing, kernel regression (GRNN) and discriminant analysis (PNN), Gram-Charlier networks, dimensionality reduction, cross-validation, and bootstrapping.

Masters, T. (1994), *Signal and Image Processing with Neural Networks: A C++ Sourcebook,* NY: Wiley, ISBN 0-471-04963-8.
Book Webpage (Publisher): http://www.wiley.com/
Additional Information: One has to search.

The best intermediate textbooks on NNs

Bishop, C.M. (1995). *Neural Networks for Pattern Recognition,* Oxford: Oxford University Press. ISBN 0-19-853849-9 (hardback) or 0-19-853864-2 (paperback), xvii+482 pages.
Author's Webpage: http://neural-server.aston.ac.uk/People/bishopc/Welcome.html
Book Webpage (Publisher):
http://www1.oup.co.uk/bin/readcat?Version=887069107&title=Neural+Networks+for+Pattern+Recognition&TOB=52305&H1=19808&H2=47489&H3=48287&H4=48306&count=1&style=full

This is definitely the best book on feedforward neural nets for readers comfortable with calculus. The book is exceptionally well organized, presenting topics in a logical progression ideal for conceptual understanding.

Geoffrey Hinton writes in the foreword:
"Bishop is a leading researcher who has a deep understanding of the material and has gone to great lengths to organize it in a sequence that makes sense. He has wisely avoided the temptation to try to cover everything and has therefore omitted interesting topics like reinforcement learning, Hopfield networks, and Boltzmann machines in order to focus on the types of neural networks that are most widely used in practical applications. He assumes that the reader has the basic mathematical literacy required for an undergraduate science degree, and using these tools he explains everything from scratch. Before introducing the multilayer perceptron, for example, he lays a solid foundation of basic statistical concepts. So the crucial concept of overfitting is introduced using easily visualized examples of one-dimensional polynomials and only later applied to neural networks. An impressive aspect of this book is that it takes the reader all the way from the simplest linear models to the very latest Bayesian multilayer neural networks without ever requiring any great intellectual leaps."

Chapter headings: Statistical Pattern Recognition; Probability Density Estimation; Single-Layer Networks; The Multi-layer Perceptron; Radial Basis Functions; Error Functions; Parameter Optimization Algorithms; Pre-processing and Feature Extraction; Learning and Generalization; Bayesian Techniques; Symmetric Matrices; Gaussian Integrals; Lagrange Multipliers; Calculus of Variations; Principal Components.

Hertz, J., Krogh, A., and Palmer, R. (1991). *Introduction to the Theory of Neural Computation.* Redwood City, CA: Addison-Wesley, ISBN 0-201-50395-6 (hardbound) and 0-201-51560-1 (paperbound)
Book Webpage (Publisher): http://www2.awl.com/gb/abp/sfi/computer.html
This is an excellent classic work on neural nets from the perspective of physics covering a wide variety of networks. Comments from readers of comp.ai.neural-nets: "My first impression is that this one is by far the best book on the topic. And it's below $30 for the paperback."; "Well written, theoretical (but not overwhelming)"; It provides a good balance of model development, computational algorithms, and applications. The mathematical derivations are especially well done"; "Nice mathematical analysis on the mechanism of different learning algorithms"; "It is NOT for mathematical beginner. If you don't have a good grasp of higher level math, this book can be really tough to get through."

The best advanced textbook covering NNs

Ripley, B.D. (1996) *Pattern Recognition and Neural Networks,* Cambridge: Cambridge University Press, ISBN 0-521-46086-7 (hardback), xii+403 pages.
Author's Webpage: http://www.stats.ox.ac.uk/~ripley/
Book Webpage (Publisher): http://www.cup.cam.ac.uk/
Additional Information: The Webpage includes errata and additional information, which hasn't been available at publishing time, for this book.
Brian Ripley's book is an excellent sequel to Bishop (1995). Ripley starts up where Bishop left off, with Bayesian inference and statistical decision theory, and then covers some of the same material on NNs as Bishop but at a higher mathematical level. Ripley also covers a variety of methods that are not discussed, or discussed only briefly, by Bishop, such as tree-based methods and belief networks. While Ripley is best appreciated by people with a background in mathematical statistics, the numerous realistic examples in his book will be of interest even to beginners in neural nets.
Chapter headings: Introduction and Examples; Statistical Decision Theory; Linear Discriminant Analysis; Flexible Discriminants; Feed-forward Neural Networks; Non-parametric Methods; Tree-structured Classifiers; Belief Networks; Unsupervised Methods; Finding Good Pattern Features; Statistical Sidelines.

Devroye, L., Györfi, L., and Lugosi, G. (1996), *A Probabilistic Theory of Pattern Recognition,* NY: Springer, ISBN 0-387-94618-7, vii+636 pages.
This book has relatively little material explicitly about neural nets, but what it has is very interesting and much of it is not found in other texts. The emphasis is on statistical proofs of universal consistency for a wide variety of methods, including histograms, (k) nearest neighbors, kernels (PNN), trees, generalized linear discriminants, MLPs, and RBF networks. There is also considerable material on validation and cross-validation. The authors say, "We did not scar the pages with backbreaking simulations or quick-and-dirty engineering solutions" (p. 7). The formula-to-text ratio is high, but the writing is quite clear, and anyone who has had a year or two of mathematical statistics should be able to follow the exposition.
Chapter headings: The Bayes Error; Inequalities and Alternate Distance Measures; Linear Discrimination; Nearest Neighbor Rules; Consistency; Slow Rates of Convergence; Error Estimation; The Regular Histogram Rule; Kernel Rules; Consistency of the k-Nearest Neighbor Rule; Vapnik-Chervonenkis Theory; Combinatorial Aspects of Vapnik-Chervonenkis Theory; Lower Bounds for Empirical Classifier Selection; The Maximum Likelihood Principle; Parametric Classification; Generalized Linear Discrimination; Complexity Regularization; Condensed and Edited Nearest

Neighbor Rules; Tree Classifiers; Data-Dependent Partitioning; Splitting the Data; The Resubstitution Estimate; Deleted Estimates of the Error Probability; Automatic Kernel Rules; Automatic Nearest Neighbor Rules; Hypercubes and Discrete Spaces; Epsilon Entropy and Totally Bounded Sets; Uniform Laws of Large Numbers; Neural Networks; Other Error Estimates; Feature Extraction.

The best books on neurofuzzy systems

Brown, M., and Harris, C. (1994), *Neurofuzzy Adaptive Modelling and Control*, NY: Prentice Hall, ISBN 0-13-134453-6.
Author's Webpage: http://www.isis.ecs.soton.ac.uk/people/m_brown.html
and http://www.ecs.soton.ac.uk/~cjh/
Book Webpage (Publisher): http://www.prenhall.com/books/esm_0131344536.html
Additional Information: Additional page at: http://www.isis.ecs.soton.ac.uk/publications/neural/mqbcjh94e.html and an abstract can be found at:
http://www.isis.ecs.soton.ac.uk/publications/neural/mqb93.html
Brown and Harris rely on the fundamental insight that that a fuzzy system is a nonlinear mapping from an input space to an output space that can be parameterized in various ways and therefore can be adapted to data using the usual neural training methods (see "What is backprop?") or conventional numerical optimization algorithms (see "What are conjugate gradients, Levenberg-Marquardt, etc.?"). Their approach makes clear the intimate connections between fuzzy systems, neural networks, and statistical methods such as B-spline regression.

The best comparison of NNs with other classification methods

Michie, D., Spiegelhalter, D.J. and Taylor, C.C. (1994), *Machine Learning, Neural and Statistical Classification*, Ellis Horwood. Author's Webpage: Donald Michie: http://www.aiai.ed.ac.uk/~dm/dm.html
Additional Information: This book is out of print but available online at http://www.amsta.leeds.ac.uk/~charles/statlog/

Other notable books

Introductory

Anderson, J.A. (1995), *An Introduction to Neural Networks*, Cambridge,MA: The MIT Press, ISBN 0-262-01144-1.
Author's Webpage: http://www.cog.brown.edu/~anderson
Book Webpage (Publisher): http://mitpress.mit.edu/book-home.tcl?isbn=0262510812 or
http://mitpress.mit.edu/book-home.tcl?isbn=0262011441 (hardback)
Additional Information: Programs and additional information can be found at: ftp://mitpress.mit.edu/pub/Intro-to-NeuralNets/
Anderson provides an accessible introduction to the AI and neurophysiological sides of NN research, although the book is weak regarding practical aspects of using NNs.
Chapter headings: Properties of Single Neurons; Synaptic Integration and Neuron Models; Essential Vector Operations; Lateral Inhibition and Sensory Processing; Simple Matrix Operations; The Linear Associator: Background and Foundations; The Linear Associator: Simulations; Early Network Models: The Perceptron; Gradient Descent Algorithms; Representation of Information; Applications of Simple Associators: Concept Formation and Object Motion; Energy and Neural Networks: Hopfield Networks and Boltzmann Machines; Nearest Neighbor Models; Adaptive Maps; The BSB Model: A Simple Nonlinear Autoassociative Neural Network; Associative Computation; Teaching Arithmetic to a Neural Network.

Hagan, M.T., Demuth, H.B., and Beale, M. (1996), *Neural Network Design*, Boston: PWS, ISBN 0-534-94332-2.
It doesn't really say much about design, but this book provides formulas and examples in excruciating detail for a wide variety of networks. It also includes some mathematical background material.
Chapter headings: Neuron Model and Network Architectures; An Illustrative Example; Perceptron Learning Rule; Signal and Weight Vector Spaces; Linear Transformations for Neural; Networks; Supervised Hebbian Learning; Performance Surfaces and Optimum Points; Performance Optimization; Widrow-Hoff Learning; Backpropagation; Variations on Backpropagation;

Associative Learning; Competitive Networks; Grossberg Network; Adaptive Resonance Theory; Stability; Hopfield Network.

Abdi, H., Valentin, D., and Edelman, B. (1999), *Neural Networks,* Sage University Papers Series on Quantitative Applications in the Social Sciences, 07-124, Thousand Oaks, CA: Sage, ISBN 0-7619-1440-4.
Inexpensive, brief (89 pages) but very detailed explanations of linear networks and the basics of backpropagation.
Chapter headings: 1. Introduction 2. The Perceptron 3. Linear Autoassociative Memories 4. Linear Heteroassociative Memories 5. Error Backpropagation 6. Useful References.

Bayesian learning

Neal, R. M. (1996) *Bayesian Learning for Neural Networks,* New York: Springer-Verlag, ISBN 0-387-94724-8.

Biological learning and neurophysiology

Koch, C., and Segev, I., eds. (1998) *Methods in Neuronal Modeling: From Ions to Networks,* 2nd ed., Cambridge, MA: The MIT Press, ISBN 0-262-11231-0.
Book Webpage: http://goethe.klab.caltech.edu/MNM/

Rolls, E.T., and Treves, A. (1997), *Neural Networks and Brain Function,* Oxford: Oxford University Press, ISBN: 0198524323.
Chapter headings: Introduction; Pattern association memory; Autoassociation memory; Competitive networks, including self-organizing maps; Error-correcting networks: perceptrons, the delta rule, backpropagation of error in multilayer networks, and reinforcement learning algorithms; The hippocampus and memory; Pattern association in the brain: amygdala and orbitofrontal cortex; Cortical networks for invariant pattern recognition; Motor systems: cerebellum and basal ganglia; Cerebral neocortex.

Schmajuk, N.A. (1996) *Animal Learning and Cognition: A Neural Network Approach,* Cambridge: Cambridge University Press, ISBN 0521456967.
Chapter headings: Neural networks and associative learning Classical conditioning: data and theories; Cognitive mapping; Attentional processes; Storage and retrieval processes; Configural processes; Timing; Operant conditioning and animal communication: data, theories, and networks; Animal cognition: data and theories; Place learning and spatial navigation; Maze learning and cognitive mapping; Learning, cognition, and the hippocampus: data and theories; Hippocampal modulation of learning and cognition; The character of the psychological law.

Collections

Orr, G.B., and Mueller, K.-R., eds. (1998), *Neural Networks: Tricks of the Trade,* Berlin: Springer, ISBN 3-540-65311-2.
Articles: Efficient BackProp; Early Stopping - But When? A Simple Trick for Estimating the Weight Decay Parameter; Controling the Hyperparameter Search in MacKay's Bayesian Neural Network Framework; Adaptive Regularization in Neural Network Modeling; Large Ensemble Averaging; Square Unit Augmented, Radially Extended, Multilayer Perceptrons; A Dozen Tricks with Multitask Learning; Solving the Ill-Conditioning in Neural Network Learning; Centering Neural Network Gradient Factors; Avoiding Roundoff Error in Backpropagating Derivatives; Transformation Invariance in Pattern Recognition - Tangent Distance and Tangent Propagation; Combining Neural Networks and Context-Driven Search for On-Line, Printed Handwriting Recognition in the Newton; Neural Network Classification and Prior Class Probabilities; Applying Divide and Conquer to Large Scale Pattern Recognition Tasks; Forecasting the Economy with Neural Nets: A Survey of Challenges and Solutions; How to Train Neural Networks.

Arbib, M.A., ed. (1995), *The Handbook of Brain Theory and Neural Networks,* Cambridge, MA: The MIT Press, ISBN 0-262-51102-9.
From The Publisher: The heart of the book, part III, comprises of 267 original articles by leaders in the various fields, arranged alphabetically by title. Parts I and II, written by the editor, are designed to help readers orient themselves to this vast range of material. Part I, Background, introduces several basic neural models, explains how the present study of brain theory and neural networks integrates brain theory, artificial intelligence, and cognitive psychology, and provides a tutorial on the concepts essential for understanding neural networks as dynamic, adaptive systems. Part II, Road Maps, provides entry into the many articles of part III through an introductory "Meta-Map" and twenty-three road maps, each of which tours all the Part III articles on the chosen theme.

Touretzky, D., Hinton, G, and Sejnowski, T., eds., (1989) *Proceedings of the 1988 Connectionist Models Summer School,* San Mateo, CA: Morgan Kaufmann, ISBN: 1558600337

NIPS:

1. Touretzky, D.S., ed. (1989), *Advances in Neural Information Processing Systems 1,* San Mateo, CA: Morgan Kaufmann, ISBN: 1558600159
2. Touretzky, D. S., ed. (1990), *Advances in Neural Information Processing Systems 2,* San Mateo, CA: Morgan Kaufmann, ISBN: 1558601007
3. Lippmann, R.P., Moody, J.E., and Touretzky, D. S., eds. (1991) *Advances in Neural Information Processing Systems 3,* San Mateo, CA: Morgan Kaufmann, ISBN: 1558601848
4. Moody, J.E., Hanson, S.J., and Lippmann, R.P., eds. (1992) *Advances in Neural Information Processing Systems 4,* San Mateo, CA: Morgan Kaufmann, ISBN: 1558602224
5. Hanson, S.J., Cowan, J.D., and Giles, C.L. eds. (1993) *Advances in Neural Information Processing Systems 5,* San Mateo, CA: Morgan Kaufmann, ISBN: 1558602747
6. Cowan, J.D., Tesauro, G., and Alspector, J., eds. (1994) *Advances in Neural Information Processing Systems 6,* San Mateo, CA: Morgan Kaufman, ISBN: 1558603220
7. Tesauro, G., Touretzky, D., and Leen, T., eds. (1995) *Advances in Neural Information Processing Systems 7,* Cambridge, MA: The MIT Press, ISBN: 0262201046
8. Touretzky, D. S., Mozer, M.C., and Hasselmo, M.E., eds. (1996) *Advances in Neural Information Processing Systems 8,* Cambridge, MA: The MIT Press, ISBN: 0262201070
9. Mozer, M.C., Jordan, M.I., and Petsche, T., eds. (1997) *Advances in Neural Information Processing Systems 9,* Cambridge, MA: The MIT Press, ISBN: 0262100657
10. Jordan, M.I., Kearns, M.S., and Solla, S.A., eds. (1998) *Advances in Neural Information Processing Systems 10,* Cambridge, MA: The MIT Press, ISBN: 0262100762
11. Kearns, M.S., Solla, S.A., amd Cohn, D.A., eds. (1999) *Advances in Neural Information Processing Systems 11,* Cambridge,MA: The MIT Press, ISBN: 0262112450
12. Solla, S.A., Leen, T., and Müller, K.-R., eds. (2000) *Advances in Neural Information Processing Systems 12,* Cambridge, MA: The MIT Press, ISBN: 0-262-19450-3

Combining networks

Sharkey, A.J.C. (1999), *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems,* London: Springer, ISBN: 185233004X

Connectionism

Elman, J.L., Bates, E.A., Johnson, M.H., Karmiloff-Smith, A., and Parisi, D. (1996) *Rethinking Innateness: A Connectionist Perspective on Development,* Cambridge, MA: The MIT Press, ISBN: 026255030X.
Chapter headings: New perspectives on development; Why connectionism? Ontogenetic development: A connectionist synthesis; The shape of change; Brain development; Interactions, all the way down; Rethinking innateness.

Plunkett, K., and Elman, J.L. (1997), *Exercises in Rethinking Innateness: A Handbook for Connectionist Simulations,* Cambridge, MA: The MIT Press, ISBN: 0262661055.
Chapter headings: Introduction and overview; The methodology of simulations; Learning to use the simulator; Learning internal representations; Autoassociation; Generalization; Translation invariance; Simple recurrent networks; Critical points in learning; Modeling stages in cognitive development; Learning the English past tense; The importance of starting small.

Feedforward networks

Fine, T.L. (1999) *Feedforward Neural Network Methodology,* NY: Springer, ISBN 0-387-98745-2.

Husmeier, D. (1999), *Neural Networks for Conditional Probability Estimation: Forecasting Beyond Point Predictions,* Berlin: Springer Verlag, ISBN 185233095.

Fuzzy logic and neurofuzzy systems

See also "General (including SVMs and Fuzzy Logic)".

Kosko, B. (1997), *Fuzzy Engineering*, Upper Saddle River, NJ: Prentice Hall, ISBN 0-13-124991-6.
Kosko's new book is a big improvement over his older neurofuzzy book and makes an excellent sequel to Brown and Harris (1994).

Nauck, D., Klawonn, F., and Kruse, R. (1997), *Foundations of Neuro-Fuzzy Systems*, Chichester: Wiley, ISBN 0-471-97151-0.
Chapter headings: Historical and Biological Aspects; Neural Networks; Fuzzy Systems; Modelling Neuro-Fuzzy Systems; Cooperative Neuro-Fuzzy Systems; Hybrid Neuro-Fuzzy Systems; The Generic Fuzzy Perceptron; NEFCON - Neuro-Fuzzy Control; NEFCLASS - Neuro-Fuzzy Classification; NEFPROX - Neuro-Fuzzy Function Approximation; Neural Networks and Fuzzy Prolog; Using Neuro-Fuzzy Systems.

General (including SVMs and Fuzzy Logic)

Many books on neural networks, machine learning, etc., present various methods as miscellaneous tools without any conceptual framework relating different methods. The best of such neural net "cookbooks" is probably Haykin's (1999) second edition.

Among conceptually-integrated books, there are two excellent books that use the Vapnil-Chervonenkis theory as a unifying theme, and provide strong coverage of support vector machines and fuzzy logic, as well as neural nets. Of these two, Kecman (2001) provides clearer explanations and better diagrams, but Cherkassky and Mulier (1998) are better organized have an excellent section on unsupervised learning, especially self-organizing maps. I have been tempted to add both of these books to the "best" list, but I have not done so because I think VC theory is of doubtful practical utility for neural nets. However, if you are especially interested in VC theory and support vector machines, then both of these books can be highly recommended. To help you choose between them, a detailed table of contents is provided below for each book.

Haykin, S. (1999), *Neural Networks: A Comprehensive Foundation*, 2nd ed., Upper Saddle River, NJ: Prentice Hall, ISBN 0-13-273350-1.
The second edition is much better than the first, which has been described as a core-dump of Haykin's brain. The second edition covers more topics, is easier to understand, and has better examples.
Chapter headings: Introduction; Learning Processes; Single Layer Perceptrons; Multilayer Perceptrons; Radial-Basis Function Networks; Support Vector Machines; Committee Machines; Principal Components Analysis; Self-Organizing Maps; Information-Theoretic Models; Stochastic Machines And Their Approximates Rooted in Statistical Mechanics; Neurodynamic Programming; Temporal Processing Using Feedforward Networks; Neurodynamics; Dynamically Driven Recurrent Networks.

Kecman, V. (2001), *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models*, Cambridge, MA: The MIT Press; ISBN: 0-262-11255-8.
URL: http://www.support-vector.ws/

```
Detailed Table of Contents:

1. Learning and Soft Computing: Rationale, Motivations, Needs, Basics
   1.1 Examples of Applications in Diverse Fields
   1.2 Basic Tools of Soft Computing: Neural Networks, Fuzzy Logic Systems, and
Support Vector Machines
       1.2.1 Basics of Neural Networks
       1.2.2 Basics of Fuzzy Logic Modeling
   1.3 Basic Mathematics of Soft Computing
       1.3.1 Approximation of Multivariate Functions
       1.3.2 Nonlinear Error Surface and Optimization
   1.4 Learning and Statistical Approaches to Regression and Classification
       1.4.1 Regression
       1.4.2 Classification
   Problems
```

Cherkassky, V.S., and Mulier, F.M. (1998), *Learning from Data : Concepts, Theory, and Methods*, NY: John Wiley & Sons; ISBN: 0-471-15493-8.

Detailed Table of Contents:

History

Hebb, D.O. (1949), *The Organization of Behavior*, NY: Wiley. Out of print.

Rosenblatt, F. (1962), *Principles of Neurodynamics*, NY: Spartan Books. Out of print.

Anderson, J.A., and Rosenfeld, E., eds. (1988), *Neurocomputing: Foundatons of Research*, Cambridge, MA: The MIT Press, ISBN 0-262-01097-6.
Author's Webpage: http://www.cog.brown.edu/~anderson
Book Webpage (Publisher): http://mitpress.mit.edu/book-home.tcl?isbn=0262510480
43 articles of historical importance, ranging from William James to Rumelhart, Hinton, and Williams.

Anderson, J. A., Pellionisz, A. and Rosenfeld, E. (Eds). (1990). *Neurocomputing 2: Directions for Research.* The MIT Press: Cambridge, MA.
Author's Webpage: http://www.cog.brown.edu/~anderson

Book Webpage (Publisher): http://mitpress.mit.edu/book-home.tcl?isbn=0262510758

Carpenter, G.A., and Grossberg, S., eds. (1991), *Pattern Recognition by Self-Organizing Neural Networks,* Cambridge, MA: The MIT Press, ISBN 0-262-03176-0
Articles on ART, BAM, SOMs, counterpropagation, etc.

Nilsson, N.J. (1965/1990), *Learning Machines,* San Mateo, CA: Morgan Kaufmann, ISBN 1-55860-123-6.

Minsky, M.L., and Papert, S.A. (1969/1988) *Perceptrons,* Cambridge, MA: The MIT Press, 1st ed. 1969, expanded edition 1988 ISBN 0-262-63111-3.

Werbos, P.J. (1994), *The Roots of Backpropagation,* NY: John Wiley & Sons, ISBN: 0471598976. Includes Werbos's 1974 Harvard Ph.D. thesis, *Beyond Regression.*

Kohonen, T. (1984/1989), *Self-organization and Associative Memory*, 1st ed. 1988, 3rd ed. 1989, NY: Springer.
Author's Webpage: http://www.cis.hut.fi/nnrc/teuvo.html
Book Webpage (Publisher): http://www.springer.de/
Additional Information: Book is out of print.

Rumelhart, D. E. and McClelland, J. L. (1986), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* Volumes 1 & 2, Cambridge, MA: The MIT Press ISBN 0-262-63112-1.
Author's Webpage: http://www-med.stanford.edu/school/Neurosciences/faculty/rumelhart.html
Book Webpage (Publisher): http://mitpress.mit.edu/book-home.tcl?isbn=0262631121

Hecht-Nielsen, R. (1990), *Neurocomputing,* Reading, MA: Addison-Wesley, ISBN 0-201-09355-3.
Book Webpage (Publisher): http://www.awl.com/

Anderson, J.A., and Rosenfeld, E., eds. (1998), *Talking Nets: An Oral History of Neural Networks,* Cambridge, MA: The MIT Press, ISBN 0-262-51111-8.

Knowledge, rules, and expert systems

Gallant, S.I. (1995), *Neural Network Learning and Expert Systems,* Cambridge, MA: The MIT Press, ISBN 0-262-07145-2.
Chapter headings:; Introduction and Important Definitions; Representation Issues; Perceptron Learning and the Pocket Algorithm; Winner-Take-All Groups or Linear Machines; Autoassociators and One-Shot Learning; Mean Squared Error (MSE) Algorithms; Unsupervised Learning; The Distributed Method and Radial Basis Functions; Computational Learning Theory and the BRD Algorithm; Constructive Algorithms; Backpropagation; Backpropagation: Variations and Applications; Simulated Annealing and Boltzmann Machines; Expert Systems and Neural Networks; Details of the MACIE System; Noise, Redundancy, Fault Detection, and Bayesian Decision Theory; Extracting Rules from Networks; Appendix: Representation Comparisons.

Cloete, I., and Zurada, J.M. (2000), *Knowledge-Based Neurocomputing,* Cambridge, MA: The MIT Press, ISBN 0-262-03274-0.
Articles: Knowledge-Based Neurocomputing: Past, Present, and Future; Architectures and Techniques for Knowledge-Based Neurocomputing; Symbolic Knowledge Representation in Recurrent Neural Networks: Insights from Theoretical Models of Computation; A Tutorial on Neurocomputing of Structures; Structural Learning and Rule Discovery; VL[subscript 1]ANN: Transformation of Rules to Artificial Neural Networks; Integrations of Heterogeneous Sources of Partial Domain Knowledge; Approximation of Differential Equations Using Neural Networks; Fynesse: A Hybrid Architecture for Self-Learning Control; Data Mining Techniques for Designing Neural Network Time Series Predictors; Extraction of Decision Trees from Artificial Neural Networks 369; Extraction of Linguistic Rules from Data via Neural Networks and Fuzzy Approximation; Neural Knowledge Processing in Expert Systems.

Learning theory

Wolpert, D.H., ed. (1995) *The Mathematics of Generalization: The Proceedings of the SFI/CNLS Workshop on Formal Approaches to Supervised Learning,* Santa Fe Institute Studies in the Sciences of Complexity, Volume XX, Reading, MA: Addison-Wesley, ISBN: 0201409836.
Articles: The Status of Supervised Learning Science circa 1994 - The Search for a Consensus; Reflections After Refereeing Papers for NIPS; The Probably Approximately Correct (PAC) and Other Learning Models; Decision Theoretic Generalizations of the PAC Model for Neural Net and Other Learning Applications; The Relationship Between PAC, the Statistical Physics Framework, the Bayesian Framework, and the VC Framework; Statistical Physics Models of Supervised Learning; On Exhaustive Learning; A Study of Maximal-Coverage Learning Algorithms; On Bayesian Model Selection; Soft Classification, a.k.a. Risk Estimation, via Penalized Log Likelihood and Smoothing Spline Analysis of Variance; Current Research; Preface to Simplifying Neural Networks by Soft Weight Sharing; Simplifying Neural Networks by Soft Weight Sharing; Error-Correcting Output Codes: A General Method for Improving Multiclass Inductive Learning Programs; Image Segmentation and Recognition.

Anthony, M., and Bartlett, P.L. (1999), *Neural Network Learning: Theoretical Foundations,* Cambridge: Cambridge University Press, ISBN 0-521-57353-X.

Vapnik, V.N. (1998) *Statistical Learning Theory,* NY: Wiley, ISBN: 0471030031
This book is much better than Vapnik's *The Nature of Statistical Learning Theory*.
Chapter headings:
0. Introduction: The Problem of Induction and Statistical Inference;
1. Two Approaches to the Learning Problem;
Appendix: Methods for Solving Ill-Posed Problems;
2. Estimation of the Probability Measure and Problem of Learning;
3. Conditions for Consistency of Empirical Risk Minimization Principle;
4. Bounds on the Risk for Indicator Loss Functions;
Appendix: Lower Bounds on the Risk of the ERM Principle;
5. Bounds on the Risk for Real-Valued Loss Functions;
6. The Structural Risk Minimization Principle;
Appendix: Estimating Functions on the Basis of Indirect Measurements;
7. Stochastic Ill-Posed Problems;
8. Estimating the Values of Functions at Given Points;
9. Perceptrons and Their Generalizations;
10. The Support Vector Method for Estimating Indicator Functions;
11. The Support Vector Method for Estimating Real-Valued Functions;
12. SV Machines for Pattern Recognition; (includes examples of digit recognition)
13. SV Machines for Function Approximations, Regression Estimation, and Signal Processing; (includes an example of positron emission tomography)
14. Necessary and Sufficient Conditions for Uniform Convergence of Frequencies to Their Probabilities;
15. Necessary and Sufficient Conditions for Uniform Convergence of Means to Their Expectations;
16. Necessary and Sufficient Conditions for Uniform One-Sided Convergence of Means to Their Expectations;
Comments and Bibliographical Remarks.

Object oriented programming

The FAQ maintainer is an old-fashioned C programmer and has no expertise in object oriented programming, so he must rely on the readers of comp.ai.neural-nets regarding the merits of books on OOP for NNs.

There are many excellent books about NNs by Timothy Masters (listed elsewhere in the FAQ) that provide C++ code for NNs. If you simply want code that works, these books should satisfy your needs. If you want code that exemplifies the highest standards of object oriented design, you will be disappointed by Masters.

The one book on OOP for NNs that seems to be consistently praised is:

Rogers, Joey (1996), *Object-Oriented Neural Networks in C++*, Academic Press, ISBN 0125931158.
Contents:
1. Introduction
2. Object-Oriented Programming Review
3. Neural-Network Base Classes
4. ADALINE Network
5. Backpropagation Neural Network
6. Self-Organizing Neural Network
7. Bidirectional Associative Memory
Appendix A Support Classes
Appendix B Listings
References and Suggested Reading

However, you will learn very little about NNs other than elementary programming techniques from Rogers. To quote a customer review at the Barnes & Noble web site (http://www.bn.com):

> A reviewer, a scientific programmer, July 19, 2000, **** Long explaination of neural net code - not of neural nets Good OO code for simple 'off the shelf' implementation, very open & fairly extensible for further cusomization. A complete & lucid explanation of the code but pretty weak on the principles, theory, and application of neural networks. Great as a code source, disappointing as a neural network tutorial.

On-line and incremental learning

Saad, D., ed. (1998), *On-Line Learning in Neural Networks*, Cambridge: Cambridge University Press, ISBN 0-521-65263-4.
Articles: Introduction; On-line Learning and Stochastic Approximations; Exact and Perturbation Solutions for the Ensemble Dynamics; A Statistical Study of On-line Learning; On-line Learning in Switching and Drifting Environments with Application to Blind Source Separation; Parameter Adaptation in Stochastic Optimization; Optimal On-line Learning in Multilayer Neural Networks; Universal Asymptotics in Committee Machines with Tree Architecture; Incorporating Curvature Information into On-line Learning; Annealed On-line Learning in Multilayer Neural Networks; On-line Learning of Prototypes and Principal Components; On-line Learning with Time-Correlated Examples; On-line Learning from Finite Training Sets; Dynamics of Supervised Learning with Restricted Training Sets; On-line Learning of a Decision Boundary with and without Queries; A Bayesian Approach to On-line Learning; Optimal Perceptron Learning: an On-line Bayesian Approach.

Optimization

Cichocki, A. and Unbehauen, R. (1993). *Neural Networks for Optimization and Signal Processing.* NY: John Wiley & Sons, ISBN 0-471-93010-5 (hardbound), 526 pages, $57.95.
Book Webpage (Publisher): http://www.wiley.com/
Additional Information: One has to search.
Chapter headings: Mathematical Preliminaries of Neurocomputing; Architectures and Electronic Implementation of Neural Network Models; Unconstrained Optimization and Learning Algorithms; Neural Networks for Linear, Quadratic Programming and Linear Complementarity Problems; A Neural Network Approach to the On-Line Solution of a System of Linear Algebraic; Equations and Related Problems; Neural Networks for Matrix Algebra Problems; Neural Networks for Continuous, Nonlinear, Constrained Optimization Problems; Neural Networks for Estimation, Identification and Prediction; Neural Networks for Discrete and Combinatorial Optimization Problems.

Pulsed/Spiking networks

Maass, W., and Bishop, C.M., eds. (1999) *Pulsed Neural Networks,* Cambridge, MA: The MIT Press, ISBN: 0262133504.

Articles: Spiking Neurons; Computing with Spiking Neurons; Pulse-Based Computation in VLSI Neural Networks; Encoding Information in Neuronal Activity; Building Silicon Nervous Systems with Dendritic Tree Neuromorphs; A Pulse-Coded Communications Infrastructure; Analog VLSI Pulsed Networks for Perceptive Processing; Preprocessing for Pulsed Neural VLSI Systems; Digital Simulation of Spiking Neural Networks; Populations of Spiking Neurons; Collective Excitation Phenomena and Their Applications; Computing and Learning with Dynamic Synapses; Stochastic Bit-Stream Neural Networks; Hebbian Learning of Pulse Timing in the Barn Owl Auditory System.

Recurrent

Medsker, L.R., and Jain, L.C., eds. (2000), *Recurrent Neural Networks: Design and Applications,* Boca Raton, FL: CRC Press, ISBN 0-8493-7181-3

Articles:
Introduction;
Recurrent Neural Networks for Optimization: The State of the Art;
Efficient Second-Order Learning Algorithms for Discrete-Time Recurrent Neural Networks;
Designing High Order Recurrent Networks for Bayesian Belief Revision;
Equivalence in Knowledge Representation: Automata, Recurrent Neural Networks, and Dynamical Fuzzy Systems;
Learning Long-Term Dependencies in NARX Recurrent Neural Networks;
Oscillation Responses in a Chaotic Recurrent Network;
Lessons from Language Learning;
Recurrent Autoassociative Networks: Developing Distributed Representations of Hierarchically Structured Sequences by Autoassociation;
Comparison of Recurrent Neural Networks for Trajectory Generation;
Training Algorithms for Recurrent Neural Nets that Eliminate the Need for Computation of Error Gradients with Application to Trajectory Production Problem;
Training Recurrent Neural Networks for Filtering and Control;
Remembering How to Behave: Recurrent Neural Networks for Adaptive Robot Behavior

Reinforcement learning

Sutton, R.S., and Barto, A.G. (1998), *Reinforcement Learning: An Introduction,* The MIT Press, ISBN: 0-262193-98-1.

Author's Webpage: http://envy.cs.umass.edu/~rich/sutton.html and http://www-anw.cs.umass.edu/People/barto/barto.html

Book Webpage (Publisher):http://mitpress.mit.edu/book-home.tcl?isbn=0262193981

Additional Information: http://www-anw.cs.umass.edu/~rich/book/the-book.html

Chapter headings: The Problem; Introduction; Evaluative Feedback; The Reinforcement Learning Problem; Elementary Solution Methods; Dynamic Programming; Monte Carlo Methods; Temporal-Difference Learning; A Unified View; Eligibility Traces; Generalization and Function Approximation; Planning and Learning; Dimensions of Reinforcement Learning; Case Studies.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996), *Neuro-Dynamic Programming,* Belmont, MA: Athena Scientific, ISBN 1-886529-10-8.

Author's Webpage: http://www.mit.edu:8001/people/dimitrib/home.html and http://web.mit.edu/jnt/www/home.html

Book Webpage (Publisher):http://world.std.com/~athenasc/ndpbook.html

Speech recognition

Bourlard, H.A., and Morgan, N. (1994), *Connectionist Speech Recognition: A Hybrid Approach,* Boston: Kluwer Academic Publishers, ISBN: 0792393961.

From The Publisher: Describes the theory and implementation of a method to incorporate neural network approaches into state-of-the-art continuous speech recognition systems based on Hidden Markov Models (HMMs) to improve their performance. In this framework, neural networks (and in particular, multilayer perceptrons or MLPs) have been restricted to well-defined subtasks of the whole system, i.e., HMM emission probability estimation and feature extraction. The book describes a successful five year international collaboration between the authors. The lessons learned

form a case study that demonstrates how hybrid systems can be developed to combine neural networks with more traditional statistical approaches. The book illustrates both the advantages and limitations of neural networks in the framework of a statistical system. Using standard databases and comparing with some conventional approaches, it is shown that MLP probability estimation can improve recognition performance. Other approaches are discussed, though there is no such unequivocal experimental result for these methods. Connectionist Speech Recognition: A Hybrid Approach is of use to anyone intending to use neural networks for speech recognition or within the framework provided by an existing successful statistical approach. This includes research and development groups working in the field of speech recognition, both with standard and neural network approaches, as well as other pattern recognition and/or neural network researchers. This book is also suitable as a text for advanced courses on neural networks or speech processing.

Statistics

Cherkassky, V., Friedman, J.H., and Wechsler, H., eds. (1991) *From Statistics to Neural Networks: Theory and Pattern Recognition Applications,* NY: Springer, ISBN 0-387-58199-5.

Kay, J.W., and Titterington, D.M. (1999) *Statistics and Neural Networks: Advances at the Interface,* Oxford: Oxford University Press, ISBN 0-19-852422-6.
Articles: Flexible Discriminant and Mixture Models; Neural Networks for Unsupervised Learning Based on Information Theory; Radial Basis Function Networks and Statistics; Robust Prediction in Many-parameter Models; Density Networks; Latent Variable Models and Data Visualisation; Analysis of Latent Structure Models with Multidimensional Latent Variables; Artificial Neural Networks and Multivariate Statistics.

White, H. (1992b), *Artificial Neural Networks: Approximation and Learning Theory,* Blackwell, ISBN: 1557863296.
Articles: There Exists a Neural Network That Does Not Make Avoidable Mistakes; Multilayer Feedforward Networks Are Universal Approximators; Universal Approximation Using Feedforward Networks with Non-sigmoid Hidden Layer Activation Functions; Approximating and Learning Unknown Mappings Using Multilayer Feedforward Networks with Bounded Weights; Universal Approximation of an Unknown Mapping and Its Derivatives; Neural Network Learning and Statistics; Learning in Artificial Neural Networks: a Statistical Perspective; Some Asymptotic Results for Learning in Single Hidden Layer Feedforward Networks; Connectionist Nonparametric Regression: Multilayer Feedforward Networks Can Learn Arbitrary Mappings; Nonparametric Estimation of Conditional Quantiles Using Neural Networks; On Learning the Derivatives of an Unknown Mapping with Multilayer Feedforward Networks; Consequences and Detection of Misspecified Nonlinear Regression Models; Maximum Likelihood Estimation of Misspecified Models; Some Results for Sieve Estimation with Dependent Observations.

Time-series forecasting

Weigend, A.S. and Gershenfeld, N.A., eds. (1994) *Time Series Prediction: Forecasting the Future and Understanding the Past,* Reading, MA: Addison-Wesley, ISBN 0201626020. Book Webpage (Publisher): http://www2.awl.com/gb/abp/sfi/complexity.html

Unsupervised learning

Kohonen, T. (1995/1997), *Self-Organizing Maps*, 1st ed. 1995, 2nd ed. 1997, Berlin: Springer-Verlag, ISBN 3540620176.

Deco, G. and Obradovic, D. (1996), *An Information-Theoretic Approach to Neural Computing,* NY: Springer-Verlag, ISBN 0-387-94666-7.

Diamantaras, K.I., and Kung, S.Y. (1996) *Principal Component Neural Networks: Theory and Applications,* NY: Wiley, ISBN 0-471-05436-4.

Van Hulle, M.M. (2000), *Faithful Representations and Topographic Maps: From Distortion- to Information-Based Self-Organization,* NY: Wiley, ISBN 0-471-34507-5.

Books for the Beginner

Caudill, M. and Butler, C. (1990). *Naturally Intelligent Systems.* MIT Press: Cambridge, Massachusetts. (ISBN 0-262-03156-6).

Book Webpage (Publisher): http://mitpress.mit.edu/book-home.tcl?isbn=0262531135

The authors try to translate mathematical formulas into English. The results are likely to disturb people who appreciate either mathematics or English. Have the authors never heard that "a picture is worth a thousand words"? What few diagrams they have (such as the one on p. 74) tend to be confusing. Their jargon is peculiar even by NN standards; for example, they refer to target values as "mentor inputs" (p. 66). The authors do not understand elementary properties of error functions and optimization algorithms. For example, in their discussion of the delta rule, the authors seem oblivious to the differences between batch and on-line training, and they attribute magical properties to the algorithm (p. 71):

> [The on-line delta] rule always takes the most efficient route from the current position of the weight vector to the "ideal" position, based on the current input pattern. The delta rule not only minimizes the mean squared error, it does so in the most efficient fashion possible--quite an achievement for such a simple rule.

While the authors realize that backpropagation networks can suffer from local minima, they mistakenly think that counterpropagation has some kind of global optimization ability (p. 202):

> Unlike the backpropagation network, a counterpropagation network cannot be fooled into finding a local minimum solution. This means that the network is guaranteed to find the correct response (or the nearest stored response) to an input, no matter what.

But even though they acknowledge the problem of local minima, the authors are ignorant of the importance of initial weight values (p. 186):

> To teach our imaginary network something using backpropagation, we must start by setting all the adaptive weights on all the neurodes in it to random values. It won't matter what those values are, as long as they are not all the same and not equal to 1.

Like most introductory books, this one neglects the difficulties of getting good generalization--the authors simply declare (p. 8) that "A neural network is able to generalize"!

Chester, M. (1993). *Neural Networks: A Tutorial*, Englewood Cliffs, NJ: PTR Prentice Hall.
Book Webpage (Publisher): http://www.prenhall.com/
Additional Information: Seems to be out of print.
Shallow, sometimes confused, especially with regard to Kohonen networks.

Dayhoff, J. E. (1990). *Neural Network Architectures: An Introduction.* Van Nostrand Reinhold: New York.
Comments from readers of comp.ai.neural-nets: "Like Wasserman's book, Dayhoff's book is also very easy to understand".

Freeman, James (1994). *Simulating Neural Networks with Mathematica*, Addison-Wesley, ISBN: 0-201-56629-X. Book Webpage (Publisher): http://cseng.aw.com/bookdetail.qry?ISBN=0-201-56629-X&ptype=0
Additional Information: Sourcecode available under: ftp://ftp.mathsource.com/pub/Publications/BookSupplements/Freeman-1993
Helps the reader make his own NNs. The mathematica code for the programs in the book is also available through the internet: Send mail to MathSource@wri.com or try http://www.wri.com/ on the World Wide Web.

Freeman, J.A. and Skapura, D.M. (1991). *Neural Networks: Algorithms, Applications, and Programming Techniques*, Reading, MA: Addison-Wesley.
Book Webpage (Publisher): http://www.awl.com/
Additional Information: Seems to be out of print.
A good book for beginning programmers who want to learn how to write NN programs while avoiding any understanding of what NNs do or why they do it.

Gately, E. (1996). *Neural Networks for Financial Forecasting.* New York: John Wiley and Sons, Inc.
Book Webpage (Publisher): http://www.wiley.com/
Additional Information: One has to search.

Franco Insana comments:

```
* Decent book for the neural net beginner
* Very little devoted to statistical framework, although there
    is some formulation of backprop theory
* Some food for thought
* Nothing here for those with any neural net experience
```

McClelland, J. L. and Rumelhart, D. E. (1988). *Explorations in Parallel Distributed Processing: Computational Models of Cognition and Perception* (software manual). The MIT Press.
Book Webpage (Publisher): http://mitpress.mit.edu/book-home.tcl?isbn=026263113X (IBM version) and
http://mitpress.mit.edu/book-home.tcl?isbn=0262631296 (Macintosh)
Comments from readers of comp.ai.neural-nets: "Written in a tutorial style, and includes 2 diskettes of NN simulation programs that can be compiled on MS-DOS or Unix (and they do too !)"; "The programs are pretty reasonable as an introduction to some of the things that NNs can do."; "There are *two* editions of this book. One comes with disks for the IBM PC, the other comes with disks for the Macintosh".

McCord Nelson, M. and Illingworth, W.T. (1990). *A Practical Guide to Neural Nets.* Addison-Wesley Publishing Company, Inc. (ISBN 0-201-52376-0).
Book Webpage (Publisher): http://cseng.aw.com/bookdetail.qry?ISBN=0-201-63378-7&ptype=1174
Lots of applications without technical details, lots of hype, lots of goofs, no formulas.

Muller, B., Reinhardt, J., Strickland, M. T. (1995). *Neural Networks.:An Introduction* (2nd ed.). Berlin, Heidelberg, New York: Springer-Verlag. ISBN 3-540-60207-0. (DOS 3.5" disk included.)
Book Webpage (Publisher): http://www.springer.de/catalog/html-files/deutsch/phys/3540602070.html
Comments from readers of comp.ai.neural-nets: "The book was developed out of a course on neural-network models with computer demonstrations that was taught by the authors to Physics students. The book comes together with a PC-diskette. The book is divided into three parts: (1) Models of Neural Networks; describing several architectures and learing rules, including the mathematics. (2) Statistical Physics of Neural Networks; "hard-core" physics section developing formal theories of stochastic neural networks. (3) Computer Codes; explanation about the demonstration programs. First part gives a nice introduction into neural networks together with the formulas. Together with the demonstration programs a 'feel' for neural networks can be developed."

Orchard, G.A. & Phillips, W.A. (1991). *Neural Computation: A Beginner's Guide.* Lawrence Earlbaum Associates: London.
Comments from readers of comp.ai.neural-nets: "Short user-friendly introduction to the area, with a non-technical flavour. Apparently accompanies a software package, but I haven't seen that yet".

Rao, V.B, and Rao, H.V. (1993). *C++ Neural Networks and Fuzzy Logic.* MIS:Press, ISBN 1-55828-298-x, US $45 incl. disks.
Covers a wider variety of networks than Masters (1993), but is shallow and lacks Masters's insight into practical issues of using NNs.

Wasserman, P. D. (1989). *Neural Computing: Theory & Practice.* Van Nostrand Reinhold: New York. (ISBN 0-442-20743-3)
This is not as bad as some books on NNs. It provides an elementary account of the mechanics of a variety of networks. But it provides no insight into why various methods behave as they do, or under what conditions a method will or will not work well. It has no discussion of efficient training methods such as RPROP or conventional numerical optimization techniques. And, most egregiously, it has no explanation of overfitting and generalization beyond the patently false statement on p. 2 that "It is important to note that the artificial neural network generalizes automatically as a result of its structure"! There is no mention of training, validation, and test sets, or of other methods for estimating generalization error. There is no practical advice on the important issue of choosing the number of hidden units. There is no discussion of early stopping or weight decay. The reader will come away from this book with a grossly oversimplified view of NNs and no concept whatsoever of how to use NNs for practical applications.

Comments from readers of comp.ai.neural-nets: "Wasserman flatly enumerates some common architectures from an engineer's perspective ('how it works') without ever addressing the underlying fundamentals ('why it works') - important basic concepts such as clustering, principal components or gradient descent are not treated. It's also full of errors, and unhelpful diagrams drawn with

what appears to be PCB board layout software from the '70s. For anyone who wants to do active research in the field I consider it quite inadequate"; "Okay, but too shallow"; "Quite easy to understand"; "The best bedtime reading for Neural Networks. I have given this book to numerous collegues who want to know NN basics, but who never plan to implement anything. An excellent book to give your manager."

Not-quite-so-introductory Literature

Kung, S.Y. (1993). *Digital Neural Networks*, Prentice Hall, Englewood Cliffs, NJ.

Book Webpage (Publisher): http://www.prenhall.com/books/ptr_0136123260.html
Levine, D. S. (2000). *Introduction to Neural and Cognitive Modeling.* 2nd ed., Lawrence Erlbaum: Hillsdale, N.J.
Comments from readers of comp.ai.neural-nets: "Highly recommended".

Maren, A., Harston, C. and Pap, R., (1990). *Handbook of Neural Computing Applications.* Academic Press. ISBN: 0-12-471260-6. (451 pages)
Comments from readers of comp.ai.neural-nets: "They cover a broad area"; "Introductory with suggested applications implementation".

Pao, Y. H. (1989). *Adaptive Pattern Recognition and Neural Networks* Addison-Wesley Publishing Company, Inc. (ISBN 0-201-12584-6)
Book Webpage (Publisher): http://www.awl.com/
Comments from readers of comp.ai.neural-nets: "An excellent book that ties together classical approaches to pattern recognition with Neural Nets. Most other NN books do not even mention conventional approaches."

Refenes, A. (Ed.) (1995). *Neural Networks in the Capital Markets.* Chichester, England: John Wiley and Sons, Inc.
Book Webpage (Publisher): http://www.wiley.com/
Additional Information: One has to search.
Franco Insana comments:

```
* Not for the beginner
* Excellent introductory material presented by editor in first 5
  chapters, which could be a valuable reference source for any
  practitioner
* Very thought-provoking
* Mostly backprop-related
* Most contributors lay good statistical foundation
* Overall, a wealth of information and ideas, but the reader has to
  sift through it all to come away with anything useful
```

Simpson, P. K. (1990). *Artificial Neural Systems: Foundations, Paradigms, Applications and Implementations.* Pergamon Press: New York.
Comments from readers of comp.ai.neural-nets: "Contains a very useful 37 page bibliography. A large number of paradigms are presented. On the negative side the book is very shallow. Best used as a complement to other books".

Wasserman, P.D. (1993). *Advanced Methods in Neural Computing.* Van Nostrand Reinhold: New York (ISBN: 0-442-00461-3).
Comments from readers of comp.ai.neural-nets: "Several neural network topics are discussed e.g. Probalistic Neural Networks, Backpropagation and beyond, neural control, Radial Basis Function Networks, Neural Engineering. Furthermore, several subjects related to neural networks are mentioned e.g. genetic algorithms, fuzzy logic, chaos. Just the functionality of these subjects is described; enough to get you started. Lots of references are given to more elaborate descriptions. Easy to read, no extensive mathematical background necessary."

Zeidenberg. M. (1990). *Neural Networks in Artificial Intelligence.* Ellis Horwood, Ltd., Chichester.
Comments from readers of comp.ai.neural-nets: "Gives the AI point of view".

Zornetzer, S. F., Davis, J. L. and Lau, C. (1990). *An Introduction to Neural and Electronic Networks.* Academic Press. (ISBN 0-12-781881-2)
Comments from readers of comp.ai.neural-nets: "Covers quite a broad range of topics (collection of articles/papers)."; "Provides a primer-like introduction and overview for a broad audience, and employs a strong interdisciplinary emphasis".

Zurada, Jacek M. (1992). *Introduction To Artificial Neural Systems.* Hardcover, 785 Pages, 317 Figures, ISBN 0-534-95460-X, 1992, PWS Publishing Company, Price: $56.75 (includes shipping, handling, and the ANS software diskette). Solutions Manual available.
Comments from readers of comp.ai.neural-nets: "Cohesive and comprehensive book on neural nets; as an engineering-oriented introduction, but also as a research foundation. Thorough exposition of fundamentals, theory and applications. Training and recall algorithms appear in boxes showing steps of algorithms, thus making programming of learning paradigms easy. Many illustrations and intuitive examples. Winner among NN textbooks at a senior UG/first year graduate level-[175 problems]." Contents: Intro, Fundamentals of Learning, Single-Layer & Multilayer Perceptron NN, Assoc. Memories, Self-organizing and Matching Nets, Applications, Implementations, Appendix)

Books with Source Code (C, C++)

Blum, Adam (1992), *Neural Networks in C++*, Wiley.

Review by Ian Cresswell. (For a review of the text, see "The Worst" below.)

Mr Blum has not only contributed a masterpiece of NN inaccuracy but also seems to lack a fundamental understanding of Object Orientation.

The excessive use of virtual methods (see page 32 for example), the inclusion of unnecessary 'friend' relationships (page 133) and a penchant for operator overloading (pick a page!) demonstrate inability in C++ and/or OO.

The introduction to OO that is provided trivialises the area and demonstrates a distinct lack of direction and/or understanding.

The public interfaces to classes are overspecified and the design relies upon the flawed neuron/layer/network model.

There is a notable disregard for any notion of a robust class hierarchy which is demonstrated by an almost total lack of concern for inheritance and associated reuse strategies.

The attempt to rationalise differing types of Neural Network into a single very shallow but wide class hierarchy is naive.

The general use of the 'float' data type would cause serious hassle if this software could possibly be extended to use some of the more sensitive variants of backprop on more difficult problems. It is a matter of great fortune that such software is unlikely to be reusable and will therefore, like all good dinosaurs, disappear with the passage of time.

The irony is that there is a card in the back of the book asking the unfortunate reader to part with a further $39.95 for a copy of the software (already included in print) on a 5.25" disk.

The author claims that his work provides an 'Object Oriented Framework ...'. This can best be put in his own terms (Page 137):

```
... garble(float noise) ...
```

Swingler, K. (1996), *Applying Neural Networks: A Practical Guide*, London: Academic Press.

Review by Ian Cresswell. (For a review of the text, see "The Worst" below.)

Before attempting to review the code associated with this book it should be clearly stated that it is supplied as an extra--almost as an afterthought. This may be a wise move.

Although not as bad as other (even commercial) implementations, the code provided lacks proper OO structure and is typical of C++ written in a C style.

Style criticisms include:

1. The use of public data fields within classes (loss of encapsulation).
2. Classes with no protected or private sections.
3. Little or no use of inheritance and/or run-time polymorphism.
4. Use of floats not doubles (a common mistake) to store values for connection weights.
5. Overuse of classes and public methods. The network class has 59 methods in its public section.
6. Lack of planning is evident for the construction of a class hierarchy.

This code is without doubt written by a rushed C programmer. Whilst it would require a C++ compiler to be successfully used, it lacks the tight (optimised) nature of good C and the high level of abstraction of good C++.

In a generous sense the code is free and the author doesn't claim any expertise in software engineering. It works in a limited sense but would be difficult to extend and/or reuse. It's fine for demonstration purposes in a stand-alone manner and for use with the book concerned.

If you're serious about nets you'll end up rewriting the whole lot (or getting something better).

The Worst

How not to use neural nets in any programming language

Blum, Adam (1992), *Neural Networks in C++*, NY: Wiley.

Welstead, Stephen T. (1994), *Neural Network and Fuzzy Logic Applications in C/C++*, NY: Wiley.

(For a review of Blum's source code, see "Books with Source Code" above.)

Both Blum and Welstead contribute to the dangerous myth that any idiot can use a neural net by dumping in whatever data are handy and letting it train for a few days. They both have little or no discussion of generalization, validation, and overfitting. Neither provides any valid advice on choosing the number of hidden nodes. If you have ever wondered where these stupid "rules of thumb" that pop up frequently come from, here's a source for one of them:

"A rule of thumb is for the size of this [hidden] layer to be somewhere between the input layer size ... and the output layer size ..." Blum, p. 60.

(John Lazzaro tells me he recently "reviewed a paper that cited this rule of thumb--and referenced this book! Needless to say, the final version of that paper didn't include the reference!")

Blum offers some profound advice on choosing inputs:

> "The next step is to pick as many input factors as possible that might be related to [the target]."

Blum also shows a deep understanding of statistics:

> "A statistical model is simply a more indirect way of learning correlations. With a neural net approach, we model the problem directly." p. 8.

Blum at least mentions some important issues, however simplistic his advice may be. Welstead just ignores them. What Welstead gives you is code--vast amounts of code. I have no idea how anyone could write *that* much code for a simple feedforward NN. Welstead's approach to validation, in his chapter on financial forecasting, is to reserve *two* cases for the validation set!

My comments apply only to the text of the above books. I have not examined or attempted to compile the code.

An impractical guide to neural nets

> Swingler, K. (1996), *Applying Neural Networks: A Practical Guide*, London: Academic Press.

(For a review of the source code, see "Books with Source Code" above.)

This book has lots of good advice liberally sprinkled with errors, incorrect formulas, some bad advice, and some very serious mistakes. Experts will learn nothing, while beginners will be unable to separate the useful information from the dangerous. For example, there is a chapter on "Data encoding and re-coding" that would be very useful to beginners if it were accurate, but the formula for the standard deviation is wrong, and the description of the softmax function is of something entirely different than softmax (see What is a softmax activation function?). Even more dangerous is the statement on p. 28 that "Any pair of variables with high covariance are dependent, and one may be chosen to be discarded." Although high correlations can be used to identify redundant inputs, it is incorrect to use high covariances for this purpose, since a covariance can be high simply because one of the inputs has a high standard deviation.

The most ludicrous thing I've found in the book is the claim that Hecht-Neilsen used Kolmogorov's theorem to show that "you will never require more than twice the number of hidden units as you have inputs" (p. 53) in an MLP with one hidden layer. Actually, Hecht-Neilsen, says "the direct usefulness of this result is doubtful, because no constructive method for developing the [output activation] functions is known." Then Swingler implies that V. Kurkova (1991, "Kolmogorov's theorem is relevant," Neural Computation, 3, 617-622) confirmed this alleged upper bound on the number of hidden units, saying that, "Kurkova was able to restate Kolmogorov's theorem in terms of a set of sigmoidal functions." If Kolmogorov's theorem, or Hecht-Nielsen's adaptation of it, could be restated in terms of known sigmoid activation functions in the (single) hidden and output layers, then Swingler's alleged upper bound would be correct, but in fact no such restatement of Kolmogorov's theorem is possible, and Kurkova did not claim to prove any such restatement. Swingler omits the crucial details that Kurkova used two hidden layers, staircase-like activation functions (not ordinary sigmoidal functions such as the logistic) in the first hidden layer, and a potentially large number of units in the second hidden layer. Kurkova later estimated the number of units required for uniform approximation within an error `epsilon` as `nm(m+1)` in the first hidden layer and `m^2(m+1)^n` in the second hidden layer, where `n` is the number of inputs and `m` "depends on `epsilon/||f||` as well as on the rate with which `f` increases distances." In other words, Kurkova says nothing to support Swinglers advice (repeated on p. 55), "Never choose h to be more than twice the number of input units." Furthermore, constructing a counter example to Swingler's advice is trivial: use one input and one output, where the output is the sine of the input, and the domain of the input extends over many cycles of the sine wave; it is obvious that many more than two hidden units are required. For some sound information on choosing the number of hidden units, see How many hidden units should I use?

Choosing the number of hidden units is one important aspect of getting good generalization, which is the most crucial issue in neural network training. There are many other considerations involved in getting good generalization, and Swingler makes several more mistakes in this area:

- There is dangerous misinformation on p. 55, where Swingler says, "If a data set contains no noise, then there is no risk of overfitting as there is nothing to overfit." It is true that overfitting is more common with noisy data, but severe overfitting can occur with noise-free data, even when there are more training cases than weights. There is an example of such overfitting under

How many hidden layers should I use?

- Regarding the use of added noise (jitter) in training, Swingler says on p. 60, "The more noise you add, the more general your model becomes." This statement makes no sense as it stands (it would make more sense if "general" were changed to "smooth"), but it could certainly encourage a beginner to use far too much jitter--see What is jitter? (Training with noise).

- On p. 109, Swingler describes leave-one-out cross-validation, which he ascribes to Hecht-Neilsen. But Swingler concludes, "the method provides you with L minus 1 networks to choose from; none of which has been validated properly," completely missing the point that cross-validation provides an estimate of the generalization error of a network trained on the entire training set of L cases--see What are cross-validation and bootstrapping? Also, there are L leave-one-out networks, not L-1.

While Swingler has some knowldege of statistics, his expertise is not sufficient for him to detect that certain articles on neural nets are statistically nonsense. For example, on pp. 139-140 he uncritically reports a method that allegedly obtains error bars by doing a simple linear regression on the target vs. output scores. To a trained statistician, this method is obviously wrong (and, as usual in this book, the formula for variance given for this method on p. 150 is wrong). On p. 110, Swingler reports an article that attempts to apply bootstrapping to neural nets, but this article is also obviously wrong to anyone familiar with bootstrapping. While Swingler cannot be blamed entirely for accepting these articles at face value, such misinformation provides yet more hazards for beginners.

Swingler addresses many important practical issues, and often provides good practical advice. But the peculiar combination of much good advice with some extremely bad advice, a few examples of which are provided above, could easily seduce a beginner into thinking that the book as a whole is reliable. It is this danger that earns the book a place in "The Worst" list.

Bad science writing

Dewdney, A.K. (1997), *Yes, We Have No Neutrons: An Eye-Opening Tour through the Twists and Turns of Bad Science*, NY: Wiley.

This book, allegedly an expose of bad science, contains only one chapter of 19 pages on "the neural net debacle" (p. 97). Yet this chapter is so egregiously misleading that the book has earned a place on "The Worst" list. A detailed criticism of this chapter, along with some other sections of the book, can be found at ftp://ftp.sas.com/pub/neural/badscience.html. Other chapters of the book are reviewed in the November, 1997, issue of *Scientific American.*

Subject: Journals and magazines about Neural Networks?

```
[to be added: comments on speed of reviewing and publishing,
              whether they accept TeX format or ASCII by e-mail, etc.]
```

A. Dedicated Neural Network Journals:

```
Title:   Neural Networks
Publish: Pergamon Press
Address: Pergamon Journals Inc., Fairview Park, Elmsford,
         New York 10523, USA and Pergamon Journals Ltd.
         Headington Hill Hall, Oxford OX3, 0BW, England
Freq.:   10 issues/year (vol. 1 in 1988)
Cost/Yr: Free with INNS or JNNS or ENNS membership ($45?),
```

```
          Individual $65, Institution $175
 ISSN #:  0893-6080
 URL:     http://www.elsevier.nl/locate/inca/841
 Remark:  Official Journal of International Neural Network Society (INNS),
          European Neural Network Society (ENNS) and Japanese Neural
          Network Society (JNNS).
          Contains Original Contributions, Invited Review Articles, Letters
          to Editor, Book Reviews, Editorials, Announcements, Software Surveys.

 Title:   Neural Computation
 Publish: MIT Press
 Address: MIT Press Journals, 55 Hayward Street Cambridge,
          MA 02142-9949, USA, Phone: (617) 253-2889
 Freq.:   Quarterly (vol. 1 in 1989)
 Cost/Yr: Individual $45, Institution $90, Students $35; Add $9 Outside USA
 ISSN #:  0899-7667
 URL:     http://mitpress.mit.edu/journals-legacy.tcl
 Remark:  Combination of Reviews (10,000 words), Views (4,000 words)
          and Letters (2,000 words).  I have found this journal to be of
          outstanding quality.
          (Note: Remarks supplied by Mike Plonski "plonski@aero.org")

 Title:   NEURAL COMPUTING SURVEYS
 Publish: Lawrence Erlbaum Associates
 Address: 10 Industrial Avenue, Mahwah, NJ  07430-2262, USA
 Freq.:   Yearly
 Cost/Yr: Free on-line
 ISSN #:  1093-7609
 URL:     http://www.icsi.berkeley.edu/~jagota/NCS/
 Remark:  One way to cope with the exponential increase in the number
          of articles published in recent years is to ignore most of
          them. A second, perhaps more satisfying, approach is to
          provide a forum that encourages the regular production --
          and perusal -- of high-quality survey articles. This is
          especially useful in an inter-disciplinary, evolving field
          such as neural networks. This journal aims to bring the
          second view-point to bear. It is intended to

          * encourage researchers to write good survey papers.
          * motivate researchers to look here first to check
            what's known on an unfamiliar topic.

 Title:   IEEE Transactions on Neural Networks
 Publish: Institute of Electrical and Electronics Engineers (IEEE)
 Address: IEEE Service Cemter, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ,
          08855-1331 USA. Tel: (201) 981-0060
```

```
  Cost/Yr: $10 for Members belonging to participating IEEE societies
  Freq.:   Quarterly (vol. 1 in March 1990)
  URL:     http://www.ieee.org/nnc/pubs/transactions.html
  Remark:  Devoted to the science and technology of neural networks
           which disclose significant  technical knowledge, exploratory
           developments and applications of neural networks from biology to
           software to hardware.  Emphasis is on artificial neural networks.
           Specific aspects include self organizing systems, neurobiological
           connections, network dynamics and architecture, speech recognition,
           electronic and photonic implementation, robotics and controls.
           Includes Letters concerning new research results.
           (Note: Remarks are from journal announcement)

  Title:   IEEE Transactions on Evolutionary Computation
  Publish: Institute of Electrical and Electronics Engineers (IEEE)
  Address: IEEE Service Cemter, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ,
           08855-1331 USA. Tel: (201) 981-0060
  Cost/Yr: $10 for Members belonging to participating IEEE societies
  Freq.:   Quarterly (vol. 1 in May 1997)
  URL:     http://engine.ieee.org/nnc/pubs/transactions.html
  Remark:  The IEEE Transactions on Evolutionary Computation will publish archival
           journal quality original papers in evolutionary computation and related
           areas, with particular emphasis on the practical application of the
           techniques to solving real problems in industry, medicine, and other
           disciplines.  Specific techniques include but are not limited to
           evolution strategies, evolutionary programming, genetic algorithms, and
           associated methods of genetic programming and classifier systems.  Papers
           emphasizing mathematical results should ideally seek to put these results
           in the context of algorithm design, however purely theoretical papers will
           be considered.  Other papers in the areas of cultural algorithms, artificial
           life, molecular computing, evolvable hardware, and the use of simulated
           evolution to gain a better understanding of naturally evolved systems are
           also encouraged.
           (Note: Remarks are from journal CFP)

  Title:   International Journal of Neural Systems
  Publish: World Scientific Publishing
  Address: USA: World Scientific Publishing Co., 1060 Main Street, River Edge,
           NJ 07666. Tel: (201) 487 9655; Europe: World Scientific Publishing
           Co. Ltd., 57 Shelton Street, London WC2H 9HE, England.
           Tel: (0171) 836 0888; Asia: World Scientific Publishing Co. Pte. Ltd.,
           1022 Hougang Avenue 1 #05-3520, Singapore 1953, Rep. of Singapore
           Tel: 382 5663.
  Freq.:   Quarterly (Vol. 1 in 1990)
  Cost/Yr: Individual $122, Institution $255 (plus $15-$25 for postage)
  ISSN #:  0129-0657 (IJNS)
```

Remark: The International Journal of Neural Systems is a quarterly
journal which covers information processing in natural
and artificial neural systems. Contributions include research papers,
reviews, and Letters to the Editor - communications under 3,000
words in length, which are published within six months of receipt.
Other contributions are typically published within nine months.
The journal presents a fresh undogmatic attitude towards this
multidisciplinary field and aims to be a forum for novel ideas and
improved understanding of collective and cooperative phenomena with
computational capabilities.
Papers should be submitted to World Scientific's UK office. Once a
paper is accepted for publication, authors are invited to e-mail
the LaTeX source file of their paper in order to expedite publication.

Title: International Journal of Neurocomputing
Publish: Elsevier Science Publishers, Journal Dept.; PO Box 211;
1000 AE Amsterdam, The Netherlands
Freq.: Quarterly (vol. 1 in 1989)
URL: http://www.elsevier.nl/locate/inca/505628

Title: Neural Processing Letters
Publish: Kluwer Academic publishers
Address: P.O. Box 322, 3300 AH Dordrecht, The Netherlands
Freq: 6 issues/year (vol. 1 in 1994)
Cost/Yr: Individuals $198, Institution $400 (including postage)
ISSN #: 1370-4621
URL: http://www.wkap.nl/journalhome.htm/1370-4621
Remark: The aim of the journal is to rapidly publish new ideas, original
developments and work in progress. Neural Processing Letters
covers all aspects of the Artificial Neural Networks field.
Publication delay is about 3 months.

Title: Neural Network News
Publish: AIWeek Inc.
Address: Neural Network News, 2555 Cumberland Parkway, Suite 299,
Atlanta, GA 30339 USA. Tel: (404) 434-2187
Freq.: Monthly (beginning September 1989)
Cost/Yr: USA and Canada $249, Elsewhere $299
Remark: Commercial Newsletter

Title: Network: Computation in Neural Systems
Publish: IOP Publishing Ltd
Address: Europe: IOP Publishing Ltd, Techno House, Redcliffe Way, Bristol
BS1 6NX, UK; IN USA: American Institute of Physics, Subscriber
Services 500 Sunnyside Blvd., Woodbury, NY 11797-2999
Freq.: Quarterly (1st issue 1990)

```
Cost/Yr: USA: $180,  Europe: 110 pounds
URL:     http://www.iop.org/Journals/ne
Remark:  Description: "a forum for integrating theoretical and experimental
         findings across relevant interdisciplinary boundaries."  Contents:
         Submitted articles reviewed by two technical referees  paper's
         interdisciplinary format and accessability."  Also Viewpoints and
         Reviews commissioned by the editors, abstracts (with reviews) of
         articles published in other journals, and book reviews.
         Comment: While the price discourages me (my comments are based
         upon a free sample copy), I think that the journal succeeds
         very well.  The highest density of interesting articles I
         have found in any journal.
         (Note: Remarks supplied by kehoe@csufres.CSUFresno.EDU)

Title:   Connection Science: Journal of Neural Computing,
         Artificial Intelligence and Cognitive Research
Publish: Carfax Publishing
Address: Europe: Carfax Publishing Company, PO Box 25, Abingdon, Oxfordshire
         OX14 3UE, UK.
         USA: Carfax Publishing Company, PO Box 2025, Dunnellon, Florida
         34430-2025, USA
         Australia: Carfax Publishing Company, Locked Bag 25, Deakin,
         ACT 2600, Australia
Freq.:   Quarterly (vol. 1 in 1989)
Cost/Yr: Personal rate:
         48 pounds (EC) 66 pounds (outside EC) US$118 (USA and Canada)
         Institutional rate:
         176 pounds (EC) 198 pounds (outside EC) US$340 (USA and Canada)

Title:   International Journal of Neural Networks
Publish: Learned Information
Freq.:   Quarterly (vol. 1 in 1989)
Cost/Yr: 90 pounds
ISSN #:  0954-9889
Remark:  The journal contains articles, a conference report (at least the
         issue I have), news and a calendar.
         (Note: remark provided by J.R.M. Smits "anjos@sci.kun.nl")

Title:   Sixth Generation Systems (formerly Neurocomputers)
Publish: Gallifrey Publishing
Address: Gallifrey Publishing, PO Box 155, Vicksburg, Michigan, 49097, USA
         Tel: (616) 649-3772, 649-3592 fax
Freq.    Monthly (1st issue January, 1987)
ISSN #:  0893-1585
Editor:  Derek F. Stubbs
Cost/Yr: $79 (USA, Canada), US$95 (elsewhere)
```

```
Remark:  Runs eight to 16 pages monthly. In 1995 will go to floppy disc-based
publishing with databases +, "the equivalent to 50 pages per issue are
planned." Often focuses on specific topics: e.g., August, 1994 contains two
articles: "Economics, Times Series and the Market," and "Finite Particle
Analysis - [part] II."  Stubbs also directs the company Advanced Forecasting
Technologies. (Remark by Ed Rosenfeld: ier@aol.com)

Title:   JNNS Newsletter (Newsletter of the Japan Neural Network Society)
Publish: The Japan Neural Network Society
Freq.:   Quarterly (vol. 1 in 1989)
Remark:  (IN JAPANESE LANGUAGE) Official Newsletter of the Japan Neural
         Network Society(JNNS)
         (Note: remarks by Osamu Saito "saito@nttica.NTT.JP")

Title:   Neural Networks Today
Remark:  I found this title in a bulletin board of october last year.
         It was a message of Tim Pattison, timpatt@augean.OZ
         (Note: remark provided by J.R.M. Smits "anjos@sci.kun.nl")

Title:   Computer Simulations in Brain Science

Title:   Internation Journal of Neuroscience

Title:   Neural Network Computation
Remark:  Possibly the same as "Neural Computation"

Title:   Neural Computing and Applications
Freq.:   Quarterly
Publish: Springer Verlag
Cost/yr: 120 Pounds
Remark:  Is the journal of the Neural Computing Applications Forum.
         Publishes original research and other information
         in the field of practical applications of neural computing.
```

B. NN Related Journals:

```
Title:   Biological Cybernetics (Kybernetik)
Publish: Springer Verlag
Remark:  Monthly (vol. 1 in 1961)

Title:   Various IEEE Transactions and Magazines
Publish: IEEE
Remark:  Primarily see IEEE Trans. on System, Man and Cybernetics;
         Various Special Issues: April 1990 IEEE Control Systems
         Magazine.; May 1989 IEEE Trans. Circuits and Systems.;
         July 1988 IEEE Trans. Acoust. Speech Signal Process.
```

```
Title:   The Journal of Experimental and Theoretical Artificial Intelligence
Publish: Taylor & Francis, Ltd.
Address: London, New York, Philadelphia
Freq.:   ? (1st issue Jan 1989)
Remark:  For submission information, please contact either of the editors:
         Eric Dietrich                        Chris Fields
         PACSS - Department of Philosophy     Box 30001/3CRL
         SUNY Binghamton                      New Mexico State University
         Binghamton, NY 13901                 Las Cruces, NM 88003-0001
         dietrich@bingvaxu.cc.binghamton.edu  cfields@nmsu.edu

Title:   The Behavioral and Brain Sciences
Publish: Cambridge University Press
Remark:  (Remarks by Don Wunsch
         This is a delightful journal that encourages discussion on a
         variety of controversial topics.  I have especially enjoyed
         reading some papers in there by Dana Ballard and Stephen
         Grossberg (separate papers, not collaborations) a few years
         back.  They have a really neat concept: they get a paper,
         then invite a number of noted scientists in the field to
         praise it or trash it.  They print these commentaries, and
         give the author(s) a chance to make a rebuttal or
         concurrence.  Sometimes, as I'm sure you can imagine, things
         get pretty lively. Their reviewers are called something like
         Behavioral and Brain Associates, and I believe they have to
         be nominated by current associates, and should be fairly
         well established in the field. The main thing is that I liked
         the articles I read.

Title:   International Journal of Applied Intelligence
Publish: Kluwer Academic Publishers
Remark:  first issue in 1990(?)

Title:   International Journal of Modern Physics C
Publish: USA: World Scientific Publishing Co., 1060 Main Street, River Edge,
         NJ 07666. Tel: (201) 487 9655; Europe: World Scientific Publishing
         Co. Ltd., 57 Shelton Street, London WC2H 9HE, England.
         Tel: (0171) 836 0888; Asia: World Scientific Publishing Co. Pte. Ltd.,
         1022 Hougang Avenue 1 #05-3520, Singapore 1953, Rep. of Singapore
         Tel: 382 5663.
Freq:    bi-monthly
Eds:     H. Herrmann, R. Brower, G.C. Fox and S Nose

Title:   Machine Learning
Publish: Kluwer Academic Publishers
```

```
  Address: Kluwer Academic Publishers
           P.O. Box 358
           Accord Station
           Hingham, MA 02018-0358 USA
  Freq.:   Monthly (8 issues per year; increasing to 12 in 1993)
  Cost/Yr: Individual $140 (1992); Member of AAAI or CSCSI $88
  Remark:  Description: Machine Learning is an international forum for
           research on computational approaches to learning.  The journal
           publishes articles reporting substantive research results on a
           wide range of learning methods applied to a variety of task
           domains.  The ideal paper will make a theoretical contribution
           supported by a computer implementation.
           The journal has published many key papers in learning theory,
           reinforcement learning, and decision tree methods.  Recently
           it has published a special issue on connectionist approaches
           to symbolic reasoning.  The journal regularly publishes
           issues devoted to genetic algorithms as well.

  Title:   INTELLIGENCE - The Future of Computing
  Published by: Intelligence
  Address: INTELLIGENCE, P.O. Box 20008, New York, NY 10025-1510, USA,
  212-222-1123 voice & fax; email: ier@aol.com, CIS: 72400,1013
  Freq.    Monthly plus four special reports each year (1st issue: May, 1984)
  ISSN #:  1042-4296
  Editor:  Edward Rosenfeld
  Cost/Yr: $395 (USA), US$450 (elsewhere)
  Remark:  Has absorbed several other newsletters, like Synapse/Connection
           and Critical Technology Trends (formerly AI Trends).
           Covers NN, genetic algorithms, fuzzy systems, wavelets, chaos
           and other advanced computing approaches, as well as molecular
           computing and nanotechnology.

  Title:   Journal of Physics A: Mathematical and General
  Publish: Inst. of Physics, Bristol
  Freq:    24 issues per year.
  Remark:  Statistical mechanics aspects of neural networks
           (mostly Hopfield models).

  Title:   Physical Review A: Atomic, Molecular and Optical Physics
  Publish: The American Physical Society (Am. Inst. of Physics)
  Freq:    Monthly
  Remark:  Statistical mechanics of neural networks.

  Title:   Information Sciences
  Publish: North Holland (Elsevier Science)
  Freq.:   Monthly
```

```
ISSN:    0020-0255
Editor:  Paul P. Wang; Department of Electrical Engineering; Duke University;
         Durham, NC 27706, USA
```

Subject: Conferences and Workshops on Neural Networks?

- The journal "Neural Networks" has a list of conferences, workshops and meetings in each issue.
- NEuroNet maintains a list of Neural Network Events at http://www.kcl.ac.uk/neuronet/events/index.html
- The IEEE Neural Network Council maintains a list of conferences at http://www.ieee.org/nnc.
- Conferences, workshops, and other events concerned with neural networks, inductive learning, genetic algorithms, data mining, agents, applications of AI, pattern recognition, vision, and related fields. are listed at Georg Thimm's web page http://www.drc.ntu.edu.sg/users/mgeorg/enter.epl

---

Subject: Neural Network Associations?

1. ## International Neural Network Society (INNS).

   INNS membership includes subscription to "Neural Networks", the official journal of the society. Membership is $55 for non-students and $45 for students per year. Address: INNS Membership, P.O. Box 491166, Ft. Washington, MD 20749.

2. ## International Student Society for Neural Networks (ISSNNets).

   Membership is $5 per year. Address: ISSNNet, Inc., P.O. Box 15661, Boston, MA 02215 USA

3. ## Women In Neural Network Research and technology (WINNERS).

   Address: WINNERS, c/o Judith Dayhoff, 11141 Georgia Ave., Suite 206, Wheaton, MD 20902. Phone: 301-933-9000.

4. ## European Neural Network Society (ENNS)

   ENNS membership includes subscription to "Neural Networks", the official journal of the society. Membership is currently (1994) 50 UK pounds (35 UK pounds for students) per year. Address: ENNS Membership, Centre for Neural Networks, King's College London, Strand, London WC2R 2LS, United Kingdom.

5. ## Japanese Neural Network Society (JNNS)

   Address: Japanese Neural Network Society; Department of Engineering, Tamagawa University; 6-1-1, Tamagawa Gakuen, Machida City, Tokyo; 194 JAPAN; Phone: +81 427 28 3457, Fax: +81 427 28 3597

6. Association des Connexionnistes en THese (ACTH)

(the French Student Association for Neural Networks); Membership is 100 FF per year; Activities: newsletter, conference (every year), list of members, electronic forum; Journal 'Valgo' (ISSN 1243-4825); WWW page: http://www.supelec-rennes.fr/acth/welcome.html ; Contact: acth@loria.fr

7. Neurosciences et Sciences de l'Ingenieur (NSI)

Biology & Computer Science Activity : conference (every year) Address : NSI - TIRF / INPG 46 avenue Felix Viallet 38031 Grenoble Cedex FRANCE

8. IEEE Neural Networks Council

Web page at http://www.ieee.org/nnc

9. SNN (Foundation for Neural Networks)

The Foundation for Neural Networks (SNN) is a university based non-profit organization that stimulates basic and applied research on neural networks in the Netherlands. Every year SNN orgines a symposium on Neural Networks. See http://www.mbfys.kun.nl/SNN/.

You can find nice lists of NN societies in the WWW at http://www.emsl.pnl.gov:2080/proj/neuron/neural/societies.html and at http://www.ieee.org:80/nnc/research/othernnsoc.html.

Subject: Mailing lists, BBS, CD-ROM?

See also "Other NN links?"in Part 7 of the FAQ.

1. Machine Learning mailing list

http://groups.yahoo.com/group/machine-learning/

The Machine Learning mailing list is an unmoderated mailing list intended for people in Computer Sciences, Statistics, Mathematics, and other areas or disciplines with interests in Machine Learning. Researchers, practitioners, and users of Machine Learning in academia, industry, and government are encouraged to join the list to discuss and exchange ideas regarding any aspect of Machine Learning, e.g., various learning algorithms, data pre-processing, variable selection mechanism, instance selection, and applications to real-world problems.

You can post, read, and reply messages on the Web. Or you can choose to receive messages as individual emails, daily summaries, daily full-text digest, or read them on the Web only.

2. The Connectionists Mailing List

http://www.cnbc.cmu.edu/other/connectionists.html

CONNECTIONISTS is a moderated mailing list for discussion of technical issues relating to neural computation, and dissemination of professional announcements such as calls for papers, book announcements, and electronic preprints. CONNECTIONISTS is focused on meeting the needs of active researchers in the field, not on answering questions from beginners.

3. Central Neural System Electronic Bulletin Board

```
URL: ftp://www.centralneuralsystem.com/pub/CNS/bbs
Supported by: Wesley R. Elsberry
              3027 Macaulay Street
              San Diego, CA 92106
Email: welsberr@inia.cls.org
Alternative URL: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/ai-
repository/ai/areas/neural/cns/0.html
```

Many MS-DOS PD and shareware simulations, source code, benchmarks, demonstration packages, information files; some Unix, Macintosh, Amiga related files. Also available are files on AI, AI Expert listings 1986-1991, fuzzy logic, genetic algorithms, artificial life, evolutionary biology, and many Project Gutenberg and Wiretap e-texts.

4. AI CD-ROM

Network Cybernetics Corporation produces the "AI CD-ROM". It is an ISO-9660 format CD-ROM and contains a large assortment of software related to artificial intelligence, artificial life, virtual reality, and other topics. Programs for OS/2, MS-DOS, Macintosh, UNIX, and other operating systems are included. Research papers, tutorials, and other text files are included in ASCII, RTF, and other universal formats. The files have been collected from AI bulletin boards, Internet archive sites, University computer deptartments, and other government and civilian AI research organizations. Network Cybernetics Corporation intends to release annual revisions to the AI CD-ROM to keep it up to date with current developments in the field. The AI CD-ROM includes collections of files that address many specific AI/AL topics including Neural Networks (Source code and executables for many different platforms including Unix, DOS, and Macintosh. ANN development tools, example networks, sample data, tutorials. A complete collection of Neural Digest is included as well.) The AI CD-ROM may be ordered directly by check, money order, bank draft, or credit card from:

```
Network Cybernetics Corporation;
4201 Wingren Road Suite 202;
Irving, TX 75062-2763;
Tel 214/650-2002;
Fax 214/650-1929;
```

The cost is $129 per disc + shipping ($5/disc domestic or $10/disc foreign) (See the comp.ai FAQ for further details)

Subject: How to benchmark learning methods?

The NN benchmarking resources page at http://wwwipd.ira.uka.de/~prechelt/NIPS_bench.html was created after a NIPS 1995 workshop on NN benchmarking. The page contains pointers to

various papers on proper benchmarking methodology and to various sources of datasets.

Benchmark studies require some familiarity with the statistical design and analysis of experiments. There are many textbooks on this subject, of which Cohen (1995) will probably be of particular interest to researchers in neural nets and machine learning (see also the review of Cohen's book by Ron Kohavi in the International Journal of Neural Systems, which can be found on-line at http://robotics.stanford.edu/users/ronnyk/ronnyk-bib.html).

Reference:

Cohen, P.R. (1995), *Empirical Methods for Artificial Intelligence*, Cambridge, MA: The MIT Press.

Subject: Databases for experimentation with NNs?

1. ## UCI machine learning database

   A large collection of data sets accessible via anonymous FTP at ftp.ics.uci.edu [128.195.1.1] in directory /pub/machine-learning-databases" or via web browser at http://www.ics.uci.edu/~mlearn/MLRepository.html

2. ## UCI KDD Archive

   The UC Irvine Knowledge Discovery in Databases (KDD) Archive at http://kdd.ics.uci.edu/ is an online repository of large datasets which encompasses a wide variety of data types, analysis tasks, and application areas. The primary role of this repository is to serve as a benchmark testbed to enable researchers in knowledge discovery and data mining to scale existing and future data analysis algorithms to very large and complex data sets. This archive is supported by the Information and Data Management Program at the National Science Foundation, and is intended to expand the current UCI Machine Learning Database Repository to datasets that are orders of magnitude larger and more complex.

3. ## The neural-bench Benchmark collection

   Accessible at http://www.boltz.cs.cmu.edu/ or via anonymous FTP at ftp://ftp.boltz.cs.cmu.edu/pub/neural-bench/. In case of problems or if you want to donate data, email contact is "neural-bench@cs.cmu.edu". The data sets in this repository include the 'nettalk' data, 'two spirals', protein structure prediction, vowel recognition, sonar signal classification, and a few others.

4. ## Proben1

   Proben1 is a collection of 12 learning problems consisting of real data. The datafiles all share a single simple common format. Along with the data comes a technical report describing a set of rules and conventions for performing and reporting benchmark tests and their results. Accessible via anonymous FTP on ftp.cs.cmu.edu [128.2.206.173] as /afs/cs/project/connect/bench/contrib/prechelt/proben1.tar.gz. and also on ftp.ira.uka.de as /pub/neuron/proben1.tar.gz. The file is about 1.8 MB and unpacks into about 20 MB.

5. ## Delve: Data for Evaluating Learning in Valid Experiments

Delve is a standardised, copyrighted environment designed to evaluate the performance of learning methods. Delve makes it possible for users to compare their learning methods with other methods on many datasets. The Delve learning methods and evaluation procedures are well documented, such that meaningful comparisons can be made. The data collection includes not only isolated data sets, but "families" of data sets in which properties of the data, such as number of inputs and degree of nonlinearity or noise, are systematically varied. The Delve web page is at http://www.cs.toronto.edu/~delve/

6. Bilkent University Function Approximation Repository

A repository of data sets collected mainly by searching resources on the web can be found at http://funapp.cs.bilkent.edu.tr/DataSets/ Most of the data sets are used for the experimental analysis of function approximation techniques and for training and demonstration by machine learning and statistics community. The original sources of most data sets can be accessed via associated links. A compressed tar file containing all data sets is available.

7. NIST special databases of the National Institute Of Standards And Technology:

Several large databases, each delivered on a CD-ROM. Here is a quick list.

- NIST Binary Images of Printed Digits, Alphas, and Text
- NIST Structured Forms Reference Set of Binary Images
- NIST Binary Images of Handwritten Segmented Characters
- NIST 8-bit Gray Scale Images of Fingerprint Image Groups
- NIST Structured Forms Reference Set 2 of Binary Images
- NIST Test Data 1: Binary Images of Hand-Printed Segmented Characters
- NIST Machine-Print Database of Gray Scale and Binary Images
- NIST 8-Bit Gray Scale Images of Mated Fingerprint Card Pairs
- NIST Supplemental Fingerprint Card Data (SFCD) for NIST Special Database 9
- NIST Binary Image Databases of Census Miniforms (MFDB)
- NIST Mated Fingerprint Card Pairs 2 (MFCP 2)
- NIST Scoring Package Release 1.0
- NIST FORM-BASED HANDPRINT RECOGNITION SYSTEM

Here are example descriptions of two of these databases:

NIST special database 2: Structured Forms Reference Set (SFRS)

The NIST database of structured forms contains 5,590 full page images of simulated tax forms completed using machine print. THERE IS NO REAL TAX DATA IN THIS DATABASE. The structured forms used in this database are 12 different forms from the 1988, IRS 1040 Package X. These include Forms 1040, 2106, 2441, 4562, and 6251 together with Schedules A, B, C, D, E, F and SE. Eight of these forms contain two pages or form faces making a total of 20 form faces represented in the database. Each image is stored in bi-level black and white raster format. The images in this database appear to be real forms prepared by individuals but the images have been automatically derived and synthesized using a computer and contain no "real" tax data. The entry field values on the forms have been automatically generated by a computer in order to make the data available without the danger of distributing privileged tax information. In addition to the images the database includes 5,590 answer files, one for each image. Each answer file contains an ASCII representation of the data found in the entry fields on the corresponding image. Image format documentation and example software are also provided. The uncompressed database totals approximately 5.9 gigabytes of data.

NIST special database 3: Binary Images of Handwritten Segmented Characters (HWSC)

Contains 313,389 isolated character images segmented from the 2,100 full-page images distributed with "NIST Special Database 1". 223,125 digits, 44,951 upper-case, and 45,313 lower-case character images. Each character image has been centered in a separate 128 by 128 pixel region, error rate of the segmentation and assigned classification is less than 0.1%. The

uncompressed database totals approximately 2.75 gigabytes of image data and includes image format documentation and example software.

The system requirements for all databases are a 5.25" CD-ROM drive with software to read ISO-9660 format. Contact: Darrin L. Dimmick; dld@magi.ncsl.nist.gov; (301)975-4147

The prices of the databases are between US$ 250 and 1895 If you wish to order a database, please contact: Standard Reference Data; National Institute of Standards and Technology; 221/A323; Gaithersburg, MD 20899; Phone: (301)975-2208; FAX: (301)926-0416

Samples of the data can be found by ftp on sequoyah.ncsl.nist.gov in directory /pub/data A more complete description of the available databases can be obtained from the same host as /pub/databases/catalog.txt

8. CEDAR CD-ROM 1: Database of Handwritten Cities, States, ZIP Codes, Digits, and Alphabetic Characters

The Center Of Excellence for Document Analysis and Recognition (CEDAR) State University of New York at Buffalo announces the availability of CEDAR CDROM 1: USPS Office of Advanced Technology The database contains handwritten words and ZIP Codes in high resolution grayscale (300 ppi 8-bit) as well as binary handwritten digits and alphabetic characters (300 ppi 1-bit). This database is intended to encourage research in off-line handwriting recognition by providing access to handwriting samples digitized from envelopes in a working post office.

```
Specifications of the database include:
+    300 ppi 8-bit grayscale handwritten words (cities,
     states, ZIP Codes)
     o    5632 city words
     o    4938 state words
     o    9454 ZIP Codes
+    300 ppi binary handwritten characters and digits:
     o    27,837 mixed alphas  and  numerics  segmented
          from address blocks
     o    21,179 digits segmented from ZIP Codes
+    every image supplied with  a  manually  determined
     truth value
+    extracted from live mail in a  working  U.S.  Post
     Office
+    word images in the test  set  supplied  with  dic-
     tionaries  of  postal  words that simulate partial
     recognition of the corresponding ZIP Code.
+    digit images included in test  set  that  simulate
     automatic ZIP Code segmentation.  Results on these
     data can be projected to overall ZIP Code recogni-
     tion performance.
+    image format documentation and software included
```

System requirements are a 5.25" CD-ROM drive with software to read ISO-9660 format. For further information, see http://www.cedar.buffalo.edu/Databases/CDROM1/ or send email to Ajay Shekhawat at <ajay@cedar.Buffalo.EDU>

There is also a CEDAR CDROM-2, a database of machine-printed Japanese character images.

9. AI-CD-ROM (see question "Other sources of information")

10. Time series

Santa Fe Competition

Various datasets of time series (to be used for prediction learning problems) are available for anonymous ftp from ftp.santafe.edu in /pub/Time-Series". Data sets include:

- Fluctuations in a far-infrared laser
- Physiological data of patients with sleep apnea;
- High frequency currency exchange rate data;
- Intensity of a white dwarf star;
- J.S. Bachs final (unfinished) fugue from "Die Kunst der Fuge"

Some of the datasets were used in a prediction contest and are described in detail in the book "Time series prediction: Forecasting the future and understanding the past", edited by Weigend/Gershenfield, Proceedings Volume XV in the Santa Fe Institute Studies in the Sciences of Complexity series of Addison Wesley (1994).

M3 Competition

3003 time series from the M3 Competition can be found at http://forecasting.cwru.edu/Data/index.html

The numbers of series of various types are given in the following table:

Interval	Micro	Industry	Macro	Finance	Demog	Other	Total
Yearly	146	102	83	58	245	11	645
Quarterly	204	83	336	76	57	0	756
Monthly	474	334	312	145	111	52	1428
Other	4	0	0	29	0	141	174
Total	828	519	731	308	413	204	3003

Rob Hyndman's Time Series Data Library

A collection of over 500 time series on subjects including agriculture, chemistry, crime, demography, ecology, economics & finance, health, hydrology & meteorology, industry, physics, production, sales, simulated series, sport, transport & tourism, and tree-rings can be found at http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/

11. Financial data

http://chart.yahoo.com/d?s=

http://www.chdwk.com/data/index.html

12. USENIX Faces

The USENIX faces archive is a public database, accessible by ftp, that can be of use to people working in the fields of human face recognition, classification and the like. It currently contains 5592 different faces (taken at USENIX conferences) and is updated twice each year. The images are mostly 96x128 greyscale frontal images and are stored in ascii files in a way that makes it easy to convert them to any usual graphic format (GIF, PCX, PBM etc.). Source code for viewers, filters, etc. is provided. Each image file takes approximately 25K.

For further information, see http://facesaver.usenix.org/

According to the archive administrator, Barbara L. Dijker (barb.dijker@labyrinth.com), there is no restriction to use them. However, the image files are stored in separate directories corresponding to the Internet site to which the person represented in the image belongs, with each directory containing a small number of images (two in the average). This makes it difficult to retrieve by ftp even a small part of the database, as you have to get each one individually.
A solution, as Barbara proposed me, would be to compress the whole set of images (in separate files of, say, 100 images) and maintain them as a specific archive for research on face processing, similar to the ones that already exist for fingerprints and others. The whole compressed database would take some 30 megabytes of disk space. I encourage anyone willing to host this database in his/her site, available for anonymous ftp, to contact her for details (unfortunately I don't have the resources to set up such a site).

Please consider that UUNET has graciously provided the ftp server for the FaceSaver archive and may discontinue that service if it becomes a burden. This means that people should not download more than maybe 10 faces at a time from uunet.

A last remark: each file represents a different person (except for isolated cases). This makes the database quite unsuitable for training neural networks, since for proper generalisation several instances of the same subject are required. However, it is still useful for use as testing set on a trained network.

13. Linguistic Data Consortium

The Linguistic Data Consortium (URL: http://www.ldc.upenn.edu/ldc/noframe.html) is an open consortium of universities, companies and government research laboratories. It creates, collects and distributes speech and text databases, lexicons, and other resources for research and development purposes. The University of Pennsylvania is the LDC's host institution. The LDC catalog includes pronunciation lexicons, varied lexicons, broadcast speech, microphone speech, mobile-radio speech, telephone speech, broadcast text, conversation text, newswire text, parallel text, and varied text, at widely varying fees.

```
Linguistic Data Consortium
University of Pennsylvania
3615 Market Street, Suite 200
Philadelphia, PA 19104-2608
Tel (215) 898-0464 Fax (215) 573-2175
Email: ldc@ldc.upenn.edu
```

14. Otago Speech Corpus

The Otago Speech Corpus contains speech samples in RIFF WAVE format that can be downloaded from http://divcom.otago.ac.nz/infosci/kel/software/RICBIS/hyspeech_main.html

15. Astronomical Time Series

Prepared by Paul L. Hertz (Naval Research Laboratory) & Eric D. Feigelson (Pennsyvania State University):

- Detection of variability in photon counting observations 1 (QSO1525+337)
- Detection of variability in photon counting observations 2 (H0323+022)
- Detection of variability in photon counting observations 3 (SN1987A)
- Detecting orbital and pulsational periodicities in stars 1 (binaries)
- Detecting orbital and pulsational periodicities in stars 2 (variables)
- Cross-correlation of two time series 1 (Sun)
- Cross-correlation of two time series 2 (OJ287)
- Periodicity in a gamma ray burster (GRB790305)
- Solar cycles in sunspot numbers (Sun)
- Deconvolution of sources in a scanning operation (HEAO A-1)
- Fractal time variability in a seyfert galaxy (NGC5506)
- Quasi-periodic oscillations in X-ray binaries (GX5-1)
- Deterministic chaos in an X-ray pulsar? (Her X-1)

URL: http://xweb.nrl.navy.mil/www_hertz/timeseries/timeseries.html

16. Miscellaneous Images

The USC-SIPI Image Database: http://sipi.usc.edu/services/database/Database.html

CityU Image Processing Lab: http://www.image.cityu.edu.hk/images/database.html

Center for Image Processing Research: http://cipr.rpi.edu/

Computer Vision Test Images: http://www.cs.cmu.edu:80/afs/cs/project/cil/ftp/html/v-images.html

Lenna 97: A Complete Story of Lenna: http://www.image.cityu.edu.hk/images/lenna/Lenna97.html

17. StatLib

The StatLib repository at http://lib.stat.cmu.edu/ at Carnegie Mellon University has a large collection of data sets, many of which can be used with NNs.

Next part is part 5 (of 7). Previous part is part 3.

```
Archive-name: ai-faq/neural-nets/part5
Last-modified: 2002-05-01
URL: ftp://ftp.sas.com/pub/neural/FAQ5.html
Maintainer: saswss@unx.sas.com (Warren S. Sarle)
```


This is part 5 (of 7) of a monthly posting to the Usenet newsgroup comp.ai.neural-nets. See the part 1 of this posting for full information what it is all about.

========== Questions ==========

Subject: Source code on the web?

The following URLs are reputed to have source code for NNs. Use at your own risk.

- C/C++
  http://www.generation5.org/xornet.shtml
  http://www.netwood.net/~edwin/Matrix/
  http://www.netwood.net/~edwin/svmt/
  http://www.geocities.com/Athens/Agora/7256/c-plus-p.html
  http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/faces.html
  http://www.cog.brown.edu/~rodrigo/neural_nets_library.html
  http://www.agt.net/public/bmarshal/aiparts/aiparts.htm
  http://www.geocities.com/CapeCanaveral/1624/
  http://www.neuroquest.com/ or http://www.grobe.org/LANE
  http://www.neuro-fuzzy.de/
  http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/neural/systems/cascor/
  http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/neural/systems/qprop/
  http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/neural/systems/rcc/
  etc.

- Java
  http://rfhs8012.fh-regensburg.de/~saj39122/jfroehl/diplom/e-index.html
  http://neuron.eng.wayne.edu/software.html
  http://www.aist.go.jp/NIBH/~b0616/Lab/Links.html
  http://www.aist.go.jp/NIBH/~b0616/Lab/BSOM1/
  http://www.neuroinformatik.ruhr-uni-bochum.de/ini/PEOPLE/loos
  http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/DemoGNG/GNG.html
  http://www.isbiel.ch/I/Projects/janet/index.html
  http://www.born-again.demon.nl/software.html
  http://www.patol.com/java/NN/index.html
  http://www-isis.ecs.soton.ac.uk/computing/neural/laboratory/laboratory.html
  http://www.neuro-fuzzy.de/
  http://sourceforge.net/projects/joone
  http://www.geocities.com/aydingurel/neural/
  http://www-eco.enst-bretagne.fr/~phan/emergence/complexe/neuron/mlp.html

- FORTRAN
  http://www.cranfield.ac.uk/public/me/fo941992/mlpcode.htm

If you are using a small computer (PC, Mac, etc.) you may want to have a look at the Central Neural System Electronic Bulletin Board (see question "Other sources of information"). There are lots of small simulator packages. Some of the CNS materials can also be found at http://www.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/areas/neural/cns/0.html

- -

Subject: Freeware and shareware packages for NN simulation?

Since the FAQ maintainer works for a software company, he does not recommend or evaluate software in the FAQ. The descriptions below are provided by the developers or distributors of the software.

Note for future submissions: Please restrict product descriptions to a maximum of 60 lines of 72 characters, in either plain-text format or, preferably, HTML format. If you include the standard header (name, company, address, etc.), you need not count the header in the 60 line maximum. Please confine your HTML to features that are supported by primitive browsers, especially NCSA Mosaic 2.0; avoid tables, for example--use <pre> instead. Try to make the descriptions objective, and avoid making implicit or explicit assertions about competing products, such as "Our product is the *only* one that does so-and-so." The FAQ maintainer reserves the right to remove excessive marketing hype and to edit submissions to conform to size requirements; if he is in a good mood, he may also correct your spelling and punctuation.

The following simulators are described below:

1. JavaNNS
2. SNNS
3. PDP++

4. Rochester Connectionist Simulator
5. UCLA-SFINX
6. NeurDS
7. PlaNet (formerly known as SunNet)
8. GENESIS
9. Mactivation
10. Cascade Correlation Simulator
11. Quickprop
12. DartNet
13. Aspirin/MIGRAINES
14. ALN Workbench
15. Uts (Xerion, the sequel)
16. Multi-Module Neural Computing Environment (MUME)
17. LVQ_PAK, SOM_PAK
18. Nevada Backpropagation (NevProp)
19. Fuzzy ARTmap
20. PYGMALION
21. Basis-of-AI-NN Software
22. Matrix Backpropagation
23. BIOSIM
24. FuNeGen
25. NeuDL -- Neural-Network Description Language
26. NeoC Explorer
27. AINET
28. DemoGNG
29. Trajan 2.1 Shareware
30. Neural Networks at your Fingertips
31. NNFit
32. Nenet v1.0
33. Machine Consciousness Toolbox
34. NICO Toolkit (speech recognition)
35. SOM Toolbox for Matlab 5
36. FastICA package for MATLAB
37. NEXUS: Large-scale biological simulations
38. Netlab: Neural network software for Matlab
39. NuTank
40. Lens
41. Joone: Java Object Oriented Neural Engine
42. NV: Neural Viewer
43. EasyNN
44. Multilayer Perceptron - A Java Implementation

See also http://www.emsl.pnl.gov:2080/proj/neuron/neural/systems/shareware.html

1. **JavaNNS: Java Neural Network Simulator**

http://www-ra.informatik.uni-tuebingen.de/forschung/JavaNNS/welcome_e.html
JavaNNS is the successor to SNNS. JavaNNS is based on the SNNS computing kernel, but has a newly developed graphical user interface written in Java set on top of it. Hence compatibility with SNNS is achieved while platform-independence is increased.

In addition to SNNS features, JavaNNS offers the capability of linking HTML browsers to it. This provides for accessing the user manual (available in HTML) or, optionally, a reference coursebook on neural networks directly from within the program.

JavaNNS is available for Windows NT / Windows 2000, Solaris and RedHat Linux. Additional ports are planed. JavaNNS is freely available and can be downloaded from the URL shown above.

Contact: Igor Fischer, Phone: +49 7071 29-77176, fischer@informatik.uni-tuebingen.de

2. SNNS 4.2

SNNS (Stuttgart Neural Network Simulator) is a software simulator for neural networks on Unix workstations developed at the Institute for Parallel and Distributed High Performance Systems (IPVR) at the University of Stuttgart. The goal of the SNNS project is to create an efficient and flexible simulation environment for research on and application of neural nets.

The SNNS simulator consists of two main components:

1. simulator kernel written in C
2. graphical user interface under X11R4 or X11R5

The simulator kernel operates on the internal network data structures of the neural nets and performs all operations of learning and recall. It can also be used without the other parts as a C program embedded in custom applications. It supports arbitrary network topologies and, like RCS, supports the concept of sites. SNNS can be extended by the user with user defined activation functions, output functions, site functions and learning procedures, which are written as simple C programs and linked to the simulator kernel. C code can be generated from a trained network.

Currently the following network architectures and learning procedures are included:

- Backpropagation (BP) for feedforward networks
  - vanilla (online) BP
  - BP with momentum term and flat spot elimination
  - batch BP
  - chunkwise BP
- Counterpropagation
- Quickprop
- Backpercolation 1
- RProp
- Generalized radial basis functions (RBF)
- ART1
- ART2
- ARTMAP
- Cascade Correlation
- Dynamic LVQ
- Backpropagation through time (for recurrent networks)

- Quickprop through time (for recurrent networks)
- Self-organizing maps (Kohonen maps)
- TDNN (time-delay networks) with Backpropagation
- Jordan networks
- Elman networks and extended hierarchical Elman networks
- Associative Memory
- TACOMA

The graphical user interface XGUI (X Graphical User Interface), built on top of the kernel, gives a 2D and a 3D graphical representation of the neural networks and controls the kernel during the simulation run. In addition, the 2D user interface has an integrated network editor which can be used to directly create, manipulate and visualize neural nets in various ways.

SNNSv4.1 has been tested on SUN SparcSt ELC,IPC (SunOS 4.1.2, 4.1.3), SUN SparcSt 2 (SunOS 4.1.2), SUN SparcSt 5, 10, 20 (SunOS 4.1.3, 5.2), DECstation 3100, 5000 (Ultrix V4.2), DEC Alpha AXP 3000 (OSF1 V2.1), IBM-PC 80486, Pentium (Linux), IBM RS 6000/320, 320H, 530H (AIX V3.1, AIX V3.2), HP 9000/720, 730 (HP-UX 8.07), and SGI Indigo 2 (IRIX 4.0.5, 5.3).

The distributed kernel can spread one learning run over a workstation cluster.

SNNS web page: http://www-ra.informatik.uni-tuebingen.de/SNNS
Ftp server: ftp://ftp.informatik.uni-tuebingen.de/pub/SNNS

- SNNSv4.1.Readme
- SNNSv4.1.tar.gz (1.4 MB, Source code)
- SNNSv4.1.Manual.ps.gz (1 MB, Documentation)

Mailing list: http://www-ra.informatik.uni-tuebingen.de/SNNS/about-ml.html

3. PDP++

URL: http://www.cnbc.cmu.edu/PDP++/PDP++.html

The PDP++ software is a neural-network simulation system written in C++. It represents the next generation of the PDP software released with the McClelland and Rumelhart "Explorations in Parallel Distributed Processing Handbook", MIT Press, 1987. It is easy enough for novice users, but very powerful and flexible for research use. PDP++ is featured in a new textbook, *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain*, by Randall C. O'Reilly and Yuko Munakata, MIT Press, 2000.

Supported algorithms include:

- Feedforward and recurrent error backpropagation. Recurrent BP includes continuous, real-time models, and Almeida-Pineda.
- Constraint satisfaction algorithms and associated learning algorithms including Boltzmann Machine, Hopfield models, mean-field networks (DBM), Interactive Activation and Competition (IAC), and continuous stochastic networks.
- Self-organizing learning including Competitive Learning, Soft Competitive Learning, simple Hebbian, and Self-organizing Maps ("Kohonen Nets").
- Mixtures-of-experts using backpropagation experts, EM updating, and a SoftMax gating module.
- Leabra algorithm that combines error-driven and Hebbian learning with k-Winners-Take-All

inhibitory competition.

The software can be obtained by anonymous ftp from:

- ftp://grey.colorado.edu/pub/oreilly/pdp++ or
- ftp://cnbc.cmu.edu/pub/pdp++/ or
- ftp://unix.hensa.ac.uk/mirrors/pdp++/

4. Rochester Connectionist Simulator

A versatile simulator program for arbitrary types of neural nets. Comes with a backprop package and a X11/Sunview interface. Available via anonymous FTP from ftp://ftp.cs.rochester.edu/pub/packages/simulator/simulator_v4.2.tar.Z There's also a patch available from ftp://ftp.cs.rochester.edu/pub/packages/simulator/simulator_v4.2.patch.1

5. UCLA-SFINX

The UCLA-SFINX, a "neural" network simulator is now in public domain. UCLA-SFINX (Structure and Function In Neural connec- tions) is an interactive neural network simulation environment designed to provide the investigative tools for studying the behavior of various neural structures. It was designed to easily express and simulate the highly regular patterns often found in large networks, but it is also general enough to model parallel systems of arbitrary interconnectivity. For more information, see http://decus.acornsw.com/vs0121/AISIG/F90/NETS/UCLA_SIM.TXT

6. NeurDS

Neural Design and Simulation System. This is a general purpose tool for building, running and analysing Neural Network Models in an efficient manner. NeurDS will compile and run virtually any Neural Network Model using a consistent user interface that may be either window or "batch" oriented. HP-UX 8.07 source code is available from http://hpux.u-aizu.ac.jp/hppd/hpux/NeuralNets/NeurDS-3.1/ or http://askdonna.ask.uni-karlsruhe.de/hppd/hpux/NeuralNets/NeurDS-3.1/

7. PlaNet5.7 (formerly known as SunNet)

A popular connectionist simulator with versions to run under X Windows, and non-graphics terminals created by Yoshiro Miyata (Chukyo Univ., Japan). 60-page User's Guide in Postscript. Send any questions to miyata@sccs.chukyo-u.ac.jp Available for anonymous ftp from ftp.ira.uka.de as /pub/neuron/PlaNet5.7.tar.gz (800 kb)

8. GENESIS

GENESIS 2.0 (GEneral NEural SImulation System) is a general purpose simulation platform which was developed to support the simulation of neural systems ranging from complex models of single neurons to simulations of large networks made up of more abstract neuronal components. Most current GENESIS applications involve realistic simulations of biological neural systems. Although the software can also model more abstract networks, other simulators are more suitable for backpropagation and similar connectionist modeling. Runs on most Unix platforms. Graphical front end XODUS. Parallel version for networks of workstations, symmetric multiprocessors, and MPPs also available. Further information via WWW at

http://www.genesis-sim.org/GENESIS/.

9. Mactivation

A neural network simulator for the Apple Macintosh. Available for ftp from ftp.cs.colorado.edu as /pub/cs/misc/Mactivation-3.3.sea.hqx

10. Cascade Correlation Simulator

A simulator for Scott Fahlman's Cascade Correlation algorithm. Available for ftp from ftp.cs.cmu.edu in directory /afs/cs/project/connect/code/supported as the file cascor-v1.2.shar (223 KB) There is also a version of recurrent cascade correlation in the same directory in file rcc1.c (108 KB).

11. Quickprop

A variation of the back-propagation algorithm developed by Scott Fahlman. A simulator is available in the same directory as the cascade correlation simulator above in file nevprop1.16.shar (137 KB)
(There is also an obsolete simulator called quickprop1.c (21 KB) in the same directory, but it has been superseeded by NevProp. See also the description of NevProp below.)

12. DartNet

DartNet is a Macintosh-based backpropagation simulator, developed at Dartmouth by Jamshed Bharucha and Sean Nolan as a pedagogical tool. It makes use of the Mac's graphical interface, and provides a number of tools for building, editing, training, testing and examining networks. This program is available by anonymous ftp from ftp.dartmouth.edu as /pub/mac/dartnet.sit.hqx (124 KB).

13. Aspirin/MIGRAINES

Aspirin/MIGRAINES 6.0 consists of a code generator that builds neural network simulations by reading a network description (written in a language called "Aspirin") and generates a C simulation. An interface (called "MIGRAINES") is provided to export data from the neural network to visualization tools. The system has been ported to a large number of platforms. The goal of Aspirin is to provide a common extendible front-end language and parser for different network paradigms. The MIGRAINES interface is a terminal based interface that allows you to open Unix pipes to data in the neural network. Users can display the data using either public or commercial graphics/analysis tools. Example filters are included that convert data exported through MIGRAINES to formats readable by Gnuplot 3.0, Matlab, Mathematica, and xgobi.

The software is available from http://www.elegant-software.com/software/aspirin/

14. ALN Workbench (a spreadsheet for Windows)

ALNBench is a free spreadsheet program for MS-Windows (NT, 95) that allows the user to import training and test sets and predict a chosen column of data from the others in the training set. It is an easy-to-use program for research, education and evaluation of ALN technology. Anyone who can use a spreadsheet can quickly understand how to use it. It facilitates interactive access to the power of the Dendronic Learning Engine (DLE), a product in commercial use.

An ALN consists of linear functions with adaptable weights at the leaves of a tree of maximum and minimum operators. The tree grows automatically during training: a linear piece splits if its error is too high. The function computed by an ALN is piecewise linear and continuous. It can learn to approximate any continuous function to arbitrarily high accuracy.

Parameters allow the user to input knowledge about a function to promote good generalization. In particular, bounds on the weights of the linear functions can be directly enforced. Some parameters are chosen automatically in standard mode, and are under user control in expert mode.

The program can be downloaded from http://www.dendronic.com/main.htm

For further information please contact:

William W. Armstrong PhD, President
Dendronic Decisions Limited
3624 - 108 Street, NW
Edmonton, Alberta,
Canada T6J 1B4
Email: arms@dendronic.com
URL: http://www.dendronic.com/
Tel. +1 403 421 0800
(Note: The area code 403 changes to 780 after Jan. 25, 1999)

15. Uts (Xerion, the sequel)

Uts is a portable artificial neural network simulator written on top of the Tool Control Language (Tcl) and the Tk UI toolkit. As result, the user interface is readily modifiable and it is possible to simultaneously use the graphical user interface and visualization tools and use scripts written in Tcl. Uts itself implements only the connectionist paradigm of linked units in Tcl and the basic elements of the graphical user interface. To make a ready-to-use package, there exist modules which use Uts to do back-propagation (tkbp) and mixed em gaussian optimization (tkmxm). Uts is available in ftp.cs.toronto.edu in directory /pub/xerion.

16. Multi-Module Neural Computing Environment (MUME)

MUME is a simulation environment for multi-modules neural computing. It provides an object oriented facility for the simulation and training of multiple nets with various architectures and learning algorithms. MUME includes a library of network architectures including feedforward, simple recurrent, and continuously running recurrent neural networks. Each architecture is supported by a variety of learning algorithms. MUME can be used for large scale neural network simulations as it provides support for learning in multi-net environments. It also provide pre- and post-processing facilities. For more information, see http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/neural/systems/mume/0.html

17. LVQ_PAK, SOM_PAK

These are packages for Learning Vector Quantization and Self-Organizing Maps, respectively. They have been built by the LVQ/SOM Programming Team of the Helsinki University of Technology, Laboratory of Computer and Information Science, Rakentajanaukio 2 C, SF-02150 Espoo, FINLAND There are versions for Unix and MS-DOS available from http://nucleus.hut.fi/nnrc/nnrc-programs.html

18. Nevada Backpropagation (NevProp)

NevProp, version 3, is a relatively easy-to-use, feedforward backpropagation multilayer perceptron simulator-that is, statistically speaking, a multivariate nonlinear regression program. NevProp3 is distributed for free under the terms of the GNU Public License and can be downloaded from http://brain.cs.unr.edu/publications/NevProp.zip and http://brain.cs.unr.edu/publications/NevPropManual.pdf

The program is distributed as C source code that should compile and run on most platforms. In addition, precompiled executables are available for Macintosh and DOS platforms. Limited support is available from Phil Goodman (goodman@unr.edu), University of Nevada Center for Biomedical Research.

MAJOR FEATURES OF NevProp3 OPERATION (* indicates feature new in version 3)

1. Character-based interface common to the UNIX, DOS, and Macintosh platforms.
2. Command-line argument format to efficiently initiate NevProp3. For Generalized Nonlinear Modeling (GNLM) mode, beginners may opt to use an interactive interface.
3. Option to pre-standardize the training data (z-score or forced range*).
4. Option to pre-impute missing elements in training data (case-wise deletion, or imputation with mean, median, random selection, or k-nearest neighbor).*
5. Primary error (criterion) measures include mean square error, hyperbolic tangent error, and log likelihood (cross-entropy), as penalized an unpenalized values.
6. Secondary measures include ROC-curve area (c-index), thresholded classification, R-squared and Nagelkerke R-squared. Also reported at intervals are the weight configuration, and the sum of square weights.
7. Allows simultaneous use of logistic (for dichotomous outputs) and linear output activation functions (automatically detected to assign activation and error function).*
8. 1-of-N (Softmax)* and M-of-N options for binary classification.
9. Optimization options: flexible learning rate (fixed global adaptive, weight-specific, quickprop), split learn rate (inversely proportional to number of incoming connections), stochastic (case-wise updating), sigmoidprime offset (to prevent locking at logistic tails).
10. Regularization options: fixed weight decay, optional decay on bias weights, Bayesian hyperpenalty* (partial and full Automatic Relevance Determination-also used to select important predictors), automated early stopping (full dataset stopping based on multiple subset cross-validations) by error criterion.
11. Validation options: upload held-out validation test set; select subset of outputs for joint summary statistics;* select automated bootstrapped modeling to correct optimistically biased summary statistics (with standard deviations) without use of hold-out.
12. Saving predictions: for training data and uploaded validation test set, save file with identifiers, true targets, predictions, and (if bootstrapped models selected) lower and upper 95% confidence limits* for each prediction.
13. Inference options: determination of the mean predictor effects and level effects (for multilevel predictor variables); confidence limits within main model or across bootstrapped models.*
14. ANN-kNN (k-nearest neighbor) emulation mode options: impute missing data elements and save to new data file; classify test data (with or without missing elements) using ANN-kNN model trained on data with or without missing elements (complete ANN-based expectation maximization).*
15. AGE (ANN-Gated Ensemble) options: adaptively weight predictions (any scale of scores) obtained from multiple (human or computational) "experts"; validate on new prediction sets; optional internal prior-probability expert.*

19. Fuzzy ARTmap

This is just a small example program. Available for anonymous ftp from park.bu.edu [128.176.121.56] ftp://cns-ftp.bu.edu/pub/fuzzy-artmap.tar.Z (44 kB).

20. PYGMALION

This is a prototype that stems from an ESPRIT project. It implements back-propagation, self organising map, and Hopfield nets. Avaliable for ftp from ftp.funet.fi [128.214.248.6] as /pub/sci/neural/sims/pygmalion.tar.Z (1534 kb). (Original site is imag.imag.fr: archive/pygmalion/pygmalion.tar.Z).

21. Basis-of-AI-NN Software

Non-GUI DOS and UNIX source code, DOS binaries and examples are available in the following different program sets and the backprop package has a Windows 3.x binary and a Unix/Tcl/Tk version:

```
[backprop, quickprop, delta-bar-delta, recurrent networks],
[simple clustering, k-nearest neighbor, LVQ1, DSM],
[Hopfield, Boltzman, interactive activation network],
[interactive activation network],
[feedforward counterpropagation],
[ART I],
[a simple BAM] and
[the linear pattern classifier]
```

For details see: http://www.dontveter.com/nnsoft/nnsoft.html

An improved professional version of backprop is also available; see Part 6 of the FAQ.

Questions to: Don Tveter, don@dontveter.com

22. Matrix Backpropagation

MBP (Matrix Back Propagation) is a very efficient implementation of the back-propagation algorithm for current-generation workstations. The algorithm includes a per-epoch adaptive technique for gradient descent. All the computations are done through matrix multiplications and make use of highly optimized C code. The goal is to reach almost peak-performances on RISCs with superscalar capabilities and fast caches. On some machines (and with large networks) a 30-40x speed-up can be measured with respect to conventional implementations. The software is available by anonymous ftp from ftp.esng.dibe.unige.it as /neural/MBP/MBPv1.1.tar.Z (Unix version), or /neural/MBP/MBPv11.zip (PC version)., For more information, contact Davide Anguita (anguita@dibe.unige.it).

23. BIOSIM

BIOSIM is a biologically oriented neural network simulator. Public domain, runs on Unix (less powerful PC-version is available, too), easy to install, bilingual (german and english), has a GUI (Graphical User Interface), designed for research and teaching, provides online help facilities, offers controlling interfaces, batch version is available, a DEMO is provided.

REQUIREMENTS (Unix version): X11 Rel. 3 and above, Motif Rel 1.0 and above, 12 MB of physical

memory, recommended are 24 MB and more, 20 MB disc space. REQUIREMENTS (PC version): PC-compatible with MS Windows 3.0 and above, 4 MB of physical memory, recommended are 8 MB and more, 1 MB disc space.

Four neuron models are implemented in BIOSIM: a simple model only switching ion channels on and off, the original Hodgkin-Huxley model, the SWIM model (a modified HH model) and the Golowasch-Buchholz model. Dendrites consist of a chain of segments without bifurcation. A neural network can be created by using the interactive network editor which is part of BIOSIM. Parameters can be changed via context sensitive menus and the results of the simulation can be visualized in observation windows for neurons and synapses. Stochastic processes such as noise can be included. In addition, biologically orientied learning and forgetting processes are modeled, e.g. sensitization, habituation, conditioning, hebbian learning and competitive learning. Three synaptic types are predefined (an excitatatory synapse type, an inhibitory synapse type and an electrical synapse). Additional synaptic types can be created interactively as desired.

Available for ftp from ftp.uni-kl.de in directory /pub/bio/neurobio: Get /pub/bio/neurobio/biosim.readme (2 kb) and /pub/bio/neurobio/biosim.tar.Z (2.6 MB) for the Unix version or /pub/bio/neurobio/biosimpc.readme (2 kb) and /pub/bio/neurobio/biosimpc.zip (150 kb) for the PC version.

Contact:
Stefan Bergdoll
Department of Software Engineering (ZXA/US)
BASF Inc.
D-67056 Ludwigshafen; Germany
bergdoll@zxa.basf-ag.de phone 0621-60-21372 fax 0621-60-43735

24. FuNeGen 1.0

FuNeGen is a MLP based software program to generate fuzzy rule based classifiers. For more information, see http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/areas/fuzzy/systems/funegen/

25. NeuDL -- Neural-Network Description Language

NeuDL is a description language for the design, training, and operation of neural networks. It is currently limited to the backpropagation neural-network model; however, it offers a great deal of flexibility. For example, the user can explicitly specify the connections between nodes and can create or destroy connections dynamically as training progresses. NeuDL is an interpreted language resembling C or C++. It also has instructions dealing with training/testing set manipulation as well as neural network operation. A NeuDL program can be run in interpreted mode or it can be automatically translated into C++ which can be compiled and then executed. The NeuDL interpreter is written in C++ and can be easly extended with new instructions. For more information, see http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/neural/systems/neudl/0.html

26. NeoC Explorer (Pattern Maker included)

The NeoC software is an implementation of Fukushima's Neocognitron neural network. Its purpose is to test the model and to facilitate interactivity for the experiments. Some substantial features: GUI, explorer and tester operation modes, recognition statistics, performance analysis, elements displaying, easy net construction. PLUS, a pattern maker utility for testing ANN: GUI, text file output, transformations. For more information, see http://www.simtel.net/pub/pd/39893.html

27. AINET

AINET is a probabilistic neural network application which runs on Windows 95/NT. It was designed specifically to facilitate the modeling task in all neural network problems. It is lightning fast and can be used in conjunction with many different programming languages. It does not require iterative learning, has no limits in variables (input and output neurons), no limits in sample size. It is not sensitive toward noise in the data. The database can be changed dynamically. It provides a way to estimate the rate of error in your prediction. It has a graphical spreadsheet-like user interface. The AINET manual (more than 100 pages) is divided into: "User's Guide", "Basics About Modeling with the AINET", "Examples", "The AINET DLL library" and "Appendix" where the theoretical background is revealed. You can get a full working copy from: http://www.ainet-sp.si/

28. DemoGNG

This simulator is written in Java and should therefore run without compilation on all platforms where a Java interpreter (or a browser with Java support) is available. It implements the following algorithms and neural network models:

- Hard Competitive Learning (standard algorithm)
- Neural Gas (Martinetz and Schulten 1991)
- Competitive Hebbian Learning (Martinetz and Schulten 1991, Martinetz 1993)
- Neural Gas with Competitive Hebbian Learning (Martinetz and Schulten 1991)
- Growing Neural Gas (Fritzke 1995)

DemoGNG is distributed under the GNU General Public License. It allows to experiment with the different methods using various probability distributions. All model parameters can be set interactively on the graphical user interface. A teach modus is provided to observe the models in "slow-motion" if so desired. It is currently **not** possible to experiment with user-provided data, so the simulator is useful basically for demonstration and teaching purposes and as a sample implementation of the above algorithms.

DemoGNG can be accessed most easily at http://www.neuroinformatik.ruhr-uni-bochum.de/ in the file /ini/VDM/research/gsn/DemoGNG/GNG.html where it is embedded as Java applet into a Web page and is downloaded for immediate execution when you visit this page. An accompanying paper entitled "Some competitive learning methods" describes the implemented models in detail and is available in html at the same server in the directory ini/VDM/research/gsn/JavaPaper/.

It is also possible to download the complete source code and a Postscript version of the paper via anonymous ftp from ftp.neuroinformatik.ruhr-uni-bochum [134.147.176.16] in directory /pub/software/NN/DemoGNG/. The software is in the file DemoGNG-1.00.tar.gz (193 KB) and the paper in the file sclm.ps.gz (89 KB). There is also a README file (9 KB). Please send any comments and questions to demogng@neuroinformatik.ruhr-uni-bochum.de which will reach Hartmut Loos who has written DemoGNG as well as Bernd Fritzke, the author of the accompanying paper.

29. Trajan 2.1 Shareware

Trajan 2.1 Shareware is a Windows-based Neural Network simulation package. It includes support for the two most popular forms of Neural Network: Multilayer Perceptrons with Back Propagation and Kohonen networks.

Trajan 2.1 Shareware concentrates on ease-of-use and feedback. It includes Graphs, Bar Charts and Data Sheets presenting a range of Statistical feedback in a simple, intuitive form. It also features extensive on-line

Help.

The Registered version of the package can support very large networks (up to 128 layers with up to 8,192 units each, subject to memory limitations in the machine), and allows simple Cut and Paste transfer of data to/from other Windows-packages, such as spreadsheet programs. The Unregistered version features limited network size and no Clipboard Cut-and-Paste.

There is also a Professional version of Trajan 2.1, which supports a wider range of network models, training algorithms and other features.

See Trajan Software's Home Page at http://www.trajan-software.demon.co.uk for further details, and a free copy of the Shareware version.

Alternatively, email andrew@trajan-software.demon.co.uk for more details.

30. Neural Networks at your Fingertips

"Neural Networks at your Fingertips" is a package of ready-to-reuse neural network simulation source code which was prepared for educational purposes by Karsten Kutza. The package consists of eight programs, each of which implements a particular network architecture together with an embedded example application from a typical application domain.
Supported network architectures are

- Adaline,
- Backpropagation,
- Hopfield Model,
- Bidirectional Associative Memory,
- Boltzmann Machine,
- Counterpropagation,
- Self-Organizing Map, and
- Adaptive Resonance Theory.

The applications demonstrate use of the networks in various domains such as pattern recognition, time-series forecasting, associative memory, optimization, vision, and control and include e.g. a sunspot prediction, the traveling salesman problem, and a pole balancer.
The programs are coded in portable, self-contained ANSI C and can be obtained from the web pages at http://www.geocities.com/CapeCanaveral/1624.

31. NNFit

NNFit (Neural Network data Fitting) is a user-friendly software that allows the development of empirical correlations between input and output data. Multilayered neural models have been implemented using a quasi-newton method as learning algorithm. Early stopping method is available and various tables and figures are provided to evaluate fitting performances of the neural models. The software is available for most of the Unix platforms with X-Windows (IBM-AIX, HP-UX, SUN, SGI, DEC, Linux). Informations, manual and executable codes (english and french versions) are available at http://www.gch.ulaval.ca/~nnfit
Contact: Bernard P.A. Grandjean, department of chemical engineering,
Laval University; Sainte-Foy (Quibec) Canada G1K 7P4;
grandjean@gch.ulaval.ca

32. Nenet v1.0

Nenet v1.0 is a 32-bit Windows 95 and Windows NT 4.0 application designed to facilitate the use of a Self-Organizing Map (SOM) algorithm.

The major motivation for Nenet was to create a user-friendly SOM algorithm tool with good visualization capabilities and with a GUI allowing efficient control of the SOM parameters. The use scenarios have stemmed from the user's point of view and a considerable amount of work has been placed on the ease of use and versatile visualization methods.

With Nenet, all the basic steps in map control can be performed. In addition, Nenet also includes some more exotic and involved features especially in the area of visualization.

Features in Nenet version 1.0:

- Implements the standard Kohonen SOM algorithm
- Supports 2 common data preprocessing methods
- 5 different visualization methods with rectangular or hexagonal topology
- Capability to animate both train and test sequences in all visualization methods
- Labelling
  - Both neurons and parameter levels can be labelled
  - Provides also autolabelling
- Neuron values can be inspected easily
- Arbitrary selection of parameter levels can be visualized with Umatrix simultaneously
- Multiple views can be opened on the same map data
- Maps can be printed
- Extensive help system provides fast and accurate online help
- SOM_PAK compatible file formats
- Easy to install and uninstall
- Conforms to the common Windows 95 application style - all functionality in one application

Nenet web site is at: http://www.mbnet.fi/~phodju/nenet/Nenet/General.html The web site contains further information on Nenet and also the downloadable Nenet files (3 disks totalling about 3 Megs)

If you have any questions whatsoever, please contact: Nenet-Team@hut.fi or phassine@cc.hut.fi

33. Machine Consciousness Toolbox

See listing for Machine Consciousness Toolbox in part 6 of the FAQ.

34. NICO Toolkit (speech recognition)

```
      Name: NICO Artificial Neural Network Toolkit
    Author: Nikko Strom
   Address: Speech, Music and Hearing, KTH, S-100 44, Stockholm, Sweden
     Email: nikko@speech.kth.se
       URL: http://www.speech.kth.se/NICO/index.html
 Platforms: UNIX, ANSI C; Source code tested on: HPUX, SUN Solaris, Linux
     Price: Free
```

The NICO Toolkit is an artificial neural network toolkit designed and optimized for automatic speech recognition applications. Networks with both recurrent connections and time-delay windows are easily

constructed. The network topology is very flexible -- any number of layers is allowed and layers can be arbitrarily connected. Sparse connectivity between layers can be specified. Tools for extracting input-features from the speech signal are included as well as tools for computing target values from several standard phonetic label-file formats.

Algorithms:

- Back-propagation through time,
- Speech feature extraction (Mel cepstrum coefficients, filter-bank)

35. SOM Toolbox for Matlab 5

SOM Toolbox, a shareware Matlab 5 toolbox for data analysis with self-organizing maps is available at the URL http://www.cis.hut.fi/projects/somtoolbox/. If you are interested in practical data analysis and/or self-organizing maps and have Matlab 5 in your computer, be sure to check this out!

Highlights of the SOM Toolbox include the following:

- Tools for all the stages of data analysis: besides the basic SOM training and visualization tools, the package includes also tools for data preprocessing and model validation and interpretation.
- Graphical user interface (GUI): the GUI first guides the user through the initialization and training procedures, and then offers a variety of different methods to visualize the data on the trained map.
- Modular programming style: the Toolbox code utilizes Matlab structures, and the functions are constructed in a modular manner, which makes it convenient to tailor the code for each user's specific needs.
- Advanced graphics: building on the Matlab's strong graphics capabilities, attractive figures can be easily produced.
- Compatibility with SOM_PAK: import/export functions for SOM_PAK codebook and data files are included in the package.
- Component weights and names: the input vector components may be given different weights according to their relative importance, and the components can be given names to make the figures easier to read.
- Batch or sequential training: in data analysis applications, the speed of training may be considerably improved by using the batch version.
- Map dimension: maps may be N-dimensional (but visualization is not supported when N > 2).

36. FastICA package for MATLAB

The FastICA algorithm for independent component analysis.

Independent component analysis, or ICA, is neural network or signal processing technique that represents a multidimensional random vector as a linear combination of nongaussian random variables ('independent components') that are as independent as possible. ICA is a nongaussian version of factor analysis, and somewhat similar to principal component analysis. ICA has many applications in data analysis, source separation, and feature extraction.

The FastICA algorithm is a computationally optimized method for performing the estimation of ICA. It uses a fixed-point iteration scheme that has been found in independent experiments to be 10-100 times faster than conventional gradient descent methods for ICA. Another advantage of the FastICA algorithm is that it can be used to estimate the independent components one-by-one, as in projection pursuit, which is very practical in

exploratory data analysis.

The FastICA package for MATLAB (versions 5 or 4) is freeware package with a graphical user interface that implements the fixed-point algorithm for ICA. The package is available on the Web at http://www.cis.hut.fi/projects/ica/fastica/.
Email contact: Aapo Hyvarinen <Aapo.Hyvarinen@hut.fi>

37. NEXUS: Large-scale biological simulations

Large-scale biological neural network simulation engine. Includes automated network construction tool that allows extremely complex networks to be generated according to user-supplied architectural specifications.

The network engine is an attempt at creating a biological neural network simulator. It consists of a C++ class, called "network". A network object houses a set of objects of another C++ class, called "neuron". The neuron class is a detailed functional simulation of a neuron (i.e. the actual chemical processes that lead to a biological neuron's behavior are not modeled explicitly, but the behavior itself is). The simulation of the neuron is handled entirely by the neuron class. The network class coordinates the functioning of the neurons that make up the neural network, as well as providing addressing services that allow the neurons to interact. It is also responsible for facilitating the interface of the neural network it houses onto any existing software into which the neural network is to be integrated.

Since a simulated neural network consisting of a large number of heavily interconnected neurons is extremely difficult to generate manually, NEXUS was developed. To create a network with NEXUS, one need only describe the network in general terms, in terms of groups of sets of specifically arranged neurons, and how the groups interface onto each other and onto themselves. This information constitutes a network architecture descriptor. A network architecture descriptor is read by NEXUS, and NEXUS uses the information to generate a network, building all the neurons and connecting them together appropriately. This system is analogous to nature's brain construction system. For example, human brains, in general, are very similar. The basic design is stored in human DNA. Since it is certainly not possible to record information about each neuron and its connections, DNA must instead contain (in some form) what is essentially a set of guidelines, a set of rules about how the brain is to be laid out. These guidelines are used to build the brain, just like NEXUS uses the guidelines set out in the network architecture descriptor to build the simulated neural network.

NEXUS and the network engine have deliberately been engineered to be highly efficient and very compact. Even so, large, complex networks require tremendous amounts of memory and processing power.

The network engine:

- flexible and elegant design; highly customizable simulation parameters; extremely efficient
- throughout, nonlinear magnitude decay modeling
- dendritic tree complexity and network connection density limited only by the computer hardware
- simulation of dendritic logic gate behaviors via a sophisticated excitation thresholding and conduction model
- detailed simulation of backprop, allowing realistic simulation of associated memory formation processes
- simulation of all known postsynaptic memory formation mechanisms (STP, STD, LTP, LTD)
- dynamic presynaptic output pattern modeling, including excitation magnitude dependent output pattern selection
- simulation of all known presynaptic activity-based output modifiers (PPF, PTP, depression)

NEXUS:

- allows networks to be designed concisely and as precisely as is necessary
- makes massively complex large-scale neural network design and construction possible
- allows existing networks to be augmented without disturbing existing network structure
- UNIX and Win32 compatible

URL: http://www.sfu.ca/~loryan/neural.html
Email: Lawrence O. Ryan <loryan@sfu.ca>

38. Netlab: Neural network software for Matlab

http://www.ncrg.aston.ac.uk/netlab/index.html

The Netlab simulation software is designed to provide the central tools necessary for the simulation of theoretically well founded neural network algorithms for use in teaching, research and applications development. It consists of a library of Matlab functions and scripts based on the approach and techniques described in Neural Networks for Pattern Recognition by Christopher M. Bishop, (Oxford University Press, 1995). The functions come with on-line help, and further explanation is available via HTML files.

The Netlab library includes software implementations of a wide range of data analysis techniques. Netlab works with Matlab version 5.0 and higher. It is not compatible with earlier versions of Matlab.

39. NuTank

NuTank stands for NeuralTank. It is educational and entertainment software. In this program one is given the shell of a 2 dimentional robotic tank. The tank has various I/O devices like wheels, whiskers, optical sensors, smell, fuel level, sound and such. These I/O sensors are connected to Neurons. The player/designer uses more Neurons to interconnect the I/O devices. One can have any level of complexity desired (memory limited) and do subsumptive designs. More complex design take slightly more fuel, so life is not free. All movement costs fuel too. One can also tag neuron connections as "adaptable" that adapt their weights in acordance with the target neuron. This allows neurons to learn. The Neuron editor can handle 3 dimention arrays of neurons as single entities with very flexible interconect patterns.

One can then design a scenario with walls, rocks, lights, fat (fuel) sources (that can be smelled) and many other such things. Robot tanks are then introduced into the Scenario and allowed interact or battle it out. The last one alive wins, or maybe one just watches the motion of the robots for fun. While the scenario is running it can be stopped, edited, zoom'd, and can track on any robot.

The entire program is mouse and graphicly based. It uses DOS and VGA and is written in TurboC++. There will also be the ability to download designs to another computer and source code will be available for the core neural simulator. This will allow one to design neural systems and download them to real robots. The design tools can handle three dimentional networks so will work with video camera inputs and such.

NuTank source code is free from http://www.xmission.com/~rkeene/NuTankSrc.ZIP
Contact: Richard Keene; Keene Educational Software
Email: rkeene@xmission.com or r.keene@center7.com

40. Lens

http://www.cs.cmu.edu/~dr/Lens

Lens (the light, efficient network simulator) is a fast, flexible, and customizable neural network package written primarily in C. It currently handles standard backpropagation networks, simple recurrent (including Jordan and Elman) and fully recurrent nets, deterministic Boltzmann machines, self-organizing maps, and interactive-activation models.

Lens runs under Windows as well as a variety of Unix platforms. It includes a graphical interface and an embedded script language (Tcl). The key to the speed of Lens is its use of tight inner-loops that minimize memory references when traversing links. Frequently accessed values are stored in contiguous memory to achieve good cache performance. It is also able to do batch-level parallel training on multiple processors.

Because it is recognized that no simulator will satisfy sophisticated users out of the box, Lens was designed to facilitate code modification. Users can create and register such things as new network or group types, new weight update algorithms, or new shell commands without altering the main body of code. Therefore, modifications can be easily transferred to new releases.

Lens is available free-of-charge to those conducting research at academic or non-profit institutions. Other users should contact Douglas Rohde for licensing information at dr+lens@cs.cmu.edu.

41. Joone: Java Object Oriented Neural Engine

http://sourceforge.net/projects/joone

Joone is a neural net engine written in Java. It's a modular, scalable, multitasking and extensible engine. It can be extended by writing new modules to implement new algorithms or new architectures starting from simple base components. It's an Open Source project and everybody can contribute to its development.

Contact: Paolo Marrone, paolo@marrone.org

42. NV: Neural Viewer

http://www.btinternet.com/~cfinnie/

A free software application for modelling and visualizing complex recurrent neural networks in 3D.

43. EasyNN

URL: http://www.easynn.com/

EasyNN is a neural network system for Microsoft Windows. It can generate multi layer neural networks from text files or grids with minimal user intervention. The networks can then be trained, validated and queried. Network diagrams, graphs, input/output data and all the network details can be displayed and printed. Nodes can be added or deleted while the network is learning. The graph, grid, network and detail displays are updated dynamically so you can see how the neural networks work. EasyNN runs on Windows 95, 98, ME, NT 4.0, 2000 or XP.

44. Multilayer Perceptron - A Java Implementation

Download java from: http://www.geocities.com/aydingurel/neural/

What can you exactly do with it? You can:

- Build nets with any number of layers and units. Layers are connected to each other consecutively, each unit in a layer is connected to all of the units on the next layer (and vice versa) if there is one,
- Set units with linear and sigmoid activation functions and set them separately for each layer,
- Set parameters for sigmoid functions and set them separately for each layer,
- Use momentum, set different momentum parameters for each layer,
- Initialize the net using your own set of weights,
- Train the net using backpropagation and with any training rate.

Contact: Aydin Gurel, aydin.gurel@lycos.com

For some of these simulators there are user mailing lists. Get the packages and look into their documentation for further info.

Next part is part 6 (of 7). Previous part is part 4.

```
Archive-name: ai-faq/neural-nets/part6
Last-modified: 2002-03-28
URL: ftp://ftp.sas.com/pub/neural/FAQ6.html
Maintainer: saswss@unx.sas.com (Warren S. Sarle)
```


This is part 6 (of 7) of a monthly posting to the Usenet newsgroup comp.ai.neural-nets. See the part 1 of this posting for full information what it is all about.

========== Questions ==========

Subject: Commercial software packages for NN simulation?

Since the FAQ maintainer works for a software company, he does not recommend or evaluate software in the FAQ. The descriptions below are provided by the developers or distributors of the software.

Note for future submissions: Please restrict product descriptions to a maximum of 60 lines of 72 characters, in either plain-text format or, preferably, HTML format. If you include the standard header (name, company, address, etc.), you need not count the header in the 60 line maximum. Please confine your HTML to features that are supported by primitive browsers, especially NCSA Mosaic 2.0; avoid tables, for example--use <pre> instead. Try to make the descriptions objective, and avoid making implicit or explicit assertions about competing products, such as "Our product is the *only* one that does so-and-so." The FAQ maintainer reserves the right to remove excessive marketing hype and to edit submissions to conform to size requirements; if he is in a good mood, he may also correct your spelling and punctuation.

The following simulators are described below:

12. NeuroSolutions
13. Qnet For Windows Version 2.0
14. NeuroLab, A Neural Network Library
15. hav.Software: havBpNet++, havFmNet++, havBpNet:J
16. IBM Neural Network Utility
17. NeuroGenetic Optimizer (NGO) Version 2.0
18. WAND
19. The Dendronic Learning Engine
20. TDL v. 1.1 (Trans-Dimensional Learning)
21. NeurOn-Line
22. Neuframe
23. OWL Neural Network Library (TM)
24. Neural Connection
25. Pattern Recognition Workbench Expo/PRO/PRO+
26. PREVia
27. Trajan 2.1 Neural Network Simulator
28. DataEngine
29. Machine Consciousness Toolbox
30. Professional Basis of AI Backprop
31. STATISTICA: Neural Networks
32. Braincel (Excel add-in)
33. DESIRE/NEUNET
34. Viscovery SOMine
35. NeuNet Pro
36. Neuronics
37. RG Software
38. Cobalt A.I. Code Builder - Neural Network Edition
39. NEURO MODEL and GenOpt

See also http://www.emsl.pnl.gov:2080/proj/neuron/neural/systems/software.html

1. BrainMaker

```
   Product: BrainMaker, BrainMaker Pro
   Company: California Scientific Software
   Address: 10024 Newtown rd, Nevada City, CA, 95959 USA
     Phone: 800 284-8112, 530 478 9040
       Fax: 530 478 9041
       URL: http://www.calsci.com/
Basic capabilities:  train backprop neural nets
Operating system:   Windows, Mac
System requirements:

Approx. price:  $195, $795

BrainMaker Pro 3.7 (DOS/Windows)     $795
    Gennetic Training add-on         $250
BrainMaker 3.7 (DOS/Windows/Mac)     $195
    Network Toolkit add-on           $150
BrainMaker 3.7 Student version       (quantity sales only, about $38 each)

BrainMaker Pro CNAPS Accelerator Board $8145

Introduction To Neural Networks book $30
```

```
30 day money back guarantee, and unlimited free technical support.
BrainMaker package includes:
 The book Introduction to Neural Networks
 BrainMaker Users Guide and reference manual
     300 pages, fully indexed, with tutorials, and sample networks
 Netmaker
     Netmaker makes building and training Neural Networks easy, by
     importing and automatically creating BrainMaker's Neural Network
     files.  Netmaker imports Lotus, Excel, dBase, and ASCII files.
 BrainMaker
     Full menu and dialog box interface, runs Backprop at 3,000,000 cps
     on a 300Mhz Pentium II; 570,000,000 cps on CNAPS accelerator.

---Features ("P" means is available in professional version only):
MMX instruction set support for increased computation speed,
Pull-down Menus, Dialog Boxes, Programmable Output Files,
Editing in BrainMaker,  Network Progress Display (P),
Fact Annotation,  supports many printers,  NetPlotter,
Graphics Built In (P),  Dynamic Data Exchange (P),
Binary Data Mode, Batch Use Mode (P), EMS and XMS Memory (P),
Save Network Periodically,  Fastest Algorithms,
512 Neurons per Layer (P: 32,000), up to 8 layers,
Specify Parameters by Layer (P), Recurrence Networks (P),
Prune Connections and Neurons (P),  Add Hidden Neurons In Training,
Custom Neuron Functions,  Testing While Training,
Stop training when...-function (P),  Heavy Weights (P),
Hypersonic Training,  Sensitivity Analysis (P),  Neuron Sensitivity (P),
Global Network Analysis (P),  Contour Analysis (P),
Data Correlator (P),  Error Statistics Report,
Print or Edit Weight Matrices,  Competitor (P), Run Time System (P),
Chip Support for Intel, American Neurologics, Micro Devices,
Genetic Training Option (P),  NetMaker,  NetChecker,
Shuffle,  Data Import from Lotus, dBASE, Excel, ASCII, binary,
Finacial Data (P),  Data Manipulation,  Cyclic Analysis (P),
User's Guide quick start booklet,
Introduction to Neural Networks 324 pp book
```

2. SAS Enterprise Miner Software

```
Product: SAS Enterprise Miner Solution

         In USA:                 In Europe:
Company: SAS Institute, Inc.     SAS Institute, European Office
Address: SAS Campus Drive        Neuenheimer Landstrasse 28-30
         Cary, NC 27513          P.O.Box 10 53 40
         USA                     D-69043 Heidelberg
                                 Germany
  Phone: (919) 677-8000          (49) 6221 4160
    Fax: (919) 677-4444          (49) 6221 474 850

    URL: http://www.sas.com/
```

To find the addresses and telephone numbers of other SAS Institute offices, including those outside the USA and Europe, connect your web browser to http://www.sas.com/offices/intro.html.

Enterprise Miner is an integrated software product that provides an end-to-end business solution for data mining based on SEMMA methodology (Sample, Explore, Modify, Model, Assess). Statistical tools include clustering, decision trees, linear and logistic regression, and neural networks. Data preparation tools include outlier detection, variable

transformations, random sampling, and the partitioning of data sets (into training, test, and validation data sets). Advanced visualization tools enable you to quickly and easily examine large amounts of data in multidimensional histograms, and to graphically compare modeling results.

The neural network tool includes multilayer perceptrons, radial basis functions, statistical versions of counterpropagation and learning vector quantization, a variety of built-in activation and error functions, multiple hidden layers, direct input-output connections, categorical variables, standardization of inputs and targets, and multiple preliminary optimizations from random initial values to avoid local minima. Training is done by powerful numerical optimization algorithms instead of tedious backprop.

3. NeuralWorks

```
Product: NeuralWorks Professional II Plus
         NeuralWorks Predict
Company: NeuralWare
 Adress: 230 East Main Street
         Suite 200
         Carnegie, PA 15106-2700
  Phone: (412) 278-6280
    FAX: (412) 278-6289

Email: sales@neuralware.com
  URL: http://www.neuralware.com/
```

NeuralWorks Professional II/PLUS is a comprehensive neural network development environment. Professional II/PLUS is available for UNIX, Linux, and Windows operating systems on a variety of hardware platforms; data and network files are fully interchangeable. The Professional II/PLUS package contains comprehensive documentation that addresses the entire neural network development and deployment process, including a tutorial, a guide to neural computing, standard and advanced reference manuals, and platform-specific installation and user guides.

NeuralWare's proprietary InstaNet facility allows quick generation of a neural network based on any one of 28 standard neural network architectures described in neural network literature. After a network is created, all parameters associated with it can be directly modified to more closely reflect the problem domain. Professional II/PLUS includes advanced features such as performance measure-based methods to inhibit over-fitting; automatic optimization of hidden layer size and the ability to prune hidden units; and an Explain facility that indicates which network inputs most influence network outputs.

NeuralWorks Predict is an integrated tool for rapidly creating and deploying prediction and classification applications. Predict combines neural network technology with genetic algorithms, statistics, and fuzzy logic to automatically find solutions for a wide range of problems. Predict incorporates years of modeling and analysis experience gained from working with customers faced with a wide variety of analysis and interpretation problems.

Predict requires no prior knowledge of neural networks. With only minimal user involvement it addresses the issues associated with building robust models from available empirical data. Predict analyzes input data to identify appropriate transforms, partitions the input data into training and test sets, selects relevant input variables, and then constructs, trains, and optimizes a neural network tailored to the problem. For advanced users, Predict also offers direct access to all key training and network parameters.

4. MATLAB Neural Network Toolbox

```
The Mathworks Inc.
3 Apple Hill Drive
Natck, MA 01760
Phone: 508-647-7000
  Fax: 508-647-7001
  URL: http://www.mathworks.com/products/neuralnet/
```

The Neural Network Toolbox is a powerful collection of MATLAB functions for the design, training, and simulation of neural networks. It supports a wide range of network architectures with an unlimited number of processing elements and interconnections (up to operating system constraints). Supported architectures and training methods include: supervised training of feedforward networks using the perceptron learning rule, Widrow-Hoff rule, several variations on backpropagation (including the fast Levenberg-Marquardt algorithm), and radial basis networks; supervised training of recurrent Elman networks; unsupervised training of associative networks including competitive and feature map layers; Kohonen networks, self-organizing maps, and learning vector quantization. The Neural Network Toolbox contains a textbook-quality Users' Guide, uses tutorials, reference materials and sample applications with code examples to explain the design and use of each network architecture and paradigm. The Toolbox is delivered as MATLAB M-files, enabling users to see the algorithms and implementations, as well as to make changes or create new functions to address a specific application.

5. Propagator

```
Contact: ARD Corporation,
         9151 Rumsey Road, Columbia, MD  21045, USA
         propagator@ard.com
Easy to use neural network training package.  A GUI implementation of
backpropagation networks with five layers (32,000 nodes per layer).
Features dynamic performance graphs, training with a validation set,
and C/C++ source code generation.
For Sun (Solaris 1.x & 2.x, $499),
    PC  (Windows 3.x, $199)
    Mac (System 7.x, $199)
Floating point coprocessor required, Educational Discount,
Money Back Guarantee, Muliti User Discount
See http://www.cs.umbc.edu/~zwa/Gator/Description.html
Windows Demo on:
  nic.funet.fi        /pub/msdos/windows/demo
  oak.oakland.edu     /pub/msdos/neural_nets
      gatordem.zip    pkzip 2.04g archive file
      gatordem.txt    readme text file
```

6. NeuroForecaster & VisuaData

```
 Product: NeuroForecaster(TM)/Genetica 4.1a
 Contact: Accel Infotech (S) Pte Ltd
          648 Geylang Road
          Republic of Singapore 1438
   Phone: +65-7446863, 3366997
     Fax: +65-3362833
     URL: http://web.singnet.com.sg/~midaz/
```

Neuroforecaster 4.1a for Windows is priced at **US$1199** per single user license. Please email accel@technet.sg for order form.

NeuroForecaster is a user-friendly ms-windows neural network program specifically designed for building sophisticated and powerful forecasting and decision-support systems (Time-Series Forecasting, Cross-Sectional Classification, Indicator Analysis)
Features:

- GENETICA Net Builder Option for automatic network optimization
- 12 Neuro-Fuzzy Network Models
- Multitasking & Background Training Mode
- Unlimited Network Capacity
- Rescaled Range Analysis & Hurst Exponent to Unveil Hidden Market

- Cycles & Check for Predictability
- Correlation Analysis to Compute Correlation Factors to Analyze the
- Significance of Indicators
- Weight Histogram to Monitor the Progress of Learning
- Accumulated Error Analysis to Analyze the Strength of Input Indicators

The following example applications are included in the package:

- Credit Rating - for generating the credit rating of bank loan applications.
- Stock market 6 monthly returns forecast
- Stock selection based on company ratios
- US$ to Deutschmark exchange rate forecast
- US$ to Yen exchange rate forecast
- US$ to SGD exchange rate forecast
- Property price valuation
- Chaos - Prediction of Mackey-Glass chaotic time series
- SineWave - For demonstrating the power of Rescaled Range Analysis and significance of window size

Techniques Implemented:

- GENETICA Net Builder Option - network creation & optimization based on Darwinian evolution theory
- Backprop Neural Networks - the most widely-used training algorithm
- Fastprop Neural Networks - speeds up training of large problems
- Radial Basis Function Networks - best for pattern classification problems
- Neuro-Fuzzy Network
- Rescaled Range Analysis - computes Hurst exponents to unveil hidden cycles & check for predictability
- Correlation Analysis - to identify significant input indicators
-

Companion Software - VisuaData for Windows A user-friendly data management program designed for intelligent technical analysis. It reads **MetaStock**, **CSI**, **Computrac** and various ASCII data file formats directly, generates over 100 popular and new technical indicators and buy/sell signals.

7. Products of NESTOR, Inc.

```
530 Fifth Avenue;
New York, NY 10036; USA;
Tel.: 001-212-398-7955
URL: http://asweb.artsci.uc.edu/classics/nestor/nestor.html

Founders:
Dr. Leon Cooper (having a Nobel Prize) and Dr. Charles Elbaum
(Brown University).

Neural Network Models:
Adaptive shape and pattern recognition (Restricted Coulomb Energy - RCE)
developed by NESTOR is one of the most powerfull Neural Network Model
used in a later products.

The basis for NESTOR products is the Nestor Learning System - NLS. Later
are developed: Character Learning System - CLS and Image Learning System
- ILS.  Nestor Development System - NDS is a development tool in
Standard C - a powerfull PC-Tool for simulation and development of
Neural Networks.

NLS is a multi-layer, feed forward system with low connectivity within
each layer and no relaxation procedure used for determining an output
response.  This unique architecture allows the NLS to operate in real
time without the need for special computers or custom hardware.

NLS is composed of multiple neural networks, each specializing in a
subset of information about the input patterns. The NLS integrates the
responses of its several parallel networks to produce a system response.
```

```
Minimized connectivity within each layer results in rapid training and
efficient memory utilization- ideal for current VLSI technology. Intel
has made such a chip - NE1000.
```

8. Ward Systems Group (NeuroShell, etc.)

```
Product: NeuroShell Predictor, NeuroShell Classifier,
         NeuroShell Run-Time Server, NeuroShell Trader,
         NeuroShell Trader Professional, NeuroShell 2,
         GeneHunter, NeuroShell Engine
Company: Ward Systems Group, Inc.
Address: Executive Park West
         5 Hillcrest Drive
         Frederick, MD 21702
         USA
  Phone: 301 662-7950
    FAX: 301 662-5666
  Email: WardSystems@msn.com
    URL: http://www.wardsystems.com
```

Ward Systems Group Product Descriptions:

- NeuroShell. Predictor - This product is used for forecasting and estimating numeric amounts such as sales, prices, workload, level, cost, scores, speed, capacity, etc. It contains two of our newest algorithms (neural and genetic) with no parameters for you to have to set. These are our most powerful networks. Reads and writes text files.

- NeuroShell. Classifier - The NeuroShell Classifier solves classification and categorization problems based on patterns learned from historical data. The Classifier produces outputs that are the probabilities of the input pattern belonging to each of several categories. Examples of categories include {acidic, neutral, alkaline}, {buy, sell, hold}, and {cancer, benign}. Like the NeuroShell Predictor, it has the latest neural and genetic classifiers with no parameters to set. These are our most powerful networks. It also reads and writes text files.

- NeuroShell. Run-Time Server - The NeuroShell Run-Time Server allows you to distribute networks created with the NeuroShell Predictor or NeuroShell Classifier from either a simple interface, from your own computer programs, or from Excel spreadsheets.

- NeuroShell Trader. - This is the complete product for anyone trading stocks, bonds, futures, commodities, currencies, derivatives, etc. It works the way you think and work: for example, it reads open, high, low, close type price streams. It contains charting, indicators, the latest neural nets, trading simulations, data downloading, and walk forward testing, all seamlessly working together to make predictions for you. The NeuroShell Trader contains our most powerful network type.

- NeuroShell Trader. Professional - The Professional incorporates the original Trader along with the ability to optimize systems with a genetic algorithm even if they don't include neural nets! For example, you can enter a traditional trading strategy (using crossovers and breakouts) and then find optimal parameters for those crossovers and breakouts. You can also use the optimizer to remove useless rules in your trading strategy.

- NeuroShell. 2 - This is our classic general purpose system highly suited to students and professors who are most comfortable with traditional neural nets (not recommended for problem solving in a business or scientific environment). It contains 16 traditional neural network paradigms (algorithms) and combines ease of use and lots of control over how the networks are trained. Parameter defaults make it easy for you to get started, but provide generous flexibility and control later for experimentation. Networks can either predict or classify. Uses spreadsheet files, but can import other types. Runtime facilities include a source code generator, and there are several options for processing many nets, 3D graphics (response surfaces), and financial or time series indicators. It does not contain our newest network types that are in the NeuroShell Predictor, the NeuroShell Classifier and the

NeuroShell Trader.

- GeneHunter. - This is a genetic algorithm product designed for optimizations like finding the best schedules, financial indicators, mixes, model variables, locations, parameter settings, portfolios, etc. More powerful than traditional optimization techniques, it includes both an Excel spreadsheet add-in and a programmer's tool kit.

- NeuroShell. Engine - This is an Active X control that contains the neural and genetic training methods that we have used ourselves in the NeuroShell Predictor, the NeuroShell Classifier, and the NeuroShell Trader. They are available to be integrated into your own computer programs for both training and firing neural networks. The NeuroShell Engine is only for the most serious neural network users, and only those who are programmers or have programmers on staff.

Contact us for a free demo diskette and Consumer's Guide to Neural Networks.

9. Neuralyst

```
Product:  Neuralyst Version 1.4;
Company:  Cheshire Engineering Corporation;
Address:  650 Sierra Madre Villa, Suite 201, Pasedena CA 91107;
  Phone:  818-351-0209;
    Fax:  818-351-8645;
    URL:  http://www.cheshireeng.com/Neuralyst/
```

Basic capabilities: training of backpropogation neural nets. Operating system: Windows or Macintosh running Microsoft Excel Spreadsheet. Neuralyst is an add-in package for Excel. Approx. price: $195 for windows or Mac.

10. Cortex-Pro

Cortex-Pro information is on WWW at:
http://www.reiss.demon.co.uk/webctx/intro.html.
You can download a working demo from there.
Contact: Michael Reiss (http://www.mth.kcl.ac.uk/~mreiss/mick.html)
email: <m.reiss@kcl.ac.uk>.

11. Partek

Partek is a young, growing company dedicated to providing our customers with the best software and services for data analysis and modeling. We do this by providing a combination of statistical analysis and modeling techniques and modern tools such as neural networks, fuzzy logic, genetic algorithms, and data visualization. These powerful analytical tools are delivered with high quality, state of the art software.

Please visit our home on the World Wide Web: www.partek.com

Partek Incorporated
5988 Mid Rivers Mall Dr.
St. Charles, MO 63304
voice: 314-926-2329
fax: 314-441-6881
email: info@partek.com
http://www.partek.com/

12. NeuroSolutions v3.0

```
Product: NeuroSolutions
Company: NeuroDimension, Inc.
```

```
          Address: 1800 N. Main St., Suite D4
                   Gainesville FL, 32609
            Phone: (800) 634-3327 or (352) 377-5144
              FAX: (352) 377-9009
            Email: info@nd.com
              URL: http://www.nd.com/
Operating System: Windows 95/98/Me/NT/2000
            Price: $195 - $1995 (educational discounts available)
```

NeuroSolutions is a highly graphical neural network development tool for Windows 95/98/Me/NT/2000. This leading edge software combines a modular, icon-based network design interface with an implementation of advanced learning procedures, such as recurrent backpropagation, backpropagation through time and genetic optimization. The result is a virtually unconstrained environment for designing neural networks for research or to solve real-world problems.

Download a free evaluation copy from **http://www.nd.com/download.htm**.

Topologies

- Multilayer perceptrons (MLPs)
- Generalized Feedforward networks
- Modular networks
- Jordan-Elman networks
- Self-Organizing Feature Map (SOFM) networks
- Radial Basis Function (RBF) networks
- Time Delay Neural Networks (TDNN)
- Time-Lag Recurrent Networks (TLRN)
- Recurrent Networks (TLRN)
- General Regression Networks (GRNN)
- Probabilistic Networks (PNN)
- Neuro-Fuzzy Networks
- Support Vector Machines (SVM)
- User-defined network topologies

Learning Paradigms

- Backpropagation
- Recurrent Backpropagation
- Backpropagation through Time
- Conjugate Gradient Learning
- Unsupervised Learning
  - Hebbian
  - Oja's
  - Sanger's
  - Competitive
  - Kohonen

Advanced Features

- ANSI C++ Source Code Generation
- Customized Components through DLLs
- Genetic Optimization
- Microsoft Excel Add-in -- NeuroSolutions for Excel
  - Visual Data Selection
  - Data Preprocessing and Analysis
  - Batch Training and Parameter Optimization
  - Sensitivity Analysis
  - Automated Report Generation

- DLL Generation Utility -- The Custom Solution Wizard
  - Encapsulate any supervised NeuroSolutions NN into a Dynamic Link Library (DLL)
  - Use the DLL to embed a NN into your own Visual Basic, Microsoft Excel, Microsoft Access or Visual C++ application
  - Support for both Recall and Learning networks available
  - Simple protocol for sending the input data and retrieving the network response
- Comprehensive Macro Language
- Fully accessible from any OLE-compliant application, such as:
  - Visual Basic
  - Microsoft Excel
  - Microsoft Access

13. Qnet For Windows Version 2.0


```
Vesta Services, Inc.
1001 Green Bay Rd, Suite 196
Winnetka, IL   60093
Phone:   (708) 446-1655
E-Mail:  VestaServ@aol.com
```


Trial Version Available: http://www.qnetv2k.com/

Vesta Services announces Qnet for Windows Version 2.0. Qnet is an advanced neural network modeling system that is ideal for developing and implementing neural network solutions under Windows. The use of neural network technology has grown rapidly over the past few years and is being employed by an increasing number of disciplines to automate complex decision making and problem solving tasks. Qnet Version 2 is a powerful, 32-bit, neural network development system for Windows NT, Windows 95 and Windows 3.1/Win32s. In addition its development features, Qnet automates access and use of Qnet neural networks under Windows.

Qnet neural networks have been successfully deployed to provide solutions in finance, investing, marketing, science, engineering, medicine, manufacturing, visual recognition... Qnet's 32-bit architecture and high-speed training engine tackle problems of large scope and size. Qnet also makes accessing this advanced technology easy. Qnet's neural network setup dialogs guide users through the design process. Simple copy/paste procedures can be used to transfer training data from other applications directly to Qnet. Complete, interactive analysis is available during training. Graphs monitor all key training information. Statistical checks measure model quality. Automated testing is available for training optimization. To implement trained neural networks, Qnet offers a variety of choices. Qnet's built-in recall mode can process new cases through trained neural networks. Qnet also includes a utility to automate access and retrieval of solutions from other Windows applications. All popular Windows spreadsheet and database applications can be setup to retrieve Qnet solutions with the click of a button. Application developers are provided with DLL access to Qnet neural networks and for complete portability, ANSI C libraries are included to allow access from virtually any platform.

Qnet for Windows is being offered at an introductory price of $199. It is available immediately and may be purchased directly from Vesta Services. Vesta Services may be reached at (voice) (708) 446-1655; (FAX) (708) 446-1674; (e-mail) VestaServ@aol.com; (mail) 1001 Green Bay Rd, #196, Winnetka, IL 60093

14. NeuroLab, A Neural Network Library

Contact: Mikuni Berkeley R & D Corporation; 4000 Lakeside Dr.; Richmond, CA
Tel: 510-222-9880; Fax: 510-222-9884; e-mail: neurolab-info@mikuni.com

NeuroLab is a block-diagram-based neural network library for Extend simulation software (developed by Imagine That, Inc.). The library aids the understanding, designing and simulating of neural network systems. The library consists of more than 70 functional blocks for artificial neural network implementation and many example models in several professional fields.The package provides icon-based functional blocks for easy implementation of simulation models. Users click, drag and connect blocks to construct a neural network and can specify network parameters--such as back propagation methods, learning rates, initial weights, and biases--in the dialog boxes of the functional blocks.

Users can modify blocks with the Extend model-simulation scripting language, ModL, and can include compiled program modules written in other languages using XCMD and XFCN (external command and external function) interfaces and DLL (dynamic linking library) for Windows. The package provides many kinds of output blocks to monitor neural network status in real time using color displays and animation and includes special blocks for control application fields. Educational blocks are also included for people who are just beginning to learn about neural networks and their applications.
The library features various types of neural networks --including Hopfield, competitive, recurrent, Boltzmann machine, single/multilayer feed-forward, perceptron, context, feature map, and counter-propagation-- and has several back-propagation options: momentum and normalized methods, adaptive learning rate, and accumulated learning.

The package runs on Macintosh II or higher (FPU recommended) with system 6.0.7 or later and PC compatibles (486 or higher recommended) with Windows 3.1 or later, and requires 4Mbytes of RAM. Models are transferable between the two platforms. NeuroLab v1.2 costs US$495 (US$999 bundled with Extend v3.1). Educational and volume discounts are available.
A free demo can be downloaded by ftp://ftp.mikuni.com/pub/neurolab or http://www.mikuni.com/. Orders, questions or suggestions can be sent by e-mail to neurolab-info@mikuni.com.

15. hav.Software: havBpNet++, havFmNet++, havBpNet:J

```
Names:      havBpNet++
            havFmNet++
            havBpNet:J
Company:    hav.Software
            P.O. Box 354
            Richmond, Tx.  77406-0354 - USA
Phone:      (281) 341-5035
Email:      hav@hav.com
Web:        http://www.hav.com/
```

- havBpNet++ is a C++ class library that implements feedforward, simple recurrent and random-ordered recurrent nets trained by backpropagation. Used for both stand-alone and embedded network training and consultation applications. A simple layer-based API, along with no restrictions on layer-size or number of layers, makes it easy to build standard 3-layer nets or much more complex multiple sub-net topologies.

  Supports all standard network parameters (learning-rate, momentum, Cascade- coefficient, weight-decay, batch training, etc.). Includes 5 activation-functions (Linear, Logistic-sigmoid, Hyperbolic-tangent, Sin and Hermite) and 3 error-functions (e^2, e^3, e^4). Also included is a special scaling utility for data with large dynamic range.

  Several data-handling classes are also included. These classes, while not required, may be used to provide convenient containers for training and consultation data. They also provide several normalization/de-normalization methods.

  havBpNet++ is delivered as fully documented source + 200 pg User/Developer Manual. Includes a special DLL version. Includes several example trainers and consulters with data sets. Also included is a fully functioning copy of the havBpETT demo (with network-save enabled).

  NOTE: a freeware version (Save disabled) of the havBpETT demo may be downloaded from the hav.Software home-page: http://www.neosoft.com/~hav or by anonymous ftp from ftp://ftp.neosoft.com/pub/users/h/hav/havBpETT/demo2.exe.

  ```
  Platforms:      Tested platforms include - PC (DOS, Windows-3.1, NT, Unix),
                  HP (HPux), SUN (Sun/OS), IBM (AIX), SGI (Irix).
                  Source and Network-save files portable across platforms.

  Licensing:      havBpNet++ is licensed by number of developers.
                  A license may be used to support development on any number
  ```

```
                and types of cpu's.
                No Royalties or other fees (except for OEM/Reseller)

Price:          Individual        $50.00 - one developer
                Site             $500.00 - multiple developers - one location
                Corporate       $1000.00 - multiple developers and locations
                OEM/Reseller    quoted individually
                (by American Express, bank draft and approved company PO)

Media:  3.5-inch floppy - ascii format (except havBpETT which is in PC-exe
                                        format).
```

- havFmNet++ is a C++ class library that implements Self-Organizing Feature Map nets. Map-layers may be from 1 to any dimension.

  havFmNet++ may be used for both stand-alone and embedded network training and consultation applications. A simple Layer-based API, along with no restrictions on layer-size or number of layers, makes it easy to build single-layer nets or much more complex multiple-layer topologies. havFmNet++ is fully compatible with havBpNet++ which may be used for pre- and post- processing.

  Supports all standard network parameters (learning-rate, momentum, neighborhood, conscience, batch, etc.). Uses On-Center-Off-Surround training controlled by a sombrero form of Kohonen's algorithm. Updates are controllable by three neighborhood related parameters: neighborhood-size, block-size and neighborhood-coefficient cutoff. Also included is a special scaling utility for data with large dynamic range.

  Several **data-handling classes** are also included. These classes, while not required, may be used to provide convenient containers for training and consultation data. They also provide several normalization/de-normalization methods.

  havFmNet++ is delivered as fully documented source plus 200 pg User/Developer Manual. Includes several example trainers and consulters with data sets.

```
  Platforms:      Tested platforms include - PC (DOS, Windows-3.1, NT, Unix),
                  HP (HPux), SUN (Sun/OS), IBM (AIX), SGI (Irix).
                  Source and Network-save files portable across platforms.

  Licensing:      havFmNet++ is licensed by number of developers.
                  A license may be used
                  to support development on any number and types of cpu's.
                  No Royalties or other fees (possible exception for OEM).

  Price:          Individual        $50.00 - one developer
                  Site             $500.00 - multiple developers - one location
                  Corporate       $1000.00 - multiple developers and locations
                  OEM/Reseller    quoted individually
                  (by American Express, bank draft and approved company PO)

  Media:  3.5-inch floppy - ascii format
```

- havBpNet:J is a Java class library that implements feedforward, simple recurrent (sequential) and random-ordered recurrent nets trained by backpropagation. Used for both stand-alone and embedded network training and consultation applications and applets. A simple layer-based API, along with no restrictions on layer-size or number of layers, makes it easy to build standard 3-layer nets or much more complex multiple sub-net topologies.

  Supports all standard network parameters (learning-rate, momentum, Cascade-coefficient, weight-decay, batch training, error threshold, etc.). Includes 5 activation-functions (Linear, Logistic-sigmoid, Hyperbolic-tangent, Sin and Hermite) and 3 error-functions (e^2, e^3, e^4). Also included is a special scaling utility for data with large dynamic range.

Several data-handling classes are also included. These classes, while not required, may be used to provide convenient containers for training and consultation data. They also provide several normalization/de-normalization methods.

```
Platforms:  Java Virtual Machines - 1.0 and later

Licensing:  No Royalties or other fees (except for OEM/Reseller)

Price:      Individual License is $55.00
            Site, Corporate and OEM/Reseller also available

Media:      Electronic distribution only.
```

16. IBM Neural Network Utility

Product Name: IBM Neural Network Utility
Distributor: Contact a local reseller or call 1-800-IBM-CALL, Dept. SA045 to order.
Basic capabilities: The Neural Network Utility Family consists of six products: client/server capable versions for OS/2, Windows, AIX, and standalone versions for OS/2 and Windows. Applications built with NNU are portable to any of the supported platforms regardless of the development platform. NNU provides a powerful, easy to use, point-and-click graphical development environment. Features include: data translation and scaling, applicaton generation, multiple network models, and automated network training. We also support fuzzy rule systems, which can be combined with the neural networks. Once trained, our APIs allow you to embed your network and/or rulebase into your own applications.
Operating Systems: OS/2, Windows, AIX, AS/400
System requirements: basic; request brochure for more details
Price: Prices start at $250
For product brochures, detailed pricing information, or any other information, send a note to nninfo@vnet.ibm.com.

17. NeuroGenetic Optimizer (NGO) Version 2.0

BioComp's leading product is the NeuroGenetic Optimizer, or NGO. As the name suggests, the NGO is a neural network development tool that uses genetic algorithms to optimize the inputs and structure of a neural network. Without the NGO, building neural networks can be tedious and time consuming even for an expert. For example, in a relatively simple neural network, the number of possible combination of inputs and neural network structures can be easily over 100 billion. The difference between an average network and an optimum network is substantial. The NGO searches for optimal neural network solutions. See our web page at http://www.bio-comp.com. for a demo that you can download and try out. Our customers who have used other neural network development tools are delighted with both the ease of use of the NGO and the quality to their results.

BioComp Systems, Inc. introduced version 1.0 of the NGO in January of 1995 and now proudly announces version 2.0. With version 2.0, the NGO is now equipped for predicting time-based information such as product sales, financial markets and instruments, process faults, etc., in addition to its current capabilities in functional modeling, classification, and diagnosis.

While the NGO embodies sophisticated genetic algorithm search and neural network modeling technology, it has a very easy to use GUI interface for Microsoft Windows. You don't have to know or understand the underlying technology to build highly effective financial models. On the other hand, if you like to work with the technology, the NGO is highly configurable to customize the NGO to your liking.

Key new features of the NGO include:

- Highly effective "Continuous Adaptive Time", Time Delay and lagged input Back Propagation neural networks with optional recurrent outputs, automatically competing and selected based on predictive accuracy.
- Walk Forward Train/Test/Validation model evaluation for assuring model robustness,
- Easy input data lagging for Back Propagation neural models,
- Neural transfer functions and techniques that assure proper extrapolation of predicted variables to new highs,

- Confusion matrix viewing of Predicted vs. Desired results,
- Exportation of models to Excel 5.0 (Win 3.1) or Excel 7.0 (Win'95/NT) through an optional Excel Add-In
- Five accuracies to choose from including; Relative Accuracy, R-Squared, Mean Squared Error (MSE), Root Mean Square (RMS) Error and Average Absolute error.

With version 2.0, the NGO is now available as a full 32 bit application for Windows '95 and Windows NT to take advantage of the 32 bit preemptive multitasking power of those platforms. A 16 bit version for Windows 3.1 is also available. Customized professional server based systems are also available for high volume automated model generation and prediction. Prices start at $195.

```
BioComp Systems, Inc.
Overlake Business Center
2871 152nd Avenue N.E.
Redmond, WA 98052, USA
1-800-716-6770 (US/Canada voice)=20  1-206-869-6770 (Local/Int'l voice)
1-206-869-6850 (Fax)                 http://www.bio-comp.com.
biocomp@biocomp.seanet.com           70673.1554@compuserve.com
```

18. WAND

Weightless Neural Design system for Education and Industry.
Developed by Novel Technical Solutions in association with Imperial College of Science, Technology and Medicine (London UK).
WAND introduces Weightless Neural Technology as applied to Image Recognition.
It includes an automated image preparation package, a weightless neural simulator and a comprehensive manual with hands-on tutorials.
Full information including a download demo can be obtained from:
http://www.sonnet.co.uk/nts/
To contact Novel Technical Solutions email: <neural@nts.sonnet.co.uk>.

19. The Dendronic Learning Engine

The Dendronic Learning Engine (DLE) Software Development Kit (SDK) allows the user to easily program machine learning technology into high performance applications. Application programming interfaces to C and C++ are provided. Supervised learning is supported, as well as the recently developed reinforcement learning of value functions and Q-functions. Fast evaluation on PC hardware is supported by ALN Decision Trees.

An ALN consists of linear functions with adaptable weights at the leaves of a tree of maximum and minimum operators. The tree grows automatically during training: a linear piece splits if its error is too high. The function computed by an ALN is piecewise linear and continuous, and can approximate any continuous function to any required accuracy. Statistics are reported on the fit of each linear piece to test data.

The DLE has been very successful in predicting electrical power loads in the Province of Alberta. The 24-hour ahead load for all of Alberta (using seven weather regions) was predicted using a single ALN. Visit http://www.dendronic.com/applications.htm to learn about other successful applications to ATM communication systems, walking prostheses for persons with incomplete spinal cord injury, and many other areas.

Operating Systems: Windows NT 4.0, 95 and higher.

Price: Prices for the SDK start at $3495 US for a single seat, and are negotiable for embedded learning systems. A runtime non-learning system can be distributed royalty-free.

The power of the DLE can be tried out using an interactive spreadsheet program, ALNBench. ALNBench is free for research, education and evaluation purposes. The program can be downloaded from http://www.dendronic.com/beta.htm (Please note the installation key given there!)

For further information please contact:

William W. Armstrong PhD, President
Dendronic Decisions Limited
3624 - 108 Street, NW
Edmonton, Alberta,
Canada T6J 1B4
Email: arms@dendronic.com
URL: http://www.dendronic.com/
Tel. +1 403 421 0800
(Note: The area code 403 changes to 780 after Jan. 25, 1999)

20. TDL v. 1.1 (Trans-Dimensional Learning)

Platform: Windows 3.*
Company: Universal Problem Solvers, Inc.
WWW-Site (UPSO): http://www.alberts.com/authorpages/00002375/prod_262.htm
or FTP-Site (FREE Demo only): ftp.coast.net, in Directory:
SimTel/win3/neurlnet, File: tdl11-1.zip and tdl11-2.zip
Cost of complete program: US$20 + (US$3 Shipping and Handling).

The purpose of TDL is to provide users of neural networks with a specific platform to conduct pattern recognition tasks. The system allows for the fast creation of automatically constructed neural networks. There is no need to resort to manually creating neural networks and twiddling with learning parameters. TDL's Wizard can help you optimize pattern recognition accuracy. Besides allowing the application user to automatically construct neural network for a given pattern recognition task, the system supports trans-dimensional learning. Simply put, this allows one to learn various tasks within a single network, which otherwise differ in the number of input stimuli and output responses utilized for describing them. With TDL it is possible to incrementally learn various pattern recognition tasks within a single coherent neural network structure. Furthermore, TDL supports the use of semi-weighted neural networks, which represent a hybrid cross between standard weighted neural networks and weightless multi-level threshold units. Combining both can result in extremely compact network structures (i.e., reduction in connections and hidden units), and improve predictive accuracy on yet unseen patterns. Of course the user has the option to create networks which only use standard weighted neurons.

System Highlights:

1. The user is in control of TDL's memory system (can decide how many examples and neurons are allocated ; no more limitations, except for your computers memory).
2. TDLs Wizard supports hassle-free development of neural networks, the goal of course being optimization of predictive accuracy on unseen patterns.
3. History option allows users to capture their favorite keystrokes and save them. Easy recall for future use.
4. Provides symbolic interface which allows the user to create:Input and output definition files, Pattern files, and Help files for objects (i.e., inputs, input values, and outputs).
5. Supports categorization of inputs. This allows the user to readily access inputs via a popup menu within the main TDL menu. The hierarchical structure of the popup menu is under the full control of the application developer (i. e., user).
6. Symbolic object manipulation tool: Allows the user to interactively design the input/output structure of an application. The user can create, delete, or modify inputs, outputs, input values, and categories.
7. Supports Rule representation: (a) Extends standard Boolean operators (i.e., and, or, not) to contain several quantifiers (i.e., atmost, atleast, exactly, between). (b) Provides mechanisms for rule revision (i.e., refinement) and extraction. (c) Allows partial rule recognition. Supported are first- and best-fit.
8. Allows co-evolution of different subpopulations (based on type of transfer function chosen for each subpopulation).
9. Provides three types of crossover operators: simple random, weighted and blocked.
10. Supports both one-shot as well as multi-shot learning. Multi-shot learning allows for the incremental acquisition of different data sets. A single expert network is constructed, capable of recognizing all the data sets supplied during learning. Quick context switching between different domains is possible.
11. Three types of local learning rules are included: perceptron, delta and fastprop.

12. Implements 7 types of unit transfer functions: simple threshold, sigmoid, sigmoid-squash, n-level threshold, new n-level-threshold, gaussian and linear.
13. Over a dozen statistics are collected during various batch training sessions. These can be viewed using the chart option.
14. A 140+ page hypertext on-line help menu is available.
15. A DEMONSTRATION of TDL can be invoked when initially starting the program.

21. NeurOn-Line

Built on Gensym Corp.'s G2(r), Gensym's NeurOn-Line(r) is a graphical, object-oriented software product that enables users to easily build neural networks and integrate them into G2 applications. NeurOn-Line is well suited for advanced control, data and sensor validation, pattern recognition, fault classification, and multivariable quality control applications. Gensym's NeurOn-Line provides neural net training and on-line deployment in a single, consistent environment. NeurOn-Line's visual programming environment provides pre-defined blocks of neural net paradigms that have been extended with specific features for real-time process control applications. These include: Backprop, Radial Basis Function, Rho, and Autoassociative networks. For more information on Gensym software, visit their home page at http://www.gensym.com.

22. Neuframe

```
Product: Neuframe v.4 - ActiveX Components
Company: Neusciences
Address: Unit 2
         Lulworth Business Centre
         Totton
         Southampton
         UK
         SO40 3WW
  Phone: +44 023 80 664011
  Email: sales@neusciences.com
    URL: http://www.neusciences.com
```

Neuframe is an easy-to-use, visual, object-oriented approach to problem solving, allowing the user to embed intelligent technologies within their applications. Neuframe provides features that enable businesses to investigate and apply Intelligence Technologies from an initial low cost experimentation platform, through to the building of embedded implementations using software components.

Includes:

- ODBC
- Data Pre-Processing
- Multi-Layered Perceptron
- Kohonen
- Kmeans
- Radial Basis Function
- Neuro-Fuzzy Logic
- Statistics and Graphics
- Code Extract and OLE for development use

Recommended Configuration - Windows 9X, NT4 or 2000, Pentium 200 with 64Mb Ram

Price: Commercial £1295 - Educational £777 (pounds sterling)

ActiveX Components

KMeans, Linear Regression, Data Encoder: Price £150 each (pounds sterling)

Multi-Layered Perceptron, Radial Basis Function, Projection Pursuit Regression, Kohonen: Price £350 each (pounds sterling)

23. OWL Neural Network Library (TM)

```
Product: OWL Neural Network Library (TM)
Company: HyperLogic Corporation
Address: PO Box 300010
         Escondido, CA 92030
         USA
  Phone: +1 619-746-2765
    Fax: +1 619-746-4089
  Email: prodinfo@hyperlogic.com
    URL: http://www.hyperlogic.com/owl.html
```

The OWL Neural Network Library provides a set of popular networks in the form of a programming library for C or C++ software development. The library is designed to support engineering applications as well as academic research efforts.

A common programming interface allows consistent access to the various paradigms. The programming environment consists of functions for creating, deleting, training, running, saving, and restoring networks, accessing node states and weights, randomizing weights, reporting the complete network state in a printable ASCII form, and formatting detailed error message strings.

The library includes 20 neural network paradigms, and facilitates the construction of others by concatenation of simpler networks. Networks included are:

- Adaptive Bidirectional Associative Memories (ABAM), including stochastic versions (RABAM). Five paradigms in all.
- Discrete Bidirectional Associative Memory (BAM), with individual bias weights for increased pattern capacity.
- Multi-layer Backpropagation with many user controls such as batching, momentum, error propagation for network concatenation, and optional computation of squared error. A compatible, non-learning integer fixed-point version is included. Two paradigms in all.
- Nonadaptive Boltzmann Machine and Discrete Hopfield Circuit.
- "Brain-State-in-a-Box" autoassociator.
- Competitive Learning Networks: Classical, Differential, and "Conscience" version. Three paradigms in all.
- Fuzzy Associative Memory (FAM).
- "Hamming Network", a binary nearest-neighbor classifier.
- Generic Inner Product Layer with user-defined signal function.
- "Outstar Layer" learns time-weighted averages. This network, concatenated with Competitive Learning, yields the "Counterpropagation" network.
- "Learning Logic" gradient descent network, due to David Parker.
- Temporal Access Memory, a unidirectional network useful for recalling binary pattern sequences.
- Temporal Pattern Network, for learning time-sequenced binary patterns.

Supported Environments:
The object code version of OWL is provided on MS-DOS format diskettes with object libraries and makefiles for both Borland and Microsoft C. An included Windows DLL supports OWL development under Windows. The package also includes Owgraphics, a mouseless graphical user interface support library for DOS.
Both graphical and "stdio" example programs are included.
The Portable Source Code version of OWL compiles without change on many hosts, including VAX, UINX, and Transputer. The source code package includes the entire object-code package.

Price:
USA and Canada: (US) $295 object code, (US) $995 with source
Outside USA and Canada: (US) $350 object code, (US) $1050 with source
Shipping, taxes, duties, etc., are the responsibility of the customer.

24. Neural Connection

Product: Neural Connection
Company: SPSS Inc.
Address: 444 N. Michigan Ave., Chicago, IL 60611
Phone: 1-800-543-2185
1-312-329-3500 (U.S. and Canada)
Fax: 1-312-329-3668 (U.S. and Canada)
Email: sales@spss.com
URL: http://www.spss.com

SPSS has offices worldwide. For inquiries outside the U.S. and Canada, please contact the U.S. office to locate the office nearest you.

Operating system : Microsoft Windows 3.1 (runs in Windows 95)
System requirements: 386 pc or better, 4 MB memory (8MB recommended), 4 MB free hard disk space, VGA or SVGA monitor, Mouse or other pointing device, Math coprocessor strongly recommended
Price: $995, academic discounts available

Description
Neural Connection is a graphical neural network tool which uses an icon-based workspace for building models for prediction, classification, time series forecasting and data segmentation. It includes extensive data management capabilities so your data preparation is easily done right within Neural Connection. Several output tools give you the ability to explore your models thoroughly so you understand your results.

Modeling and Forecasting tools
* 3 neural network tools: Multi-Layer Perceptron, Radial Basis Function, Kohonen network
* 3 statistical analysis tools: Multiple linear regression, Closest class means classifier, Principal component analysis

Data Management tools
* Filter tool: transformations, trimming, descriptive statistics, select/deselect variables for analysis, histograms
* Time series window: single- or multi-step prediction, adjustable step size
* Network combiner
* Simulator
* Split dataset: training, validation and test data
* Handles missing values

Output Options
* Text output: writes ASCII and SPSS (.sav) files, actual values, probabilities, classification results table, network output
* Graphical output: 3-D contour plot, rotation capabilties, WhatIf? tool includes interactive sensititivity plot, cross section, color contour plot
* Time series plot

Production Tools
* Scripting language for batch jobs and interactive applications
* Scripting language for building applications

Documentation
* User s guide includes tutorial, operations and algorithms

```
* Guide to neural network applications

Example Applications
* Finance - predict account attrition
* Marketing - customer segmentation
* Medical - predict length of stay in hospital
* Consulting - forecast construction needs of federal court systems
* Utilities - predict number of service requests
* Sales - forecast product demand and sales
* Science - classify climate types
```

25. Pattern Recognition Workbench Expo/PRO/PRO+

Name: Pattern Recognition Workbench Expo/PRO/PRO+
Company: Unica Technologies, Inc.
Address: 55 Old Bedford Rd., Lincoln, MA 01773 USA
Phone, Fax: (617) 259-5900, (617) 259-5901
Email: unica@unica-usa.com

Basic capabilities:

- Supported architectures and training methods include backpropagation, radial basis functions, K nearest neighbors, Gaussian mixture, Nearest cluster, K means clustering, logistic regression, and more.
- *Experiment managers* interactively control model development by walking you through problem definition and set-up;
  - Provides icon-based management of experiments and reports.
  - Easily performs automated input feature selection searches and automated algorithm parameter searches (using intelligent search methods including genetic algorithms)
  - Statistical model validation (cross-validation, bootstrap validation, sliding-window validation).
- "*Giga-spreadsheets*" hold 16,000 columns by 16 million rows of data each (254 billion cells)!
- Intelligent spreadsheet supports data preprocessing and manipulation with over 100 built-in macro functions. Custom *user functions* can be built to create a library of re-usable macro functions.
- C source code generation, DLLs, and real-time application linking via DDE/OLE links.
- Interactive graphing and data visualization (line, histogram, 2D and 3D scatter graphs).

Operating system: Windows 3.1, WFW 3.11, Windows 95, Windows NT (16- and 32-bit versions available)

System requirements: Intel 486+, 8+ MB memory, 5+ MB disk space

Approx. price: software starts at $995.00 (call for more info)
Solving Pattern Recognition Problems text book: $49.95
Money-back guarantee

Comments: Pattern Recognition Workbench (PRW) is a comprehensive environment/tool for solving pattern recognition problems using neural network, machine learning, and traditional statistical technologies. With an intuitive, easy-to-use graphical interface, PRW has the flexibility to address many applications. With features such as automated model generation (via input feature selection and algorithm parameter searches), experiment management, and statistical validation, PRW provides all the necessary tools from formatting and preprocessing your data to setting up, running, and evaluating experiments, to deploying your solution. PRW's automated model generation capability can generate literally *hundreds* of models, selecting the best ones from a thorough search space, ultimately resulting in better solutions!

26.

PREVia

PREVia is a simple Neural Network-based forecasting tool. The current commercial version is available in French and English (the downloadable version is in English). A working demo version of PREVia is available for download at: http://www.elseware.fr/.

Introducing Previa

Based on a detailed analysis of the forecasting decision process, Previa was jointly designed and implemented by experts in economics and finance, and neural network systems specialists including both mathematicians and computer scientists. Previa thus enables the experimentation, testing, and validation of numerous models. In a few hours, the forecasting expert can conduct a systematic experimentation, generate a study report, and produce an operational forecasting model. The power of Previa stems from the model type used, i.e., neural networks. Previa offers a wide range of model types, hence allowing the user to create and test several forecasting systems, and to assess each of them with the same set of criteria. In this way, Previa offers a working environment where the user can rationalise his or her decision process. The hardware requirements of Previa are: an IBM-compatible PC with Windows 3.1 (c) or Windows95. For the best performance, an Intel 486DX processor is recommended. Previa is delivered as a shrink-wrapped application software, as well as a dynamic link library (DLL) for the development of custom software. The DLL contains all the necessary functions and data structures to manipulate time series, neural networks, and associated algorithms. The DLL can also be used to develop applications with Visual BasicTM. A partial list of features:

```
* Definition of a forecast equation : *
  Definition of the variable to forecast and explanatory variables.
  Automatic harmonisation of the domains and periodicities involved in
     the equation.
* Choice of a neuronal model associated with the forecasting equation :  *
  Automatic or manual definition of multi-layered architectures.
  Temporal models with loop-backs of intermediate layers.
* Fine-tuning of a neuronal model by training *
  Training by gradient back-propagation.
  Automatic simplification of architectures.
  Definition of training objectives by adaptation of the optimisation
     criterion.
  Definition of model form constraints.
  Graphing of different error criteria.
* Analysis of a neuronal model: *
  View of Hinton graph associated with each network layer.
  Connection weight editing.
  Calculation of sensitivity and elasticity of the variable to forecast,
     in relation to the explanatory variables.
  Calculation of the hidden series produced by the neural network.
* Neural Network-Based Forecasting *
  Operational use of a neural network.
* Series Analysis *
  Visualisation of a series curve.  Editing of the series values.
  Smoothing (simple, double, Holt & Winters)
  Study of the predictability of a series (fractal dimension)
  Comparison of two series.  Visualisation of X-Y graphs.
```

27. Trajan 2.0 Neural Network Simulator

Trajan Software Ltd,
Trajan House,
68 Lesbury Close,
Chester-le-Street,
Co. Durham,
DH2 3SR,
United Kingdom.

WWW: http://www.trajan-software.demon.co.uk

Email: andrew@trajan-software.demon.co.uk

Tel: +44 191 388 5737. (8:00-22:00 GMT).

Features.

Trajan 2.1 Professional is a Windows-based Neural Network includes support for a wide range of Neural Network types, training algorithms, and graphical and statistical feedback on Neural Network performance.

Features include:

1. **Full 32-bit power.** Trajan 2.1 is available in a 32-bit version for use on Windows 95 and Windows NT platforms, supporting virtually-unlimited network sizes (available memory is a constraint). A 16-bit version (network size limited to 8,192 units per layer) is also available for use on Windows 3.1.
2. **Network Architectures.** Includes Support for Multilayer Perceptrons, Kohonen networks, Radial Basis Functions, Linear models, Probabilistic and Generalised Regression Neural Networks. Training algorithms include the very fast, modern Levenburg-Marquardt and Conjugate Gradient Descent algorithms, in addition to Back Propagation (with time-dependent learning rate and momentum, shuffling and additive noise), Quick Propagation and Delta-Bar-Delta for Multilayer Perceptrons; K-Means, K-Nearest Neighbour and Pseudo-Inverse techniques for Radial Basis Function networks, Principal Components Analysis and specialised algorithms for Automatic Network Design and Neuro-Genetic Input Selection. Error plotting, automatic cross verification and a variety of stopping conditions are also included.
3. **Custom Architectures.** Trajan allows you to select special Activation functions and Error functions; for example, to use Softmax and Cross-entropy for Probability Estimation, or City-Block Error function for reduced outlier-sensitivity. There are also facilities to "splice" networks together and to delete layers from networks, allowing you to rapidly create pre- and post-processing networks, including Autoassociative Dimensionality Reduction networks.
4. **Simple User Interface.** Trajan's carefully-designed interface gives you access to large amounts of information using Graphs, Bar Charts and Datasheets. Trajan automatically calculates overall statistics on the performance of networks in both classification and regression. Virtually all information can be transferred via the Clipboard to other Windows applications such as Spreadsheets.
5. **Pre- and Post-processing.** Trajan 2.1 supports a range of pre- and post-processing options, including Minimax scaling, Winner-takes-all, Unit-Sum and Unit-Length vector. Trajan also assigns classifications based on user-specified Accept and Reject thresholds.

6. **Embedded Use.** The Trajan Dynamic Link Library gives full programmatic access to Trajan's facilities, including network creation, editing and training. Trajan 2.1 come complete with sample applications written in 'C' and Visual Basic.

There is also a demonstration version of the Software available; please download this to check whether Trajan 2.1 fulfils your needs.

28. DataEngine

```
Product: DataEngine, DataEngine ADL, DataEngine V.i

Company: MIT GmbH
Address: Promenade 9
         52076 Aachen
         Germany

  Phone: +49 2408 94580
    Fax: +49 2408 94582
  EMail: mailto:info@mitgmbh.de
    URL:
```

http://www.mitgmbh.de DataEngine is a software tool for data analysis implementing Fuzzy Rule Based Systems, Fuzzy Cluster Methods, Neural Networks, and Neural-Fuzzy Systems in combination with conventional methods of mathematics, statistics, and signal processing. DataEngine ADL enables you to integrate classifiers or controllers developed with DataEngine into your own software environment. It is offered as a DLL for MS/Windows or as a C++ library for various platforms and compilers. DataEngine V.i is an add-on tool for LabView (TM) that enables you to integrate Fuzzy Logic and Neural Networks into LabView through virtual instruments to build systems for data analysis as well as for Fuzzy Control tasks.

29. Machine Consciousness Toolbox

Can a machine help you understand the mechanisms of consciousness?
A visual, interactive application for investigating artificial consciousness as inspired by the biological brain.
Free, fully functional, introductory version with user manual and tutorials to download.
Based on the MAGNUS neural architecture developed at Imperial College, London, UK.
Developed by Novel Technical Solutions.
Full information and download from: http://www.sonnet.co.uk/nts

[Note from FAQ maintainer: While this product explores some of the prerequisites of consciousness in an interesting way, it does not deal with deeper philosophical issues such as qualia.]

30. Professional Basis of AI Backprop

Backprop, rprop, quickprop, delta-bar-delta, supersab, recurrent networks for Windows 95 and Unix/Tcl/Tk. Includes C++ source, examples, hypertext documentation and the ability to use trained networks in C++ programs. $30 for regular people and $200 for businesses and government agencies. For details see: http://www.dontveter.com/probp/probp.html or http://www.unidial.com/~drt/probp/probp.html

Questions to: Don Tveter, don@dontveter.com

31. STATISTICA: Neural Networks

```
Product: STATISTICA: Neural Networks version 4.0
Company: StatSoft, Inc.
Address: 2300 E. 14th St.
         Tulsa, OK  74104
         USA
  Phone: (918) 749-1119
    Fax: (918) 749-2217
  Email: info@statsoft.com
    URL: http://www.statsoft.com/
```

STATISTICA Neural Networks is a comprehensive application capable of designing a wide range of neural network architectures, employing both widely-used and highly-specialized training algorithms. STATISTICA Neural Networks was developed by StatSoft, Inc., the makers of STATISTICA software, and is available and supported through a world-wide network of StatSoft subsidiaries.

The current version STATISTICA Neural Networks 4.0 offers features such as sophisticated training algorithms, an Intelligent Problem Solver that walks the user step-by-step through the analysis, a Neuro-Genetic Input Selection facility, a large selection of supplementary graphs and statistics (e.g., ROC, Response Surfaces, Sensitivity Analysis, Regression and Classification statistics), complete support for API (Application Programming Interface), and the ability to interface with STATISTICA data files and graphs.

STATISTICA Neural Networks includes traditional learning algorithms, such as back propagation and sophisticated training algorithms such as Conjugate Gradient Descent and Levenberg-Marquardt iterative procedures. Typically, choosing the right architecture of a neural network is a difficult and time-consuming "trial and error" process, but

STATISTICA Neural Networks specifically does this for the user. STATISTICA Neural Networks features an Intelligent Problem Solver that utilizes heuristics and sophisticated optimization strategies to determine the best network architecture and walks the user step-by-step through the analysis. The Intelligent Problem Solver compares different network types (including Linear, Radial Basis Function, Multilayer Perceptron, and Bayesian networks), determines the number of hidden units, and chooses the Smoothing factor for Radial Basis Function networks.

The process of obtaining the right input variables in exploratory data analysis -- typically the case when neural networks are used -- also is facilitated by STATISTICA Neural Networks. Neuro-Genetic Input Selection procedures aid in determining the input variables that should be used in training the network. It uses an optimization strategy to compare the possible combinations of input variables to determine which set is most effective. STATISTICA Neural Networks offers complete API support so advanced users (or designers of corporate "knowledge seeking" or "data mining" systems) may be able to integrate the advanced computational engines of the Neural Networks module into their custom applications.

STATISTICA Neural Networks can be used as a stand-alone application or can interface directly with STATISTICA. It reads and writes STATISTICA data files and graphs.

32. Braincel (Excel add-in)

```
Product: Braincel
Company: Promised Land Technologies
Address: 195 Church Street 11th Floor
         New Haven, CT 06510
         USA
  Phone: (800) 243-1806 (Outside USA: (203) 562-7335)
    Fax: (203) 624-0655
  Email: gideon@miracle.net
    URL: http://www.promland.com/
         also see http://www.jurikres.com/catalog/ms_bcel.htm
```

Braincel is an add-in to Excel using a training method called backpercolation.

33. DESIRE/NEUNET

```
Product: DESIRE/NEUNET
Company: G.A. and T.M. Korn Industrial Consultants
Address: 7750 South Lakeshore Road, # 15
         Chelan, WA 98816
         USA
  Phone: (509) 687-3390
  Email: gatmkorn@aol.com
    URL: http://members.aol.com/gatmkorn
  Price: $ 775 (PC)
         $ 2700 (SPARCstation)
         $ 1800 (SPARCstation/educational)
         Free educational version
```

DESIRE (Direct Executing SImulation in REal time) is a completely interactive system for dynamic-system simulation (up to 6,000 differential equations plus up to 20,000 difference equations in scalar or matrix form; 13 integration rules) for control, aerospace, and chemical engineering, physiological modeling, and ecology. Easy programming of multirun studies (statistics, optimization). Complex frequency-response plots, fast Fourier transforms, screen editor, connection to database program.

DESIRE/NEUNET adds interactive neural-network simulation (to 20,000 interconnections) and fuzzy logic. Simulates complete dynamic systems controlled by neural networks and/or fuzzy logic. Users can develop their own neural networks using an easily readable matrix notation, as in:

```
VECTOR layer2= tanh(W * layer1 + bias)
```

Over 200 examples include:

- backpropagation, creeping random search
- Hopfield networks, bidirectional associative memories
- perceptrons, transversal filters and predictors
- competitive learning, counterpropagation
- very fast emulation of adaptive resonance
- simulates fuzzy-logic control and radial-basis functions

SCREEN-EDITED PROGRAMS RUN IMMEDIATELY AT HIGH SPEED

Package includes 2 textbooks by G.A. Korn: "Neural Networks and Fuzzy-logic Control on Personal Computers and Workstations" (MIT Press, 1995), and "Interactive Dynamic-system Simulatiuon under Windows 95 and NT" (Gordon and Breach, 1998). Please see our Web site for complete tables of contents of books, screen shots, list of users: http://members.aol.com/gatmkorn

Complete educational versions of DESIRE/NEUNET for Windows 95 and NT can now be downloaded FREE as a .zip file which you un-zip to get an automatic INSTALLSHIELD installation program. The educational version is identical to our full industrial version except for smaller data areas (6 instead of 6000 differential equations, smaller neural networks). It will run most examples from both textbooks.

34. Viscovery SOMine

```
          Product: Viscovery SOMine
          Company: eudaptics software gmbh
          Address: Helferstorferstr. 5/8
                   A-1010 Vienna
                   AUSTRIA
            Phone: (+43 1) 532 05 70
              Fax: (+43 1) 532 05 70 -21
           e-mail: office@eudaptics.co.at
              URL: http://www.eudaptics.co.at/
Operating System: Windows 95, Windows NT 4.0
           Prices: $1495 (commercial),
                   $695 (non-commercial, i.e. universities)
```

Viscovery SOMine is a powerful and easy-to-use tool for exploratory data analysis and data mining. Employing an enhanced version of Self Organizing Maps it puts complex data into order based on its similarity. The resulting map can be used to identify and evaluate the features hidden in the data. The result is presented in a graphical way which allows the user to analyze non-linear relationships without requiring profound statistical knowledge. The system further supports full pre- and postprocessing, cluster search, association/recall, prediction, statistics, filtering, and animated system state monitoring. Through the implementation of techniques such as SOM scaling, the speed in creating maps is increased compared to the original SOM algorithm.

Download a free evaluation copy from http://www.eudaptics.co.at/demo.htm

35. NeuNet Pro

```
Product: NeuNet Pro
Company: CorMac Technologies Inc.
Address: 34 North Cumberland Street
         Thunder Bay, ON P7A 4L4
         Canada
  Phone: (807) 345-7114
    Fax: (807) 345-7114
```

```
Email: sales@cormactech.com
  URL: http://www.cormactech.com
```

NeuNet Pro is a complete neural network development system:

- Requires Windows 95, Windows 98, or Windows NT4(sp3).
- Powerful, easy to use, point and click, graphical development environment.
- Choose either SFAM or Back Propagation.
- Access data directly from MDB database file.
- Data may contain up to 255 fields.
- Split data into training set and testing set, up to 32,000 rows each.
- Comprehensive graphical reporting of prediction accuracy.
- Table browse, confusion matrix, scatter graph, and time series graph.
- Context sensitive help file with 70 page manual.
- Includes eight sample projects and assortment of sample data.
- Additional sample data available from URL above.
- Entire program may be downloaded from URL above.

36. Neuronics

```
   Product: NNetView
   Company: Neuronics Inc.
   Address: Technoparkstr. 1
            CH-8005 Zurich Switzerland
     Phone: +41 1 445 16 40
       Fax: +41 1 445 16 44
       URL: http://www.neuronics.com
Operating systems: Windows 95/98/NT and on MacOS.
    Prices: 120 EUR for Simulator
            310 EUR for Full version incl. camera- and serial interface
            1260 EUR for Class licence
```

NNetView is a simulator for neural networks that allows you to connect the network directly to the video camera input as well as the input/output of the serial port of your PC. In addition, the network can include a free architecture of 2-dimensional layers with connections in all directions. The most interesting feature is that you can combine different standard network types (such as Backpropagation, Hebb, Reinforcement Learning, Kohonen or Hopfield networks). NNetView allows to learn images (such as faces or colored objects) online and to make robots and other machines adaptive.

37. RG Software

```
 Product: NN50.DLL Neural Network Software Development Kit (SDK)
 Company: RG Software Corporation
 Address: 6838 W. Cholla
          Peoria, Arizona 85345
   Phone: (623) 773-2396
     FAX: (623) 334-3421
   Email: sales@rgsoftware.com
     URL: http://www.rgsoftware.com


The NN50.DLL Neural Network SDK allows you to incorporate a quickprop
neural network in your favorite development environment (VB, FoxPro,
C++, Assembler, Java and just about any other Windows programming
language) with minimal investment and effort. NN50.DLL trains very fast,
saves and loads weights, can be used online, stopped and called during
training and more.  NN50's input relevance function is based on
information gathered from
```

ftp://ftp.sas.com/pub/neural/importance.html.
NN50 is compatible with Windows 95, Windows 98, Windows NT and Windows 2000. Unix and Linux versions are also available. Please see
http://www.rgsoftware.com
for details.

38. Cobalt A.I. Code Builder - Neural Network Edition

```
Product: Code Builder - Neural Network Edition
Company: Cobalt A.I. Software
  Phone: (623) 487-3813
  Email: sales@cobaltai.com
    URL: http://www.cobaltai.com
  Price: U.S. $59.00
```

Cobalt A.I. Code Builder - Neural Network Edition is a neural network source code generator. Design neural networks and generate efficient object oriented source code for C++, Java or Visual Basic (VB, Excel, Access, Word). Other languages (FoxPro, C# and VB.Net) will soon be integrated. Code Builder uses a "Project Wizard" to analyze data and suggest inputs, then generates source code accordingly.

39. NEURO MODEL and GenOpt

```
Product: NEURO MODEL and GenOpt
Company: ATLAN-tec KG
Address: Hans-Martin Schleyer Strasse 18A
         47877
         Germany
  Phone: +49 2154 92 48 222
    FAX: +49 2154 92 48 100
  Email: nm-sales@atlan-tec.com
    URL: http://www.atlan-tec.com
```

Neuro Model (NM) is a Windows based ANN Development Package which does not require any scientific knowledge of ANN. Designed for the process industry, NM showed its best performance with real world data sets. Preprocessing of all data sets by a proprietary cluster algorithm provide reliable and consistant information for training. The internal combination of different mathematical methods eliminates the problem of local optima and overfitting. Modelling of dynamic non-linearity with implemented time behaviour enables the package to predict process conditions online in complexe enviroments like chemical reactors. Extensive reports of all net details include information about training parameters and statistics alerts. Software validation conform FDA could be done by worldwide co-operation with Pharmaplan (FRESENIUS AG).
Ask for FREE (Fullversion on loan) student License for your project.

- Customized Components through DLLs
- Non Linear Genetic Optimization through GEN OPT
- Microsoft Excel Add-in - NM Runtimer for Excel
- Runtimer for Windows NT, 2000, UNIX, DEC-VAX, SOLARIS, LINUX
- Visual Data Selection
- Data Preprocessing, Analysis and Modification
- Batch Training and Parameter Optimization
- Extensive Information by different Sensitivity and Accuracy Analysis
- Security Net Algorithm shows statistical confidence for predicted results
- Comparison of predicted vs. desired results incl. confidence range for the whole topography
- Automated Report Generation incl. DES crypted fingerprint
- Sophisticated Graphic Machine

- Multi Language Library for international Corporate Licenses

Applications References available for:

- Pharma & Bioscience
- Clinical Medicine Research
- Chemical Industry
- Waste Water Treatment Plants
- Food Industry
- Plastic Processing
- Prediction of Energy Consumption Behaviour for Electricity / Water / Gas network

- -

Next part is part 7 (of 7). Previous part is part 5.

```
Archive-name: ai-faq/neural-nets/part7
Last-modified: 2002-04-09
URL: ftp://ftp.sas.com/pub/neural/FAQ7.html
Maintainer: saswss@unx.sas.com (Warren S. Sarle)
```


This is part 7 (of 7) of a monthly posting to the Usenet newsgroup comp.ai.neural-nets. See the part 1 of this posting for full information what it is all about.

========== Questions ==========

Subject: Neural Network hardware?

Overview articles:

- Ienne, Paolo and Kuhn, Gary (1995), "Digital Systems for Neural Networks", in Papamichalis, P. and Kerwin, R., eds., *Digital Signal Processing Technology*, Critical Reviews Series CR57 Orlando, FL: SPIE Optical Engineering, pp 314-45, ftp://mantraftp.epfl.ch/mantra/ienne.spie95.A4.ps.gz or ftp://mantraftp.epfl.ch/mantra/ienne.spie95.US.ps.gz
- ftp://ftp.mrc-apu.cam.ac.uk/pub/nn/murre/neurhard.ps (1995)
- ftp://ftp.urc.tue.nl/pub/neural/hardware_general.ps.gz (1993)

Various NN HW information can be found in the Web site http://www1.cern.ch/NeuralNets/nnwInHepHard.html (from people who really use such stuff!). Several applications are described in http://www1.cern.ch/NeuralNets/nnwInHepExpt.html

More information on NN chips can be obtained from the Electronic Engineers Toolbox web page. Go to http://www.eg3.com/ebox.htm, type "neural" in the quick search box, click on "chip co's" and then on "search".

Further WWW pointers to NN Hardware:

- http://msia02.msi.se/~lindsey/nnwAtm.html
- http://www.genobyte.com/article.html

Here is a short list of companies:

1. ## HNC, INC.

   ```
   HNC Inc.
   5930 Cornerstone Court West
   San Diego, CA 92121-3728

   619-546-8877  Phone
   619-452-6524  Fax
   ```

   HNC markets:
   - Database Mining Workstation (DMW), a PC based system that builds models of relationships and patterns in data.
   - The SIMD Numerical Array Processor (SNAP). It is an attached parallel array processor in a

VME chassis with between 16 and 64 parallel floating point processors. It provides between 640 MFLOPS and 2.56 GFLOPS for neural network and signal processing applications. A Sun SPARCstation serves as the host. The SNAP won the IEEE 1993 Gordon Bell Prize for best price/performance for supercomputer class systems.

2. SAIC (Sience Application International Corporation)

```
10260 Campus Point Drive
MS 71, San Diego
CA 92121
(619) 546 6148
Fax: (619) 546 6736
```

3. Micro Devices

```
30 Skyline Drive
Lake Mary
FL 32746-6201
(407) 333-4379
```

MicroDevices makes MD1220 - 'Neural Bit Slice'. Each of the products mentioned sofar have very different usages. Although this sounds similar to Intel's product, the architectures are not.

4. Intel Corp

```
2250 Mission College Blvd
Santa Clara, Ca 95052-8125
Attn ETANN, Mail Stop SC9-40
(408) 765-9235
```

Intel was making an experimental chip (which is no longer produced): 80170NW - Electrically trainable Analog Neural Network (ETANN) It has 64 'neurons' on it - almost fully internally connectted and the chip can be put in an hierarchial architecture to do 2 Billion interconnects per second. Support software by

```
California Scientific Software
10141 Evening Star Dr #6
Grass Valley, CA 95945-9051
(916) 477-7481
```

Their product is called 'BrainMaker'.

5. Tubb Research Limited

```
7a Lavant Street
Peterfield
Hampshire
GU32 2EL
```

```
United Kingdom
Tel: +44 730 60256
```

6. **Adaptive Solutions Inc**

```
1400 NW Compton Drive
Suite 340
Beaverton, OR 97006
U. S. A.
Tel: 503-690-1236;   FAX: 503-690-1249
```

7. **NeuroDynamX, Inc.**

```
P.O. Box 14
Marion, OH  43301-0014
Voice (740) 387-5074
Fax: (740) 382-4533
Internet:  jwrogers@on-ramp.net
http://www.neurodynamx.com
```

InfoTech Software Engineering purchased the software and trademarks from NeuroDynamX, Inc. and, using the NeuroDynamX tradename, continues to publish the DynaMind, DynaMind Developer Pro and iDynaMind software packages.

8. **NeuroClassifier**

URL: http://www.ice.el.utwente.nl/Finished/Neuro/
Email: peter.masa@csemne.ch

9. **NeuriCam, S.r.l.**

```
Via S. Maria Maddalena, 38100 Trento, Italy
Tel: +39 0461 260 552
Fax: +39 0461 260 617
Email: info@neuricam.com
```

NC3001 TOTEM - Digital Processor for Neural Networks
TOTEM is a digital VLSI parallel processor for fast learning and recognition with artificial neural networks. Its high processing power, low power dissipation and limited chip size make it ideally suited for embedded applications. The architecture is optimised for the implementation of the Reactive Tabu Search learning algorithm, a competitive alternative to back-propagation which leads to a very compact VLSI implementation.

And here is an incomplete overview of known Neural Computers with their newest known reference.

- Special Computers

1. AAP-2 Takumi Watanabe, Yoshi Sugiyama, Toshio Kondo, and Yoshihiro Kitamura.
   "Neural network simulation on a massively parallel cellular array processor: AAP-2." In International Joint Conference on Neural Networks, 1989.
2. ANNA B.E.Boser, E.Sackinger, J.Bromley, Y.leChun, and L.D.Jackel.
   "Hardware Requirements for Neural Network Pattern Classifiers." In *IEEE Micro*, 12(1), pages 32-40, February 1992.
3. Analog Neural Computer Paul Mueller et al.
   "Design and performance of a prototype analog neural computer." In Neurocomputing, 4(6):311-323, 1992.
4. APx -- Array Processor Accelerator F.Pazienti.
   "Neural networks simulation with array processors." In *Advanced Computer Technology, Reliable Systems and Applications; Proceedings of the 5th Annual Computer Conference*, pages 547-551. IEEE Comput. Soc. Press, May 1991. ISBN: 0-8186-2141-9.
5. ASP -- Associative String Processor A.Krikelis.
   "A novel massively associative processing architecture for the implementation artificial neural networks." In *1991 International Conference on Acoustics, Speech and Signal Processing*, volume 2, pages 1057-1060. IEEE Comput. Soc. Press, May 1991.
6. BSP400 Jan N.H. Heemskerk, Jacob M.J. Murre, Jaap Hoekstra, Leon H.J.G. Kemna, and Patrick T.W. Hudson.
   "The bsp400: A modular neurocomputer assembled from 400 low-cost microprocessors." In International Conference on Artificial Neural Networks. Elsevier Science, 1991.
7. BLAST J.G.Elias, M.D.Fisher, and C.M.Monemi.
   "A multiprocessor machine for large-scale neural network simulation." In *IJCNN91-Seattle: International Joint Conference on Neural Networks*, volume 1, pages 469-474. IEEE Comput. Soc. Press, July 1991. ISBN: 0-7883-0164-1.
8. CNAPS Neurocomputer H.McCartor
   "Back Propagation Implementation on the Adaptive Solutions CNAPS Neurocomputer." In *Advances in Neural Information Processing Systems*, 3, 1991.
9. GENES~IV and MANTRA~I Paolo Ienne and Marc A. Viredaz
   "GENES~IV: A Bit-Serial Processing Element for a Multi-Model Neural-Network Accelerator." Journal of VLSI Signal Processing, volume 9, no. 3, pages 257--273, 1995.
10. MA16 -- Neural Signal Processor U.Ramacher, J.Beichter, and N.Bruls.
    "Architecture of a general-purpose neural signal processor." In *IJCNN91-Seattle: International Joint Conference on Neural Networks*, volume 1, pages 443-446. IEEE Comput. Soc. Press, July 1991. ISBN: 0-7083-0164-1.
11. Mindshape Jan N.H. Heemskerk, Jacob M.J. Murre Arend Melissant, Mirko Pelgrom, and Patrick T.W. Hudson.
    "Mindshape: a neurocomputer concept based on a fractal architecture." In International Conference on Artificial Neural Networks. Elsevier Science, 1992.
12. mod 2 Michael L. Mumford, David K. Andes, and Lynn R. Kern.
    "The mod 2 neurocomputer system design." In IEEE Transactions on Neural Networks, 3(3):423-433, 1992.
13. NERV R.Hauser, H.Horner, R. Maenner, and M.Makhaniok.
    "Architectural Considerations for NERV - a General Purpose Neural Network Simulation System." In *Workshop on Parallel Processing: Logic, Organization and Technology -- WOPPLOT 89*, pages 183-195. Springer Verlag, Mars 1989. ISBN: 3-5405-5027-5.
14. NP -- Neural Processor D.A.Orrey, D.J.Myers, and J.M.Vincent.
    "A high performance digital processor for implementing large artificial neural networks." In *Proceedings of of the IEEE 1991 Custom Integrated Circuits Conference*, pages 16.3/1-4. IEEE Comput. Soc. Press, May 1991. ISBN: 0-7883-0015-7.
15. RAP -- Ring Array Processor N.Morgan, J.Beck, P.Kohn, J.Bilmes, E.Allman, and J.Beer.

"The ring array processor: A multiprocessing peripheral for connectionist applications." In *Journal of Parallel and Distributed Computing*, pages 248-259, April 1992.

16. RENNS -- REconfigurable Neural Networks Server O.Landsverk, J.Greipsland, J.A.Mathisen, J.G.Solheim, and L.Utne.
"RENNS - a Reconfigurable Computer System for Simulating Artificial Neural Network Algorithms." In *Parallel and Distributed Computing Systems, Proceedings of the ISMM 5th International Conference*, pages 251-256. The International Society for Mini and Microcomputers - ISMM, October 1992. ISBN: 1-8808-4302-1.
17. SMART -- Sparse Matrix Adaptive and Recursive Transforms P.Bessiere, A.Chams, A.Guerin, J.Herault, C.Jutten, and J.C.Lawson.
"From Hardware to Software: Designing a `Neurostation'." In *VLSI design of Neural Networks*, pages 311-335, June 1990.
18. SNAP -- Scalable Neurocomputer Array Processor E.Wojciechowski.
"SNAP: A parallel processor for implementing real time neural networks." In *Proceedings of the IEEE 1991 National Aerospace and Electronics Conference; NAECON-91*, volume 2, pages 736-742. IEEE Comput.Soc.Press, May 1991.
19. Toroidal Neural Network Processor S.Jones, K.Sammut, C.Nielsen, and J.Staunstrup.
"Toroidal Neural Network: Architecture and Processor Granularity Issues." In *VLSI design of Neural Networks*, pages 229-254, June 1990.
20. SMART and SuperNode P. Bessi`ere, A. Chams, and P. Chol.
"MENTAL : A virtual machine approach to artificial neural networks programming." In NERVES, ESPRIT B.R.A. project no 3049, 1991.

- Standard Computers

1. EMMA-2 R.Battiti, L.M.Briano, R.Cecinati, A.M.Colla, and P.Guido.
"An application oriented development environment for Neural Net models on multiprocessor Emma-2." In *Silicon Architectures for Neural Nets; Proceedings for the IFIP WG.10.5 Workshop*, pages 31-43. North Holland, November 1991. ISBN: 0-4448-9113-7.
2. iPSC/860 Hypercube D.Jackson, and D.Hammerstrom
"Distributing Back Propagation Networks Over the Intel iPSC/860 Hypercube" In *IJCNN91-Seattle: International Joint Conference on Neural Networks*, volume 1, pages 569-574. IEEE Comput. Soc. Press, July 1991. ISBN: 0-7083-0164-1.
3. SCAP -- Systolic/Cellular Array Processor Wei-Ling L., V.K.Prasanna, and K.W.Przytula.
"Algorithmic Mapping of Neural Network Models onto Parallel SIMD Machines." In *IEEE Transactions on Computers*, 40(12), pages 1390-1401, December 1991. ISSN: 0018-9340.

Subject: What are some applications of NNs?

There are vast numbers of published neural network applications. If you don't find something from your field of interest below, try a web search. Here are some useful search engines:
http://www.google.com/
http://search.yahoo.com/
http://www.altavista.com/
http://www.deja.com/

General

- The Pacific Northwest National Laboratory: http://www.emsl.pnl.gov:2080/proj/neuron/neural/ including a list of commercial applications at http://www.emsl.pnl.gov:2080/proj/neuron/neural/products/
- The Stimulation Initiative for European Neural Applications: http://www.mbfys.kun.nl/snn/siena/cases/
- The DTI NeuroComputing Web's Applications Portfolio: http://www.globalweb.co.uk/nctt/portfolo/
- The Applications Corner, NeuroDimension, Inc.: http://www.nd.com/appcornr/purpose.htm
- The BioComp Systems, Inc. Solutions page: http://www.bio-comp.com
- Chen, C.H., ed. (1996) *Fuzzy Logic and Neural Network Handbook*, NY: McGraw-Hill, ISBN 0-07-011189-8.
- The series *Advances in Neural Information Processing Systems* containing proceedings of the conference of the same name, published yearly by Morgan Kauffman starting in 1989 and by The MIT Press in 1995.

Agriculture

- P.H. Heinemann, Automated Grading of Produce: http://server.age.psu.edu/dept/fac/Heinemann/phhdocs/visionres.html
- Deck, S., C.T. Morrow, P.H. Heinemann, and H.J. Sommer, III. 1995. Comparison of a neural network and traditional classifier for machine vision inspection. Applied Engineering in Agriculture. 11(2):319-326.
- Tao, Y., P.H. Heinemann, Z. Varghese, C.T. Morrow, and H.J. Sommer III. 1995. Machine vision for color inspection of potatoes and apples. Transactions of the American Society of Agricultural Engineers. 38(5):1555-1561.

Automotive

- "No Hands Across America Journal" - steering a car: http://cart.frc.ri.cmu.edu/users/hpm/project.archive/reference.file/Journal.html
  Photos: http://www.techfak.uni-bielefeld.de/ags/ti/personen/zhang/seminar/intelligente-autos/tour.html

Chemistry

- PNNL, General Applications of Neural Networks in Chemistry and Chemical Engineering: http://www.emsl.pnl.gov:2080/proj/neuron/neural/bib/chemistry.html.
- Prof. Dr. Johann Gasteiger, Neural Networks and Genetic Algorithms in Chemistry: http://www2.ccc.uni-erlangen.de/publications/publ_topics/publ_topics-12.html
- Roy Goodacre, pyrolysis mass spectrometry: http://gepasi.dbs.aber.ac.uk/roy/pymshome.htm and Fourier transform infrared (FT-IR) spectroscopy: http://gepasi.dbs.aber.ac.uk/roy/ftir/ftirhome.htm contain applications of a variety of NNs as well as PLS (partial least squares) and other statistical methods.
- Situs, a program package for the docking of protein crystal structures to single-molecule, low-resolution maps from electron microscopy or small angle X-ray scattering: http://chemcca10.ucsd.edu/~situs/
- An on-line application of a Kohonen network with a 2-dimensional output layer for prediction of protein secondary structure percentages from UV circular dichroism spectra: http://www.embl-heidelberg.de/~andrade/k2d/.

Criminology

- Computer Aided Tracking and Characterization of Homicides and Sexual Assaults (CATCH): http://lancair.emsl.pnl.gov:2080/proj/neuron/papers/kangas.spie99.abs.html

Face recognition

- Face Recognition Home Page: http://www.cs.rug.nl/~peterkr/FACE/face.html
- Konen, W., "Neural information processing in real-world face-recognition applications," http://www.computer.muni.cz/pubs/expert/1996/trends/x4004/konen.htm
- Jiang, Q., "Principal Component Analysis and Neural Network Based Face Recognition," http://people.cs.uchicago.edu/~qingj/ThesisHtml/
- Lawrence, S., Giles, C.L., Tsoi, A.C., Back, A.D. (1997), "Face Recognition: A Convolutional Neural Network Approach," IEEE Transactions on Neural Networks, 8, 98-113, http://www.neci.nec.com/~lawrence/papers/face-tnn97/latex.html

Finance and economics

- Athanasios Episcopos, References on Neural Net Applications to Finance and Economics: http://www.compulink.gr/users/episcopo/neurofin.html
- Franco Busetti, Heuristics and artificial intelligence in finance and investment: http://www.geocities.com/francorbusetti/
- Trippi, R.R. & Turban, E. (1993), *Neural Networks in Finance and Investing*, Chicago: Probus.
- Zirilli, J.S. (1996), *Financial Prediction Using Neural Networks,* International Thomson Publishing, ISBN 1850322341, http://www6.bcity.com/mjfutures/
- Andreas S. Weigend, Yaser Abu-Mostafa, A. Paul N. Refenes (eds.) (1997) *Decision Technologies for Financial Engineering: Proceedings of the Fourth International Conference on Neural Networks in the Capital Markets* (Nncm '96) Publisher: World Scientific Publishing Company, ISBN: 9810231245

Games, sports, gambling

- General:

  Jay Scott, Machine Learning in Games: http://satirist.org/learn-game/index.html

  METAGAME Game-Playing Workbench: ftp://ftp.cl.cam.ac.uk/users/bdp/METAGAME

  R.S. Sutton, "Learning to predict by the methods of temporal differences", Machine Learning 3, p. 9-44 (1988).

  David E. Moriarty and Risto Miikkulainen (1994). "Evolving Neural Networks to Focus Minimax Search," In Proceedings of Twelfth National Conference on Artificial Intelligence (AAAI-94, Seattle, WA), 1371-1377. Cambridge, MA: MIT Press, http://www.cs.utexas.edu/users/nn/pages/publications/neuro-evolution.html

  Games World '99 at http://gamesworld99.free.fr/menuframe.htm

- Backgammon:

  G. Tesauro and T.J. Sejnowski (1989), "A Parallel Network that learns to play Backgammon," Artificial Intelligence, vol 39, pp. 357-390.

  G. Tesauro and T.J. Sejnowski (1990), "Neurogammon: A Neural Network Backgammon Program," IJCNN Proceedings, vol 3, pp. 33-39, 1990.

  G. Tesauro (1995), "Temporal Difference Learning and TD-Gammon," Communications of the ACM, 38, 58-68, http://www.research.ibm.com/massive/tdl.html

  Pollack, J.P. and Blair, A.D. (1997), "Co-Evolution in the Successful Learning of Backgammon Strategy," Brandeis University Computer Science Technical Report CS-97-193, http://www.demo.cs.brandeis.edu/papers/long.html#hcgam97

- Bridge:

  METAGAME: ftp://ftp.cl.cam.ac.uk/users/bdp/bridge.ps.Z

  He Yo, Zhen Xianjun, Ye Yizheng, Li Zhongrong (19??), "Knowledge acquisition and reasoning based on neural networks - the research of a bridge bidding system," INNC '90, Paris, vol 1, pp. 416-423.

  M. Kohle and F. Schonbauer (19??), "Experience gained with a neural network that learns to play bridge," Proc. of the 5th Austrian Artificial Intelligence meeting, pp. 224-229.

- Checkers/Draughts:

  Mark Lynch (1997), "NeuroDraughts: an application of temporal difference learning to draughts," http://www.ai.univie.ac.at/~juffi/lig/Papers/lynch-thesis.ps.gz Software available at http://satirist.org/learn-game/archive/NeuroDraughts-1.00.zip

  K. Chellapilla and D. B. Fogel, "Co-Evolving Checkers Playing Programs using Only Win, Lose, or Draw," SPIE's AeroSense'99: Applications and Science of Computational Intelligence II, Apr. 5-9, 1999, Orlando, Florida, USA, http://vision.ucsd.edu/~kchellap/Publications.html

  David Fogel (1999), *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (2nd edition), IEEE, ISBN: 078035379X

  David Fogel (2001), *Blondie24: Playing at the Edge of AI*, Morgan Kaufmann Publishers, ISBN: 1558607838
  According to the publisher, this is:

  > ... the first book to bring together the most advanced work in the general use of evolutionary computation for creative results. It is well suited for the general computer science audience.
  >
  > Here's the story of a computer that taught itself to play checkers far better than its creators ever could. Blondie24 uses a program that emulates the basic principles of Darwin

evolution to discover on its own how to excel at the game. Through this entertaining story, the book provides the reader some of the history of AI and explores its future.

Unlike Deep Blue, the celebrated chess machine that beat Garry Kasparov, the former world champion chess player, this evolutionary program didn't have access to other games played by human grand masters, or databases of moves for the endgame. It created its own means for evaluating the patterns of pieces that it experienced by evolving artificial neural networks--mathematical models that loosely describe how a brain works.

See http://www.natural-selection.com/NSIPublicationsOnline.htm for a variety of online papers by Fogel.

Not NNs, but classic papers:

A.L. Samuel (1959), "Some studies in machine learning using the game of checkers," IBM journal of Research and Development, vol 3, nr. 3, pp. 210-229.

A.L. Samuel (1967), "Some studies in machine learning using the game of checkers 2 - recent progress," IBM journal of Research and Development, vol 11, nr. 6, pp. 601-616.

- Chess:

  Sebastian Thrun, NeuroChess: http://satirist.org/learn-game/systems/neurochess.html

  Luke Pellen, Octavius: http://home.seol.net.au/luke/octavius/

  Louis Savain (AKA Nemesis), Animal, a spiking neural network that the author hopes will learn to play a passable game of chess after he implements the motivation mechanism: http://home1.gte.net/res02khr/AI/Temporal_Intelligence.htm

- Dog racing:

  H. Chen1, P. Buntin, L. She, S. Sutjahjo, C. Sommer, D. Neely (1994), "Expert Prediction, Symbolic Learning, and Neural Networks: An Experiment on Greyhound Racing," IEEE Expert, December 1994, 21-27, http://ai.bpa.arizona.edu/papers/dog93/dog93.html

- Football (Soccer):

  Kuonen Diego, "Statistical Models for Knock-out Soccer Tournaments", http://dmawww.epfl.ch/~kuonen/CALCIO/ (not neural nets, but relevant)

- Go:

  David Stoutamire (19??), "Machine Learning, Game Play, and Go," Center for Automation and Intelligent Systems Research TR 91-128, Case Western Reserve University. http://www.stoutamire.com/david/publications.html

  David Stoutamire (1991), *Machine Learning Applied to Go*, M.S. thesis, Case Western Reserve

University, ftp://ftp.cl.cam.ac.uk/users/bdp/go.ps.Z

Schraudolph, N., Dayan, P., Sejnowski, T. (1994), "Temporal Difference Learning of Position Evaluation in the Game of Go," In: Neural Information Processing Systems 6, Morgan Kaufmann 1994, ftp://bsdserver.ucsf.edu/Go/comp/td-go.ps.Z

P. Donnelly, P. Corr & D. Crookes (1994), "Evolving Go Playing Strategy in Neural Networks", AISB Workshop on Evolutionary Computing, Leeds, England, ftp://www.joy.ne.jp/welcome/igs/Go/computer/egpsnn.ps.Z or ftp://ftp.cs.cuhk.hk/pub/neuro/GO/techreports/egpsnn.ps.Z

Markus Enzenberger (1996), "The Integration of A Priori Knowledge into a Go Playing Neural Network," http://www.cgl.ucsf.edu/go/Programs/neurogo-html/neurogo.html

Norman Richards, David Moriarty, and Risto Miikkulainen (1998), "Evolving Neural Networks to Play Go," Applied Intelligence, 8, 85-96, http://www.cs.utexas.edu/users/nn/pages/publications/neuro-evolution.html

Dahl, F. A. (1999), "Honte, a Go-playing program using neural nets", http://www.ai.univie.ac.at/icml-99-ws-games/papers/dahl.ps.gz

- Go-Moku:

  Freisleben, B., "Teaching a Neural Network to Play GO-MOKU," in I. Aleksander and J. Taylor, eds, Artificial Neural Networks 2, Proc. of ICANN-92, Brighton UK, vol. 2, pp. 1659-1662, Elsevier Science Publishers, 1992

  Katz, W.T. and Pham, S.P. "Experience-Based Learning Experiments using Go-moku", Proc. of the 1991 IEEE International Conference on Systems, Man, and Cybernetics, 2: 1405-1410, October 1991.

- Olympics:

  E.M.Condon, B.L.Golden, E.A.Wasil (1999), "Predicting the success of nations at the Summer Olympics using neural networks", Computers & Operations Research, 26, 1243-1265.

- Pong:

  http:// www.engin.umd.umich.edu/~watta/MM/pong/pong5.html

- Reversi/Othello:

  David E. Moriarty and Risto Miikkulainen (1995). Discovering Complex Othello Strategies through Evolutionary Neural Networks. Connection Science, 7, 195-209, http://www.cs.utexas.edu/users/nn/pages/publications/neuro-evolution.html

  Yoshioka, T., Ishii, S., and Ito, M., Strategy acquisition for the game ``Othello" based on reinforcement learning, IEICE Transactions on Information and Systems E82-D 12, 1618-1626, 1999, http://mimi.aist-

nara.ac.jp/~taku-y/

- Tic-Tac-Toe/Noughts and Crosses:

  Fogel, David Bb (1993), "Using evolutionary programming to construct neural networks that are capable of playing tic-tac-toe," Intern. Conf. on Neural Networks 1993, IEEE, San Francisco, CA, pp. 875-880.

  Richard S. Sutton and Andrew G. Barto (1998), *Reinforcement Learning: An Introduction* The MIT Press, ISBN: 0262193981, http://www-anw.cs.umass.edu/~rich/book/the-book.html

  Yongzheng Zhang, Chen Teng, Sitan Wei (2000), "Game playing with Evolutionary Strategies and Modular Neural Networks: Tic-Tac-Toe," http://www.cs.dal.ca/~mheywood/GAPproject/EvolvingGamePlay.html

  Rob Ellison, "Neural Os and Xs," http://www.catfood.demon.co.uk/beta/game.html (An online Javascript demo, but you may not live long enough to teach the network to play a mediocre game. I'm not sure what kind of network it uses, but maybe you can figure that out if you read the source.)

Industry

- PNNL, Neural Network Applications in Manufacturing: http://www.emsl.pnl.gov:2080/proj/neuron/neural/bib/manufacturing.html.
- PNNL, Applications in the Electric Power Industry: http://www.emsl.pnl.gov:2080/proj/neuron/neural/bib/power.html.
- PNNL, Process Control: http://www.emsl.pnl.gov:2080/proj/neuron/neural/bib/process.html.
- Raoul Tawel, Ken Marko, and Lee Feldkamp (1998), "Custom VLSI ASIC for Automotive Applications with Recurrent Networks", http://www.jpl.nasa.gov/releases/98/ijcnn98.pdf
- Otsuka, Y. et al. "Neural Networks and Pattern Recognition of Blast Furnace Operation Data" Kobelco Technology Review, Oct. 1992, 12
- Otsuka, Y. et al. "Applications of Neural Network to Iron and Steel Making Processes" 2. International Conference on Fuzzy Logic and Neural Networks, Iizuka, 1992
- Staib, W.E. "Neural Network Control System for Electric Arc Furnaces" M.P.T. International, 2/1995, 58-61
- Portmann, N. et al. "Application of Neural Networks in Rolling Automation" Iron and Steel Engineer, Feb. 1995, 33-36
- Gorni, A.A. (2000), "The modelling of hot rolling processes using neural networks: A bibliographical review", http://www.geocities.com/SiliconValley/5978/neural_1998.html
- Murat, M. E., and Rudman, A. J., 1992, Automated first arrival picking: A neural network approach: Geophysical Prospecting, 40, 587-604.

Materials science

- Phase Transformations Research Group (search for "neural"): http://www.msm.cam.ac.uk/phase-trans/pubs/ptpuball.html

Medicine

- PNNL, Applications in Medicine and Health:

http://www.emsl.pnl.gov:2080/proj/neuron/neural/bib/medicine.html.

Music

- Mozer, M. C. (1994), "Neural network music composition by prediction: Exploring the benefits of psychophysical constraints and multiscale processing," Connection Science, 6, 247-280, http://www.cs.colorado.edu/~mozer/papers/music.html.
- Griffith, N., and Todd, P.M., eds. (1999), *Musical Networks: Parallel Distributed Perception and Performance*, Cambridge, MA: The MIT Press, ISBN 0-262-07181-9.

Robotics

- Institute of Robotics and System Dynamics: http://www.robotic.dlr.de/LEARNING/
- UC Berkeley Robotics and Intelligent Machines Lab: http://robotics.eecs.berkeley.edu/
- Perth Robotics and Automation Laboratory: http://telerobot.mech.uwa.edu.au/
- University of New Hampshire Robot Lab: http://www.ece.unh.edu/robots/rbt_home.htm

Weather forecasting and atmospheric science

- UBC Climate Prediction Group: http://www.ocgy.ubc.ca/projects/clim.pred/index.html
- Artificial Intelligence Research In Environmental Science: http://www.salinas.net/~jpeak/airies/airies.html
- MET-AI, an mailing list for meteorologists and AI researchers: http://www.comp.vuw.ac.nz/Research/met-ai
- Caren Marzban, Ph.D., Research Scientist, National Severe Storms Laboratory: http://www.nhn.ou.edu/~marzban/
- David Myers's references on NNs in atmospheric science: http://terra.msrc.sunysb.edu/~dmyers/ai_refs

Weird

Zaknich, Anthony and Baker, Sue K. (1998), "A real-time system for the characterisation of sheep feeding phases from acoustic signals of jaw sounds," Australian Journal of Intelligent Information Processing Systems (AJIIPS), Vol. 5, No. 2, Winter 1998.

Abstract
This paper describes a four-channel real-time system for the detection and measurement of sheep rumination and mastication time periods by the analysis of jaw sounds transmitted through the skull. The system is implemented using an 80486 personal computer, a proprietary data acquisition card (PC-126) and a custom made variable gain preamplifier and bandpass filter module. Chewing sounds are transduced and transmitted to the system using radio microphones attached to the top of the sheep heads. The system's main functions are to detect and estimate rumination and mastication time periods, to estimate the number of chews during the rumination and mastication periods, and to provide estimates of the number of boli in the rumination sequences and the number of chews per bolus. The individual chews are identified using a special energy threshold detector. The rumination and mastication time periods are determined by neural network classifier using a combination of time and frequency domain features extracted from successive 10 second acoustic signal blocks.

- -

Subject: What to do with missing/incomplete data?

The problem of missing data is very complex.

For unsupervised learning, conventional statistical methods for missing data are often appropriate (Little and Rubin, 1987; Schafer, 1997). There is a concise introduction to these methods in the University of Texas statistics FAQ at http://www.utexas.edu/cc/faqs/stat/general/gen25.html.

For supervised learning, the considerations are somewhat different, as discussed by Sarle (1998). The statistical literature on missing data deals almost exclusively with training rather than prediction (e.g., Little, 1992). For example, if you have only a small proportion of cases with missing data, you can simply throw those cases out for purposes of training; if you want to make predictions for cases with missing inputs, you don't have the option of throwing those cases out! In theory, Bayesian methods take care of everything, but a full Bayesian analysis is practical only with special models (such as multivariate normal distributions) or small sample sizes. The neural net literature contains a few good papers that cover prediction with missing inputs (e.g., Ghahramani and Jordan, 1997; Tresp, Neuneier, and Ahmad 1995), but much research remains to be done.

References:

Donner, A. (1982), "The relative effectiveness of procedures commonly used in multiple regression analysis for dealing with missing values," American Statistician, 36, 378-381.

Ghahramani, Z. and Jordan, M.I. (1994), "Supervised learning from incomplete data via an EM approach," in Cowan, J.D., Tesauro, G., and Alspector, J. (eds.) *Advances in Neural Information Processing Systems 6*, San Mateo, CA: Morgan Kaufman, pp. 120-127.

Ghahramani, Z. and Jordan, M.I. (1997), "Mixture models for Learning from incomplete data," in Greiner, R., Petsche, T., and Hanson, S.J. (eds.) *Computational Learning Theory and Natural Learning Systems, Volume IV: Making Learning Systems Practical*, Cambridge, MA: The MIT Press, pp. 67-85.

Jones, M.P. (1996), "Indicator and stratification methods for missing explanatory variables in multiple linear regression," J. of the American Statistical Association, 91, 222-230.

Little, R.J.A. (1992), "Regression with missing X's: A review," J. of the American Statistical Association, 87, 1227-1237.

Little, R.J.A. and Rubin, D.B. (1987), *Statistical Analysis with Missing Data*, NY: Wiley.

McLachlan, G.J. (1992) *Discriminant Analysis and Statistical Pattern Recognition*, Wiley.

Sarle, W.S. (1998), "Prediction with Missing Inputs," in Wang, P.P. (ed.), JCIS '98 Proceedings, Vol II, Research Triangle Park, NC, 399-402, ftp://ftp.sas.com/pub/neural/JCIS98.ps.

Schafer, J.L. (1997), *Analysis of Incomplete Multivariate Data*, London: Chapman & Hall, ISBN 0 412 04061 1.

Tresp, V., Ahmad, S. and Neuneier, R., (1994), "Training neural networks with deficient data", in Cowan, J.D., Tesauro, G., and Alspector, J. (eds.) *Advances in Neural Information Processing Systems 6*,

San Mateo, CA: Morgan Kaufman, pp. 128-135.

Tresp, V., Neuneier, R., and Ahmad, S. (1995), "Efficient methods for dealing with missing data in supervised learning", in Tesauro, G., Touretzky, D.S., and Leen, T.K. (eds.) *Advances in Neural Information Processing Systems 7*, Cambridge, MA: The MIT Press, pp. 689-696.

Subject: How to forecast time series (temporal sequences)?

In most of this FAQ, it is assumed that the training cases are statistically independent. That is, the training cases consist of pairs of input and target vectors, `(X_i,Y_i), i=1,...,N,` such that the conditional distribution of `Y_i` given all the other training data, (`X_j, j=1,...,N,` and `Y_j, j=1,...i-1,i+1,...N`) is equal to the conditional distribution of `Y_i` given `X_i` regardless of the values in the other training cases. Independence of cases is often achieved by random sampling.

The most common violation of the independence assumption occurs when cases are observed in a certain order relating to time or space. That is, case `(X_i,Y_i)` corresponds to time `T_i,` with `T_1 < T_2 < ... < T_N.` It is assumed that the current target `Y_i` may depend not only on `X_i` but also on `(X_i,Y_i)` in the recent past. If the `T_i` are equally spaced, the simplest way to deal with this dependence is to include additional inputs (called lagged variables, shift registers, or a tapped delay line) in the network. Thus, for target `Y_i,` the inputs may include `X_i, Y_{i-1}, X_{i-1}, Y_{i-1}, X_{i-2},` etc. (In some situations, `X_i` would not be known at the time you are trying to forecast `Y_i` and would therefore be excluded from the inputs.) Then you can train an ordinary feedforward network with these targets and lagged variables. The use of lagged variables has been extensively studied in the statistical and econometric literature (Judge, Griffiths, Hill, Lütkepohl and Lee, 1985). A network in which the only inputs are lagged target values is called an "autoregressive model." The input space that includes all of the lagged variables is called the "embedding space."

If the `T_i` are *not* equally spaced, everything gets much more complicated. One approach is to use a smoothing technique to interpolate points at equally spaced intervals, and then use the interpolated values for training instead of the original data.

Use of lagged variables increases the number of decisions that must be made during training, since you must consider which lags to include in the network, as well as which input variables, how many hidden units, etc. Neural network researchers have therefore attempted to use partially recurrent networks instead of feedforward networks with lags (Weigend and Gershenfeld, 1994). Recurrent networks store information about past values in the network itself. There are many different kinds of recurrent architectures (Hertz, Krogh, and Palmer 1991; Mozer, 1994; Horne and Giles, 1995; Kremer, 199?). For example, in time-delay neural networks (Lang, Waibel, and Hinton 1990), the outputs for predicting target `Y_{i-1}` are used as inputs when processing target `Y_i.` Jordan networks (Jordan, 1986) are similar to time-delay neural networks except that the feedback is an exponential smooth of the sequence of output values. In Elman networks (Elman, 1990), the hidden unit activations that occur when processing target `Y_{i-1}` are used as inputs when processing target `Y_i.`

However, there are some problems that cannot be dealt with via recurrent networks alone. For example, many time series exhibit trend, meaning that the target values tend to go up over time, or that the target values tend to go down over time. For example, stock prices and many other financial variables usually go up. If today's price

is higher than all previous prices, and you try to forecast tomorrow's price using today's price as a lagged input, you are extrapolating, and extrapolating is unreliable. The simplest methods for handling trend are:

- First fit a linear regression predicting the target values from the time, `Y_i = a + b T_i + noise`, where `a` and `b` are regression weights. Compute residuals `R_i = Y_i - (a + b T_i)`. Then train the network using `R_i` for the target and lagged values. This method is rather crude but may work for deterministic linear trends. Of course, for nonlinear trends, you would need to fit a nonlinear regression.

- Instead of using `Y_i` as a target, use `D_i = Y_i - Y_{i-1}` for the target and lagged values. This is called differencing and is the standard statistical method for handling nondeterministic (stochastic) trends. Sometimes it is necessary to compute differences of differences.

For an elementary discussion of trend and various other practical problems in forecasting time series with NNs, such as seasonality, see Masters (1993). For a more advanced discussion of NN forecasting of economic series, see Moody (1998).

There are several different ways to compute forecasts. For simplicity, let's assume you have a simple time series, `Y_1, ..., Y_99`, you want to forecast future values `Y_f` for `f > 99`, and you decide to use three lagged values as inputs. The possibilities include:

Single-step, one-step-ahead, or open-loop forecasting:

Train a network with target `Y_i` and inputs `Y_{i-1}, Y_{i-2},` and `Y_{i-3}.` Let the scalar function computed by the network be designated as `Net(.,.,.)` taking the three input values as arguments and returning the output (predicted) value. Then:
forecast `Y_100` as `Net(Y_99,Y_98,Y_97)`
forecast `Y_101` as `Net(Y_100,Y_99,Y_98)`
forecast `Y_102` as `Net(Y_101,Y_100,Y_99)`
forecast `Y_103` as `Net(Y_102,Y_101,Y_100)`
forecast `Y_104` as `Net(Y_103,Y_102,Y_101)`
and so on.

Multi-step or closed-loop forecasting:

Train the network as above, but:
forecast `Y_100` as `P_100 = Net(Y_99,Y_98,Y_97)`
forecast `Y_101` as `P_101 = Net(P_100,Y_99,Y_98)`
forecast `Y_102` as `P_102 = Net(P_101,P_100,Y_99)`
forecast `Y_103` as `P_103 = Net(P_102,P_101,P_100)`
forecast `Y_104` as `P_104 = Net(P_103,P_102,P_101)`
and so on.

`N`-step-ahead forecasting:

For, say, `N=3`, train the network as above, but:
compute `P_100 = Net(Y_99,Y_98,Y_97)`
compute `P_101 = Net(P_100,Y_99,Y_98)`
forecast `Y_102` as `P_102 = Net(P_101,P_100,Y_99)`
forecast `Y_103` as `P_103 = Net(P_102,P_101,Y_100)`
forecast `Y_104` as `P_104 = Net(P_103,P_102,Y_101)`
and so on.

Direct simultaneous long-term forecasting:

Train a network with multiple targets `Y_i, Y_{i+1},` and `Y_{i+2}` and inputs `Y_{i-1}, Y_{i-2},` and `Y_{i-3}.` Let the vector function computed by the network be designated as `Net3(.,.,.),` taking the three input values as arguments and returning the output (predicted) vector. Then:
forecast `(Y_100,Y_101,Y_102)` as `Net3(Y_99,Y_98,Y_97)`

Which method you choose for computing forecasts will obviously depend in part on the requirements of your application. If you have yearly sales figures through 1999 and you need to forecast sales in 2003, you clearly can't use single-step forecasting. If you need to compute forecasts at a thousand different future times, using direct simultaneous long-term forecasting would require an extremely large network.

If a time series is a random walk, a well-trained network will predict `Y_i` by simply outputting `Y_{i-1}.` If you make a plot showing both the target values and the outputs, the two curves will almost coincide, except for being offset by one time step. People often mistakenly intrepret such a plot to indicate good forecasting accuracy, whereas in fact the network is virtually useless. In such situations, it is more enlightening to plot multi-step forecasts or `N`-step-ahead forecasts.

For general information on time-series forecasting, see the following URLs:

- Forecasting FAQs: http://forecasting.cwru.edu/faqs.html
- Forecasting Principles: http://hops.wharton.upenn.edu/forecast/
- Investment forecasts for stocks and mutual funds: http://www.coe.uncc.edu/~hphillip/

References:

Elman, J.L. (1990), "Finding structure in time," Cognitive Science, 14, 179-211.

Hertz, J., Krogh, A., and Palmer, R. (1991). *Introduction to the Theory of Neural Computation.* Addison-Wesley: Redwood City, California.

Horne, B. G. and Giles, C. L. (1995), "An experimental comparison of recurrent neural networks," In Tesauro, G., Touretzky, D., and Leen, T., editors, Advances in Neural Information Processing Systems 7, pp. 697-704. The MIT Press.

Jordan, M. I. (1986), "Attractor dynamics and parallelism in a connectionist sequential machine," In Proceedings of the Eighth Annual conference of the Cognitive Science Society, pages 531-546. Lawrence Erlbaum.

Judge, G.G., Griffiths, W.E., Hill, R.C., Lütkepohl, H., and Lee, T.-C. (1985), *The Theory and Practice of Econometrics*, NY: John Wiley & Sons.

Kremer, S.C. (199?), "Spatio-temporal Connectionist Networks: A Taxonomy and Review," http://hebb.cis.uoguelph.ca/~skremer/Teaching/27642/dynamic2/review.html.

Lang, K. J., Waibel, A. H., and Hinton, G. (1990), "A time-delay neural network architecture for isolated word recognition," Neural Networks, 3, 23-44.

Masters, T. (1993). *Practical Neural Network Recipes in C++*, San Diego: Academic Press.

Moody, J. (1998), "Forecasting the economy with neural nets: A survey of challenges and solutions," in Orr, G,B., and Mueller, K-R, eds., *Neural Networks: Tricks of the Trade*, Berlin: Springer.

Mozer, M.C. (1994), "Neural net architectures for temporal sequence processing," in Weigend, A.S. and Gershenfeld, N.A., eds. (1994) *Time Series Prediction: Forecasting the Future and Understanding the Past*, Reading, MA: Addison-Wesley, 243-264, http://www.cs.colorado.edu/~mozer/papers/timeseries.html.

Weigend, A.S. and Gershenfeld, N.A., eds. (1994) *Time Series Prediction: Forecasting the Future and Understanding the Past*, Reading, MA: Addison-Wesley.

Subject: How to learn an inverse of a function?

Ordinarily, NNs learn a function `Y = f(X)`, where `Y` is a vector of outputs, `X` is a vector of inputs, and `f()` is the function to be learned. Sometimes, however, you may want to learn an inverse of a function `f(),` that is, given `Y,` you want to be able to find an `X` such that `Y = f(X).` In general, there may be many different `Xs` that satisfy the equation `Y = f(X).`

For example, in robotics (DeMers and Kreutz-Delgado, 1996, 1997), `X` might describe the positions of the joints in a robot's arm, while `Y` would describe the location of the robot's hand. There are simple formulas to compute the location of the hand given the positions of the joints, called the "forward kinematics" problem. But there is no simple formula for the "inverse kinematics" problem to compute positions of the joints that yield a given location for the hand. Furthermore, if the arm has several joints, there will usually be many different positions of the joints that yield the same location of the hand, so the forward kinematics function is many-to-one and has no unique inverse. Picking any `X` such that `Y = f(X)` is OK if the only aim is to position the hand at `Y.` However if the aim is to generate a series of points to move the hand through an arc this may be insufficient. In this case the series of `Xs` need to be in the same "branch" of the function space. Care must be taken to avoid solutions that yield inefficient or impossible movements of the arm.

As another example, consider an industrial process in which `X` represents settings of control variables imposed by an operator, and `Y` represents measurements of the product of the industrial process. The function `Y = f(X)` can be learned by a NN using conventional training methods. But the goal of the analysis may be to find control settings `X` that yield a product with specified measurements `Y,` in which case an inverse of `f(X)` is required. In industrial applications, financial considerations are important, so not just any setting `X` that yields the desired result `Y` may be acceptable. Perhaps a function can be specified that gives the cost of `X` resulting from energy consumption, raw materials, etc., in which case you would want to find the `X` that minimizes the cost function while satisfying the equation `Y = f(X).`

The obvious way to try to learn an inverse function is to generate a set of training data from a given forward function, but designate `Y` as the input and `X` as the output when training the network. Using a least-squares error function, this approach will fail if `f()` is many-to-one. The problem is that for an input `Y,` the net will not learn any single `X` such that `Y = f(X),` but will instead learn the arithmetic mean of all the `Xs` in the training set that satisfy the equation (Bishop, 1995, pp. 207-208). One solution to this difficulty is to construct a network that learns a mixture approximation to the conditional distribution of `X` given `Y` (Bishop, 1995, pp. 212-221).

However, the mixture method will not work well in general for an X vector that is more than one-dimensional, such as `Y = X_1^2 + X_2^2,` since the number of mixture components required may increase exponentially with the dimensionality of `X.` And you are still left with the problem of extracting a single output vector from the mixture distribution, which is nontrivial if the mixture components overlap considerably. Another solution is to use a highly robust error function, such as a redescending M-estimator, that learns a single mode of the conditional distribution instead of learning the mean (Huber, 1981; Rohwer and van der Rest 1996). Additional regularization terms or constraints may be required to persuade the network to choose appropriately among several modes, and there may be severe problems with local optima.

Another approach is to train a network to learn the forward mapping `f()` and then numerically invert the function. Finding X such that `Y = f(X)` is simply a matter of solving a nonlinear system of equations, for which many algorithms can be found in the numerical analysis literature (Dennis and Schnabel 1983). One way to solve nonlinear equations is turn the problem into an optimization problem by minimizing `sum(Y_i-f(X_i))^2.` This method fits in nicely with the usual gradient-descent methods for training NNs (Kindermann and Linden 1990). Since the nonlinear equations will generally have multiple solutions, there may be severe problems with local optima, especially if some solutions are considered more desirable than others. You can deal with multiple solutions by inventing some objective function that measures the goodness of different solutions, and optimizing this objective function under the nonlinear constraint `Y = f(X)` using any of numerous algorithms for nonlinear programming (NLP; see Bertsekas, 1995, and other references under "What are conjugate gradients, Levenberg-Marquardt, etc.?") The power and flexibility of the nonlinear programming approach are offset by possibly high computational demands.

If the forward mapping `f()` is obtained by training a network, there will generally be some error in the network's outputs. The magnitude of this error can be difficult to estimate. The process of inverting a network can propagate this error, so the results should be checked carefully for validity and numerical stability. Some training methods can produce not just a point output but also a prediction interval (Bishop, 1995; White, 1992). You can take advantage of prediction intervals when inverting a network by using NLP methods. For example, you could try to find an X that minimizes the width of the prediction interval under the constraint that the equation `Y = f(X)` is satisfied. Or instead of requiring `Y = f(X)` be satisfied exactly, you could try to find an X such that the prediction interval is contained within some specified interval while minimizing some cost function.

For more mathematics concerning the inverse-function problem, as well as some interesting methods involving self-organizing maps, see DeMers and Kreutz-Delgado (1996, 1997). For NNs that are relatively easy to invert, see the Adaptive Logic Networks described in the software sections of the FAQ.

References:

Bertsekas, D. P. (1995), *Nonlinear Programming*, Belmont, MA: Athena Scientific.

Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press.

DeMers, D., and Kreutz-Delgado, K. (1996), "Canonical Parameterization of Excess motor degrees of freedom with self organizing maps", IEEE Trans Neural Networks, 7, 43-55.

DeMers, D., and Kreutz-Delgado, K. (1997), "Inverse kinematics of dextrous manipulators," in Omidvar, O., and van der Smagt, P., (eds.) *Neural Systems for Robotics*, San Diego: Academic Press, pp. 75-116.

Dennis, J.E. and Schnabel, R.B. (1983) *Numerical Methods for Unconstrained Optimization and*

Nonlinear Equations, Prentice-Hall

Huber, P.J. (1981), *Robust Statistics*, NY: Wiley.

Kindermann, J., and Linden, A. (1990), "Inversion of Neural Networks by Gradient Descent," Parallel Computing, 14, 277-286, ftp://icsi.Berkeley.EDU/pub/ai/linden/KindermannLinden.IEEE92.ps.Z

Rohwer, R., and van der Rest, J.C. (1996), "Minimum description length, regularization, and multimodal data," Neural Computation, 8, 595-609.

White, H. (1992), "Nonparametric Estimation of Conditional Quantiles Using Neural Networks," in Page, C. and Le Page, R. (eds.), *Proceedings of the 23rd Sympsium on the Interface: Computing Science and Statistics*, Alexandria, VA: American Statistical Association, pp. 190-199.

Subject: How to get invariant recognition of images under translation, rotation, etc.?

See:

Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press, section 8.7.

Masters, T. (1994), *Signal and Image Processing with Neural Networks: A C++ Sourcebook*, NY: Wiley.

Soucek, B., and The IRIS Group (1992), *Fast Learning and Invariant Object Recognition*, NY: Wiley.

Squire, D. (1997), *Model-Based Neural Networks for Invariant Pattern Recognition*, http://cuiwww.unige.ch/~squire/publications.html

Laurenz Wiskott, bibliography on "Unsupervised Learning of Invariances in Neural Systems" http://www.cnl.salk.edu/~wiskott/Bibliographies/LearningInvariances.html

---

Subject: How to recognize handwritten characters?

URLS:

- Don Tveter's *The Pattern Recognition Basis of AI* at http://www.dontveter.com/basisofai/char.html
- Andras Kornai's homepage at http://www.cs.rice.edu/~andras/
- Yann LeCun's homepage at http://www.research.att.com/~yann/
  Data sets of handwritten digits can be found at http://www.research.att.com/~yann/exdb/mnist/

Other references:

Hastie, T., and Simard, P.Y. (1998), "Metrics and models for handwritten character recognition," Statistical Science, 13, 54-65.

Jackel, L.D. et al., (1994) "Comparison of Classifier Methods: A Case Study in Handwritten Digit Recognition", 1994 International Conference on Pattern Recognition, Jerusalem

LeCun, Y., Jackel, L.D., Bottou, L., Brunot, A., Cortes, C., Denker, J.S., Drucker, H., Guyon, I., Muller, U.A., Sackinger, E., Simard, P., and Vapnik, V. (1995), "Comparison of learning algorithms for handwritten digit recognition," in F. Fogelman and P. Gallinari, eds., International Conference on Artificial Neural Networks, pages 53-60, Paris.

Orr, G.B., and Mueller, K.-R., eds. (1998), *Neural Networks: Tricks of the Trade*, Berlin: Springer, ISBN 3-540-65311-2.

Subject: What about pulsed or spiking NNs?

The standard reference is:

Maass, W., and Bishop, C.M., eds. (1999) *Pulsed Neural Networks*, Cambridge, MA: The MIT Press, ISBN: 0262133504.

For more information on this book, see the section on "Pulsed/Spiking networks" under "Other notable books" in part 4 of the FAQ. Also see Professor Maass's web page at http://www.igi.tugraz.at/maass/.

Some other interesting URLs include:

- Laboratory of Computational Neuroscience (LCN) at the Swiss Federal Institute of Technology Lausanne, http://diwww.epfl.ch/mantra/mantra_bioneuro.html
- The notoriously hyped Berger-Liaw Neural Network Speaker-Independent Speech Recognition System, http://www.usc.edu/ext-relations/news_service/releases/stories/36013.html

---

Subject: What about Genetic Algorithms?

There are a number of definitions of GA (Genetic Algorithm). A possible one is

```
A GA is an optimization program
that starts with
a population of encoded procedures,       (Creation of Life :-> )
mutates them stochastically,              (Get cancer or so :-> )
```

```
and uses a selection process                (Darwinism)
to prefer the mutants with high fitness
and perhaps a recombination process         (Make babies :-> )
to combine properties of (preferably) the succesful mutants.
```

Genetic algorithms are just a special case of the more general idea of "evolutionary computation". There is a newsgroup that is dedicated to the field of evolutionary computation called comp.ai.genetic. It has a detailed FAQ posting which, for instance, explains the terms "Genetic Algorithm", "Evolutionary Programming", "Evolution Strategy", "Classifier System", and "Genetic Programming". That FAQ also contains lots of pointers to relevant literature, software, other sources of information, et cetera et cetera. Please see the comp.ai.genetic FAQ for further information.

For an entertaining introduction to evolutionary training of neural nets, see:

> David Fogel (2001), *Blondie24: Playing at the Edge of AI*, Morgan Kaufmann Publishers, ISBN: 1558607838

There are other books and papers by Fogel and his colleagues listed under "Checkers/Draughts" in the "Games, sports, gambling" section above.

For an extensive review, see:

> Yao, X. (1999), "Evolving Artificial Neural Networks," Proceedings of the IEEE, 87, 1423-1447, http://www.cs.bham.ac.uk/~xin/journal_papers.html

Here are some other on-line papers about evolutionary training of NNs:

- Backprop+GA: http://geneura.ugr.es/~pedro/G-Prop.htm
- LVQ+GA: http://geneura.ugr.es/g-lvq/g-lvq.html
- Very long chromosomes: ftp://archive.cis.ohio-state.edu/pub/neuroprose/korning.nnga.ps.Z

More URLs on genetic algorithms and NNs:

- Omri Weisman and Ziv Pollack's web page on "Neural Network Using Genetic Algorithms" at http://www.cs.bgu.ac.il/~omri/NNUGA/
- Christoph M. Friedrich's web page on Evolutionary algorithms and Artificial Neural Networks has a bibloigraphy and links to researchers at http://www.tussy.uni-wh.de/~chris/gann/gann.html
- Andrew Gray's Hybrid Systems FAQ at the University of Otago at http://divcom.otago.ac.nz:800/COM/INFOSCI/SMRL/people/andrew/publications/faq/hybrid/hybrid.htm
- Differential Evolution: http://www.icsi.berkeley.edu/~storn/code.html

For general information on GAs, try the links at http://www.shef.ac.uk/~gaipp/galinks.html and

http://www.cs.unibo.it/~gaioni

Subject: What about Fuzzy Logic?

Fuzzy logic is an area of research based on the work of L.A. Zadeh. It is a departure from classical two-valued sets and logic, that uses "soft" linguistic (e.g. large, hot, tall) system variables and a continuous range of truth values in the interval [0,1], rather than strict binary (True or False) decisions and assignments.

Fuzzy logic is used where a system is difficult to model exactly (but an inexact model is available), is controlled by a human operator or expert, or where ambiguity or vagueness is common. A typical fuzzy system consists of a rule base, membership functions, and an inference procedure.

Most fuzzy logic discussion takes place in the newsgroup comp.ai.fuzzy (where there is a fuzzy logic FAQ) but there is also some work (and discussion) about combining fuzzy logic with neural network approaches in comp.ai.neural-nets.

Early work combining neural nets and fuzzy methods used competitive networks to generate rules for fuzzy systems (Kosko 1992). This approach is sort of a crude version of bidirectional counterpropagation (Hecht-Nielsen 1990) and suffers from the same deficiencies. More recent work (Brown and Harris 1994; Kosko 1997) has been based on the realization that a fuzzy system is a nonlinear mapping from an input space to an output space that can be parameterized in various ways and therefore can be adapted to data using the usual neural training methods (see "What is backprop?") or conventional numerical optimization algorithms (see "What are conjugate gradients, Levenberg-Marquardt, etc.?").

A neural net can incorporate fuzziness in various ways:

- The inputs can be fuzzy. Any garden-variety backprop net is fuzzy in this sense, and it seems rather silly to call a net "fuzzy" solely on this basis, although Fuzzy ART (Carpenter and Grossberg 1996) has no other fuzzy characteristics.
- The outputs can be fuzzy. Again, any garden-variety backprop net is fuzzy in this sense. But competitive learning nets ordinarily produce crisp outputs, so for competitive learning methods, having fuzzy output is a meaningful distinction. For example, fuzzy c-means clustering (Bezdek 1981) is meaningfully different from (crisp) k-means. Fuzzy ART does *not* have fuzzy outputs.
- The net can be interpretable as an adaptive fuzzy system. For example, Gaussian RBF nets and B-spline regression models (Dierckx 1995, van Rijckevorsal 1988) are fuzzy systems with adaptive weights (Brown and Harris 1994) and can legitimately be called neurofuzzy systems.
- The net can be a conventional NN architecture that operates on fuzzy numbers instead of real numbers (Lippe, Feuring and Mischke 1995).
- Fuzzy constraints can provide external knowledge (Lampinen and Selonen 1996).

More information on neurofuzzy systems is available online:

- The Fuzzy Logic and Neurofuzzy Resources page of the Image, Speech and Intelligent Systems (ISIS) research group at the University of Southampton, Southampton, Hampshire, UK: http://www-isis.ecs.soton.ac.uk/research/nfinfo/fuzzy.html.
- The Neuro-Fuzzy Systems Research Group's web page at Tampere University of Technology, Tampere,

Finland: http://www.cs.tut.fi/~tpo/group.html and http://dmiwww.cs.tut.fi/nfs/Welcome_uk.html

- Marcello Chiaberge's Neuro-Fuzzy page at http://polimage.polito.it/~marcello.
- The homepage of the research group on Neural Networks and Fuzzy Systems at the Institute of Knowledge Processing and Language Engineering, Faculty of Computer Science, University of Magdeburg, Germany, at http://www.neuro-fuzzy.de/
- Jyh-Shing Roger Jang's home page at http://www.cs.nthu.edu.tw/~jang/ with information on ANFIS (Adaptive Neuro-Fuzzy Inference Systems), articles on neuro-fuzzy systems, and more links.
- Andrew Gray's Hybrid Systems FAQ at the University of Otago at http://divcom.otago.ac.nz:800/COM/INFOSCI/SMRL/people/andrew/publications/faq/hybrid/hybrid.htm

References:

Bezdek, J.C. (1981), *Pattern Recognition with Fuzzy Objective Function Algorithms*, New York: Plenum Press.

Bezdek, J.C. & Pal, S.K., eds. (1992), *Fuzzy Models for Pattern Recognition*, New York: IEEE Press.

Brown, M., and Harris, C. (1994), *Neurofuzzy Adaptive Modelling and Control*, NY: Prentice Hall.

Carpenter, G.A. and Grossberg, S. (1996), "Learning, Categorization, Rule Formation, and Prediction by Fuzzy Neural Networks," in Chen, C.H. (1996), pp. 1.3-1.45.

Chen, C.H., ed. (1996) *Fuzzy Logic and Neural Network Handbook*, NY: McGraw-Hill, ISBN 0-07-011189-8.

Dierckx, P. (1995), *Curve and Surface Fitting with Splines*, Oxford: Clarendon Press.

Hecht-Nielsen, R. (1990), *Neurocomputing*, Reading, MA: Addison-Wesley.

Klir, G.J. and Folger, T.A.(1988), *Fuzzy Sets, Uncertainty, and Information*, Englewood Cliffs, N.J.: Prentice-Hall.

Kosko, B.(1992), *Neural Networks and Fuzzy Systems*, Englewood Cliffs, N.J.: Prentice-Hall.

Kosko, B. (1997), *Fuzzy Engineering*, NY: Prentice Hall.

Lampinen, J and Selonen, A. (1996), "Using Background Knowledge for Regularization of Multilayer Perceptron Learning", Submitted to International Conference on Artificial Neural Networks, ICANN'96, Bochum, Germany.

Lippe, W.-M., Feuring, Th. and Mischke, L. (1995), "Supervised learning in fuzzy neural networks," Institutsbericht Angewandte Mathematik und Informatik, WWU Muenster, I-12, http://wwwmath.uni-muenster.de/~feuring/WWW_literatur/bericht12_95.ps.gz

Nauck, D., Klawonn, F., and Kruse, R. (1997), *Foundations of Neuro-Fuzzy Systems*, Chichester: Wiley, ISBN 0-471-97151-0.

van Rijckevorsal, J.L.A. (1988), "Fuzzy coding and B-splines," in van Rijckevorsal, J.L.A., and de

Leeuw, J., eds., Component and Correspondence Analysis, Chichester: John Wiley & Sons, pp. 33-54.

Subject: Unanswered FAQs

- How many training cases do I need?
- How should I split the data into training and validation sets?
- What error functions can be used?
- How can I select important input variables?
- Should NNs be used in safety-critical applications?

Subject: Other NN links?

Search engines

- Yahoo: http://www.yahoo.com/Science/Engineering/Electrical_Engineering/Neural_Networks/
- Neuroscience Web Search: http://www.acsiom.org/nsr/neuro.html

Archives of NN articles and software

Neuroprose ftp archive site

ftp://archive.cis.ohio-state.edu/pub/neuroprose This directory contains technical reports as a public service to the connectionist and neural network scientific community.

Finnish University Network archive site

A large collection of neural network papers and software at ftp://ftp.funet.fi/pub/sci/neural/ Contains all the public domain software and papers that they have been able to find. All of these files have been transferred from FTP sites in U.S. and are mirrored about every 3 months at fastest. Contact: neural-adm@ftp.funet.fi

SEL-HPC Article Archive

http://liinwww.ira.uka.de/bibliography/Misc/SEL-HPC.html

Machine Learning Papers

http://gubbio.cs.berkeley.edu/mlpapers/

Plain-text Tables of Contents of NN journals

Pattern Recognition Group, Department of Applied Physics,
Faculty of Applied Sciences, Delft University of Technology,
http://www.ph.tn.tudelft.nl/PRInfo/PRInfo/journals.html

- **The Collection of Computer Science Bibliographies: Bibliographies on Neural Networks**

  http://liinwww.ira.uka.de/bibliography/Neural/index.html

- **BibTeX data bases of NN journals**

  The Center for Computational Intelligence maintains BibTeX data bases of various NN journals, including IEEE Transactions on Neural Networks, Machine Learning, Neural Computation, and NIPS, at http://www.ci.tuwien.ac.at/docs/ci/bibtex_collection.html or ftp://ftp.ci.tuwien.ac.at/pub/texmf/bibtex/bib/.

- **NN events server**

  There is a WWW page for Announcements of Conferences, Workshops and Other Events on Neural Networks at IDIAP in Switzerland. WWW-Server: http://www.idiap.ch/html/idiap-networks.html.

- **Academic programs list**

  Rutvik Desai <rutvik@c3serve.c3.lanl.gov> has a compilation of acedemic programs offering interdeciplinary studies in computational neuroscience, AI, cognitive psychology etc. at http://www.cs.indiana.edu/hyplan/rudesai/cogsci-prog.html

  Links to neurosci, psychology, linguistics lists are also provided.

- **Neurosciences Internet Resource Guide**

  This document aims to be a guide to existing, free, Internet-accessible resources helpful to neuroscientists of all stripes. An ASCII text version (86K) is available in the Clearinghouse of Subject-Oriented Internet Resource Guides as follows:

  ftp://una.hh.lib.umich.edu/inetdirsstacks/neurosci:cormbonario,
  gopher://una.hh.lib.umich.edu/00/inetdirsstacks/neurosci:cormbonario,
  http://http2.sils.umich.edu/Public/nirg/nirg1.html.

- **Other WWW sites**

  In World-Wide-Web (WWW, for example via the xmosaic program) you can read neural network information for instance by opening one of the following uniform resource locators (URLs):
  http://www-xdiv.lanl.gov/XCM/neural/neural_announcements.html Los Alamos neural announcements and general information,

http://www.ph.kcl.ac.uk/neuronet/ (NEuroNet, King's College, London),
http://www.eeb.ele.tue.nl (Eindhoven, Netherlands),
http://www.emsl.pnl.gov:2080/docs/cie/neural/ (Pacific Northwest National Laboratory, Richland, Washington, USA),
http://www.cosy.sbg.ac.at/~rschwaig/rschwaig/projects.html (Salzburg, Austria),
http://http2.sils.umich.edu/Public/nirg/nirg1.html (Michigan, USA),
http://www.lpac.ac.uk/SEL-HPC/Articles/NeuralArchive.html (London),
http://rtm.science.unitn.it/ Reactive Memory Search (Tabu Search) page (Trento, Italy),
http://www.wi.leidenuniv.nl/art/ (ART WWW site, Leiden, Netherlands),
http://nucleus.hut.fi/nnrc/ Helsinki University of Technology.
http://www.pitt.edu/~mattf/NeuroRing.html links to neuroscience web pages
http://www.arcade.uiowa.edu/hardin-www/md-neuro.htmlHardin Meta Directory web page for Neurology/Neurosciences.
Many others are available too; WWW is changing all the time.

That's all folks (End of the Neural Network FAQ).

Acknowledgements: Thanks to all the people who helped to get the stuff above into the posting. I cannot name them all, because I would make far too many errors then. :->

No? Not good? You want individual credit?
OK, OK. I'll try to name them all. But: no guarantee....

THANKS FOR HELP TO:
(in alphabetical order of email adresses, I hope)

- Steve Ward <71561.2370@CompuServe.COM>
- Allen Bonde <ab04@harvey.gte.com>
- Accel Infotech Spore Pte Ltd <accel@solomon.technet.sg>
- Ales Krajnc <akrajnc@fagg.uni-lj.si>
- Alexander Linden <al@jargon.gmd.de>
- Matthew David Aldous <aldous@mundil.cs.mu.OZ.AU>
- S.Taimi Ames <ames@reed.edu>
- Axel Mulder <amulder@move.kines.sfu.ca>
- anderson@atc.boeing.com
- Andy Gillanders <andy@grace.demon.co.uk>
- Davide Anguita <anguita@ICSI.Berkeley.EDU>
- Avraam Pouliakis <apou@leon.nrcps.ariadne-t.gr>
- Kim L. Blackwell <avrama@helix.nih.gov>
- Mohammad Bahrami <bahrami@cse.unsw.edu.au>
- Paul Bakker <bakker@cs.uq.oz.au>
- Stefan Bergdoll <bergdoll@zxd.basf-ag.de>
- Jamshed Bharucha <bharucha@casbs.Stanford.EDU>
- Carl M. Cook <biocomp@biocomp.seanet.com>
- Yijun Cai <caiy@mercury.cs.uregina.ca>
- L. Leon Campbell <campbell@brahms.udel.edu>

- Cindy Hitchcock <cindyh@vnet.ibm.com>
- Clare G. Gallagher <clare@mikuni2.mikuni.com>
- Craig Watson <craig@magi.ncsl.nist.gov>
- Yaron Danon <danony@goya.its.rpi.edu>
- David Ewing <dave@ndx.com>
- David DeMers <demers@cs.ucsd.edu>
- Denni Rognvaldsson <denni@thep.lu.se>
- Duane Highley <dhighley@ozarks.sgcl.lib.mo.us>
- Dick.Keene@Central.Sun.COM
- DJ Meyer <djm@partek.com>
- Donald Tveter <don@dontveter.com>
- Daniel Tauritz <dtauritz@wi.leidenuniv.nl>
- Wlodzislaw Duch <duch@phys.uni.torun.pl>
- E. Robert Tisdale <edwin@flamingo.cs.ucla.edu>
- Athanasios Episcopos <episcopo@fire.camp.clarkson.edu>
- Frank Schnorrenberg <fs0997@easttexas.tamu.edu>
- Gary Lawrence Murphy <garym@maya.isis.org>
- gaudiano@park.bu.edu
- Lee Giles <giles@research.nj.nec.com>
- Glen Clark <opto!glen@gatech.edu>
- Phil Goodman <goodman@unr.edu>
- guy@minster.york.ac.uk
- Horace A. Vallas, Jr. <hav@neosoft.com>
- Joerg Heitkoetter <heitkoet@lusty.informatik.uni-dortmund.de>
- Ralf Hohenstein <hohenst@math.uni-muenster.de>
- Ian Cresswell <icressw@leopold.win-uk.net>
- Gamze Erten <ictech@mcimail.com>
- Ed Rosenfeld <IER@aol.com>
- Franco Insana <INSANA@asri.edu>
- Janne Sinkkonen <janne@iki.fi>
- Javier Blasco-Alberto <jblasco@ideafix.cps.unizar.es>
- Jean-Denis Muller <jdmuller@vnet.ibm.com>
- Jeff Harpster <uu0979!jeff@uu9.psi.com>
- Jonathan Kamens <jik@MIT.Edu>
- J.J. Merelo <jmerelo@geneura.ugr.es>
- Dr. Jacek Zurada <jmzura02@starbase.spd.louisville.edu>
- Jon Gunnar Solheim <jon@kongle.idt.unit.no>
- Josef Nelissen <jonas@beor.informatik.rwth-aachen.de>
- Joey Rogers <jrogers@buster.eng.ua.edu>
- Subhash Kak <kak@gate.ee.lsu.edu>
- Ken Karnofsky <karnofsky@mathworks.com>
- Kjetil.Noervaag@idt.unit.no
- Luke Koops <koops@gaul.csd.uwo.ca>
- Kurt Hornik <Kurt.Hornik@tuwien.ac.at>
- Thomas Lindblad <lindblad@kth.se>
- Clark Lindsey <lindsey@particle.kth.se>
- Lloyd Lubet <llubet@rt66.com>
- William Mackeown <mackeown@compsci.bristol.ac.uk>
- Maria Dolores Soriano Lopez <maria@vaire.imib.rwth-aachen.de>
- Mark Plumbley <mark@dcs.kcl.ac.uk>
- Peter Marvit <marvit@cattell.psych.upenn.edu>

- masud@worldbank.org
- Miguel A. Carreira-Perpinan<mcarreir@moises.ls.fi.upm.es>
- Yoshiro Miyata <miyata@sccs.chukyo-u.ac.jp>
- Madhav Moganti <mmogati@cs.umr.edu>
- Jyrki Alakuijala <more@ee.oulu.fi>
- Jean-Denis Muller <muller@bruyeres.cea.fr>
- Michael Reiss <m.reiss@kcl.ac.uk>
- mrs@kithrup.com
- Maciek Sitnik <msitnik@plearn.edu.pl>
- R. Steven Rainwater <ncc@ncc.jvnc.net>
- Nigel Dodd <nd@neural.win-uk.net>
- Barry Dunmall <neural@nts.sonnet.co.uk>
- Paolo Ienne <Paolo.Ienne@di.epfl.ch>
- Paul Keller <pe_keller@ccmail.pnl.gov>
- Peter Hamer <P.G.Hamer@nortel.co.uk>
- Pierre v.d. Laar <pierre@mbfys.kun.nl>
- Michael Plonski <plonski@aero.org>
- Lutz Prechelt <prechelt@ira.uka.de> [creator of FAQ]
- Richard Andrew Miles Outerbridge <ramo@uvphys.phys.uvic.ca>
- Rand Dixon <rdixon@passport.ca>
- Robin L. Getz <rgetz@esd.nsc.com>
- Richard Cornelius <richc@rsf.atd.ucar.edu>
- Rob Cunningham <rkc@xn.ll.mit.edu>
- Robert.Kocjancic@IJS.si
- Randall C. O'Reilly <ro2m@crab.psy.cmu.edu>
- Rutvik Desai <rudesai@cs.indiana.edu>
- Robert W. Means <rwmeans@hnc.com>
- Stefan Vogt <s_vogt@cis.umassd.edu>
- Osamu Saito <saito@nttica.ntt.jp>
- Scott Fahlman <sef+@cs.cmu.edu>
- <seibert@ll.mit.edu>
- Sheryl Cormicle <sherylc@umich.edu>
- Ted Stockwell <ted@aps1.spa.umn.edu>
- Stephanie Warrick <S.Warrick@cs.ucl.ac.uk>
- Serge Waterschoot <swater@minf.vub.ac.be>
- Thomas G. Dietterich <tgd@research.cs.orst.edu>
- Thomas.Vogel@cl.cam.ac.uk
- Ulrich Wendl <uli@unido.informatik.uni-dortmund.de>
- M. Verleysen <verleysen@dice.ucl.ac.be>
- VestaServ@aol.com
- Sherif Hashem <vg197@neutrino.pnl.gov>
- Matthew P Wiener <weemba@sagi.wistar.upenn.edu>
- Wesley Elsberry <welsberr@orca.tamu.edu>
- Dr. Steve G. Romaniuk <ZLXX69A@prodigy.com>

Special thanks to Gregory E. Heath <heath@ll.mit.edu> and Will Dwinnell <predictor@delphi.com> for years of stimulating and educational discussions on comp.ai.neurtal-nets.

The FAQ was created in June/July 1991 by Lutz Prechelt; he also maintained the FAQ until November 1995. Warren Sarle maintains the FAQ since December 1995.

Bye

Warren & Lutz

Previous part is part 6.

Neural network FAQ / Warren S. Sarle, saswss@unx.sas.com

Measurement theory: Frequently asked questions

Warren S. Sarle
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513, USA
saswss@unx.sas.com

URL: ftp://ftp.sas.com/pub/neural/measurement.html

Originally published in the Disseminations of the International Statistical Applications Institute, 4th edition, 1995, Wichita: ACG Press, pp. 61-66. Revised March 18, 1996.

Contents

Measurement theory is a branch of applied mathematics that is useful in measurement and data analysis. The fundamental idea of measurement theory is that measurements are not the same as the attribute being measured. Hence, if you want to draw conclusions about the attribute, you must take into account the nature of the correspondence between the attribute and the measurements.

The mathematical theory of measurement is elaborated in:

> Krantz, D. H., Luce, R. D., Suppes, P., and Tversky, A. (1971). Foundations of measurement. (Vol. I: Additive and polynomial representations.). New York: Academic Press.

Suppes, P., Krantz, D. H., Luce, R. D., and Tversky, A. (1989). Foundations of measurement. (Vol. II: Geometrical, threshold, and probabilistic respresentations). New York: Academic Press.

Luce, R. D., Krantz, D. H., Suppes, P., and Tversky, A. (1990). Foundations of measurement. (Vol. III: Representation, axiomatization, and invariance). New York: Academic Press.

Measurement theory was popularized in psychology by S. S. Stevens, who originated the idea of levels of measurement. His relevant articles include:

Stevens, S. S. (1946), On the theory of scales of measurement. Science, 103, 677-680.

Stevens, S. S. (1951), Mathematics, measurement, and psychophysics. In S. S. Stevens (ed.), Handbook of experimental psychology, pp 1-49). New York: Wiley.

Stevens, S. S. (1959), Measurement. In C. W. Churchman, ed., Measurement: Definitions and Theories, pp. 18-36. New York: Wiley. Reprinted in G. M. Maranell, ed., (1974) Scaling: A Sourcebook for Behavioral Scientists, pp. 22-41. Chicago: Aldine.

Stevens, S. S. (1968), Measurement, statistics, and the schemapiric view. Science, 161, 849-856.

What is measurement?

Measurement of some attribute of a set of things is the process of assigning numbers or other symbols to the things in such a way that relationships of the numbers or symbols reflect relationships of the attribute being measured. A particular way of assigning numbers or symbols to measure something is called a *scale* of measurement.

Suppose we have a collection of straight sticks of various sizes and we assign a number to each stick by measuring its length using a ruler. If the number assigned to one stick is greater than the number assigned to another stick, we can conclude that the first stick is longer than the second. Thus a relationship among the numbers (greater than) corresponds to a relationship among the sticks (longer than). If we lay two sticks end-to-end in a straight line and measure their combined length, then the number we assign to the concatenated sticks will equal the sum of the numbers assigned to the individual sticks (within measurement error). Thus another relationship among the numbers (addition) corresponds to a relationship among the sticks (concatenation).

Why should I care about measurement theory?

Measurement theory helps us to avoid making meaningless statements. A typical example of such a meaningless statement is the claim by the weatherman on the local TV station that it was twice as warm today as yesterday because it was 40 degrees Fahrenheit today but only 20 degrees yesterday. This statement is meaningless because one measurement (40) is twice the other measurement (20) only in certain arbitrary scales of measurement, such as Fahrenheit. The relationship 'twice-as' applies only to the numbers, not the attribute being measured (temperature).

When we measure something, the resulting numbers are usually, to some degree, arbitrary. We *choose* to use a 1 to 5 rating scale instead of a -2 to 2 scale. We choose to use Fahrenheit instead of Celsius. We choose to use miles per gallon instead of gallons per mile. The conclusions of a statistical analysis should not depend on these arbitrary decisions, because we could have made the decisions differently. We want the statistical analysis to say

something about reality, not simply about our whims regarding meters or feet.

Suppose we have a rating scale where several judges rate the goodness of flavor of several foods on a 1 to 5 scale. If we want to draw conclusions about the measurements, i.e. the 1-to-5 ratings, then we need not be concerned about measurement theory. For example, if we want to test the hypothesis that the foods have equal mean ratings, we might do a two-way ANOVA on the ratings.

But if we want to draw conclusions about flavor, then we *must* consider how flavor relates to the ratings, and that is where measurement theory comes in. Ideally, we would want the ratings to be linear functions of the flavors with the same slope for each judge; if so, the ANOVA can be used to make inferences about mean goodness-of-flavors, providing we can justify all the appropriate statistical assumptions. But if the judges have different slopes relating ratings to flavor, or if these functions are not linear, then this ANOVA will *not* allow us to make inferences about mean goodness-of-flavor. Note that this issue is not about statistical interaction; even if there is no evidence of interaction in the ratings, the judges may have different functions relating ratings to flavor.

We need to consider what information we have about the functions relating ratings to flavor for each judge. Perhaps the only thing we are sure of is that the ratings are monotone increasing functions of flavor. In this case, we would want to use a statistical analysis that is valid no matter what the particular monotone increasing functions are. One way to do this is to choose an analysis that yields invariant results no matter what monotone increasing functions the judges happen to use, such as a Friedman test. The study of such invariances is a major concern of measurement theory.

However, no measurement theorist would claim that measurement theory provides a complete solution to such problems. In particular, measurement theory generally does not take random measurement error into account, and if such errors are an important aspect of the measurement process, then additional methods, such as latent variable models, are called for. There is no clear boundary between measurement theory and statistical theory; for example, a Rasch model is both a measurement model and a statistical model.

What are permissible transformations?

Permissible transformations are transformations of a scale of measurement that preserve the relevant relationships of the measurement process. *Permissible* is a technical term; use of this term does not imply that other transformations are prohibited for data analysis any more than use of the term *normal* for probability distributions implies that other distributions are pathological. If Stevens had used the term *mandatory* rather than *permissible*, a lot of confusion might have been avoided.

In the example of measuring sticks, changing the unit of measurement (say, from centimeters to inches) multiplies the measurements by a constant factor. This multiplication does not alter the correspondence of the relationships 'greater than' and 'longer than', nor the correspondence of addition and concatenation. Hence, change of units is a permissible transformation with respect to these relationships.

What are levels of measurement?

There are different levels of measurement that involve different properties (relations and operations) of the numbers or symbols that constitute the measurements. Associated with each level of measurement is a set of permissible transformations. The most commonly discussed levels of measurement are as follows:

Nominal

- Two things are assigned the same symbol if they have the same value of the attribute.
- Permissible transformations are any one-to-one or many-to-one transformation, although a many-to-one transformation loses information.
- Examples: numbering of football players; numbers assigned to religions in alphabetical order, e.g. atheist=1, Buddhist=2, Christian=3, etc.

Ordinal

- Things are assigned numbers such that the order of the numbers reflects an order relation defined on the attribute. Two things `x` and `y` with attribute values `a(x)` and `a(y)` are assigned numbers `m(x)` and `m(y)` such that if `m(x) > m(y)`, then `a(x) > a(y)`.
- Permissible transformations are any monotone increasing transformation, although a transformation that is not strictly increasing loses information.
- Examples: Moh's scale for hardness of minerals; grades for academic performance (A, B, C, ...); blood sedimentation rate as a measure of intensity of pathology.

Interval

- Things are assigned numbers such that differences between the numbers reflect differences of the attribute. If `m(x) - m(y) > m(u) - m(v)`, then `a(x) - a(y) > a(u) - a(v)`.
- Permissible transformations are any affine transformation `t(m) = c * m + d`, where `c` and `d` are constants; another way of saying this is that the origin and unit of measurement are arbitrary.
- Examples: temperature in degrees Fahrenheit or Celsius; calendar date.

Log-interval

- Things are assigned numbers such that ratios between the numbers reflect ratios of the attribute. If `m(x) / m(y) > m(u) / m(v)`, then `a(x) / a(y) > a(u) / a(v)`.
- Permissible transformations are any power transformation `t(m) = c * m ** d`, where `c` and `d` are constants.
- Examples: density (mass/volume); fuel efficiency in mpg.

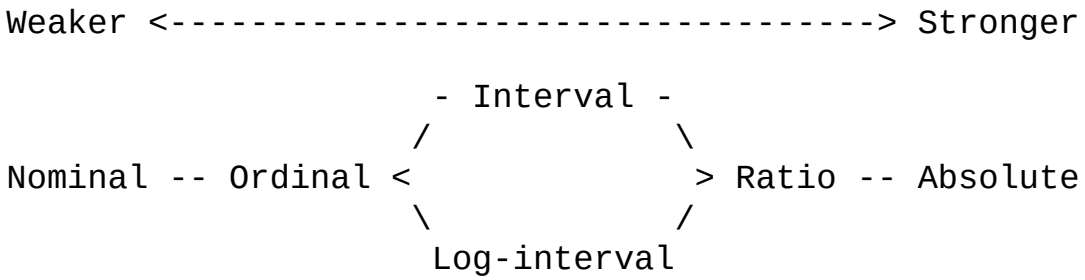
Ratio

- Things are assigned numbers such that differences and ratios between the numbers reflect differences and ratios of the attribute.
- Permissible transformations are any linear (similarity) transformation `t(m) = c * m`, where c is a constant; another way of saying this is that the unit of measurement is arbitrary.
- Examples: Length in centimeters; duration in seconds; temperature in degrees Kelvin.

Absolute

- Things are assigned numbers such that all properties of the numbers reflect analogous properties of the attribute.
- The only permissible transformation is the identity transformation.
- Examples: number of children.

These measurement levels form a partial order based on the sets of permissible transformations:



In real life, a scale of measurement may not correspond precisely to any of these levels of measurement. For example, there can be a mixture of nominal and ordinal information in a single scale, such as in questionnaires that have several non-response categories. It is common to have scales that lie somewhere between the ordinal and interval levels in that the measurements can be assumed to be a smooth monotone function of the attribute.

For many subjective rating scales (such as the 'strongly agree,' 'agree,' ... 'strongly disagree' variety) it cannot be shown that the intervals between successive ratings are exactly equal, but with reasonable care and diagnostics it may be safe to say that no interval represents a difference more than two or three times greater than another interval.

Unfortunately, there are also many situations where the measurement process is too ill-defined for measurement theory to apply. In such cases, it may still be fruitful to consider what arbitrary choices were made in the course of measurement, what effect these choices may have had on the measurements, and whether some plausible class of permissible transformations can be determined.

What about binary (0/1) variables?

For a binary variable, the classes of one-to-one transformations, monotone increasing/decreasing transformations, and affine transformations are identical--you can't do anything with a one-to-one transformation that you can't do with an affine tranformation. Hence binary variables are at least at the interval level. If the variable connotes presence/absence or if there is some other distinguishing feature of one category, a binary variable may be at the ratio or absolute level.

Nominal variables are often analyzed in linear models by coding binary dummy variables. This procedure is justified since binary variables are at the interval level or higher.

Is measurement level a fixed, immutable property of the data?

Measurement level depends on the correspondence between the measurements and the attribute. Given a set of data, one cannot say what the measurement level is without knowing what attribute is being measured. It is possible that a certain data set might be treated as measuring different attributes at different times for different purposes.

Consider a rat in a Skinner box who pushes a lever to get food pellets. The number of pellets dispensed in the course of an experiment is obviously an absolute-level measurement of the number of pellets dispensed. If number of pellets is considered as a measure of some other attribute, the measurement level may differ. As a measure of amount of food dispensed, the number of pellets is at the ratio level under the assumption that the pellets are of equal size; if the pellets are not of equal size, a more elaborate measurement model is required, perhaps one involving random measurement error if the pellets are dispensed in random order. As a measure of duration during the experiment, the number of pellets is at an ordinal level. As a measure of response effort, the number of pellets might be approximately ratio level, but we would need to consider whether the rat's responses were executed in a consistent way, whether the rat may miss the lever, and so forth. As a measure of amount of reward, the number of pellets could only be justified by some very strong assumptions about the nature of rewards; the measurement level would depend on the precise nature of those assumptions. The main virtue of measurement theory is that it encourages people to consider such issues.

Once a set of measurements have been made on a particular scale, it is possible to transform the measurements to yield a new set of measurements at a different level. It is always possible to transform from a stronger level to a weaker level. For example, a temperature measurement in degrees Kelvin is at the ratio level. If we convert the measurements to degrees Celsius, the level is interval. If we rank the measurements, the level becomes ordinal. In some cases it is possible to convert from a weaker scale to a stronger scale. For example, correspondence analysis can convert nominal or ordinal measurements to an interval scale under appropriate assumptions.

Isn't an ordinal scale just an interval scale with error?

You can view an ordinal scale as an interval scale with error if you really want to, but the errors are not independent, additive, or identically distributed as required for many statistical methods. The errors would involve complicated dependencies to maintain monotonicity with the interval scale. In the example above with number of pellets as a measure of duration, the errors would be cumulative, not additive, and the error variance would increase over time. Hence for most statistical purposes, it useless to consider an ordinal scale as an interval scale with measurement error.

What does measurement level have to do with discrete vs. continuous?

Measurement level has nothing to do with discrete vs. continuous variables.

The distinction between discrete and continuous random variables is commonly used in statistical theory, but that distinction is rarely of importance in practice. A continuous random variable has a continuous cumulative distribution function. A discrete random variable has a stepwise-constant cumulative distribution function. A discrete random variable can take only a finite number of distinct values in any finite interval. There exist random variables that are neither continuous nor discrete; for example, if Z is a standard normal random variable and Y=max(0,Z), then Y is neither continuous nor discrete, but has characteristics of both.

While measurements are always discrete due to finite precision, attributes can be conceptually either discrete or continuous regardless of measurement level. Temperature is usually regarded as a continuous attribute, so temperature measurement to the nearest degree Kelvin is a ratio-level measurement of a continuous attribute. However, quantum mechanics holds that the universe is fundamentally discrete, so temperature may actually be a discrete attribute. In ordinal scales for continuous attributes, ties are impossible (or have probability zero). In ordinal scales for discrete attributes, ties are possible. Nominal scales usually apply to discrete attributes. Nominal scales for continuous attributes can be modeled but are rarely used.

Don't the theorems in a statistics textbook prove the validity of statistical methods without reference to measurement theory?

Mathematical statistics is concerned with the connection between inference and data. Measurement theory is concerned with the connection between data and reality. Both statistical theory and measurement theory are necessary to make inferences about reality.

Does measurement level detemine what statistics are valid?

Measurement theory cannot determine some single statistical method or model as appropriate for data at a specific level of measurement. But measurement theory does in fact show that some some statistical methods are inappropriate for certain levels of measurement if we want to make meaningful inferences about the attribute being measured.

If we want to make statistical inferences regarding an attribute based on a scale of measurement, the statistical method must yield invariant or equivariant results under the permissible transformations for that scale of measurement. If this invariance or equivariance does not hold, then the statistical inferences apply only to the measurements, not to the attribute that was measured.

If we record the temperature in degrees Fahrenheit in Cary, NC, at various times, we can compute statistics such as the mean, standard deviation, and coefficient of variation. Since Fahrenheit is an interval scale, only statistics that are invariant or equivariant under change of origin or unit of measurement are meaningful. The mean is meaningful because it is equivariant under change of origin or unit. The standard deviation is meaningful because it is invariant under change of origin and equivariant under change of unit. But the coefficient of variation is meaningless because it lacks such invariance or equivariance. The mean and standard deviation can easily be converted back and forth from Fahrenheit to Celsius, but we cannot compute the coefficient of variation in degrees Celsius if we know only the coefficient of variation in degrees Fahrenheit.

Paul Thompson provides an example where interval and ratio levels are confused:

```
> ... I recently published a
> paper in Psychiatric Research.  We discuss the BPRS, a very common rating
> scale in psychiatry.  Oddly enough, in the BPRS in the US, '1' means NO
> PATHOLOGY.  However, a frequent statistic computed is percent improved:
>
>      PI = (BPRS(Base)-BPRS(6week) ) / BPRS(Base) * 100
>
> If you use the '1 implies no pathology' model, you are not measuring
> according to a ratio scale, which requires a true 0.  We show that this
> has very bad characteristics, which include a flat impossibility for
> a certain % improvement at certain points in the scale.
>
> This is pretty trivial, but should have an effect.  As one reviewer said,
> 'This is so obvious that I am surprised that no one has ever thought of it
> before.'  Nonetheless, the scale was being misused.
```

It is clear that if we are estimating a parameter that lacks invariance or equivariance under permissible transformations, we are estimating a chimera. The situation for hypothesis testing is more subtle. It is nonsense to test a null hypothesis the truth of which is not invariant under permissible transformations. For example, it would be meaningless to test the null hypothesis that the mean temperature in Cary in July is twice the mean temperature in December using a Fahrenheit or Celsius scale--we would need a ratio scale for that hypothesis to be meaningful.

But it is possible for the null hypothesis to be meaningful even if the error rates for a given test are not invariant. Suppose that we had an ordinal scale of temperature, and the null hypothesis was that the distribution of temperatures in July is identical to the distribution in December. The truth of this hypothesis is invariant under strictly increasing monotone transformations and is therefore meaningful under an ordinal scale. But if we do a t-test of this hypothesis, the error rates will not be invariant under monotone transformations. Hard-core measurement theorists would therefore consider a t-test inappropriate. But given a null hypothesis, there are usually many different tests that can be performed with accurate or conservative significance levels but with different levels of power against different alternatives. The fact that different tests have different error rates does not make any of them correct or incorrect. Hence a soft-core measurement theorist might argue that invariance of error rates is not a prerequisite for a meaningful hypothesis test--only invariance of the null hypothesis is

required.

Nevertheless, the hard-core policy rules out certain tests that, while not incorrect in a strict sense, are indisputably poor tests in terms of having absurdly low power. Consider the null hypothesis that two random variables are independent of each other. This hypothesis is invariant under one-to-one transformations of either variable. Suppose we have two nominal variables, say, religion and preferred statistical software product, to which we assign arbitrary numbers. After verifying that at least one of the two variables is approximately normally distributed, we could test the null hypothesis using a Pearson product-moment correlation, and this would be a valid test. However, the power of this test would be so low as to be useless unless we were lucky enough to assign numbers to categories in such a way as to reveal the dependence as a linear relationship. Measurement theory would suggest using a test that is invariant under one-to-one transformations, such as a chi-squared test of independence in a contingency table. Another possibility would be to use a Pearson product-moment correlation after assigning numbers to categories in such a way as to maximize the correlation (although the usual sampling distribution of the correlation coefficient would not apply). In general, we can test for independence by maximizing some measure of dependence over all permissible transformations.

However, it must be emphasized that there is no need to *restrict* the transformations in a statistical analysis to those that are permissible. That is not what *permissible transformation* means. The point is that statistical methods should be used that give invariant results under the class of permissible transformations, because those transformations do not alter the meaning of the measurements. *Permissible* was undoubtedly a poor choice of words, but Stevens was quite clear about what he meant. For example (Stevens 1959):

> In general, the more unrestricted the permissible transformations, the more restricted the statistics. Thus, nearly all statistics are applicable to measurements made on ratio scales, but only a very limited group of statistics may be applied to measurements made on nominal scales.

The connection between measurement level and statistical analysis has been hotly disputed in the psychometric and statistical literature by people who fail to distinguish between inferences regarding the attribute and inferences regarding the measurements. If one is interested only in making inferences about the measurements without regard to their meaning, then measurement level is, of course, irrelevant to choice of statistical method. The classic example is Lord (1953), 'On the Statistical Treatment of Football Numbers', American Psychologist, 8, 750-751, who argued that statistical methods could be applied regardless of level of measurement, and concocted a silly example involving the jersey numbers assigned to football players, which Lord claimed were nominal-level measurements of the football players. Lord contrived a situation in which freshmen claimed they were getting lower numbers than the sophomores, so the purpose of the analysis was to make inferences about the numbers, not about some attribute measured by the numbers. It was therefore quite reasonable to treat the numbers as if they were on an absolute scale. However, this argument completely misses the point by eliminating the measured attribute from the scenario.

The confusion between measurements and attributes was perpetuated by Velleman and Wilkinson (1993), 'Nominal, Ordinal, Interval, and Ratio Typologies Are Misleading', *The American Statistician*, 47, 65-72. Velleman and Wilkinson set up a series of straw men and knocked some of them down, while consistently misunderstanding the meaning of *meaning* and of *permissible transformation*. For example, they claimed that the number of cylinders in an automobile engine can be treated, depending on the circumstances, as nominal, ordinal, interval, or ratio, and hence the concept of measurement level 'simplifies the matter so far as to be false.' In fact, the number of cylinders is at the absolute level of measurement. Thus, measurement theory would dictate that any statistical analysis of number of cylinders must be invariant under an identity transformation. Obviously, *any* analysis is invariant under an identity transformation, so all of the analyses that Velleman and Wilkinson claimed might be appropriate *are* acceptable according to measurement theory. What is false is not measurement theory but Velleman and Wilkinson's backwards interpretation of it.

It is important to understand that the level of measurement of a variable does not mandate how that variable must appear in a statistical model. However, the measurement level does suggest reasonable ways to use a variable by default. Consider the analysis of fuel efficiency in automobiles. If we are interested in the average distance that can be driven with a given amount of gas, we should analyze miles per gallon. If we are interested in the average amount of gas required to drive a given distance, we should analyze gallons per mile. Both miles per gallon and gallons per mile are measurements of fuel efficiency, but they may yield quite different results in a statistical analysis, and there may be no clear reason to use one rather than the other. So how can we make inferences regarding fuel efficiency that do not depend on the choice between these two scales of measurement? We can do that by recognizing that both miles per gallon and gallons per mile are measurements of the same attribute on a log-interval scale, and hence that the logarithm of either can be treated as a measurement on an interval scale. Thus, if we were doing a regression, it would be reasonable to begin the analysis using log(mpg). If evidence of nonlinearity were detected, then other transformations could still be considered.

But measurement level has been shown empirically to be irrelevant to statistical results, hasn't it?

What has been shown is that various statistical methods are more or less robust to distortions that could arise from smooth monotone transformations; in other words, there are cases where it makes little difference whether we treat a measurement as ordinal or interval. But there can hardly be any doubt that it often makes a huge difference whether we treat a measurement as nominal or ordinal, and confusion between interval and ratio scales is a common source of nonsense.

Suppose we are doing a two-sample t-test; we are sure that the assumptions of ordinal measurement are satisfied, but we are not sure whether an equal-interval assumption is justified. A smooth monotone tranformation of the entire data set will generally have little effect on the p value of the t-test. A robust variant of a t-test will likely be affected even less (and, of course, a rank version of a t-test will be affected not at all). It should come as no surprise then that a decision between an ordinal or an interval level of measurement is of no great importance in such a situation, but anyone with lingering doubts on the matter may consult the simulations in Baker, B. O., Hardyck, C, and Petrinovich, L. F. (1966) 'Weak measurement vs. strong statistics: An empirical critique of S.S. Stevens' proscriptions on statistics,' Educational and Psychological Measurement, 26, 291-309, for a demonstration of the obvious.

On the other hand, suppose we were comparing the variability instead of the location of the two samples. The F test for equality of variances is not robust, and smooth monotone transformations of the data could have a large effect on the p value. Even a more robust test could be highly sensitive to smooth monotone transformations if the samples differed in location.

Measurement level is of greatest importance in situations where the meaning of the null hypothesis depends on measurement assumptions. Suppose the data are 1-to-5 ratings obtained from two groups of people, say males and females, regarding how often the subjects have sex: frequently, sometimes, rarely, etc. Suppose that these two groups interpret the term 'frequently' differently as applied to sex; perhaps males consider 'frequently' to mean twice a day, while females consider it to mean once a week. Females may report having sex more 'frequently' than men on the 1-to-5 scale, even if men in fact have sex more frequently as measured by sexual acts per unit of time. Hence measurement considerations are crucial to the interpretation of the results.

What are some more examples of how measurement level relates to statistical methodology?

As mentioned earlier, it is meaningless to claim that it was twice as warm today as yesterday because it was 40 degrees Fahrenheit today but only 20 degrees yesterday. Fahrenheit is not a ratio scale, and there is no meaningful sense in which 40 degrees is twice as warm as 20 degrees. It would be just as meaningless to compute the geometric mean or coefficient of variation of a set of temperatures in degrees Fahrenheit, since these statistics are not invariant or equivariant under change of origin. There are many other statistics that can be meaningfully applied only to data at a sufficiently strong level of measurement.

Consider some measures of location: the mode requires a nominal or stronger scale, the median requires an ordinal or stronger scale, the arithmetic mean requires an interval or stronger scale, and the geometric mean or harmonic mean require a ratio or stronger scale.

Consider some measures of variation: entropy requires a nominal or stronger scale, the standard deviation require an interval or stronger scale, and the coefficient of variation requires a ratio or stronger scale.

Simple linear regression with an intercept requires that both variables be on an interval or stronger scale. Regression through the origin requires that both variables be on a ratio or stronger scale.

A generalized linear model using a normal distribution requires the dependent variable to be on an interval or stronger scale. A gamma distribution requires a ratio or stronger scale. A Poisson distribution requires an absolute scale.

The general principle is that an appropriate statistical analysis must yield invariant or equivariant results for all permissible transformations. Obviously, we cannot actually conduct an infinite number of analyses of a real data set corresponding to an infinite class of transformations. However, it is often straightforward to verify or falsify the invariance mathematically. The application of this idea to summary statistics such as means and coefficients of variation is fairly widely understood.

Confusion arises when we come to linear or nonlinear models and consider transformations of variables. Recall that Stevens did *not* say that transformations that are not 'permissible' are prohibited. What Stevens said was that we should consider *all* 'permissible' transformations and verify that our conclusions are invariant.

Consider, for example, the problem of estimating the parameters of a nonlinear model by maximum likelihood (ML), and comparing various models by likelihood ratio (LR) tests . We would want the LR tests to be invariant under the permissible transformations of the variables. One way to do this is to parameterize the model so that any permissible transformation can be inverted by a corresponding change in the parameter estimates. In other words, we can make the ML and LR tests invariant by making the inverse-permissible transformations mandatory (this is the same set of transformations as the permissible transformations except for a degeneracy here and there which I won't worry about).

To illustrate, suppose we are modeling a variable Y as a function `f()` of variables `N, O, I, L, R`, and `A` at the nominal, ordinal, etc. measurement levels, respectively. Then we can ensure the desired invariance by setting up the model as:

```
Y = f( arb(N), mon(O), a+bI, cL^d, eR, A, ...)
```

where `arb()` is any (estimated) function, `mon()` is any (estimated) monotone function, and `a, b, c, d`, and `e` are parameters. Then any permissible transformations of `N, O, I, L, R`, and `A` can be absorbed by the estimation of the `arb()` and `mon()` functions and the parameters. The function `f()` can involve *any other* transformations such as sqrts or logs or whatever. `f()` can be as complicated as you like--the presence of the

permissible transformations as part of the model to be estimated guarantees the desired invariance.

If we were designing software for fitting linear or nonlinear models, we might want to provide these 'permissible' or 'mandatory' transformations in a convenient way. This, in fact, was the motivation for numerous programs developed by psychometricians that anticipated many of the features of ACE and generalized additive models.

What's the bottom line?

Measurement theory shows that strong assumptions are required for certain statistics to provide meaningful information about reality. Measurement theory encourages people to think about the meaning of their data. It encourages critical assessment of the assumptions behind the analysis. It encourages responsible real-world data analysis.

LBA for WSS, 18 Mar 1996

Neural Network and Statistical Jargon

Warren S. Sarle saswss@unx.sas.com Apr 29, 1996

URL: ftp://ftp.sas.com/pub/neural/jargon

The neural network (NN) and statistical literatures contain many of the same concepts but usually with different terminology. Sometimes the same term or acronym is used in both literatures but with different meanings. Only in very rare cases is the same term used with the same meaning, although some cross-fertilization is beginning to happen. Below is a list of such corresponding terms or definitions.

Particularly loose correspondences are marked by a ~ between the two columns. A < indicates that the term on the left is roughly a subset of the term on the right, and a > indicates the reverse. Terminology in both fields is often vague, so precise equivalences are not always possible. The list starts with some basic definitions.

There is disagreement in the NN literature on how to count layers. Some people count inputs as a layer and some don't. I specify the number of hidden layers instead. This is awkward but unambiguous.

Definition	Statistical Jargon
generalizing from noisy data and assessment of the accuracy thereof	Statistical inference
the set of all cases one wants to be able to generalize to	Population
a function of the values in a population, such as the mean or a globally optimal synaptic weight	Parameter
a function of the values in a sample, such as the mean or a learned synaptic weight	Statistic

Neural Network Jargon	Definition
Neuron, neurode, unit, node, processing element	a simple linear or nonlinear computing element that accepts one or more inputs, computes a function thereof, and may direct the result to one or more other neurons
Neural networks	a class of flexible nonlinear regression and discriminant models, data reduction models, and nonlinear dynamical systems consisting of an often large number of neurons interconnected in often complex ways and often organized into layers

Neural Network Jargon	Statistical Jargon
Statistical methods	Linear regression and discriminant

	analysis, simulated annealing, random search
Architecture	Model
Training, Learning, Adaptation	Estimation, Model fitting, Optimization
Classification	Discriminant analysis
Mapping, Function approximation	Regression
Supervised learning	Regression, Discriminant analysis
Unsupervised learning, Self-organization	Principal components, Cluster analysis, Data reduction
Competitive learning	Cluster analysis
Hebbian learning, Cottrell/Munro/Zipser technique	Principal components
Training set	Sample, Construction sample
Test set, Validation set	Hold-out sample
Pattern, Vector, Example, Case	Observation, Case
Reflectance pattern	an observation normalized to sum to 1
Binary(0/1), Bivalent or Bipolar(-1/1)	Binary, Dichotomous
Input	Independent variables, Predictors, Regressors, Explanatory variables, Carriers
Output	Predicted values
Forward propagation	Prediction
Training values Target values	Dependent variables, Responses, Observed values
Training pair	Observation containing both inputs and target values
Shift register, (Tapped) (time) delay (line), Input window	Lagged variable
Errors	Residuals
Noise	Error term
Generalization	Interpolation, Extrapolation, Prediction
Error bars	Confidence interval
Prediction	Forecasting
Adaline (ADAptive LInear NEuron)	Linear two-group discriminant analysis (not Fisher's but generic)
(No-hidden-layer) perceptron ~	Generalized linear model (GLIM)

Activation function, Signal function, Transfer function	>	Inverse link function in GLIM
Softmax		Multiple logistic function
Squashing function		bounded function with infinite domain
Semilinear function		differentiable nondecreasing function
Phi-machine		Linear model
Linear 1-hidden-layer perceptron		Maximum redundancy analysis, Principal components of instrumental variables
1-hidden-layer perceptron	~	Projection pursuit regression
Weights, Synaptic weights	<	(Regression) coefficients, Parameter estimates
Bias	~	Intercept
the difference between the expected value of a statistic and the corresponding true value (parameter)		Bias
Shortcuts, Jumpers, Bypass connections, direct linear feedthrough (direct connections from input to output)	~	Main effects
Functional links		Interaction terms or transformations
Second-order network		Quadratic regression, Response-surface model
Higher-order network		Polynomial regression, Linear model with interaction terms
Instar, Outstar		iterative algorithms of doubtful convergence for approximating an arithmetic mean or centroid
Delta rule, adaline rule, Widrow-Hoff rule, LMS (Least Mean Squares) rule		iterative algorithm of doubtful convergence for training a linear perceptron by least squares, similar to stochastic approximation
training by minimizing the median of the squared errors		LMS (Least Median of Squares)
Generalized delta rule		iterative algorithm of doubtful convergence for training a nonlinear perceptron by least squares, similar to stochastic approximation
Back propagation		Computation of derivatives for a multilayer perceptron and various algorithms (such as the generalized delta rule) based thereon
Weight decay, Regularization	>	Shrinkage estimation, Ridge regression
Jitter		random noise added to the inputs to smooth the estimates

Growing, Pruning, Brain damage, Self-structuring, Ontogeny		Subset selection, Model selection, Pre-test estimation
Optimal brain surgeon		Wald test
LMS (Least mean squares)		OLS (Ordinary least squares) (see also "LMS rule" above)
Relative entropy, Cross entropy		Kullback-Leibler divergence
Evidence framework		Empirical Bayes estimation
OLS (Orthogonal least squares)		Forward stepwise regression
Probabilistic neural network		Kernel discriminant analysis
General regression neural network		Kernel regression
Topologically distributed encoding	<	(Generalized) Additive model
Adaptive vector quantization		iterative algorithms of doubtful convergence for K-means cluster analysis
Adaptive Resonance Theory 2a	~	Hartigan's leader algorithm
Learning vector quantization		a form of piecewise linear discriminant analysis using a preliminary cluster analysis
Counterpropagation		Regressogram based on k-means clusters
Encoding, Autoassociation		Dimensionality reduction (Independent and dependent variables are the same)
Heteroassociation		Regression, Discriminant analysis (Independent and dependent variables are different)
Epoch		Iteration
Continuous training, Incremental training, On-line training, Instantaneous training		Iteratively updating estimates one observation at a time via difference equations, as in stochastic approximation
Batch training, Off-line training		Iteratively updating estimates after each complete pass over the data as in most nonlinear regression algorithms