URL: ftp://ftp.sas.com/pub/neural/FAQ.html

Neural Networks FAQ's

Warren S. Sarle, 1997

Part 1: Introduction

Part 2: Learning

Part 3: Generalization

How is generalization possible?
How does noise affect generalization?
What is overfitting and how can I avoid it?
What is jitter? (Training with noise)
What is early stopping?
What is weight decay?
What is Bayesian learning?
How many hidden layers should I use?
How many hidden units should I use?
How can generalization error be estimated?
What are cross-validation and bootstrapping?

Part 4: Books, data, etc.

Books and articles about Neural Networks?
Journals and magazines about Neural Networks?
The most important conferences concerned with Neural Networks?
Neural Network Associations?
On-line and machine-readable information about NNs?
How to benchmark learning methods?
Databases for experimentation with NNs?

Part 5: Free software

Freeware and shareware packages for NN simulation?

Part 6: Commercial software

Commercial software packages for NN simulation?

Part 7: Hardware, etc.

Neural Network hardware?

How to learn an inverse of a function?
How to get invariant recognition of images under translation, rotation, etc.?
Unanswered FAQs

Archive-name: ai-faq/neural-nets/part1
Last-modified: 1997-09-27
URL: ftp://ftp.sas.com/pub/neural/FAQ.html
Maintainer: saswss@unx.sas.com (Warren S. Sarle)
Copyright 1997 by Warren S. Sarle, Cary, NC, USA.

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 at 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 Mosaic, under the URL: "ftp://ftp.sas.com/pub/neural/FAQ.html".

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 on 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.

This FAQ is not meant to discuss any topic exhaustively.

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 or html or TEX or any word-processor format.
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:

- Deja News at http://www.dejanews.com/
- ftp://ftp.cs.cmu.edu/user/ai/pubs/news/comp.ai.neural-nets
- http://asknpac.npac.syr.edu

  According to Gang Cheng, gcheng@npac.syr.edu, the Northeast Parallel Architecture Center (NPAC), Syracue University, maintains an archive system for searching/reading USENET newsgroups and mailing lists. Two search/navigation interfaces accessible by any WWW browser are provided: one is an advanced search interface allowing queries with various options such as query by mail

header, by date, by subject (keywords), by sender. The other is a Hypermail-like navigation interface for users familiar with Hypermail.

For more information on newsgroup archives, see
http://starbase.neosoft.com/~claird/news.lists/newsgroup_archives.html

---

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)?

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 recognize dogs from examples of dogs) and exhibit some capability for generalization beyond the training data.

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, S. (1994), *Neural Networks: A Comprehensive Foundation,* NY: Macmillan, 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, A. (1993), *Neural Networks for Pattern Recognition,* Cambridge, MA: The MIT Press, 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, J.M. (1992), *Introduction To Artificial Neural Systems,* Boston: PWS Publishing Company, p. xv:

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

For more information on "What is a neural network?", with examples and diagrams, see Leslie S. Smith's on-line introduction at:
http://www.cs.stir.ac.uk/~lss/NNIntro/InvSlides.html.

Another excellent introduction to NNs is Donald Tveter's Backpropagator's Review at http://www.mcs.com/~drt/bprefs.html, which contains both answers to additional FAQs and an annotated neural net bibliography emphasizing on-line articles.

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.

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 mapping between vector spaces 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 are statistically consistent estimators of arbitrary measurable, 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 (Farag\'o and Lugosi, 1993; Lugosi and Zeger 1995; Devroye, Gy\"orfi, and Lugosi, 1996) under similar assumptions.

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 (Judd, 1990).

NNs are, at least today, difficult to apply successfully to problems that concern manipulation of symbols and memory. And there are no methods for training NNs that can magically create information that is not contained in the training data.

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

For examples of NN applications, see:

- The Pacific Northwest National Laboratory web pages at http://www.emsl.pnl.gov:2080/docs/cie/neural/ including a list of commercial applications at http://www.emsl.pnl.gov:2080/docs/cie/neural/products/
- The Stimulation Initiative for European Neural Applications web page at http://www.mbfys.kun.nl/snn/siena/cases/

- Roy Goodacre's web pages on pyrolysis mass spectrometry at http://gepasi.dbs.aber.ac.uk/roy/pymshome.htm and Fourier transform infrared (FT-IR) spectroscopy at 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.
- The DTI NeuroComputing Web's Applications Portfolio at http://www.globalweb.co.uk/nctt/portfolo/
- The Applications Corner, provided by NeuroDimension, Inc., at http://www.nd.com/appcornr/purpose.htm
- The BioComp Systems, Inc. Solutions page at http://www.bio-comp.com
- Athanasios Episcopos's web page with References on Neural Net Applications to Finance and Economics at http://www.compulink.gr/users/episcopo/neurofin.html
- Chen, C.H., ed. (1996) *Fuzzy Logic and Neural Network Handbook,* NY: McGraw-Hill, ISBN 0-07-011189-8.
- Trippi, R.R. & Turban, E. (1993), *Neural Networks in Finance and Investing,* Chicago: Probus.
- 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.

There is 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. According to J.J. Merelo:

> You only need to submit 41 CD values ranging from 200 nm to 240 nm (given in deg cm^2 dmol^-1 multiplied by 0.001) and the k2d server gives back the estimated percentages of helix, beta and rest of secondary structure of your protein plus an estimation of the accuracy of the prediction.

The address of the k2d server is http://kal-el.ugr.es/k2d/spectra.html. The home page of the k2d program is at http://kal-el.ugr.es/k2d/k2d.html or http://www.embl-heidelberg.de/~andrade/k2d.html.

References:

> Devroye, L., Gy\"orfi, L., and Lugosi, G. (1996), *A Probabilistic Theory of Pattern Recognition,* NY: Springer.
> Farag\'o, A. and Lugosi, G. (1993), "Strong Universal Consistency of Neural Network Classifiers," IEEE Transactions on Information Theory, 39, 1146-1151.
> 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.
> 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 existing ones) are invented every week. Below is a collection of some of the most well known methods, not claiming to be complete.

The main categorization of these methods is the distinction between supervised and unsupervised learning:

- In supervised learning, there is a "teacher" who in the learning phase "tells" the net how well it performs ("reinforcement learning") or what the correct behavior would have been ("fully supervised learning").
- In unsupervised learning the net is autonomous: it just looks at the data it is presented with, finds out about some of the properties of the data set and learns to reflect these properties in its output. What exactly these properties are, that the network can learn to recognise, depends on the particular network model and

learning method. Usually, the net learns some compressed representation of the data.

Many of these learning methods are closely connected with a certain (class of) network topology.

Now here is the list, just giving some names:

1. UNSUPERVISED LEARNING (i.e. without a "teacher"):
   1). Feedback Nets:
      a). Additive Grossberg (AG)
      b). Shunting Grossberg (SG)
      c). Binary Adaptive Resonance Theory (ART1)
      d). Analog Adaptive Resonance Theory (ART2, ART2a)
      e). Discrete Hopfield (DH)
      f). Continuous Hopfield (CH)
      g). Discrete Bidirectional Associative Memory (BAM)
      h). Temporal Associative Memory (TAM)
      i). Adaptive Bidirectional Associative Memory (ABAM)
      j). Kohonen Self-organizing Map/Topology-preserving map (SOM/TPM)
      k). Competitive learning
   2). Feedforward-only Nets:
      a). Learning Matrix (LM)
      b). Driver-Reinforcement Learning (DR)
      c). Linear Associative Memory (LAM)
      d). Optimal Linear Associative Memory (OLAM)
      e). Sparse Distributed Associative Memory (SDM)
      f). Fuzzy Associative Memory (FAM)
      g). Counterprogation (CPN)

2. SUPERVISED LEARNING (i.e. with a "teacher"):
   1). Feedback Nets:
      a). Brain-State-in-a-Box (BSB)
      b). Fuzzy Congitive Map (FCM)
      c). Boltzmann Machine (BM)
      d). Mean Field Annealing (MFT)
      e). Recurrent Cascade Correlation (RCC)
      f). Backpropagation through time (BPTT)
      g). Real-time recurrent learning (RTRL)
      h). Recurrent Extended Kalman Filter (EKF)
   2). Feedforward-only Nets:
      a). Perceptron
      b). Adaline, Madaline
      c). Backpropagation (BP)
      d). Cauchy Machine (CM)
      e). Adaptive Heuristic Critic (AHC)
      f). Time Delay Neural Network (TDNN)
      g). Associative Reward Penalty (ARP)
      h). Avalanche Matched Filter (AMF)
      i). Backpercolation (Perc)

j). Artmap
k). Adaptive Logic Network (ALN)
l). Cascade Correlation (CasCor)
m). Extended Kalman Filter(EKF)
n). Learning Vector Quantization (LVQ)
o). Probabilistic Neural Network (PNN)
p). General Regression Neural Network (GRNN)

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; 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. 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; this reduction of the learning rate makes each codebook vector the mean of 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. 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. 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-Max" (Lloyd, 1982; Max, 1960) 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-mean, 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. A SOM tries to find clusters such that any two clusters that are close to each other in the grid 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. The best way to undestand this is to look at the pictures in Kohonen (1995) or various other NN textbooks.

  A SOM works by smoothing the codebook vectors in a manner somewhat 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). Kohonen's algorithm is heuristic, and it is not clear exactly what a Kohonen SOM learns, but recently some new approaches to SOMs have been developed that have better theoretical justification; see "What does unsupervised learning learn?"

  It is important to shrink the smoothing neighborhoods during training. If you do not start with large neighborhoods and shrink them during training, the network can easily get stuck in bad local optima. But Kohonen (1995) is not clear on whether the neighborhoods should shrink to zero. On p. 80 he says that the final neighborhoods "can" contain the nearest neighbors, but on p. 128, regarding the batch SOM algorithm, he says that the final neighborhoods "may" contain only the single cluster. But in the latter case, as Kohonen points out, the SOM is basically a very fancy initialization algorithm for batch k-means, and you could lose the topological mapping properties of the SOM (Kohonen, 1995, p. 111).

  In a 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.

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://nucleus.hut.fi/nnrc/
- 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.hut.fi/~jpronkko/nenet.html

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.

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., Gy\"orfi, 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.

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), *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.
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.
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.
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?

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". 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. 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 [weights] of the classifier.

Validation set:
: A set of examples used to tune the parameters 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.

Bishop (1995), another 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.

The crucial point is that a test set, by definition, 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.

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.
- 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:

- Kohonen's self-organizing maps.
- 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), and an excellent comparison of unsupervised Kohonen networks and k-means clustering by Balakrishnan, Cooper, Jacob, and Lewis (1994).

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.

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.

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.

Hand, D.J. (1981) *Discrimination and Classification,* 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.
McCullagh, P. and Nelder, J.A. (1989) *Generalized Linear Models,* 2nd ed., London: Chapman & Hall.
Michie, D., Spiegelhalter, D.J. and Taylor, C.C. (1994), *Machine Learning, Neural and Statistical Classification*, Ellis Horwood.
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)
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.

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.

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 also has links to information on neuro-genetic methods.

For general information on GAs, try the links at http://www.shef.ac.uk/~gaipp/galinks.html

---

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) 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.
- 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.
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
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.

Next part is part 2 (of 7).

PART 2

Archive-name: ai-faq/neural-nets/part2
Last-modified: 1997-06-21
URL: ftp://ftp.sas.com/pub/neural/FAQ2.html
Maintainer: saswss@unx.sas.com (Warren S. Sarle)
Copyright 1997 by Warren S. Sarle, Cary, NC, USA. Answers provided by other authors as cited below are copyrighted by those authors, who by submitting the answers for the FAQ give permission for the answer to be reproduced as part of the FAQ in any of the ways specified in part 1 of the FAQ.
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: How many learning methods for NNs exist? Which?

There are many many learning methods for NNs by now. Nobody knows exactly how many. New ones (or at least variations of existing ones) are invented every week. Below is a collection of some of the most well known methods, not claiming to be complete.
The main categorization of these methods is the distinction between supervised and unsupervised learning:

- In supervised learning, there is a "teacher" who in the learning phase "tells" the net how well it performs ("reinforcement learning") or what the correct behavior would have been ("fully supervised learning").
- In unsupervised learning the net is autonomous: it just looks at the data it is presented with, finds out about some of the properties of the data set and learns to reflect these properties in its output. What exactly these properties are, that the network can learn to recognise, depends on the particular network model and learning method. Usually, the net learns some compressed representation of the data.

Many of these learning methods are closely connected with a certain (class of) network topology.
Now here is the list, just giving some names:

1. UNSUPERVISED LEARNING (i.e. without a "teacher"):
   1). Feedback Nets:
      a). Additive Grossberg (AG)
      b). Shunting Grossberg (SG)
      c). Binary Adaptive Resonance Theory (ART1)
      d). Analog Adaptive Resonance Theory (ART2, ART2a)
      e). Discrete Hopfield (DH)
      f). Continuous Hopfield (CH)
      g). Discrete Bidirectional Associative Memory (BAM)
      h). Temporal Associative Memory (TAM)
      i). Adaptive Bidirectional Associative Memory (ABAM)
      j). Kohonen Self-organizing Map/Topology-preserving map (SOM/TPM)
      k). Competitive learning
   2). Feedforward-only Nets:
      a). Learning Matrix (LM)
      b). Driver-Reinforcement Learning (DR)
      c). Linear Associative Memory (LAM)
      d). Optimal Linear Associative Memory (OLAM)
      e). Sparse Distributed Associative Memory (SDM)
      f). Fuzzy Associative Memory (FAM)
      g). Counterprogation (CPN)

2. SUPERVISED LEARNING (i.e. with a "teacher"):
   1). Feedback Nets:
      a). Brain-State-in-a-Box (BSB)
      b). Fuzzy Congitive Map (FCM)
      c). Boltzmann Machine (BM)

d). Mean Field Annealing (MFT)
e). Recurrent Cascade Correlation (RCC)
f). Backpropagation through time (BPTT)
g). Real-time recurrent learning (RTRL)
h). Recurrent Extended Kalman Filter (EKF)
2). Feedforward-only Nets:
a). Perceptron
b). Adaline, Madaline
c). Backpropagation (BP)
d). Cauchy Machine (CM)
e). Adaptive Heuristic Critic (AHC)
f). Time Delay Neural Network (TDNN)
g). Associative Reward Penalty (ARP)
h). Avalanche Matched Filter (AMF)
i). Backpercolation (Perc)
j). Artmap
k). Adaptive Logic Network (ALN)
l). Cascade Correlation (CasCor)
m). Extended Kalman Filter(EKF)
n). Learning Vector Quantization (LVQ)
o). Probabilistic Neural Network (PNN)
p). General Regression Neural Network (GRNN)

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 error gradient for a feedforward network, a straightforward but elegant application of the chain rule of elementary calculus (Werbos 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 (Poljak 1964; Bertsekas 1995, 78-79).

Standard backprop can be used for incremental (on-line) training (in which the weights are updated after processing each case) but it 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."

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 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). 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.?".

More on-line info on backprop:

Donald Tveter's Backpropagator's Review at http://www.mcs.com/~drt/bprefs.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.

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

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. (1994), *The Roots of Backpropagation,* NY: John Wiley & Sons.

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.

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.

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 error function diverge, so there is no learning at all. If the error function is quadratic, as in linear models, good learning rates can be computed from the Hessian matrix (Bertsekas and Tsitsiklis, 1996). If the error 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.

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.?").
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. One promising method is given by LeCun, Simard, and Pearlmutter (1993), but I have no personal experience with it. If you have any solid evidence that this or other methods of automatically setting the learning rate in incremental training actually work in a wide variety of 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., *Advances in Neural Information Processing Systems 4,* pp. 1009-1016.

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.

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). 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), 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.
- For a moderate number of weights, various quasi-Newton algorithms are efficient.
- For a large number of weights, various conjugate-gradient algorithms are efficient.

For objective functions that are not continuously differentiable, the Nelder-Mead simplex algorithm and simulated annealing can be used.
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?").
For a survey of optimization software, see More\' 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.
- 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.
- Lester Ingber's page on Adaptive Simulated Annealing (ASA), karate, etc. at http://www.ingber.com/ or http://www.alumni.caltech.edu/~ingber/

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.

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

More\', 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.

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

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.

Subject: How should categories be coded?

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 M\"uller, 1994). If the categories are Red, Green, and Blue, then the data would look like this:

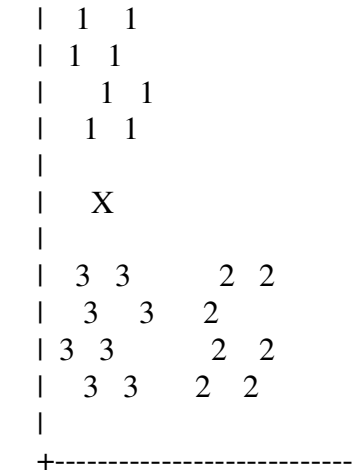
Category	Dummy variables
Red	1 0 0
Green	0 1 0
Blue	0 0 1

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.

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 valid 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

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 the only advantage of effects coding is that the dummy variables require no standardizing (see "Should I normalize/standardize/rescale the data?").

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" 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-coded network are just the differences of successive weights in the 1-of-C-coded 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.

Finke, M. and M\"uller, 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.

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

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. A bias term can be treated as a connection weight from an input with a constant value of one. Hence the bias can be learned just like any other weight. For a linear output unit, a bias term is equivalent to an intercept in a linear regression model.

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 input, 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 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 units. 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 can be used.
Regarding bias-like values 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 composition 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, although for backpropagation learning it 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.
For the output units, you should choose an activation function suited to the distribution of the target values. Bounded activation functions such as the logistic are particularly useful when the target values have a bounded range. But if the target values have no known bounded range, it is better to use an unbounded activation function, most often the identity function (which amounts to no activation function). If the target values are positive but have no known upper bound, you can use an exponential output activation function (but beware of overflow if you are writing your own code).
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", ftp://psyche.mit.edu/pub/jordan/uai.ps.Z.

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

Subject: What is a softmax activation function?

The purpose of the softmax activation function is to make the sum of the outputs equal to one, so that the outputs are interpretable as posterior probabilities. 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:

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

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.

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?

This answer was provided by Janne Sinkkonen:

Curse of dimensionality (Bellman 1961) refers to the exponential growth of hypervolume as a function of dimensionality. What has this to do with the NNs?

Well, NNs are mappings from an input space to an output space. Thus, loosely speaking, an NN needs to "monitor" or cover or represent every part of its input space in order to know how the space should be mapped. Covering the input space takes resources, and the amount of resources needed is proportional to the hypervolume of the input space. This notion seems to hold generally, but formalizing "resources" and "every part of the input space" would take us so deep that we could eventually surface to a different world on the other side of the deepness. :)

Here is an example. Think of an unsupervised competitive one-layer network that models data scattered uniformly over a unit hypercube. The network tries to share its units (resources) more or less equally over the hypercube (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 cube. By simulations or by thinking it can be shown that the total number of units required to keep the average distance constant increases exponentially with the dimensionality of the cube.

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.

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?

Notation:

- a_j is the altitude 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 (MLP) 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. 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(f*log(a_j) - s_j^-2 * [(w_ij-x_i)^2])

ORBFEQ

Ordinary radial basis function (RBF) network with equal widths

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

NRBFUN

Normalized RBF network with unequal widths and heights

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

NRBFEV

Normalized RBF network with equal volumes

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

NRBFEH

Normalized RBF network with equal heights (and unequal widths)

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

NRBFEW

Normalized RBF network with equal widths (and unequal heights)

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

NRBFEQ

Normalized RBF network with equal widths and heights

h_j = softmax(- s^-2 * [(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 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.

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.
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 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 error function.

- Backward elimination begins with all candidate centers in the network. At each step the center is removed that least increases the error 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 error criterion 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 error criterion 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 error 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, 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, and it is quite unnecessary if you use an efficient training algorithm (see "What are conjugate gradients, Levenberg-Marquardt, etc.?")
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 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.

Subquestion: 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 or better.
Standardizing input variables also 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.

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)

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 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:

$$\text{midrange} = \frac{\max X_i + \min X_i}{2}$$

$$\text{range} = \max X_i - \min X_i$$

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

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.

Subquestion: 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 = value of the raw target variable Y for the ith training case

Z_i = 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:

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

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?

Subquestion: 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.

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.

Subquestion: 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 with trying. If an input has high kurtosis but low skewness, a tanh transform can reduce the influence of extreme values:

$$\tanh\left(c \frac{\text{input - mean}}{\text{stand. dev.}} \right)$$

where c is a constant that controls how far the extreme values are brought in towards the mean. 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 us still in the process of being written. As of 1996-11-27, the latest draft can be found via web browser at ftp://ftp.sas.com/pub/neural/importance.html, but this is a temporary URL, subject to change at any time.

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".

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 oft-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 the ARTMAP algorithms is that they have no mechanism to avoid overfitting and hence should not be used with noisy data.

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 different view of ART, see Sarle, W.S. (1995), "Why Statisticians Should Not FART," ftp://ftp.sas.com/pub/neural/fart.doc.

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.

Subject: What is PNN?

PNN or "Probabilistic Neural Network" is Donald Specht's term for kernel discriminant analysis. 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.

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.

McLachlan, G.J. (1992) *Discriminant Analysis and Statistical Pattern Recognition,* Wiley.

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.

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. 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.

GRN 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.

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". The main form of competitive learning in the NN literature is adaptive vector quantization (AVQ, also called a Kohonen network), of which Kosko (1992) provides a good description. In AVQ, each of the competitive hidden units corresponds to a cluster center, and the error function is the sum of squared 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 classes of cluster analysis (MacQueen 1967; Anderberg 1973; Balakrishnan, Cooper, Jacob, and Lewis 1994). The k-means model is a special case of the normal mixture model (McLachlan and Basford 1988), which has also been found to be very useful in neural networking (e.g., Bishop 1995).

Hebbian learning is the other most 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 (Kohonen 1995; Mulier and Cherkassky 1995). But the original SOM algorithm does not optimize an "energy" function (Kohonen 1995, pp. 126, 237) and so is not simply an information-compression method like most other unsupervised learning networks. 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 AVQ and no longer has topological ordering properties (Kohonen 1995, p. 111). A less heuristic and more principled approach to self-organizing maps is possible via Bayesian learning, using a mixture model with a prior distribution of mixture components that favors having the means of the mixture components fall near a smooth subspace; Utsugi (1996, 1997) has done some work in this direction.

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.

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.

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.

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), *Self-Organizing Maps,* Berlin: Springer-Verlag.

Kosko, B.(1992), *Neural Networks and Fuzzy Systems,* Englewood Cliffs, N.J.: Prentice-Hall.
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.
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.
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.
Utsugi, A. (1996), "Topology selection for self-organizing maps," Network: Computation in Neural Systems, 7, 727-740.
Utsugi, A. (1997), "Hyperparameter selection for self-organizing maps," Neural Computation, to appear.
Yin, H. and Allinson, N.M. (1995), "On the Distribution and Convergence of Feature Space in Self-Organizing Maps," Neural Computation, 7, 1178-1187.

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.
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, also has links to information on neuro-genetic methods.

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 essentially the same thing as bidirectional counterpropagation (Hecht-Nielsen 1990) and suffers from the same deficiencies. More recent work (Brown and Harris 1994) 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.

- 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.

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

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.

Next part is part 3 (of 7). Previous part is part 1.

PART 3

Archive-name: ai-faq/neural-nets/part3
Last-modified: 1997-07-29
URL: ftp://ftp.sas.com/pub/neural/FAQ3.html
Maintainer: saswss@unx.sas.com (Warren S. Sarle)
Copyright 1997 by Warren S. Sarle, Cary, NC, USA. Answers provided by other authors as cited below are copyrighted by those authors, who by submitting the answers for the FAQ give permission for the answer to be reproduced as part of the FAQ in any of the ways specified in part 1 of the FAQ.
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.
There are three conditions that are typically necessary (although not sufficient) for good generalization.

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.

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, Gy\"orfi, 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).

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., Gy\"orfi, 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.

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. (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.

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.

Subject: How does noise affect generalization?

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, 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 is exacerbated by overfitting (Moody 1992).
Noise in the inputs also 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.

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

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", NIPS 4, 847-854.

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

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

White, H. (1992), *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 (1993). 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 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).

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 can 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 five effective approaches to avoiding underfitting and overfitting, and hence getting good generalization:

- Model selection
- Jittering
- Weight decay
- Early stopping
- Bayesian learning

There 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.
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", NIPS 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)

Smith, M. (1993), *Neural Networks for Statistical Modeling,* NY: Van Nostrand Reinhold.

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 Holmstr\"om 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

$$\sum_{i,k} (x_{ij}+z_{ijk})^2 = \sum_{i,k} (x_{ij}^2 + z_{ijk}^2 + 2\, x_{ij}\, z_{ijk})$$

which is approximately, for c large,

$$c\left(\sum_i x_{ij}^2 + ns^2\right)$$

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

$$B(ns^2) = inv(c(X'X+ns^2 I))cX'Y = inv(X'X+ns^2 I)X'Y$$

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:

$$\begin{aligned}
&(Y;0-(X;dI)B)'(Y;0-(X;dI)B)\\
&= (Y;0)'(Y;0) - 2(Y;0)'(X;dI)B + B'(X;dI)'(X;dI)B\\
&= Y'Y - 2Y'XB + B'(X'X+d^2I)B\\
&= Y'Y - 2Y'XB + B'X'XB + B'(d^2I)B\\
&= (Y-XB)'(Y-XB)+d^2B'B
\end{aligned}$$

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:

$$s_{l+1}^2 = \frac{p(Y-XB(0))'(Y-XB(0))}{n(n-p)B(s_l^2)'B(s_l^2)}$$

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.

Holmstr\"om, L. and Koistinen, P. (1992) "Using additive noise in back-propagation training", IEEE Transaction on Neural Networks, 3, 24-38.

Koistinen, P. and Holmstr\"om, L. (1992) "Kernel regression and backpropagation training with noise," NIPS4, 1033-1039.

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 section 3.3 of Prechelt (1994).

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 function. 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 Sjo\"berg 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.

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)
Sjo\"berg, 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 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:

$$\sum_i \frac{(w_i)^2}{(w_i)^2 + c^2}$$

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.

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," Institute of Statistics and Decision Sciences Working Paper 95-31, available at 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).
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 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?

Some books and articles offer "rules of thumb" for choosing an architecture--Number_of_inputs plus Number_of_outputs divided by two, maybe with a square root in there somewhere--but such rules are total garbage. There is no way to determine a good network architecture just from the number of inputs and outputs. It depends critically on the number of training cases, the amount of noise, and the complexity of the function or classification you are trying to learn. There are problems with one input and one output that require thousands of hidden units, and problems with a thousand inputs and a thousand outputs that require only one hidden unit, or none at all.

Other rules relate to the number of cases available: use at most so many hidden units that the number of weights in the network times 10 is smaller than the number of cases. Such rules are only concerned with overfitting and are unreliable as well. All one can say is that 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. Geman, Bienenstock, and Doursat (1992) discuss how the number of hidden units affects the bias/variance trade-off.

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.

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. But Lawrence, Giles, and Tsoi (1996) have shown that standard online backprop can have considerable difficulty reducing training error to a level near the globally optimal value, hence 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. The FAQ maintainer obtained similar results 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 1995). 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.

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
Neal, R.M. (1995), *Bayesian Learning for Neural Networks,* Ph.D. thesis, University of Toronto, ftp://ftp.cs.toronto.edu/pub/radford/thesis.ps.Z.
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)
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, 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). AIC and FPE tend to overfit with NNs. Rissanen's Minimum Description Length principle (MDL; Rissanen 1978, 1987) is closely related to SBC. 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 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", NIPS 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

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.

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)

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).

In k-fold cross-validation, you divide the data into k subsets of 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. A more elaborate and expensive version of cross-validation involves leaving out all possible subsets of a given size.

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 error function, 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).

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).

Leave-one-out cross-validation often works well for continuous error functions such as the mean squared error, but it may perform poorly for noncontinuous 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 sample size between the full-sample analysis and the cross-validation analyses. A value of 10 for k is popular.

Leave-one-out cross-validation can also run into trouble with various model-selection methods, such as choosing a subset of the inputs or choosing the number of hidden units. The problem again 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 cross-validation to work better than leave-one-out.

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 200 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). Use of bootstrapping for NNs is described in Tibshirani (1996) and Masters (1995).

Cross-validation and bootstrapping become considerably more complicated for time series data; see Hjorth (1994) and Snijders (1988).

References (see also http://www.statistics.com/books.html):

Breiman, L. (1996), "Heuristics of instability and stabilization in model selection," Annals of Statistics, 24, 2350-2383.

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.

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.

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.

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.

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.

Next part is part 4 (of 7). Previous part is part 2.

PART 4

Archive-name: ai-faq/neural-nets/part4
Last-modified: 1997-08-12
URL: ftp://ftp.sas.com/pub/neural/FAQ4.html
Maintainer: saswss@unx.sas.com (Warren S. Sarle)
Copyright 1997 by Warren S. Sarle, Cary, NC, USA. Reviews provided by other authors as cited below are copyrighted by those authors, who by submitting the reviews for the FAQ give permission for the review to be reproduced as part of the FAQ in any of the ways specified in part 1 of the FAQ.
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 ==========

Subject: Books and articles about Neural Networks?

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 popular introduction to NNs
Hinton, G.E. (1992), "How Neural Networks Learn from Experience", Scientific American, 267 (September), 144-151.
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 on practical use of NNs

Smith, M. (1993). *Neural Networks for Statistical Modeling,* NY: Van Nostrand Reinhold.
Smith is not a statistician, but he tries. The book has entire brief chapters on overfitting and validation (early stopping and split-sample 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.
Weiss, S.M. and Kulikowski, C.A. (1991), *Computer Systems That Learn*, Morgan Kaufmann. ISBN 1 55860 065 5.
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.

The best elementary textbook on using and programming NNs

Masters, Timothy (1994). *Practical Neural Network Recipes in C++*, Academic Press, ISBN 0-12-479040-2, US $45 incl. disks.
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.

The best elementary textbooks on NN research

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.
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.

Anderson, J.A. (1995), *An Introduction to Neural Networks,* Cambridge,MA: The MIT Press, ISBN 0-262-01144-1.
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. Recommended for classroom use if the instructor provides supplementary material on how to get good generalization.

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.
This is definitely the best book on neural nets for practical applications for readers comfortable with calculus. 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."
Hertz, J., Krogh, A., and Palmer, R. (1991). *Introduction to the Theory of Neural Computation.* Addison-Wesley: Redwood City, California. ISBN 0-201-50395-6 (hardbound) and 0-201-51560-1 (paperbound)
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.

Brian Ripley's new 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.

Devroye, L., Gy\"orfi, 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.

The best books on image and signal processing with NNs

Masters, T. (1994), *Signal and Image Processing with Neural Networks: A C++ Sourcebook*, NY: Wiley.

Cichocki, A. and Unbehauen, R. (1993). *Neural Networks for Optimization and Signal Processing.* NY: John Wiley & Sons, ISBN 0-471-930105 (hardbound), 526 pages, $57.95.

Comments from readers of comp.ai.neural-nets:"Partly a textbook and partly a research monograph; introduces the basic concepts, techniques, and models related to neural networks and optimization, excluding rigorous mathematical details. Accessible to a wide readership with a differential calculus background. The main coverage of the book is on recurrent neural networks with continuous state variables. The book title would be more appropriate without mentioning signal processing. Well edited, good illustrations."

The best book on time-series forecasting with NNs

Weigend, A.S. and Gershenfeld, N.A., eds. (1994) *Time Series Prediction: Forecasting the Future and Understanding the Past*, Addison-Wesley: Reading, MA.

The best book on neurofuzzy systems

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

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. This book is a welcome antidote to the quasi-mystical approach of Kosko.

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.

Books for the Beginner

Aleksander, I. and Morton, H. (1990). *An Introduction to Neural Computing.* Chapman and Hall. (ISBN 0-412-37780-2).
Comments from readers of comp.ai.neural-nets:: "This book seems to be intended for the first year of university education."
Beale, R. and Jackson, T. (1990). *Neural Computing, an Introduction.* Adam Hilger, IOP Publishing Ltd : Bristol. (ISBN 0-85274-262-2).
Comments from readers of comp.ai.neural-nets: "It's clearly written. Lots of hints as to how to get the adaptive models covered to work (not always well explained in the original sources). Consistent mathematical terminology. Covers perceptrons, error-backpropagation, Kohonen self-org model, Hopfield type models, ART, and associative memories."
Caudill, M. and Butler, C. (1990). *Naturally Intelligent Systems.* MIT Press: Cambridge, Massachusetts. (ISBN 0-262-03156-6).
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.
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. 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.
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.
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
Hecht-Nielsen, R. (1990). *Neurocomputing.* Addison Wesley.
Comments from readers of comp.ai.neural-nets: "A good book", "comprises a nice historical overview and a chapter about NN hardware. Well structured prose. Makes important concepts clear."
McClelland, J. L. and Rumelhart, D. E. (1988). *Explorations in Parallel Distributed Processing: Computational Models of Cognition and Perception* (software manual). The MIT Press.
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).
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.)
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 Physiscs 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 & 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, but 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)
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."

The Classics

Kohonen, T. (1984). *Self-organization and Associative Memory.* Springer-Verlag: New York. (2nd Edition: 1988; 3rd edition: 1989).
Comments from readers of comp.ai.neural-nets: "The section on Pattern mathematics is excellent."
Rumelhart, D. E. and McClelland, J. L. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (volumes 1 & 2). The MIT Press.
Comments from readers of comp.ai.neural-nets: "As a computer scientist I found the two Rumelhart and McClelland books really heavy going and definitely not the sort of thing to read if you are a beginner."; "It's quite readable, and affordable (about $65 for both volumes)."; "THE Connectionist bible".

Introductory Journal Articles

Hinton, G. E. (1989). Connectionist learning procedures. Artificial Intelligence, Vol. 40, pp. 185--234.
Comments from readers of comp.ai.neural-nets: "One of the better neural networks overview papers, although the distinction between network topology and learning algorithm is not always very clear. Could very well be used as an introduction to neural networks."
Knight, K. (1990). Connectionist, Ideas and Algorithms. Communications of the ACM. November 1990. Vol.33 nr.11, pp 59-74.
Comments from readers of comp.ai.neural-nets:"A good article, while it is for most people easy to find a copy of this journal."
Kohonen, T. (1988). An Introduction to Neural Computing. Neural Networks, vol. 1, no. 1. pp. 3-16.
Comments from readers of comp.ai.neural-nets: "A general review".

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, vol 323 (9 October), pp. 533-536.
Comments from readers of comp.ai.neural-nets: "Gives a very good potted explanation of backprop NN's. It gives sufficient detail to write your own NN simulation."

Not-quite-so-introductory Literature

Anderson, J. A. and Rosenfeld, E. (Eds). (1988). *Neurocomputing: Foundations of Research.* The MIT Press: Cambridge, MA.
Comments from readers of comp.ai.neural-nets: "An expensive book, but excellent for reference. It is a collection of reprints of most of the major papers in the field."
Anderson, J. A., Pellionisz, A. and Rosenfeld, E. (Eds). (1990). *Neurocomputing 2: Directions for Research.* The MIT Press: Cambridge, MA.
Comments from readers of comp.ai.neural-nets: "The sequel to their well-known Neurocomputing book."
Bourlard, H.A., and Morgan, N. (1994), *Connectionist Speech Recognition: A Hybrid Approach,* Boston: Kluwer Academic Publishers.
Deco, G. and Obradovic, D. (1996), *An Information-Theoretic Approach to Neural Computing,* NY: Springer-Verlag.
Haykin, S. (1994). *Neural Networks, a Comprehensive Foundation.* Macmillan, New York, NY.
Comments from readers of comp.ai.neural-nets: "A very readable, well written intermediate text on NNs Perspective is primarily one of pattern recognition, estimation and signal processing. However, there are well-written chapters on neurodynamics and VLSI implementation. Though there is emphasis on formal mathematical models of NNs as universal approximators, statistical estimators, etc., there are also examples of NNs used in practical applications. The problem sets at the end of each chapter nicely complement the material. In the bibliography are over 1000 references."
Khanna, T. (1990). *Foundations of Neural Networks.* Addison-Wesley: New York.
Comments from readers of comp.ai.neural-nets: "Not so bad (with a page of erroneous formulas (if I remember well), and #hidden layers isn't well described)."; "Khanna's intention in writing his book with math analysis should be commended but he made several mistakes in the math part".
Kung, S.Y. (1993). *Digital Neural Networks,* Prentice Hall, Englewood Cliffs, NJ.
Levine, D. S. (1990). *Introduction to Neural and Cognitive Modeling.* Lawrence Erlbaum: Hillsdale, N.J.
Comments from readers of comp.ai.neural-nets: "Highly recommended".
Lippmann, R. P. (April 1987). An introduction to computing with neural nets. IEEE Acoustics, Speech, and Signal Processing Magazine. vol. 2, no. 4, pp 4-22.
Comments from readers of comp.ai.neural-nets: "Much acclaimed as an overview of neural networks, but rather inaccurate on several points. The categorization into binary and continuous- valued input neural networks is rather arbitrary, and may work confusing for the unexperienced reader. Not all networks discussed are of equal importance."
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".
Masters, T. (1995) *Advanced Algorithms for Neural Networks: A C++ Sourcebook,* NY: John Wiley and Sons, ISBN 0-471-10588-0
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.

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

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.

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++*, Wiley.
Welstead, Stephen T. (1994), *Neural Network and Fuzzy Logic Applications in C/C++*, 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.

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
WWW: 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://www-mitpress.mit.edu/jrnls-catalog/neural.html
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: 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)
WWW: http://www.elsevier.nl/locate/inca/505628

Title: Neural Processing Letters
Publish: D facto publications
Address: 45 rue Masui; B-1210 Brussels, Belgium
Phone: (32) 2 245 43 63; Fax: (32) 2 245 46 94
Freq: 6 issues/year (vol. 1 in September 1994)
Cost/Yr: BEF 4400 (about $140)
ISSN #: 1370-4621
URL: http://www.dice.ucl.ac.be/neural-nets/NPL/NPL.html
FTP: ftp://ftp.dice.ucl.ac.be/pub/neural-nets/NPL
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
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: Complex Systems
Publish: Complex Systems Publications
Address: Complex Systems Publications, Inc., P.O. Box 6149, Champaign, IL 61821-8149, USA
Freq.: 6 times per year (1st volume is 1987)
ISSN #: 0891-2513
Cost/Yr: Individual $75, Institution $225
Remark: Journal COMPLEX SYSTEMS devotes to rapid publication of research on science, mathematics, and engineering of systems with simple

components but complex overall behavior. Send mail to "jcs@jaguar.ccsr.uiuc.edu" for additional info. (Remark is from announcement on Net)

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: (Expensive as hell, I'm sure.) 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. I'm reasonably sure they are still at it--I think I saw them make a call for reviewers a few months ago. 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. That's probably more than I really know about it but maybe if you post it someone who knows more about it will correct any errors I have made. The main thing is that I liked the articles I read. (Note: remarks by Don Wunsch)

Title: International Journal of Applied Intelligence

Publish: Kluwer Academic Publishers
Remark: first issue in 1990(?)

Title: Bulletin of Mathematical Biology

Title: Intelligence

Title: Journal of Mathematical Biology

Title: Journal of Complex System

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

C. Journals loosely related to NNs:

Title: JOURNAL OF COMPLEXITY
Remark: (Must rank alongside Wolfram's Complex Systems)

Title: IEEE ASSP Magazine
Remark: (April 1987 had the Lippmann intro. which everyone likes to cite)

Title: ARTIFICIAL INTELLIGENCE
Remark: (Vol 40, September 1989 had the survey paper by Hinton)

Title: COGNITIVE SCIENCE
Remark: (the Boltzmann machine paper by Ackley et al appeared here
in Vol 9, 1983)

Title: COGNITION
Remark: (Vol 28, March 1988 contained the Fodor and Pylyshyn
critique of connectionism)

Title: COGNITIVE PSYCHOLOGY
Remark: (no comment!)

Title: JOURNAL OF MATHEMATICAL PSYCHOLOGY
Remark: (several good book reviews)

Subject: The most important conferences concerned with Neural Networks?

[to be added: has taken place how often yet; most emphasized topics;
where to get proceedings/calls-for-papers etc.]

A. Dedicated Neural Network Conferences:

1. Neural Information Processing Systems (NIPS) Annually since 1988 in Denver, Colorado; late November or early December. Interdisciplinary conference with computer science, physics, engineering, biology, medicine, cognitive science topics. Covers all aspects of NNs. Proceedings appear several months after the conference as a book from Morgan Kaufman, San Mateo, CA.
2. International Conference on Neural Networks (ICNN) sponsored by IEEE-NNC and INNS "in the spirit of earlier IJCNN's," http://www.mindspring.com/~pci-inc/ICNN97/
3. Annual Conference on Neural Networks (ACNN)
4. International Conference on Artificial Neural Networks (ICANN) Annually in Europe. First was 1991. Major conference of European Neur. Netw. Soc. (ENNS)
5. European Symposium on Artificial Neural Networks (ESANN). Anually since 1993 in Brussels, Belgium; late April; conference on the fundamental aspects of artificial neural networks: theory, mathematics, biology, relations between neural networks and other disciplines, statistics, learning, algorithms, models and architectures, self-organization, signal processing, approximation of functions, evolutive learning, etc. Contact: Michel Verleysen, D facto conference services, 45 rue Masui, B-1210 Brussels, Belgium, phone: +32 2 245 43 63, fax: + 32 2 245 46 94, e-mail: esann@dice.ucl.ac.be
6. Artificial Neural Networks in Engineering (ANNIE) Anually since 1991 in St. Louis, Missouri; held in November. (Topics: NN architectures, pattern recognition, neuro-control, neuro-engineering systems. Contact: ANNIE; Engineering Management Department; 223 Engineering Management Building; University of Missouri-Rolla; Rolla, MO 65401; FAX: (314) 341-6567)
7. International Joint Conference on Neural Networks (IJCNN) formerly co-sponsored by INNS and IEEE, no longer held.
8. WCNN. Sponsored by INNS, subsumed into ICNN at their final meeting in August, 1996, and will no longer be held..
9. many many more....

B. Other Conferences

1. International Joint Conference on Artificial Intelligence (IJCAI)
2. Intern. Conf. on Acustics, Speech and Signal Processing (ICASSP)
3. Intern. Conf. on Pattern Recognition. Held every other year. Has a connectionist subconference. Information: General Chair Walter G. Kropatsch <krw@prip.tuwien.ac.at>
4. Annual Conference of the Cognitive Science Society
5. IEEE International Conference on Evolutionary Computation (ICEC) EPS Evolutionary Programming Conference (EP)
   http://www.engr.iupui.edu/et/ieee_con.htm
   Sponsored in part by the IEEE Neural Networks Council (NNC)
6. International Conference on Genetic Algorithms (ICGA)
   http://isl.cps.msu.edu/GA/icga97/
7. [Vision Conferences?]

C. Pointers to Conferences

1. The journal "Neural Networks" has a list of conferences, workshops and meetings in each issue. This is quite interdisciplinary.
2. There is a regular posting on comp.ai.neural-nets from Paultje Bakker: "Upcoming Neural Network Conferences", which lists names, dates, locations, contacts, and deadlines. It is also available on the WWW from http://www.neuronet.ph.kcl.ac.uk/neuronet/bakker.html.
3. The IEEE Neural Network Council maintains an up-to-date list of conferences at http://www.ieee.org/nnc.

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/docs/cie/neural/societies.html and at
http://www.ieee.org:80/nnc/research/othernnsoc.html.

Subject: On-line and machine-readable information about NNs?

1. **Introductory Web Pages**

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

   One of the best introductory sources is Donald Tveter's Backpropagator's Review at http://www.mcs.com/~drt/bprefs.html, which contains both answers to additional FAQs and an annotated neural net bibliography emphasizing on-line articles.

   More introductory material is available on line from the "DACS Technical Report Summary: Artificial Neural Networks Technology" at
   http://www.utica.kaman.com/techs/neural/neural.html
   and "ARTIFICIAL INTELLIGENCE FOR THE BEGINNER" at
   http://home.clara.net/mll/ai/

2. **Neuron Digest**

   Internet Mailing List. From the welcome blurb: "Neuron-Digest is a list (in digest form) dealing with all aspects of neural networks (and any type of network or neuromorphic system)" To subscribe, send email to neuron-request@psych.upenn.edu. The ftp archives (including back issues) are available from psych.upenn.edu in pub/Neuron-Digest or by sending email to "archive-server@psych.upenn.edu". comp.ai.neural-net readers also find the messages in that newsgroup in the form of digests.

3. **Usenet groups comp.ai.neural-nets (Oha!) and comp.theory.self-org-sys.**

There is a periodic posting on comp.ai.neural-nets sent by srctran@world.std.com (Gregory Aharonian) about Neural Network patents.

4. **Central Neural System Electronic Bulletin Board**

   Modem: 409-737-5222; Sysop: Wesley R. Elsberry; 4160 Pirates' Beach, Galveston, TX 77554; welsberr@orca.tamu.edu. 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 etexts. No user fees have ever been charged. Home of the NEURAL_NET Echo, available thrugh FidoNet, RBBS-Net, and other EchoMail compatible bulletin board systems.

5. **Neuroprose ftp archive site: archive.cis.ohio-state.edu**

   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.

6. **Neural ftp archive site: ftp.funet.fi**

   Is administrating a large collection of neural network papers and software at the Finnish University Network file archive site ftp.funet.fi in directory /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

7. **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/.

8. **USENET newsgroup comp.org.issnnet**

   Forum for discussion of academic/student-related issues in NNs, as well as information on ISSNNet (see question "associations") and its activities.

9. **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)

10. 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.

11. 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.neuronet.ph.kcl.ac.uk (NEuroNet, King's College, London),
http://www.eeb.ele.tue.nl (Eindhoven, Netherlands),
http://www.emsl.pnl.gov:2080/docs/cie/neural/ (Richland, Washington),
http://www.cosy.sbg.ac.at/~rschwaig/rschwaig/projects.html (Salzburg, Austria),
http://http2.sils.umich.edu/Public/nirg/nirg1.html (Michigan),
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.
http://www.acsiom.org/nsr/neuro.htmlNeuroscience Web Search.
Many others are available too; WWW is changing all the time.

12. 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.

13. 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.

14. **INTCON mailing list**

    INTCON (Intelligent Control) is a moderated mailing list set up to provide a forum for communication and exchange of ideas among researchers in neuro-control, fuzzy logic control, reinforcement learning and other related subjects grouped under the topic of intelligent control. Send your subscribe requests to intcon-request@phoenix.ee.unsw.edu.au

---

Subject: Databases for experimentation with NNs?

1. **UCI machine learning database**

   A large collection of data sets accessible via anonymous FTP at 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. **The neural-bench Benchmark collection**

   Accessible WWW 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.

3. **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 [129.13.10.90] as /pub/neuron/proben.tar.gz. The file is about 1.8 MB and unpacks into about 20 MB.

4. **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

5. **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)
  - 5632 city words
  - 4938 state words
  - 9454 ZIP Codes
+ 300 ppi binary handwritten characters and digits:
  - 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 dictionaries 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 recognition 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.

6. **AI-CD-ROM (see question "Other sources of information")**
7. **Time series archive**

Various datasets of time series (to be used for prediction learning problems) are available for anonymous ftp from ftp.santafe.edu [192.12.12.1] in /pub/Time-Series". Problems are for example: 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).

8. **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 ftp://src.doc.ic.ac.uk/pub/packages/faces/README Do NOT do a directory listing in the top directory of the face archive, as it contains over 2500 entries!

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.

9. Lloyd Lubet's Financial, Business, and Economic Data Warehouse

The database consists of 23 tables containing 896 Monthly Indicators, Indexes, and Statistics for over 21 years. 7 tables deal directly with historic stock prices and market indexes. You can download this information either as a zip or ascii text file. For easy access, the database includes a table of contents and highly descriptive column headers.

Also available: White Paper #1: Avoiding the Pitfalls of high-tech statistics. This is a beginner's guide completely illustrated with some nice visualizations. It is accompanied by an 8 page notebook.

URL: http://www.rt66.com/~llubet (Copyright 1996 Accelerated Systems, Inc.)
Email: llubet@rt66.com

10. 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.

PART 5

Archive-name: ai-faq/neural-nets/part5
Last-modified: 1997-09-10
URL: ftp://ftp.sas.com/pub/neural/FAQ5.html
Maintainer: saswss@unx.sas.com (Warren S. Sarle)
The copyright for the description of each product is held by the producer or distributor of the product or whoever it was who supplied the description for the FAQ, who by submitting it for the FAQ gives permission for the description to be reproduced as part of the FAQ in any of the ways specified in part 1 of the FAQ.
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: 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 software 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 most 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. Rochester Connectionist Simulator
2. UCLA-SFINX
3. NeurDS
4. PlaNet (formerly known as SunNet)
5. GENESIS
6. Mactivation
7. Cascade Correlation Simulator

8. Quickprop
9. DartNet
10. SNNS
11. Aspirin/MIGRAINES
12. Adaptive Logic Network Educational Kit
13. NeuralShell
14. PDP++
15. Uts (Xerion, the sequel)
16. Neocognitron simulator
17. Multi-Module Neural Computing Environment (MUME)
18. LVQ_PAK, SOM_PAK
19. Nevada Backpropagation (NevProp)
20. Fuzzy ARTmap
21. PYGMALION
22. Basis-of-AI-NN Software
23. Matrix Backpropagation
24. WinNN
25. BIOSIM
26. The Brain
27. FuNeGen
28. NeuDL -- Neural-Network Description Language
29. NeoC Explorer
30. AINET
31. DemoGNG
32. PMNEURO 1.0a
33. nn/xnn
34. NNDT
35. Trajan 2.1 Shareware
36. Neural Networks at your Fingertips
37. NNFit
38. Nenet v1.0

See also http://www.emsl.pnl.gov:2080/docs/cie/neural/systems/shareware.html

1. **Rochester Connectionist Simulator**

   A quite 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.cs.rochester.edu in directory pub/packages/simulator as the files README (8 KB), and rcs_v4.2.tar.Z (2.9 MB),

2. **UCLA-SFINX**

   ftp retina.cs.ucla.edu [131.179.16.6]; Login name: sfinxftp; Password: joshua; directory: pub; files : README; sfinx_v2.0.tar.Z; Email info request : sfinx@retina.cs.ucla.edu

3. **NeurDS**

simulator for DEC systems supporting VT100 terminal. available for anonymous ftp from gatekeeper.dec.com [16.1.0.2] in directory: pub/DEC as the file NeurDS031.tar.Z (111 Kb)

4. **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.Z (800 kb) or from boulder.colorado.edu [128.138.240.1] as /pub/generic-sources/PlaNet5.7.tar.Z

5. **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. Available by ftp from ftp://genesis.bbb.caltech.edu/pub/genesis. Further information via WWW at http://www.bbb.caltech.edu/GENESIS/.

6. **Mactivation**

   A neural network simulator for the Apple Macintosh. Available for ftp from ftp.cs.colorado.edu [128.138.243.151] as /pub/cs/misc/Mactivation-3.3.sea.hqx

7. **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).

8. **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.)

9. **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).

10. SNNS 4.1

"Stuttgart Neural Network Simulator" from the University of Stuttgart, Germany. A luxurious simulator for many types of nets; with X11 interface: Graphical 2D and 3D topology editor/visualizer, training visualisation, multiple pattern set handling etc. Currently supports backpropagation (vanilla, online, with momentum term and flat spot elimination, batch, time delay), counterpropagation, quickprop, backpercolation 1, generalized radial basis functions (RBF), RProp, ART1, ART2, ARTMAP, Cascade Correlation, Recurrent Cascade Correlation, Dynamic LVQ, Backpropagation through time (for recurrent networks), batch backpropagation through time (for recurrent networks), Quickpropagation through time (for recurrent networks), Hopfield networks, Jordan and Elman networks, autoassociative memory, self-organizing maps, time-delay networks (TDNN), RBF_DDA, simulated annealing, Monte Carlo, Pruned Cascade-Correlation, Optimal Brain Damage, Optimal Brain Surgeon, Skeletonization, and is user-extendable (user-defined activation functions, output functions, site functions, learning procedures). C code generator snns2c. Works on SunOS, Solaris, IRIX, Ultrix, OSF, AIX, HP/UX, NextStep, and Linux. Distributed kernel can spread one learning run over a workstation cluster. Available for ftp from ftp.informatik.uni-stuttgart.de [129.69.211.2] in directory /pub/SNNS as SNNSv4.1.tar.gz (1.4 MB, Source code) and SNNSv4.1.Manual.ps.gz (1 MB, Documentation). There are also various other files in this directory (e.g. the source version of the manual, a Sun Sparc executable, older versions of the software, some papers, an implementation manual, and the software in several smaller parts). It may be best to first have a look at the file SNNSv4.1.Readme. This file contains a somewhat more elaborate short description of the simulator. More information can be found in the WWW under http://www.informatik.uni-stuttgart.de/ipvr/bv/projekte/snns/snns.html

11. 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 two FTP sites: from CMU's simulator collection on pt.cs.cmu.edu [128.2.254.155] in /afs/cs/project/connect/code/unsupported/am6.tar.Z and from UCLA's cognitive

science machine ftp.cognet.ucla.edu [128.97.50.19] in /pub/alexis/am6.tar.Z (2 MB).

12. Adaptive Logic Network Educational Kit (for Windows)

The Atree 3.0 Educational Kit (EK) serves to develop simple applications using adaptive Logic Networks (ALNs). In an ALN, logic functions AND and OR make up all hidden layers but the first, which uses simple perceptrons. Though this net can't *compute* real-valued outputs, since its outputs are strictly boolean, it can easily and naturally *represent* real valued functions by giving a 0 above the function's graph and a 1 otherwise. This approach is extremely useful, since it allows the user to impose constraints on the functions to be learned (monotonicity, bounds on slopes, convexity,...). Very rapid computation of functions is done by an *ALN decision tree* at whose leaves are small expressions of minimum and maximum operations acting on linear functions.

Two simple languages describe ALNs and the steps of training an ALN. Execution software for ALN decision trees resulting from training is provided in C source form for experimenters. EK and a "120-page" User's Guide are obtained by anonymous ftp from ftp.cs.ualberta.ca in directory /pub/atree/atree3/. Get the file atree3ek.exe (~900K) .

The above User's Guide with an introduction to basic ALN theory is available on WWW at http://www.cs.ualberta.ca/~arms/guide/ch0.htm . This Educational Kit software is the same as the commercial Atree 3.0 program except that it allows only two input variables and is licensed for educational uses only. A built-in 2D and 3D plotting capability is useful to help the user understand how ALNs work.

13. NeuralShell

Formerly available from FTP site quanta.eng.ohio-state.edu [128.146.35.1] as /pub/NeuralShell/NeuralShell.tar". Currently (April 94) not available and undergoing a major reconstruction. Not to be confused with NeuroShell by Ward System Group (see below under commercial software).

14. PDP++

The PDP++ software is a new 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.
The current version is 1.0, our first non-beta release. It has been extensively tested and should be completely usable. Works on Unix with X-Windows.
Features: Full GUI (InterViews), realtime network viewer, data viewer, extendable object-oriented design, CSS scripting language with source-level debugger, GUI macro recording.
Algorithms: Feedforward and several recurrent BP, Boltzmann machine, Hopfield, Mean-field, Interactive activation and competition, continuous stochastic networks.
The software can be obtained by anonymous ftp from

ftp://cnbc.cmu.edu/pub/pdp++/ and from ftp://unix.hensa.ac.uk/mirrors/pdp++/.
For more information, see our WWW page at
http://www.cnbc.cmu.edu/PDP++/PDP++.html.
There is a 250 page (printed) manual and an HTML version available on-line at the above address.

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. Neocognitron simulator

The simulator is written in C and comes with a list of references which are necessary to read to understand the specifics of the implementation. The unsupervised version is coded without (!) C-cell inhibition. Available for anonymous ftp from unix.hensa.ac.uk [129.12.21.7] in /pub/neocognitron.tar.Z (130 kB).

17. 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. The modules are provided in a library. Several "front-ends" or clients are also available. X-Window support by editor/visualization tool Xmume. MUME can be used to include non-neural computing modules (decision trees, ...) in applications. MUME is available for educational institutions by anonymous ftp on mickey.sedal.su.oz.au [129.78.24.170] after signing and sending a licence: /pub/license.ps (67 kb). Contact: Marwan Jabri, SEDAL, Sydney University Electrical Engineering, NSW 2006 Australia, marwan@sedal.su.oz.au

18. 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

19. Nevada Backpropagation (NevProp)

NevProp is a free, easy-to-use feedforward backpropagation (multilayer perceptron) program. It uses an interactive character-based interface, and is distributed as C source code that should compile and run on most platforms. (Precompiled executables are available for Macintosh and DOS.) The original version was Quickprop 1.0 by Scott Fahlman, as translated from Common Lisp by Terry Regier. We added early-stopped training based on a held-out subset of data, c index (ROC curve area) calculation, the ability to force gradient descent (per-epoch or per-pattern), and additional options. FEATURES (NevProp version 1.16): UNLIMITED (except by machine memory) number of input PATTERNS; UNLIMITED number of input, hidden, and output UNITS; Arbitrary CONNECTIONS among the various layers' units; Clock-time or user-specified RANDOM SEED for initial random weights; Choice of regular GRADIENT DESCENT or QUICKPROP; Choice of PER-EPOCH or PER-PATTERN (stochastic) weight updating; GENERALIZATION to a test dataset; AUTOMATICALLY STOPPED TRAINING based on generalization; RETENTION of best-generalizing weights and predictions; Simple but useful GRAPHIC display to show smoothness of generalization; SAVING of results to a file while working interactively; SAVING of weights file and reloading for continued training; PREDICTION-only on datasets by applying an existing weights file; In addition to RMS error, the concordance, or c index is displayed. The c index (area under the ROC curve) shows the correctness of the RELATIVE ordering of predictions AMONG the cases; ie, it is a measure of discriminative power of the model. AVAILABILITY: The most updated version of NevProp will be made available by anonymous ftp from the University of Nevada, Reno: On ftp.scs.unr.edu [134.197.10.130] in the directory "pub/goodman/nevpropdir", e.g. README.FIRST (45 kb) or nevprop1.16.shar (138 kb). Version 2 (not yet released) is intended to have some new features: more flexible file formatting (including access to external data files; option to prerandomize data order; randomized stochastic gradient descent; option to rescale predictor (input) variables); linear output units as an alternative to sigmoidal units for use with continuous-valued dependent variables (output targets); cross-entropy (maximum likelihood) criterion function as an alternative to square error for use with categorical dependent variables (classification/symbolic/nominal targets); and interactive interrupt to change settings on-the-fly. Limited support is available from Phil Goodman (goodman@unr.edu), University of Nevada Center for Biomedical Research.

20. 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).

21. 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).

22. Basis-of-AI-NN Software

DOS and UNIX C source code, examples and DOS binaries are available in the following different program sets:

[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: Basis of AI NN software at http://www.mcs.com/~drt/svbp.html .

An improved professional version of backprop is also available, $30 for regular people, $200 for businesses and governmental agencies. See: Basis of AI Professional Backprop at http://www.mcs.com/~drt/probp.html .

Questions to: Don Tveter, drt@mcs.com

23. 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 risc6000.dibe.unige.it [130.251.89.154] as /pub/MBPv1.1.tar.Z (Unix version), /pub/MBPv11.zip.Z (MS-DOS version), /pub/mpbv11.ps (Documentation). For more information, contact Davide Anguita (anguita@dibe.unige.it).

24. WinNN

WinNN is a shareware Neural Networks (NN) package for windows 3.1. WinNN incorporates a very user friendly interface with a powerful computational engine. WinNN is intended to be used as a tool for beginners and more advanced neural networks users, it provides an alternative to using more expensive and hard to use packages. WinNN can implement feed forward multi-layered NN and uses a modified fast back-propagation for training. Extensive on line help. Has various neuron functions. Allows on the fly testing of the network performance and generalization. All training parameters can be easily modified while WinNN is training. Results can be saved on disk or copied to the clipboard. Supports plotting of the outputs and weight distribution. Available for ftp from ftp.cc.monash.edu.au as /pub/win3/programr/winnn97.zip (747 kB).

25. 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

26. The Brain

The Brain is an advanced neural network simulator for PCs that is simple enough to be used by non-technical people, yet sophisticated enough for serious research work. It is based upon the backpropagation learning algorithm. Three sample networks are included. The documentation included provides you with an introduction and overview of the concepts and applications of neural networks as well as outlining the features and capabilities of The Brain. The Brain requires 512K memory and MS-DOS or PC-DOS version 3.20 or later (versions for other OS's and machines are available). A 386 (with maths coprocessor) or higher is recommended for serious use of The Brain. Shareware payment required. Demo version is restricted to number of units the network can handle due to memory contraints on PC's. Registered version allows use of extra memory. External documentation included: 39Kb, 20 Pages. Source included: No (Source comes with registration). Available via anonymous ftp from ftp.tu-clausthal.de as /pub/msdos/science/brain12.zip (78 kb) and from ftp.technion.ac.il as /pub/contrib/dos/brain12.zip (78 kb) Contact: David Perkovic; DP Computing; PO Box 712; Noarlunga Center SA 5168; Australia; Email: dip@mod.dsto.gov.au (preferred) or dpc@mep.com or perkovic@cleese.apana.org.au

27. FuNeGen 1.0

FuNeGen is a MLP based software program to generate fuzzy rule based classifiers. A limited version (maximum of 7 inputs and 3 membership functions for each input) for PCs is available for anonymous ftp from obelix.microelectronic.e-technik.th-darmstadt.de in directory /pub/neurofuzzy. For further information see the file read.me. Contact: Saman K. Halgamuge

28. 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. NeuDL is available from the anonymous ftp site at The University of Alabama: cs.ua.edu (130.160.44.1) in the file /pub/neudl/NeuDLver021.tar. The tarred file contains the interpreter source code (in C++) a user manual, a paper about NeuDL, and about 25 sample NeuDL programs. A document demonstrating NeuDL's capabilities is also available from the ftp site: /pub/neudl/NeuDL/demo.doc /pub/neudl/demo.doc. For more information contact the author: Joey Rogers (jrogers@buster.eng.ua.edu).

29. 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. Available for anonymous FTP from OAK.Oakland.Edu (141.210.10.117) as /SimTel/msdos/neurlnet/neocog10.zip (193 kB, DOS version)

30. AINET

aiNet is a shareware Neural Networks (NN) application for MS-Windows 3.1. It does not require learning, has no limits in parameters (input & output neurons), no limits in sample size. It is not sensitive toward noise in the data. Database can be changed dynamically. It provides a way to estimate the rate of error in your prediction. Missing values are handled automatically. It has graphical spreadsheet-like user interface and on-line help system. It provides also several different charts types. aiNet manual (90 pages) is divided into: "User's Guide", "Basics About Modeling with the AINET", "Examples". Special requirements: Windows 3.1, VGA or better. Can be downloaded from ftp://ftp.cica.indiana.edu/pub/pc/win3/programr/ainet100.zip or from ftp://oak.oakland.edu/SimTel/win3/math/ainet100.zip

31. 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.

32. PMNEURO 1.0a
33. PMNEURO 1.0a is available at:
34.
35. ftp://ftp.uni-stuttgart.de/pub/systems/os2/misc/pmneuro.zip
36.
37. PMNEURO 1.0a creates neuronal networks (backpropagation); propagation
38. results can be used as new training input for creating new networks and
39. following propagation trials.
40. nn/xnn
41. Name: nn/xnn
42. Company: Neureka ANS

Address: Klaus Hansens vei 31B
5037 Solheimsviken
NORWAY
Phone: +47 55 20 15 48
Email: neureka@bgif.no
URL: http://www.bgif.no/neureka/
Operating systems:
nn: UNIX or MS-DOS,
xnn: UNIX/X-windows, UNIX flavours: OSF1, Solaris, AIX, IRIX, Linux (1.2.13)
System requirements: Min. 20 Mb HD + 4 Mb RAM available. If only the nn/netpack part is used (i.e. not the GUI), much less is needed.
Approx. price: Free for 30 days after installation, fully functional
After 30 days: USD 250,-
35% educational discount.

Basic capabilities:

A comprehensive shareware system for developing and simulating artificial neural networks. You can download the software from the URL given above.

nn is a high-level neural network specification language. The current version is best suited for feed-forward nets, but recurrent models can and have been implemented as well. The nn compiler can generate C code or executable programs, with a powerful command line interface, but everything may also be controlled via the graphical interface (xnn). It is possible for the user to write C routines that can be called from inside the nn specification, and to use the nn specification as a function that is called from a C program. These features makes nn well suited for application development. Please note that no programming is necessary in order to use the network models that come with the system (netpack).

xnn is a graphical front end to networks generated by the nn compiler, and to the compiler itself. The xnn graphical interface is intuitive and easy to use for beginners, yet powerful, with many possibilities for visualizing network data. Data may be visualized during training, testing or 'off-line'.

netpack: A number of networks have already been implemented in nn and can be used directly: MAdaline, ART1, Backpropagation, Counterpropagation, Elman, GRNN, Hopfield, Jordan, LVQ, Perceptron, RBFNN, SOFM (Kohonen). Several others are currently being developed.

The pattern files used by the networks, have a simple and flexible format, and can easily be generated from other kinds of data. The data file generated by the network, can be saved in ASCII or binary format. Functions for converting and pre-processing data are available.

92. NNDT

NNDT

Neural Network Development Tool
Evaluation version 1.4
Bjvrn Saxen
1995

http://www.abo.fi/~bjsaxen/nndt.html
ftp://ftp.abo.fi/pub/vt/bjs/

The NNDT software is as a tool for neural network training. The user interface is developed with MS Visual Basic 3.0 professional edition. DLL routines (written in C) are used for most of the mathematics. The program can be run on a personal computer with MS Windows, version 3.1.

Evaluation version

This evaluation version of NNDT may be used free of charge for personal and educational use. The software certainly contains limitations and bugs, but is still a working version which has been developed for over one year. Comments, bug reports and suggestions for improvements can be sent to:

bjorn.saxen@abo.fi
or
Bjorn Saxen
Heat Engineering Laboratory
Abo Akademi University
Biskopsgatan 8
SF-20500 Abo
Finland

Remember, this program comes free but with no guarantee!

A user's guide for NNDT is delivered in PostScript format. The document is split into three parts and compressed into a file called MANUAL.ZIP. Due to many bitmap figures included, the total size of the uncompressed files is very large, approx 1.5M.

Features and methods

The network algorithms implemented are of the so called supervised type. So far, algorithms for multi-layer perceptron (MLP) networks of feed-forward and recurrent types are included. The MLP networks are trained with the Levenberg-Marquardt method.

The training requires a set of input signals and corresponding output signals, stored in a file referred to as pattern file. This is the only

file the user must provide. Optionally, parameters defining the pattern file columns, network size and network configuration may be stored in a file referred to as setup file.

NNDT includes a routine for graphical presentation of output signals, node activations, residuals and weights during run. The interface also provides facilities for examination of node activations and weights as well as modification of weights.

A Windows help file is included, help is achieved at any time during NNDT execution by pressing F1.

Installation

Unzip NNDTxx.ZIP to a separate disk or to a temporary directory e.g. to c:\tmp. The program is then installed by running SETUP.EXE. See INSTALL.TXT for more details.

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.

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.

161. 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

162. Nenet v1.0

163. Nenet v1.0 is a 32-bit Windows 95 and Windows NT 4.0 application
164. designed to facilitate the use of a Self-Organizing Map (SOM) algorithm.
165.
166. The major motivation for Nenet was to create a user-friendly SOM
167. algorithm tool with good visualization capabilities and with a GUI
168. allowing efficient control of the SOM parameters. The use scenarios have
169. stemmed from the user's point of view and a considerable amount of work
170. has been placed on the ease of use and versatile visualization methods.
171.
172. With Nenet, all the basic steps in map control can be performed. In
173. addition, Nenet also includes some more exotic and involved features
174. especially in the area of visualization.
175.
176. Features in Nenet version 1.0:
177. + Implements the standard Kohonen SOM algorithm
178. + Supports 2 common data preprocessing methods
179. + 5 different visualization methods with rectangular or hexagonal
180. topology
181. + 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.hut.fi/~jpronkko/nenet.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

For some of these simulators there are user mailing lists. Get the packages and look into their documentation for further info.

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"). Modem: 409-737-5222; Sysop: Wesley R. Elsberry; 4160 Pirates' Beach, Galveston, TX, USA; welsberr@orca.tamu.edu. There are lots of small simulator packages, the CNS ANNSIM file set. There is an ftp mirror site for the CNS ANNSIM file set at me.uta.edu [129.107.2.20] in the /pub/neural directory. Most ANN offerings are in /pub/neural/annsim.

Next part is part 6 (of 7). Previous part is part 4.

PART 6

Archive-name: ai-faq/neural-nets/part6
Last-modified: 1997-06-04
URL: ftp://ftp.sas.com/pub/neural/FAQ6.html
Maintainer: saswss@unx.sas.com (Warren S. Sarle)
The copyright for the description of each product is held by the producer or distributor of the product or whoever it was who supplied the description for the FAQ, who by submitting it for the FAQ gives permission for the description to be reproduced as part of the FAQ in any of the ways specified in part 1 of the FAQ.
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:

1. BrainMaker
2. SAS Neural Network Application
3. NeuralWorks
4. MATLAB Neural Network Toolbox
5. Propagator
6. NeuroForecaster

7. Products of NESTOR, Inc.
8. NeuroShell2/NeuroWindows
9. NuTank
10. Neuralyst
11. NeuFuz4
12. Cortex-Pro
13. Partek
14. NeuroSolutions v3.0
15. Qnet For Windows Version 2.0
16. NeuroLab, A Neural Network Library
17. havBpNet++
18. havFmNet++
19. IBM Neural Network Utility
20. NeuroGenetic Optimizer (NGO) Version 2.0
21. WAND
22. Atree 3.0 Adaptive Logic Network Development System
23. TDL v. 1.1 (Trans-Dimensional Learning)
24. NeurOn-Line
25. NeuFrame, NeuroFuzzy, NeuDesk and NeuRun
26. OWL Neural Network Library (TM)
27. Neural Connection
28. Pattern Recognition Workbench Expo/PRO/PRO+
29. PREVia
30. Neural Bench
31. Trajan 2.1 Neural Network Simulator
32. DataEngine

See also http://www.emsl.pnl.gov:2080/docs/cie/neural/systems/software.html

1. **BrainMaker**
2. Name: BrainMaker, BrainMaker Pro
3. Company: California Scientific Software
4. Address: 10024 Newtown rd, Nevada City, CA, 95959 USA
5. Phone,Fax: 916 478 9040, 916 478 9041
6. Email: calsci!mittmann@gvgpsa.gvg.tek.com (flakey connection)
7. Basic capabilities: train backprop neural nets
8. Operating system: DOS, Windows, Mac
9. System requirements:
10. Uses XMS or EMS for large models(PCs only): Pro version
11. Approx. price: $195, $795
12.
13. BrainMaker Pro 3.0 (DOS/Windows) $795
14. Gennetic Training add-on $250
15. ainMaker 3.0 (DOS/Windows/Mac) $195
16. Network Toolkit add-on $150
17. BrainMaker 2.5 Student version (quantity sales only, about $38 each)
18.
19. BrainMaker Pro C30 Accelerator Board
20. w/ 5Mb memory $9750
21. w/32Mb memory $13,000

Intel iNNTS NN Development System $11,800
Intel EMB Multi-Chip Board $9750
Intel 80170 chip set $940

Introduction To Neural Networks book $30

California Scientific Software can be reached at:
Phone: 916 478 9040 Fax: 916 478 9041 Tech Support: 916 478 9035
Mail: 10024 newtown rd, Nevada City, CA, 95959, USA
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 750,000 cps
on a 33Mhz 486.
---Features ("P" means is available in professional version only):
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

66. SAS Neural Network Application

Name: SAS Neural Network Application

	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
Email: software@sas.sas.com

Operating systems: Windows 3.1, OS/2, HP/UX, Solaris, AIX

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. The SAS Neural Network Application trains a variety of neural nets and includes a graphical user interface, on-site training and customisation. Features include 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, missing value handling, categorical variables, standardization of inputs and targets, and multiple preliminary optimizations from random initial values to avoid local minima. Training is done by state-of-the-art numerical optimization algorithms instead of tedious backprop.

80. NeuralWorks

Name: NeuralWorks Professional II Plus (from NeuralWare)
Company: NeuralWare Inc.
Adress: RIDC Park West
202 Park West Drive
Pittsburgh, PA 15275
Phone: (412) 787-8222
FAX: (412) 787-8220
Email: sales@neuralware.com
URL: http://www.neuralware.com/

Distributor for Europe:
Scientific Computers GmbH.
Franzstr. 107, 52064 Aachen
Germany
Tel. (49) +241-26041
Fax. (49) +241-44983
Email. info@scientific.de

Basic capabilities:
supports over 30 different nets: backprop, art-1,kohonen,
modular neural network, General regression, Fuzzy art-map,
probabilistic nets, self-organizing map, lvq, boltmann,
bsb, spr, etc...
Extendable with optional package.
ExplainNet, Flashcode (compiles net in .c code for runtime),
user-defined io in c possible. ExplainNet (to eliminate

extra inputs), pruning, savebest,graph.instruments like
correlation, hinton diagrams, rms error graphs etc..
Operating system : PC,Sun,IBM RS6000,Apple Macintosh,SGI,Dec,HP.
System requirements: varies. PC:2MB extended memory+6MB Harddisk space.
Uses windows compatible memory driver (extended).
Uses extended memory.
Approx. price : call (depends on platform)
Comments : award winning documentation, one of the market
leaders in NN software.

MATLAB Neural Network Toolbox

Contact: The MathWorks, Inc. Phone: 508-653-1415
24 Prime Park Way FAX: 508-653-2997
Natick, MA 01760 email: info@mathworks.com

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.

(Comment from Nigel Dodd, nd@neural.win-uk.net): there is also a free Neural Network Based System Identification Toolbox available from http://kalman.iau.dtu.dk/Projects/proj/nnsysid.html that contains many of the supervised training algorithms, some of which are duplicated in C code which

should run faster. This free toolbox does regularisation and pruning which the costly one doesn't attempt (as of Nov 1995).

(Message from Eric A. Wan, ericwan@eeap.ogi.edu) FIR/TDNN Toolbox for MATLAB: Beta version of a toolbox for FIR (Finite Impulse Response) and TD (Time Delay) Neural Networks. For efficient stochastic implementation, algorithms are written as MEX compatible c-code which can be called as primitive functions from within MATLAB. Both source and compiled functions are available. URL: http://www.eeap.ogi.edu/~ericwan/fir.html

120. Propagator

121. Contact: ARD Corporation,
122. 9151 Rumsey Road, Columbia, MD 21045, USA
123. propagator@ard.com
124. Easy to use neural network training package. A GUI implementation of
125. backpropagation networks with five layers (32,000 nodes per layer).
126. Features dynamic performance graphs, training with a validation set,
127. and C/C++ source code generation.
128. For Sun (Solaris 1.x & 2.x, $499),
129. PC (Windows 3.x, $199)
130. Mac (System 7.x, $199)
131. Floating point coprocessor required, Educational Discount,
132. Money Back Guarantee, Muliti User Discount
133. See http://www.cs.umbc.edu/~zwa/Gator/Description.html
134. Windows Demo on:
135. nic.funet.fi /pub/msdos/windows/demo
136. oak.oakland.edu /pub/msdos/neural_nets
137. gatordem.zip pkzip 2.04g archive file
138. gatordem.txt readme text file

139. NeuroForecaster & VisuaData

Name: *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, Internet: accel@technet.sg, accel@singapore.com

Neuroforecaster 4.1a for Windows is priced at **US$1199** per single user license. Please email us (accel@technet.sg) for order form.

For more information and evaluation copy please visit http://www.singapore.com/products/nfga.
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.

140. Products of NESTOR, Inc.
141. 530 Fifth Avenue;
142. New York, NY 10036; USA;
Tel.: 001-212-398-7955

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.

143. NeuroShell2/NeuroWindows

NeuroShell 2 combines powerful neural network architectures, a Windows icon driven user interface, and sophisticated utilities for MS-Windows machines. Internal format is spreadsheet, and users can specify that NeuroShell 2 use their own spreadsheet when editing. Includes both Beginner's and Advanced systems, a Runtime capability, and a choice of 15 Backpropagation, Kohonen, PNN and GRNN architectures. Includes Rules, Symbol Translate, Graphics, File Import/Export modules (including MetaStock from Equis International) and NET-PERFECT to prevent overtraining. Options available: Market Technical Indicator Option ($295), Market Technical Indicator Option with Optimizer ($590), and Race Handicapping Option ($149). NeuroShell price: $495.

NeuroWindows is a programmer's tool in a Dynamic Link Library (DLL) that can create as many as 128 interactive nets in an application, each with 32 slabs in a single network, and 32K neurons in a slab. Includes Backpropagation, Kohonen, PNN, and GRNN paradigms. NeuroWindows can mix supervised and unsupervised nets. The DLL may be called from Visual Basic, Visual C, Access Basic, C, Pascal, and VBA/Excel 5. NeuroWindows price: $369.

GeneHunter is a genetic algorithm with a Dynamic Link Library of genetic algorithm functions that may be called from programming languages such as Visual Basicd or C. For non-programmers, GeneHunter also includes an Exceld Add-in

program which allows the user to solve an optimization problem from an Excel spreadsheet.

Contact:
Ward Systems Group, Inc.;
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
Contact us for a free demo diskette and Consumer's Guide to Neural Networks.

144. 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. Eventualy I expect to do a port to UNIX and multi thread the sign. I also expect to do a Mac port and maybe NT or OS/2

Copies of NuTank cost $50 each.
Contact: Richard Keene; Keene Educational Software
URL: http://www.xmission.com/~rkeene Email: rkeene@parkcity.com or rkeene@xmission.com

NuTank shareware with the Save options disabled is available via anonymous ftp from the Internet, see the file /pub/incoming/nutank.readme on the host cher.media.mit.edu.

145. Neuralyst

146. Name: Neuralyst Version 1.4;
147. Company: Cheshire Engineering Corporation;
148. Address: 650 Sierra Madre Villa, Suite 201, Pasedena CA 91107;
149. Phone: 818-351-0209;
150. Fax: 818-351-8645;

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.

151. NeuFuz4

152. Name: NeuFuz4
153. Company: National Semiconductor Corporation
154. Address: 2900 Semiconductor Drive, Santa Clara, CA, 95052,
155. or: Industriestrasse 10, D-8080 Fuerstenfeldbruck, Germany,
156. or: Sumitomo Chemical Engineering Center, Bldg. 7F 1-7-1, Nakase,
157. Mihama-Ku, Chiba-City, Ciba Prefecture 261, JAPAN,
158. or: 15th Floor, Straight Block, Ocean Centre, 5 Canton Road, Tsim
159. Sha Tsui East, Kowloon, Hong Kong,
160. Phone: (800) 272-9959 (Americas),
161. : 011-49-8141-103-0 Germany
162. : 0l1-81-3-3299-7001 Japan
163. : (852) 737-1600 Hong Kong
164. Email: neufuz@esd.nsc.com (Neural net inquiries only)
165. URL: http://www.commerce.net/directories/participants/ns/home.html
166.
167. Basic capabilities:
168. Uses backpropagation techniques to initially select fuzzy rules
169. and membership functions. The result is a fuzzy associative memory (FAM)
170. which implements an approximation of the training data.
171. Operating Systems: 486DX-25 or higher with math co-processor
172. DOS 5.0 or higher with Windows 3.1, mouse,
173. VGA or better, minimum 4 MB RAM, and parallel port.
174. Approx. price : depends on version - see below.
175. Comments :
176. Not for the serious Neural Network researcher, but good for a person
177. who has little understanding of Neural Nets - and wants to keep it that
178. way. The systems are aimed at low end controls applications in
179. automotive, industrial, and appliance areas. NeuFuz is a neural-fuzzy
180. technology which uses backpropagation techniques to initially select
181. fuzzy rules and membership functions. Initial stages of design using
182. NeuFuz technology are performed using training data and
183. backpropagation. The result is a fuzzy associative memory (FAM) which
184. implements an approximation of the training data. By implementing a
185. FAM, rather than a multi-layer perceptron, the designer has a solution
186. which can be understood and tuned to a particular application using
187. Fuzzy Logic design techniques.

188. There are several different versions, some with COP8 Code Generator
189. (COP8 is National's family of 8-bit microcontrollers) and
190. COP8 in-circuit emulator (debug module).

191. 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>.

192. 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/

193. NeuroSolutions v3.0

NeuroSolutions is a graphical neural network simulation tool. Because of its object-oriented design, NeuroSolutions provides the flexibility needed to construct a wide range of learning paradigms and network topologies. Its GUI and extensive probing ability streamline the experimentation process by providing real-time analysis of the network during learning.

Download a free evaluation copy from **http://www.nd.com/demo/demo.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)

- User-defined network topologies

Learning Paradigms

- Backpropagation
- Recurrent Backpropagation
- Backpropagation through Time
- Unsupervised Learning
  - Hebbian
  - Oja's
  - Sanger's
  - Competitive
  - Kohonen

Advanced Features

- ANSI C++ Source Code Generation
- Customized Components through DLLs
- Microsoft Excel Add-in -- NeuroSolutions for Excel
  - Visual Data Selection
  - Data Preprocessing and Analysis
  - Batch Training and Parameter Optimization
  - Sensitivity Analysis
  - Automated Report Generation
- Comprehensive Macro Language
- Fully accessible from any OLE-compliant application, such as:
  - Visual Basic
  - Microsoft Excel
  - Microsoft Access

Price: $195 - $1995 (educational discounts available)
O.S.: Windows 95, Windows NT

NeuroDimension, Inc.
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/

194. Qnet For Windows Version 2.0
195. Vesta Services, Inc.
196. 1001 Green Bay Rd, Suite 196
197. Winnetka, IL 60093
198. Phone: (708) 446-1655
199. E-Mail: VestaServ@aol.com

Trial Version Available: ftp://oak.oakland.edu/SimTel/win3/neurlnet/qnetv2t.zip

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

200. 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.

201. havBpNet++

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).

hav.Software
P.O. Box 354
Richmond, Tx. 77406-0354 - USA
Phone: (713) 341-5035
Email: hav@neosoft.com
Web: http://www.neosoft.com/~hav

202. havFmNet++

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

hav.Software
P.O. Box 354
Richmond, Tx. 77406-0354 - USA
Phone: (713) 341-5035
Email: hav@neosoft.com
Web: http://www.neosoft.com/~hav

203. 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.

204. 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

263. 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.neuronet.ph.kcl.ac.uk/neuronet/software/nts/neural.html
To contact Novel Technical Solutions email: <neural@nts.sonnet.co.uk>.

264. Atree 3.0 Adaptive Logic Network Development System (for Windows)

Contact:
Dendronic Decisions Limited
3624 - 108 Street
Edmonton, Alberta
Canada T6J 1B4

Tel/Fax +1 (403) 438-8285
or email William W. Armstrong, President (arms@cs.ualberta.ca)
or use the Dendronic forum on CompuServe (GO DENDRONIC)
or visit Dendronic Decisions' website: www.dendronic.com .

Atree 3.0 trains feedforward networks having simple perceptrons in the first hidden layer and logic gates AND and OR in other hidden layers. Functions from real inputs to a real output can be represented, and are computed by using maximum and minimum operations on linear functions, accelerated by an ALN decision tree.

Users can specify *constraints* on monotonicity, derivatives (slopes) and convexity of functions being learned. Such expert knowledge can be used to ensure the result of training satisfies requirements of known physical or economic laws. Functions can be *inverted* without additional training, a capability useful in control applications.

The execution library, which computes learned functions at high speed using ALN decision trees, is offered in source form (code suitable for Windows and Unix is available free of charge). Atree 3.0 outputs ALN decision trees in human-readable form (for checking) as well as in binary form (for fast reloading). The commercial license allows redistribution and modification of execution code.

Atree 3.0 may be used for data analysis, prediction, pattern recognition and for real-time control applications that must run on a typical computer (such as a PC). Scripts can be run automatically and can be called from macros in Microsoft Excel and MS Access or from other applications. Many samples showing how to use Atree 3.0 are provided.

A manual of approximately 120 pages is included. It covers the Atree 3.0 software and some theory of ALNs.

Price: $269 Canadian (about $199 US) plus shipping and handling with generous academic and quantity discounts. Visit www.dendronic.com for a price list or call toll-free 1-888-370-5926 (in North America) for ordering information.

The software can be tried out with the help of an Educational Kit (EK) and an extensive User Guide which are available on the WWW. See information about the Atree 3.0 Educational Kit.

265. TDL v. 1.1 (Trans-Dimensional Learning)

266. Platform: Windows 3.*
267. Company: Universal Problem Solvers, Inc.
268. WWW-Site (UPSO): http://pages.prodigy.com/FL/lizard/index.html
269. or FTP-Site (FREE Demo only): ftp.coast.net, in Directory:
270. SimTel/win3/neurlnet, File: tdl11-1.zip and tdl11-2.zip
271. Cost of complete program: US$20 + (US$3 Shipping and Handling).
272.
273. The purpose of TDL is to provide users of neural networks with a specific
274. platform to conduct pattern recognition tasks. The system allows for the fast
275. creation of automatically constructed neural networks. There is no need to resort
276. to manually creating neural networks and twiddling with learning parameters.
277. TDL's Wizard can help you optimize pattern recognition accuracy. Besides
278. allowing the application user to automatically construct neural network for a given
279. pattern recognition task, the system supports trans-dimensional learning. Simply put,
280. this allows one to learn various tasks within a single network, which otherwise differ
281. in the number of input stimuli and output responses utilized for describing them.
282. With TDL it is possible to incrementally learn various pattern recognition tasks within
283. a single coherent neural network structure. Furthermore, TDL supports the use of
284. semi-weighted neural networks, which represent a hybrid cross between standard
285. weighted neural networks and weightless multi-level threshold units. Combining both
286. can result in extremely compact network structures (i.e., reduction in connections and
287. hidden units), and improve predictive accuracy on yet unseen patterns. Of course the
288. user has the option to create networks which only use standard weighted neurons.
289.
290. System Highlights:
291. (1) The user is in control of TDL's memory system (can decide how many examples
292. and neurons are allocated ; no more limitations, except for your computers memory).
293. (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.

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.

NeuFrame, NeuroFuzzy, NeuDesk and NeuRun

Name: NeuFrame, NeuroFuzzy, NeuDesk and NeuRun

Company: NCS
Address: Unit 3
Lulworth Business Centre
Totton
Southampton
UK
SO40 3WW
Phone: +44 (0)1703 667775
Fax: +44 (0)1703 663730
Email: robby@ncs-skylake.co.uk
URL: http://www.demon.co.uk/skylake/software.html

NeuFrame
NeuFrame provides an easy-to-use, visual, object-oriented approach to problem solving using intelligence technologies, including neural networks and neurofuzzy techniques. It provides features to enable businesses to investigate and apply intelligence technologies from initial experimentation through to building embedded implementations using software components.
* Minimum Configuration- Windows 3.1 with win32s, 386DX 33MHz, 8Mb memory, 5Mb free disk space,VGA graphics, mouse
* Recommended Configuration - Windows 95/NT 486DX 50MHz or above 16Mb memory,150Mb or above hard disc, VGA graphics, Mouse.
* Price Commercial 749 (Pounds Sterling), Educational 435

NeuroFuzzy
This is an optional module for use within the NeuFrame enviroment. Fuzzy logic differs from neural networks in the sense that neural systems are constructed purely from available data whereas fuzzy systems are expert designed. The relative merits of the two approaches is very much application dependent as it relies on the availability of training data and expert knowledge. Conventional knowledge based systems can also be used to represent expert knowledge within computer systems, but fuzzy logic provides a richer representation. Benefits include:
i)Combines the benefits of rule based fuzzy logic and learning from experience of neural networks.
ii) Automatically generate optimised neurofuzzy models.
iii) Encapsulate expert fuzzy knowledge within the model.
iv) Fuzzy models may be modified or created according to the data presented to the network.
v) Includes a pure Fuzzy logic editor .
* Requires NeuFrame
* Price Commercial 249 (Pounds Sterling), Educational 149

NeuDesk
NeuDesk 2 makes the implementation of a neural network solution very accessible. Running within the Windows 3 environment, NeuDesk 2 is easy for non-specialists to use. Data required for training the neural network can be entered manually, by cut and paste from any other Windows

application, or imported from a number of different file formats.
Training the network is achieved by a few straightforward
point-and-click operations.
* Recommended Configuration - Microsoft Windows 3.0 or higher with a
minimum of 2Mb RAM and 20Mbyte hard disk and mouse. 4Mb RAM, 387
co-processor and a 40Mb hard disk are recommended.
* Price Commercial 349 (Pounds Sterling), Educational 149

NeuRun
NeuRun provides for the embedding of intelligence technologies developed
in NeuFrame or NeuDesk to be embedded into other programs and
environments. NeuRun simplifies and speeds up the process of embedding
neural networks in your favourite Windows applications to provide
on-line intelligence for decisions, predictions, suggestions or
classifications. Implementations are easy to duplicate and deploy and
can be readily updated if the problem conditions change over time.
Typical of the application programs that can be enhanced using NeuRun 3
are: Excel, Microsoft's spreadsheet for creating powerful data
manipulations and analysis applications and Visual Basic which can be
used to generate user defined screens and functions for custom operator
interfaces, data entry forms, control panels etc.
* Price Commercial 50 (Pounds Sterling), Educational 50

OWL Neural Network Library (TM)

Name: 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/hl

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.

Neural Connection

Name: 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

525. Example Applications
526. * Finance - predict account attrition
527. * Marketing - customer segmentation
528. * Medical - predict length of stay in hospital
529. * Consulting - forecast construction needs of federal court systems
530. * Utilities - predict number of service requests
531. * Sales - forecast product demand and sales
532. * Science - classify climate types
533.

534. 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!

535. 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.com/prod01.htm. This software is being used mainly in France, by banks and some of the largest investment companies, such as: Banque de France (French Central Bank), AXA Asset Management, Credit Lyonnais Asset Management, Caisse des Depots AM, Banque du Luxembourg and others. In order to enhance the research and applications using PREVia, it has been given free of charge to European Engineering and Business Schools, such as Ecole Centrale Paris, London Business School, EURIA, CEFI, Universite du Luxembourg. Interested universities and schools should contact Elseware at the forementioned page.

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.

536. Neural Bench

537. Name: Neural Bench.
538. Company: Neural Bench Development.
539. Address: 142717, Computer Technology Laboratory, VNIIGAZ, pos. Razvilka,
540. Address: Leninsky rajon, Moskovskay oblast, Russia.
541. Phone: 7 (095) 355-9191
542. Fax: 7 (095) 399-1677

543. Email: develop.nb@vniigaz.com
544. URL: http://www.vniigaz.com/nb/
545. Basic capabilities: data processing, train, analysis, statistic,
546. user interface, code generation, Java support...
547. Operating system: Windows 3.x, Windows 95, Windows NT

Neural Bench package includes:

- Neural Bench neural networks simulator - comprehensive both easy-to-use and sophisticated shell. It includes: Neural Network Builder, Neural Network Wizard, Scale and Randomization, Neurons and Layers Editing, Topology Adjustments, Background Learning, Testing, User Dialog Interface, Source Code Wizard, and much more.
- Data Processor with Data Wizard is a powerful neural network data pre- and postprocessing instrument. Multiple testing and analyses options are available. For the purpose of the complex data visualization the special chart tool provided.
- Dialog Editor with Dialog Wizard is a special utility for the user-defined dialog creation and editing. You can include in your custom dialogs any standard elements as well as bitmap editing elements and different graphics files.

548. 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.

549. DataEngine

550. Name: DataEngine, DataEngine ADL, DataEngine V.i
551.
552. Company: MIT GmbH
553. Address: Promenade 9
554. 52076 Aachen
555. Germany
556.
557. Phone: +49 2408 94580
558. Fax: +49 2408 94582
559. EMail: mailto:info@mitgmbh.de
560. URL: http://www.mitgmbh.de
561.

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.

Next part is part 7 (of 7). Previous part is part 5.

PART 7

Archive-name: ai-faq/neural-nets/part7
Last-modified: 1997-08-11
URL: ftp://ftp.sas.com/pub/neural/FAQ7.html
Maintainer: saswss@unx.sas.com (Warren S. Sarle)
Copyright 1997 by Warren S. Sarle, Cary, NC, USA. Answers provided by other authors as cited below are copyrighted by those authors, who by submitting the answers for the FAQ give permission for the answer to be reproduced as part of the FAQ in any of the ways specified in part 1 of the FAQ.
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?

Thomas Lindblad notes on 96-12-30:

> The reactive tabu search algorithm has been implemented by the Italians, in Trento. ISA and VME and soon PCI boards are available. We tested the system with the IRIS and SATIMAGE data and it did better than most other chips.
>
> The Neuroclassifier is available from Holland still and is also the fastest nnw chip or a transient time less than 100 ns.
>
> JPL is making another chip, ARL in WDC is making another, so there are a few things going on ...

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
Further WWW pointers to NN Hardware:
http://msia02.msi.se/~lindsey/nnwAtm.html
Here is a short list of companies:

1. **HNC, INC.**
2. HNC Inc.
3. 5930 Cornerstone Court West
4. San Diego, CA 92121-3728
5.
6. 619-546-8877 Phone
7. 619-452-6524 Fax
8. HNC markets:
9. Database Mining Workstation (DMW), a PC based system that
10. builds models of relationships and patterns in data. AND
11. The SIMD Numerical Array Processor (SNAP). It is an attached
12. parallel array processor in a VME chassis with between 16 and 64 parallel
13. floating point processors. It provides between 640 MFLOPS and 2.56 GFLOPS
14. for neural network and signal processing applications. A Sun SPARCstation
15. serves as the host. The SNAP won the IEEE 1993 Gordon Bell Prize for best
16. price/performance for supercomputer class systems.
17. **SAIC (Sience Application International Corporation)**
18. 10260 Campus Point Drive
19. MS 71, San Diego
20. CA 92121
21. (619) 546 6148
22. Fax: (619) 546 6736
23. **Micro Devices**
24. 30 Skyline Drive
25. Lake Mary
26. FL 32746-6201
27. (407) 333-4379
28. MicroDevices makes MD1220 - 'Neural Bit Slice'
29. Each of the products mentioned sofar have very different usages.
30. Although this sounds similar to Intel's product, the
31. architectures are not.
32. **Intel Corp**
33. 2250 Mission College Blvd
34. Santa Clara, Ca 95052-8125
35. Attn ETANN, Mail Stop SC9-40
36. (408) 765-9235
37. 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'.

48. NeuralWare, Inc

Penn Center West
Bldg IV Suite 227
Pittsburgh
PA 15276
They only sell software/simulator but for many platforms.

54. Tubb Research Limited

7a Lavant Street
Peterfield
Hampshire
GU32 2EL
United Kingdom
Tel: +44 730 60256

61. Adaptive Solutions Inc

1400 NW Compton Drive
Suite 340
Beaverton, OR 97006
U. S. A.
Tel: 503-690-1236; FAX: 503-690-1249

67. NeuroDynamX, Inc.

P.O. Box 14
Marion, OH 43301-0014
Voice (614) 387-5074 Fax: (614) 382-4533
Internet: jwrogers@on-ramp.net

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.

77. IC Tech, Inc.

* NRAM (Neural Retrieve Associative Memory) is available as a stand-alone chip or
a functional unit which can be embedded inside another chip, e.g., a digital signal
processor or SRAM. Data storage procedure is compatible with conventional
memories, i.e., a single presentation of the data is sufficient. Set-up and hold
times are comparable with existing devices of similar technology dimensions.
Data retrieval capability is where NRAM excels. When addressed, this content
addressable memory produces the one previously-stored pattern that matches the
presented data sequence most closely. If no matching pattern is found, no data is

returned. This set of error-correction and smart retrieval tasks are accomplished without
comparators, processors, or other external logic. Number of data bits is adjustable.
Optimized circuitry consumes little power. Many applications of NRAM exist in rapid
search of large databases, template matching, and associative recall.

* NRAM (neural retrieve associate memory) development environment includes PC card
with on board NRAM chip and C++ source code to address the device.

Contact:

IC Tech, Inc.
2157 University Park Dr.
Okemos, MI 48864
(517) 349-4544
(517) 349-2559 (FAX)
http://www.ic-tech.com
ictech@ic-tech.com

And here is an incomplete overview of known Neural Computers with their newest known reference.

\subsection*{Digital}
\subsubsection{Special Computers}

{\bf 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.

{\bf ANNA}
B.E.Boser, E.Sackinger, J.Bromley, Y.leChun, and L.D.Jackel.\\
Hardware Requirements for Neural Network Pattern Classifiers.\\
In {\it IEEE Micro}, 12(1), pages 32-40, February 1992.

{\bf Analog Neural Computer}
Paul Mueller et al.
Design and performance of a prototype analog neural computer.
In Neurocomputing, 4(6):311-323, 1992.

{\bf APx -- Array Processor Accelerator}\\
F.Pazienti.\\
Neural networks simulation with array processors.
In {\it 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.

{\bf ASP -- Associative String Processor}\\
A.Krikelis.\\
A novel massively associative processing architecture for the
implementation artificial neural networks.\\
In {\it 1991 International Conference on Acoustics, Speech and
Signal Processing}, volume 2, pages 1057-1060. IEEE Comput. Soc. Press,
May 1991.

{\bf 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.

{\bf BLAST}\\
J.G.Elias, M.D.Fisher, and C.M.Monemi.\\
A multiprocessor machine for large-scale neural network simulation.
In {\it IJCNN91-Seattle: International Joint Conference on Neural
Networks}, volume 1, pages 469-474. IEEE Comput. Soc. Press, July 1991.
ISBN: 0-7883-0164-1.

{\bf CNAPS Neurocomputer}\\
H.McCartor\\
Back Propagation Implementation on the Adaptive Solutions CNAPS
Neurocomputer.\\
In {\it Advances in Neural Information Processing Systems}, 3, 1991.

{\bf 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.

{\bf MA16 -- Neural Signal Processor}
U.Ramacher, J.Beichter, and N.Bruls.\\
Architecture of a general-purpose neural signal processor.\\
In {\it IJCNN91-Seattle: International Joint Conference on Neural
Networks}, volume 1, pages 443-446. IEEE Comput. Soc. Press, July 1991.
ISBN: 0-7083-0164-1.

{\bf 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.

{\bf 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.

{\bf NERV}\\
R.Hauser, H.Horner, R. Maenner, and M.Makhaniok.\\
Architectural Considerations for NERV - a General Purpose Neural
Network Simulation System.\\
In {\it Workshop on Parallel Processing: Logic, Organization and
Technology -- WOPPLOT 89}, pages 183-195. Springer Verlag, Mars 1989.
ISBN: 3-5405-5027-5.

{\bf 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 {\it 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.

{\bf 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 {\it Journal of Parallel and Distributed Computing}, pages
248-259, April 1992.

{\bf 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 {\it 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.

{\bf 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 {\it VLSI design of Neural Networks}, pages 311-335, June 1990.

{\bf SNAP -- Scalable Neurocomputer Array Processor}
E.Wojciechowski.\\
SNAP: A parallel processor for implementing real time neural networks.\\
In {\it Proceedings of the IEEE 1991 National Aerospace and Electronics
Conference; NAECON-91}, volume 2, pages 736-742. IEEE Comput.Soc.Press,
May 1991.

{\bf Toroidal Neural Network Processor}\\
S.Jones, K.Sammut, C.Nielsen, and J.Staunstrup.\\
Toroidal Neural Network: Architecture and Processor Granularity Issues.\\
In {\it VLSI design of Neural Networks}, pages 229-254, June 1990.

{\bf 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.

\subsubsection{Standard Computers}

{\bf 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 {\it 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.

{\bf iPSC/860 Hypercube}\\
D.Jackson, and D.Hammerstrom\\
Distributing Back Propagation Networks Over the Intel iPSC/860 Hypercube}\\
In {\it IJCNN91-Seattle: International Joint Conference on Neural Networks}, volume 1, pages 569-574. IEEE Comput. Soc. Press, July 1991.
ISBN: 0-7083-0164-1.

{\bf 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 {\it IEEE Transactions on Computers}, 40(12), pages 1390-1401, December 1991. ISSN: 0018-9340.

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.

Subject: Unanswered FAQs

If you have good answers for any of these questions, please send them to the FAQ maintainer at saswss@unx.sas.com.

- How many training cases do I need?
- How should I split the data into training and validation sets?
- What error functions can be used?
- What are some good constructive training algorithms?
- How can I invert a network?
- How can I select important input variables?
- How to handle missing data?
- Should NNs be used in safety-critical applications?
- My net won't learn! What should I do???
- My net won't generalize! What should I do???

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 <drt@mcs.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>
- Gregory E. Heath <heath@ll.mit.edu>
- 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@kal-el.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>

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