Archive-name: ai-faq/neural-nets/part3
Last-modified: 2001-05-21
URL: ftp://ftp.sas.com/pub/neural/FAQ3.html
Maintainer: saswss@unx.sas.com (Warren S. Sarle)

This is part 3 (of 7) of a monthly posting to the Usenet newsgroup comp.ai.neural-nets. See the part 1 of this posting for full information what it is all about.

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

Subject: How is generalization possible?

During learning, the outputs of a supervised neural net come to approximate the target values given the inputs in the training set. This ability may be useful in itself, but more often the purpose of using a neural net is to generalize--i.e., to have the outputs of the net approximate target values given inputs that are *not* in the training set. Generalizaton is not always possible, despite the blithe assertions of some authors. For example, Caudill and Butler, 1990, p. 8, claim that "A neural network is able to generalize", but they provide no justification for this claim, and they completely neglect the complex issues involved in getting good generalization. Anyone who reads comp.ai.neural-nets is well aware from the numerous posts pleading for help that artificial neural networks do not automatically generalize.

Generalization requires prior knowledge, as pointed out by Hume (1739/1978), Russell (1948), and

Goodman (1954/1983) and rigorously proved by Wolpert (1995a, 1996a, 1996b). For any practical application, you have to know what the relevant inputs are (you can't simply include every imaginable input). You have to know a restricted class of input-output functions that contains an adequate approximation to the function you want to learn (you can't use a learning method that is capable of fitting every imaginable function). And you have to know that the cases you want to generalize to bear some resemblance to the training cases. Thus, there are three conditions that are typically necessary--although not sufficient--for good generalization:

1. The first necessary condition is that the inputs to the network contain sufficient information pertaining to the target, so that there exists a mathematical function relating correct outputs to inputs with the desired degree of accuracy. You can't expect a network to learn a nonexistent function--neural nets are not clairvoyant! For example, if you want to forecast the price of a stock, a historical record of the stock's prices is rarely sufficient input; you need detailed information on the financial state of the company as well as general economic conditions, and to avoid nasty surprises, you should also include inputs that can accurately predict wars in the Middle East and earthquakes in Japan. Finding good inputs for a net and collecting enough training data often take far more time and effort than training the network.

2. The second necessary condition is that the function you are trying to learn (that relates inputs to correct outputs) be, in some sense, smooth. In other words, a small change in the inputs should, most of the time, produce a small change in the outputs. For continuous inputs and targets, smoothness of the function implies continuity and restrictions on the first derivative over most of the input space. Some neural nets can learn discontinuities as long as the function consists of a finite number of continuous pieces. Very nonsmooth functions such as those produced by pseudo-random number generators and encryption algorithms cannot be generalized by neural nets. Often a nonlinear transformation of the input space can increase the smoothness of the function and improve generalization.

   For classification, if you do not need to estimate posterior probabilities, then smoothness is not theoretically necessary. In particular, feedforward networks with one hidden layer trained by minimizing the error rate (a very tedious training method) are universally consistent classifiers if the number of hidden units grows at a suitable rate relative to the number of training cases (Devroye, Györfi, and Lugosi, 1996). However, you are likely to get better generalization with realistic sample sizes if the classification boundaries are smoother.

   For Boolean functions, the concept of smoothness is more elusive. It seems intuitively clear that a Boolean network with a small number of hidden units and small weights will compute a "smoother" input-output function than a network with many hidden units and large weights. If you know a good reference characterizing Boolean functions for which good generalization is possible, please inform the FAQ maintainer (saswss@unx.sas.com).

3. The third necessary condition for good generalization is that the training cases be a sufficiently large and representative subset ("sample" in statistical terminology) of the set of all cases that you want to generalize to (the "population" in statistical terminology). The importance of this condition is related to the fact that there are, loosely speaking, two different types of generalization: interpolation and extrapolation. Interpolation applies to cases that are more or less surrounded by nearby training cases; everything else is extrapolation. In particular, cases that are outside the range of the training data require extrapolation. Cases inside large "holes" in the training data may also effectively require extrapolation. Interpolation can often be done reliably, but extrapolation is notoriously unreliable. Hence it is important to have sufficient training data to avoid the need for extrapolation. Methods for selecting good training sets are discussed in numerous statistical textbooks on sample surveys and experimental design.

Thus, for an input-output function that is smooth, if you have a test case that is close to some training cases, the correct output for the test case will be close to the correct outputs for those training cases. If you have an adequate sample for your training set, every case in the population will be close to a sufficient number of training cases. Hence, under these conditions and with proper training, a neural net will be able to generalize reliably to the population.

If you have more information about the function, e.g. that the outputs should be linearly related to the inputs, you can often take advantage of this information by placing constraints on the network or by fitting a more specific model, such as a linear model, to improve generalization. Extrapolation is much more reliable in linear models than in flexible nonlinear models, although still not nearly as safe as interpolation. You can also use such information to choose the training cases more efficiently. For example, with a linear model, you should choose training cases at the outer limits of the input space instead of evenly distributing them throughout the input space.

References:

Caudill, M. and Butler, C. (1990). *Naturally Intelligent Systems.* MIT Press: Cambridge, Massachusetts.

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

Goodman, N. (1954/1983), *Fact, Fiction, and Forecast,* 1st/4th ed., Cambridge, MA: Harvard University Press.

Holland, J.H., Holyoak, K.J., Nisbett, R.E., Thagard, P.R. (1986), *Induction: Processes of Inference, Learning, and Discovery,* Cambridge, MA: The MIT Press.

Howson, C. and Urbach, P. (1989), *Scientific Reasoning: The Bayesian Approach,* La Salle, IL: Open Court.

Hume, D. (1739/1978), *A Treatise of Human Nature,* Selby-Bigge, L.A., and Nidditch, P.H. (eds.), Oxford: Oxford University Press.

Plotkin, H. (1993), *Darwin Machines and the Nature of Knowledge,* Cambridge, MA: Harvard University Press.

Russell, B. (1948), *Human Knowledge: Its Scope and Limits,* London: Routledge.

Stone, C.J. (1977), "Consistent nonparametric regression," Annals of Statistics, 5, 595-645.

Stone, C.J. (1982), "Optimal global rates of convergence for nonparametric regression," Annals of Statistics, 10, 1040-1053.

Vapnik, V.N. (1995), *The Nature of Statistical Learning Theory,* NY: Springer.

Wolpert, D.H. (1995a), "The relationship between PAC, the statistical physics framework, the Bayesian framework, and the VC framework," in Wolpert (1995b), 117-214.

Wolpert, D.H. (ed.) (1995b), *The Mathematics of Generalization: The Proceedings of the SFI/CNLS Workshop on Formal Approaches to Supervised Learning,* Santa Fe Institute Studies in the Sciences of Complexity, Volume XX, Reading, MA: Addison-Wesley.

Wolpert, D.H. (1996a), "The lack of a priori distinctions between learning algorithms," Neural

Computation, 8, 1341-1390.

Wolpert, D.H. (1996b), "The existence of a priori distinctions between learning algorithms," Neural Computation, 8, 1391-1420.

Subject: How does noise affect generalization?

"Statistical noise" means variation in the target values that is unpredictable from the inputs of a specific network, regardless of the architecture or weights. "Physical noise" refers to variation in the target values that is inherently unpredictable regardless of what inputs are used. Noise in the inputs usually refers to measurement error, so that if the same object or example is presented to the network more than once, the input values differ.

Noise in the actual data is never a good thing, since it limits the accuracy of generalization that can be achieved no matter how extensive the training set is. On the other hand, injecting artificial noise (jitter) into the inputs during training is one of several ways to improve generalization for smooth functions when you have a small training set.

Certain assumptions about noise are necessary for theoretical results. Usually, the noise distribution is assumed to have zero mean and finite variance. The noise in different cases is usually assumed to be independent or to follow some known stochastic model, such as an autoregressive process. The more you know about the noise distribution, the more effectively you can train the network (e.g., McCullagh and Nelder 1989).

If you have noise in the target values, what the network learns depends mainly on the error function. For example, if the noise is independent with finite variance for all training cases, a network that is well-trained using least squares will produce outputs that approximate the conditional mean of the target values (White, 1990; Bishop, 1995). Note that for a binary 0/1 variable, the mean is equal to the probability of getting a 1. Hence, the results in White (1990) immediately imply that for a categorical target with independent noise using 1-of-C coding (see "How should categories be encoded?"), a network that is well-trained using least squares will produce outputs that approximate the posterior probabilities of each class (see Rojas, 1996, if you want a simple explanation of why least-squares estimates probabilities). Posterior probabilities can also be learned using cross-entropy and various other error functions (Finke and Müller, 1994; Bishop, 1995). The conditional median can be learned by least-absolute-value training (White, 1992a). Conditional modes can be approximated by yet other error functions (e.g., Rohwer and van der Rest 1996). For noise distributions that cannot be adequately approximated by a single location estimate (such as the mean, median, or mode), a network can be trained to approximate quantiles (White, 1992a) or mixture components (Bishop, 1995; Husmeier, 1999).

If you have noise in the target values, the mean squared generalization error can never be less than the variance of the noise, no matter how much training data you have. But you can estimate the mean of the target values, conditional on a given set of input values, to any desired degree of accuracy by obtaining a sufficiently large and representative training set, assuming that the function you are trying to learn is one that can indeed be learned by the type of net you are using, and assuming that the complexity of the network is regulated appropriately (White 1990).

Noise in the target values increases the danger of overfitting (Moody 1992).

Noise in the inputs limits the accuracy of generalization, but in a more complicated way than does noise in the targets. In a region of the input space where the function being learned is fairly flat, input noise will

have little effect. In regions where that function is steep, input noise can degrade generalization severely.

Furthermore, if the target function is Y=f(X), but you observe noisy inputs X+D, you cannot obtain an arbitrarily accurate estimate of f(X) given X+D no matter how large a training set you use. The net will not learn f(X), but will instead learn a convolution of f(X) with the distribution of the noise D (see "What is jitter?)"

For more details, see one of the statistically-oriented references on neural nets such as:

Bishop, C.M. (1995), *Neural Networks for Pattern Recognition,* Oxford: Oxford University Press, especially section 6.4.

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

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

Husmeier, D. (1999), *Neural Networks for Conditional Probability Estimation: Forecasting Beyond Point Predictions,* Berlin: Springer Verlag, ISBN 185233095.

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

Moody, J.E. (1992), "The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems", in Moody, J.E., Hanson, S.J., and Lippmann, R.P., *Advances in Neural Information Processing Systems 4,* 847-854.

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

Rohwer, R., and van der Rest, J.C. (1996), "Minimum description length, regularization, and multimodal data," Neural Computation, 8, 595-609.

Rojas, R. (1996), "A short proof of the posterior probability property of classifier neural networks," Neural Computation, 8, 41-43.

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

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

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

Subject: What is overfitting and how can I avoid it?

The critical issue in developing a neural network is generalization: how well will the network make predictions for cases that are not in the training set? NNs, like other flexible nonlinear estimation methods

such as kernel regression and smoothing splines, can suffer from either underfitting or overfitting. A network that is not sufficiently complex can fail to detect fully the signal in a complicated data set, leading to underfitting. A network that is too complex may fit the noise, not just the signal, leading to overfitting. Overfitting is especially dangerous because it can easily lead to predictions that are far beyond the range of the training data with many of the common types of NNs. Overfitting can also produce wild predictions in multilayer perceptrons even with noise-free data.

For an elementary discussion of overfitting, see Smith (1996). For a more rigorous approach, see the article by Geman, Bienenstock, and Doursat (1992) on the bias/variance trade-off (it's not really a dilemma). We are talking about statistical bias here: the difference between the average value of an estimator and the correct value. Underfitting produces excessive bias in the outputs, whereas overfitting produces excessive variance. There are graphical examples of overfitting and underfitting in Sarle (1995, 1999).

The best way to avoid overfitting is to use *lots* of training data. If you have at least 30 times as many training cases as there are weights in the network, you are unlikely to suffer from much overfitting, although you may get some slight overfitting no matter how large the training set is. For noise-free data, 5 times as many training cases as weights may be sufficient. But you can't arbitrarily reduce the number of weights for fear of underfitting.

Given a fixed amount of training data, there are at least six approaches to avoiding underfitting and overfitting, and hence getting good generalization:

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

The first five approaches are based on well-understood theory. Methods for combining networks do not have such a sound theoretical basis but are the subject of current research. These six approaches are discussed in more detail under subsequent questions.

The complexity of a network is related to both the number of weights and the size of the weights. Model selection is concerned with the number of weights, and hence the number of hidden units and layers. The more weights there are, relative to the number of training cases, the more overfitting amplifies noise in the targets (Moody 1992). The other approaches listed above are concerned, directly or indirectly, with the size of the weights. Reducing the size of the weights reduces the "effective" number of weights--see Moody (1992) regarding weight decay and Weigend (1994) regarding early stopping. Bartlett (1997) obtained learning-theory results in which generalization error is related to the L_1 norm of the weights instead of the VC dimension.

Overfitting is not confined to NNs with hidden units. Overfitting can occur in generalized linear models (networks with no hidden units) if either or both of the following conditions hold:

1. The number of input variables (and hence the number of weights) is large with respect to the number of training cases. Typically you would want at least 10 times as many training cases as input variables, but with noise-free targets, twice as many training cases as input variables would be more than adequate. These requirements are smaller than those stated above for networks with hidden layers, because hidden layers are prone to creating ill-conditioning and other pathologies.

2. The input variables are highly correlated with each other. This condition is called

"multicollinearity" in the statistical literature. Multicollinearity can cause the weights to become extremely large because of numerical ill-conditioning--see "How does ill-conditioning affect NN training?"

Methods for dealing with these problems in the statistical literature include ridge regression (similar to weight decay), partial least squares (similar to Early stopping), and various methods with even stranger names, such as the lasso and garotte (van Houwelingen and le Cessie, ????).

References:

Bartlett, P.L. (1997), "For valid generalization, the size of the weights is more important than the size of the network," in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) *Advances in Neural Information Processing Systems 9,* Cambrideg, MA: The MIT Press, pp. 134-140.

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

Moody, J.E. (1992), "The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems", in Moody, J.E., Hanson, S.J., and Lippmann, R.P., *Advances in Neural Information Processing Systems 4,* 847-854.

Sarle, W.S. (1995), "Stopped Training and Other Remedies for Overfitting," *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics,* 352-360, ftp://ftp.sas.com/pub/neural/inter95.ps.Z (this is a *very large* compressed postscript file, *747K*, 10 pages)

Sarle, W.S. (1999), "Donoho-Johnstone Benchmarks: Neural Net Results," ftp://ftp.sas.com/pub/neural/dojo/dojo.html

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

van Houwelingen,H.C., and le Cessie, S. (????), "Shrinkage and penalized likelihood as methods to improve predictive accuracy," http://www.medstat.medfac.leidenuniv.nl/ms/HH/Files/shrinkage.pdf and http://www.medstat.medfac.leidenuniv.nl/ms/HH/Files/figures.pdf

Weigend, A. (1994), "On overfitting and the effective number of hidden units," *Proceedings of the 1993 Connectionist Models Summer School*, 335-342.

Subject: What is jitter? (Training with noise)

Jitter is artificial noise deliberately added to the inputs during training. Training with jitter is a form of smoothing related to kernel regression (see "What is GRNN?"). It is also closely related to regularization methods such as weight decay and ridge regression.

Training with jitter works because the functions that we want NNs to learn are mostly smooth. NNs can learn functions with discontinuities, but the functions must be piecewise continuous in a finite number of regions if our network is restricted to a finite number of hidden units.

In other words, if we have two cases with similar inputs, the desired outputs will usually be similar. That means we can take any training case and generate new training cases by adding small amounts of jitter to the inputs. As long as the amount of jitter is sufficiently small, we can assume that the desired output will

not change enough to be of any consequence, so we can just use the same target value. The more training cases, the merrier, so this looks like a convenient way to improve training. But too much jitter will obviously produce garbage, while too little jitter will have little effect (Koistinen and Holmström 1992).

Consider any point in the input space, not necessarily one of the original training cases. That point could possibly arise as a jittered input as a result of jittering any of several of the original neighboring training cases. The average target value at the given input point will be a weighted average of the target values of the original training cases. For an infinite number of jittered cases, the weights will be proportional to the probability densities of the jitter distribution, located at the original training cases and evaluated at the given input point. Thus the average target values given an infinite number of jittered cases will, by definition, be the Nadaraya-Watson kernel regression estimator using the jitter density as the kernel. Hence, training with jitter is an approximation to training with the kernel regression estimator as target. Choosing the amount (variance) of jitter is equivalent to choosing the bandwidth of the kernel regression estimator (Scott 1992).

When studying nonlinear models such as feedforward NNs, it is often helpful first to consider what happens in linear models, and then to see what difference the nonlinearity makes. So let's consider training with jitter in a linear model. Notation:

```
x_ij is the value of the jth input (j=1, ..., p) for the
     ith training case (i=1, ..., n).
X={x_ij} is an n by p matrix.
y_i is the target value for the ith training case.
Y={y_i} is a column vector.
```

Without jitter, the least-squares weights are `B = inv(X'X)X'Y`, where "inv" indicates a matrix inverse and "'" indicates transposition. Note that if we replicate each training case `c` times, or equivalently stack `c` copies of the `X` and `Y` matrices on top of each other, the least-squares weights are `inv(cX'X)cX'Y = (1/c)inv(X'X)cX'Y = B`, same as before.

With jitter, `x_ij` is replaced by `c` cases `x_ij+z_ijk, k=1, ..., c`, where `z_ijk` is produced by some random number generator, usually with a normal distribution with mean 0 and standard deviation `s`, and the `z_ijk's` are all independent. In place of the `n` by `p` matrix `X`, this gives us a big matrix, say `Q`, with `cn` rows and `p` columns. To compute the least-squares weights, we need `Q'Q`. Let's consider the `j`th 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,v`th off-diagonal element of `Q'Q`, which is

$$\sum_{i,k} (x_{iu}+z_{iuk})(x_{iv}+z_{ivk})$$

which is approximately, for `c` large,

$$c\left(\sum_i x_{iu}\, x_{iv}\right)$$

which is just `c` times the corresponding element of `X'X`. Thus, `Q'Q` equals `c(X'X+ns^2I)`, where `I` is an

identity matrix of appropriate size. Similar computations show that the crossproduct of Q with the target values is cX'Y. Hence the least-squares weights with jitter of variance s^2 are given by

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

In the statistics literature, B(ns^2) is called a ridge regression estimator with ridge value ns^2.

If we were to add jitter to the target values Y, the cross-product X'Y would not be affected for large c for the same reason that the off-diagonal elements of X'X are not afected by jitter. Hence, adding jitter to the targets will not change the optimal weights; it will just slow down training (An 1996).

The ordinary least squares training criterion is (Y-XB)'(Y-XB). Weight decay uses the training criterion (Y-XB)'(Y-XB)+d^2B'B, where d is the decay rate. Weight decay can also be implemented by inventing artificial training cases. Augment the training data with p new training cases containing the matrix dI for the inputs and a zero vector for the targets. To put this in a formula, let's use A;B to indicate the matrix A stacked on top of the matrix B, so (A;B)'(C;D)=A'C+B'D. Thus the augmented inputs are X;dI and the augmented targets are Y;0, where 0 indicates the zero vector of the appropriate size. The squared error for the augmented training data is:

```
(Y;0-(X;dI)B)'(Y;0-(X;dI)B)
= (Y;0)'(Y;0) - 2(Y;0)'(X;dI)B + B'(X;dI)'(X;dI)B
= Y'Y - 2Y'XB + B'(X'X+d^2I)B
= Y'Y - 2Y'XB + B'X'XB + B'(d^2I)B
= (Y-XB)'(Y-XB)+d^2B'B
```

which is the weight-decay training criterion. Thus the weight-decay estimator is:

```
inv[(X;dI)'(X;dI)](X;dI)'(Y;0) = inv(X'X+d^2I)X'Y
```

which is the same as the jitter estimator B(d^2), i.e. jitter with variance d^2/n. The equivalence between the weight-decay estimator and the jitter estimator does *not* hold for nonlinear models unless the jitter variance is small relative to the curvature of the nonlinear function (An 1996). However, the equivalence of the two estimators for linear models suggests that they will often produce similar results even for nonlinear models. Details for nonlinear models, including classification problems, are given in An (1996).

B(0) is obviously the ordinary least-squares estimator. It can be shown that as s^2 increases, the Euclidean norm of B(ns^2) decreases; in other words, adding jitter causes the weights to shrink. It can also be shown that under the usual statistical assumptions, there always exists some value of ns^2 > 0 such that B(ns^2) provides better expected generalization than B(0). Unfortunately, there is no way to calculate a value of ns^2 from the training data that is guaranteed to improve generalization. There are other types of shrinkage estimators called Stein estimators that *do* guarantee better generalization than B(0), but I'm not aware of a nonlinear generalization of Stein estimators applicable to neural networks.

The statistics literature describes numerous methods for choosing the ridge value. The most obvious way is to estimate the generalization error by cross-validation, generalized cross-validation, or bootstrapping, and to choose the ridge value that yields the smallest such estimate. There are also quicker methods based on empirical Bayes estimation, one of which yields the following formula, useful as a first guess:

```
 2    p(Y-XB(0))'(Y-XB(0))
s   = --------------------
 1      n(n-p)B(0)'B(0)
```

You can iterate this a few times:

```
 2      p(Y-XB(0))'(Y-XB(0))
s     = --------------------
 1+1              2     2
```

$$\frac{n(n-p)}{p}B(s_l)'B(s_l)$$

Note that the more training cases you have, the less noise you need.

References:

An, G. (1996), "The effects of adding noise during backpropagation training on a generalization performance," Neural Computation, 8, 643-674.

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

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

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

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

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

Vinod, H.D. and Ullah, A. (1981) *Recent Advances in Regression Methods,* NY: Marcel-Dekker.

Subject: What is early stopping?

NN practitioners often use nets with many times as many parameters as training cases. E.g., Nelson and Illingworth (1991, p. 165) discuss training a network with 16,219 parameters with only 50 training cases! The method used is called "early stopping" or "stopped training" and proceeds as follows:

1. Divide the available data into training and validation sets.
2. Use a large number of hidden units.
3. Use very small random initial values.
4. Use a slow learning rate.
5. Compute the validation error rate periodically during training.
6. Stop training when the validation error rate "starts to go up".

It is crucial to realize that the validation error is *not* a good estimate of the generalization error. One method for getting an unbiased estimate of the generalization error is to run the net on a third set of data, the test set, that is not used at all during the training process. For other methods, see "How can generalization error be estimated?"

Early stopping has several advantages:

- It is fast.
- It can be applied successfully to networks in which the number of weights far exceeds the sample size.
- It requires only one major decision by the user: what proportion of validation cases to use.

But there are several unresolved practical issues in early stopping:

- How many cases do you assign to the training and validation sets? Rules of thumb abound, but appear to be no more than folklore. The only systematic results known to the FAQ maintainer are in Sarle (1995), which deals only with the case of a single input. Amari et al. (1995) attempts a theoretical approach but contains serious errors that completely invalidate the results, especially the incorrect assumption that the direction of approach to the optimum is distributed isotropically.
- Do you split the data into training and validation sets randomly or by some systematic algorithm?
- How do you tell when the validation error rate "starts to go up"? It may go up and down numerous times during training. The safest approach is to train to convergence, then go back and see which iteration had the lowest validation error. For more elaborate algorithms, see Prechelt (1994, 1998).

Statisticians tend to be skeptical of stopped training because it appears to be statistically inefficient due to the use of the split-sample technique; i.e., neither training nor validation makes use of the entire sample, and because the usual statistical theory does not apply. However, there has been recent progress addressing both of the above concerns (Wang 1994).

Early stopping is closely related to ridge regression. If the learning rate is sufficiently small, the sequence of weight vectors on each iteration will approximate the path of continuous steepest descent down the error surface. Early stopping chooses a point along this path that optimizes an estimate of the generalization error computed from the validation set. Ridge regression also defines a path of weight vectors by varying the ridge value. The ridge value is often chosen by optimizing an estimate of the generalization error computed by cross-validation, generalized cross-validation, or bootstrapping (see "What are cross-validation and bootstrapping?") There always exists a positive ridge value that will improve the expected generalization error in a linear model. A similar result has been obtained for early stopping in linear models (Wang, Venkatesh, and Judd 1994). In linear models, the ridge path lies close to, but does not coincide with, the path of continuous steepest descent; in nonlinear models, the two paths can diverge widely. The relationship is explored in more detail by Sjöberg and Ljung (1992).

References:

S. Amari, N.Murata, K.-R. Muller, M. Finke, H. Yang. Asymptotic Statistical Theory of Overtraining and Cross-Validation. METR 95-06, 1995, Department of Mathematical Engineering and Information Physics, University of Tokyo, Hongo 7-3-1, Bunkyo-ku, Tokyo 113, Japan.

Finnof, W., Hergert, F., and Zimmermann, H.G. (1993), "Improving model selection by nonconvergent methods," Neural Networks, 6, 771-783.

Nelson, M.C. and Illingworth, W.T. (1991), *A Practical Guide to Neural Nets*, Reading, MA: Addison-Wesley.

Orr, G.B., and Mueller, K.-R., eds. (1998), *Neural Networks: Tricks of the Trade,* Berlin: Springer, ISBN 3-540-65311-2.

Prechelt, L. (1998), "Early stopping--But when?" in Orr and Mueller (1998), 55-69.

Prechelt, L. (1994), "PROBEN1--A set of neural network benchmark problems and benchmarking rules," Technical Report 21/94, Universitat Karlsruhe, 76128 Karlsruhe, Germany, ftp://ftp.ira.uka.de/pub/papers/techreports/1994/1994-21.ps.gz.

Sarle, W.S. (1995), "Stopped Training and Other Remedies for Overfitting," *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics,* 352-360, ftp://ftp.sas.com/pub/neural/inter95.ps.Z (this is a *very large* compressed postscript file, *747K*, 10 pages)

Sjöberg, J. and Ljung, L. (1992), "Overtraining, Regularization, and Searching for Minimum in

Neural Networks," Technical Report LiTH-ISY-I-1297, Department of Electrical Engineering, Linkoping University, S-581 83 Linkoping, Sweden, http://www.control.isy.liu.se .

Wang, C. (1994), *A Theory of Generalisation in Learning Machines with Neural Network Application*, Ph.D. thesis, University of Pennsylvania.

Wang, C., Venkatesh, S.S., and Judd, J.S. (1994), "Optimal Stopping and Effective Machine Complexity in Learning," NIPS6, 303-310.

Weigend, A. (1994), "On overfitting and the effective number of hidden units," *Proceedings of the 1993 Connectionist Models Summer School*, 335-342.

Subject: What is weight decay?

Weight decay adds a penalty term to the error function. The usual penalty is the sum of squared weights times a decay constant. In a linear model, this form of weight decay is equivalent to ridge regression. See "What is jitter?" for more explanation of ridge regression.

Weight decay is a subset of regularization methods. The penalty term in weight decay, by definition, penalizes large weights. Other regularization methods may involve not only the weights but various derivatives of the output function (Bishop 1995).

The weight decay penalty term causes the weights to converge to smaller absolute values than they otherwise would. Large weights can hurt generalization in two different ways. Excessively large weights leading to hidden units can cause the output function to be too rough, possibly with near discontinuities. Excessively large weights leading to output units can cause wild outputs far beyond the range of the data if the output activation function is not bounded to the same range as the data. To put it another way, large weights can cause excessive variance of the output (Geman, Bienenstock, and Doursat 1992). According to Bartlett (1997), the size (L_1 norm) of the weights is more important than the number of weights in determining generalization.

Other penalty terms besides the sum of squared weights are sometimes used. *Weight elimination* (Weigend, Rumelhart, and Huberman 1991) uses:

```
          (w_i)^2
sum  --------------
 i   (w_i)^2 + c^2
```

where `w_i` is the `ith` weight and `c` is a user-specified constant. Whereas decay using the sum of squared weights tends to shrink the large coefficients more than the small ones, weight elimination tends to shrink the small coefficients more, and is therefore more useful for suggesting subset models (pruning).

The generalization ability of the network can depend crucially on the decay constant, especially with small training sets. One approach to choosing the decay constant is to train several networks with different amounts of decay and estimate the generalization error for each; then choose the decay constant that minimizes the estimated generalization error. Weigend, Rumelhart, and Huberman (1991) iteratively update the decay constant during training.

There are other important considerations for getting good results from weight decay. You must either standardize the inputs and targets, or adjust the penalty term for the standard deviations of all the inputs and targets. It is usually a good idea to omit the biases from the penalty term.

A fundamental problem with weight decay is that different types of weights in the network will usually require different decay constants for good generalization. At the very least, you need three different decay constants for input-to-hidden, hidden-to-hidden, and hidden-to-output weights. Adjusting all these decay constants to produce the best estimated generalization error often requires vast amounts of computation.

Fortunately, there is a superior alternative to weight decay: hierarchical Bayesian learning. Bayesian learning makes it possible to estimate efficiently numerous decay constants.

References:

Bartlett, P.L. (1997), "For valid generalization, the size of the weights is more important than the size of the network," in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) *Advances in Neural Information Processing Systems 9,* Cambrideg, MA: The MIT Press, pp. 134-140.

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

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

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

Weigend, A. S., Rumelhart, D. E., & Huberman, B. A. (1991). Generalization by weight-elimination with application to forecasting. In: R. P. Lippmann, J. Moody, & D. S. Touretzky (eds.), Advances in Neural Information Processing Systems 3, San Mateo, CA: Morgan Kaufmann.

Subject: What is Bayesian Learning?

By Radford Neal.

Conventional training methods for multilayer perceptrons ("backprop" nets) can be interpreted in statistical terms as variations on maximum likelihood estimation. The idea is to find a *single* set of weights for the network that maximize the fit to the training data, perhaps modified by some sort of weight penalty to prevent overfitting.

The Bayesian school of statistics is based on a different view of what it means to learn from data, in which probability is used to represent uncertainty about the relationship being learned (a use that is shunned in conventional--i.e., frequentist--statistics). Before we have seen any data, our *prior* opinions about what the true relationship might be can be expresssed in a probability distribution over the network weights that define this relationship. After we look at the data (or after our program looks at the data), our revised opinions are captured by a *posterior* distribution over network weights. Network weights that seemed plausible before, but which don't match the data very well, will now be seen as being much less likely, while the probability for values of the weights that do fit the data well will have increased.

Typically, the purpose of training is to make predictions for future cases in which only the inputs to the network are known. The result of conventional network training is a single set of weights that can be used to make such predictions. In contrast, the result of Bayesian training is a posterior *distribution* over network weights. If the inputs of the network are set to the values for some new case, the posterior distribution over network weights will give rise to a distribution over the outputs of the network, which is known as the *predictive distribution* for this new case. If a single-valued prediction is needed, one might use the mean of the predictive distribution, but the full predictive distribution also tells you how uncertain

this prediction is.

Why bother with all this? The hope is that Bayesian methods will provide solutions to such fundamental problems as:

- How to judge the uncertainty of predictions. This can be solved by looking at the predictive distribution, as described above.
- How to choose an appropriate network architecture (eg, the number hidden layers, the number of hidden units in each layer).
- How to adapt to the characteristics of the data (eg, the smoothness of the function, the degree to which different inputs are relevant).

Good solutions to these problems, especially the last two, depend on using the right prior distribution, one that properly represents the uncertainty that you probably have about which inputs are relevant, how smooth the function is, how much noise there is in the observations, etc. Such carefully vague prior distributions are usually defined in a hierarchical fashion, using *hyperparameters*, some of which are analogous to the weight decay constants of more conventional training procedures. The use of hyperparameters is discussed by Mackay (1992a, 1992b, 1995) and Neal (1993a, 1996), who in particular use an "Automatic Relevance Determination" scheme that aims to allow many possibly-relevant inputs to be included without damaging effects.

Selection of an appropriate network architecture is another place where prior knowledge plays a role. One approach is to use a very general architecture, with lots of hidden units, maybe in several layers or groups, controlled using hyperparameters. This approach is emphasized by Neal (1996), who argues that there is no statistical need to limit the complexity of the network architecture when using well-designed Bayesian methods. It is also possible to choose between architectures in a Bayesian fashion, using the "evidence" for an architecture, as discussed by Mackay (1992a, 1992b).

Implementing all this is one of the biggest problems with Bayesian methods. Dealing with a distribution over weights (and perhaps hyperparameters) is not as simple as finding a single "best" value for the weights. Exact analytical methods for models as complex as neural networks are out of the question. Two approaches have been tried:

1. Find the weights/hyperparameters that are most probable, using methods similar to conventional training (with regularization), and then approximate the distribution over weights using information available at this maximum.
2. Use a Monte Carlo method to sample from the distribution over weights. The most efficient implementations of this use dynamical Monte Carlo methods whose operation resembles that of backprop with momentum.

The first method comes in two flavours. Buntine and Weigend (1991) describe a procedure in which the hyperparameters are first integrated out analytically, and numerical methods are then used to find the most probable weights. MacKay (1992a, 1992b) instead finds the values for the hyperparameters that are most likely, integrating over the weights (using an approximation around the most probable weights, conditional on the hyperparameter values). There has been some controversy regarding the merits of these two procedures, with Wolpert (1993) claiming that analytically integrating over the hyperparameters is preferable because it is "exact". This criticism has been rebutted by Mackay (1993). It would be inappropriate to get into the details of this controversy here, but it is important to realize that the procedures based on analytical integration over the hyperparameters do *not* provide exact solutions to any of the problems of practical interest. The discussion of an analogous situation in a different statistical context by O'Hagan (1985) may be illuminating.

Monte Carlo methods for Bayesian neural networks have been developed by Neal (1993a, 1996). In this approach, the posterior distribution is represented by a sample of perhaps a few dozen sets of network weights. The sample is obtained by simulating a Markov chain whose equilibrium distribution is the posterior distribution for weights and hyperparameters. This technique is known as "Markov chain Monte Carlo (MCMC)"; see Neal (1993b) for a review. The method is exact in the limit as the size of the sample and the length of time for which the Markov chain is run increase, but convergence can sometimes be slow in practice, as for any network training method.

Work on Bayesian neural network learning has so far concentrated on multilayer perceptron networks, but Bayesian methods can in principal be applied to other network models, as long as they can be interpreted in statistical terms. For some models (eg, RBF networks), this should be a fairly simple matter; for others (eg, Boltzmann Machines), substantial computational problems would need to be solved.

Software implementing Bayesian neural network models (intended for research use) is available from the home pages of David MacKay and Radford Neal.

There are many books that discuss the general concepts of Bayesian inference, though they mostly deal with models that are simpler than neural networks. Here are some recent ones:

Bernardo, J. M. and Smith, A. F. M. (1994) *Bayesian Theory*, New York: John Wiley.

Gelman, A., Carlin, J.B., Stern, H.S., and Rubin, D.B. (1995) *Bayesian Data Analysis,* London: Chapman & Hall, ISBN 0-412-03991-5.

O'Hagan, A. (1994) *Bayesian Inference* (Volume 2B in Kendall's Advanced Theory of Statistics), ISBN 0-340-52922-9.

Robert, C. P. (1995) *The Bayesian Choice*, New York: Springer-Verlag.

The following books and papers have tutorial material on Bayesian learning as applied to neural network models:

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

Lee, H.K.H (1999), *Model Selection and Model Averaging for Neural Networks,* Doctoral dissertation, Carnegie Mellon University, Pittsburgh, USA, http://lib.stat.cmu.edu/~herbie/thesis.html

MacKay, D. J. C. (1995) "Probable networks and plausible predictions - a review of practical Bayesian methods for supervised neural networks", available at ftp://wol.ra.phy.cam.ac.uk/pub/www/mackay/network.ps.gz.

Mueller, P. and Insua, D.R. (1995) "Issues in Bayesian Analysis of Neural Network Models," Neural Computation, 10, 571-592, (also Institute of Statistics and Decision Sciences Working Paper 95-31), ftp://ftp.isds.duke.edu/pub/WorkingPapers/95-31.ps

Neal, R. M. (1996) *Bayesian Learning for Neural Networks,* New York: Springer-Verlag, ISBN 0-387-94724-8.

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

Thodberg, H. H. (1996) "A review of Bayesian neural networks with an application to near infrared spectroscopy", *IEEE Transactions on Neural Networks*, 7, 56-72.

Some other references:

Bernardo, J.M., DeGroot, M.H., Lindley, D.V. and Smith, A.F.M., eds., (1985), *Bayesian Statistics 2,* Amsterdam: Elsevier Science Publishers B.V. (North-Holland).

Buntine, W. L. and Weigend, A. S. (1991) "Bayesian back-propagation", *Complex Systems*, 5, 603-643.

MacKay, D. J. C. (1992a) "Bayesian interpolation", *Neural Computation*, 4, 415-447.

MacKay, D. J. C. (1992b) "A practical Bayesian framework for backpropagation networks," *Neural Computation*, 4, 448-472.

MacKay, D. J. C. (1993) "Hyperparameters: Optimize or Integrate Out?", available at ftp://wol.ra.phy.cam.ac.uk/pub/www/mackay/alpha.ps.gz.

Neal, R. M. (1993a) "Bayesian learning via stochastic dynamics", in C. L. Giles, S. J. Hanson, and J. D. Cowan (editors), *Advances in Neural Information Processing Systems 5*, San Mateo, California: Morgan Kaufmann, 475-482.

Neal, R. M. (1993b) *Probabilistic Inference Using Markov Chain Monte Carlo Methods*, available at ftp://ftp.cs.utoronto.ca/pub/radford/review.ps.Z.

O'Hagan, A. (1985) "Shoulders in hierarchical models", in J. M. Bernardo, M. H. DeGroot, D. V. Lindley, and A. F. M. Smith (editors), *Bayesian Statistics 2,* Amsterdam: Elsevier Science Publishers B.V. (North-Holland), 697-710.

Sarle, W. S. (1995) "Stopped Training and Other Remedies for Overfitting," *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, 352-360, ftp://ftp.sas.com/pub/neural/inter95.ps.Z (this is a *very large* compressed postscript file, *747K*, 10 pages)

Wolpert, D. H. (1993) "On the use of evidence in neural networks", in C. L. Giles, S. J. Hanson, and J. D. Cowan (editors), *Advances in Neural Information Processing Systems 5*, San Mateo, California: Morgan Kaufmann, 539-546.

Finally, David MacKay maintains a FAQ about Bayesian methods for neural networks, at http://wol.ra.phy.cam.ac.uk/mackay/Bayes_FAQ.html .

Comments on Bayesian learning

By Warren Sarle.

Bayesian purists may argue over the proper way to do a Bayesian analysis, but even the crudest Bayesian computation (maximizing over both parameters and hyperparameters) is shown by Sarle (1995) to generalize better than early stopping when learning nonlinear functions. This approach requires the use of slightly informative hyperpriors and at least twice as many training cases as weights in the network. A full Bayesian analysis by MCMC can be expected to work even better under even broader conditions. Bayesian learning works well by frequentist standards--what MacKay calls the "evidence framework" is used by frequentist statisticians under the name "empirical Bayes." Although considerable research remains to be done, Bayesian learning seems to be the most promising approach to training neural networks.

Bayesian learning should not be confused with the "Bayes classifier." In the latter, the distribution of the

inputs given the target class is assumed to be known exactly, and the prior probabilities of the classes are assumed known, so that the posterior probabilities can be computed by a (theoretically) simple application of Bayes' theorem. The Bayes classifier involves no learning--you must already know everything that needs to be known! The Bayes classifier is a gold standard that can almost never be used in real life but is useful in theoretical work and in simulation studies that compare classification methods. The term "Bayes rule" is also used to mean any classification rule that gives results identical to those of a Bayes classifier.

Bayesian learning also should not be confused with the "naive" or "idiot's" Bayes classifier (Warner et al. 1961; Ripley, 1996), which assumes that the inputs are conditionally independent given the target class. The naive Bayes classifier is usually applied with categorical inputs, and the distribution of each input is estimated by the proportions in the training set; hence the naive Bayes classifier is a frequentist method.

The term "Bayesian network" often refers not to a neural network but to a belief network (also called a causal net, influence diagram, constraint network, qualitative Markov network, or gallery). Belief networks are more closely related to expert systems than to neural networks, and do not necessarily involve learning (Pearl, 1988; Ripley, 1996). Here are some URLs on Bayesian belief networks:

- http://bayes.stat.washington.edu/almond/belief.html
- http://www.cs.orst.edu/~dambrosi/bayesian/frame.html
- http://www2.sis.pitt.edu/~genie
- http://www.research.microsoft.com/dtg/msbn

References for comments:

Pearl, J. (1988) *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference,* San Mateo, CA: Morgan Kaufmann.

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

Warner, H.R., Toronto, A.F., Veasy, L.R., and Stephenson, R. (1961), "A mathematical model for medical diagnosis--application to congenital heart disease," J. of the American Medical Association, 177, 177-184.

Subject: How to combine networks?

Methods for combining networks are a subject of active research. Many different methods with different purposes have been proposed. The properties and relationships of these methods are just beginning to be understood. Some methods, such as boosting, are remedies for underfitting. Other methods, such as bagging, are mainly remedies for overfitting or instability. Bayesian learning naturally leads to model averaging (Hoeting et al., 1999). A good general reference is the book edited by Sharkey (1999), especially the article by Drucker (1999). Regarding the effects of bagging and weight decay used together, see Taniguchi and Tresp (1997).

Here is a list of terms used for various methods of combining models, mostly taken from Christoph M. Friedrich's web page (see below):

- Adaboost
- ADDEMUP
- arcing: adaptive recombination of classifiers

- bagging: bootstrap aggregation
- bag-stacking: bagging plus stacking
- boosting
- cascading
- combination of classifiers
- committees of networks
- consensus theory
- cragging: cross aggregation (like k-fold cross validation)
- dagging: disjoint-sample aggregation
- dag-stacking: dagging plus stacking
- divide and conquer classifiers
- ensembles
- hagging: half-sample aggregation
- mixture of experts
- multiple classifier systems:
- multi-stage and multi-level classifiers
- OLC: optimal linear combination
- pandemonium of reflective agents
- sieving algorithms
- stacking: feeding outputs of several models (and possibly the the original inputs) into a second-level model
- voting

URLs:

- Christoph M. Friedrich's web page, "Combinations of Classifiers and Regressors Bibliography and Guide to Internet Resources" at http://www.tussy.uni-wh.de/~chris/ensemble/ensemble.html
- Tirthankar RayChaudhuri's web page on combining estimators at http://www-comp.mpce.mq.edu.au/~tirthank/combest.html
- Robert E. Schapire's boosting page at http://www.research.att.com/~schapire/boost.html
- http://www.boosting.org/

References:

Clemen, Robert T. (1989), "Combining forecasts: A review and annotated bibliography", International Journal of Forecasting, Vol 5, pp 559-584.

Drucker, H. (1999), "Boosting using neural networks," in Sharkey (1999), pp. 51-78.

Hoeting, J. A., Madigan, D., Raftery, A.E., and Volinsky, C.T. (1999) "Bayesian Model Averaging: A Tutorial (with discussion)," Statistical Science, 14:4, 382-417. Corrected version available at http://www.stat.washington.edu/www/research/online/hoeting1999.pdf

Sharkey, A.J.C. (1999), *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems,* London: Springer.

Taniguchi, M., and Tresp, V. (1997), "Averaging regularized estimators," Neural Computation, 9, 1163-1178.

Subject: How many hidden layers should I use?

You may not need any hidden layers at all. Linear and generalized linear models are useful in a wide variety of applications (McCullagh and Nelder 1989). And even if the function you want to learn is mildly nonlinear, you may get better generalization with a simple linear model than with a complicated nonlinear model if there is too little data or too much noise to estimate the nonlinearities accurately.

In MLPs with step/threshold/Heaviside activation functions, you need two hidden layers for full generality (Sontag 1992). For further discussion, see Bishop (1995, 121-126).

In MLPs with any of a wide variety of continuous nonlinear hidden-layer activation functions, one hidden layer with an arbitrarily large number of units suffices for the "universal approximation" property (e.g., Hornik, Stinchcombe and White 1989; Hornik 1993; for more references, see Bishop 1995, 130, and Ripley, 1996, 173-180). But there is no theory yet to tell you how many hidden units are needed to approximate any given function.

If you have only one input, there seems to be no advantage to using more than one hidden layer. But things get much more complicated when there are two or more inputs. To illustrate, examples with two inputs and one output will be used so that the results can be shown graphically. In each example there are 441 training cases on a regular 21-by-21 grid. The test sets have 1681 cases on a regular 41-by-41 grid over the same domain as the training set. If you are reading the HTML version of this document via a web browser, you can see surface plots based on the test set by clicking on the file names mentioned in the folowing text. Each plot is a gif file, approximately 9K in size.

Consider a target function of two inputs, consisting of a Gaussian hill in the middle of a plane (hill.gif). An MLP with an identity output activation function can easily fit the hill by surrounding it with a few sigmoid (logistic, tanh, arctan, etc.) hidden units, but there will be spurious ridges and valleys where the plane should be flat (h_mlp_6.gif). It takes dozens of hidden units to flatten out the plane accurately (h_mlp_30.gif).

Now suppose you use a logistic output activation function. As the input to a logistic function goes to negative infinity, the output approaches zero. The plane in the Gaussian target function also has a value of zero. If the weights and bias for the output layer yield large negative values outside the base of the hill, the logistic function will flatten out any spurious ridges and valleys. So fitting the flat part of the target function is easy (h_mlpt_3_unsq.gif and h_mlpt_3.gif). But the logistic function also tends to lower the top of the hill.

If instead of a rounded hill, the target function was a mesa with a large, flat top with a value of one, the logistic output activation function would be able to smooth out the top of the mesa just like it smooths out the plane below. Target functions like this, with large flat areas with values of either zero or one, are just what you have in many noise-free classificaton problems. In such cases, a single hidden layer is likely to work well.

When using a logistic output activation function, it is common practice to scale the target values to a range of .1 to .9. Such scaling is bad in a noise-free classificaton problem, because it prevents the logistic function from smoothing out the flat areas (h_mlpt1-9_3.gif).

For the Gaussian target function, [.1,.9] scaling would make it easier to fit the top of the hill, but would reintroduce undulations in the plane. It would be better for the Gaussian target function to scale the target values to a range of 0 to .9. But for a more realistic and complicated target function, how would you know the best way to scale the target values?

By introducing a second hidden layer with one sigmoid activation function and returning to an identity output activation function, you can let the net figure out the best scaling (h_mlp1_3.gif). Actually, the bias and weight for the output layer scale the output rather than the target values, and you can use whatever range of target values is convenient.

For more complicated target functions, especially those with several hills or valleys, it is useful to have several units in the second hidden layer. Each unit in the second hidden layer enables the net to fit a separate hill or valley. So an MLP with two hidden layers can often yield an accurate approximation with fewer weights than an MLP with one hidden layer. (Chester 1990).

To illustrate the use of multiple units in the second hidden layer, the next example resembles a landscape with a Gaussian hill and a Gaussian valley, both elliptical (hillanvale.gif). The table below gives the RMSE (root mean squared error) for the test set with various architectures. If you are reading the HTML version of this document via a web browser, click on any number in the table to see a surface plot of the corresponding network output.

The MLP networks in the table have one or two hidden layers with a tanh activation function. The output activation function is the identity. Using a squashing function on the output layer is of no benefit for this function, since the only flat area in the function has a target value near the middle of the target range.

Hill and Valley Data: RMSE for the Test Set
(Number of weights in parentheses)
MLP Networks

HUs in First Layer	HUs in Second Layer				
	0	1	2	3	4
1	0.204(5)	0.204(7)	0.189(10)	0.187(13)	0.185(16)
2	0.183(9)	0.163(11)	0.147(15)	0.094(19)	0.096(23)
3	0.159(13)	0.095(15)	0.054(20)	0.033(25)	0.045(30)
4	0.137(17)	0.093(19)	0.009(25)	0.021(31)	0.016(37)
5	0.121(21)	0.092(23)		0.010(37)	0.011(44)
6	0.100(25)	0.092(27)		0.007(43)	0.005(51)
7	0.086(29)	0.077(31)			
8	0.079(33)	0.062(35)			
9	0.072(37)	0.055(39)			
10	0.059(41)	0.047(43)			
12	0.047(49)	0.042(51)			
15	0.039(61)	0.032(63)			
20	0.025(81)	0.018(83)			
25	0.021(101)	0.016(103)			
30	0.018(121)	0.015(123)			
40	0.012(161)	0.015(163)			
50	0.008(201)	0.014(203)			

For an MLP with only one hidden layer (column 0), Gaussian hills and valleys require a large number of hidden units to approximate well. When there is one unit in the second hidden layer, the network can represent one hill or valley easily, which is what happens with three to six units in the first hidden layer. But having only one unit in the second hidden layer is of little benefit for learning two hills or valleys. Using two units in the second hidden layer enables the network to approximate two hills or valleys easily; in this example, only four units are required in the first hidden layer to get an excellent fit. Each additional unit in the second hidden layer enables the network to learn another hill or valley with a relatively small number of units in the first hidden layer, as explained by Chester (1990). In this example, having three or four units in the second hidden layer helps little, and actually produces a worse approximation when there are four units in the first hidden layer due to problems with local minima.

Unfortunately, using two hidden layers exacerbates the problem of local minima, and it is important to use lots of random initializations or other methods for global optimization. Local minima with two hidden

layers can have extreme spikes or blades even when the number of weights is much smaller than the number of training cases. One of the few advantages of standard backprop is that it is so slow that spikes and blades will not become very sharp for practical training times.

More than two hidden layers can be useful in certain architectures such as cascade correlation (Fahlman and Lebiere 1990) and in special applications, such as the two-spirals problem (Lang and Witbrock 1988) and ZIP code recognition (Le Cun et al. 1989).

RBF networks are most often used with a single hidden layer. But an extra, linear hidden layer before the radial hidden layer enables the network to ignore irrelevant inputs (see How do MLPs compare with RBFs?") The linear hidden layer allows the RBFs to take elliptical, rather than radial (circular), shapes in the space of the inputs. Hence the linear layer gives you an elliptical basis function (EBF) network. In the hill and valley example, an ORBFUN network requires nine hidden units (37 weights) to get the test RMSE below .01, but by adding a linear hidden layer, you can get an essentially perfect fit with three linear units followed by two radial units (20 weights).

References:

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

Chester, D.L. (1990), "Why Two Hidden Layers are Better than One," IJCNN-90-WASH-DC, Lawrence Erlbaum, 1990, volume 1, 265-268.

Fahlman, S.E. and Lebiere, C. (1990), "The Cascade Correlation Learning Architecture," NIPS2, 524-532, ftp://archive.cis.ohio-state.edu/pub/neuroprose/fahlman.cascor-tr.ps.Z.

Hornik, K., Stinchcombe, M. and White, H. (1989), "Multilayer feedforward networks are universal approximators," Neural Networks, 2, 359-366.

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

Lang, K.J. and Witbrock, M.J. (1988), "Learning to tell two spirals apart," in Touretzky, D., Hinton, G., and Sejnowski, T., eds., *Procedings of the 1988 Connectionist Models Summer School,* San Mateo, CA: Morgan Kaufmann.

Le Cun, Y., Boser, B., Denker, J.s., Henderson, D., Howard, R.E., Hubbard, W., and Jackel, L.D. (1989), "Backpropagation applied to handwritten ZIP code recognition", Neural Computation, 1, 541-551.

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

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

Sontag, E.D. (1992), "Feedback stabilization using two-hidden-layer nets", IEEE Transactions on Neural Networks, 3, 981-990.

Subject: How many hidden units should I use?

The best number of hidden units depends in a complex way on:

- the numbers of input and output units
- the number of training cases
- the amount of noise in the targets
- the complexity of the function or classification to be learned
- the architecture
- the type of hidden unit activation function
- the training algorithm
- regularization

In most situations, there is no way to determine the best number of hidden units without training several networks and estimating the generalization error of each. If you have too few hidden units, you will get high training error and high generalization error due to underfitting and high statistical bias. If you have too many hidden units, you may get low training error but still have high generalization error due to overfitting and high variance. Geman, Bienenstock, and Doursat (1992) discuss how the number of hidden units affects the bias/variance trade-off.

Some books and articles offer "rules of thumb" for choosing an architecture; for example:

- "A rule of thumb is for the size of this [hidden] layer to be somewhere between the input layer size ... and the output layer size ..." (Blum, 1992, p. 60).
- "To calculate the number of hidden nodes we use a general rule of: (Number of inputs + outputs) * (2/3)" (from the FAQ for a commercial neural network software company).
- "you will never require more than twice the number of hidden units as you have inputs" in an MLP with one hidden layer (Swingler, 1996, p. 53). See the section in Part 4 of the FAQ on The Worst books for the source of this myth.)
- "How large should the hidden layer be? One rule of thumb is that it should never be more than twice as large as the input layer." (Berry and Linoff, 1997, p. 323).
- "Typically, we specify as many hidden nodes as dimensions [principal components] needed to capture 70-90% of the variance of the input data set." (Boger and Guterman, 1997)

These rules of thumb are nonsense because they ignore the number of training cases, the amount of noise in the targets, and the complexity of the function. Even if you restrict consideration to minimizing training error on data with lots of training cases and no noise, it is easy to construct counterexamples that disprove these rules of thumb. For example:

- There are 100 Boolean inputs and 100 Boolean targets. Each target is a conjunction of some subset of inputs. No hidden units are needed.

- There are two continuous inputs X and Y which take values uniformly distributed on a square [0,8] by [0,8]. Think of the input space as a chessboard, and number the squares 1 to 64. The categorical target variable C is the square number, so there are 64 output units. For example, you could generate the data as follows (this is the SAS programming language, but it should be easy to translate into any other language):

```
data chess;
   step = 1/4;
   do x = step/2 to 8-step/2 by step;
      do y = step/2 to 8-step/2 by step;
         c = 8*floor(x) + floor(y) + 1;
         output;
      end;
   end;
run;
```

  No hidden units are needed.

- The classic two-spirals problem has two continuous inputs and a Boolean classification target. The data can be generated as follows:

```
data spirals;
   pi = arcos(-1);
   do i = 0 to 96;
      angle = i*pi/16.0;
      radius = 6.5*(104-i)/104;
      x = radius*cos(angle);
      y = radius*sin(angle);
      c = 1;
      output;
      x = -x;
      y = -y;
      c = 0;
      output;
   end;
run;
```

  With one hidden layer, about 50 tanh hidden units are needed. Many NN programs may actually need closer to 100 hidden units to get zero training error.

- There is one continuous input X that takes values on [0,100]. There is one continuous target Y = sin(X). Getting a good approximation to Y requires about 20 to 25 tanh hidden units. Of course, 1 sine hidden unit would do the job.

Some rules of thumb relate the total number of trainable weights in the network to the number of training cases. A typical recommendation is that the number of weights should be no more than 1/30 of the number of training cases. Such rules are only concerned with overfitting and are at best crude approximations. Also, these rules do not apply when regularization is used. It is true that without regularization, if the number of training cases is *much* larger (but no one knows exactly *how* much larger) than the number of weights, you are unlikely to get overfitting, but you may suffer from underfitting. For a noise-free quantitative target variable, twice as many training cases as weights may be more than enough to avoid overfitting. For a very noisy categorical target variable, 30 times as many training cases as weights may not be enough to avoid overfitting.

An intelligent choice of the number of hidden units depends on whether you are using early stopping or some other form of regularization. If not, you must simply try many networks with different numbers of hidden units, estimate the generalization error for each one, and choose the network with the minimum estimated generalization error. For examples using statistical criteria to choose the number of hidden units, see ftp://ftp.sas.com/pub/neural/dojo/dojo.html.

Using conventional optimization algorithms (see "What are conjugate gradients, Levenberg-Marquardt, etc.?"), there is little point in trying a network with more weights than training cases, since such a large network is likely to overfit.

Using standard online backprop, however, Lawrence, Giles, and Tsoi (1996, 1997) have shown that it can be difficult to reduce training error to a level near the globally optimal value, even when using more weights than training cases. But increasing the number of weights makes it easier for standard backprop to find a good local optimum, so using "oversize" networks can reduce both training error and generalization error.

If you are using early stopping, it is essential to use *lots* of hidden units to avoid bad local optima (Sarle 1995). There seems to be no upper limit on the number of hidden units, other than that imposed by computer time and memory requirements. Weigend (1994) makes this assertion, but provides only one example as evidence. Tetko, Livingstone, and Luik (1995) provide simulation studies that are more convincing. Similar results were obtained in conjunction with the simulations in Sarle (1995), but those results are not reported in the paper for lack of space. On the other hand, there seems to be no advantage to using more hidden units than you have training cases, since bad local minima do not occur with so many hidden units.

If you are using weight decay or Bayesian estimation, you can also use lots of hidden units (Neal 1996). However, it is not strictly necessary to do so, because other methods are available to avoid local minima, such as multiple random starts and simulated annealing (such methods are not safe to use with early stopping). You can use one network with lots of hidden units, or you can try different networks with different numbers of hidden units, and choose on the basis of estimated generalization error. With weight decay or MAP Bayesian estimation, it is prudent to keep the number of weights less than half the number of training cases.

Bear in mind that with two or more inputs, an MLP with one hidden layer containing only a few units can fit only a limited variety of target functions. Even simple, smooth surfaces such as a Gaussian bump in two dimensions may require 20 to 50 hidden units for a close approximation. Networks with a smaller number of hidden units often produce spurious ridges and valleys in the output surface (see Chester 1990 and "How do MLPs compare with RBFs?") Training a network with 20 hidden units will typically require anywhere from 150 to 2500 training cases if you do not use early stopping or regularization. Hence, if you have a smaller training set than that, it is usually advisable to use early stopping or regularization rather than to restrict the net to a small number of hidden units.

Ordinary RBF networks containing only a few hidden units also produce peculiar, bumpy output functions. Normalized RBF networks are better at approximating simple smooth surfaces with a small number of hidden units (see How do MLPs compare with RBFs?).

There are various theoretical results on how fast approximation error decreases as the number of hidden units increases, but the conclusions are quite sensitive to the assumptions regarding the function you are trying to approximate. See p. 178 in Ripley (1996) for a summary. According to a well-known result by Barron (1993), in a network with `I` inputs and `H` units in a single hidden layer, the root integrated squared error (RISE) will decrease at least as fast as `H^(-1/2)` under some quite complicated smoothness assumptions. Ripley cites another paper by DeVore et al. (1989) that says if the function has only `R` bounded derivatives, RISE may decrease as slowly as `H^(-R/I)`. For some examples with from 1 to 4 hidden layers see How many hidden layers should I use?" and "How do MLPs compare with RBFs?"

For learning a finite training set exactly, bounds for the number of hidden units required are provided by Elisseeff and Paugam-Moisy (1997).

References:

Barron, A.R. (1993), "Universal approximation bounds for superpositions of a sigmoid function," IEEE Transactions on Information Theory, 39, 930-945.

Berry, M.J.A., and Linoff, G. (1997), *Data Mining Techniques,* NY: John Wiley & Sons.

Blum, A. (1992), *Neural Networks in C++*, NY: Wiley.

Boger, Z., and Guterman, H. (1997), "Knowledge extraction from artificial neural network models," IEEE Systems, Man, and Cybernetics Conference, Orlando, FL.

Chester, D.L. (1990), "Why Two Hidden Layers are Better than One," IJCNN-90-WASH-DC, Lawrence Erlbaum, 1990, volume 1, 265-268.

DeVore, R.A., Howard, R., and Micchelli, C.A. (1989), "Optimal nonlinear approximation," Manuscripta Mathematica, 63, 469-478.

Elisseeff, A., and Paugam-Moisy, H. (1997), "Size of multilayer networks for exact learning: analytic approach," in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) *Advances in Neural Information Processing Systems 9,* Cambrideg, MA: The MIT Press, pp.162-168.

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

Lawrence, S., Giles, C.L., and Tsoi, A.C. (1996), "What size neural network gives optimal generalization? Convergence properties of backpropagation," Technical Report UMIACS-TR-96-22 and CS-TR-3617, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, http://www.neci.nj.nec.com/homepages/lawrence/papers/minima-tr96/minima-tr96.html

Lawrence, S., Giles, C.L., and Tsoi, A.C. (1997), "Lessons in Neural Network Training: Overfitting May be Harder than Expected," Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI-97, AAAI Press, Menlo Park, California, pp. 540-545, http://www.neci.nj.nec.com/homepages/lawrence/papers/overfitting-aaai97/overfitting-aaai97-bib.html

Neal, R. M. (1996) *Bayesian Learning for Neural Networks,* New York: Springer-Verlag, ISBN 0-387-94724-8.

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

Sarle, W.S. (1995), "Stopped Training and Other Remedies for Overfitting," *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics,* 352-360, ftp://ftp.sas.com/pub/neural/inter95.ps.Z (this is a *very large* compressed postscript file, *747K*, 10 pages)

Swingler, K. (1996), *Applying Neural Networks: A Practical Guide,* London: Academic Press.

Tetko, I.V., Livingstone, D.J., and Luik, A.I. (1995), "Neural Network Studies. 1. Comparison of Overfitting and Overtraining," J. Chem. Info. Comp. Sci., 35, 826-833.

Weigend, A. (1994), "On overfitting and the effective number of hidden units," *Proceedings of the 1993 Connectionist Models Summer School*, 335-342.

Subject: How can generalization error be estimated?

There are many methods for estimating generalization error.

Single-sample statistics: AIC, SBC, MDL, FPE, Mallows' C_p, etc.

In linear models, statistical theory provides several simple estimators of the generalization error under various sampling assumptions (Darlington 1968; Efron and Tibshirani 1993; Miller 1990). These estimators adjust the training error for the number of weights being estimated, and in some cases for the noise variance if that is known.

These statistics can also be used as crude estimates of the generalization error in nonlinear models when you have a "large" training set. Correcting these statistics for nonlinearity requires substantially more computation (Moody 1992), and the theory does not always hold for neural networks due to violations of the regularity conditions.

Among the simple generalization estimators that do not require the noise variance to be known, Schwarz's Bayesian Criterion (known as both SBC and BIC; Schwarz 1978; Judge et al. 1980; Raftery 1995) often works well for NNs (Sarle 1995, 1999). AIC and FPE tend to overfit with NNs. Rissanen's Minimum Description Length principle (MDL; Rissanen 1978, 1987, 1999) is closely related to SBC. A special issue of *Computer Journal* contains several articles on MDL, which can be found online at http://www3.oup.co.uk/computer_journal/hdb/Volume_42/Issue_04/
Several articles on SBC/BIC are available at the University of Washigton's web site at http://www.stat.washington.edu/tech.reports

For classification problems, the formulas are not as simple as for regression with normal noise. See Efron (1986) regarding logistic regression.

Split-sample or hold-out validation.

The most commonly used method for estimating generalization error in neural networks is to reserve part of the data as a "test" set, which must not be used in *any* way during training. The test set must be a representative sample of the cases that you want to generalize to. After training, run the network on the test set, and the error on the test set provides an unbiased estimate of the generalization error, provided that the test set was chosen randomly. The disadvantage of split-sample validation is that it reduces the amount of data available for both training and validation. See Weiss and Kulikowski (1991).

Cross-validation (e.g., leave one out).

Cross-validation is an improvement on split-sample validation that allows you to use all of the data for training. The disadvantage of cross-validation is that you have to retrain the net many times. See "What are cross-validation and bootstrapping?".

Bootstrapping.

Bootstrapping is an improvement on cross-validation that often provides better estimates of generalization error at the cost of even more computing time. See "What are cross-validation and bootstrapping?".

If you use any of the above methods to choose which of several different networks to use for prediction purposes, the estimate of the generalization error of the best network will be optimistic. For example, if you train several networks using one data set, and use a second (validation set) data set to decide which network is best, you must use a third (test set) data set to obtain an unbiased estimate of the generalization

error of the chosen network. Hjorth (1994) explains how this principle extends to cross-validation and bootstrapping.

References:

Darlington, R.B. (1968), "Multiple Regression in Psychological Research and Practice," Psychological Bulletin, 69, 161-182.

Efron, B. (1986), "How biased is the apparent error rate of a prediction rule?" J. of the American Statistical Association, 81, 461-470.

Efron, B. and Tibshirani, R.J. (1993), *An Introduction to the Bootstrap,* London: Chapman & Hall.

Hjorth, J.S.U. (1994), *Computer Intensive Statistical Methods: Validation, Model Selection, and Bootstrap,* London: Chapman & Hall.

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

Moody, J.E. (1992), "The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems", in Moody, J.E., Hanson, S.J., and Lippmann, R.P., *Advances in Neural Information Processing Systems 4,* 847-854.

Raftery, A.E. (1995), "Bayesian Model Selection in Social Research," in Marsden, P.V. (ed.), *Sociological Methodology 1995,* Cambridge, MA: Blackwell, ftp://ftp.stat.washington.edu/pub/tech.reports/bic.ps.z or http://www.stat.washington.edu/tech.reports/bic.ps

Rissanen, J. (1978), "Modelling by shortest data description," Automatica, 14, 465-471.

Rissanen, J. (1987), "Stochastic complexity" (with discussion), J. of the Royal Statistical Society, Series B, 49, 223-239.

Rissanen, J. (1999), "Hypothesis Selection and Testing by the MDL Principle," Computer Journal, 42, 260-269, http://www3.oup.co.uk/computer_journal/hdb/Volume_42/Issue_04/

Sarle, W.S. (1995), "Stopped Training and Other Remedies for Overfitting," *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics,* 352-360, ftp://ftp.sas.com/pub/neural/inter95.ps.Z (this is a *very large* compressed postscript file, *747K*, 10 pages)

Sarle, W.S. (1999), "Donoho-Johnstone Benchmarks: Neural Net Results," ftp://ftp.sas.com/pub/neural/dojo/dojo.html

Weiss, S.M. & Kulikowski, C.A. (1991), *Computer Systems That Learn,* Morgan Kaufmann.

Subject: What are cross-validation and bootstrapping?

Cross-validation and bootstrapping are both methods for estimating generalization error based on "resampling" (Weiss and Kulikowski 1991; Efron and Tibshirani 1993; Hjorth 1994; Plutowski, Sakata, and White 1994; Shao and Tu 1995). The resulting estimates of generalization error are often used for choosing among various models, such as different network architectures.

Cross-validation

In k-fold cross-validation, you divide the data into k subsets of (approximately) equal size. You train the net k times, each time leaving out one of the subsets from training, but using only the omitted subset to compute whatever error criterion interests you. If k equals the sample size, this is called "leave-one-out" cross-validation. "Leave-v-out" is a more elaborate and expensive version of cross-validation that involves leaving out all possible subsets of v cases.

Note that cross-validation is quite different from the "split-sample" or "hold-out" method that is commonly used for early stopping in NNs. In the split-sample method, only a single subset (the validation set) is used to estimate the generalization error, instead of k different subsets; i.e., there is no "crossing". While various people have suggested that cross-validation be applied to early stopping, the proper way of doing so is not obvious.

The distinction between cross-validation and split-sample validation is extremely important because cross-validation is markedly superior for small data sets; this fact is demonstrated dramatically by Goutte (1997) in a reply to Zhu and Rohwer (1996). For an insightful discussion of the limitations of cross-validatory choice among several learning methods, see Stone (1977).

Jackknifing

Leave-one-out cross-validation is also easily confused with jackknifing. Both involve omitting each training case in turn and retraining the network on the remaining subset. But cross-validation is used to estimate generalization error, while the jackknife is used to estimate the bias of a statistic. In the jackknife, you compute some statistic of interest in each subset of the data. The average of these subset statistics is compared with the corresponding statistic computed from the entire sample in order to estimate the bias of the latter. You can also get a jackknife estimate of the standard error of a statistic. Jackknifing can be used to estimate the bias of the training error and hence to estimate the generalization error, but this process is more complicated than leave-one-out cross-validation (Efron, 1982; Ripley, 1996, p. 73).

Choice of cross-validation method

Cross-validation can be used simply to estimate the generalization error of a given model, or it can be used for model selection by choosing one of several models that has the smallest estimated generalization error. For example, you might use cross-validation to choose the number of hidden units, or you could use cross-validation to choose a subset of the inputs (subset selection). A subset that contains all relevant inputs will be called a "good" subsets, while the subset that contains all relevant inputs but no others will be called the "best" subset. Note that subsets are "good" and "best" in an asymptotic sense (as the number of training cases goes to infinity). With a small training set, it is possible that a subset that is smaller than the "best" subset may provide better generalization error.

Leave-one-out cross-validation often works well for estimating generalization error for continuous error functions such as the mean squared error, but it may perform poorly for discontinuous error functions such as the number of misclassified cases. In the latter case, k-fold cross-validation is preferred. But if k gets too small, the error estimate is pessimistically biased because of the difference in training-set size between the full-sample analysis and the cross-validation analyses. (For model-selection purposes, this bias can actually help; see the discussion below of Shao, 1993.) A value of 10 for k is popular for estimating generalization error.

Leave-one-out cross-validation can also run into trouble with various model-selection methods. Again, one problem is lack of continuity--a small change in the data can cause a large change in the model selected (Breiman, 1996). For choosing subsets of inputs in linear regression, Breiman and Spector (1992) found 10-fold and 5-fold cross-validation to work better than leave-one-out. Kohavi (1995) also obtained

good results for 10-fold cross-validation with empirical decision trees (C4.5). Values of k as small as 5 or even 2 may work even better if you analyze several different random k-way splits of the data to reduce the variability of the cross-validation estimate.

Leave-one-out cross-validation also has more subtle deficiencies for model selection. Shao (1995) showed that in linear models, leave-one-out cross-validation is asymptotically equivalent to AIC (and Mallows' C_p), but leave-v-out cross-validation is asymptotically equivalent to Schwarz's Bayesian criterion (called SBC or BIC) when v = n[1-1/(log(n)-1)], where n is the number of training cases. SBC provides consistent subset-selection, while AIC does not. That is, SBC will choose the "best" subset with probability approaching one as the size of the training set goes to infinity. AIC has an asymptotic probability of one of choosing a "good" subset, but less than one of choosing the "best" subset (Stone, 1979). Many simulation studies have also found that AIC overfits badly in small samples, and that SBC works well (e.g., Hurvich and Tsai, 1989; Shao and Tu, 1995). Hence, these results suggest that leave-one-out cross-validation should overfit in small samples, but leave-v-out cross-validation with appropriate v should do better. However, when true models have an infinite number of parameters, SBC is not efficient, and other criteria that are asymptotically efficient but not consistent for model selection may produce better generalization (Hurvich and Tsai, 1989).

Shao (1993) obtained the surprising result that for selecting subsets of inputs in a linear regression, the probability of selecting the "best" does not converge to 1 (as the sample size n goes to infinity) for leave-v-out cross-validation unless the proportion v/n approaches 1. At first glance, Shao's result seems inconsistent with the analysis by Kearns (1997) of split-sample validation, which shows that the best generalization is obtained with v/n strictly between 0 and 1, with little sensitivity to the precise value of v/n for large data sets. But the apparent conflict is due to the fundamentally different properties of cross-validation and split-sample validation.

To obtain an intuitive understanding of Shao (1993), let's review some background material on generalization error. Generalization error can be broken down into three additive parts, noise variance + estimation variance + squared estimation bias. Noise variance is the same for all subsets of inputs. Bias is nonzero for subsets that are not "good", but it's zero for all "good" subsets, since we are assuming that the function to be learned is linear. Hence the generalization error of "good" subsets will differ only in the estimation variance. The estimation variance is (2p/t)s^2 where p is the number of inputs in the subset, t is the training set size, and s^2 is the noise variance. The "best" subset is better than other "good" subsets only because the "best" subset has (by definition) the smallest value of p. But the t in the denominator means that differences in generalization error among the "good" subsets will all go to zero as t goes to infinity. Therefore it is difficult to guess which subset is "best" based on the generalization error even when t is very large. It is well known that unbiased estimates of the generalization error, such as those based on AIC, FPE, and C_p, do not produce consistent estimates of the "best" subset (e.g., see Stone, 1979).

In leave-v-out cross-validation, t=n-v. The differences of the cross-validation estimates of generalization error among the "good" subsets contain a factor 1/t, not 1/n. Therefore by making t small enough (and thereby making each regression based on t cases bad enough), we can make the differences of the cross-validation estimates large enough to detect. It turns out that to make t small enough to guess the "best" subset consistently, we have to have t/n go to 0 as n goes to infinity.

The crucial distinction between cross-validation and split-sample validation is that with cross-validation, after guessing the "best" subset, we train the linear regression model for that subset using all n cases, but with split-sample validation, only t cases are ever used for training. If our main purpose were really to choose the "best" subset, I suspect we would still have to have t/n go to 0 even for split-sample validation. But choosing the "best" subset is not the same thing as getting the best generalization. If we are more interested in getting good generalization than in choosing the "best" subset, we do not want to

make our regression estimate based on only `t` cases as bad as we do in cross-validation, because in split-sample validation that bad regression estimate is what we're stuck with. So there is no conflict between Shao and Kearns, but there is a conflict between the two goals of choosing the "best" subset and getting the best generalization in split-sample validation.

Bootstrapping

Bootstrapping seems to work better than cross-validation in many cases (Efron, 1983). In the simplest form of bootstrapping, instead of repeatedly analyzing subsets of the data, you repeatedly analyze subsamples of the data. Each subsample is a random sample with replacement from the full sample. Depending on what you want to do, anywhere from 50 to 2000 subsamples might be used. There are many more sophisticated bootstrap methods that can be used not only for estimating generalization error but also for estimating confidence bounds for network outputs (Efron and Tibshirani 1993). For estimating generalization error in classification problems, the .632+ bootstrap (an improvement on the popular .632 bootstrap) is one of the currently favored methods that has the advantage of performing well even when there is severe overfitting. Use of bootstrapping for NNs is described in Baxt and White (1995), Tibshirani (1996), and Masters (1995). However, the results obtained so far are not very thorough, and it is known that bootstrapping does not work well for some other methodologies such as empirical decision trees (Breiman, Friedman, Olshen, and Stone, 1984; Kohavi, 1995), for which it can be excessively optimistic.

For further information

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

More information on jackknife and bootstrap confidence intervals is available at ftp://ftp.sas.com/pub/neural/jackboot.sas (this is a plain-text file).

References:

Baxt, W.G. and White, H. (1995) "Bootstrapping confidence intervals for clinical input variable effects in a network trained to identify the presence of acute myocardial infarction", Neural Computation, 7, 624-638.

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

Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J. (1984), *Classification and Regression Trees,* Belmont, CA: Wadsworth.

Breiman, L., and Spector, P. (1992), "Submodel selection and evaluation in regression: The X-random case," International Statistical Review, 60, 291-319.

Dijkstra, T.K., ed. (1988), *On Model Uncertainty and Its Statistical Implications,* Proceedings of a workshop held in Groningen, The Netherlands, September 25-26, 1986, Berlin: Springer-Verlag.

Efron, B. (1982) *The Jackknife, the Bootstrap and Other Resampling Plans,* Philadelphia: SIAM.

Efron, B. (1983), "Estimating the error rate of a prediction rule: Improvement on cross-validation," J. of the American Statistical Association, 78, 316-331.

Efron, B. and Tibshirani, R.J. (1993), *An Introduction to the Bootstrap,* London: Chapman & Hall.

Efron, B. and Tibshirani, R.J. (1997), "Improvements on cross-validation: The .632+ bootstrap method," J. of the American Statistical Association, 92, 548-560.

Goutte, C. (1997), "Note on free lunches and cross-validation," Neural Computation, 9, 1211-1215, ftp://eivind.imm.dtu.dk/dist/1997/goutte.nflcv.ps.gz.

Hjorth, J.S.U. (1994), *Computer Intensive Statistical Methods Validation, Model Selection, and Bootstrap,* London: Chapman & Hall.

Hurvich, C.M., and Tsai, C.-L. (1989), "Regression and time series model selection in small samples," Biometrika, 76, 297-307.

Kearns, M. (1997), "A bound on the error of cross validation using the approximation and estimation rates, with consequences for the training-test split," Neural Computation, 9, 1143-1161.

Kohavi, R. (1995), "A study of cross-validation and bootstrap for accuracy estimation and model selection," International Joint Conference on Artificial Intelligence (IJCAI), pp. ?, http://robotics.stanford.edu/users/ronnyk/

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

Plutowski, M., Sakata, S., and White, H. (1994), "Cross-validation estimates IMSE," in Cowan, J.D., Tesauro, G., and Alspector, J. (eds.) *Advances in Neural Information Processing Systems 6,* San Mateo, CA: Morgan Kaufman, pp. 391-398.

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

Shao, J. (1993), "Linear model selection by cross-validation," J. of the American Statistical Association, 88, 486-494.

Shao, J. (1995), "An asymptotic theory for linear model selection," Statistica Sinica ?.

Shao, J. and Tu, D. (1995), *The Jackknife and Bootstrap,* New York: Springer-Verlag.

Snijders, T.A.B. (1988), "On cross-validation for predictor evaluation in time series," in Dijkstra (1988), pp. 56-69.

Stone, M. (1977), "Asymptotics for and against cross-validation," Biometrika, 64, 29-35.

Stone, M. (1979), "Comments on model selection criteria of Akaike and Schwarz," J. of the Royal Statistical Society, Series B, 41, 276-278.

Tibshirani, R. (1996), "A comparison of some error estimates for neural network models," Neural Computation, 8, 152-163.

Weiss, S.M. and Kulikowski, C.A. (1991), *Computer Systems That Learn,* Morgan Kaufmann.

Zhu, H., and Rohwer, R. (1996), "No free lunch for cross-validation," Neural Computation, 8, 1421-1426.

Subject: How to compute prediction and confidence intervals (error bars)?

(This answer is only about half finished. I will get around to the other half eventually.)

In addition to estimating over-all generalization error, it is often useful to be able to estimate the accuracy of the network's predictions for individual cases.

Let:

```
Y      = the target variable
y_i    = the value of Y for the ith case
X      = the vector of input variables
x_i    = the value of X for the ith case
N      = the noise in the target variable
n_i    = the value of N for the ith case
m(X)   = E(Y|X) = the conditional mean of Y given X
w      = a vector of weights for a neural network
w^     = the weight obtained via training the network
p(X,w) = the output of a neural network given input X and weights w
p_i    = p(x_i,w)
L      = the number of training (learning) cases, (y_i,x_i), i=1, ..., L
Q(w)   = the objective function
```

Assume the data are generated by the model:

```
Y = m(X) + N
E(N|X) = 0
N and X are independent
```

The network is trained by attempting to minimize the objective function `Q(w)`, which, for example, could be the sum of squared errors or the negative log likelihood based on an assumed family of noise distributions.

Given a test input `x_0`, a `100c%` prediction interval for `y_0` is an interval `[LPB_0,UPB_0]` such that `Pr(LPB_0 <= y_0 <= UPB_0) = c`, where `c` is typically .95 or .99, and the probability is computed over repeated random selection of the training set and repeated observation of `Y` given the test input `x_0`. A `100c%` confidence interval for `p_0` is an interval `[LCB_0,UCB_0]` such that `Pr(LCB_0 <= p_0 <= UCB_0) = c`, where again the probability is computed over repeated random selection of the training set. Note that `p_0` is a nonrandom quantity, since `x_0` is given. A confidence interval is narrower than the corresponding prediction interval, since the prediction interval must include variation due to noise in `y_0`, while the confidence interval does not. Both intervals include variation due to sampling of the training set and possible variation in the training process due, for example, to random initial weights and local minima of the objective function.

Traditional statistical methods for nonlinear models depend on several assumptions (Gallant, 1987):

1. The inputs for the training cases are either fixed or obtained by simple random sampling or some similarly well-behaved process.
2. `Q(w)` has continuous first and second partial derivatives with respect to `w` over some convex, bounded subset `S_W` of the weight space.
3. `Q(w)` has a unique global minimum at `w^`, which is an interior point of `S_W`.
4. The model is well-specified, which requires (a) that there exist weights `w$` in the interior of `S_W` such that `m(x) = p(x,w$)`, and (b) that the assumptions about the noise distribution are correct. (Sorry about the `w$` notation, but I'm running out of plain text symbols.)

These traditional methods are based on a linear approximation to `p(x,w)` in a neighborhood of `w$`, yielding a quadratic approximation to `Q(w)`. Hence the Hessian of `Q(w)` (the square matrix of second-order partial derivatives with respect to `w`) frequently appears in these methods.

Assumption (3) is not satisfied for neural nets, because networks with hidden units always have multiple global minima, and the global minima are often improper. Hence, confidence intervals for the weights cannot be obtained using standard Hessian-based methods. However, Hwang and Ding (1997) have shown that confidence intervals for predicted values can be obtained because the predicted values are statistically identified even though the weights are not.

Cardell, Joerding, and Li (1994) describe a more serious violation of assumption (3), namely that that for some `m(x)`, no finite global minimum exists. In such situations, it may be possible to use regularization methods such as weight decay to obtain valid confidence intervals (De Veaux, Schumi, Schweinsberg, and Ungar, 1998), but more research is required on this subject, since the derivation in the cited paper assumes a finite global minimum.

For large samples, the sampling variability in `w^` can be approximated in various ways:

- Fisher's information matrix, which is the expected value of the Hessian of `Q(w)` divided by `L`, can be used when `Q(w)` is the negative log likelihood (Spall, 1998).
- The delta method, based on the Hessian of `Q(w)` or the Gauss-Newton approximation using the cross-product Jacobian of `Q(w)`, can also be used when `Q(w)` is the negative log likelihood (Tibshirani, 1996; Hwang and Ding, 1997; De Veaux, Schumi, Schweinsberg, and Ungar, 1998).
- The sandwich estimator, a more elaborate Hessian-based method, relaxes assumption (4) (Gallant, 1987; White, 1989; Tibshirani, 1996).
- Bootstrapping can be used without knowing the form of the noise distribution and takes into account variability introduced by local minima in training, but requires training the network many times on different resamples of the training set (Tibshirani, 1996; Heskes 1997).

References:

Cardell, N.S., Joerding, W., and Li, Y. (1994), "Why some feedforward networks cannot learn some polynomials," Neural Computation, 6, 761-766.

De Veaux,R.D., Schumi, J., Schweinsberg, J., and Ungar, L.H. (1998), "Prediction intervals for neural networks via nonlinear regression," Technometrics, 40, 273-282.

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

Heskes, T. (1997), "Practical confidence and prediction intervals," in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) *Advances in Neural Information Processing Systems 9,* Cambrideg, MA: The MIT Press, pp. 176-182.

Hwang, J.T.G., and Ding, A.A. (1997), "Prediction intervals for artificial neural networks," J. of the American Statistical Association, 92, 748-757.

Nix, D.A., and Weigend, A.S. (1995), "Learning local error bars for nonlinear regression," in Tesauro, G., Touretzky, D., and Leen, T., (eds.) *Advances in Neural Information Processing Systems 7,* Cambridge, MA: The MIT Press, pp. 489-496.

Spall, J.C. (1998), "Resampling-based calculation of the information matrix in nonlinear statistical models," Proceedings of the 4th Joint Conference on Information Sciences, October 23-28, Research Triangle PArk, NC, USA, Vol 4, pp. 35-39.

Tibshirani, R. (1996), "A comparison of some error estimates for neural network models," Neural Computation, 8, 152-163.

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

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