Neural Networks and Statistical Models

Proceedings of the Nineteenth Annual SAS Users Group International Conference, April, 1994

Warren S. Sarle, SAS Institute Inc., Cary, NC, USA

Abstract

There has been much publicity about the ability of artificial neural networks to learn and generalize. In fact, the most commonly used artificial neural networks, called multilayer perceptrons, are nothing more than nonlinear regression and discriminant models that can be implemented with standard statistical software. This paper explains what neural networks are, translates neural network jargon into statistical jargon, and shows the relationships between neural networks and statistical models such as generalized linear models, maximum redundancy analysis, projection pursuit, and cluster analysis.

Introduction

Neural networks are a wide class of flexible nonlinear regression and discriminant models, data reduction models, and nonlinear dynamical systems. They consist of an often large number of "neurons," i.e. simple linear or nonlinear computing elements, interconnected in often complex ways and often organized into layers.

Artificial neural networks are used in three main ways:

- as models of biological nervous systems and "intelligence"
- as real-time adaptive signal processors or controllers implemented in hardware for applications such as robots
- as data analytic methods

This paper is concerned with artificial neural networks for data analysis.

The development of artificial neural networks arose from the attempt to simulate biological nervous systems by combining many simple computing elements (neurons) into a highly interconnected system and hoping that complex phenomena such as "intelligence" would emerge as the result of self-organization or learning. The alleged potential intelligence of neural networks led to much research in implementing artificial neural networks in hardware such as VLSI chips. The literature remains confused as to whether artificial neural networks are supposed to be realistic biological models or practical machines. For data analysis, biological plausibility and hardware implementability are irrelevant.

The alleged intelligence of artificial neural networks is a matter of dispute. Artificial neural networks rarely have more than a few hundred or a few thousand neurons, while the human brain has about one hundred billion neurons. Networks comparable to a human brain in complexity are still far beyond the capacity of the fastest, most highly parallel computers in existence. Artificial neural networks, like many statistical methods, are capable of processing vast amounts of data and making predictions that are sometimes surprisingly accurate; this does not make them "intelligent" in the usual sense of the word. Artificial neural networks "learn" in much the same way that many statistical algorithms do estimation, but usually much more slowly than statistical algorithms. If artificial neural networks are intelligent, then many statistical methods must also be considered intelligent.

Few published works provide much insight into the relationship between statistics and neural networks—Ripley (1993) is probably the best account to date. Weiss and Kulikowski (1991) provide a good elementary discussion of a variety of classification methods including statistical and neural methods. For those interested in more than the statistical aspects of neural networks, Hinton (1992) offers a readable introduction without the inflated claims common in popular accounts. The best book on neural networks is Hertz, Krogh, and Palmer (1991), which can be consulted regarding most neural net issues for which explicit citations are not given in this paper. Hertz et al. also cover nonstatistical networks such as Hopfield networks and Boltzmann machines. Masters (1993) is a good source of practical advice on neural networks. White (1992) contains reprints of many useful articles on neural networks and statistics at an advanced level.

Models and Algorithms

When neural networks (henceforth NNs, with the adjective "artificial" implied) are used for data analysis, it is important to distinguish between NN *models* and NN *algorithms*.

Many NN models are similar or identical to popular statistical techniques such as generalized linear models, polynomial regression, nonparametric regression and discriminant analysis, projection pursuit regression, principal components, and cluster analysis, especially where the emphasis is on prediction of complicated phenomena rather than on explanation. These NN models can be very useful. There are also a few NN models, such as counterpropagation, learning vector quantization, and self-organizing maps, that have no precise statistical equivalent but may be useful for data analysis.

Many NN researchers are engineers, physicists, neurophysiologists, psychologists, or computer scientists who know little about statistics and nonlinear optimization. NN researchers routinely reinvent methods that have been known in the statistical or mathematical literature for decades or centuries, but they often fail to understand how these methods work (e.g., Specht 1991). The common implementations of NNs are based on biological or engineering criteria, such as how easy it is to fit the net on a chip, rather than on well-established statistical and optimization criteria.

Standard NN learning algorithms are inefficient because they are designed to be implemented on massively parallel computers but are, in fact, usually implemented on common serial computers such as ordinary PCs. On a serial computer, NNs can be trained more efficiently by standard numerical optimization algorithms such as those used for nonlinear regression. Nonlinear regression algorithms can fit most NN models orders of magnitude faster than the standard NN algorithms.

Another reason for the inefficiency of NN algorithms is that they are often designed for situations where the data are not stored, but each observation is available transiently in a real-time environment. Transient data are inappropriate for most types of statistical analysis. In statistical applications, the data are usually stored and are repeatedly accessible, so statistical algorithms can be faster and more stable than NN algorithms.

Hence, for most practical data analysis applications, the usual NN algorithms are not useful. You do not need to know anything about NN training methods such as backpropagation to use NNs.

Jargon

Although many NN models are similar or identical to well-known statistical models, the terminology in the NN literature is quite different from that in statistics. For example, in the NN literature:

- variables are called *features*
- independent variables are called *inputs*
- predicted values are called *outputs*
- dependent variables are called *targets* or *training values*
- residuals are called *errors*
- estimation is called *training*, *learning*, *adaptation*, or *self-organization*.
- an estimation criterion is called an *error function*, *cost function*, or *Lyapunov function*
- observations are called *patterns* or *training pairs*
- parameter estimates are called *(synaptic) weights*
- interactions are called *higher-order neurons*
- transformations are called *functional links*
- regression and discriminant analysis are called *supervised learning* or *heteroassociation*
- data reduction is called *unsupervised learning*, *encoding*, or *autoassociation*
- cluster analysis is called *competitive learning* or *adaptive vector quantization*
- interpolation and extrapolation are called *generalization*

The statistical terms *sample* and *population* do not seem to have NN equivalents. However, the data are often divided into a *training set* and *test set* for cross-validation.

Network Diagrams

Various models will be displayed as network diagrams such as the one shown in Figure 1, which illustrates NN and statistical terminology for a simple linear regression model. Neurons are represented by circles and boxes, while the connections between neurons are shown as arrows:

- Circles represent observed variables, with the name shown inside the circle.
- Boxes represent values computed as a function of one or more arguments. The symbol inside the box indicates the type of function. Most boxes also have a corresponding parameter called a *bias*.
- Arrows indicate that the source of the arrow is an argument of the function computed at the destination of the arrow. Each arrow usually has a corresponding *weight* or parameter to be estimated.
- Two long parallel lines indicate that the values at each end are to be fitted by least squares, maximum likelihood, or some other estimation criterion.

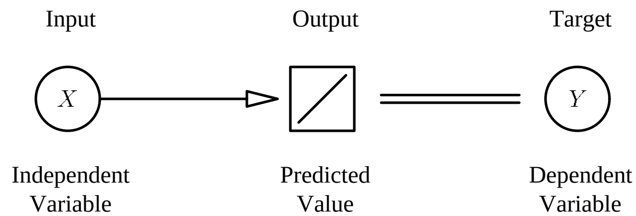

Figure 1: Simple Linear Regression

Perceptrons

A (*simple*) *perceptron* computes a linear combination of the inputs (possibly with an intercept or *bias* term) called the *net input*. Then a possibly nonlinear *activation* function is applied to the net input to produce the output. An activation function maps any real input into a usually bounded range, often 0 to 1 or -1 to 1. Bounded activation functions are often called *squashing* functions. Some common activation functions are:

- linear or identity: $\text{act}(x) = x$
- hyperbolic tangent: $\text{act}(x) = \tanh(x)$
- logistic: $\text{act}(x) = (1 + e^{-x})^{-1} = (\tanh(x/2) + 1)/2$
- threshold: $\text{act}(x) = 0$ if $x < 0$, 1 otherwise
- Gaussian: $\text{act}(x) = e^{-x^2/2}$

Symbols used in the network diagrams for various types of neurons and activation functions are shown in Figure 2.

A perceptron can have one or more outputs. Each output has a separate bias and set of weights. Usually the same activation function is used for each output, although it is possible to use different activation functions.

Notation and formulas for a perceptron are as follows:

$$
\begin{aligned}
n_x &= \text{number of independent variables (inputs)} \\
x_i &= \text{independent variable (input)} \\
a_j &= \text{bias for output layer} \\
b_{ij} &= \text{weight from input to output layer} \\
q_j &= \text{net input to output layer} = a_i + \sum_{j=1}^{n_x} b_{ij} x_i \\
p_j &= \text{predicted value (output values)} = \text{act}(q_j) \\
y_j &= \text{dependent variable (training values)} \\
r_j &= \text{residual (error)} = y_j - p_j
\end{aligned}
$$

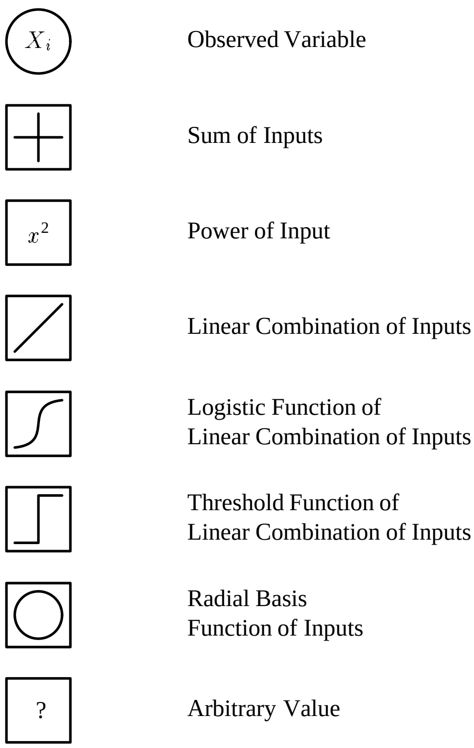

Figure 2: Symbols for Neurons

Perceptrons are most often trained by least squares, i.e., by attempting to minimize $\sum\sum r_j^2$, where the summation is over all outputs and over the training set.

A perceptron with a linear activation function is thus a linear regression model (Weisberg 1985; Myers 1986), possibly multiple or multivariate, as shown in Figure 3.

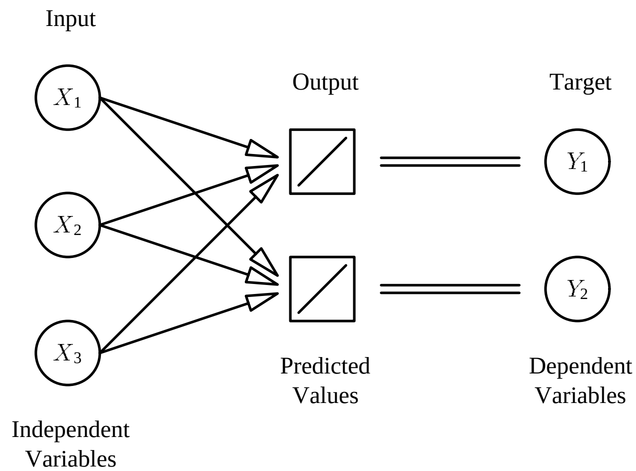

Figure 3: Simple Linear Perceptron = Multivariate Multiple Linear Regression

A perceptron with a logistic activation function is a logistic regression model (Hosmer and Lemeshow 1989) as shown in Figure 4.

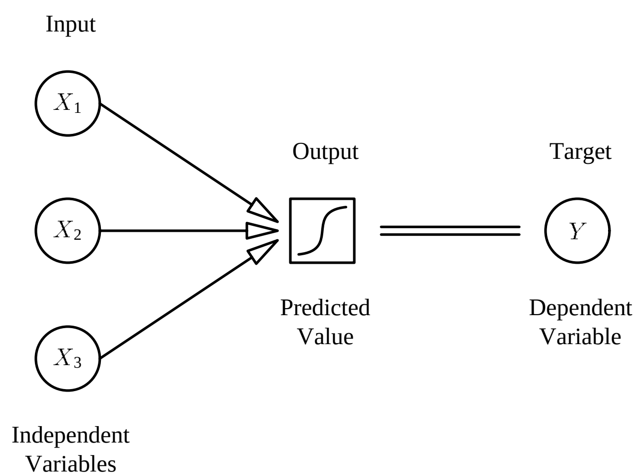

Figure 4: Simple Nonlinear Perceptron = Logistic Regression

A perceptron with a threshold activation function is a linear discriminant function (Hand 1981; McLachlan 1992; Weiss and Kulikowski 1991). If there is only one output, it is also called an *adaline*, as shown in Figure 5. With multiple outputs, the threshold perceptron is a multiple discriminant function. Instead of a threshold activation function, it is often more useful to use a multiple logistic function to estimate the conditional probabilities of each class. A multiple logistic function is called a *softmax* activation function in the NN literature.

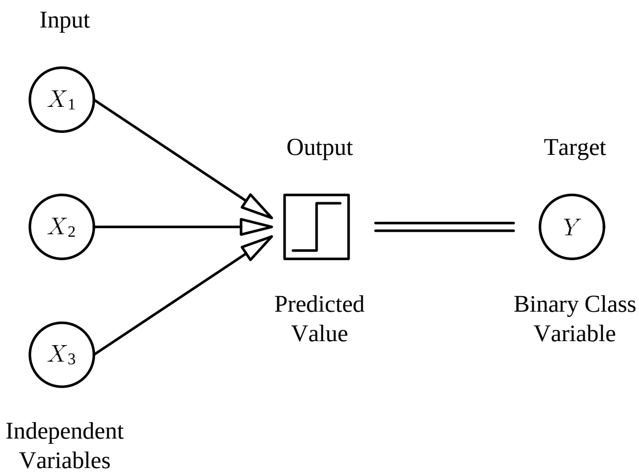

Figure 5: Adaline = Linear Discriminant Function

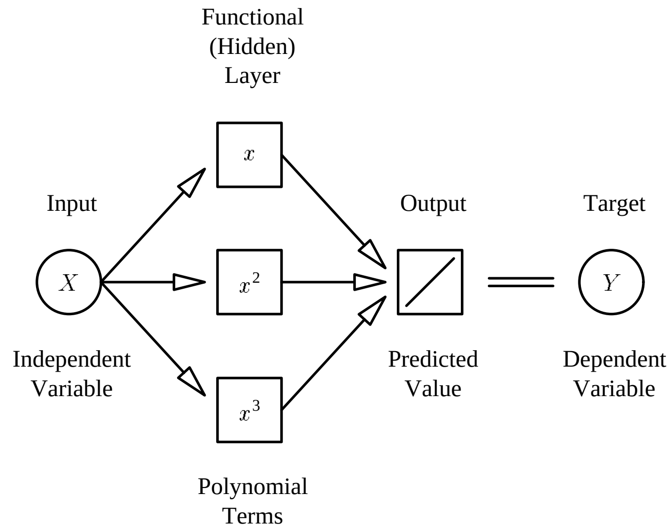

Figure 6: Functional Link Network = Polynomial Regression

The activation function in a perceptron is analogous to the inverse of the link function in a generalized linear model (GLIM) (McCullagh and Nelder 1989). Activation functions are usually bounded, whereas inverse link functions, such as the identity, reciprocal, and exponential functions, often are not. Inverse link functions are required to be monotone in most implementations of GLIMs, although this restriction is only for computational convenience. Activation functions are sometimes nonmonotone, such as Gaussian or trigonometric functions.

GLIMs are fitted by maximum likelihood for a variety of distributions in the exponential class. Perceptrons are usually trained by least squares. Maximum likelihood for binomial proportions is also used for perceptrons when the target values are between 0 and 1, usually with the number of binomial trials assumed to be constant, in which case the criterion is called *relative entropy* or *cross entropy*. Occasionally other criteria are used to train perceptrons. Thus, in theory, GLIMs and perceptrons are almost the same thing, but in practice the overlap is not as great as it could be in theory.

Polynomial regression can be represented by a diagram of the form shown in Figure 6, in which the arrows from the inputs to the polynomial terms would usually be given a constant weight of 1. In NN terminology, this is a type of *functional link* network (Pao 1989). In general, functional links can be transformations of any type that do not require extra parameters, and the activation function for the output is the identity, so the model is linear in the parameters. Elaborate functional link networks are used in applications such as image processing to perform a variety of impressive tasks (Souček and The IRIS Group 1992).

Multilayer Perceptrons

A functional link network introduces an extra *hidden layer* of neurons, but there is still only one layer of weights to be estimated. If the model includes estimated weights between the inputs and the hidden layer, and the hidden layer uses nonlinear activation functions such as the logistic function, the model becomes genuinely nonlinear, i.e., nonlinear in the parameters. The resulting model is called a *multilayer perceptron* or MLP. An MLP for simple nonlinear regression is shown in Figure 7. An MLP can also have multiple inputs and outputs, as shown in Figure 8. The number of hidden neurons can be less than the number of inputs or outputs, as shown in Figure 9. Another useful variation is to allow direct connections from the input layer to the output layer, which could be called *main effects* in statistical terminology.

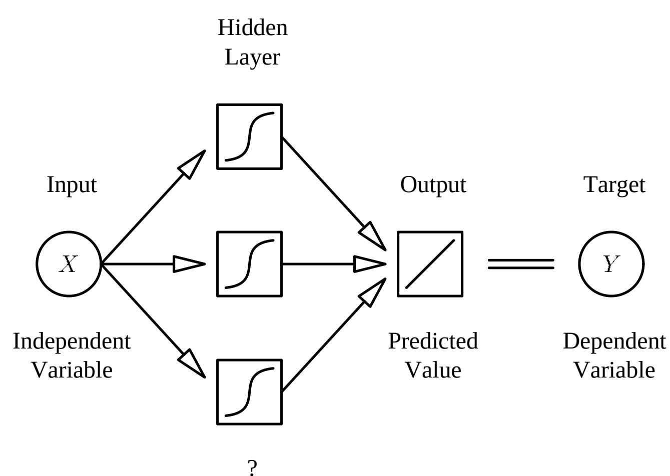

Figure 7: Multilayer Perceptron = Simple Nonlinear Regression

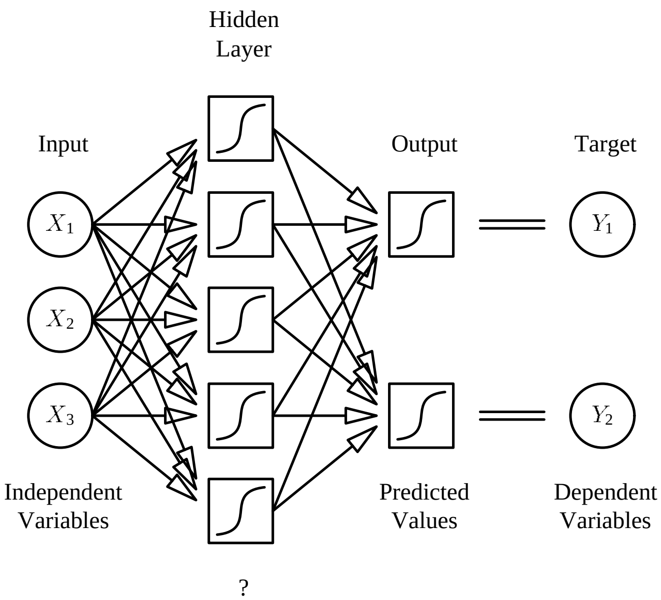

Figure 8: Multilayer Perceptron = Multivariate Multiple Nonlinear Regression

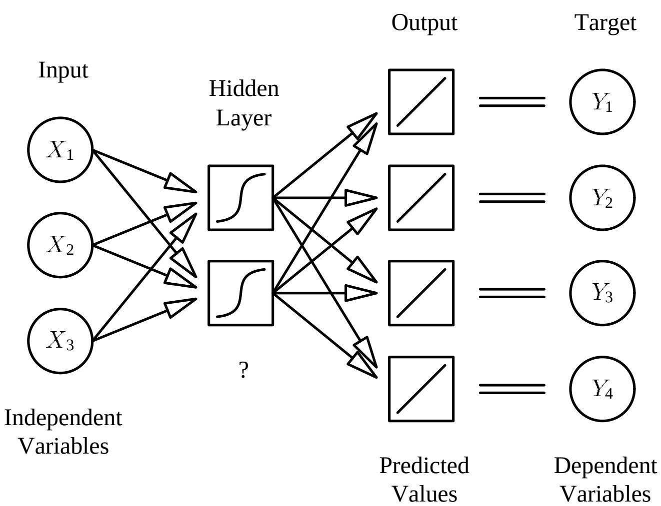

Figure 9: Multilayer Perceptron = Nonlinear Regression Again

Notation and formulas for the MLP in Figure 8 are as follows:

$$
\begin{aligned}
n_x &= \text{number of independent variables (inputs)} \\
n_h &= \text{number of hidden neurons} \\
x_i &= \text{independent variable (input)} \\
a_j &= \text{bias for hidden layer} \\
b_{ij} &= \text{weight from input to hidden layer} \\
g_j &= \text{net input to hidden layer} = a_j + \sum_{i=1}^{n_x} b_{ij} x_i \\
h_j &= \text{hidden layer values} = \text{act}_h(g_j) \\
c_k &= \text{bias for output (intercept)} \\
d_{jk} &= \text{weight from hidden layer to output} \\
q_k &= \text{net input to output layer} = c_k + \sum_{j=1}^{n_h} d_{jk} h_k \\
p_k &= \text{predicted value (output values)} = \text{act}_o(q_k) \\
y_k &= \text{dependent variable (training values)} \\
r_k &= \text{residual (error)} = y_k - p_k
\end{aligned}
$$

where act_h and act_o are the activation functions for the hidden and output layers, respectively.

MLPs are general-purpose, flexible, nonlinear models that, given enough hidden neurons and enough data, can approximate virtually any function to any desired degree of accuracy. In other words, MLPs are *universal approximators* (White 1992). MLPs can be used when you have little knowledge about the form of the relationship between the independent and dependent variables.

You can vary the complexity of the MLP model by varying the number of hidden layers and the number of hidden neurons in each hidden layer. With a small number of hidden neurons, an MLP is a parametric model that provides a useful alternative to polynomial regression. With a moderate number of hidden neurons, an MLP can be considered a quasi-parametric model similar to projection pursuit regression (Friedman and Stuetzle 1981). An MLP with one hidden layer is essentially the same as the projection pursuit regression model except that an MLP uses a predetermined functional form for the activation function in the hidden layer, whereas projection pursuit uses a flexible nonlinear smoother. If the number of hidden neurons is allowed to increase with the sample size, an MLP becomes a nonparametric sieve (White 1992) that provides a useful alternative to methods such as kernel regression (Hä rdle 1990) and smoothing splines (Eubank 1988; Wahba 1990). MLPs are especially valuable because you can vary the complexity of the model from a simple parametric model to a highly flexible, nonparametric model.

Consider an MLP for fitting a simple nonlinear regression curve, using one input, one linear output, and one hidden layer with a logistic activation function. The curve can have as many wiggles in it as there are hidden neurons (actually, there can be even more wiggles than the number of hidden neurons, but estimation tends to become more difficult in that case). This simple MLP acts very much like a polynomial regression or least-squares smoothing spline (Eubank 1988). Since polynomials are linear in the parameters, they are fast to fit, but there are numerical accuracy problems if you try to fit too many wiggles. Smoothing splines are also linear in the parameters and don't have the numerical problems of high-order polynomials, but splines present the problem of deciding where to locate the knots.

MLPs with a nonlinear activation function are genuinely nonlinear in the parameters and therefore take much more computer time to fit than polynomials or splines. MLPs may be more numerically stable than high-order polynomials. MLPs do not require you to specify knot locations, but they may suffer from local minima in the optimization process. MLPs have different extrapolation properties than polynomials—polynomials go off to infinity, but MLPs flatten out—but both can do very weird things when extrapolated. All three methods raise similar questions about how many wiggles to fit.

Unlike splines and polynomials, MLPs are easy to extend to multiple inputs and multiple outputs without an exponential increase in the number of parameters.

MLPs are usually trained by an algorithm called the *generalized delta rule*, which computes derivatives by a simple application of the chain rule called *backpropagation*. Often the term *backpropagation* is applied to the training method itself or to a network trained in this manner. This confusion is symptomatic of the general failure in the NN literature to distinguish between models and estimation methods.

Use of the generalized delta rule is slow and tedious, requiring the user to set various algorithmic parameters by trial and error. Fortunately, MLPs can be easily trained with general purpose nonlinear modeling or optimization programs such as the procedures NLIN in SAS/STAT® software, MODEL in SAS/ETS® software, NLP in SAS/OR® software, and the various NLP routines in SAS/IML® software. There is extensive statistical theory regarding nonlinear models (Bates and Watts 1988; Borowiak 1989; Cramer 1986; Edwards 1972; Gallant 1987; Gifi 1990; Hä rdle 1990; Ross 1990; Seber and Wild 1989). Statistical software can be used to produce confidence intervals, prediction intervals, diagnostics, and various graphical displays, all of which rarely appear in the NN literature.

Unsupervised Learning

The NN literature distinguishes between supervised and unsupervised learning. In supervised learning, the goal is to predict one or more target variables from one or more input variables. Supervision consists of the use of target values in training. Supervised learning is usually some form of regression or discriminant analysis. MLPs are the most common variety of supervised network.

In unsupervised learning, the NN literature claims that there is no target variable, and the network is supposed to train itself to extract "features" from the independent variables, as shown in Figure 10. This conceptualization is wrong. In fact, the goal in most forms of unsupervised learning is to construct feature variables from which the observed variables, which are really both input *and* target variables, can be predicted.

Unsupervised Hebbian learning constructs quantitative features. In most cases, the dependent variables are predicted by linear regression from the feature variables. Hence, as is well-known from statistical theory, the optimal feature variables are the principal components of the dependent variables (Hotelling 1933; Jackson 1991; Jolliffe 1986; Rao 1964). There are many variations, such as Oja's rule and Sanger's rule, that are just inefficient algorithms for approximating principal components.

The statistical model of principal component analysis is shown in Figure 11. In this model there are no inputs. The boxes containing ?s indicate that the values for these neurons can be computed in any way whatsoever, provided the least-squares fit of the model is optimized. Of course, it can be proven that the optimal values for the ? boxes are the principal component scores, which can be computed as linear combinations of the observed variable. Hence the model can also be expressed as in Figure 12, in which the observed variables are shown as both inputs and target values. The input layer and hidden layer in this model are the same as the unsupervised learning model in Figure 10. The rest of Figure 12 is implied by unsupervised Hebbian learning, but this fact is rarely acknowledged in the NN literature.

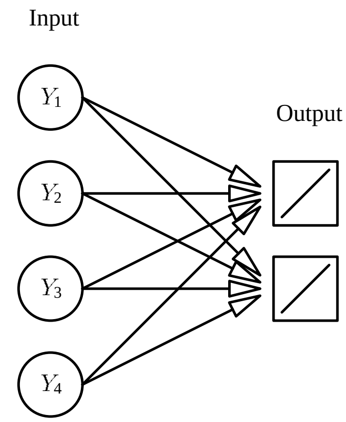

Figure 10: Unsupervised Hebbian Learning

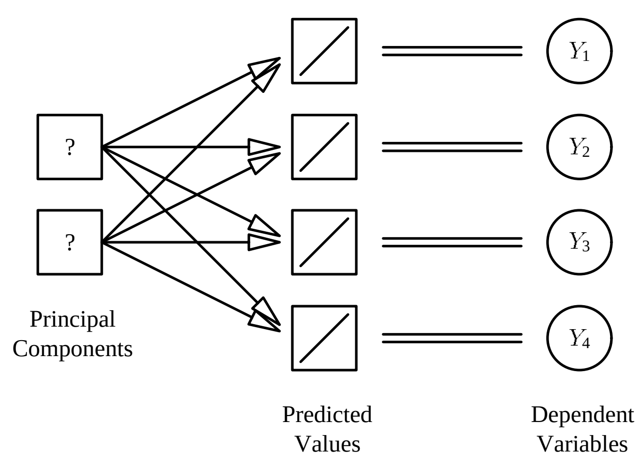

Figure 11: Principal Component Analysis

Unsupervised competitive learning constructs binary features. Each binary feature represents a subset or cluster of the observations. The network is the same as in Figure 10 except that only one output neuron is activated with an output of 1 while all the other output neurons are forced to 0. Neurons of this type are often called *winner-take-all* neurons or *Kohonen* neurons.

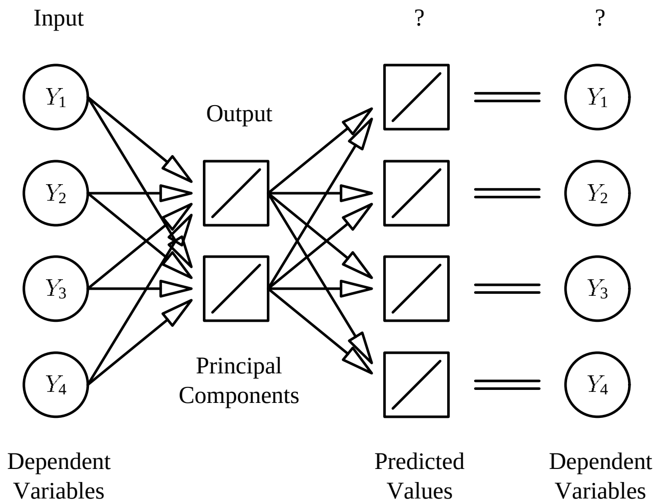

Figure 12: Principal Component Analysis---Alternative Model

The winner is usually determined to be the neuron with the largest net input, in other words, the neuron whose weights are most similar to the input values as measured by an inner-product similarity measure. For an inner-product similarity measure to be useful, it is usually necessary to normalize both the weights of each neuron and the input values for each observation. In this case, inner-product similarity is equivalent to Euclidean distance. However, the normalization requirement greatly limits the applicability of the network. It is generally more useful to define the net input as the Euclidean distance between the synaptic weights and the input values, in which case the competitive learning network is very similar to k-means clustering (Hartigan 1975) except that the usual training algorithms are slow and nonconvergent. Many superior clustering algorithms have been developed in statistics, numerical taxonomy, and many other fields, as described in countless articles and numerous books such as Everitt (1980), Massart and Kaufman (1983), Anderberg (1973), Sneath and Sokal (1973), Hartigan (1975), Titterington, Smith, and Makov (1985), McLachlan and Basford (1988), Kaufmann and Rousseeuw (1990), and Spath (1980).

In *adaptive vector quantization* (AVQ), the inputs are acknowledged to be target values that are predicted by the means of the cluster to which a given observation belongs. This network is therefore essentially the same as that in Figure 12 except for the winner-take-all activation functions. In other words, AVQ is least-squares cluster analysis. However, the usual AVQ algorithms do not simply compute the mean of each cluster but approximate the mean using an iterative, nonconvergent algorithm. It is far more efficient to use any of a variety of algorithms for cluster analysis such as those in the FASTCLUS procedure.

Feature mapping is a form of nonlinear dimensionality reduction that has no statistical analog. There are several varieties of feature mapping, of which Kohonen's (1989) *self-organizing map* (SOM) is the best known. Methods such as principal components and multidimensional scaling can be used to map from a continuous high-dimensional space to a continuous low-dimensional space. SOM maps from a continuous space to a discrete space. The continuous space can be of higher dimensionality, but this is not necessary. The discrete space is represented by an array of competitive output neurons. For example, a continuous space of five inputs might be mapped to 100 output neurons in a 10×10 array; i.e., any given set of input values would turn on one of the 100 outputs. Any two neurons that are neighbors in the output array would correspond to two sets of points in the input space that are close to each other.

Hybrid Networks

Hybrid networks combine supervised and unsupervised learning. Principal component regression (Myers 1986) is an example of a well-known statistical method that can be viewed as a hybrid network with three layers. The independent variables are the input layer, and the principal components of the independent variables are the hidden, unsupervised layer. The predicted values from regressing the dependent variables on the principal components are the supervised output layer.

Counterpropagation networks are widely touted as hybrid networks that learn much more rapidly than backpropagation networks. In counterpropagation networks, the variables are divided into two sets, say $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$. The goal is to be able to predict both the x variables from the y variables and the y variables from the x variables. The counterpropagation network effectively performs a cluster analysis using both the x and y variables. To predict x given y in a particular observation, compute the distance from the observation to each cluster mean using only the y variables, find the nearest cluster, and predict x as the mean of the x variables in the nearest cluster. The method for predicting y given x obviously reverses the roles of x and y.

The usual counterpropagation algorithm is, as usual, inefficient and nonconvergent. It is far more efficient to use the FASTCLUS procedure to do the clustering and to use the IMPUTE option to make the predictions. FASTCLUS offers the advantage that you can predict any subset of variables from any other disjoint subset of variables.

In practice, bidirectional prediction such as that done by counterpropagation is rarely needed. Hence, counterpropagation is usually used for prediction in only one direction. As such, counterpropagation is a form of nonparametric regression in which the smoothing parameter is the number of clusters. If training is unidirectional, then counterpropagation is a regressogram estimator (Tukey 1961) with the bins determined clustering the input cases. With bidirectional training, both the input and target variables are used in forming the clusters; this makes the clusters more adaptive to the local slope of the regression surface but can create problems with heteroscedastic data, since the smoothness of the estimate depends on the local variance of the target variables. Bidirectional training also adds the complication of choosing the relative weight of the input and target variables in the cluster analysis. Counterpropagation would clearly have advantages for discontinuous regression functions but is ineffective at discounting independent variables with little or no predictive value. For continuous regression functions, counterpropagation could be improved by some additional smoothing. The NN literature usually uses interpolation, but kernel smoothing would be superior in most cases. Kernel-smoothed counterpropagation would be a variety of binned kernel regression estimation using clusters for the bins, similar to the clustered form of GRNN (Specht 1991).

Learning vector quantization (LVQ) (Kohonen 1989) has both supervised and unsupervised aspects, although it is not a hybrid network in the strict sense of having separate supervised and unsupervised layers. LVQ is a variation of nearest-neighbor discriminant analysis. Rather than finding the nearest neighbor in the entire training set to classify an input vector, LVQ finds the nearest point in a set of prototype vectors, with several protypes for each class. LVQ differs from edited and condensed k-nearest-neighbor methods (Hand 1981) in that the prototypes are not members of the training set but are computed using algorithms similar to AVQ. A somewhat similar method proceeds by clustering each class separately and then using the cluster centers as prototypes. The clustering approach is better if you want to estimate posterior membership probabilities, but LVQ may be more effective if the goal is simply classification.

Radial Basis Functions

In an MLP, the net input to the hidden layer is a linear combination of the inputs as specified by the weights. In a radial basis function (RBF) network (Wasserman 1993), as shown in Figure 13, the hidden neurons compute radial basis functions of the inputs, which are similar to kernel functions in kernel regression (Härdle 1990). The net input to the hidden layer is the distance from the input vector to the weight vector. The weight vectors are also called *centers*. The distance is usually computed in the Euclidean metric, although it is sometimes a weighted Euclidean distance or an inner product metric. There is usually a bandwidth s_j associated with each hidden node, often called *sigma*. The activation function can be any of a variety of functions on the nonnegative real numbers with a maximum at zero, approaching zero at infinity, such as $e^{-x^2/2}$. The outputs are computed as linear combinations of the hidden values with an identity activation function.

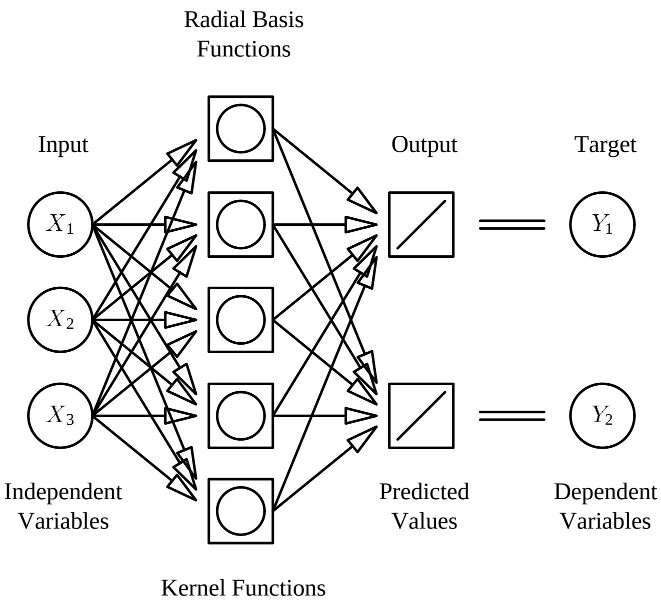

Figure 13: Radial Basis Function Network

For comparison, typical formulas for an MLP hidden neuron and an RBF neuron are as follows:

$$\begin{aligned}
\text{MLP}: g_j &= a_j + \sum_{i=1}^{n_x} b_{ij} x_i \\
h_j &= (1 + e^{-g_j})^{-1} \\
\text{RBF}: g_j &= \left[\sum_{i=1}^{n_x} \frac{(b_{ij} - x_i)^2}{2 s_j} \right]^{1/2} \\
h_j &= e^{-g_j^2/2}
\end{aligned}$$

The region near each RBF center is called the *receptive field* of the hidden neuron. RBF neurons are also called *localized receptive fields*, *locally tuned processing units*, or *potential functions*. RBF networks are closely related to *regularization networks*. The *modified Kanerva model* (Prager and Fallside 1989) is an RBF network with a threshhold activation function. The *Restricted Coulomb Energy*™ *System* (Cooper, Elbaum and Reilly 1982) is another threshold RBF network used for classification. There is a discrete variant of RBF networks called the *cerebellum model articulation controller* (CMAC) (Miller, Glanz and Kraft 1990).

Sometimes the hidden layer values are normalized to sum to 1 (Moody and Darken 1988) as is commonly done in kernel regression (Nadaraya 1964; Watson 1964). Then if each observation is taken as an RBF center, and if the weights are taken to be the target values, the outputs are simply weighted averages of the target values, and the network is identical to the well-known Nadaraya-Watson kernel regression estimator. This method has been reinvented twice in the NN literature (Specht 1991; Schiøler and Hartmann 1992).

Specht has popularized both kernel regression, which he calls a *general regression neural network* (GRNN) and kernel discriminant analysis, which he calls a *probabilistic neural network* (PNN). Specht's (1991) claim that a GRNN is effective with "only a few samples" and even with "sparse data in a multidimensional ... space" is directly contradicted by statistical theory. For parametric models, the error in prediction typically decreases in proportion to $n^{-1/2}$, where n is the sample size. For kernel regression estimators, the error in prediction typically decreases in proportion to $n^{-p/(2p+d)}$, where p is the number of derivatives of the regression function and d is the number of inputs (Härdle 1990, 93). Hence, kernel methods tend to require larger sample sizes than paramteric methods, especially in multidimensional spaces.

Since an RBF network can be viewed as a nonlinear regression model, the weights can be estimated by any of the usual methods for nonlinear least squares or maximum likelihood, although this would yield a vastly overparameterized model if every observation were used as an RBF center. Usually, however, RBF networks are treated as hybrid networks. The inputs are clustered, and the RBF centers are set equal to the cluster means. The bandwidths are often set to the nearest-neighbor distance from the center (Moody and Darken 1988), although this is not a good idea because nearest-neighbor distances are excessively variable; it works better to determine the bandwidths from the cluster variances. Once the centers and bandwidths are determined, estimating the weights from the hidden layer to the outputs reduces to linear least squares.

Another method for training RBF networks is to consider each case as a potential center and then select a subset of cases using any of the usual methods for subset selection in linear regression. If forward stepwise selection is used, the method is called *orthogonal least squares* (OLS) (Chen et al. 1991).

Adaptive Resonance Theory

Some NNs are based explicitly on neurophysiology. Adaptive resonance theory (ART) is one of the best known classes of such networks. ART networks are defined algorithmically in terms of detailed differential equations, not in terms of anything recognizable as a statistical model. In practice, ART networks are implemented using analytical solutions or approximations to these differential equations. ART does not estimate parameters in any useful statistical sense and may produce degenerate results when trained on "noisy" data typical of statistical applications. ART is therefore of doubtful benefit for data analysis.

ART comes in several varieties, most of which are unsupervised, and the simplest of which is called ART 1. As Moore (1988) pointed out, ART 1 is basically similar to many iterative clustering algorithms in which each case is processed by:

1. finding the "nearest" cluster seed/prototype/template to that case
2. updating that cluster seed to be "closer" to the case

where "nearest" and "closer" can be defined in hundreds of different ways. However, ART 1 differs from most other clustering methods in that it uses a two-stage (lexicographic) measure of nearness. Both inputs and seeds are binary. Most binary similarity measures can be defined in terms of a 2 $\times$ 2 table giving the numbers of matches and mismatches:

		seed/prototype/template	
		1	0
Input	1	A	B
	0	C	D

For example, Hamming distance is the number of mismatches, $B + C$, and the Jaccard coefficient is the number of positive matches normalized by the number of features present, $A/(A + B + C)$.

To oversimplify matters slightly, ART 1 defines the "nearest" seed as the seed with the minimum value of $A/(A + C)$ that *also* satisfies the requirement that $A/(A + B)$ exceeds a specified *vigilance* threshold. An input and seed that satisfy the vigilance threshold are said to *resonate*. If the input fails to resonate with any existing seed, a new seed identical to the input is created, as in Hartigan's (1975) leader algorithm.

If the input resonates with an existing seed, the seed is updated by the logical *and* operator, i.e., a feature is present in the updated seed if and only if it was present both in the input and in the seed before updating. Thus, a seed represents the features common to all of the cases assigned to it. If the input contains noise in the form of 0s where there should be 1s, then the seeds will tend to degenerate toward the zero vector and the clusters will proliferate.

The ART 2 network is for quantitative data. It differs from ART 1 mainly in having an elaborate iterative scheme for normalizing the inputs. The normalization is supposed to reduce the cluster proliferation that plagues ART 1 and to allow for varying background levels in visual pattern recognition. Fuzzy ART (Carpenter, Grossberg, and Rosen 1991) is for bounded quantitative data. It is similar to ART 1 but uses the fuzzy operators *min* and *max* in place of the logical *and* and *or* operators. ARTMAP (Carpenter, Grossberg, and Reynolds 1991) is an ARTistic variant of counterpropagation for supervised learning.

ART has its own jargon. For example, data are called an *arbitrary sequence of input patterns*. The current observation is stored in *short term memory* and cluster seeds are *long term memory*. A cluster is a *maximally compressed pattern recognition code*. The two stages of finding the nearest seed to the input are performed by an *Attentional Subsystem* and an *Orienting Subsystem*, which performs *hypothesis testing*, which simply refers to the comparison with the vigilance threshhold, not to hypothesis testing in the statistical sense.

Multiple Hidden Layers

Although an MLP with one hidden layer is a universal approximator, there exist various applications in which more than one hidden layer can be useful. Sometimes a highly nonlinear function can be approximated with fewer weights when multiple hidden layers are used than when only one hidden layer is used.

Maximum redundancy analysis (Rao 1964; Fortier 1966; van den Wollenberg 1977) is a linear MLP with one hidden layer used for dimensionality reduction, as shown in Figure 14. A nonlinear generalization can be implemented as an MLP by adding another hidden layer to introduce the nonlinearity as shown in Figure 15. The linear hidden layer is a *bottleneck* that accomplishes the dimensionality reduction.

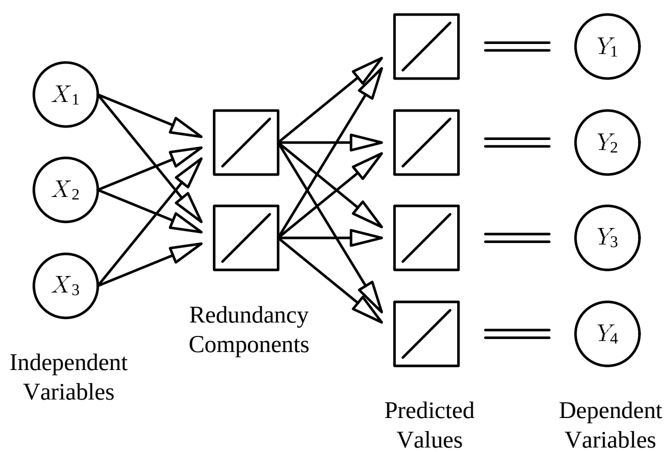

Figure 14: Linear Multilayer Perceptron = Maximum Redundancy Analysis

Principal component analysis, as shown in Figure 12, is another linear model for dimensionality reduction in which the inputs and targets are the same variables. In the NN literature, models with the same inputs and targets are called *encoding* or *autoassociation* networks, often with only one hidden layer. However, one hidden layer is not sufficient to improve upon principal components, as can be seen from Figure 11. A nonlinear generalization of principal components can be implemented as an MLP with three hidden layers, as shown in Figure 16. The first and third hidden

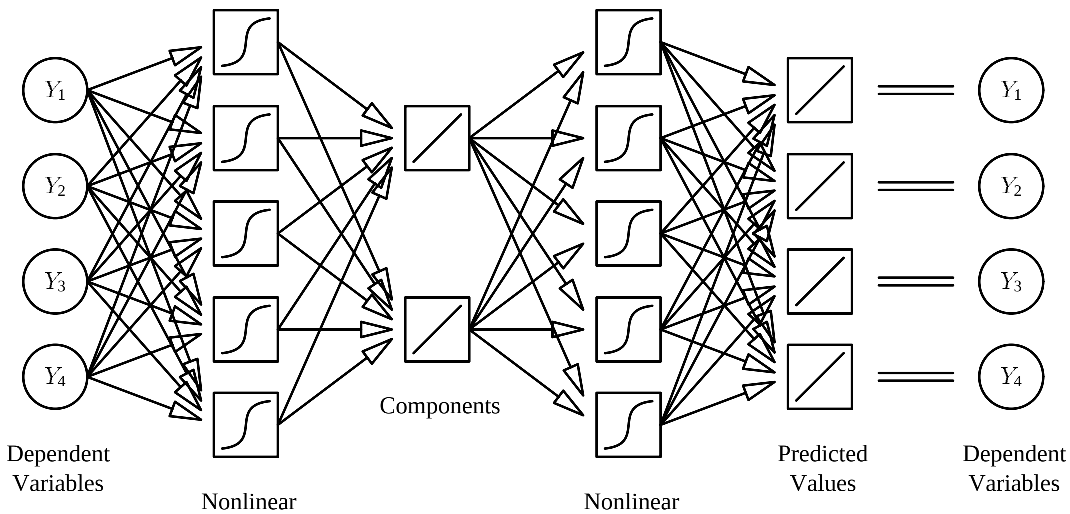

Figure 16: Nonlinear Analog of Principal Components

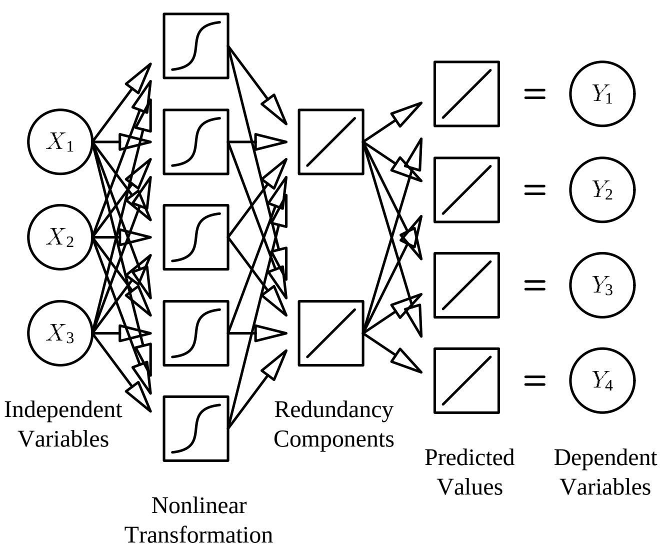

Figure 15: Nonlinear Maximum Redundancy Analysis

layers provide the nonlinearity, while the second hidden layer is the bottleneck.

Nonlinear additive models provide a compromise in complexity between multiple linear regression and a fully flexible nonlinear model such as an MLP, a high-order polynomial, or a tensor spline model. In a generalized additive model (GAM) (Hastie and Tibshirani 1990), a nonlinear transformation estimated by a nonparametric smoother is applied to each input, and these values are added together. The TRANSREG procedure fits nonlinear additive models using B splines. *Topologically distributed encoding* (TDE) (Geiger 1990) uses Gaussian basis functions. A nonlinear additive model can also be implemented as a NN as shown in Figure 17. Each input is connected to a small subnetwork to provide the nonlinear transformations. The outputs of the subnetworks are summed to give the output of the complete network. This network could be reduced to a single hidden layer, but the additional hidden layers aid interpretation of the results.

By adding another linear hidden layer to the GAM network, a projection pursuit network can be constructed as shown in Figure 18. This network is similar to projection pursuit regression (Friedman and Stuetzle 1981) except that subnetworks provide the nonlinearities instead of nonlinear smoothers.

Conclusion

The goal of creating artificial intelligence has lead to some fundamental differences in philosophy between neural engineers and statisticians. Ripley (1993) provides an illuminating discussion of the philosophical and practical differences between neural and statistical methodology. Neural engineers want their networks to be black boxes requiring no human intervention—data in, predictions out. The marketing hype claims that neural networks can be used with no experience and automatically learn whatever is required; this, of course, is nonsense. Doing a simple linear regression requires a nontrivial amount of statistical expertise.

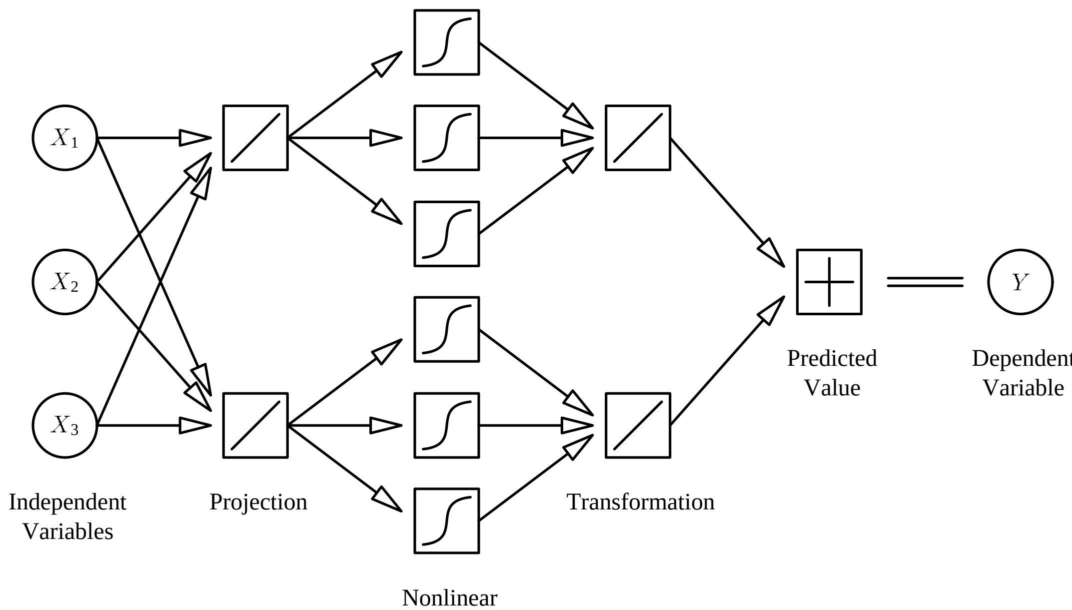

Figure 18: Projection Pursuit Network

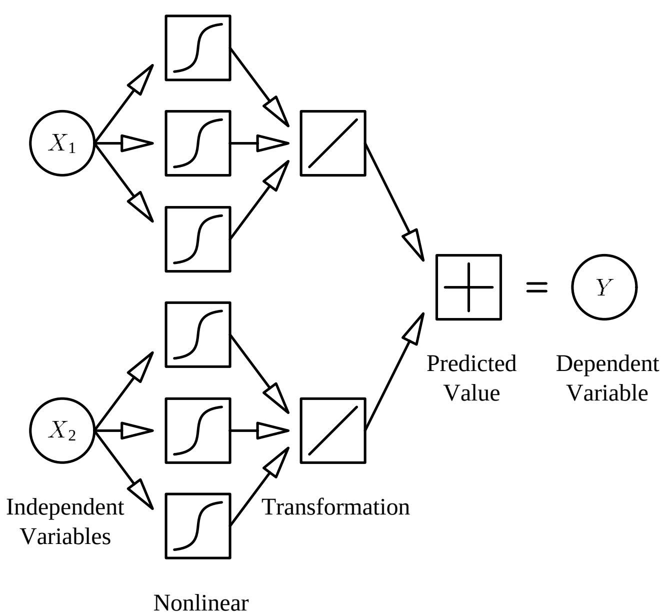

Figure 17: Generalized Additive Network

Using a multiple nonlinear regression model such as an MLP requires even more knowledge and experience.

Statisticians depend on human intelligence to understand the process under study, generate hypotheses and models, test assumptions, diagnose problems in the model and data, and display results in a comprehensible way, with the goal of explaining the phenomena being investigated. A vast array of statistical methods are used even in the analysis of simple experimental data, and experience and judgment are required to choose appropriate methods. Even so, an applied statistician may spend more time on defining the problem and determining what are the appropriate questions to ask than on statistical computation. It is therefore unlikely that applied statistics will be reduced to an automatic process or "expert system" in the foreseeable future. It is even more unlikely that artificial neural networks will ever supersede statistical methodology.

Neural networks and statistics are not competing methodologies for data analysis. There is considerable overlap between the two fields. Neural networks include several models, such as MLPs, that are useful for statistical applications. Statistical methodology is directly applicable to neural networks in a variety of ways, including estimation criteria, optimization algorithms, confidence intervals, diagnostics, and graphical methods. Better communication between the fields of statistics and neural networks would benefit both.

References

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

Bates, D.M. and Watts, D.G. (1988), *Nonlinear Regression Analysis and Its Applications*, New York: John Wiley & Sons.

Borowiak, D.S. (1989), *Model Discrimination for Nonlinear Regression Models*, New York: Marcel-Dekker.

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

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

Chen et al (1991), "Orthogonal least squares algorithm for radial-basis-function networks" *IEEE Transactions on Neural Networks*, 2, 302-309.

Cooper, L.N., Elbaum, C. and Reilly, D.L. (1982), *Self Organizing General Pattern Class Separator and Identifier*, U.S. Patent 4,326,259.

Cramer, J. S. (1986), *Econometric Applications of Maximum Likelihood Methods*, Cambridge, UK: Cambridge University Press.

Edwards, A.W.F (1972), *Likelihood*, Cambridge, UK: Cambridge University Press.

Everitt, B.S. (1980), *Cluster Analysis*, 2nd Edition, London: Heineman Educational Books Ltd.

Fortier, J.J. (1966), "Simultaneous Linear Prediction," *Psychometrika*, 31, 369-381.

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

Gallant, A.R. (1987), *Nonlinear Statistical Models*, New York: John Wiley & Sons.

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

Gifi, A. (1990), *Nonlinear Multivariate Analysis*, Chichester, UK: John Wiley & Sons.

Hand, D.J. (1981), *Discrimination and Classification*, New York: John Wiley & Sons.

Härdle, W. (1990), *Applied Nonparametric Regression*, Cambridge, UK: Cambridge University Press.

Hartigan, J.A. (1975), *Clustering Algorithms*, New York: John Wiley & Sons.

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

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

Hinton, G.E. (1992), "How Neural Networks Learn from Experience," *Scientific American*, 267 (September), 144-151.

Hotelling, H. (1933), "Analysis of a Complex of Statistical Variables into Principal Components," *Journal of Educational Psychology*, 24, 417-441, 498-520.

Hosmer, D.W. and Lemeshow, S. (1989), *Applied Logistic Regression*, New York: John Wiley & Sons.

Huber, P.J. (1985), "Projection pursuit," *Annals of Statistics*, 13, 435-475.

Jackson, J.E. (1991), *A User's Guide to Principal Components*, New York: John Wiley & Sons.

Jolliffe, I.T. (1986), *Principal Component Analysis*, New York: Springer-Verlag.

Jones, M.C. and Sibson, R. (1987), "What is projection pursuit?" *Journal of the Royal Statistical Society*, Series A, 150, 1-38.

Kaufmann, L. and Rousseeuw, P.J. (1990), *Finding Groups in Data*, New York: John Wiley & Sons.

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

McLachlan, G.J. (1992), *Discriminant Analysis and Statistical Pattern Recognition*, New York: John Wiley & Sons.

McLachlan, G.J. and Basford, K.E. (1988), *Mixture Models*, New York: Marcel Dekker, Inc.

Massart, D.L. and Kaufman, L. (1983), *The Interpretation of Analytical Chemical Data by the Use of Cluster Analysis*, New York: John Wiley & Sons.

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

Miller III, W.T., Glanz, F.H. and Kraft III, L.G. (1990), "CMAC: an associative neural network alternative to backpropagation," *Proceedings IEEE*, 78, 1561-1567.

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

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

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

Pao, Y (1989), *Adaptive Pattern Recognition and Neural Networks*, Reading, MA: Addison-Wesley.

Prager, R.W. and Fallside, F. (1989), "The Modified Kanerva Model for Automatic Speech Recognition," *Computer Speech and Language*, 3, 61-81.

Rao, C.R. (1964), "The Use and Interpretation of Principal Component Analysis in Applied Research," *Sankya*, Series A, 26, 329-358.

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

Ross, G.J.S. (1990), *Nonlinear Estimation*, New York: Springer-Verlag.

Schiøler, H. and Hartmann, U. (1992), "Mapping Neural Network Derived from the Parzen Window Estimator," *Neural Networks*, 5, 903-909.

Seber, G.A.F and Wild, C.J. (1989), *Nonlinear Regression*, New York: John Wiley & Sons.

Sneath, P.H.A. and Sokal, R.R. (1973), *Numerical Taxonomy*, San Francisco: W.H. Freeman.

Souček, B. and The IRIS Group (1992), *Fast Learning and Invariant Object Recognotion: The Sixth-Generation Breakthrough*, New York, John Wiley & Sons,

Spath, H. (1980), *Cluster Analysis Algorithms*, Chichester, UK: Ellis Horwood.

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

Titterington, D.M., Smith, A.F.M., and Makov, U.E. (1985), *Statistical Analysis of Finite Mixture Distributions*, New York: John Wiley & Sons.

Tukey, J.W. (1961), "Curves as Parameters and Touch Estimation," *Proceedings of the 4th Berkeley Symposium*, 681-694.

van den Wollenberg, A.L. (1977), "Redundancy Analysis—An Alternative to Canonical Correlation Analysis," *Psychometrika*, 42, 207-219.

Wasserman, P.D. (1993), *Advanced Methods in Neural Computing*, New York: Van Nostrand Reinhold.

Watson, G.S. (1964), "Smooth regression analysis," *Sankhya*, Series A, 26, 359-72.

Weisberg, S. (1985), *Applied Linear Regression*, New York: John Wiley & Sons.

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

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