A Brief Introduction to Neural Networks

Richard D. De Veaux
Williams College

Lyle H. Ungar
University of Pennsylvania

Abstract

Artificial neural networks are being used with increasing frequency for high dimensional problems of regression or classification. This article provides a tutorial overview of neural networks, focusing on back propagation networks as a method for approximating nonlinear multivariable functions. We explain, from a statistician's vantage point, why neural networks might be attractive and how they compare to other modern regression techniques.

KEYWORDS: neural networks; function approximation; backpropagation.

1 Introduction

Networks that mimic the way the brain works; computer programs that actually *learn* patterns; forecasting without having to know statistics. These are just some of the many claims and attractions of artificial neural networks. Neural networks (we will henceforth drop the term artificial, unless we need to distinguish them from biological neural networks) seem to be everywhere these days, and at least in their advertising, are able to do everything that statistics can do without all the fuss and bother of having to do anything except buy a piece of software.

Neural networks have been successfully used for many different applications; Pointers to some of the vast literature are given at the end of this article. In this article we will attempt to explain how one particular type of neural network, feedforward networks with sigmoidal activation functions ("backpropagation networks") actually works, how it is "trained", and how it compares with some more well known statistical techniques.

As an example of why someone would want to use a neural network, consider the problem of recognizing hand written ZIP codes on letters. This is a classification problem, where the inputs (predictor variables) might be a vector of grey scale pixels measuring the darkness of each small subdivision of the character. The desired response, y, is the digit of the ZIP

code. The modeling problem could be thought of as finding an appropriate relationship $y = f(x;\theta)$, where θ is a set of unknown parameters. This is a *large* problem: x may be of length 1,000 or more. The form of f is unknown — certainly not linear — and may require tens of thousands of parameters to fit accurately. Large data sets with hundreds of thousands of different handwritten ZIP codes are available for estimating the parameters. Neural networks offer a means of efficiently modeling such large and complex problems.

Neural networks differ in philosophy from many statistical methods in several ways. First, a network usually has many more parameters than a typical statistical model. Because they are so numerous, and because so many combinations of parameters result in similar predictions, the parameters become uninterpretable and the network serves as a black box estimator. Therefore, the network, in general, does not aid in understanding the underlying process generating the data. However, this is acceptable, and even desirable in many other applications. The post office wants to automatically read ZIP codes, but does not care what the form of the functional relationship is between the pixels and the numbers they represent. This is fortunate, since there is little hope of finding a functional relationship which will cover all of the variations in handwriting which are encountered. Some of the many applications where hundreds of variables may be input into models with thousands of parameters include modeling of chemical plants, robots and financial markets, and pattern recognition problems such as speech, vision and handwritten character recognition.

Neural network methods share a loose inspiration from biology in that they are represented as networks of simple neuron-like processors. A typical "neuron" takes in a set of inputs, sums them together, takes some function of them, and passes the output through a weighted connection to another neuron. Neural networks are often represented as shown in Figure 1. At first glance, such figures are not very revealing to a statistician. But as we will show, many familiar statistical techniques can be represented in this framework.

Each neuron may be viewed simply as a predictor variable, or as a combination of predictor variables. The connection weights are the unknown parameters which are set by a "training method". That is, they are estimated. The architecture, i.e., the choice of input and output variables, the number of nodes and hidden layers, and their connectivity, are usually taken as given. More will be said about this later.

Actual biological neural networks are incomparably more complex than their artificial counterparts. Real neurons are living cells with complex biochemistry. They transmit and process information by the build-up and release of ions and other chemicals, and so have memory on many different time scales. Thus, their outputs are often sequences of voltage spikes, which may increase or decrease in frequency depending on what signals the neuron received earlier. Real neural networks also have a complex connectivity, and a variety of mechanisms by which the networks elements and their connections may be altered.

Artificial neural networks retain only a small amount of this complexity and use simpler neurons and connections, but keep the idea of local computation. Many different network architectures are used, typically with hundreds or thousands of adjustable parameters. The

resulting equation forms are general enough to solve a large class of nonlinear classification and estimation problems and complex enough to hide a multitude of sins.

One advantage of the network representation is that it can be implemented in massively parallel computers with each neuron simultaneously doing its calculations. Several commercial neural network chips and boards are available, but most people still use neural network "simulators" on standard serial machines. This paper will not discuss parallel hardware implementation, but it is important to remember that parallel processing was one of the motivations behind the development of artificial neural networks.

We will focus primarily on neural networks for multivariable nonlinear function approximation (regression) and somewhat less on the related problem of classification. The most widely used network for these problems is the multilayer feedforward network. This is often called the "backpropagation network", although backpropagation actually refers to the training (or estimation) method. For a more advanced introduction to other types of neural networks which serve as classifiers or as data clustering methods, see Lippman (1987) or Ripley (1994). We formally define the backpropagation network in section 2. Here we also introduce several examples that will serve us throughout the paper and discuss how to estimate the parameters in a neural network. Section 3 involves the tradeoffs between training and overfitting. A brief overview of neural networks other than the backpropagation network is contained in section 4. A comparison to familiar statistical methods in contained in section 5. We conclude with a brief discussion in section 6.

2 Feedforward networks with sigmoidal activation functions

Consider the problem of predicting a response y_l from a set of predictor variables, $x_1, \ldots, x_L$ The backpropagation network with one hidden layer is nothing more than a statistical model of the following form:

$$\hat{y}_l = \sigma^{(2)} \left(\sum_{k=1}^{K} w_{2kl} \quad \sigma^{(1)} \left(\sum_{j=1}^{J} w_{1jk} x_j + \theta_j \right) + \theta_k \right) \tag{1}$$

where the functions $\sigma^{(1)}$ and $\sigma^{(2)}$ are generic transformations, known as activation functions, which may be linear or non-linear. Usually, $\sigma^{(1)}$ is a sigmoidal function such as the logistic $\sigma^{(1)}(z) = 1/(1+e^{-z})$ or $\sigma^{(1)}(z) = tan{-}1(z)$. For regression problems, $\sigma^{(2)}$ is often the identify, while for classification problems it is sigmoidal as well. Sigmoidal activation functions are also known as a squashing functions for obvious reasons. The terms θ_j and θ_k are constants that act as intercept terms in the model, called bias nodes.

For each k, the inner summation over j produces a weighted linear combination of the predictors. There are K such linear combinations, referred to as "hidden layer" variables, so

called because they are simply constructs of the original predictors that cannot be observed directly. The output, $\hat{y}_l$ is then just a (possibly non-) linear transformation of a weighted combination of possibly non-linear transformations of these variables. By allowing one or both of the transformations to be non-linear and by letting the number of hidden variables (or nodes) be large enough, the network has the flexibility to fit any reasonable function.

Again consider Figure 1. Here the predictor variables (or nodes) are shown on the left, connected by links to the hidden nodes. The links represent the weights. Sometimes a σ is put in the node to represent the transformation. The hidden nodes act as both predictors and responses: predictors to the node(s) on their right and responses to the nodes on their left. The bias nodes are indicated by the square nodes.

The bias node is an example of the nomenclature that can cause confusion for statisticians first encountering neural networks. Users of neural networks and statisticians often use different words to describe the same or very similar concepts. The following table contains a short glossary giving rough "translations" of some of the most important words we will be using.

Glossary

neural nets	statistics
weight	coefficient
bias term	constant term or intercept
input	predictor
output	response
exemplar	observation
training, learning	parameter estimation
backpropagation	an optimization method similar to steepest descent
sigmoidal	bounded, monotonic and differentiable
sigmoidal activation function	for example, $\sigma(x) = (1 + e^{-x})^{-1}$
radial basis function	a radially symmetric function, e.g. a Gaussian pdf

As an example of a very simple network, consider the network illustrated in figure 2, which contains three input nodes labeled x_1, x_2 and x_3, a bias node, θ_1, one output y and no hidden layer. If we let the activation functions be linear (the identity) throughout, then this is just a multiple linear regression in three predictor variables. If we replace the weights w by more familiar looking notation, equation 1 becomes:

$$\hat{y} = \sigma^{(1)}\left(\theta_1 + \sum_{j=1}^{3} w_{1j1}x_j\right) = b_0 + b_1x_1 + b_2x_2 + b_3x_3. \tag{2}$$

Now, add a hidden layer to this network, as shown in Figure 3, and take (possibly) nonlinear transformations of the intermediate values x_{11} and x_{12} and the final value. This gives a graphical representation of Equation 1 for $J = 3$, $K = 2$ and one response.

The nodes x_{21} and x_{22} in the hidden layer receive their input from the first layer. If $/sigma^{(1)}$ and $/sigma^{(2)}$ are identity transforms, one can understand the redundancy in neural nets. Then, for example, $x_{21} = \theta_{21} + w_{111}x_1 + w_{121}x_2 + w_{131}x_3$, a linear regression on x_1, x_2 and x_3. The output $y = \theta_2 + w_{211}x_{21} + w_{221}x_{22}$ is a linear regression on x_{21} and x_{22}. Rewriting the previous expression in terms of the original (first layer) input variables illustrates neural network overparameterization:

$$\hat{y} = \theta_2 + w_{211}(\theta_{21} + w_{111}x_1 + w_{121}x_2 + w_{131}x_3) + w_{221}(\theta_{22} + w_{112}x_1 + w_{122}x_2 + w_{132}x_3)$$

which can be written as

$$\begin{aligned} \hat{y} = (w_{201} + w_{211} * \theta_{21} + w_{221} * \theta_{22}) \quad + \\ (w_{211} * w_{111} + w_{221} * w_{112})x_1 + (w_{211} * w_{121} + w_{221} * w_{122})x_2 \quad + \\ (w_{211} * w_{131} + w_{221} * w_{132})x_3 \end{aligned}$$

There is clearly redundancy in this formulation of the regression of y on x_1, x_2 and x_3! Moreover, the individual coefficients, or weights w_{jkl} have been rendered virtually meaningless, since there are an infinite number of choices of them that lead to the same model. With nonlinear activation functions, one avoids the degenerate redundancy of the previous example, but the problems of overfitting and meaningless weight coefficients remain.

More layers of the network can also be used. Two hidden layers can be beneficial, but theory indicates that more than two hidden layers provides only marginal benefit in spite of the major increase in training time. Such networks are called "feedforward" because information is passed only in a "forward direction" from the inputs to the outputs; There are no connections from neurons backward to neurons located closer to the input. (See, by way of contrast, the description of recurrent nets below.)

2.1 Network training

Thus far, we have not addressed the method of estimating the parameters in the model — the weights. Generally, the weights are chosen so as to minimize the total residual sum of squares across all outputs y_l

$$E = \sum_l \sum_i (\hat{y}_{li} - y_{li})^2 \tag{3}$$

where y_{li} is the lth component of the response $\mathbf{y_i}$ to the ith data point in the training set. The sum is thus over both the responses l and all the data points i in the training set. Other error criteria can be - and occasionally are - used as well, including asymmetric cost functions. Often a "weight decay" term or regularization penalty such as the sum of the squares of all the weights is added to equation 3 in an attempt to avoid overfitting.

Once the network structure and the objective function have been chosen, this is simply a nonlinear least squares problem, and can be solved using any of the standard nonlinear least squares methods. For example, the weights can be estimated by any number of optimization methods such as steepest descent. For relatively small problems on serial machines, conjugate gradient methods work well. On parallel machines, or when one is not concerned with efficiency, a simple gradient descent method is often used.

The famous "backpropagation" algorithm or "delta learning rule" is simply (stochastic) gradient descent. At each iteration, each weight w_i is adjusted proportionally to its effect on the error

$$\Delta w_i = \alpha \frac{dE}{dw_i}. \tag{4}$$

The constant, α, controls the size of the gradient descent step taken and is usually taken to be around 0.1 to 0.5. If α is too small, learning is slow. If it is too large, instability results and the w_i do not converge. The derivative dE/dw_i can be calculated purely locally, so that each neuron needs information only from neurons directly connected to it. The basic derivation follows the chain rule: the error in the output of a node may be ascribed in part to errors in the immediately connected weights and in part to errors in the outputs of "upstream" nodes (Rumelhart, Hinton and Williams 1986). Both "batch" and "online" variations of the method are used in which, respectively, the weights are updated based on dE/dw_i for the entire data set (as above) or the weights are updated after each single additional data point is seen. The online method is particular useful for real time applications such as vehicle control based on video camera data.

When the chain rule is implemented on a massively parallel machine, it gives rise to the "backpropagation" in backpropagation networks: the error at an output neuron can be partly ascribed to the weights of the immediate inputs to the neuron, and partially to errors in the outputs of the hidden layers. Errors in the output of each hidden layer are due to errors in the weights on its inputs. The error is thus propagated back from the output through the network to the different weights, always by repeated use of the chain rule.

One can achieve faster convergence by adding a "momentum term" proportional to the previous change, Δw_i^{t-1}:

$$\Delta w_i = \alpha \frac{dE}{dw_i} + \eta \Delta w_i^{t-1}. \tag{5}$$

The momentum term approximates the conjugate gradient direction. There is an enormous literature on more efficient methods of selecting weights, most of which describes ways of adjusting α and η. On standard serial computers, it is generally more efficient to use conjugate gradient methods, which use second derivatives (the Hessian) and line search, rather than gradient descent to determine α and η.

Selecting weights is, of course, a highly nonlinear problem since the function being minimized, E, has many local minima. When different initial values for the weights are selected, different models will be derived. Small random values are traditionally used to initialize

the network weights. Alternatively, networks can be initialized to give predictions prior to training that replicate an approximate model of the data obtained, for example, by linear regression. Although the *models* resulting from different initial weights have very different parameter values, the fit and prediction errors generally do not vary too dramatically.

It is important to repeat that these networks are typically not used for interpretation. Since all neurons in a given layer are structurally identical, there is a high degree of collinearity between the outputs of the different neurons, so individual coefficients become meaningless. Also, the network structure permits high order interactions, so there are typically no simple substructures in the model to look at — in contrast with, for example, additive models. (However, neural network users often plot the pattern of outputs of the hidden neurons for different inputs for use as "feature detectors".) As mentioned above, this is not necessarily a problem, since one typically uses the network for prediction or classification and not for interpretation. For large, complex modeling problems like character recognition or speech-to-text translation, there need not be a simple, intelligible model which is accurate.

Neural networks have a number of attractive properties which, given the usual lists of technical preconditions, can be rigorously proven. Given enough neurons, and hence enough adjustable parameters, any well-behaved function can be approximated. Less obviously, neural networks can be viewed as providing a set of adaptive basis functions which put more basis functions where the data are known to vary more rapidly. This allows much more efficient representation for high dimensional inputs (many predictors) than using fixed basis functions such as polynomial regression or Fourier approximations (Barron 1996).

2.2 Example

To better understand how neural nets work on real problems, we begin with a multiple linear regression example using a set of data from a polymer process with 10 predictor variables $x_1, \ldots, x_{10}$ and a response, y. (Psichogios et al. 1993; Data are available via ftp at ftp.cis.upenn.edu: pub/ungar/chemdata).

It turns out that x_5 is a linear combination of the other predictors, so for now, we restrict ourselves to a regression of y on $x_1, ..., x_4, x_6, ..., x_{10}$. Linear regression gives the values shown in Table 1.

Regression Coefficients	
x1	-0.8298
x2	-0.7587
x3	-0.5282
x4	-0.7159
x6	0.8045
x7	0.0183
x8	0.0730
x9	0.0640
x10	-0.3943
Intercept	1.0147

Table 1

A two layer network with 9 input nodes (the $x's$), a bias node and linear activation functions was started from a random initial point for the weights. The estimated weights are shown in Table 2.

Neural network weights – linear activation functions –No hidden layer	
x1	-0.8298
x2	-0.7587
x3	-0.5282
x4	-0.7159
x6	0.8045
x7	0.0183
x8	0.0730
x9	0.0640
x10	-0.3943
bias	1.0147

Table 2

The network, as expected, reproduces exactly the linear regression.

One of the advantages of a neural network approach touted by their advocates is shown by leaving in the collinear predictor, x_5. In this case, the neural network gives the solution shown in Table 3.

Neural network weights – all 10 predictors	
x1	-0.1828
x2	-0.2411
x3	-0.3988
x4	-0.4571
x5	0.1294
x6	0.8045
x7	0.0183
x8	0.0730
x9	0.0640
x10	-0.3943
bias	0.3288

Table 3

The predicted values from this model agree with the predicted values from the regression model to eight decimal places.

Of course, for all of the above models, a plot of the predicted versus observed values is revealing, and is shown in Figure 4.

Clearly, a transformation of y and/or some of the predictors is called for. If one is interested only in better predictions, a neural network solution might be called for. To do this, we first add a hidden layer of five nodes. These five hidden layer nodes will be linear combinations of the original input variables. As we have seen, this alone will have no effect on the predicted values in our example. However, if we let them be sigmoidal nodes, (in particular we use the logistic function), we see a dramatic change in the predicted values shown in Figure 5.

Obviously one has to pay some price for this increased accuracy in the predictions. We have now estimated some 55 parameters (using 61 observations!), and one would expect the predictions to improve over the ten parameter linear regression model. The choice of how many nodes, and how many hidden layers is best determined by cross validation, although other strategies are often employed. These issues are discussed in the next section.

3 Architecture and Overfitting

Two more related problems must be addressed: how to determine the architecture (how many neurons and layers) and how to avoid overfitting. Since neural networks are highly redundant and overparameterized, it is easy to fit the noise in the data as well as the signal. Put differently, there is a tradeoff between reducing bias (fitting the training data well) and variance, which effects how well future data will be fit (Geman and Beinstock, 1992).

Although neural network practitioners often try a number of different architectures to see which give the best prediction, and many schemes have been proposed for automatically

selecting network structure (e.g. by removing less important neurons or links), the most common procedure is to select a network structure which has more than enough parameters and then to avoid the worst aspects of overfitting by the network training procedure. This is the strategy that was employed in the example above.

In most problems that a neural network practitioner would attack, data are abundant. In such cases, one could set aside some proportion (say half) of the data to use only for prediction after the model has been trained (parameters estimated). We call this the test set. The half used for estimation is called the training set. If one then plots mean squared error as a function of number of iterations in the solution procedure, the error on the training data decreases monotonically, while the error on the test set decreases initially and then passes through a shallow minimum and slowly increases. (See Figure 6.) The values of the weights from the minimum prediction error are then used for the final model. For small data sets, where splitting the data into two groups will leave too few observations for either estimation or validation, in principal at least, resampling or leave-one-out training may be used either to determine when to stop training or to determine the optimal network size.

One would expect that for a given data set there would be an optimal number of hidden neurons, with the optimum lying between one neuron (high bias, low variance) and a very large number of neurons (low bias, high variance). This is true for some data sets, but, counterintuitively, for most large data sets, increasing the number of hidden nodes continues to improve prediction accuracy, as long as cross validation is used to stop training. To our knowledge, although there has been extensive speculation, the reason for this has not been explained.

Since estimating the weights in a neural network is a nonlinear least squares problem, all of the standard nonlinear least squares solution and analysis methods can be applied. In particular, in recent years, a number of regularization methods have been proposed to control the smoothness, and hence the degree of overfitting, in neural networks. As just one example, one can use Bayesian techniques to select network structures and to penalize large weights or reduce overfitting (Buntine and Weigend 1991, MacKay 1992).

3.1 Example

Continuing with our regression example, we will use the hidden layer sigmoidal network with 5 nodes in the hidden layer, as before. Although it is possible to choose the architecture by cross-validation, we will illustrate the procedure of using what we presume to be an adequate architecture (with 55 parameters for 61 observations) and to select the amount of training by cross-validation instead.

To do this, we randomly select some percentage of the data (typically 10%) to use as the test data and use the remaining data as the training data. We then train the network, monitoring the residual sum of squares. We repeat this procedure with different test data several times. Looking at Figure 6, we see that the test set error decreases initially, but reaches a minimum at about 200 iterations. The training set error continues to decrease as one might expect.

Retraining the network on the entire data set using 200 iterations, gives a residual sum of squares of 0.0304 with the plot of predicted vs. actual values shown in Figure 7.

3.2 Second Example

If the data have a particularly simple structure a flexible method like a neural network model should be able to reproduce it. To illustrate this, we generated 100 observations from the following model:

$$y = 10 + 2x_1 - 3x_2 + x_3 + \epsilon$$

where $\epsilon \sim N(0, 1)$ A linear regression program gave:

```
b =
   10.3726
    1.2671          R^2 = 49.53
   -2.9396
    0.7958

Residual sum of squares = 99.30
```

With enough nodes and sigmoidal transfer functions, we could get a perfect fit to these data. A network with 20 nodes in the hidden layer, allowed to train to convergence, produced a residual sum of squares of 25.07.

```
>> weights20 = make_bpn(X,y,nodes,tf,[],2000);
GRADIENTS FUNCTIONS  OBJECTIVE    METHOD
     0        1         9083
     1        5        268.7   steepest descent
     2        8        250.6   FR conj grad
...
  2650     7953        25.07   FR conj grad
  2651     7956        25.07   FR conj grad
Training terminated: no more progress
```

Suspecting that we may have overfit to the data here, we cross-validate by resampling 90% of the data as training data and 10% of the data as test data. We observe the average error sum of squares on the test data as we allow the network to train:

This suggests using 20 iterations (as opposed to 7956!!!). Restarting the training but this time stopping the training at 20 iterations, we find:

```
>> weightscv20 = make_bpn(X,y,nodes,tf,[],20);
GRADIENTS FUNCTIONS  OBJECTIVE    METHOD
     0        1         8811
     1        4          279   steepest descent
     2        7        257.8   FR conj grad
```

```
    ...
    6       20        98.78   FR conj grad
Maximum number of function calls exceeded
```

Notice that the residual sum of squares is nearly identical to the linear regression fit. Moreover, a plot of the predicted values would indicate that predicted values from this network are nearly identical to linear regression!!

In conclusion, this cross-validation method starts with a large network, one large enough so that if allowed to train until convergence would most likely result in fitting the training data nearly perfectly. To avoid this, we stop the training early, affecting some sort of regularization to a random starting point. To decide how early to stop training this large network, we use the randomly chosen test sets multiple times and see where the performance on the test set is optimized. This *ad hoc* method seems to produce a reasonable trade off between fitting and overfitting and is the most commonly used method in practice. As mentioned above, many statisticians find early stopping too *ad hoc* and prefer a more systematic regularization such replacing the error criterion Equation 3 with

$$E = \sum_l \sum_i (\hat{y}_{li} - y_{li})^2 + \beta \sum_j w_j^2 \tag{6}$$

where j ranges over all the weights and β is chosen using cross validation.

4 Variations on the network theme

One of the reasons that researchers are excited about neural networks is that the network structure suggests new equation forms. Although networks of the form described above account for most of the commercially used networks, there is an extremely large research effort on alternate equational forms. This section gives a glimpse of some of the diversity in equational forms that are being studied.

Neural networks can come in many forms—so many that it appears that anything can be turned into a neural net (see e.g, Barron et al. 1992). This is partly true, in that any equation can be written as a set of nodes which perform operations like addition and multiplication, and a set of links which are associated with adjustable parameters. However, there is a common spirit between the many flavors of neural networks. Neural networks are inspired by biology and as such, most have neurons which either have a logistic "transfer function," representing the saturation which occurs in real neurons, or a Gaussian transfer function, representing "local receptive fields." (See below.) Equally importantly, but less often followed, neural networks tend to have purely local training methods, i.e. one need not compute a Jacobian or Hessian for the whole system of equations, but parameter updating can be done using only information which is available locally at each neuron.

One important non-biological variation on neural nets is to incorporate prior knowledge about a problem into the network. Neural networks can be built which have only partial

connectivity, when one believes that not all input variables interact. Networks can also incorporate first principles equations such as mass and energy balances (Psichogios and Ungar 1992), or known linear approximations to the process being modeled. They can take data which have been processed with a smoothing filter or can work with the loading vectors produced by linear projection methods such as Principle Components Analysis (PCA) or Partial Least Squares (PLS) (Holcomb and Morari 1992, Qin and McAvoy 1992).

In another set of variations, artificial neural network can be made which are continuous in time. Besides more accurately reflecting biological neurons (an unclear advantage), continuous time networks are more naturally implemented in some types of computer chips (DeWeerth et al. 1991).

4.1 Recurrent networks

One of the more important variations on feedforward networks arises when the network is supplemented with delay lines, so that outputs from the hidden layers can be remembered and used as inputs at a later time. Such networks are said to be recurrent.

In linear time series analysis it is common to use ARMA models of the form

$$y(t) = \alpha_o x(t) + \alpha_1 x(t-1) + \alpha_2 x(t-2) + \ldots + \beta_1 y(t-1) + \beta_2 y(t-2) + \ldots \tag{7}$$

It is not clear how to best generalize this to the nonlinear case

$$y(t) = f(x(t), x(t-1), x(t-2), \ldots, y(t-1), y(t-2), \ldots), \tag{8}$$

where there may be arbitrary interactions between the lagged predictor and response variables. Neural networks provide "nonparametric" methods of estimating such functions.

The most obvious neural network structure to use is a nonlinear ARMA (NARMA) model where past values of x and y are used as inputs. (See Figure 8a.) A more parsimonious model can, however, be built by developing a model with internal memory. (See Figure 8b.) Instead of giving the network lagged values of x and y, the network receives the current value of x and uses internal memory. The links labeled with "-1" in Figure 8b are delay lines which give as output the input that they received on the previous time step. The outputs of neurons which receive delayed inputs are weighted combinations of the inputs and so they can be thought of as "remembering" features. The network thus stores only the most important combinations of past x values, giving models which require fewer adjustable parameters than ARMA models. Efficient training of such networks requires clever methods (Werbos 1990).

4.2 Radial basis functions

There are many other memory-intensive methods of approximating multivariable nonlinear functions which are loosely inspired by neurons. Radial basis functions (RBFs) (Broomhead and Low 1988; Moody and Darken 1989; Poggio and Girosi 1990) are one of the most important. Although they are often called neural networks, and although they do have

some biological motivation in the "receptive fields" found in animal vision systems, the training methods used for radial basis functions are even less like those of real neurons than backpropagation.

The idea behind radial basis functions is to approximate the unknown function $y(x)$ as a weighted sum of basis functions, $\phi(x; \boldsymbol{\mu}, \Sigma)$, which are n-dimensional Gaussian probability density functions. Typically Σ is chosen to be diagonal, with all elements equal to σ. Then,

$$\hat{y} = \sum \alpha_j \phi(x; \boldsymbol{\mu}_j, \sigma_j). \tag{9}$$

Once the basis functions are picked (*i.e.* their centers, $\boldsymbol{\mu}_j$, and widths, σ_j, are determined), the coefficients α_j can be found by linear regression. The basis functions can be placed at regular intervals in the input space, but this does not give the efficient scaling properties that follow from choosing the basis functions "optimally" for the data at hand. A preferable method is to group the data into clusters using k-means clustering, and center a basis function at the centroid of each cluster. The width of the Gaussians can then be taken as half the distance to the nearest cluster center.

RBFs are more local than sigmoidal networks. Changes to the data in one region of the input space have virtually no effect on the model in other regions. This offers many advantages and one disadvantage. The disadvantage is that RBFs appear to work less well for very high dimensional input spaces. Among the advantages are that the local basis functions are well-suited to online adaptation. If the basis functions are not changed, then retraining the network each time new data are observed only requires a linear regression. Equally importantly, unlike sigmoidal networks, new data will not affect predictions in other regimes of the input space for these networks. This has been used to good effect in adaptive process control applications (Sanner and Slotine 1991, Narendra and Parthasarathy 1990).

Radial basis functions also have the advantage that their output provides a local measure of data density. When there are no data points in a region of input space, there will be no basis functions centered nearby, and since the output of each Gaussian decreases with distance from its center, the total output will be small. This property of RBFs can be used to build a network which warns when it is extrapolating (Leonard, Kramer and Ungar 1992).

Again, many variations have been studied: the Gaussians can be selected to be elliptical, rather than spherical, or other forms of local basis function can be used.

5 Comparison with other methods

Statisticians and engineers have developed many methods for nonparametric multiple regression. Many are limited to linear systems or systems with a single dependent and independent variable, and as such, do not address the same problem as neural networks. Others such as multivariate regression splines (MARS, Friedman 1991), generalized additive models (Hastie and Tibshirani 1986) projection pursuit regression (Friedman and Stutzle 1981), compete directly with neural nets. It is informative to compare these methods in their philosophy and their effectiveness on different types of problems.

Regression methods can be characterized either as projection methods, where linear or non-linear combinations of the original variables are used as predictors (such as principal components regression), or subset selection methods, where a subset of the original set of input variables is used in the regression (*e.g.* stepwise regression). Table 4 characterizes several popular methods. Neural networks are a nonlinear projection method: they use non-linear combinations of optimally weighted combinations of the original variables to predict the response.

	Projection	Selection
Linear	PCA,PLS	Stepwise linear regression
Nonlinear	Neural Networks Nonlinear PLS Proj Pursuit	MARS, CART GAM

Table 4

As a first generalization of standard linear regression, one might consider an additive model of the form:

$$E(Y|X_1,\ldots,X_p) = \alpha + \sum_i^p f_j(X_j) \tag{10}$$

where the f_j are unknown but smooth functions, generally estimated by a scatterplot smoother. (Friedman and Silverman 1989, Hastie and Tibshirani 1990). One can generalize such models by assuming that a function of $E(Y|X_1,\ldots,X_p)$ is equal to the right hand side of equation 10:

$$g(E(Y|X_1,\ldots,X_p)) = \alpha + \sum_i^p f_j(X_j) \tag{11}$$

The function $g()$ is, as with generalized linear models, referred to as the link function, and the model in equation 11 is known as a generalized additive model (Hastie and Tibshirani, 1986, 1990). One can add terms $f_{jk}(X_j, X_k)$ to equation 11. However, with many predictors, the computational burden of choosing which pairs to consider can soon become overwhelming. Partially as an attempt to alleviate this problem, Friedman devised the multivariate adaptive regression splines (MARS) algorithm (Friedman 1991). MARS selects both the amount of smoothing for each predictor and the interaction order of the predictors automatically. The final model is of the form:

$$E(Y|X_1,\ldots,X_p) = \alpha + \sum_{m=1}^{M} \alpha_m \prod_{k=1}^{K_m} [s_{km}(x_{\nu(k,m)} - t_{km})]_+ \tag{12}$$

Here $s_{km} = \pm 1$ and $x_{\nu(k,m)}$ is one of the original predictors. (For more information see Friedman 1991). One of the main advantages of MARS is the added interpretability gained by variable selection, and the graphical displays of low order main effect and interaction terms. However, in cases of highly collinear predictors, where some dimension reduction is warranted, MARS can behave erratically and the interpretability advantage may be lost (De Veaux *et al.* 1993a or 1993b).

Projection pursuit regression also allows interactions by allowing linear combinations:

$$E(Y|X_1, \ldots, X_p) = \alpha + \sum f_{im}(\sum \alpha_{im}^{(j)} X_j). \tag{13}$$

The 'directions' are found iteratively by numerically minimizing the fraction of variance unexplained by a smooth of y versus $\sum \alpha_{im}^{(j)} X_j$ (Friedman and Stuetzle 1981). As linear combinations are added to the predictors, the residuals from the smooth are substituted for the response.

Models that use the original predictors and their interactions, like generalized additive models and MARS may be more interpretable than a projection method like projection pursuit regression if the original predictors have an intrinsic meaning to the investigator. On the other hand, if the predictors are correlated, the dimension reduction of the projection methods may render the final model more useful.

All of the statistical methods above differ in some degree from the artificial neural network models we have discussed in philosophy, in that they search for and exploit low dimensional structure in the data. They then attempt to model and display it, in order to gain an understanding of the underlying phenomenon. Backpropagation nets often compete quite favorably, and in fact can outperform these statistical methods in terms of prediction, but lack the interpretability. While all of the above methods can in theory be extended to the case of multiple responses, at present one must estimate each response separately. Neural networks, on the other hand can be used directly to predict multiple responses.

6 Conclusions

The advent of computers with sufficient speed and memory to fit models with tens of thousands of parameters is leading to a new approach to modeling, typified by neural networks. Neural networks are typically used by modelers who have a different philosophy than statisticians: neural network models are very accurate—if sufficient data of sufficient quality are available—but not easily interpreted. Because of both the network's overparameterization and the numerous local minima in its parameter search, the final parameter estimates are unstable, much like any highly multicollinear system. However, prediction on points much like the training data will be unaffected. How the prediction is affected on very different data is less clear. Statisticians have historically preferred models where the terms and coefficients can be examined and explained. This may partly explain — along with a distrust of the hype surrounding neural nets — why the area of neural networks has until recently suffered from relative neglect by statisticians (but cf. Ripley 1993, White 1989, Geman *et al.* 1992 and others).

One of the attractions of neural networks to their users is that they promise to avoid the need to learn other more complex methods — in short, they promise to avoid the need for statistics! This is largely not true: for example, outliers should be removed before training neural nets, and users should pay attention to distributions of errors as well as summaries such as the mean (see De Veaux *et al.* 1993b). However, the neural network promise of "statistics-free statistics" is partly true. Neural networks do not have implicit assumptions of linearity, normal or iid errors, etc. as many statistical methods do, and the bounded equational form of sigmoids and Gaussians appears to be much more resistant to the ill-effects of outliers than polynomial basis functions. Compared to subset selection methods, neural networks are relatively robust to high influence points and outliers. Although many of the claims made about neural networks are exaggerated, they are proving to be a useful tool and have solved many problems where other methods have failed.

Many open questions remain. There is a need for better understanding of the effect of nonuniform data distribution and its effect on local error. Most neural networks currently used do not provide any confidence limits (but see Leonard et al, 1992, Hwang et al., 1996, Baxt and White 1995, Deveaux 1996). Current methods of model selection are inefficient. Better methods for cross-validation and local error estimation are needed. There is some disagreement on the advantages of preprocessing data (e.g. filtering time series data or taking ratios of predictor variables). Better experimental design is needed for nonlinear dynamical systems.

Neural networks seem to have established themselves as a viable tool for nonlinear regression, and should join projection pursuit regression, CART, MARS, ACE, and generalized additive models in the toolkit of modern statisticians. The prevalence of large data sets and fast computers will increasingly push statisticians to use and refine such regression methods which do not require specification of a simple model containing few parameters.

7 Where to find information about neural networks

There is a prodigious literature on neural networks. A good place to find more basic information is in textbooks such as (Bishop 1995, Ripley 1996, Haykin 1994; Herz, Krogh and Palmer 1991) and in review articles by statisticians such as (Cheng and Titterington 1994; Ripley 1994) and for chemometricians, (Wythoff 1993). A particularly nice set of references is available in the Neural Network FAQ (FAQ.NN) available at the web site http://wwwipd.ira.uka.de/prechelt/FAQ/neural-net-faq-html and on the newsgroup "comp.ai.neural-nets".

For more up-to-date research, see recent proceedings of the NIPS or IJCNN conferences. Several books provide good descriptions of specific applications; see, for example, the collection of articles Neural Networks for Control, edited by W.T.Miller et al (1990), which focuses on robotics, Kulkarni (1994) for image understanding, Mommone (1994) for speech and vision, Pomerleau (1990) for autonomous vehicle navigation and Zupan and Gasteiger (1991) for chemistry.

For more neural network papers and software than you want, see the ftp site ftp.funet.fi in directory /pub/sci/neural.

8 Acknowledgments

The authors would like to thank Robert D. Small, J. Stuart Hunter and Brian Ripley, among others, who saw earlier versions of the draft and whose insightful comments led to significant improvements in the exposition. The authors would also like to thank the editor and two anonymous referees for their constructive comments. This work was funded in part by NSF Grant CTS95-04407.

References

Barron, A.R. (1994), "Approximation and Estimation Bounds for Artificial Neural Networks." *Machine Learning* 14, 115-133

Barron, A.R., Barron, R.L., and Wegman, E.J. (1992), "Statistical learning networks: A unifying view", in *Computer Science and Statistics: Proceedings of the 20th Symposium on the Interface*, edited by E.J. Wegman, 192-203.

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

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

Broomhead, D.S., and D.Lowe (1988), "Multivarible functional interpolation and adaptive networks." *Complex Systems* 2, 321-355.

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

Cheng, B. and D.M. Titterington (1994), "Neural Networks: a review from a statistical perspective, with discussion." Stat. Science 9(1) 2-54

De Veaux, R. D., Psichogios, D. C., and Ungar, L. H. (1993a) "A Tale of Two Non-Parametric Estimation Schemes: MARS and Neural Networks." in *Fourth International Workshop on Artificial Intelligence and Statistics.*

De Veaux, R. D., Psichogios, D. C., and Ungar, L. H. (1993b) "A Comparison of Two Non-Parametric Estimation Schemes: MARS and Neural Networks." *Computers and Chemical Engineering*, **17**(8), 819-837.

De Veaux, R. D., et al. (1996) "Applying Regression Prediction Intervals to Neural Networks" in preparation.

DeWeerth, S.P., Nielsen, L., Mead, C.A., Astrom, K.J. (1991), "A Simple neuron servo," *IEEE Transactions on Neural Nets*, **2**(2), 248-251.

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

Friedman, J.H. and Silverman, B., (1987) "Flexible parsimonious smoothing and additive modeling ",Stanford Technical Report, Sept. 1987.

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

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

Hastie, T. and Tibshirani, R., (1986),"Generalized Additive Models." *Statistical Science* **1:3**, 297-318.

Hastie, T.J. and Tibshirani, R.J., Generalized Additive Models, Chapman and Hall, London, 1990

Haykin, S., Neural Networks: A comprehensive Foundation, Macmillan, NY, 1994

Hertz,J., Krogh A., and Palmer, R.G. (1991). *Introduction to the Theory of Neural Computation*, Addison-Welsey, Reading, MA.

Holcomb, T. R., and Morari, M. (1992) "PLS/Neural Networks." *Computers and Chemical Engineering* **16:4**, 393-411.

Hwang, J.T. Gene and A. Adam Ding. "Prediction Intervals in Artificial Neural Networks" (1996).

Kulkarni, A.D. (1994), "Artificial neural networks for image understanding" Van Nostrand Renhold, New York.

Leonard J., Kramer, M., and Ungar, L.H. (1992) "Using radial basis functions to approximate a function and its error bounds." *IEEE Transactions on Neural Nets*, **3**(4), 624-627.

Lippmann, R.P., (1987) "An introduction to computing with neural nets." *IEEE ASSP Magazine*, 4-22.

MacKay, D.J.C., (1992) "A Practical Bayesian Framework for Backpropagation Networks." *Neural Computation*, **4**, 448-472.

Mammone, R.J., editor, (1994) "Artificial neural networks for speech and vision" Chapman and Hall, New York.

Miller, III, W.T, Sutton, R.S., and Werbos. P.J., eds. (1990) *Neural Networks for Control* MIT Press, Cambridge, Mass..

Moody, J. and Darken, C.J., (1989) "Fast learning in networks of locally tuned processing units." *Neural Computation*, **1**, 281-329.

Narendra, K. S., and Parthasarathy, K. (1990) "Identification and Control of Dynamical Systems Using Neural Networks." *IEEE Transactions on Neural Networks* **1**, 4-27.

Poggio, T., and Girosi, F. (1990) "Regularization algorithms for learning that are equivalent to multilayer networks." *Science* **247**, 978-982.

Pomerleau, D.A. (1990) "Neural network based autonomous navigation." *Vision and Navigation: The CMU Navlab* Charles Thorpe, (Ed.) Kluwer Academic Publishers.

Psichogios, D.C. and Ungar, L.H. (1992) "A Hybrid Neural Network — First Principles Approach to Process Modeling." *AIChE Journal*, **38**(10), 1499-1511.

Qin, S. J., and McAvoy, T. J. (1992) "Nonlinear PLS Modeling Using Neural Networks." *Computers and Chemical Engineering* **16:4**, 379-391.

Ripley, B.D. (1993) "Statistical Aspects of Neural Networks." In *Networks and Chaos - Statistical and Probabilistic Aspects* (eds. O.E. Barndorff-Nielsen, J. L. Jensen and W.S. Kendall),40-123,Chapman and Hall, London.

Ripley, B.D. (1994), "Neural networks and related methods for classification." Journal of the Royal Statistical Society Series B 56(3) 409-437

Ripley, B.D. (1996) "Statistical Aspects of Neural Pattern Recognition and Neural Networks" Cambridge University Press.

Rumelhart, D., Hinton, G., and Williams, R. (1986) "Learning Internal Representations by Error Propagation." *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol 1: Foundations* Cambridge: MIT Press, 318-362.

Sanner, R.M. and Slotine J.-J.E., (1991) "Direct Adaptive Control with Gaussian Networks." *Proc. 1991 Automatic Control Conference*, **3**, 2153-2159.

Werbos, P., (1990) "Backpropagation through time: what it does and how to do it." *Proceedings of the IEEE* **78**:1550-60.

White, H., (1989) "Learning in neural networks: a statistical perspective." *Neural Computation*, **1**(4), 425- 464.

Wythoff, B.J. (1993) "Backpropagation neural networks - a tutorial." Chemometrics and Intelligent laboratory systems 18(2) 115-155

Zupan, J. and Gasteiger, J. (1991) "Neural networks: a new method for solving chemical problems or just a passing phase?" *Analytica Chimica Acta*, **248**, 1-30.

Figure Captions

Figure 1: Feedforward Sigmoidal ("Backpropagation") Network

Figure 2: Linear Regression Network

Figure 3: Hidden Layer Regression Network

Figure 4: Plot of values predicted by linear regression vs. actual values for the polymer process data.

Figure 5: Plot of values predicted by a sigmoidal network trained to completion vs. actual values for the polymer process data.

Figure 6: Plot of mean squared error as a function of number of iterations for five different training sets and five different test sets. Note that the training set error decreases monotonically, while the testing set error initially decreases and then increases due to overfitting.

Figure 7: Plot of values predicted by a sigmoidal network trained for 200 iterations vs. actual values for the polymer process data.

Figure 8: Recurrent Neural Network

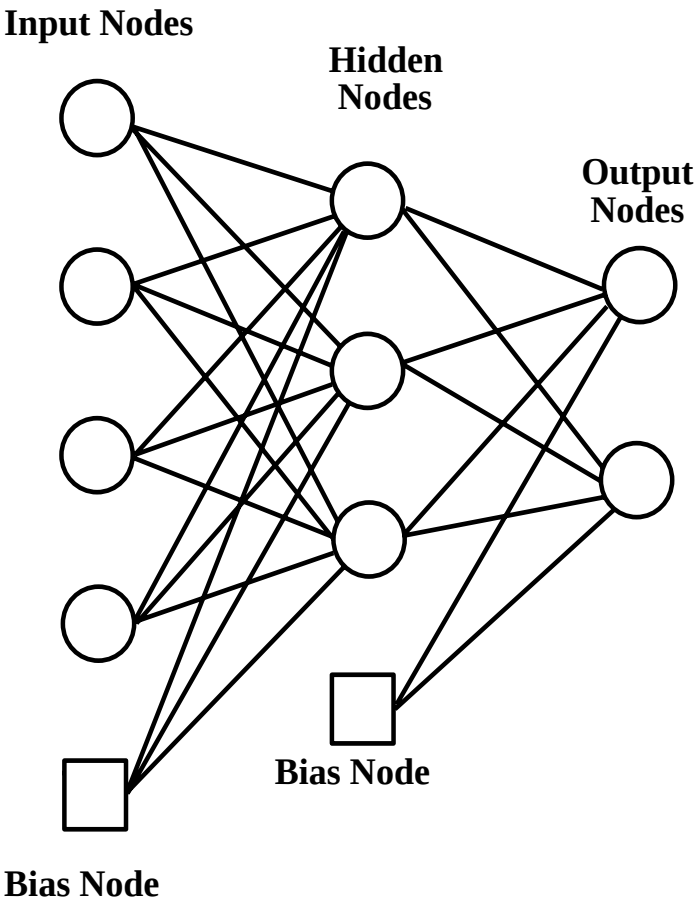

Figure 1

Linear Regression Network

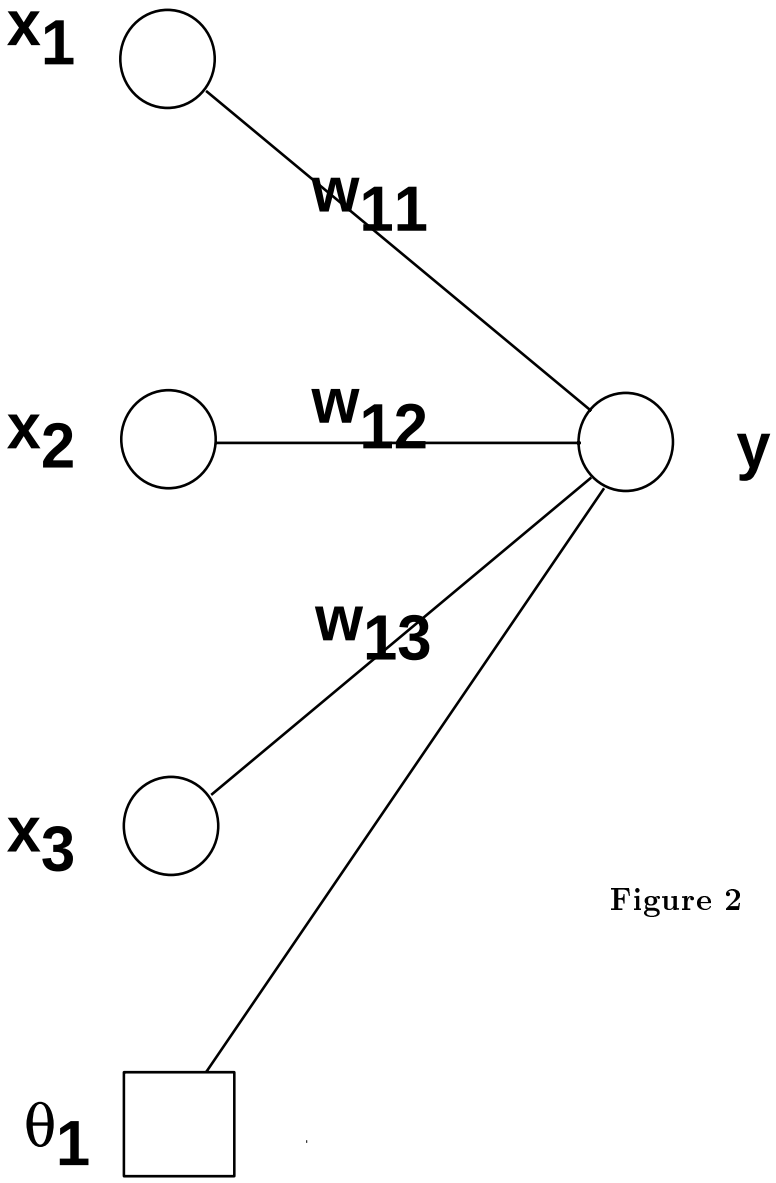

Figure 2

Hidden Layer Regression Network

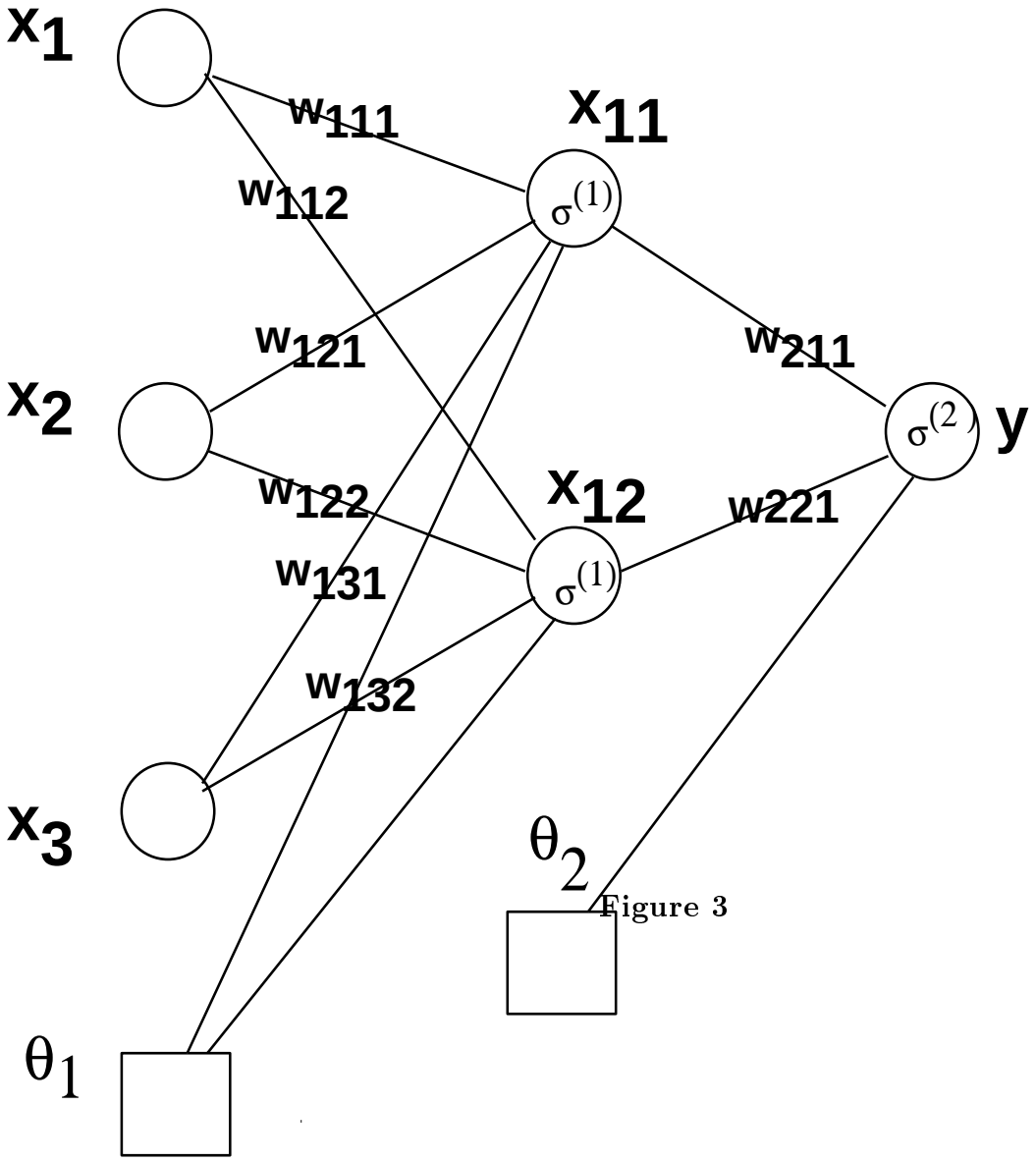

Figure 3

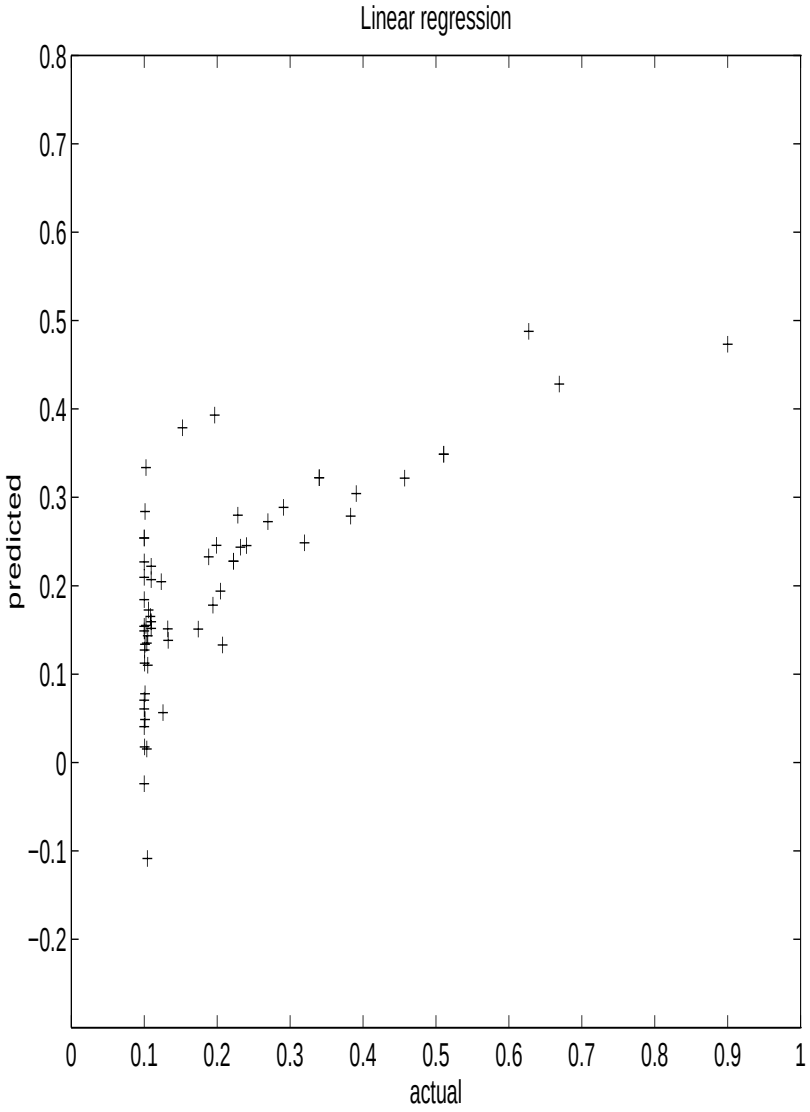
Linear regression
predicted
actual
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Figure 4

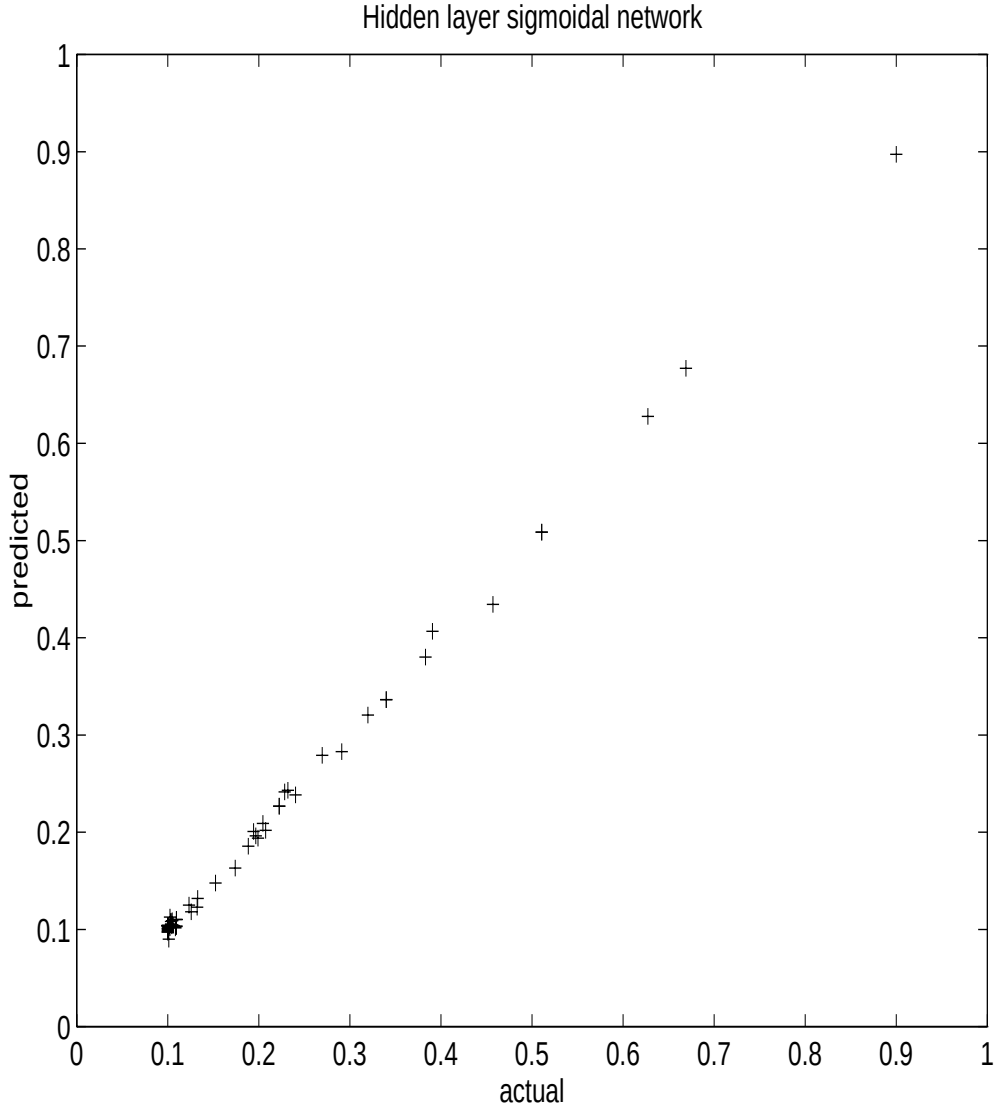
Hidden layer sigmoidal network
predicted
actual
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Figure 5

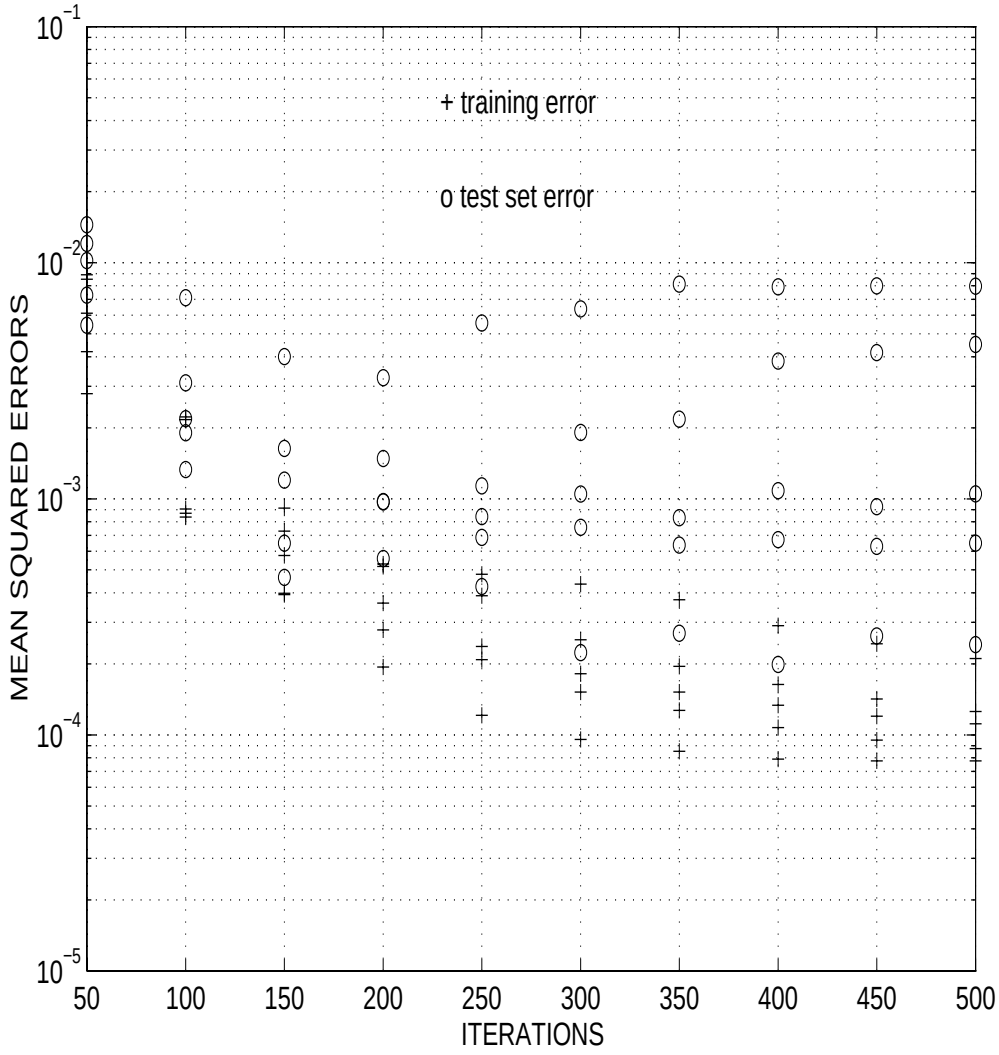
+ training error
o test set error
MEAN SQUARED ERRORS
ITERATIONS
10^-1
10^-2
10^-3
10^-4
10^-5
50
100
150
200
250
300
350
400
450
500

Figure 6

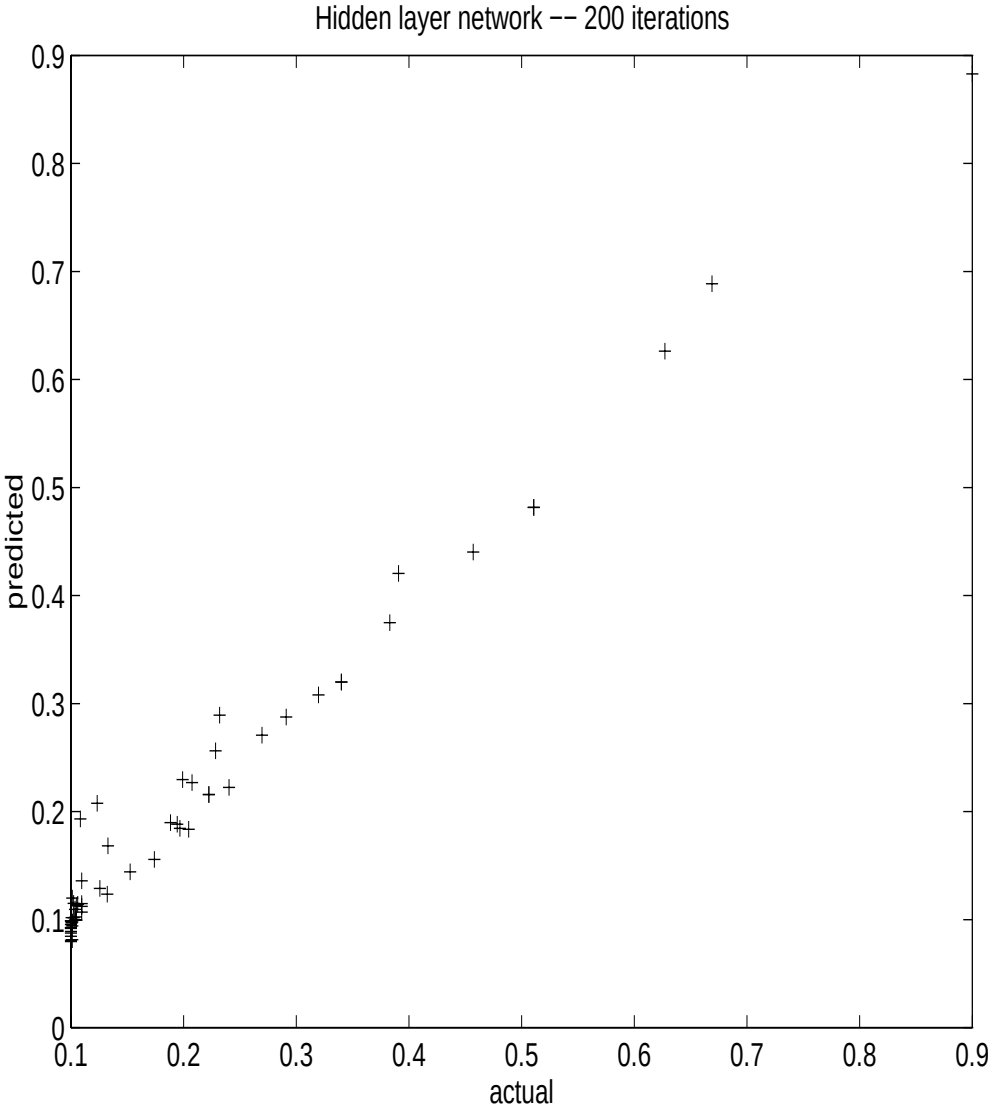
Hidden layer network −− 200 iterations
predicted
actual
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

Figure 7

Figure 8