A neural network construction algorithm which maximizes the likelihood function

Rudy Setiono
Department of Information Systems and Computer Science
National University of Singapore
Kent Ridge, Singapore 0511
Republic of Singapore
department website: rudys@iscs.nus.sg
email: rudys@iscs.nus.sg
phone: (65) 772-6297

---

1Running heading: Likelihood maximizing network
Abstract

A new method for constructing a feedforward neural network is proposed. The method starts with a single hidden unit and more units are added to the hidden layer one at a time until a network that completely recognizes all its input patterns is constructed. The novel idea about this method is that the network is trained to maximize a certain likelihood function and not to minimize the more widely used mean squared error function. We show that when a new hidden unit is added to the network, this likelihood function is guaranteed to increase and this increase ensures the finite termination of the method. We also provide a wide range of numerical results. The method was tested on the $n$-bit parity problems and the spiral problem. It was able to construct networks having less than $n$ hidden units that solve the $n$-bit parity problems for $n = 4, 5, 6, 7$ and $8$. The method was also tested on some real world data and the networks it constructed were shown to be able to predict patterns not in the training set with more than 95% accuracy.
1. Introduction

Neural network techniques have recently attracted a great deal of interest of many researchers as more problem areas in which these techniques can be applied are continually being discovered. These techniques have been used in diverse areas such as engineering, medicine and business. Feedforward neural networks that are trained via the back propagation method have been widely used for these applications. One of the drawback of the backpropagation method (Rumelhart & McClelland, 1986) is that one needs to determine the number of hidden units to be included in the network before the training of the network can be started. Different problem applications require network with different topology and often the number of hidden units is determined by trial and error. Once the number of hidden units is decided, it is not unusual that the training of the network leads to a local minimum point of the sum of the squared error function where the recognition rate of the network is not very satisfactory. In this paper, we propose an algorithm for constructing feedforward neural networks. This algorithm automatically adds hidden units to the network topology as they are needed. It is also guaranteed to terminate with a final network that correctly classifies all its inputs.

The problem that we will consider here is that of distinguishing elements from two disjoint sets in n-dimensional space. We wish to construct a feedforward neural network that maps a given n-dimensional input pattern into one of the two element in the binary set \{0, 1\}. The topology of a feedforward neural network is depicted in Figure 1 below.
Output $\in \{0, 1\}$

Input: $x \in \mathbb{R}^n$

Figure 1: Neural network with 7 hidden units.
The minimum number of hidden units required by the network to correctly classify all the input patterns is usually determined by trial and error. This is a very time consuming process, since one normally starts with a large number of hidden units and subsequently prunes the units that are redundant (Chung, 1992; Mozer, 1989).

Algorithms which automatically generate neural networks have been proposed by many researchers. These methods include the cascade correlation algorithm (Fahlman & Lebiere, 1989), the tiling algorithm (Mezard, 1989), the self-organizing neural network (Tenorio & Lee, 1990), and the upstart algorithm (Frean, 1990). For a given problem, these algorithms will generally build networks with many layers. It is widely known however, that a single hidden layer feedforward neural network can form an arbitrary decision boundaries if there are sufficient number of hidden units in the hidden layer (Hartman et al., 1990; Hornik, 1991).

The dynamic node creation method proposed by Ash (1989) is an algorithm which constructs neural networks with a single hidden layer. The method creates feedforward neural networks by sequentially adding hidden units to the hidden layer. The algorithm proposed in this paper is similar to the dynamic node creation method. It starts with a single hidden layer network consisting of a single hidden unit and find a set of optimal weights for this network. If the network with these weights does not solve the problem, then one hidden unit is added to the network and the network is retrained. The process is repeated until a network that correctly classifies all the input patterns has been constructed.

In our previous work (Setiono & Hui, 1994) the problem of finding a set of optimal weights for the growing network was cast as the following unconstrained optimization problem to minimize the sum of the squared errors (Mangasarian, 1993):

\[
f(w, v) := \sum_{i=1}^{k} (S^i - t^i)^2, \tag{1}
\]

where

\[ S^i = s \left( \sum_{j=1}^{h} g(x^i w^j + \tau^j) v^j \right) \]

\[ h = \text{fixed integer number of hidden units} \]

\[ k = \text{fixed integer number of samples } x^i \in \mathbb{R}^n \]

\[ t^i = 0 \text{ or } 1 \text{ target value for } x^i, i = 1, 2, \ldots, k \]

\[ v^j = \text{real number weights of outgoing arcs from hidden units, } j = 1, 2, \ldots, h \]

\[ w^j = n\text{-vector weights of incoming arcs to hidden units, } j = 1, 2, \ldots, h \]

\[ x^i = \text{given } n\text{-dimensional vectors samples, } i = 1, 2, \ldots, k \]

\[ s(\xi) = 1 \text{ if } \xi > 0 \text{ else } s(\xi) = 0. \]
$g(\xi) =$ hidden unit activation function.

$\tau_j =$ a real valued bias or threshold of hidden unit $j$.

The discontinuous function $s(\xi)$ is normally replaced by the sigmoid function

$$\sigma(\xi) = 1/(1 + e^{-\xi})$$

in order to render the error function $f(w, v)$ differentiable.

The hidden unit activation function $g(\xi)$ is usually either the sigmoid function (2) or the hyperbolic tangent function

$$\psi(\xi) = (e^\xi - e^{-\xi})/(e^\xi + e^{-\xi}).$$

For all the results reported in this paper, we have used the hyperbolic tangent function (3) as the hidden unit activation function. We also omit the bias $\tau_j$ from all hidden units by considering it as the weight $w_{i+1}^j$ whose corresponding input $x_{i+1}^j = 1, \forall i = 1, 2, \ldots, k$. We will still however, denote the dimensionality of $x$ and $w$ by $n$ throughout this paper.

Instead of the sum of the squared error function (1), any function that attains its minimum or maximum when the output values from the network are equal to the target values can be used to compute these weights. Baum and Wilczek (1988) suggested that the weights of the network are computed such that they maximize the likelihood function:

$$g(w, v) := \prod \left( \prod_{i \in I} S^i \prod_{i \notin I} (1 - S^i) \right),$$

or equivalently,

$$h(w, v) := \sum_{i \in I} \log S^i + \sum_{i \notin I} \log(1 - S^i),$$

where $I := \{i|t^i = 1\}$.

An extension of the case where the $t^i$ are probabilities having values in $[0, 1]$ instead of the actual target values are also given and a natural interpretation of this likelihood function is described (see also Hertz et al., 1991). The reason why we have picked to use this likelihood function over the usual sum of squared error function (1) for constructing a neural network is detailed in the Section 2. In this section we also formally describe our algorithm for neural network construction. In Section 3, we will describe a variant of quasi-Newton method for unconstrained minimization. This method is chosen because of its speed of convergence compared to the conventional backpropagation method for neural network training. Results from our numerical experiments are given in Section 4, and finally a brief summary of the paper is given in Section 5.

A brief word about our notation now. For a vector $x$ in the $n$-dimensional real space $\mathbb{R}^n$, the norm $\|x\|$ denotes the Euclidean distance of $x$ from the origin, that is, $\|x\| = (\sum_{i=1}^{n} x_i^2)^{1/2}$. For a matrix $A \in \mathbb{R}^{m \times n}$, $A^T$ will denote the transpose.
of $A$. The superscript $T$ is also used to denote the scalar product of two vectors in $\mathbb{R}^n$, that is $x^T y = \sum_{i=1}^{n} x_i y_i$. For clarity, often this superscript is dropped from a scalar product. For a differentiable function $f(x)$, the gradient of $f(x)$ is denoted by $\nabla f(x)$. Throughout this paper, we consider an input pattern to be correctly classified if the following condition is satisfied

$$|e^i| = |S^i - t^i| \leq 0.1.$$  

A problem is said to be solved if and only if for all patterns $x^i, i = 1, 2, \ldots k$ the above condition is satisfied.

2. Constructing feedforward neural network

The limitation of the backpropagation method for training feedforward neural network is well known. During training, the algorithm often gets stuck at a local minimum of the error function where the recognition rate of the network is not very good. Many approaches have been proposed to remedy this situation. These include the use of random optimization techniques (Baba, 1989), genetic algorithms (Baba, 1992; Whitley & Hanson, 1989), and simulated annealing (Makram-Ebeid et al., 1989).

The purpose of the algorithm that we are proposing here is to improve the recognition rate by constructing the network dynamically. This is accomplished by adding more hidden units to the network one at a time and retraining the expanded network until a network that correctly classifies all the input patterns has been constructed. The algorithm is summarized below.

Maximum likelihood neural network construction algorithm

1. Set the initial number of hidden units $h$ to 1.

2. Initialize $w^1 \in \mathbb{R}^n$ and $v^1 \in \mathbb{R}$ randomly.

3. Find a point that maximizes the log likelihood function (5). Denote this solution by $(\bar{w}, \bar{v})$.

4. If this solution results in a network that correctly classifies all the input patterns, then stop.

5. Add one unit to the hidden layer and pick $w^{h+1} \in \mathbb{R}^n$ randomly and set $v^{h+1} = 0$. Set $h = h + 1$ and use the point $(\bar{w}, w^{h+1}, \bar{v}, v^{h+1})$ as a starting point for maximizing the function (5) for the new expanded network.

6. Go to Step 3.

There are several advantages to this approach. Firstly, the need to determine the number of hidden units before training has been eliminated. Secondly, the network is started with a single hidden unit and additional units are appended
to the network only when they are needed. Training a small neural network is more desirable since it requires less computations than training a large network. This is in contrast to some methods that start with an oversize network and then prune the redundant units by removing some arcs that do not meet certain criterion for inclusion in the network. Finally, as will be shown below, when the network is trained by maximizing the log likelihood function (5) via a variant of the quasi-Newton method, the function value can only improve when a new hidden unit is added. In contrast, an addition of a new hidden unit in the dynamic node creation (DNC) algorithm of Ash (1989) may increase the value of the error function before this error function levels off after some backpropagation iterations. The algorithm we are proposing also eliminates some parameters that are needed by the DNC algorithm such as the trigger slope and the window width from the algorithm.

A variant of the quasi-Newton method, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method has been well-known for its robustness and efficiency for solving general unconstrained minimization problems (Scales, 1985; Shanno & Phua, 1978) and for neural network training (Watrous, 1987). We have chosen to use a modified version of this method to solve the maximization problem (5). Since it more common in nonlinear optimization to minimize a function as opposed to maximize it, let us reformulate this problem as

$$\min_{w,v} F^h(w,v) := -\sum_{i \in I} \log S^i - \sum_{i \notin I} \log(1 - S^i),$$

where the superscript $h$ on the function $F$ has been added to emphasize that the function we are considering corresponds to a network with $h$ hidden units.

An interesting property of the sigmoid function (2) is that its derivative can be expressed in term of the function itself as follows:

$$\sigma'(\xi) = \sigma(\xi)(1 - \sigma(\xi)).$$

The derivative of the hyperbolic function (3) can also be expressed in term of the function as follows

$$\psi'(\xi) = 1 - \psi(\xi)^2.$$  

Let us define the error

$$e^i = S^i - t^i, \ \forall i = 1, 2, \ldots, k,$$

then by applying the chain rule, the components of the gradient of the function $F^h(w,v)$ can be computed efficiently:

$$\frac{\partial F^h(w,v)}{\partial w^m_i} = -\sum_{i \in I} \left[ (S^i - t^i) \times v^m \times (1 - \psi(x^i w^m)^2) \times x^i_t \right]$$

$$+ \sum_{i \notin I} \left[ S^i \times v^m \times (1 - \psi(x^i w^m)^2) \times x^i_t \right]$$
\[
\frac{\partial F_h(w, v)}{\partial v^m} = -\sum_{i \in I} \left[ (1 - S^i) \times \psi(x^i w^m) \right] + \sum_{i \notin I} \left[ S^i \times \psi(x^i w^m) \right]
\]

for all \( m = 1, 2, \ldots, h \) and \( \ell = 1, 2, \ldots, n \).

Let \((\overline{w}, \overline{v}) \in \mathbb{R}^{(n+1) \times h}\) be a point such that \(\nabla F_h(\overline{w}, \overline{v}) = 0\) and suppose that the network with \(h\) hidden units corresponding to this set of weights fails to recognize all its input. Let \(w^{h+1} \in \mathbb{R}^n\) be a randomly generated vector, it is clear that \(F^{h+1}((\overline{w}, w^{h+1}), \overline{v}, 0) = F^h(\overline{w}, \overline{v})\). Our principal aim is to find and show that there exists some \(v \in \mathbb{R}\) such that

\[
F^{h+1}((\overline{w}, w^{h+1}), \overline{v}, v) < F^h(\overline{w}, \overline{v}).
\]

For simplicity of derivations, we hold \(w^{h+1}\) constant and define a new function of a single variable

\[
\mathcal{F}(v) = F^{h+1}((\overline{w}, w^{h+1}), \overline{v}, v)
\]

\[
= -\sum_{i \in I} \log \left[ \sigma \left( \Delta^i + \delta^i v \right) \right] - \sum_{i \notin I} \log \left[ 1 - \sigma \left( \Delta^i + \delta^i v \right) \right],
\]

where

\[
\Delta^i = \sum_{j=1}^k \psi(x^j w^i) v^j,
\]

\[
\delta^i = \psi(x^i w^{h+1}).
\]

It follows that the first and second derivatives of this function are

\[
\mathcal{F}'(v) = -\sum_{i \in I} \left[ 1 - \sigma \left( \Delta^i + \delta^i v \right) \right] \delta^i + \sum_{i \notin I} \left[ \sigma \left( \Delta^i + \delta^i v \right) \right] \delta^i
\]

\[
\mathcal{F}''(v) = \sum_{i=1}^k (\delta^i)^2 \left[ \sigma \left( \Delta^i + \delta^i v \right) \left( 1 - \sigma \left( \Delta^i + \delta^i v \right) \right) \right].
\]

Hence we have that the derivative of this function at zero is

\[
\mathcal{F}'(0) = -\sum_{i \in I} \left[ 1 - \sigma(\Delta^i) \right] \delta^i + \sum_{i \notin I} \sigma(\Delta^i) \delta^i
\]

\[
= \sum_{i=1}^k e^i \psi(x^i w^{h+1}),
\]

and that the second derivative is bounded above as follows

\[
|\mathcal{F}''(v)| \leq k/4, \ \forall v \in \mathbb{R}.
\]

We are now ready to state our main result below which shows the decrease in the value of the function \(F(w, v)\) when a new hidden unit is added to the network.
Theorem 1 Let $w^{h+1}$ be any n-dimensional vector, and let $\lambda = 4/k$ be a step-size, then

$$F^{h+1}(\overline{w}, w^{h+1}, \overline{v}, \lambda v) \leq F^h(\overline{w}, \overline{v}) - \frac{2}{k}\left[\sum_{i=1}^{k} e^i\psi(x^i w^{h+1})\right]^2,$$  \hspace{1cm} (11)

where $v = -\mathcal{F}(0)$.

Proof: By definition of the function $\mathcal{F}$, we have that

$$F^{h+1}(\overline{w}, w^{h+1}, \overline{v}, \lambda v) = \mathcal{F}(\lambda v).$$

From the second order Taylor expansion of this function, we have

$$\mathcal{F}(\lambda v) = \mathcal{F}(0) + \lambda \mathcal{F}'(0)v + \frac{1}{2} (\lambda v)^2 \mathcal{F}''(\rho \lambda v), \quad 0 < \rho < 1.$$

By letting $v = -\mathcal{F}'(0)$, we obtain

$$\mathcal{F}(\lambda v) = \mathcal{F}(0) - \lambda \left(\mathcal{F}'(0)\right)^2 + \frac{1}{2} \lambda^2 \left(\mathcal{F}'(0)\right)^2 \mathcal{F}''(\rho \lambda v)$$

$$\leq \mathcal{F}(0) - \lambda \left(\mathcal{F}'(0)\right)^2 + \frac{k}{8} \lambda^2 \left(\mathcal{F}'(0)\right)^2.$$

The inequality above is obtained from the fact the second derivative $\mathcal{F}''(v)$ is bounded by $k/4$. Now let us set $\lambda = 4/k$, we have

$$\mathcal{F}(\lambda v) \leq \mathcal{F}(0) - \frac{2}{k} \left(\mathcal{F}'(0)\right)^2$$

$$= \mathcal{F}(0) - \frac{2}{k} \left[\sum_{i=1}^{k} e^i\psi(x^i w^{h+1})\right]^2$$

$$= F^h(\overline{w}, \overline{v}) - \frac{2}{k} \left[\sum_{i=1}^{k} e^i\psi(x^i w^{h+1})\right]^2. \hspace{1cm} Q.E.D.$$

From the above theorem, it is clear that after a new hidden node is added into the network, the function value can be improved best by finding a vector $w^{h+1}$ such that the quantity

$$\left[\sum_{i=1}^{k} e^i\psi(x^i w^{h+1})\right]^2$$

is maximized. For a randomly generated $w^{h+1}$ it is very unlikely that this quantity will be zero. Thus, even before the new expanded network is retrained, if we pick $w^{h+1}$ randomly and set $w^{h+1} = -4 \left(\sum_{i=1}^{k} e^i\psi(x^i w^{h+1})\right)/k$, there is already a decrease in the function value. This is the reason why the log likelihood function has been used in conjunction with our network construction algorithm.
If the sum of the squared error function $f(w, v)$ (1) is used for training the network instead of the log likelihood function (6), then the function $\mathcal{F}(v)$ will become

$$\mathcal{F}(v) = \sum_{i=1}^{k} \left( \sigma(\Delta^i + \delta^i v) - t^i \right)^2. \quad (12)$$

The derivative of this function at zero is

$$\mathcal{F}'(0) = 2 \sum_{i=1}^{k} \left[ e^i \times \sigma(\Delta^i) \times (1 - \sigma(\Delta^i)) \times \delta^i \right]. \quad (13)$$

Due to rounding error, the product $e^i \sigma(\Delta^i)(1 - \sigma(\Delta^i))$ is often zero. This happens when each of the the network output $S^i$ is either very close to zero or very close to one. When the training of a network with $h$ hidden units converges to a point $(\overline{w}, \overline{v})$ such that $e^i$ is equal 0, or 1, or -1 for all $i = 1, 2, \ldots, k$, then the derivative (13) will be zero and the point $(\overline{w}, w^{h+1}, \overline{v}, 0)$ with any $w^{h+1} \in \mathbb{R}^n$ is in fact a local minimum of the function $f(w, v)$ for a new expanded network with $h + 1$ hidden units. Since there is no decrease in the function value, the addition of a new hidden unit will be futile and the recognition rate of the network will not improve. It has also been observed that neural network training with a fixed number of hidden units required less iterations if one substitutes the log likelihood function (5) for the sum of the squared error function (Lang & Witbrock, 1988; van Ooyen & Nienhuis, 1992).

3. BFGS method for neural network training

The standard backpropagation method for neural network training attempts to find a local minimum of the error function (1) by applying the gradient method. For ease of notation, let us denote $z = (w, v)$. Given an approximate set of weights $z^k$, the method updates these weights using the equation

$$z^{k+1} = z^k - \alpha^k \nabla_z f(z^k), \quad (14)$$

where $\alpha^k$ is a small positive real scalar called the learning rate. This learning rate is adjusted heuristically from iteration to iteration and typically held at some value close to zero.

Because of the slow convergence of the method, many researchers have looked for alternative algorithms for finding the minimum of the error function. A survey of unconstrained optimization methods that have been used for neural network training is given in (Battiti, 1992). One of the method mentioned in this survey is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method. In the past years, this method has been shown to be the most successful update among the many variants of a class of unconstrained optimization methods collectively known as the quasi-Newton methods. The method can be summarized as follows.
BFGS algorithm for function minimization

1. Choose any $z^1$ as a starting point. Let $H^1 = I$, set $k = 1$. Let $\epsilon > 0$ be a small terminating scalar and let $c_1$ and $c_2$ be two constants such that $0 < c_1 \leq c_2 < 1$ and $c_1 < 0.5$.

2. If $\|\nabla f(z^k)\| \leq \epsilon \max \{1, \|z^k\|\}$ then Stop. Otherwise go to Step 3.

3. Compute the direction $d^k = -H^k \nabla f(z^k)$.

4. Calculate a step length $\lambda^k$ such that the line search conditions

$$f(z^k + \lambda^k d^k) \leq f(z^k) + \lambda^k (d^k)^T \nabla f(z^k),$$

$$ (d^k)^T \nabla f(z^k + \lambda^k d^k) \geq c_2 (d^k)^T \nabla f(z^k),$$

are satisfied and let

$$z^{k+1} = z^k + \lambda^k d^k,$$

$$\gamma^k = \nabla f(z^{k+1}) - \nabla f(z^k),$$

$$\delta^k = z^{k+1} - z^k.$$

5. Update the matrix $H^k$:

$$ H^{k+1} = \left( I - \frac{\delta^k (\gamma^k)^T}{(\delta^k)^T \gamma^k} \right) H^k \left( I - \frac{\delta^k (\gamma^k)^T}{(\delta^k)^T \gamma^k} \right)^T + \frac{\delta^k (\delta^k)^T}{(\delta^k)^T \gamma^k}. $$

6. Set $k = k + 1$ and go to Step 2.

The convergence of this method has been well documented (Dennis Jr. & Schnabel, 1983). One notable difference between this method and the backpropagation method with fixed learning rate is that the line search condition (15) ensures that the function value is monotonically decreasing. This fact combined with the result of Theorem 1 guarantee that our network construction algorithm will eventually terminate with a final network having the value of the likelihood function (5) within a prescribed tolerance.

If the dimensionality of the minimization problem is $N$, note that for the steepest descent method with a fixed step length, only two $N$-vectors need to be stored, the current estimate of the minimum $z^k$ and the gradient of the function at this point, $\nabla f(z^k)$. When a line search procedure is implemented in conjunction with this method, two more $N$-dimensional vectors are needed by the procedure to store the new estimate of the minimum, $z^k - \lambda \nabla f(z^k)$ for some $\lambda > 0$ and the gradient at this new estimate.

In addition to these four $N$-vectors, the quasi-Newton method requires extra storage for holding the vector $H^k \gamma^k$ and the matrix $H^k$. Since this matrix is symmetric, an additional $N(N + 1)/2$ real words of storage will be sufficient.
Hence, the total storage requirement for the BFGS method is \( N(N+1)/2 \) plus several scalar variables for storing the various constants and scalar products. Although this \( O(N^2) \) storage requirement and the \( O(N^2) \) floating point operations needed at each iteration to update the matrix \( H^k \) may seem computationally unattractive, we believe that the speed of convergence of the method more than compensates this drawback.

The matrix \( H^k \) of the BFGS method is updated at each iteration by adding a rank-two correction matrix. More recently, a quasi-Newton method that allows a choice between a rank-one update and a rank-two update at each iteration of the algorithm has been proposed by Phua and Setiono (1992). This method has been shown to be faster than the standard BFGS method for a wide range of nonlinear optimization problems. It is similar to the BFGS method above except for the updating of matrix \( H^k \). For this method, Step 5 of the BFGS Algorithm above is replaced by the following:

If \( (\delta^k - H^k \gamma^k)^T \gamma^k > 0 \), then compute \( H^{k+1} \) as follows:

\[
H^{k+1} = H^k + \frac{(\delta^k - H^k \gamma^k) (\delta^k - H^k \gamma^k)^T}{(\delta^k - H^k \gamma^k)^T \gamma^k}.
\]

Otherwise, compute

\[
t^k = \frac{6 \left[ f(z^k) - f(z^{k+1}) + (\delta^k)^T \nabla f(z^{k+1}) \right]}{\gamma^k} - 2,
\]

and compute \( H^{k+1} \) as follows:

\[
H^{k+1} = \left( I - \frac{\delta^k (\gamma^k)^T}{\delta^k (\gamma^k)^T} \right) H^k \left( I - \frac{\delta^k (\gamma^k)^T}{\delta^k (\gamma^k)^T} \right)^T + \frac{1}{t^k} \frac{\delta^k (\delta^k)^T}{\delta^k (\gamma^k)^T}.
\]

It can be easily shown that if \( t^k > 0 \), then the matrix \( H^{k+1} \) will be positive definite. The positive definiteness of this matrix in turn ensures that the direction \( d^{k+1} \) will be a descent direction. Since for general nonlinear functions there is no guarantee that the value of \( t^k \) will be positive, in our implementation we restrict \( t^k \) to the interval \([0.1, 10] \), that is,

if \( t^k < 0.1 \) then set \( t^k = 0.1 \),

if \( t^k > 10 \) then set \( t^k = 10 \).

For the computational results that we report here, we have used this new method for solving the minimization problem (6).
4. Computational results and discussion

The neural network construction algorithm described in the Section 2 has been implemented in FORTRAN language. The program was run on a DECsystem 4000 machine running DEC OSF/1 V3.0 operating system. All random numbers required by the algorithm were generated using a pseudorandom number generator. The range of these random numbers was [-1,1]. Each network with $h$ hidden units was trained until a set of weights $(w^k, v^k)$ that satisfied the terminating conditions

$$\left\| \nabla F^h(w^k, v^k) \right\| \leq 10^{-8} \max \left\{ 1, \left\| w^k, v^k \right\| \right\},$$

or

$$\left| \left( F^h(w^k, v^k) - F^h(w^{k-1}, v^{k-1}) \right) / F^h(w^{k-1}, v^{k-1}) \right| \leq 10^{-6}$$

had been found.

If we solve the maximization problem

$$\max_w \left\| \sum_{i=1}^{k} e^i \psi(x^i w) \right\|,$$  \hspace{1cm} (20)

where $e^i, i = 1, 2, \ldots, k$ are the errors from the most recently constructed network and let $w^{h+1}$ be a solution of this problem, then the maximum decrease in the function value $F^h$ after a new hidden unit is added can be obtained by letting $\psi^{h+1} = -4\hat{F}^h(0)/k$. We found that this procedure did not improve the overall performance of the algorithm significantly. Instead of solving the problem (20), we chose the initial weights $w^{h+1}$ in Step 5 of the algorithm randomly in [-1,1] and set $v^{h+1}$ to zero. This was done four times. With each random set of initial weights, the new expanded network was trained and if none of these four networks completely recognized all the input patterns, the one that gave the best function value was used as the starting point for the new network with one more hidden unit.

Experiment 1: The parity problem

The parity problem is a well known difficult problem that has often been used for testing the performance of a neural network training algorithm. The input set consists of $2^n$ patterns in $n$-dimensional space and each pattern is an $n$-bit binary vector. The target value $t^i$ is equal to 1 if the number of one’s in the pattern is odd and it is 0 otherwise. We have experimented with values of $n$ ranging from 4 to 8. For each value of $n$, the program was run 50 times using 50 different random starting points. The results from these runs for the 4-bit parity problem are summarized in Table I below.

In this table, the column Freq. indicates the number of runs (out of 50) that stopped with the corresponding number of hidden units. For example, in
<table>
<thead>
<tr>
<th>Hidden Units</th>
<th>Freq.</th>
<th>Total Iterations</th>
<th>Total Func. Eval</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>490</td>
<td>632</td>
<td>0.20</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>417</td>
<td>496</td>
<td>0.21</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>495</td>
<td>599</td>
<td>0.28</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>660</td>
<td>777</td>
<td>0.42</td>
</tr>
</tbody>
</table>

Table 1: Results for the 4-bit parity problem

<table>
<thead>
<tr>
<th>Hidden Units</th>
<th>Freq.</th>
<th>Total Iterations</th>
<th>Total Func. Eval</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>484</td>
<td>702</td>
<td>0.33</td>
</tr>
<tr>
<td>4</td>
<td>41</td>
<td>531</td>
<td>663</td>
<td>0.37</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>563</td>
<td>707</td>
<td>0.46</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>755</td>
<td>997</td>
<td>0.78</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>903</td>
<td>1142</td>
<td>1.01</td>
</tr>
</tbody>
</table>

Table 2: Results for the 5-bit parity problem

one experiment the algorithm terminated with a network consisting of just 3 hidden units that is capable of recognizing all the inputs correctly. The columns Iteration and Func. Eval. denote the average number of iterations and function/gradient evaluations needed by the algorithm to construct the final network. The number of function/gradient evaluations is greater than the total number of iterations because for some search direction \( d_k \) more than one evaluations are needed to find a step length \( \lambda_k \) satisfying both conditions (15) and (16). The last column of the table lists the average execution time of the algorithm.

Eighty percent of the runs ended with a network with the expected four hidden units. The average number of function/gradient evaluations needed was 496. This figure compares favorably with the more than 4000 trials required by the DNC method of Ash even when the cost of updating the approximate Hessian matrix is taken into account.

The results obtained by the algorithm for the parity 5, 6, 7 and 8 problems are listed in Tables II-V.

From these tables we can make the following observations:

1. The neural network construction algorithm is very robust and efficient. For each problem, it has been able to construct a final network that achieves 100% correctness rate.
2. There seems to be no identifiable relationship between the final number of hidden units and the total execution time required to construct the network for these problems. Due to the highly nonlinear nature of the function (6), it is impossible to predict the number of iterations needed by the SR1/BFGS algorithm to find a local minimum point. For some starting point, the SR1/BFGS algorithm may take relatively small number of iterations to find a local minimum of the function $F^{h+1}(w, v)$, where the function value is not much better than the value at a local minimum of $F^{h}(w, v)$. In this case, it is likely that the neural network construction algorithm constructs a network that has higher than average number of hidden units. On the other hand, the SR1/BFGS algorithm may take many iterations to converge to a local minimum point where the function value is much better than the previous optimal value. In this case, the final network constructed will be likely to have smaller number of hidden units.

3. It has been shown by Sontag (1992) that the N-bit parity problem is solvable by a three-layer network with $(N/2) + 1$ hidden units if $N$ is even, and $(N + 1)/2$ hidden units if $N$ is odd. For $N = 4, 5, \ldots, 8$, the algorithm has been able to construct such networks.

<table>
<thead>
<tr>
<th>Hidden Units</th>
<th>Freq.</th>
<th>Total Iterations</th>
<th>Total Func. Eval</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>7</td>
<td>1216</td>
<td>1691</td>
<td>2.17</td>
</tr>
<tr>
<td>5</td>
<td>33</td>
<td>2009</td>
<td>2938</td>
<td>4.38</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>884</td>
<td>1147</td>
<td>1.80</td>
</tr>
</tbody>
</table>

Table 3: Results for the 6-bit parity problem

<table>
<thead>
<tr>
<th>Hidden Units</th>
<th>Freq.</th>
<th>Total Iterations</th>
<th>Total Func. Eval</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>1833</td>
<td>2654</td>
<td>5.93</td>
</tr>
<tr>
<td>5</td>
<td>29</td>
<td>2768</td>
<td>3872</td>
<td>9.93</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>3095</td>
<td>4418</td>
<td>12.94</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>2730</td>
<td>3839</td>
<td>12.16</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1435</td>
<td>1731</td>
<td>7.28</td>
</tr>
</tbody>
</table>

Table 4: Results for the 7-bit parity problem
<table>
<thead>
<tr>
<th>Hidden Units</th>
<th>Freq.</th>
<th>Total Iterations</th>
<th>Total Func. Eval</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>3253</td>
<td>4760</td>
<td>22.02</td>
</tr>
<tr>
<td>6</td>
<td>29</td>
<td>5341</td>
<td>7402</td>
<td>40.09</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>4743</td>
<td>6850</td>
<td>40.50</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>4210</td>
<td>6263</td>
<td>43.53</td>
</tr>
</tbody>
</table>

Table 5: Results for the 8-bit parity problem
Experiment 2: The mushroom Problem

In the previous experiment, the entire data set available have been used to construct the final network. Although the algorithm has been proven to be highly successful in constructing networks with minimal number of hidden units, it is not evident whether these final networks generalize well, that is whether they can predict well patterns that are not in the training set.

The patterns in the data set for this experiment correspond to species of mushroom in the Agaricus and Lepiota family. The data can be obtained from the University of California-Irvine repository via ftp from ics.uci.edu (Murphy & Aha, 1992). It was reported that an accuracy rate of 95 % had been obtained (Schlimmer, 1987; Iba et al., 1988). There are a total of 8124 patterns in the data set, one thousand of which are selected randomly to be used for training and the rest for testing. Each pattern has 22 attributes that describe the characteristics of a mushroom species. The problem is to classify the mushroom as either edible or poisonous. The number of poisonous samples in the training set is 503, while the number of edible samples is 497. The corresponding numbers in the testing set are 3413 and 3711, respectively.

The results of 50 runs of our construction algorithm are summarized in Table VI. As in Experiment 1, the construction algorithm was terminated only when 100 % accuracy on the training data had been achieved. In the last column of Table VI, we list the average accuracy of the networks constructed on the testing set. From the figures in this table, we can conclude that not only the algorithm consistently constructed networks with small number of hidden units, the networks it generated also have very good generalization performance.

<table>
<thead>
<tr>
<th>Hidden Units</th>
<th>Freq.</th>
<th>Total Iterations</th>
<th>Total Func. Eval</th>
<th>Time (seconds)</th>
<th>Accuracy (percent) on test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>746</td>
<td>1160</td>
<td>31.76</td>
<td>98.87</td>
</tr>
<tr>
<td>4</td>
<td>23</td>
<td>1057</td>
<td>1695</td>
<td>53.80</td>
<td>98.93</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>1325</td>
<td>2069</td>
<td>70.15</td>
<td>98.84</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1449</td>
<td>2264</td>
<td>86.17</td>
<td>98.91</td>
</tr>
</tbody>
</table>

Table 6: Results for the mushroom data set, training set = 1000, test set = 7124

Experiment 3: The University of Wisconsin Breast Cancer Diagnosis Problem

The Wisconsin Breast Cancer Data (WBCD) is a large data set that consists of 699 patterns of which 458 are benign samples and 241 are malignant samples. Each of these patterns consists of nine measurements taken from fine
needle aspirates from a patient's breast (Wolberg & Mangasarian, 1990). A linear programming based method for pattern separation called the Multisurface Method has been proposed by Mangasarian (1968) and a computer program that implements this method for the WBCD has been in use at the University of Wisconsin Hospital since 1990 (Mangasarian et al., 1990).

We have conducted two sets of experiments using this data set. In the first experiment, the first half of the data set (350 patterns, 191 benign and 159 malignant) were selected and used to train and construct the network. The remaining 349 patterns were then used as a testing set. In the second experiment, the first two-thirds of the data set (466 patterns, 279 benign and 187 malignant) were used for constructing the network and the resulting network was subsequently tested on the remaining 233 patterns. As in Experiments 1 and 2, we run our algorithm 50 times using 50 different random starting points. The results are summarized in Tables VII and VIII below.

<table>
<thead>
<tr>
<th>Hidden Units</th>
<th>Freq.</th>
<th>Total Iterations</th>
<th>Total Func. Eval</th>
<th>Time (seconds)</th>
<th>Accuracy (percent) on test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>15</td>
<td>2021</td>
<td>2453</td>
<td>10.97</td>
<td>96.29</td>
</tr>
<tr>
<td>4</td>
<td>18</td>
<td>2428</td>
<td>3005</td>
<td>14.60</td>
<td>96.32</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>2138</td>
<td>2837</td>
<td>14.68</td>
<td>96.52</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>2549</td>
<td>3344</td>
<td>20.06</td>
<td>95.92</td>
</tr>
</tbody>
</table>

Table 7: Results for WBCD with training set = 350, test set = 349

When the algorithm was run using half of the data, networks having 3 to 6 hidden units were constructed. There is no significant difference in the generalization capability of these networks. The average accuracy of the networks on the test set are listed in the last column of Table VII.

Table VIII shows that exactly 90% of the 50 runs ended with a final network having no more than 5 hidden units when two-thirds of the data were used for

<table>
<thead>
<tr>
<th>Hidden Units</th>
<th>Freq.</th>
<th>Total Iterations</th>
<th>Total Func. Eval</th>
<th>Time (seconds)</th>
<th>Accuracy (percent) on test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>2279</td>
<td>2790</td>
<td>16.67</td>
<td>97.00</td>
</tr>
<tr>
<td>4</td>
<td>19</td>
<td>2799</td>
<td>3321</td>
<td>22.35</td>
<td>97.09</td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td>2945</td>
<td>3685</td>
<td>26.47</td>
<td>96.88</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2767</td>
<td>3452</td>
<td>25.96</td>
<td>96.14</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>2417</td>
<td>2980</td>
<td>22.62</td>
<td>95.71</td>
</tr>
</tbody>
</table>

Table 8: Results for WBCD with training set = 466, test set = 233
training and constructing the network. Networks with hidden units ranging from 3 to 7 were constructed. There is no significant difference in the generalization capability of these networks either. The 96 % recognition rate achieved by the networks with 6 hidden units is about 1 % higher than the rate obtained by Bennett and Mangasarian (1992) for similar sized networks trained via the backpropagation method.
Experiment 4: The spiral problem

The problem of distinguishing two intertwined spirals (Lang & Witbrock, 1988) is a non-trivial one (see Fig 2). Figure 2(a) depicts the 194 patterns that form the two intertwined spirals. These patterns were generated by a program provided by Lang and Witbrock (1988). The dense spiral patterns shown in Figure 2(b) containing a total of 970 patterns were obtained after a slight modification to the pattern generating program. Solutions to these problems have been obtained by feedforward networks with several hidden layers having connection connecting every layer to all succeeding layers (Lang & Witbrock, 1988) or by networks where there are connections between hidden units such as those generated by the cascade correlation algorithm (Fahlman & Lebiere, 1989). Solutions from the standard single hidden layer networks have been reported only for a substantially reduced problem (Robbins et al., 1993).

![Sparse and dense spiral patterns](image)

Figure 2: Sparse and dense spiral patterns

Since we expected a rather large number of hidden units would be needed to solve the problem, we started the algorithm with 2 hidden units in the network and instead of one, two hidden units were added to the network at a time. In the previous three experiments, after a hidden unit was added, the new network was retrained four times using four different random weights for the arcs connected to the new hidden unit. For this experiment, due to the large amount of computations required in finding the optimal weights, the new network was retrained only once after every addition of two new hidden units.

We ran the algorithm ten times with the sparse spiral patterns as the input data using different initial random weights. The results are summarized in
Table IX. Networks having 32 to 42 hidden units were constructed. The median number of hidden units is 34. It is interesting to note that the total number of weights in a network with 34 hidden units is 136 (2 times 34 for the connections between the input and the hidden units, 34 for the connections between the hidden units and the output unit and an additional 34 bias values in the hidden units). This total number of weights is very close to the 138 that the five layer network of Lang and Witbrock has.

The total number of function/gradient evaluations is on the average more than those reported by Lang and Witbrock, however the actual computation time of our algorithm may be less since the algorithm always starts with a small number of weights in the network.

We tested the generalization capability of the networks constructed by using the dense spiral patterns as the test data. The results are given in the last column of Table IX. These results suggest that networks with small number of hidden units perform as well as those with more hidden units. In fact, the worst performance was shown by a network with 40 hidden units and the best performance was shown by a network with 32 hidden units.

<table>
<thead>
<tr>
<th>Hidden Units</th>
<th>Total Iterations</th>
<th>Total Func. Eval.</th>
<th>Accuracy (percent) on dense patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>34</td>
<td>34226</td>
<td>80.31</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>31051</td>
<td>83.30</td>
</tr>
<tr>
<td>3</td>
<td>34</td>
<td>17839</td>
<td>81.75</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>26615</td>
<td>82.47</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>35412</td>
<td>76.91</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
<td>24091</td>
<td>81.34</td>
</tr>
<tr>
<td>7</td>
<td>42</td>
<td>20091</td>
<td>82.37</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>34426</td>
<td>82.78</td>
</tr>
<tr>
<td>9</td>
<td>34</td>
<td>29510</td>
<td>80.31</td>
</tr>
<tr>
<td>10</td>
<td>38</td>
<td>42387</td>
<td>82.47</td>
</tr>
<tr>
<td>Ave.</td>
<td>35.8</td>
<td>29565</td>
<td>81.40</td>
</tr>
</tbody>
</table>

Table 9: Results of ten trials of the feedforward neural network construction algorithm on the sparse spiral patterns.

We ran the program again with the dense spiral patterns using the same ten sets of random initial weights used for the sparse patterns. The results are summarized in Table X. The median number of hidden units of the networks constructed for these patterns is 32 and the average number is actually about three less than that for the sparse patterns. This result is rather unexpected since many theoretical studies show that the number of hidden units necessary is directly proportional to the number of patterns. Our simulation results suggest that a problem can be solved by a smaller network when there are more data
used in the training, at least for the case of a well structured data set such as
the two spiral patterns.

Figure 3 shows the progression of learning by the algorithm during one of the
ten runs where a final network with 34 hidden units was constructed after 25613
iterations. Using the backpropagation method with three hidden layers and five
units in each hidden layer, it was reported by Lang and Witbrock that 64000
iterations were required to train the network. The output activation diagrams
during some stages of the network generation process are shown in Figure 4.
In each square, black represents a zero activation value, white represents one,
and grey represents intermediate values. From this figure we see that it takes
around 30 hidden units before the spiral patterns are learned by the network.
It takes a few more hidden units before each of the pattern in the training set
is classified to within 0.1 of its actual target value of either 0 or 1. Figure 4 (h)
indicates that most of the points in the square surrounding the training set is
classified as 0 or 1.

<table>
<thead>
<tr>
<th></th>
<th>Hidden Units</th>
<th>Total Iterations</th>
<th>Total Func. Eval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>38602</td>
<td>46004</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
<td>25613</td>
<td>29071</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>36932</td>
<td>42739</td>
</tr>
<tr>
<td>4</td>
<td>36</td>
<td>24171</td>
<td>28863</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>33011</td>
<td>37870</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
<td>26189</td>
<td>30288</td>
</tr>
<tr>
<td>7</td>
<td>36</td>
<td>34516</td>
<td>38748</td>
</tr>
<tr>
<td>8</td>
<td>38</td>
<td>46044</td>
<td>52172</td>
</tr>
<tr>
<td>9</td>
<td>32</td>
<td>37048</td>
<td>43762</td>
</tr>
<tr>
<td>10</td>
<td>28</td>
<td>26145</td>
<td>30501</td>
</tr>
</tbody>
</table>

| Ave | 32.6 | 32827 | 38002 |

Table 10: Results of ten trials of the feedforward neural network construction algorithm on the dense spiral patterns

23
5. Conclusion

We have proposed an algorithm for constructing a feedforward neural network. The algorithm starts with a network topology having a single hidden unit in the hidden layer and adds more hidden units one at a time as they are needed to improve the accuracy of the network. We have shown that when the network is trained to maximize the likelihood function, each addition of a hidden unit is guaranteed to increase the value of this likelihood function. Our implementation of this algorithm that incorporates the quasi-Newton method for training has demonstrated that the method is very effective. The algorithm was tested many times using different sets of data and it has always been able to successfully construct a final network that achieves 100% accuracy. For the parity problems, the algorithm was able to construct networks having less than \( n \) hidden units that correctly classified all \( 2^n \) input patterns for \( n = 4, 5, \ldots, 8 \). For the mushroom data set and the Wisconsin breast cancer data, our experiment results indicate that the networks constructed by the algorithm can predict new patterns with more than 95% rate of accuracy. We have also solved the spiral problem using this algorithm.
Figure 3: Progression of the neural network construction algorithm. The decreasing curve represents the value of the log likelihood error function. A jump in the bold staircase curve indicates the point when two hidden units are added to the neural network.
Figure 4: Evolution of a network with 34 hidden units. 
Plot of the output pattern for network with (a) 4 hidden units (b) 8 hidden units (c) 12 hidden units (d) 16 hidden units.
Figure 4: Evolution of a network with 34 hidden units (continued).
Plot of the output pattern for network with (e) 22 hidden units (f) 30 hidden units (g) 32 hidden units (h) 34 hidden units.


