**Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights**

Derrick Nguyen and Bernard Widrow
Information Systems Laboratory
Stanford University
Stanford, CA 94305

# Abstract

A two-layer neural network can be used to approximate any nonlinear function. The behavior of the hidden nodes that allows the network to do this is described. Networks with one input are analyzed first, and the analysis is then extended to networks with multiple inputs. The result of this analysis is used to formulate a method for initialization of the weights of neural networks to reduce training time. Training examples are given and the learning curve for these examples are shown to illustrate the decrease in necessary training time.

# Introduction

Two-layer feed forward neural networks have been proven capable of approximating any arbitrary functions [1], given that they have sufficient numbers of nodes in their hidden layers. We offer a description of how this works, along with a method of speeding up the training process by choosing the networks' initial weights. The relationship between the inputs and the output of a two-layer neural network may be described by Equation (1)

$$y = \sum_{i=0}^{H-1} v_i \cdot sigmoid(W_i^t X + w_{bi}) \tag{1}$$

where $y$ is the network's output, X is the input vector, $H$ is the number of hidden nodes, $W_i$ is the weight vector of the $ith$ node of the hidden layer, $w_{bi}$ is the bias weight of the ith hidden node, $v_i$ is the weight of the output layer which connects the $ith$ hidden unit to the output.

## The behavior of hidden nodes in two-layer networks with one input

To illustrate the behavior of the hidden nodes, a two-layer network with one input is trained to approximate a function of one variable $d(x)$. That is, the network is trained to produce $d(x)$ given $x$ as input using the back-propagation algorithm [2]. The output of the network is given as

$$y = \sum_{i=0}^{H-1} v_i \cdot sigmoid(w_i x + w_{bi}) \tag{2}$$

It is useful to define $y_i$ to be the $ith$ term of the sum above

$$y_i = v_i \cdot sigmoid(w_i x + w_{bi}) \tag{3}$$

which is simply the $ith$ hidden node's output multiplied by $v_i$. The sigmoid function used here is the hyperbolic tangent function

$$sigmoid(x) = [exp(x) - exp(-x)]/[exp(x) + exp(-x)] \tag{4}$$

which is approximately linear with slope 1 for $x$ between $-1$ and 1 but saturates to $-1$ or $+1$ as $x$ becomes large in magnitude. Each term of the sum in equation (2) is therefore simply a linear function of $x$ over a small interval. The size of each interval is determined by $w_i$, with larger $w_i$ yielding a smaller interval. The location of the interval is then determined by $w_{bi}$, i.e. the center of the interval is located at $x = -w_{bi}/w_i$. The slope of $y_i(x)$ in the interval is approximately $v_i w_i$. During training the network learns to implement the desired function $d(x)$ by building piece-wise linear approximations $y_i(x)$ to the function $d(x)$. The pieces are then summed to form the complete approximation.

To illustrate the idea, a network with 4 hidden units is trained to approximate the function $d(x)$ shown in Figure 1. The initial values of the weights $v_i$, $w_i$, and $w_{bi}$ are chosen randomly from a uniform distribution between $-0.5$ and 0.5. The values of $y_i(x)$ before and after training is shown in Figure 2, along with the final output $y(x)$.

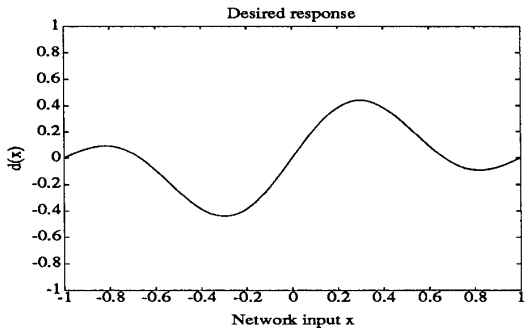


Figure 1: Desired response for first example

**Improving learning speed**

In the example above, we picked small random values as initial weights of the neural network. Most researchers do the same when training networks with the back propagation algorithm. However, as seen in the example, the weights need to move in such a manner that the region of interest is divided into small intervals. It is then reasonable to consider speeding up the training process by setting the initial weights of the hidden layer so that each hidden node is assigned its own interval at the start of training. The network is trained as before, each hidden node still having the freedom to adjust its interval size and location during training. However, most of these adjustments will probably be small since the majority of the weight movements were eliminated by our method of setting their initial values.

In the example above, $d(x)$ is to be approximated by the neural network over the region $(-1, 1)$, which has length 2. There are $H$ hidden units, therefore each hidden unit will be responsible for an interval of length $2/H$ on the average. Since $sigmoid(w_i x + w_{bi})$ is approximately linear over

$$-1 < w_i x + w_{bi} < 1, \tag{5}$$

this yields the interval

$$-1/w_i - w_{bi} < x < 1/w_i - w_{bi} \tag{6}$$

which has length $2/w_i$. Therefore

$$2/w_i = 2/H \tag{7}$$

$$w_i = H \tag{8}$$

However, it is preferable to have the intervals overlap slightly, and so we will use $w_i = 0.7H$. Next, $w_{bi}$ is picked so that the intervals are located randomly in the region $-1 < x < 1$. The center of an interval is located at

$$x = -w_{bi}/w_i = \text{uniform random value between } -1 \text{ and } 1 \tag{9}$$

and so we will set

$$w_{bi} = \text{uniform random value between } -|w_i| \text{ and } |w_i| \tag{10}$$

A network with weights initialized in this manner was trained to approximate the same $d(x)$ as in the previous section. Figure 3 shows $y_i(x)$ along with $y(x)$ before and after training. Figure 4 shows the mean square error as a function of training time for both the case of weights initialized as above and the case of weights initialized to random values picked uniformly between $-0.5$ and 0.5. All other training parameters are the same for both runs. Note how after training the domain $x$ is divided up into small intervals, with each hidden node forming a linear approximation to $d(x)$ over its own interval. As expected, we achieved a huge reduction in training time.

# Networks with multiple inputs

The output of a neural network with more than one input may be written as

$$y = \sum_{i=0}^{H-1} v_i \cdot sigmoid(W_i^t X + w_{bi}) \tag{11}$$

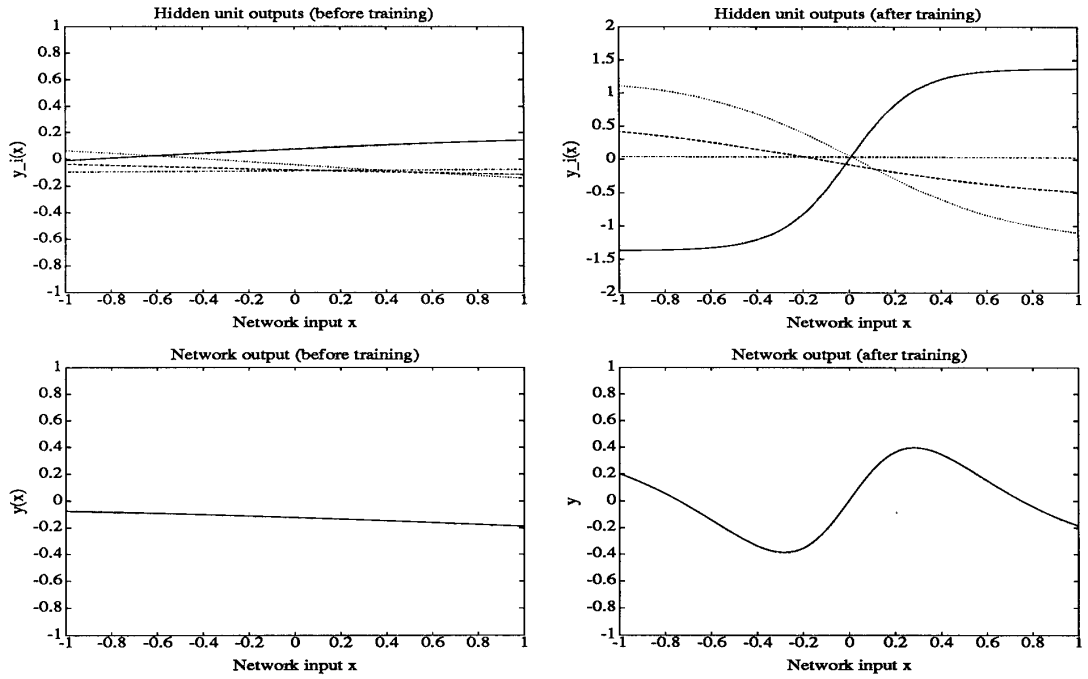


Figure 2: Outputs of network and hidden units before and after training with weights initialized to random values between -0.5 and 0.5

where $X$ and $W_i$ are now vectors of dimension $N$. We will again define $y_i(X)$ to be the $ith$ term of the sum in equation (11).

$$y_i(X) = v_i \cdot sigmoid(W_i^t X + w_{bi}) \tag{12}$$

The interpretation of $y_i(X)$ is a little more difficult. A typical $y_i(X)$ and its Fourier transform $Y_i(U)$ for the 2-input case is shown in Figure 5. Note that $Y_i(U)$ is a line impulse going through the origin of the transform space U. The orientation of the line impulse is dependent upon the direction of the vector $W_i$. This motivates us to interpret $Y_i$ as a part of an approximation of a slice through the origin of the Fourier transform $D(U)$ of $d(x)$.

Consider a slice of the Fourier transform $D(U)$ of $d(x)$. This slice, which we will call $D_i(U)$, goes through the origin of the transform space $U$. The time domain version of $D_i(U)$, $d_i(X)$, is a simple function of $W_i^t X$ where the $W_i$ is determined by the direction of the slice. A 2-dimensional $d(x)$, its Fourier transform D(U), a slice $D_i(U)$, and the inverse transform of the slice $d_i(X)$ is shown in Figure 6. Since $d_i(X)$ is a function of a single variable $W_i^t X$, it may be approximated by a neural network as shown in the previous section. The different approximations to the $d_i(X)$'s are then summed up to form the complete approximation to $d(X)$.

In summary, the direction of $W_i$ determines the direction of the $ith$ slice of $D(U)$, and the magnitude of $W_i$ determines the interval size in making piece-wise linear approximations to the inverse transform of the $ith$ slice of $D(U)$. The value of $w_{bi}$ determines the location of the interval. Finally, $v_i$ determines the slope of the linear approximation.

## Picking initial weights to speed training

Just as in the case of one input, it is reasonable to expect that picking weights so that the hidden units are scattered in the input space $X$ will substantially improve learning speed of networks with multiple inputs, and this section describes a method of doing so. It will be assumed that the elements of the input vector $X$ range from $-1$ to 1 in values. First, the elements of $W_i$ are assigned values from a uniform random distributation between -1 and 1 so that its direction is random. Next, we adjust the magnitude of the weight vectors $W_i$ so that each hidden node is linear over only a small interval. Let us assume

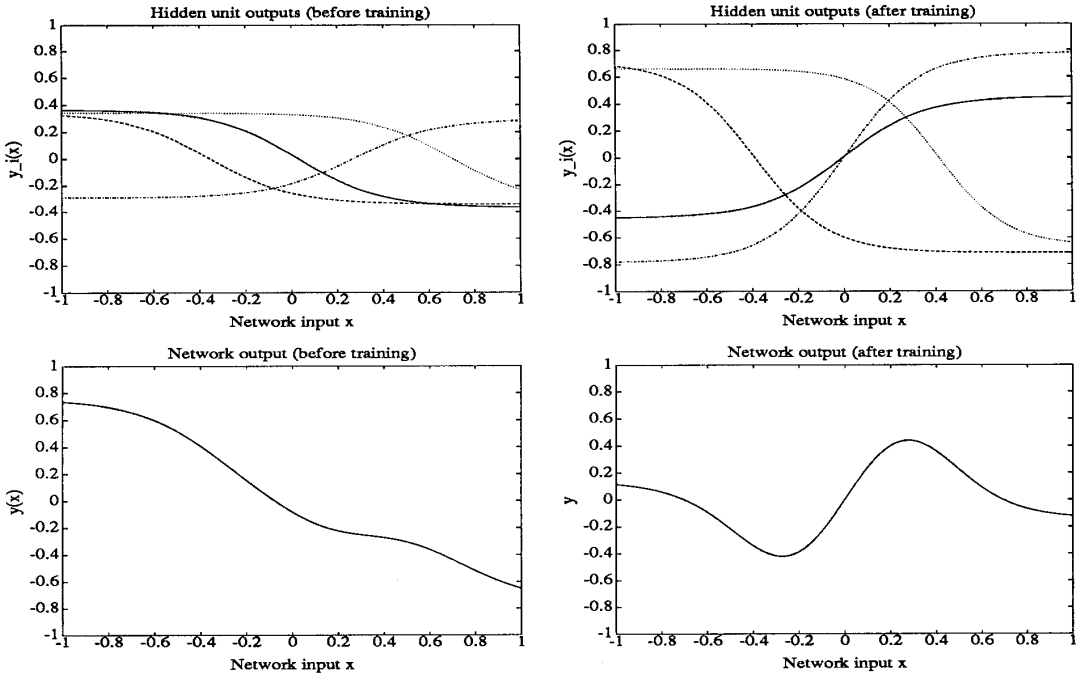


Figure 3: Outputs of network and hidden units before and after training with weight initialized by method described in text

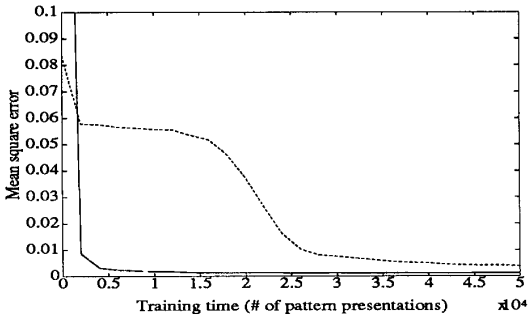


Figure 4: Learning curves from training of a network to approximate $d(x)$ described above. The solid curve is due to the training of a net initialized as described in the text. The dashed curve is due to a net whose weights are initialized to random values between $-0.5$ and 0.5

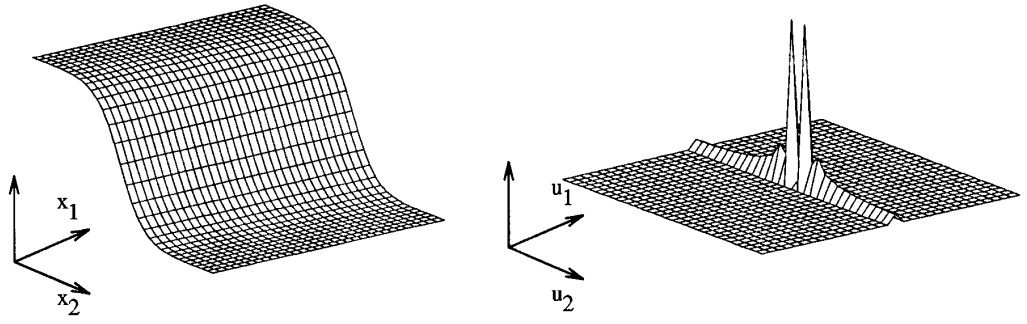


Figure 5: A $y_i(X)$ and its 2-D Fourier transform

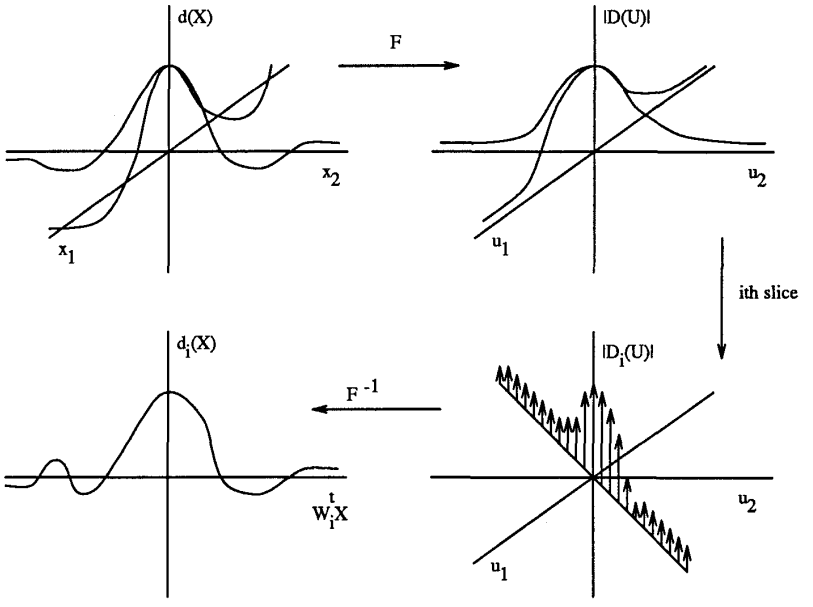


Figure 6: $d(x)$, its Fourier transform $D(U)$, a slice $D_i(U)$ of $D(U)$, and the inverse transform $d_i(X)$ of $D_i(U)$

that there are $H$ hidden nodes, and these $H$ hidden nodes will be used to form $S$ slices, and $I$ intervals per slice. Therefore,

$$H = S \cdot I \tag{13}$$

Since before training, we have no knowledge of how many slices the network will produce, we will set the weights of the network so that $S = I^{N-1}$. Each element of the input vector $X$ ranges from $-1$ to 1, which means the length of each the interval is approximately $2/I$. The magnitude of $W_i$ is then adjusted as follows

$$|W_i| = I \tag{14}$$

$$= H^{\frac{1}{N}} \tag{15}$$

In our experiments, we set the magnitude of $W_i$ to $0.7 \cdot H^{\frac{1}{N}}$ to provide some overlap between the intervals. Next, we locate the center of the interval at a random location along the slice by setting

$$w_{bi} = \text{uniform random number between } -|W_i| \text{ and } |W_i| \tag{16}$$

The weight initialization scheme above was used in training a neural network with two inputs to approximate the surface shown in Figure 7. The function describing this surface is

$$d(x_1, x_2) = 0.5 \sin(\pi x_1^2) \sin(2\pi x_2) \tag{17}$$

A network with 21 hidden units was used. Plots of the mean square error vs. training time are also shown in Figure 7 for the case of weights initialized as above and the case of weights initialized to random values between $-0.5$ and 0.5. With the weights initialized as above, the network achieved a lower mean square error in a much shorter time.

## Summary

This paper describes how a two-layer neural network can approximate any nonlinear function by forming a union of piece-wise linear segments. A method is given for picking initial weights for the network to

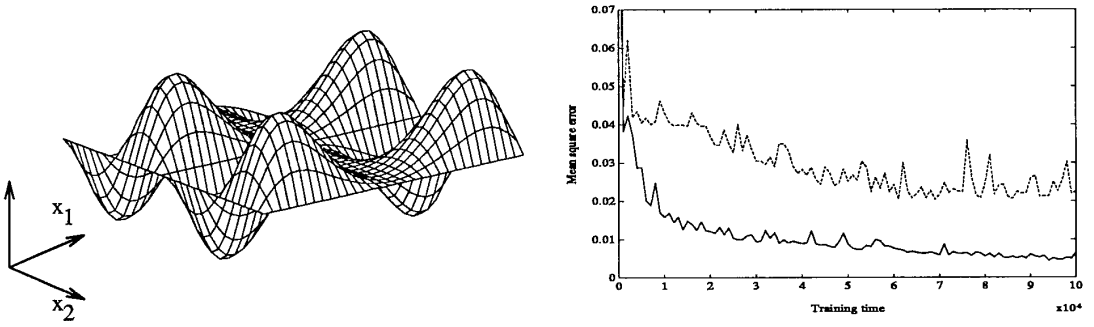


Figure 7: A 2-D desired function and the learning curves that resulted in training a neural net to approximate it. The solid curve is due to the training of a net initialized as described in the text. The dashed curve is due to a net whose weights are initialized to random values between $-0.5$ and 0.5

decrease training time. The authors have used the method to initialize adaptive weights over a large number of different training problems, and have achieved major improvements in learning speed in every case. The improvement is best when a large number of hidden units is used with a complicated desired response. We have used the method to train our "Truck-Backer-Upper" [3] and were able to decrease the training time from about 2 days to 4 hours.

The behavior of 2-layer neural networks, as described in this paper, suggests a different way of analyzing the networks. Each hidden node is responsible for approximating a small part of $d(X)$. We can think of this as sampling $d(X)$, and so the number of hidden nodes needed to make a good approximation is related to the bandwidth of $d(X)$. This gives us an approximate determination of the number of hidden nodes necessary to approximate a given $d(X)$. Since required the number of hidden nodes is related to the complexity of $d(X)$ and bandwidth is a good measure of complexity, our estimate of the number of hidden nodes is generally good. This work is in progress and full results will be reported soon.

# References

[1] B. Irie and S. Miyake. Capabilities of three-layered perceptrons. In *Proceedings of the IEEE International Conference on Neural Networks*, pages I–641, 1988.

[2] D. E. Rumelhart, G.E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 8. The MIT Press, Cambridge, Mass., 1986.

[3] D. Nguyen and B. Widrow. The truck backer-upper: An example of self-learning in neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages II–357–363. IEEE, June 1989.