A Relaxation/Regression Algorithm for Efficient Training of Multilayer Neural Networks

R. D. Koller
koller@uwyo.edu

B. M. Wilamowski
wilam@uwyo.edu

University of Wyoming
Department of Electrical Engineering
Laramie, WY 82070

Abstract

A new method for training multilayer neural networks has been developed. This method combines the speed of a least squares approach with the iterative nature of backpropagation. This method converges quickly, typically within 10 iterations, where back propagation can take tens of thousands of iterations.

Introduction

The use of backpropagation in the training of multilayer neural networks is common. Unfortunately, backpropagation is very slow. Relatively simple problems can take tens of thousands of iterations, occupying hours of computer time. Although many methods have been introduced to speed up backpropagation (1) (2), convergence can still take hours of comuter time.

In the training of single layer networks methods of least squares approximation are nice because of their speed (3). Unfortunately, least squares approximations require the knowledge of the desired input and output patterns. This makes the application of least squares difficult in multilayer neural network problems, where the hidden layer outputs are unknown. Solutions to this problem have been developed, but frequently they are very complex (4). Additionally, least squares has difficulty converging when the training set contains outlier data.

This paper will discuss a new method for multilayer neural network training. This method uses an iterative least squares method to solve multilayer problems. In addition to the training procedure, a solution to the outlier problem is included. The lengthy mathematical foundation for this algorithm is not included in this short four page summary, but will be discussed at the time of presentation.

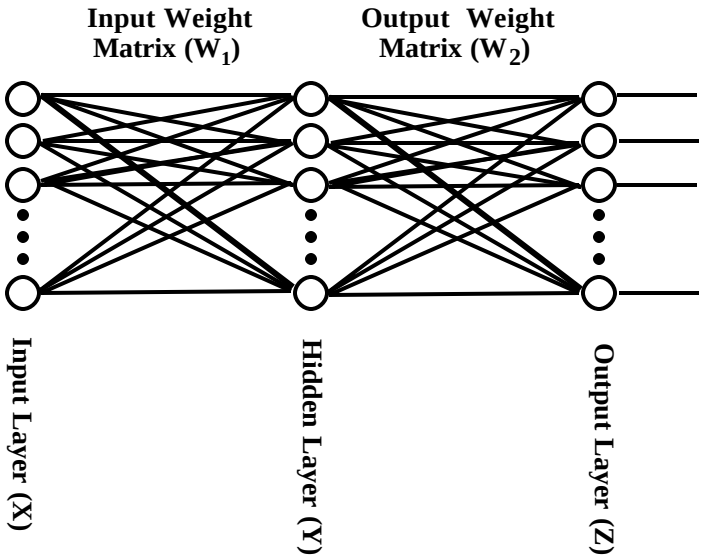

Figure 1. Multilayer neural network structure

Training Procedure

This training procedure works by applying least squares to each layer of the neural network structure. At first this seems impossible because the hidden layers are unknown; however, based on some drastic assumptions, this problem is bypassed. Given the number of input neurons, n_i, the number of hidden neurons, n_h, the number of output neurons, n_o, and the number of patterns, n_p, the typical multilayer neural network structure is shown in Figure 1. **X**, **Y**, and **Z** are matrices with sizes $[n_i \times n_p]$, $[n_h \times n_p]$, and $[no \times n_p]$ respectively. The weight matrices, $\mathbf{W}_1$ and $\mathbf{W}_2$, have sizes $[n_h \times n_i]$ and $[n_o \times n_h]$ respectively.

The outputs of the hidden layer, **Y**, are calculated using equation (1) and the network outputs, **Z**, are found using equation (2).

$$\mathbf{Y} = \tanh(\mathbf{W}_1 \mathbf{X}) \tag{1}$$

$$\mathbf{Z} = \tanh(\mathbf{W}_1 \mathbf{Y}) = \tanh(\mathbf{W}_2 \tanh(\mathbf{W}_1 \mathbf{X})) \tag{2}$$

The first step in the training procedure is to initialize the hidden layer outputs using randomly selected values in the range (-1, 1). At this point, the first assumption is made. It is assumed that these randomly selected values happened to be selected correctly. Based on this assumption and the known outputs, **Z**, the weight matrix $\mathbf{W}_2$ is calculated using (3). Using the recently calculated $\mathbf{W}_2$ and the known outputs, the outputs on the hidden layer are updated using (4).

$$\mathbf{W}_2 = \tanh^{-1}(\mathbf{Z})\ \mathbf{Y}^T(\mathbf{Y}\ \mathbf{Y}^T)^{-1} \tag{3}$$

$$\mathbf{Y} = (\mathbf{W}_2^T\ \mathbf{W}_2)^{-1}\ \mathbf{W}_2^T\ \tanh^{-1}(\mathbf{Z}) \tag{4}$$

Now, using this newly updated **Y** and the known input matrix, **X**, calculate $\mathbf{W}_1$ using (5). The final step in this iteration is to recalculate **Y** one more time, using the newly found $\mathbf{W}_1$ and the known **X** in equation (1). At this point, the iterations repeat with the updating of $\mathbf{W}_2$.

$$\mathbf{W}_1 = \tanh^{-1}(\mathbf{Y})\ \mathbf{X}^T(\mathbf{X}\ \mathbf{X}^T)^{-1} \tag{5}$$

It is important to note that the last step before exiting this training algorithm is to recalculate $\mathbf{W}_2$ for the final time. This assures a consistently trained path from the input layer to the output layer. Obviously this method assumes non-square matrices. In the case of square weight matrices, some of the redundant back calculations become trivial.

As mentioned above, a least squares approach still has problems with outlier data, leading to a non-optimal solution. This problem has not been covered yet in the training algorithm. In the case of non-square matrix equations, each equation row may be weighted by a scalar. This scalar should be based on the distance of the data point from the data cluster. An appropriate function is given in equation (6), where s_i is the scalar which will be multiplied to the i^{th} row of the matrix equation and o_i is network output from the last iteration.

$$s_i = \operatorname{sech}^2(\tanh^{-1}(o_i)) \tag{6}$$

By applying the scaling factor to a complete row, the equation is left unaltered; however, its weight in the least squares calculation is changed. By weighting outlier points less than cluster points, a more appropriate least squares solution is achieved. This problem will be demonstrated further in an example at the end of the the Results section.

Results

The first example for this multilayer training method is the two tower problem. In this problem, two clusters of data sit in a field of noise. Figure 2 shows the input data points and their classification superimposed on a contour plot of the results of training. Figure 3 clearly shows the two towers in a mesh plot. These results were accomplished after only 3 iterations, taking less than a minute. As a comparison, the same netword was trained using backpropagation. Several trials, with varying initial conditions, were attempted, but after 250,000 iterations none of them converged to solutions comparable to those demonstrated here. The trained network for this example had two inputs, 15 hidden neurons, and one output. The training data consisted of 65 data points: 15 class 1, 15 class 2, and 35 class 3.

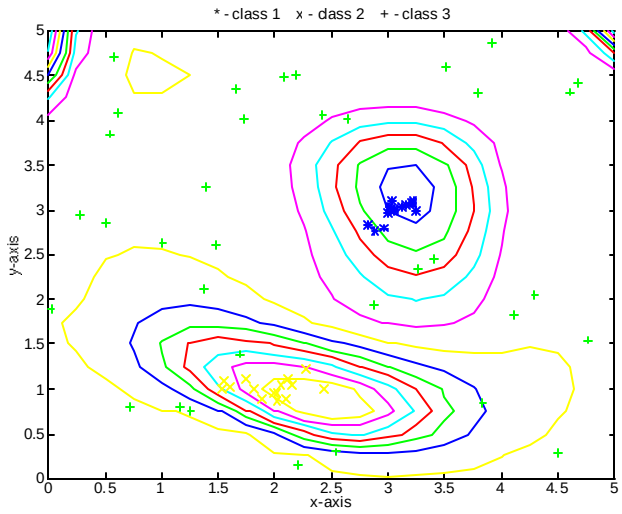

Figure 2. Contour Plot of the network output over the training space including class data

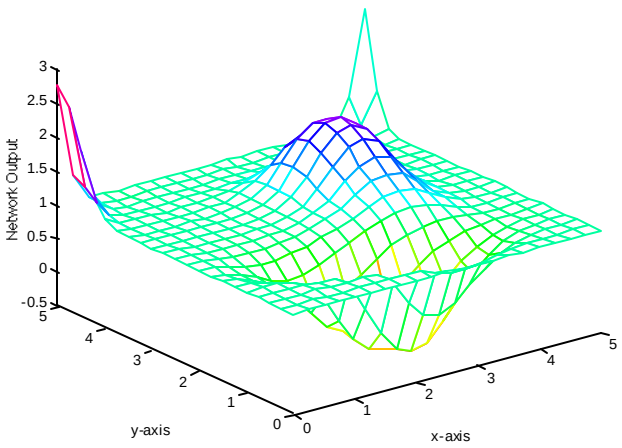

Figure 3. 3-D Mesh Plot of the network output over the training space

In the second example, this training algorithm was used for training temporal patterns. In this example, speech data was sampled at 22 data points. The first 5 derivatives were fed back through a neural network to generate the $6^{\underline{th}}$ derivative, so the network had five inputs and one output. Unlike the first example where the network clustered the data classifications, this example requires the network to perform a non-linear input-output mapping. Figure 4 shows the results of training using only 2 hidden neurons. In spite of using an extremely under powered network, the results are encouraging. In Figure 5, 15 hidden neurons were selected, and as expected, the network output matched the desired output much more closely.

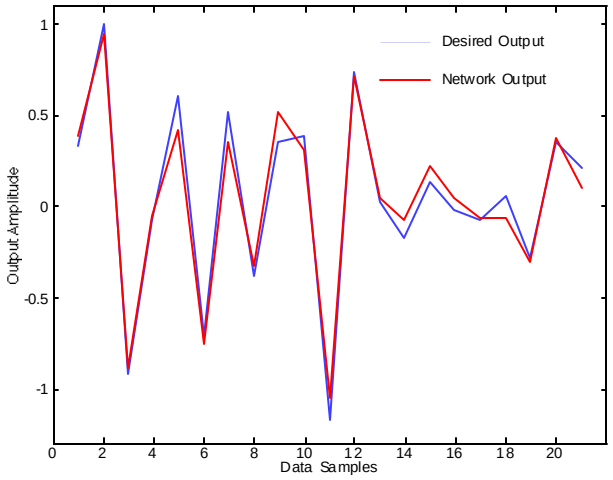

Figure 4. Temporal Example with 2 hidden neurons

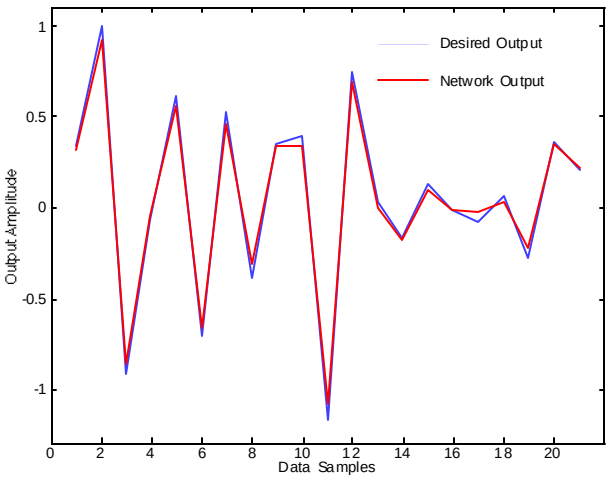

Figure 5. Temporal Example with 15 hidden neurons

The final example demonstrates the results of equation scaling. In Figure 6, the solid line shows how a straight forward application of least squares can cause incorrect classification. It is obvious in this simple example that the point at (8,1) is an outlier point, and removing it would cause a more correct classification line. Unfortunately, outliers are not always so obvious. Table 1 shows how this scaling procedure recognizes outliers and by applying a very small scale factor, essentially removes it from influence in the least squares procedure. The dashed line in Figure 6 indicates the new line calculated with the scaled data. This new line correctly separates all of the data points into their correct classification.

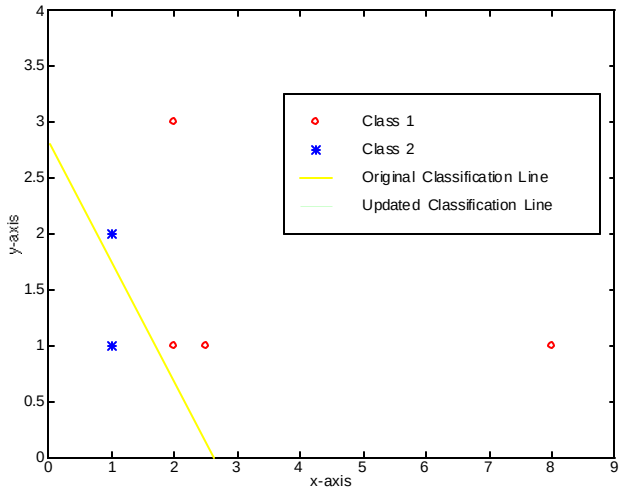

Figure 6. Classification line before and scaling

	Original			Scale Factor	Updated		
	x	y	Output		x	y	Output
1	1	1	0.9	0.9508	0.9508	0.9508	0.8557
2	1	2	0.9	0.9944	0.9944	1.9888	0.895
3	2	1	-0.9	0.9905	1.9809	0.9905	-0.8914
4	8	1	-0.9	0.0655	0.5243	0.0655	-0.059
5	2.5	1	-0.9	0.9355	2.3387	0.9355	-0.8419
6	2	3	-0.9	0.6357	1.2715	1.9072	-0.5722

Table 1. Training Data before and after scaling

Conclusions

This special method for application of least squares methods to multilayer neural networks was successful. This method has been tested on a variety of examples far too numerous to show here. The examples shown here demonstrate the flexibility of this training algorithm for two of the typical applications of neural networks: cluster classification and non-linear mapping.

Due to lack of space, a comparison between this method and backpropagation is not included; however, it was found that given enough time, backpropagation can converge to results comparable to the method presented. Unfortunately, this might take hundreds of thousands of iterations and hours of computer time. On the other hand, this least squares method converges to the results shown in fewer than 10 iterations, taking less than a minute.

One of the potential areas of improvement for this algorithm is a more educated initial guess. As discussed in the algorithm procedure, the hidden layer neuron outputs are randomly selected. This random selection might place separation lines in very awkward positions. By manually setting the initial conditions on the hidden neurons to evenly spaced quadrants in the training space, this algorithm might converge more quickly to better solutions.

References

1. Miniani, A. A., Williams, R. D. , Acceleration of Back-Propagation Through Learning Rate and Momentum Adaptation, *Proceedings of International Joint Conference on Neural Networks*, 676-679, 1990.

2. Sperduti, A., Starita, A., Speed Up Llearning and Networ kOptimization with Extended Back-Propagation, *Neural Networks*, 365-383, 1993.

3. Kohenen, T., Ruohonen, M., Representation of Associated Data by Matrix Operators, *IEEE Trans. on Computers*, 701-702, 1973.

4. Singhal, S., Wu, L. Training Multilayer Perceptrons with the Extended Kalman Filter, *Advances in Neural Information Processing Systems 1*, 29-37, 1989.