Project 4 Report: Backpropagation of Neural Network Simulation

Patrick Elam

CS420

April 1, 2015

Introduction:

One method of training an artificial neural network is through backward propagation of errors. This method can is done by first propagating forward through the network using training input to generate the output of the network, which is influenced by the weight of the linkages between neurons. Then the algorithm propagates backwards through the network to determine a delta value for all of the neurons. The algorithm then again propagates through the network forward, multiplying the delta values by the learning rate and the previous output of the given node to update the weights such that the next iteration will yield a value closer to that of which is correct (according to the training inputs). This method can be helpful for optimizing artificial neural networks.

Simulator and Calculations:

Alex Chaloux and I wrote a program in C++ that would simulate a neural network and would support its training through backpropagation. We allowed for the simulator to take a number of variables, such that we could adjust these variables to find the correlations between a change in the variable and the closeness to the correct output value of the network. The barometer on which we judged the output of the network was the root mean squared value of the difference between the expected and actual output of the network, labeled the RMSE of the validation phase.

For training inputs, we generated random values for X, Y and Z and plugged them into two equations, and saved the output values of the equations as expected outputs for the corresponding inputs. These two equations were the following:

Problem 1:[3]

$$f(x,y) = \frac{1+\sin(\pi x/2)\cos(\pi y/2)}{2}, \quad x \in (-2,2), y \in (-2,2).$$

Problem 2:

$$f(x,y,z) = \frac{3}{13}\left[\frac{x^2}{2} + \frac{y^2}{3} + \frac{z^2}{4}\right], \quad x \in (-2,2), y \in (-2,2), z \in (-2,2).$$

The list of training, testing, and validation numbers are given by the files training1.txt and training2.txt, testing1.txt and testing2.txt, and validation1.txt and validation2.txt, respectively.

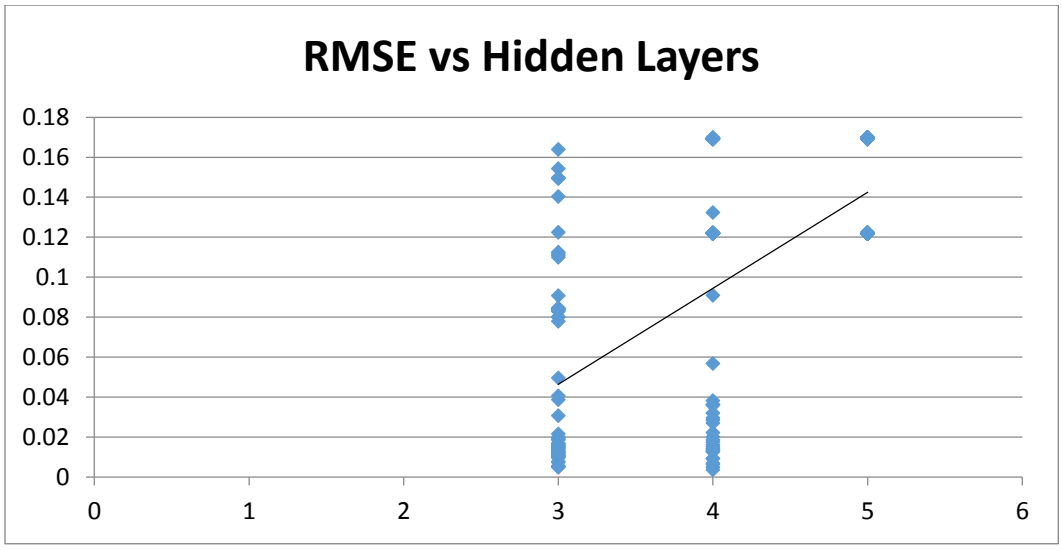
After generating training, testing, and validation numbers, we decided on the variations of the variables of which we would test our network and backpropagation algorithm with. We ran simulations with 3, 4, and 5 hidden layers, with each layer having between 2 and 20 neurons, with consistent patterns. We also used learning rates of 0.1, 0.25, 0.5, 1.0 and 2.0. We also varied the number of times, or epochs, we trained the network

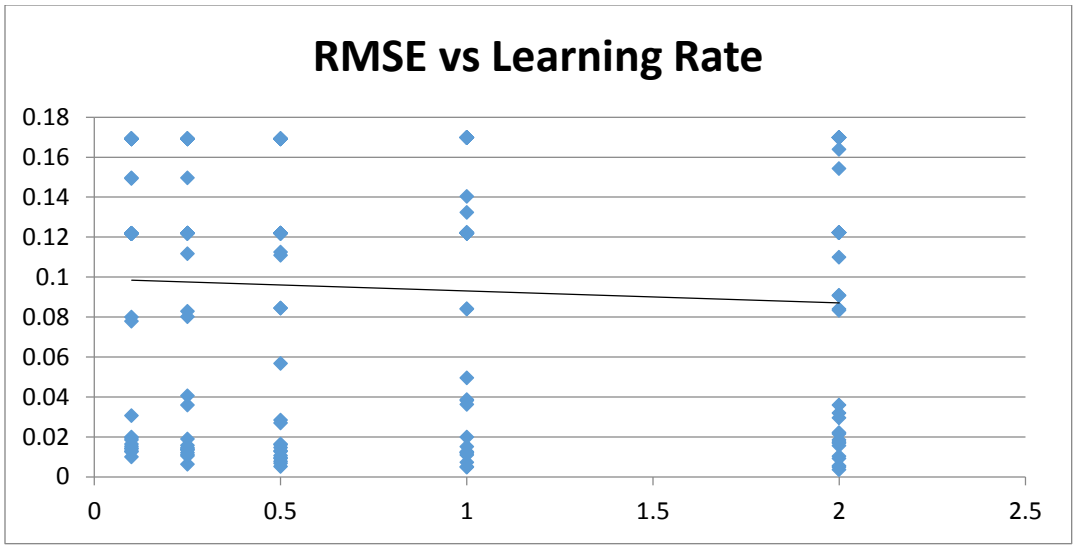
before validating its output; the options for number of epochs were 1000, 10000, 40000, 100000. For each combination of these variables, we produced a configuration file for each given problem. With 180 combinations of variables, we ended up with 360 separate configuration files with which to run through our simulator.

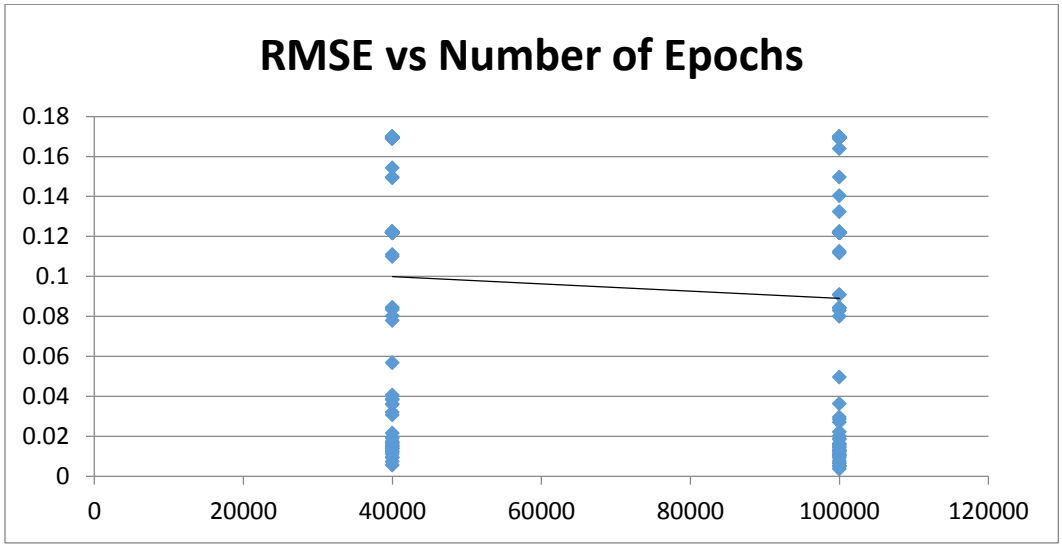
We wrote a script to run the simulator on each configuration file and had the program output a csv with data for each run. The weights of each node in the network were randomly initialized, and then the simulator propagated forward through the network and summed the product of the weight of the input linkage and the previous neuron's sigma (output) value. This value was then sent through the sigma function to calculate the output of the current neuron. After the entire network was traversed, it was traversed backwards to calculate delta values for each neuron. These delta values, which represented the amount the given weight needed to change, were calculated by summing the product of the sigma value of the current neuron, one minus the sigma value of the current neuron, the delta value of the previous neuron, and the weight of the linkages between these two neurons. After this process, the network was traversed forward again to update the weights of the linkages and the biases of the neurons. The weights new values were given as the summation of the product of the learning rate, the delta value of the neuron, and the sigma value of the previous neuron. The new bias value was the summation of the product of the learning rate and the delta value of the neuron.

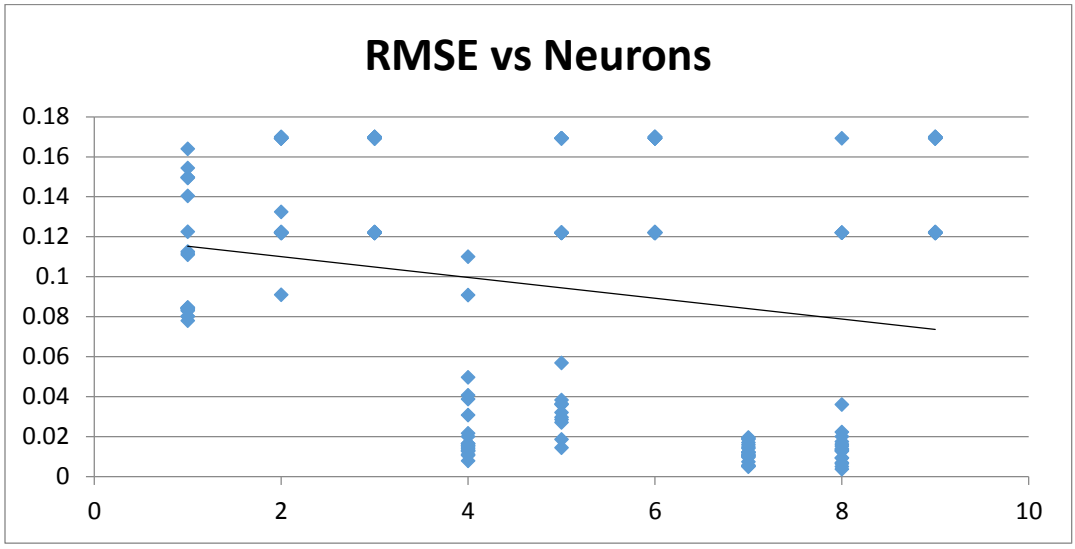
Results:

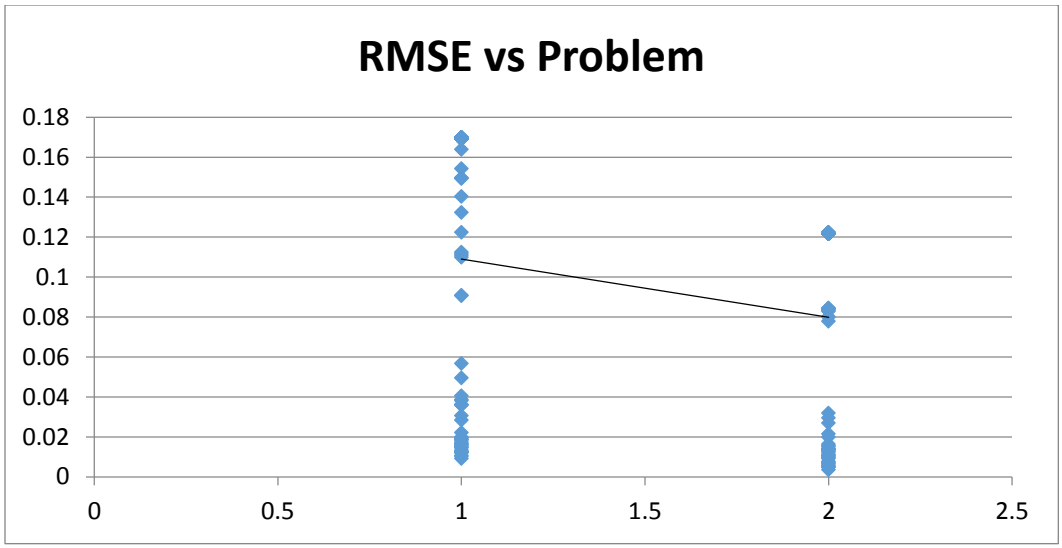
The output csv files (all of which are contained in the 'out' directory, generated by each variation of the variables) contained the output value after every epoch of training, and the final validation RMSE. We then collected all of these final validation RMSE values and compared them to the input variables:




RMSE vs Learning Rate
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
0.5
1
1.5
2
2.5


RMSE vs Number of Epochs
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
20000
40000
60000
80000
100000
120000


RMSE vs Neurons
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
2
4
6
8
10



Conclusions:

As it can be seen, the output has many outlying points, but the correlations are clear when averaging the points. It can be seen that as the number of hidden layers grew, the higher the final validation RMSE value was. The cause of this might be that if there are more layers, an error in one of the weights might have more of a chance to propagate down the line, causing inaccuracy. It can also be seen from the graphs that as the learning rate increased the validation RMSE decreased. The cause of this is that as the learning rate increased, the faster the training values changed the weights of the linkages and the biases of the neurons, meaning higher accuracy as iterations went on. From the graph of the number of epochs compared to the RMSE, we can see that as the number of epochs rose, the smaller the RMSE value, meaning that as more training was done, the output of the network got much more accurate. When graphing the RMSE values against the number of neurons in a layer, it can be seen that a higher number of neurons is better at reducing the final RMSE value. This is probably because the neurons and weights were given random weights and biases at the beginning. With more neurons per hidden layer, the greater the range of random values, and thus the less those values would individually effect the output. It is seen that the RMSE values for the second problem were lower than the RMSE values for the first problem. This is probably due to the fact that the second problem had three input values instead of two.

Finally, it can be seen that using backpropagation with an artificial neural network can be an accurate way of solving problems. This can be seen by the fact that, given the right conditions, the lowest RMSE value our network produces was 0.003. Given more time and computational power, an even better combination of input values could have been produced, resulting in even higher accuracy of problem solving.