John M. Blood
CS420 -Spring 2012
Project 4 Back Propagation
3/25/2012

Project 4: Back Propagation

Supporting Documentation:
backPropagation1.c: program for problem 1
backpropagation2.c: program for problem 2
3 test csv files: and, or, xor
22 problem 1 csv files
22 problem 2 csv files

Objective: To program an artificial neural network utilizing the back propagation algorithm, and to train, test and validate the training. Additionally, the objective was to explore the ideal parameter mixture to produce optimal network performance.

Hypothesis: Based on the lectures, I predict that excessively high numbers of neurons will lead to rote learning, low numbers of neurons will lead to an underpowered/under performing network. I also predict that bottlenecked networks will perform better and the learning rate will be important to gaining optimal performance. Too high of a learning rate and we might miss the global minimum error basin.

Method: I programmed this project in the C language. There are two executables because the function I used, fscanf requires the format of the data to be known. In problem one the format is (input input output) wheras in problem two the format is (input input input output), therefore I felt it better to separate the two. The primary challenge was keeping the sub scripts straight, but other challenges included incorrect/inconsistent instructions from the slides and website which led to a network that would not perform better than .34 root mean squared error. This was corrected and the network can obtain between .004 and .007 performance on the validation sets. I wrote and used a test script with different parameters to test the performance of the net. The first three were simple examples “and”, “or”, and “xor” to ensure the rest of the data was valid.

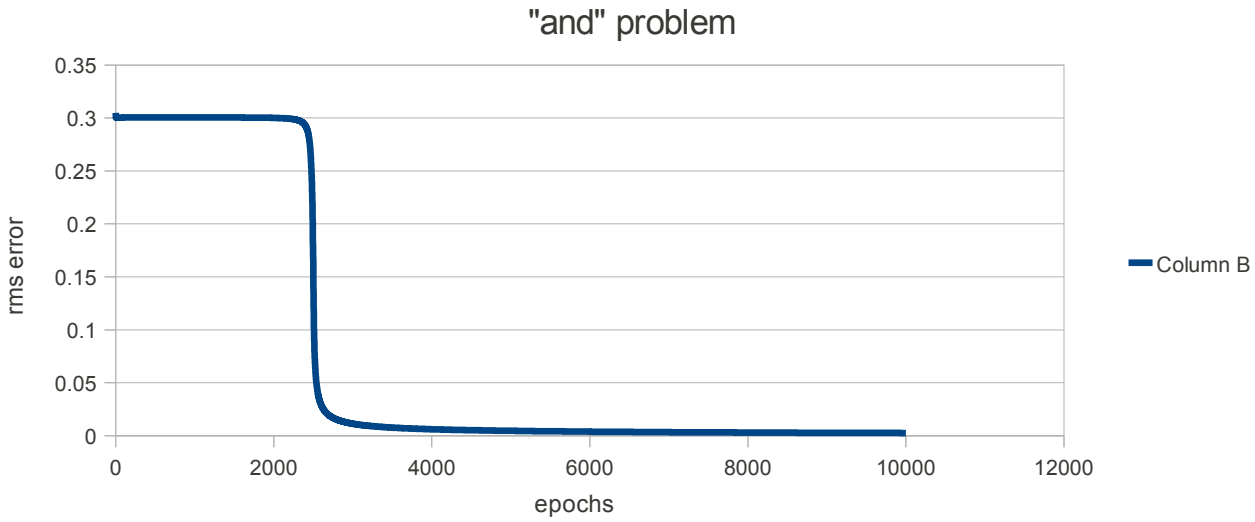


Figure 1: 'and' problem run for 10000 epochs with 3 hidden layers in config [4, 8, 4] and 2.5 rate

As can be seen, from this test, the weights on the network were adjusted appropriately to bring the validation error down to .002645 which is exceptional, but expected for such a simple problem.

Results: I attempted to adjust the learning rate to get the optimal result. The data from the trials are listed below. Note that the only parameter changed was the learning rate.

Learning Rate	Validation Error
0.25	0.050558
1	0.053274
2.5	0.060383
5	0.093525
25	0.316681

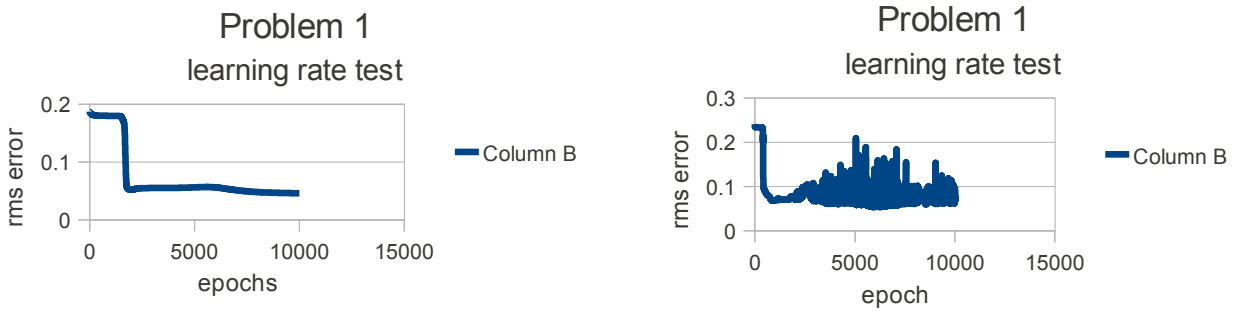


Figure 2: left: test with .25 rate right: test with 5 rate

The results here show that the slower learning rate takes a larger number of epochs to drop the error rate, however it results in an overall lower validation error score. That is, the network has a better ability to generalize to data it hasn't seen before. The high rate of learning shows very erratic behavior after around 2000 epochs, but the error drops very quickly from about .25 to .07 and is still performs at about 90% with generalized data. The next parameter experiment was with the number of epochs that the program was allowed to run.

Number of Epochs	Validation Error
1000	0.059769
2000	0.051654
4000	0.064870
10000	0.095016
50000	0.075579

This experiment was done with a higher learning rate, so longer epochs still resulted in the erratic behavior, It may be plausible to think that the number of epochs run could be optimized as a ratio of the learning rate, but this was not explored in depth in this project. From the straight data, it appears as though the minimal error occurs between 2000 and 4000 epochs for the parameters used, and then as a characteristic of the on-line learning method used, the error begins to increase again.

The next parameter that was tested was the number of layers. For the sake of consistency, the number of hidden nodes was fixed at 22, but their distribution was changed according to the following pattern:

(22), (11 11), (6 10 6), (6 5 5 6), (3 3 10 3 3), (3 3 5 5 3 3)

Hidden Layers	Validation Error
1	0.053686
2	0.051515
3	0.057754
4	0.203558
5	0.203558
6	0.203558

*error is from problem 1

The results of the layers experiment were surprising. I had to double check the testing data on the last three results to ensure that nothing was wrong with the test script. The weights were different, but they converged to the same value showing that it was in fact correct. These results show that with the same number of nodes, the distribution and number of layers is incredibly important. With an increased number of layers we showed a marked decrease in the ability to generalize the data within the same number of epochs. As a test to see if the error would drop, I re-ran the experiment with 8 layers (6 hidden) **out to 50,000 epochs** and it never converged lower than .19 rms error. This experiment seems to show that fewer layers with more nodes per layer converge quicker than a large number of layers.

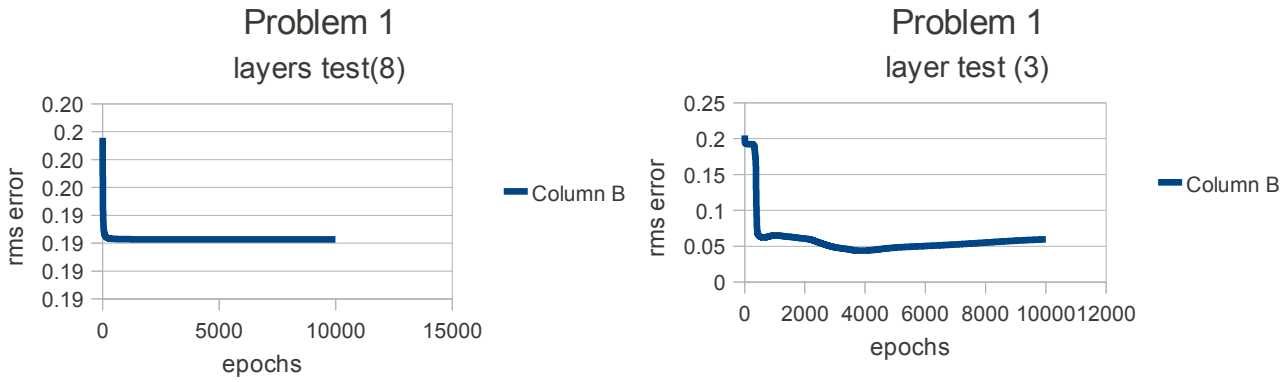
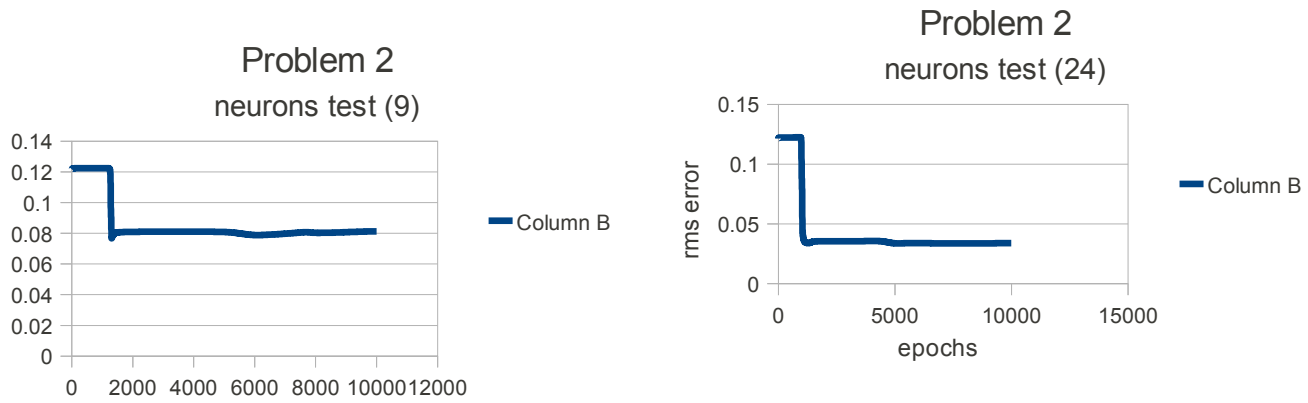


Figure 3: left: 6 hidden layers, did not converge in 10000 epochs. Right: 1 hidden, converged in 4000 epochs

The final parameter that was tested was the number of nodes with a fixed number of layers. This test was not only to check how well the network did with different numbers of neurons, but also to test different configurations such as bottle necks, reverse bottle necks, and straight even neurons across all layers of the network. For the purpose of this experiment, the term “bottle neck” refers to a higher number of neurons in the central hidden layers than the outer hidden layers, a “reverse bottle neck” is a structure with more neurons toward the outer layers than toward the inner layers, and “straight” refers to the same number of neurons across all hidden layers.

Number of Neurons	Configuration	Validation Error
9	straight	0.081254
16	Bottle neck	0.037043
18	straight	0.037773
24	Bottle neck	0.034316
15	Reverse bottle neck	0.035316
60	Bottle neck	0.036779

*All validation errors are from problem 2



The above graphs represent the worst (left) and the best (right) performing architectures that were tested in this section of the experiment. The straight architecture with nine neurons represents a grossly under-powered network for the problem. While the architecture on the right with twenty four neurons and a bottle neck is well suited to the problem in size and shape. Also notable is that the reverse bottle neck performed nearly as well as the best network but with less nodes. This may indicate that the reverse bottle neck is an ideal structure. The architecture with sixty neurons does not perform much better than the others. In fact it is near the bottom of the scale. This illustrates the tendency for rote learning to decrease the ability to generalize.

Conclusion: The results between the two problems was comparable except in actual numeric results. Problem 2 seemed to produce significantly lower validation error rates, In general, the network did not seem to do as well on the validation data as it did on the testing data, but the difference was typically negligible .002 or less difference. The parameter adjustment shows that a high learning rate leads to missing the minima and erratic oscillation. Running the program for different numbers of epochs produced results where lower numbers decreased the validation error, and if the program ran for too long, the error would increase (rote learning). This is a result of the online learning used. The number of layers experiment produced interesting results. When the hidden layers was four or more, the error increased significantly. This occurred in problems one and two. As to number of nodes and structure, it appears that the bottle neck structure and reverse bottle neck structure produced optimal results. Straight structures and structures with too few nodes performed poorly. One consideration is that the weights were all initialized randomly which could account for some of the differences .