Adam Yee

Comp 251

Project 2 Report: Neural Network Implementation and Analysis on car.data

11/30/2011

car.data provided by UCI machine learning repository: http://archive.ics.uci.edu/ml/

Table of contents:

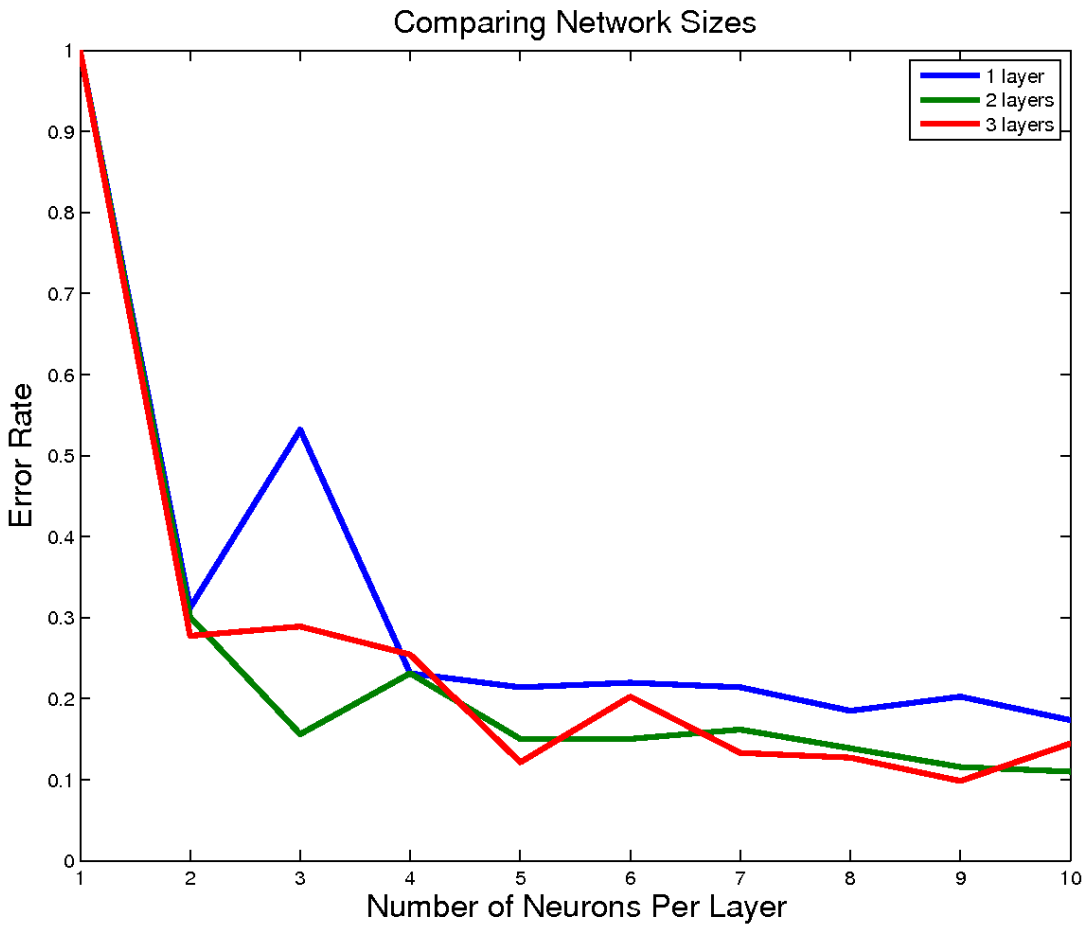
Analysis

When testing neural networks I used 90% of car.data as a grow set (training set) and the remaining 10% as a tuning set (test set). Each time, the 1728 examples of car.data are shuffled and they're order is randomized since car.data is sorted by default. This ensured that the network was trained with a variety of samples in hopes that it was exposed to many different dataset examples.

Finding The Best Network

In Figure 1, I used 60 epochs and a learning rate of 0.5 to train each network configuration by ranging the number of hidden layers from 1-3 and the number of neurons per hidden layer from 1-10. Figure 1 shows the results of a full test sweep across all configurations. By observing the data from Figure 1 and completing two more full sweep tests, I determined 3 hidden layers to give the best error rate.

Figure 1

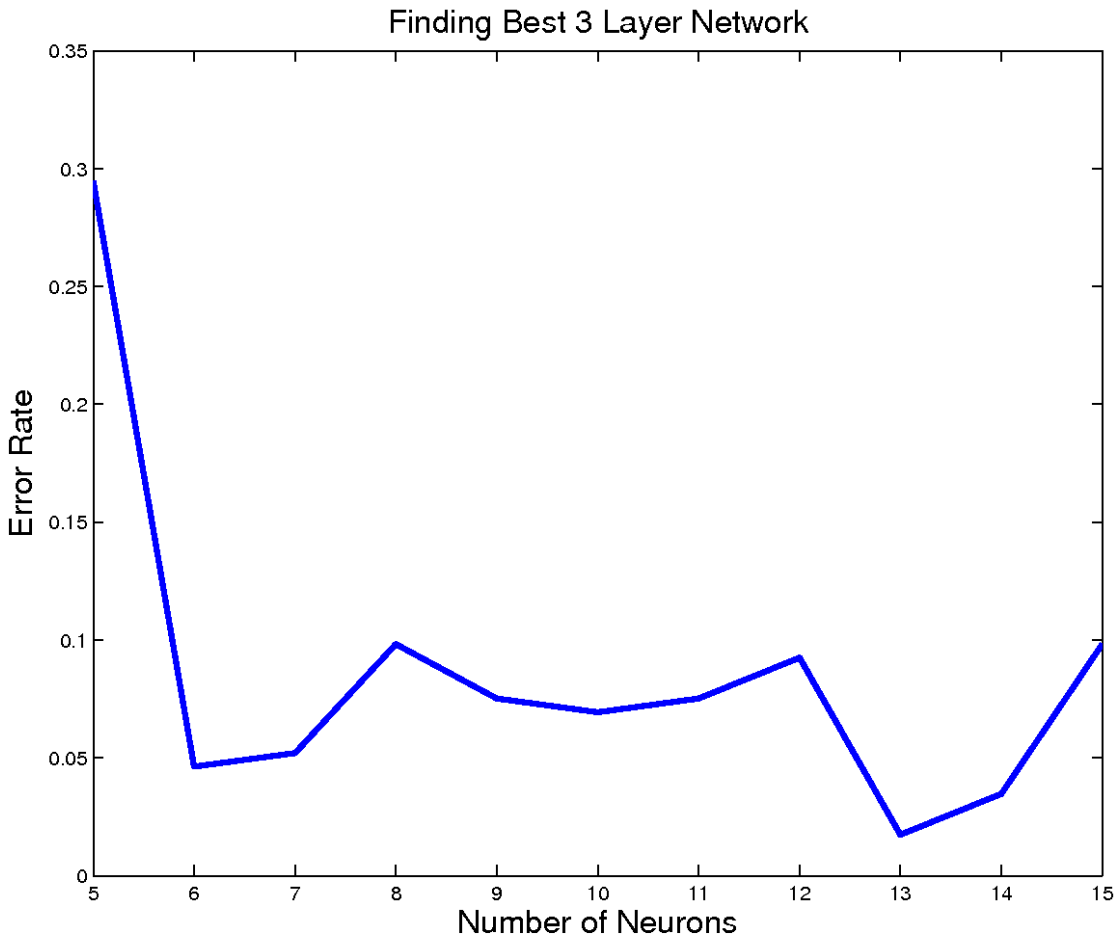


In order to find the best 3 layer network, I varied the number of neurons per hidden layer from 5-15 and found the best error rates to come from a 13-14 neuron per hidden layer network as seen in Figure 2 (using 80 epochs and a learning rate of 0.5 per network configuration). After

testing 13-14 neuron configurations more thoroughly, the errors rates produced from testing on a 10% tuning set averaged 5-10% and performed better than the other configurations on average.

In Figure 2, the graphed error rates are the lowest from each configuration, which means the error rates shown are not all from the 80th epoch. But a total of 80 epochs were performed for each configuration.

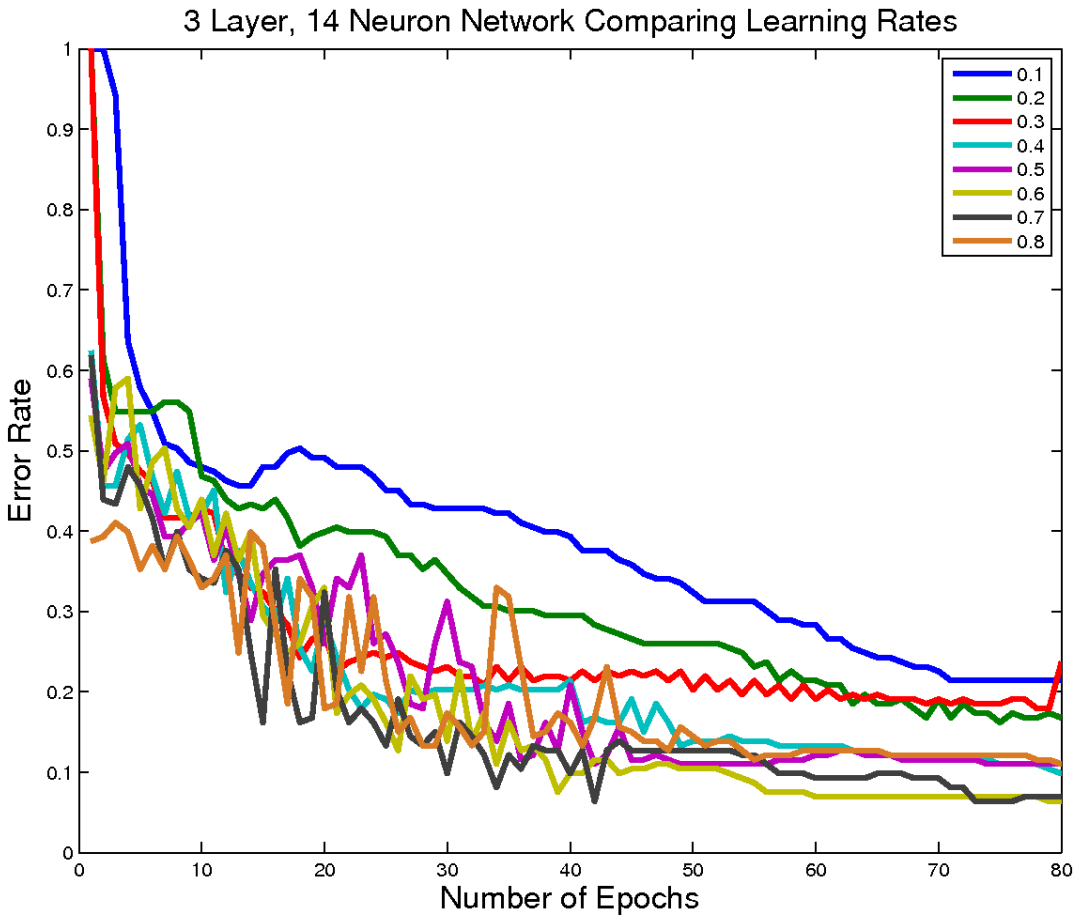
Figure 2



With a 3 hidden layer, 14 neuron per hidden layer network, I tuned the learning rate by gathering error rate results from a learning rate range from 0.1-0.9. The data from a 0.9 learning rate is not shown because it has very inconsistent error rate results. The error rate fluctuations from epoch to epoch showed that a learning rate of 0.9 produces very unstable networks. Furthermore, the 0.9 learning rate produced higher error rates (30%) as the number of epochs approached 80.

Figure 3 shows that small learning rates have more stable progression from epoch to epoch, but cannot reach the lower error rates of mid to high learning rates. For learning rates 0.4 and greater, the error rates stabilize and plateau past the 60 epoch mark.

Figure 3



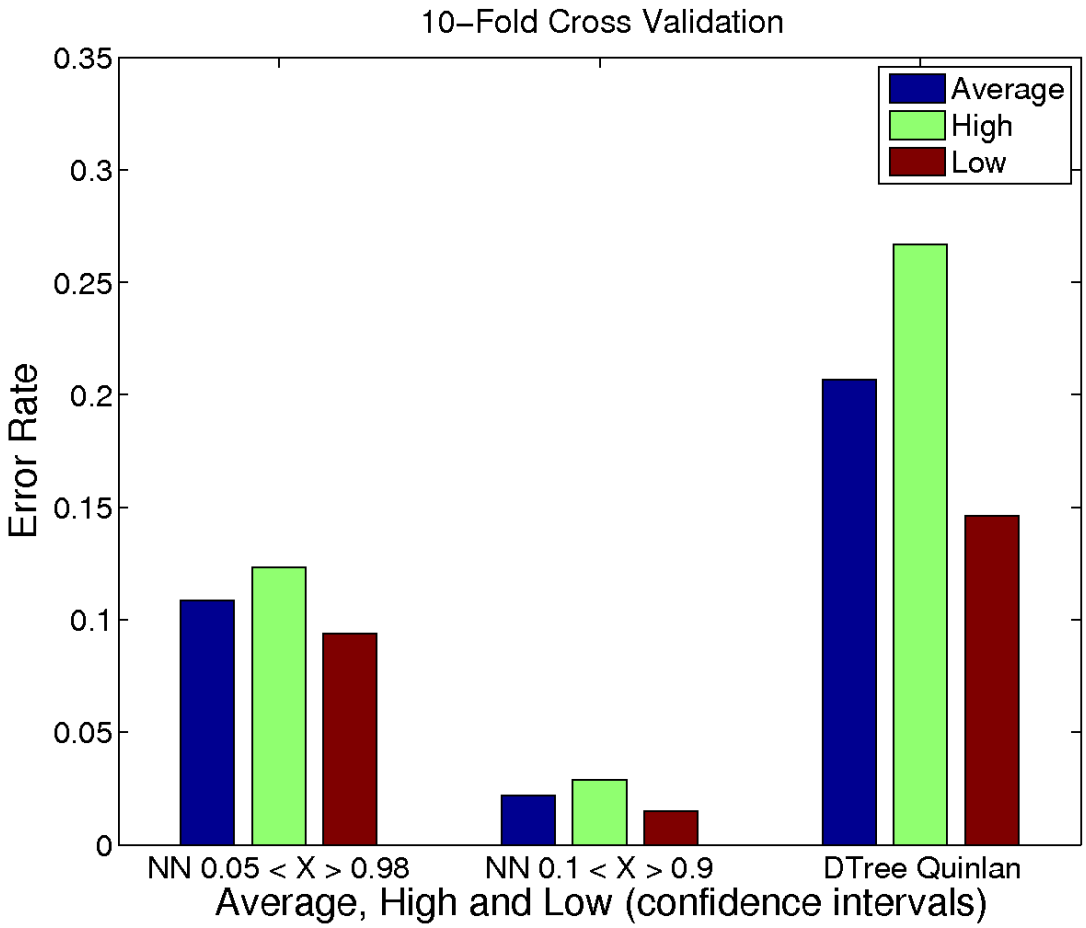
Learning rates 0.6 and 0.7 produces the best error rates (both a low of 6.35%) for the particular test shown in Figure 3. Learning rate 0.7 reached the 6.35% error rate before learning rate 0.6, which confirms higher learning rates learn faster.

Figure 3 also analyzes the optimal number of epochs to produce the best error rates. As mentioned above, the error rate stabilizes around the 60th epoch. *From the data shown in Figure 3, the optimal epoch ranges from 65 to 75 and the best learning rate from 0.6 to 0.7*.

Analyzing Average Error Rate Of car.data With 10-Fold Cross Validation

10-fold cross validation on the best neural network (3 layers, 13 neurons, 0.65 learning rate, 68 epochs) produced the error rates shown in Figure 4. The first two sets of data show the error rates of the same network, but with different classification limits (ranging from 0 to 1). The limits are used by the output layer when classifying each dataset example given to the network. The lower limit is used for the output layer neurons that are not supposed to fire (not match) and the upper limit is for one output layer neuron that is supposed to fire (match the proper category). The more strict limits (0.05 < output > 0.98) produce higher error rates of 11%. The less strict and more flexible limits (0.1 < output > 0.9) produce lower and better error rates of 2.2%. If any neuron miss-fires, the output is counted as a miss-match even if the neuron that was supposed to fire, did. The only time a correct category classification is counted is when the correct output neuron gives a true positive (greater than upper limit) and all other neurons give true negatives (smaller than lower limit). When compared to the Quinlan Decision Tree (3rd data set shown in Figure 4), the neural network performs better by 10% with strict limits and 20% with flexible limits.

Figure 4



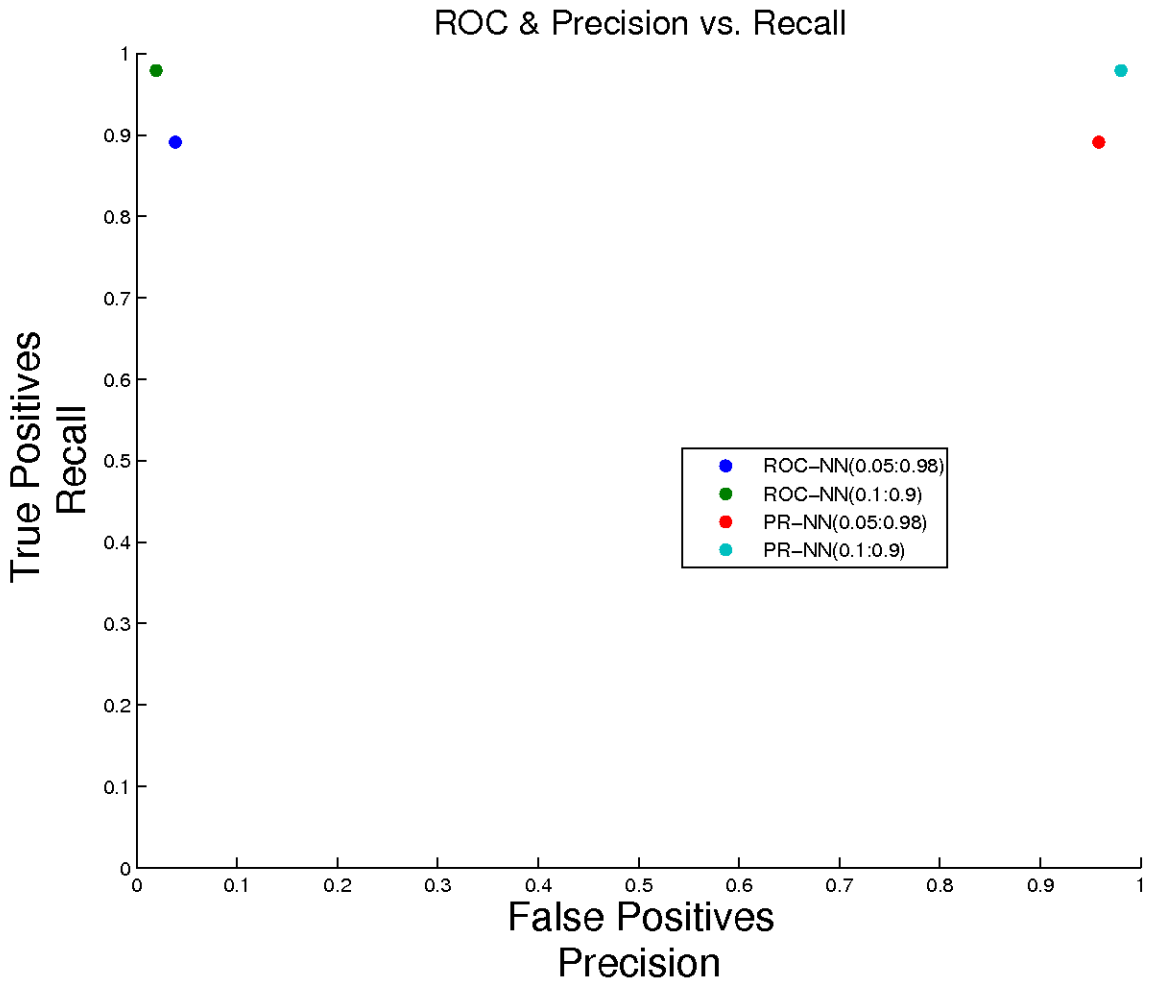
ROC and Precision vs. Recall

True and false outputs produced from 10-fold cross validation are outlined in Table 1. Figure 5 shows the ROC and Precision vs. Recall for the two separate limits used on the best network determined in the above section.

Table 1

Limits	True Positive	False Positive	True Negative	False Negative
0.05 < output > 0.98	1540	67	1661	188
0.1 < output > 0.9	1692	34	1694	36

Figure 5



Discussion

In order to more accurately identify the best network, I could have saved a randomized grow set and tuning set to file to be used on each network configuration. This way, each network can be trained with the same data and tuned with the same data to positively identify the best network configuration.

Analyzing a higher range of epochs (greater than 80) might show that the error rate can further decrease and also show at which epoch the error rate begins to worsen (if at all).

The average runtime for training the best neural network (3, 13, 0.65) on 68 epochs took 1:42 minutes. The time taken to build the decision tree is less than 5 seconds. The trade off for lower error rate comes at the price of longer training times due to higher amount of epochs. The best network analyzed in this project is small. Much larger networks can take longer. For my 3 hidden layer network, it is estimated that each neuron added to the hidden layers added about 8 seconds to the total training time (for a fixed amount of epochs) and each additional epoch increased the training time about 2 seconds.

Program Output and Usage

Please see README.txt submitted with project code for extra details about usage.

There are two files used in this project:
neuralnetwork.py
NNanalysis.py

Here are examples of program usage and output:

neuralnetwork.py

```
adam@adam-laptop-2:~/workspace/Comp251Project2_NN/src$ python2.7
neuralnetwork.py
Comp251Project2_NN - Adam Yee
Create the Neural Network by specifying the following...
Enable logging for testing? (y or n) y
(Layers) Range [0-3]: 3
(Neurons) Range [0-50]: 13
(Learning Rate) 0.0 > lr < 1.0: .65
(Upper Classification Limit) 0.0 > upper < 1.0: .9
(Lower Classification Limit) 0.0 > lower < 1.0: .1
Train the network with how many epochs?
(Epochs) Range [1-1000]: 100
Training the network, please wait...
Total training time: 162.541932821
The network has been trained to the 100th epoch and is now ready for use

1) print
2) dump to "weights.txt"
3) Run test set
9) exit
choice > 3
Error rate: 0.028901734104, TP[168], FP[2], TN[171], FN[5]
All test output written to NN-single-test.txt.
1) print
2) dump to "weights.txt"
3) Run test set
9) exit
choice > 9
```

Printing and dumping give the same outputs; printing to the screen (terminal) and dumping to the file “weights.txt”. For brevity, examples of this output is not included in this report.

NNanalysis.py

```
adam@adam-laptop-2:~/workspace/Comp251Project2_NN/src$ python2.7 NNanalysis.py
1) find optima epoch
2) perform k-fold cross validation
choice > 1
Network size fixed (3 layer, 13 neurons per layer).
Neuron learning rate set to 0.65
Number of epochs set to 80. The lowest epoch error rate is recorded.
analysis output appended to finding-optima-epoch.txt.
Calculating, please wait...
Analysis complete, please see finding-optima-epoch.txt for results.

adam@adam-laptop-2:~/workspace/Comp251Project2_NN/src$ python2.7 NNanalysis.py
1) find optima epoch
2) perform k-fold cross validation
choice > 2
Number of epochs > 15
Network size fixed (3 layer, 13 neurons per layer).
Neuron learning rate set to 0.65
analysis output appended to NN-kfoldcrossvalidation.txt.
Calculating, please wait...
Analysis complete, please see NN-kfoldcrossvalidation.txt for results.
```