CS 4341 Project 2 Report

Author: Kewen Gu(kgu) & Zhaochen Ding(zding2)

**Test your ANN algorithm with the number of hidden neurons set at 5. Change the holdout percentage and the samples used for testing.**

1. Holdout Percentage is 10%. 10% of the data is used for testing.

|  |  |  |
| --- | --- | --- |
| Number of Hidden Layers | Holdout Percentage | Error Rate |
| 5 | 0% - 10% | 25% |
| 5 | 20% - 30% | 20% |
| 5 | 40% - 50% | 30% |
| 5 | 60% - 70% | 25% |
| 5 | 90% - 100% | 25% |
| Average Error rate | - | 25% |

1. Holdout Percentage is 30%. 30% of the data is used for testing.

|  |  |  |
| --- | --- | --- |
| Number of Hidden Layers | Holdout Percentage | Error Rate |
| 5 | 0% - 30% | 15% |
| 5 | 20% - 50% | 25% |
| 5 | 40% - 70% | 31.7% |
| 5 | 50% - 80% | 28.3% |
| 5 | 70% - 100% | 28.3% |
| Average Error rate | - | 25.7% |

1. Holdout Percentage is 50%. 50% of the data is used for testing.

|  |  |  |
| --- | --- | --- |
| Number of Hidden Layers | Holdout Percentage | Error Rate |
| 5 | 0% - 50% | 20% |
| 5 | 10% - 60% | 40% |
| 5 | 20% - 70% | 30% |
| 5 | 30% - 80% | 30% |
| 5 | 50% - 100% | 29% |
| Average Error rate | - | 27.6% |

Conclusion:

We used 15 combinations of testing data and holdout percentages to test our algorithm. 5 different portion of 10%, 30% and 50% separately was selected as hold out.

During the testing we found two points, one is that the difference of same holdout percentage will not be that big, except for some extreme data, most error rates inside a same holdout percentage remains similar. Another point we realized is that once the holdout percentage increase, the average error rate will also increase, 25% at 10% holdout, 25.7% at 30%, and 27.6% at 50%. We consider that main reason is the lack of training data before heading into testing.

**Test your ANN algorithm with the number of hidden neurons ranging from 2 to 10. Run the algorithm on the first 80% of the training data and test on the remaining 20%. Plot the error rate on the test data and discuss why the plot looks the way it does.**

The table below shows the error rate VS the number of hidden layers.

|  |  |  |
| --- | --- | --- |
| Number of Hidden Layers | Holdout Percentage | Error Rate |
| 2 | 20% | 32.5% |
| 3 | 20% | 32.5% |
| 4 | 20% | 32.5% |
| 5 | 20% | 32.5% |
| 6 | 20% | 30.0% |
| 7 | 20% | 30.0% |
| 8 | 20% | 30.0% |
| 9 | 20% | 27.5% |
| 10 | 20% | 30.0% |

Conclusion:

From the plot above, we can see that in first several hidden layer number changes, the error rate didn’t have big change, but when the hidden layer reached 5, the error rate started to decrease, and went stable again. The second drop took place when hidden layer changed from 8 to 9. But when we increase the number to 10, the error rate actually increased, probably because of overfitting. The weights are overly modified to meet the training set, therefore the result from the validation set is biased.

**Discuss any simplifying assumptions that you made while implementing the neural network.  How resistant would your implementation be to changes in the number of input nodes? hidden layers? output nodes? network topology?**

By default, our neural network has two input layers, one output layer, five hidden layers, and 20% of holdout testing data. Our training set and testing set are picked randomly among the how data set for every run. Therefore, we may get the different results for the same configuration. In our implementation, we use two “weight” matrices to store the weights of both the input and output layers. And we defined three “activation” arrays to store the activation values of each layer. As a result, our implementation is very flexible to the change on number of input nodes, number of hidden layers, number of output nodes, and the network topology.

**Explain what the back-propagation algorithm does, in your own words.**

In back-propagation algorithm, we first propagate the input forward to compute the outputs. Then we calculate the delta values for each layers, and propagate deltas backward from output layer to input layer. Finally, we update every weight in the network using deltas.

**What is the purpose of the sigmoid function?**

The sigmoid function is applied to the result of the neural network element computation. The sigmoid function also satisfies a property between the derivative and itself so the computation will be easier to perform.