ROB537: HW1

Ammar Kothari

October 2, 2017

1 Question 1

1.1 Part A

The number of hidden units primarily affects the speed that the model converges at. Looking at the figure below, 20 hidden units appears to be the best number. However, by the third epoch, 5 hidden units has similar performance and by the ninth epoch, 100 hidden units has converged. This is likely because the function to classify the outputs is relatively simple. For a more complex problem, the number of hidden units would also affect the accuracy of the classifier. However, having too many hidden units can allow the model to overfit. This can be more clearly seen in Part B.

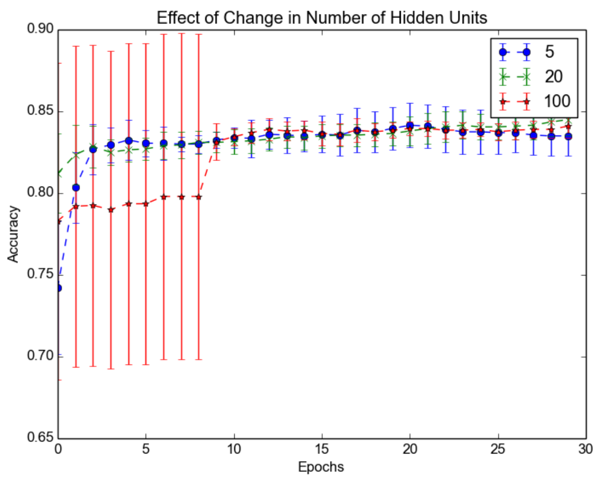

Figure 1: Effect of changing number of hidden units

1.2 Part B

The biggest effect that training time has is that as training time increases, performance begins to degrade. The model is overfitting to the training set which is making it less generalizable. As a result, performance degrades on the test set. In an extreme case, too little training time will cause the model not to achieve maximum performance, but for this data set, that is less than 5 epochs. The results of short training time can be seen in Part A.

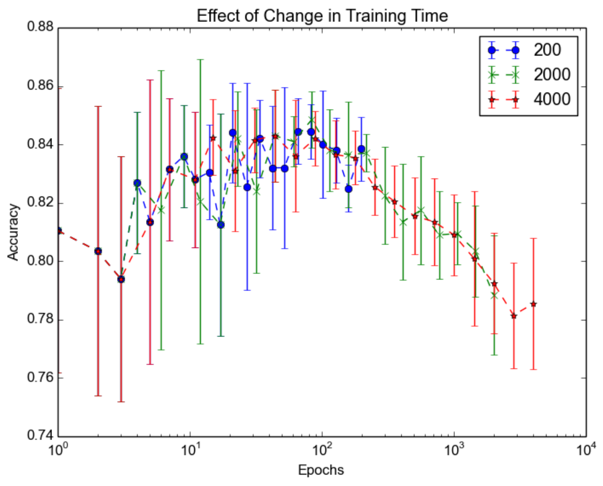

Figure 2: Effect of changing traning time

1.3 Part C

The size of the learning rate affects how quickly the classifier converges. With very small steps, the classifier takes a long time to converge. With very large steps, the classifier diverges and always performs poorly. An appropriate step size allows for convergence to happen in a reasonable amount of time.

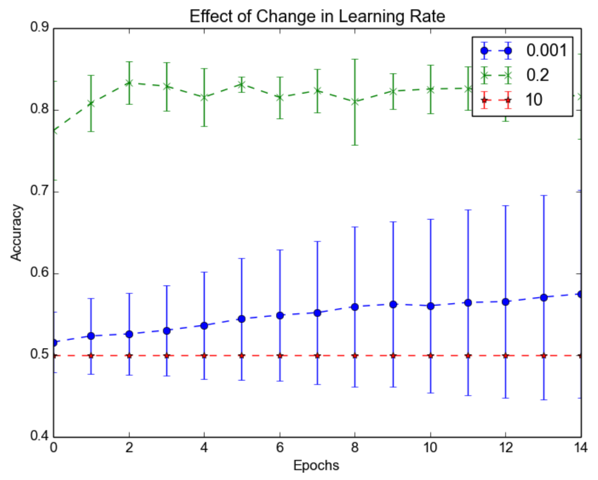

Figure 3: Effect of changing learning rate

1.4 Part D

I found an interesting bug (feature?) in Python when using multiprocessing. The cause (I think) is that because the processes spawn so quickly, they all use the same machine time to generate their random weight initializations. This results in all of the newtorks having the same results. This is even more surprising since the data is shuffled for every epoch. I believe that again the shuffling was happening the same for all of the trials. I was eventually able to cause the weights to be different by seeding the generator with different values for each trial. An intersting problem that only occurs when using multiprocessing.

Some other parameters that the can change convergence time and network performance are the initialization weights of the network and shuffling the data during processing. I examined how initialization of the weights affects the algorithm. The legend shows the magnitude of the initial values. For very small magnitudes of initial weights, the network does not change much over the course of training. It always predicts the same value which results in 50% accuracy because of the distribution of the testing set. For very large values, numerical saturation occurs and the network begins to output NaNs which are always classified as incorrect. This results in 0% success.

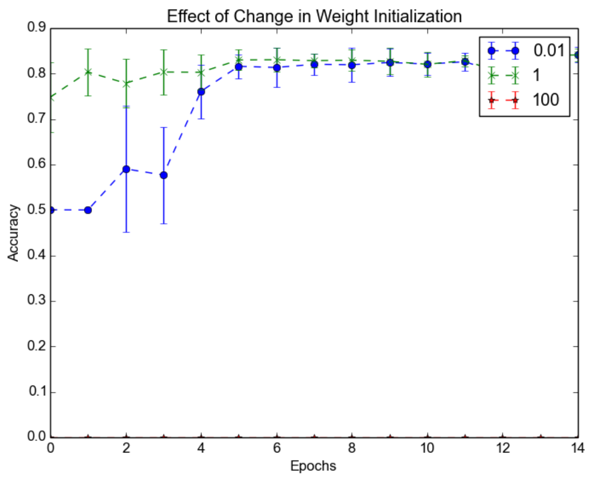

Figure 4: Effect of classifier on different test data sets

1.5 Part E

Different data sets have similar performance. Performance on dataset two appears to be the best. Test set 2 has the closest distribution of passes and fails as the model. Although, this does not completely explain the difference. The biggest issue is likely that the training data sets differs the least from set 2. Set 1 has many more passes than fails. This might suggest that the model tends to predict more passes than fails (false positives). Since Set 3 has more fails than passes, this bias may also explain the poorer performance of the classifier on the third data set.

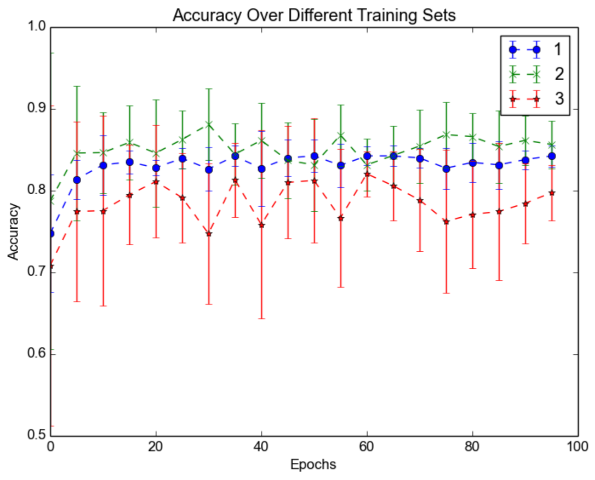

Figure 5: Effect of classifier on different test data sets

2 Question 2

I think the distribution of passing and failing examples in the data is causing worse performance. One method for improving the quality of the second network is to train using a different loss function. Instead of focusing on only on correct classification, adding increased penalty to false positives and false negatives may help the network find a better decision boundary.

2.1 Part A

In contrast to the first data set, more hidden units appears to converge faster. Potentially, because there are many more pass examples than fail examples in this training data set, the smaller network has a hard time finding the decision boundary.

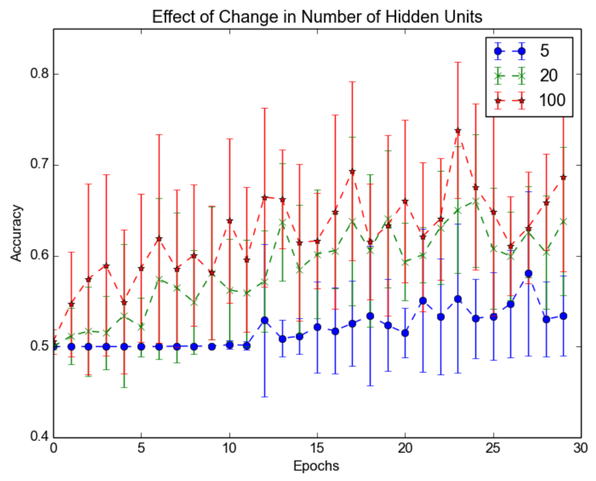

Figure 6: Effect of changing number of hidden units

2.2 Part B

Similar to the first data set, at some point, training causes overfitting to the training set and performance on the test set decreases. However, the time for convergence is much longer for this data set as compared to the first data set. 200 epochs appears to be insufficient to reach convergence.

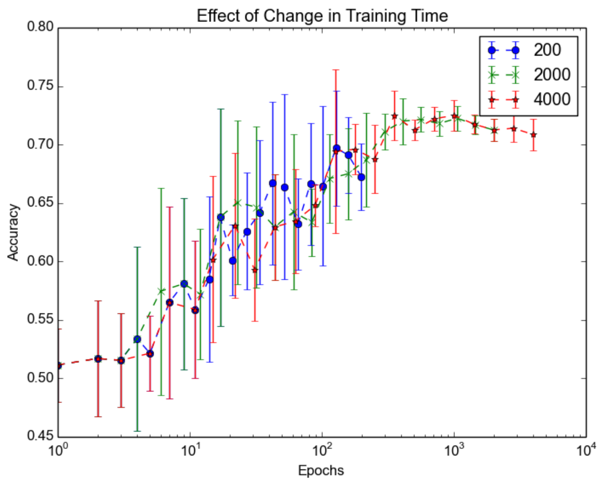

Figure 7: Effect of changing traning time

2.3 Part C

Learning rate has a similar effect as for the first data set. In this case, the very small learning rate appears to be unable to find a good solution. It appears to get stuck in a terrible solution which predicts the same value every time.

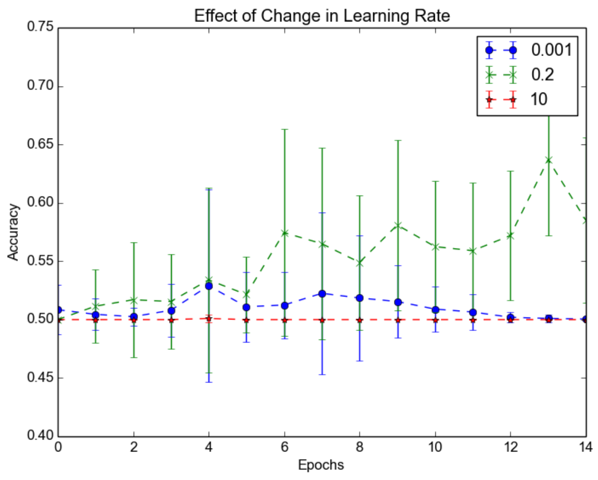

Figure 8: Effect of changing learning rate

2.4 Part D

I examined how initialization of the weights affects the algorithm. The legend shows the magnitude of the initial values. The results are similar to the first data set. Too large and the network saturates and too small and the network is unable to learn.

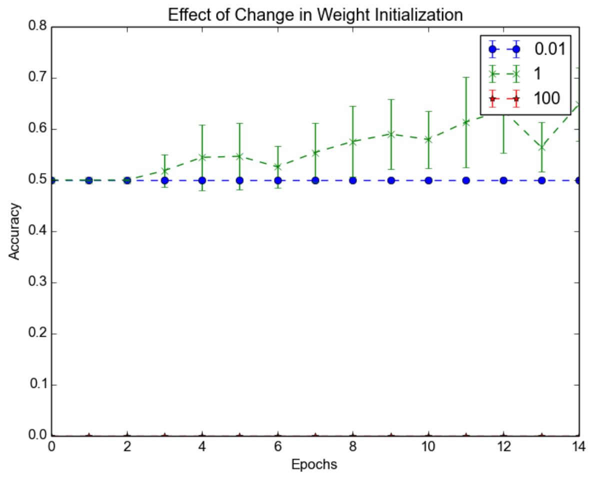

Figure 9: Effect of classifier on different test data sets

2.5 Part E

The second network seems to excel where the first network struggles, but overall has worse average performance across all the data sets. The number of passing and failing examples in the second test set is most similar to the second training set. Due to the similarity, the network performs well on the second test set. The first test set has an even number of passing and failing examples and the performance is average. The third data set has mostly passing examples. The large difference in the distribution of passing and failing examples causes the network to predict poorly.

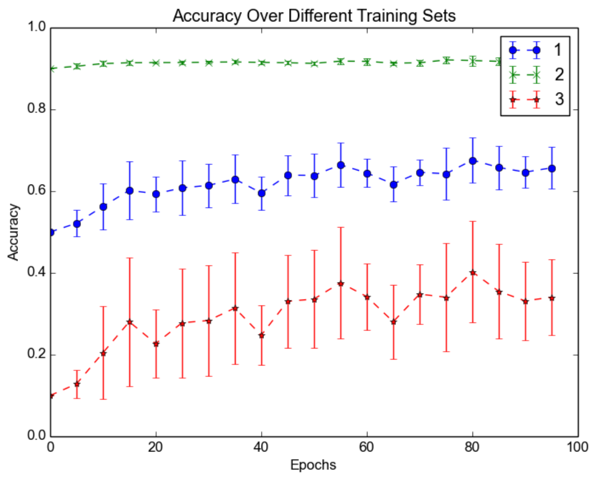

Figure 10: Effect of classifier on different test data sets