****

**CZ3006 Net-Centric Computing**

**Assignment 1 Lab Report**

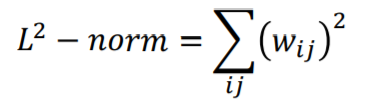
**PHANG JUN YU**

**U1522651D**

Declaration: This assignment was completed individually, and no parts were left unfinished.

**Introduction**

**Methods**

**Ridge Regression**: For ridge regression, the formula to calculate the L2 Norm is 

The L2 norm is multiplied by the decay parameter and the product is added to the final cost function. The reason for doing this is to prevent a case of overfitting where weights gain extremely large values and become extremely skewed, removing their ability to generalize. By adding the L2 norm of the weights to the cost, the minimizer that strives to minimize the cost would also cause the weights to refrain from getting too large.

In the code, the L2 norm is obtained using the tensorflow function *tf.nn.l2\_loss* for each weight and then summing them together.

The final loss is calculated as the initial loss + (L2 norm \* decay parameter).

**Search Space**: This is usually calculated by running a simple for loop. The records are kept in an array and plotted to find the optimal value.

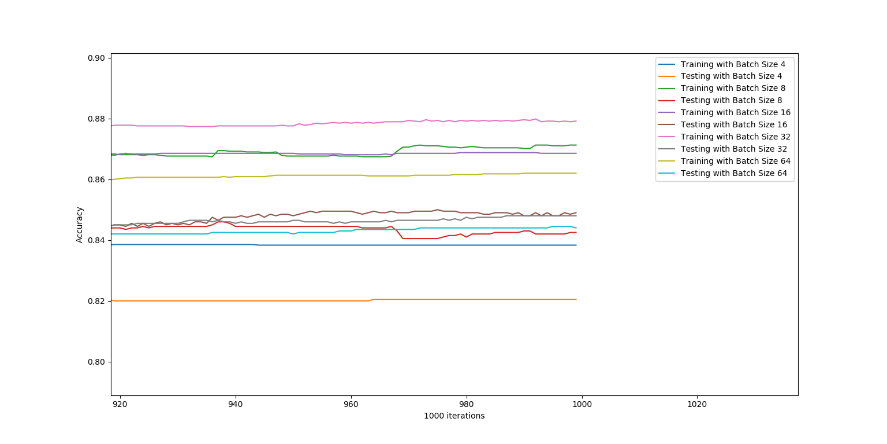
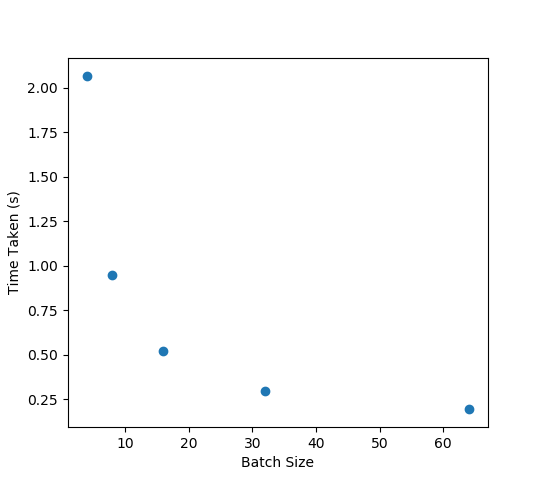
**Cross-Fold Validation**: Cross-Fold validation is used to make more efficient use of data. By re-using different parts of the data for different purposes, we can get a more accurate assessment of the overall accuracy of the model as the variance is reduced since it has been tested on “more” data. In this project, cross-fold validation is implemented in a sequential manner. For k folds, the code loops through k iterations and the test set will be the interval of (n/k) data points where n is total number of data points.

**Dropout**: Dropout is another method to reduce overfitting. In the code, this is simply implemented with the tensorflow variable when using *run()*. The code simply adds in an additional variable: *keep\_prob* and we set the value to be 0.9 as specified in the question.

**Time-performance optimization:** In this project, there are several questions which require a decision to be made on the optimal value based on the performance of accuracy against time taken. The strategy for deciding is always made by first taking the two or three highest values that give the best accuracy. The optimal value is decided by taking the value that has the greatest increase in performance to decrease in time taken ratio – in other words, efficiency.

**Testing with small datasets:** In “Small Datasets Figures”, the graphs were all obtained by running the code through train and test sets of size 1000. In “Figures”, the results were obtained by evaluation on the full train and test sets.

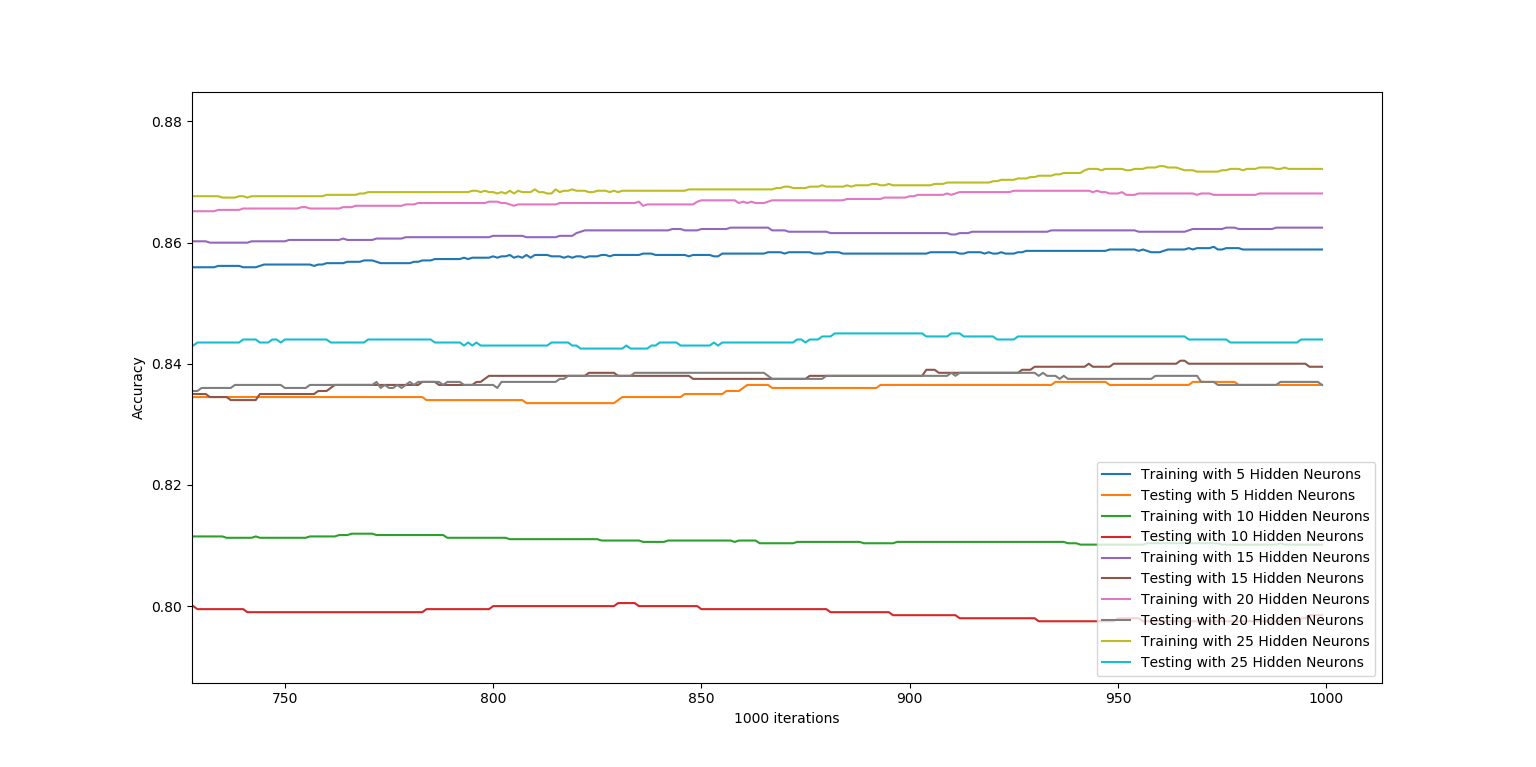
**Experiments and Results**

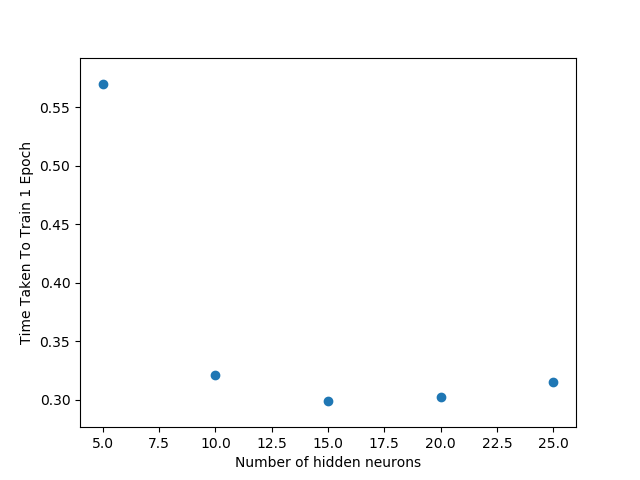
**1A Q2 – Finding optimal batch size**

As can be seen from the first graph, the best testing accuracies obtained were from batch sizes of 16, 32 and 64. Since both 16 and 32 have very similar accuracies while 64 is a significant distance away, we will focus on the comparison between 16 and 32. Timing-wise, there is a significant decrease in time taken when batch size increase from 16 to 32 as compared to the negligible increase in performance. This means that a batch size of 32 is optimal since the accuracy is almost the same while the performance time is much better.

**Optimal Batch Size = 32**

**1AQ3 – Optimal number of hidden neurons**



****

In a similar analysis as the previous sub-question, the optimal number of hidden neurons here would be **15**.

**1AQ4 – Optimal decay parameter**

**Conclusion**