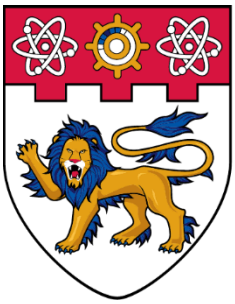


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4042

NEURAL NETWORKS & DEEP LEARNING

ASSIGNMENT 1

KOH TAT YOU @ ARTHUR KOH

(U1821604B)

Table of Contents

PART A: CLASSIFICATION PROBLEM

We employed the use of a *3-way data split* on the provided cardiotocogram data to partition the data into 3 subsets: *training*, *validation,* and *test* datasets. The reasons for doing so are detailed as follow:

Training Dataset
- Learning of model parameters.
- Training of model.

Validation Dataset
- Comparison of error values.
- Selection of best model.

Test Dataset
- Estimation of true error values.
- Assess model performance.

We accomplish this 3-way data split through a 2-step process. The steps and the reasons for doing so are as follow:

1. The provided cardiotocogram data is first shuffled.
   - It is likely that patterns of the same class are *clustered in proximity* to each other within the provided dataset.
   - Thus, shuffling is done to remove any *inherent clustering patterns* that might introduce unwanted bias.
2. The provided cardiotocogram data is then split into a 7:3 ratio.
   - 70% of the data is regarded as the *training* set.
   - 30% of the data is regarded as the *test* set.

Input Feature Scaling

We utilized *min-max normalization* for input feature scaling. This is done to ensure that the network properly learns the weights for all features, instead of focusing primarily on a few dominant features that would skew the learning outcome of the network and introduce unwanted bias.

Model Building

We constructed a 3-layer feedforward neural network for classifying the provided cardiotocogram data. The architectural details of the model's layers are detailed as follow:

Input Layer

- 21 nodes were used.
- Each node is meant to correspond to the **NUM_FEATURES** provided for each pattern (21 in total).

Hidden Layer

- 10 neurons were used.
- The number of neurons is determined by the provided assignment scenario specifications in this instance.

Output Layer

- 1 neuron was used.
- This neuron is meant to perform classification by outputting a batch 3-dimensional vectors.

A 3-dimensional vector is utilized to store the resultant output. The resultant output represents the class (N, S and P) probability prediction values corresponding to each input pattern. The classification class with the highest probability amongst the 3 classes will be then determined as the predicted class for a particular input pattern. Other aspects relating to the model are detailed as follow:

Parameter Optimization

- Parameters are updated and optimized using gradient descent, in order to minimize the cost function.

Cost Function

- A function defined as the summation of multi-class cross entropy values.

Accuracy

- A metric defined as the percentage of correct resultant output class predictions (N, S and P).

Assessment of the model performance during the model training process is done by plotting the following 4 performance metric values:

1. *Training Accuracy*
2. *Test Accuracy*
3. *Training Error*
4. *Test Error*

5-Fold Cross Validation

We chose to perform 5-fold cross-validation for the training set, as dictated in the assignment brief. This is to ensure that every singular example within the training dataset will be used for both training and validation. There are 2 types of folds that will be involved, namely:

Cross-Validation Folds

- Meant to facilitate the gradient descent learning of optimal weight values.

Validation Folds

- Meant to facilitate determining each model's error value.

As per dictated in the assignment brief, we are required to select an optimal model before we can use it for testing. Thus, to select an optimal model, we must use a metric to decide on which model is the best. As such, the use of 5-fold cross validation will provide us with an accuracy metric value that can be used to select the most optimal model for testing. In this instance, the model with the highest cross-validation accuracy will be the model chosen for testing. After selecting the model, its hyper-parameter values will then be used to train the entirety of the training set. This entire training set consist of folds for validation and training, respectively, and will be used with the selected model to determine a new set of optimal weights that can be used for evaluative and predictive purposes on the previously created test set.

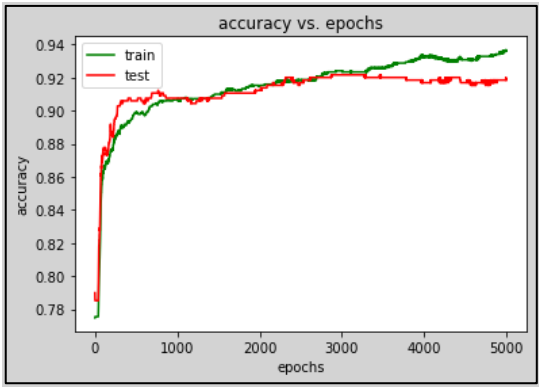
The parameters in the model are initialized to the following values as such:

NUM_FEATURES:	21
NUM_CLASSES:	3
hidden_units:	10
learning_rate:	0.01
beta:	10^{-6}
batch_size:	32
epochs:	5001

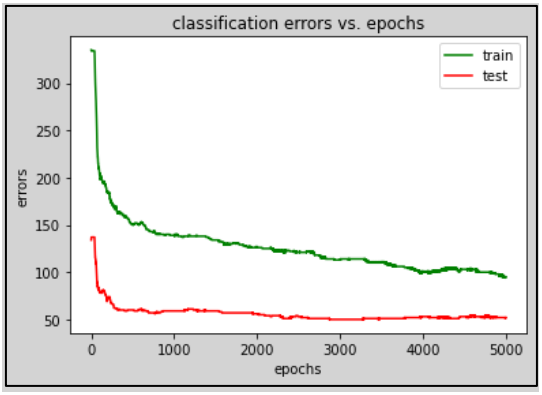
EXPERIMENTS & RESULTS

Question 1(a)

We used the training dataset to train the model and plot accuracies on training and testing data against epoch iterations, as specified in the assignment brief. This is graphed below as follow:



Graphed above is the train and test accuracy values plotted against the number of epochs.



Graphed above is the train and test error values plotted against the number of epochs.

Question 1(b)

We are also required to determine the epoch iteration that produces the best error convergence value. To do so, we printed metric values every 1000 epoch iterations as such:

```
[Iteration 1]
 train accuracy: 0.7748655676841736
 test accuracy:  0.7899686694145203
 train error:    335
 test error:     134

[Iteration 1000]
 train accuracy: 0.9065860509872437
 test accuracy:  0.907523512840271
 train error:    139
 test error:     56

[Iteration 2000]
 train accuracy: 0.914650559425354
 test accuracy:  0.9122257232666016
 train error:    127
 test error:     50

[Iteration 3000]
 train accuracy: 0.9240591526031494
 test accuracy:  0.9216300845146179
 train error:    113
 test error:     55

[Iteration 4000]
 train accuracy: 0.9327957034111023
 test accuracy:  0.9169278740882874
 train error:    100
 test error:     53

[Iteration 5000]
 train accuracy: 0.9361559152603149
 test accuracy:  0.9184952974319458
 train error:    95
 test error:     52
```

As we will be looking at the test error values every 1000 epoch iterations, the iteration-test error pair values we will be comparing amongst are tabled from the printed metric values on the left as follow:

Epoch Iterations	Test Error Value
1	134
1000	56
2000	50
3000	55
4000	53
5000	52

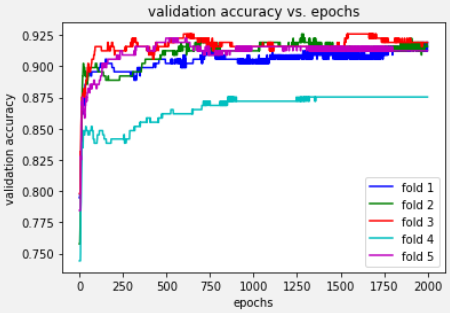
From the table above, we can see that the test error at 2000 epoch iterations is the lowest value amongst all the test error values produced by each epoch iteration.

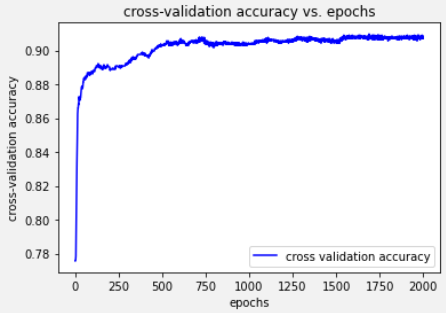
As such, we conclude that the approximate number of epoch iterations required for test error, and hence model convergence, is at **2000** epoch iterations.

Question 2(a)

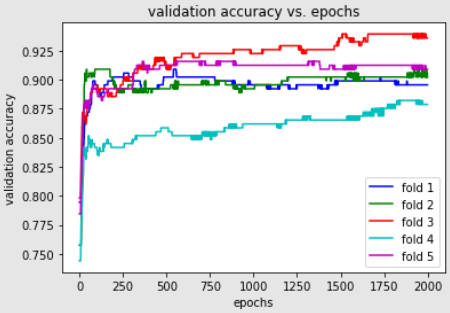
We have plotted cross-validation accuracies against epochs for batch sizes in the graphs below:

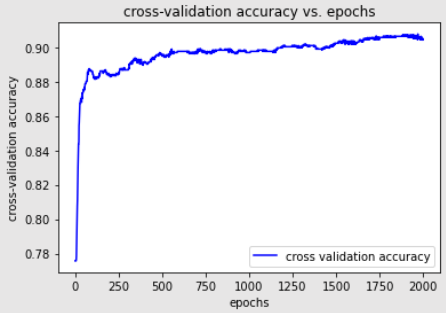
Batch Size [04]



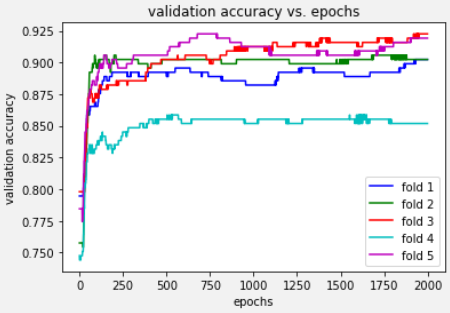


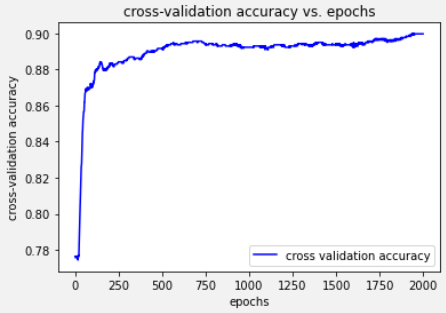
Batch Size [08]



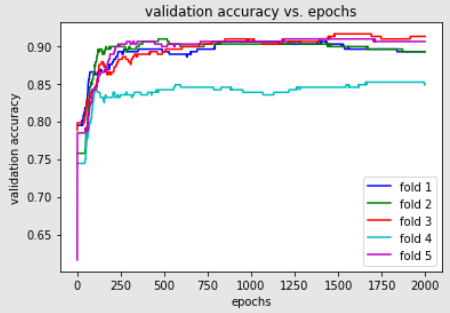


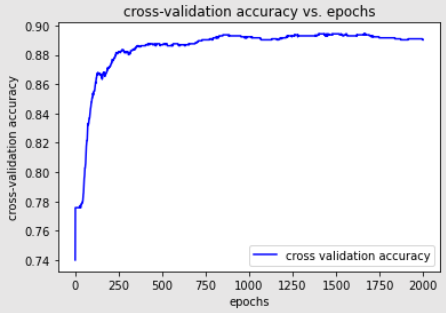
Batch Size [16]



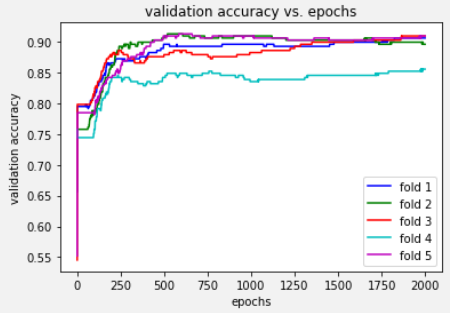


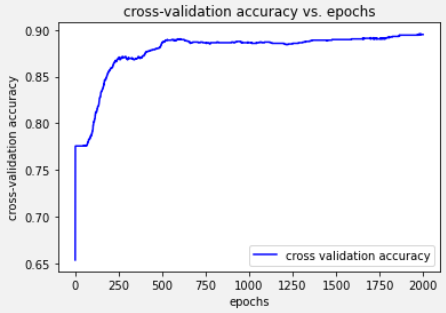
Batch Size [32]





Batch Size [64]



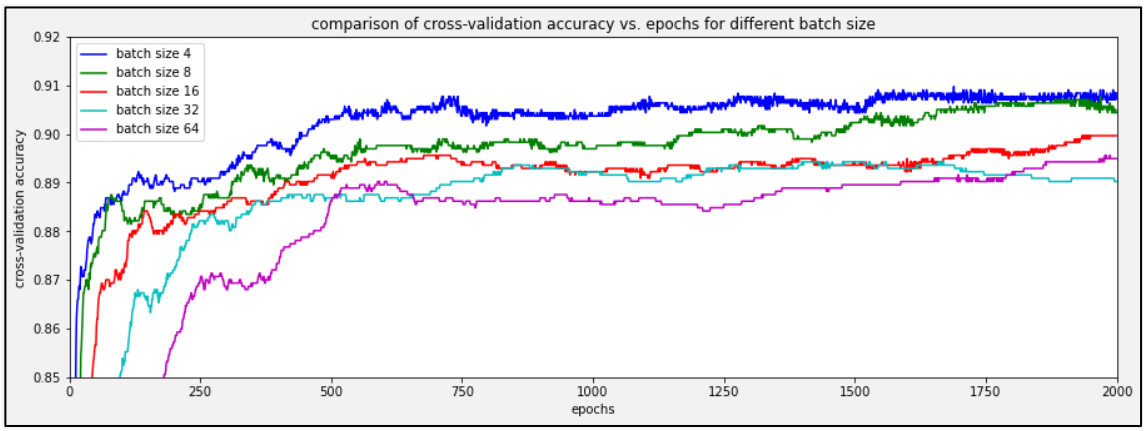


We plotted two sets of graphs for each batch size. As dictated in the assignment brief, the batch sizes we have plotted for are in the value range of: {4, 8, 16, 32, 64}.

For each batch size, we plotted validation accuracy values for each fold against the number of epoch iterations for the first set of graphs on the left side of the previous page. Conversely, we plotted cross-validation accuracy values against the number of epoch iterations for the second set of graphs on the right side of the previous page.

From the left graphs in the previous page, we observe that fold 4 has a *generally much lower* validation accuracy compared to the other folds. One possible reason why this might be the case could be due to *data imbalance* in fold 4, which introduces unwanted bias that decreases validation accuracy.

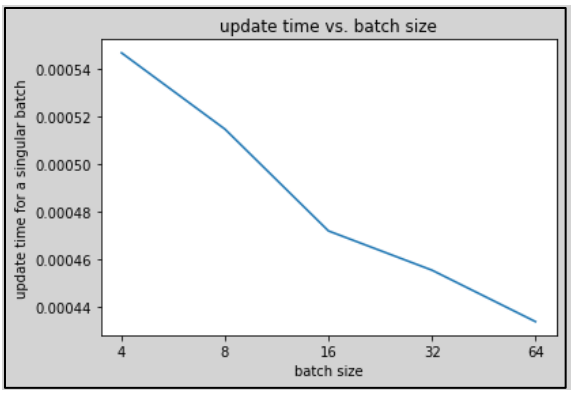
In order to compare the relative performance of the model for each batch size in the previously provided value range of {4, 8, 16, 32, 64}, we plotted and compared the cross-validation accuracy against the epoch iterations for different batch sizes in the graph below:

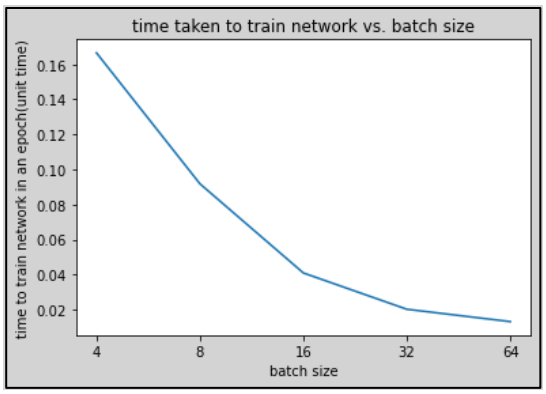


From the graph above, we make the following observations:

- Batch sizes 8 ranks below batch size 4 in cross-validation accuracy values, with the exception at around 100 epochs iterations in the beginning.
- Batch size 16 generally ranks below batch size 8 in terms of cross-validation accuracy value. The closest it comes to batch size 8 is at epoch iteration ~200.
- Batch size 32 ranks below batch size 16 before epoch iteration ~800. From epochs ~800 to ~1600, batch size 32 performs similarly to batch size 16.
- Batch size 64 performs the worst of the batch sizes lot, generally producing the lowest cross-validation accuracy values throughout.
- Batch size 64 performs slightly better than batch size 32 at epoch iterations ~500 to ~600 and ~1800 to ~2000.
- The greater the batch size value, the longer it takes for a usable (>0.85) cross validation accuracy to be reached

We are also required to plot the time taken to train the network for one singular epoch iteration against different batch sizes. This is graphed in the next page as such:



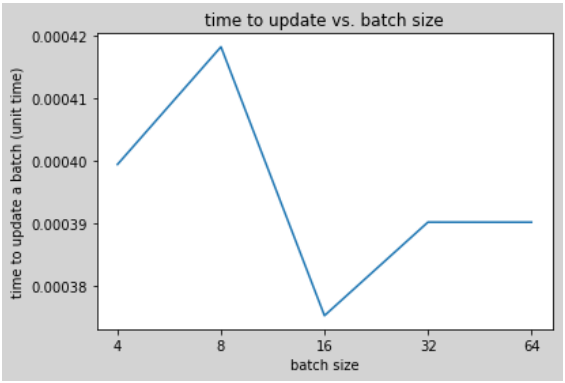


For each epoch, the time taken to train the network is derived using the formula below:

$$\frac{Training\ set\ size}{Batch\ size\ \times Time\ needed\ per\ batch\ update}$$

We observe from the right graph immediately above that there is a decreasing trend in the form of an inverse relation between the time taken to train for each epoch to the batch size. Thus, the time needed to train a singular epoch decreases at a decreasing rate, as batch size increases.

An anomaly is noted for the left graph above. Intuitively, we postulate that the time taken to update a smaller batch should be less than that of a larger batch. However, in the resultant graph we have plotted, we observe that the results are the inverse. Upon re-replotting the graph of the comparison, we are presented with these results instead:



Subsequent runs also produce other variations in the time required to update a batch size which do not consistently follow our initial intuition.

While there are many possible contributing factors for this phenomena, one possible reason could be due to the cache size of the GPU used for processing, as multiple displays were being used in concurrency.

Question 2(b)

We are required to choose an optimal batch size. In order determine this, we look at 2 criteria, namely: *cross validation accuracy* and *time to update a batch*.

Criteria 1: Cross Validation Accuracy

Batch size 16 is at a nice middle ground, performing generally in the average range between all the batch sizes we have tested.

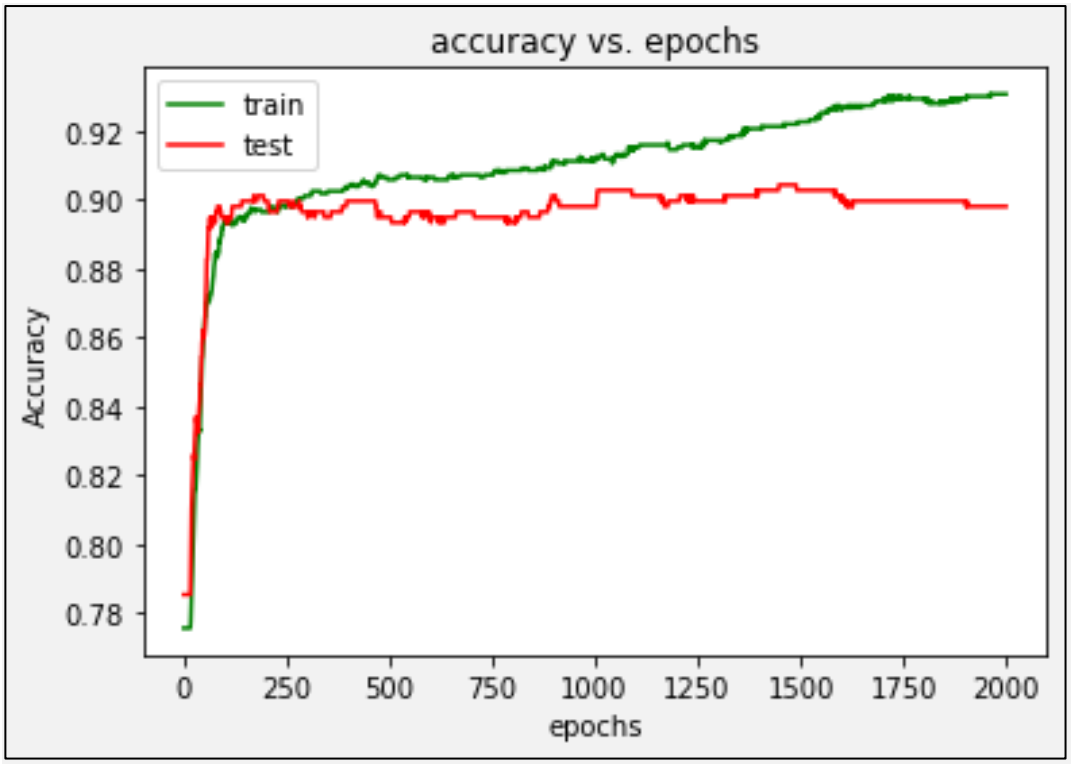
Criteria 2: Time to update a batch

Batch size 16 is lower end of the average range, with a steep drop from the time taken for batch size 8.

To balance a fine line between cross-validation accuracy and the time to update a batch in the network, we have chosen to select **batch size 16** as the optimal batch size.

Question 2(c)

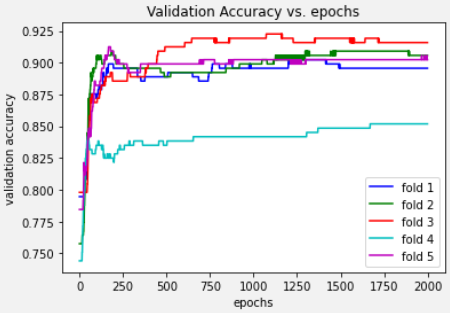
We plotted the train and test accuracies against epochs for the *optimal batch sizes* in the graph below as such:

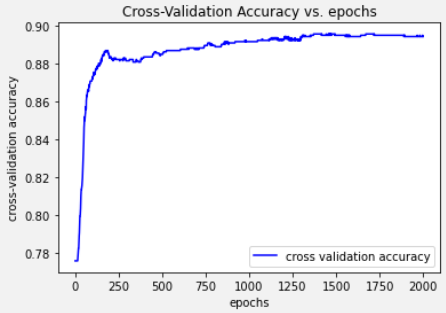


Question 3(a)

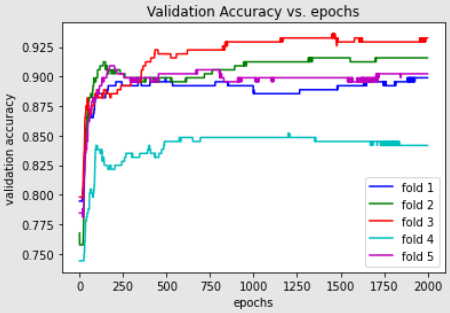
We plotted cross-validation accuracies vs. epochs for different hidden-layer neuron numbers:

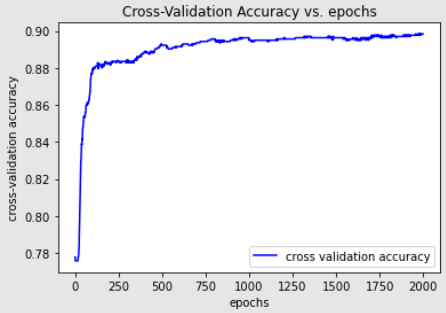
Hidden Neurons [05]



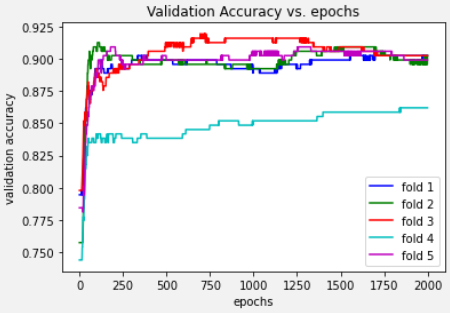


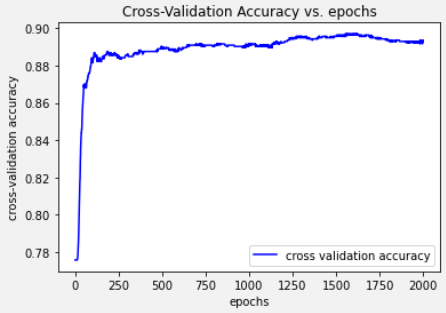
Batch Hidden Neurons [10]



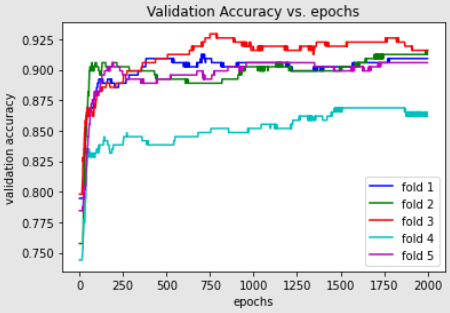


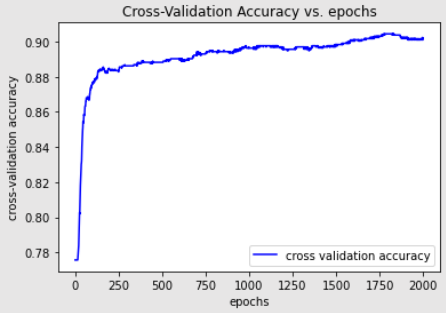
Hidden Neurons [15]



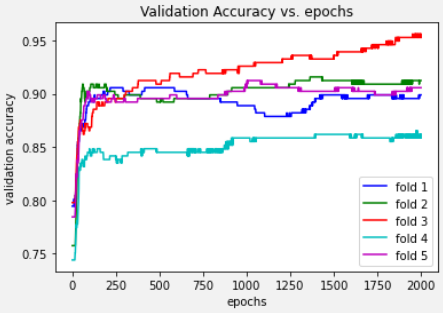


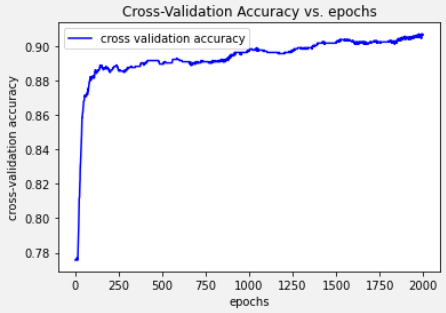
Hidden Neurons [20]



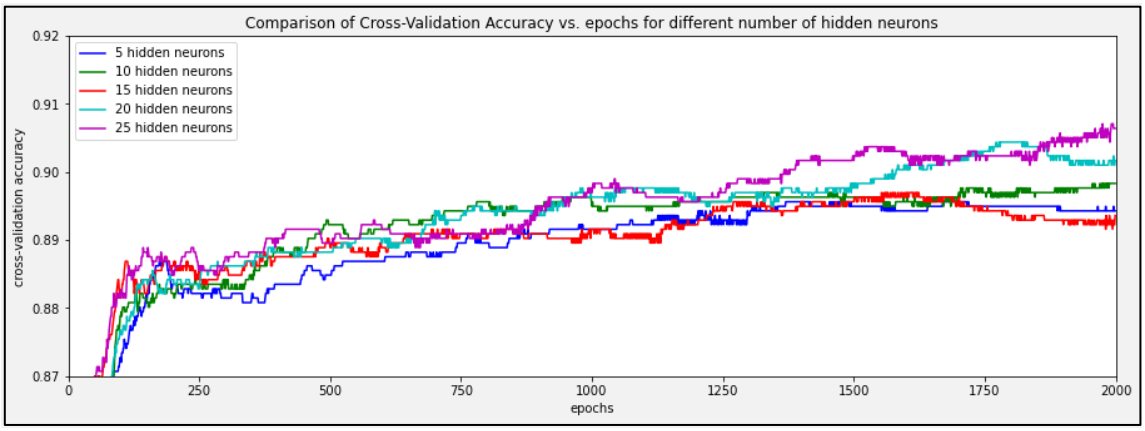


Hidden Neurons [25]





In order to compare the relative performance of the model for each number of hidden neurons, we plotted and compared the cross-validation accuracy against the epoch iterations for number of hidden neurons in the graph below:

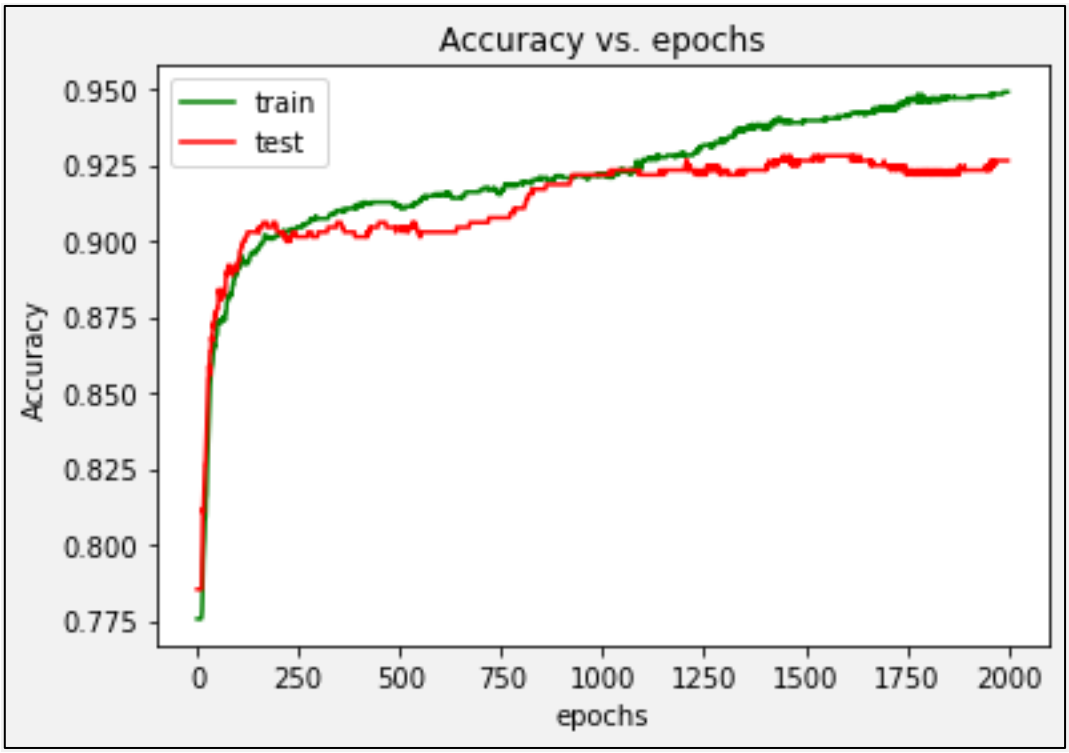


Question 3(b)

From the immediate graph above, we observe that the use of 25 hidden neurons produces a model with the highest cross-validation accuracy at the end of 2000 epochs. Throughout the entire epoch training cycle, the use of 25 hidden neurons produces a generally high-to-highest values of cross-validation accuracy. Thus, we select **25 hidden neurons** as the optimal value.

Question 3(c)

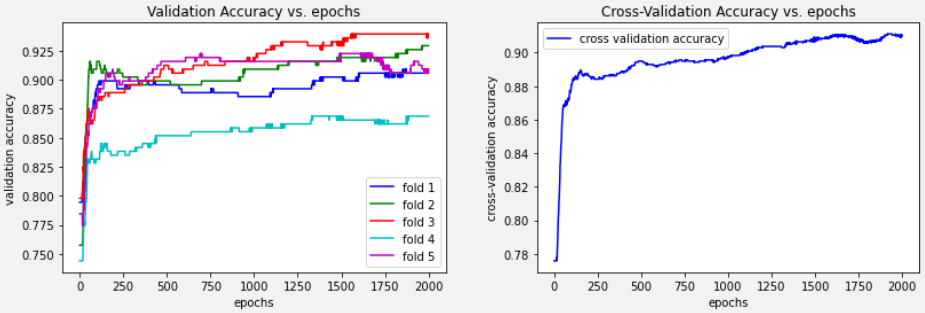
We plotted the train and test accuracies against epochs for the *optimal number of neurons* in the graph below as such:



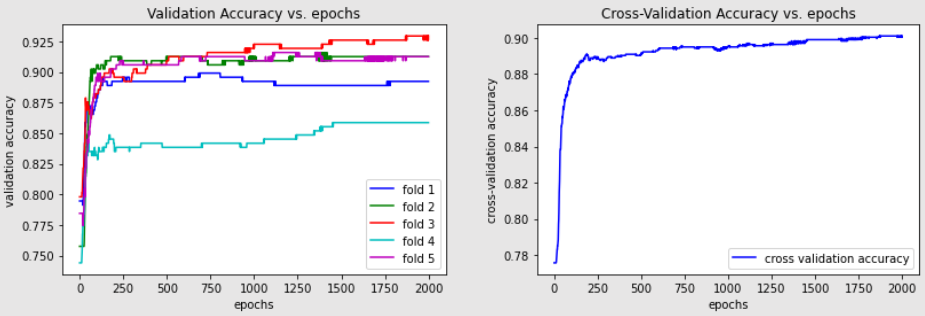
Question 4(a)

We plotted cross-validation accuracies vs. epochs for different decay values:

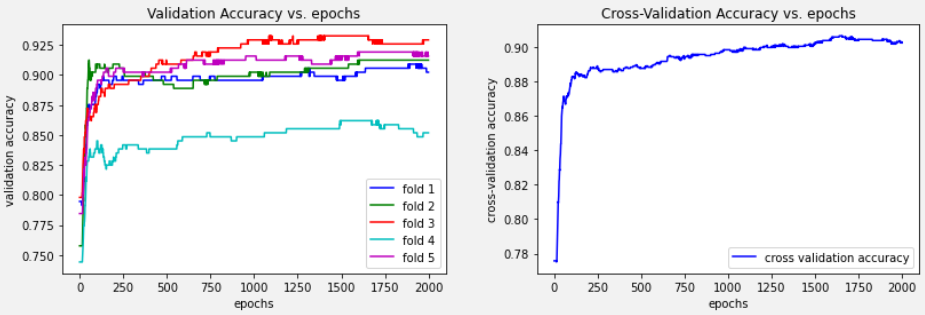
Decay Value [0]



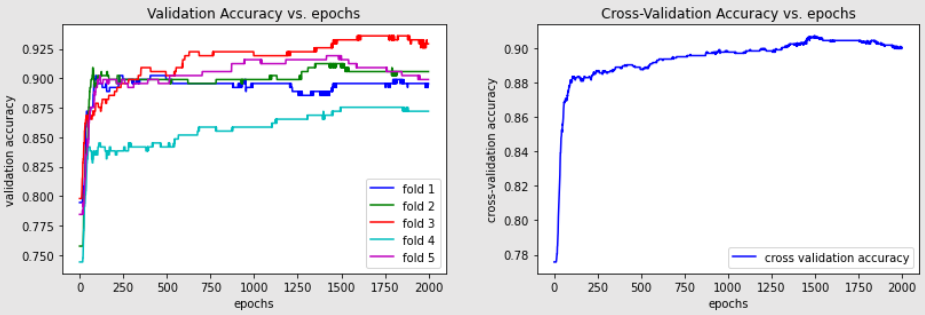
Decay Value [10^{-3}]



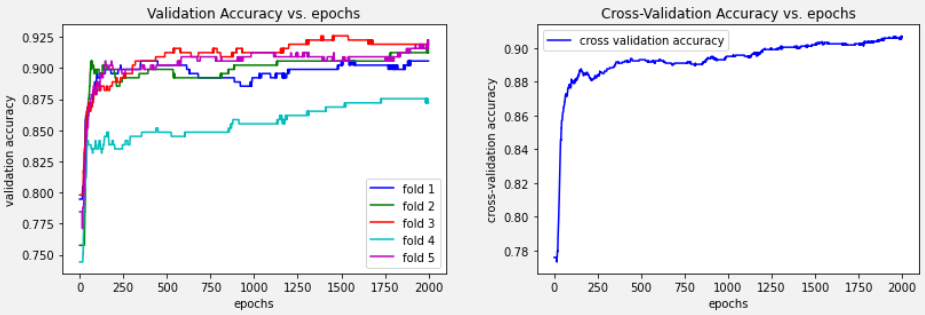
Decay Value [10^{-6}]



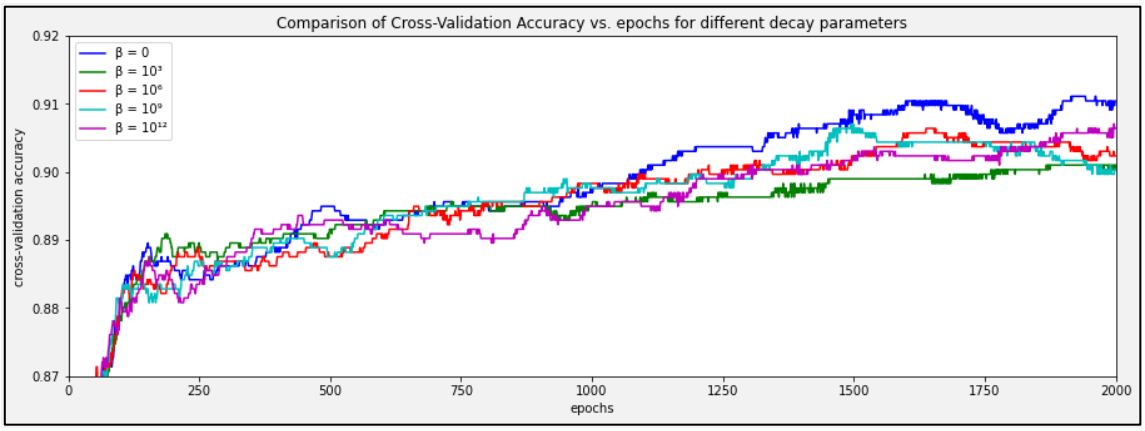
Decay Value [10^{-9}]



Decay Value [10^{-12}]



In order to compare the relative performance of the model for each decay value, we plotted and compared the cross-validation accuracy against the epoch iterations for decay values in the graph below as such:

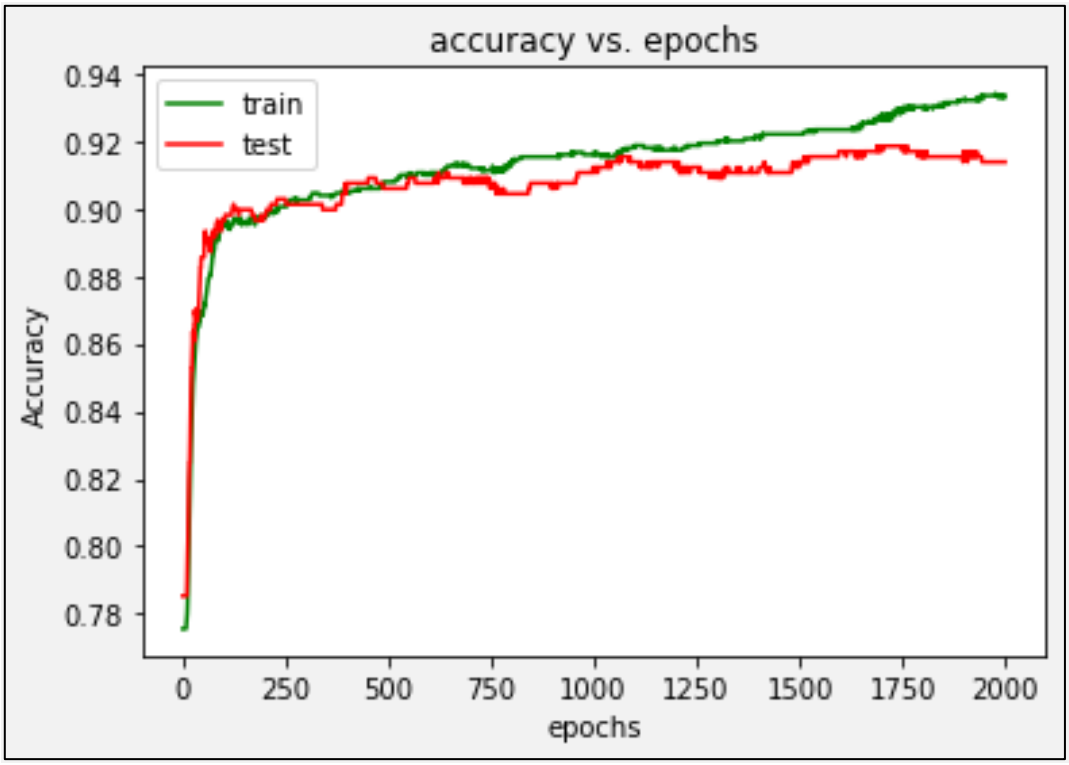


Question 4(b)

From the immediate graph above, we observe that the use of β = 0 for the decay value produces a model with the highest cross-validation accuracy at the end of 2000 epochs. Throughout the entire epoch training cycle, the use of β = 0 produces a generally high-to-highest values of cross-validation accuracy. Thus, we select **β = 0** as the optimal decay value.

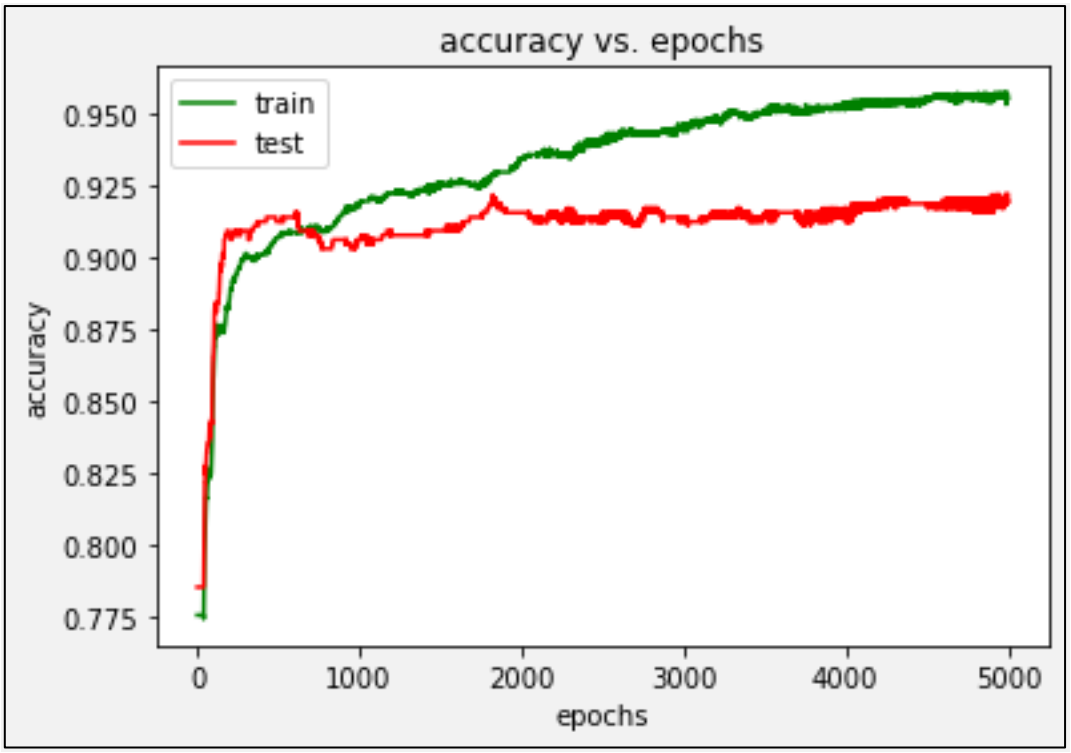
Question 4(c)

We plotted the train and test accuracies against epochs for the *optimal decay parameter value* (***β***) in the graph below as such:



Question 5(a)

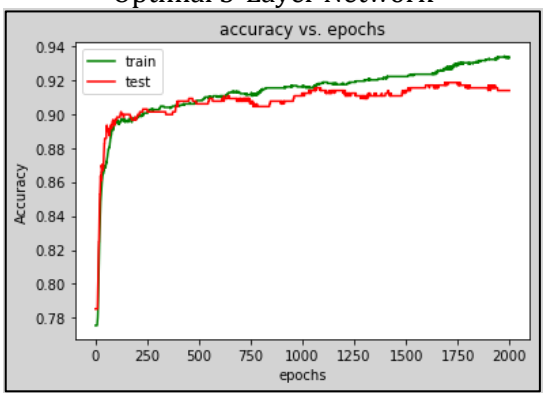
Similarly to 1(a), we used the training dataset to train the 4-layer network and plot accuracies on training and testing data against epoch iterations. This is graphed below as follow:



Question 5(b)

Shown below are the respective networks' performance and logged iteration printout values:

Optimal 3-Layer Network

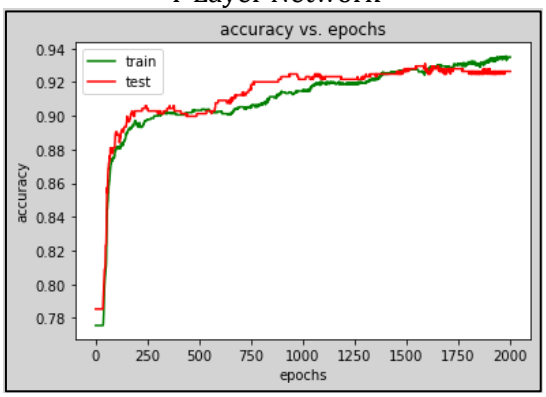



```
[iteration 1]
train accuracy: 0.7755376100540161
test accuracy:  0.7852664589881897

[iteration 1000]
train accuracy: 0.9159946441650391
test accuracy:  0.9122257232666016

[iteration 2000]
train accuracy: 0.9334677457809448
test accuracy:  0.9137930870056152
```

4-Layer Network




```
[Iteration 1]
train accuracy: 0.7755376100540161
test accuracy:  0.7852664589881897

[Iteration 1000]
train accuracy: 0.914650559425354
test accuracy:  0.9216300845146179

[Iteration 2000]
train accuracy: 0.9348118305206299
test accuracy:  0.9263322949409485
```

From the above information, we observe that the performance for both networks are very similar to each other, with both networks achieving very close accuracy values.

While the values are very close, it is important to point out that for the same number of iterations, the 4-Layer network variant produces a higher accuracy value. As such, we postulate that the 4-Layer network variant will be able to achieve test convergence faster than the Optimal 3-Layer network variant.

While only 2000 epoch iterations were used, it is highly probable that greater differences will be highlighted with the use of a much larger number of epoch iterations (e.g. 10,000 epochs).

Additionally, if a model converges faster, it also inductively follows that said model could potentially overfit faster too. As a possible example, at 2000 epoch iterations for the 4-Layer network variant, the test and training accuracy values are very close to each other. While this does not mean for sure that overfitting is in effect (noise could be a cause of it instead), it points towards one possible problem that could arise from the usage of an arbitrary increased number of network layers.

CONCLUSION

The optimization of neural networks can be a very tedious task. This is because there are many factors involved in the training of a neural network beyond just the architectural considerations of designing a neural network model.

- For instance, we observed in Question 2(a) that the results pertaining to the duration required to process a singular epoch for each batch size was the opposite of our intuitive postulation – larger batch sizes were observed from the graph to take a shorter time to process, as opposed to a longer time. Upon replotting the graph, we discovered that the plotted time taken changed again for the various batch sizes.

- This meant that it was not possible to get consistent results for these minute-level operations in the, and that the real world differences are only observable when the magnitude of these processes are increased (e.g. time taken for a larger set of epoch iterations, rather than just a singular iteration).

- From a hardware standpoint, this could be plausible, as there could be constraints due to monitor usage (e.g. dual monitors that consume GPU compute resources, as well as watching videos while the model training is in progress in the background).

You cannot have both your cake and be able to eat it. Another issue with optimizing neural networks stems from resource limitations, in the form processing power, or simply put, *time*.

- For instance, batch size 4 clearly gives the best results. However, we did not choose it as the time taken to train a neural network with such a batch size is unfeasibly long. From this, it is quite telling that mini-batch gradient descent seems to be a more feasible real-world choice, instead of stochastic gradient descent.

- While time is an important consideration for training a neural network, what is often overlooked is also the time required to properly validate and select the best hyper-parameters for training a neural network model. The process of trying to accomplish this was a very long process for this assignment, as the results varied between re-runs of the model training process with the exact same parameters, and any issues with the code (even small ones) would result in lengthy wait for re-training to be completed.

Model convergence is not a guarantee.

- In some ways, it seems impossible to replicate and recreate the exact same results for neural network outputs, especially when dealing with huge epoch sizes that are influenced by many variables, such as the data input, batch sizes, decay values, network architectural designs, and even hardware considerations.
- One key reason for this is due to random seed values used, which gives a different starting point for every neural network that is trained (even with the same parameters, all other things considered).

PART B: REGRESSION PROBLEM

INTRODUCTION

The task is to predict the chance of getting admitted into a Master's Program based on several parameters present within the admissions_predict dataset. In order to do this prediction task, we are instructed to design a *3-layer feedforward neural network*.

METHODOLOGY

Train-Test Split

We employed the use of a *2-way data split* on the provided cardiotocogram data to partition the data into 2 subsets: *training*, and *test* datasets. The reasons for doing so are as follow:

Training Dataset

- Learning of model parameters.
- Training of model.

Test Dataset

- Estimation of true error values.
- Assess model performance.

Before we split the data into their respective training and test datasets, we employ shuffling to perform random subsampling. Model performance also be proved through cross validation during training, owing to the limited amount of provided data.

Input Feature Scaling

We utilized feature-wise Z-Score Standardization to normalize the input features for our model. This is to minimize the likelihood that network does not learn weights for other features due the presence of dominant feature(s) that overshadow other, less dominant features.

Model Building

We constructed a 3-layer feedforward neural network for classifying the provided Graduate Admissions data. The architectural details of the model's layers are detailed as follow:

Input Layer

- 7 nodes were used.
- Each node is meant to correspond to the `NUM_FEATURES` provided for each pattern (7 in total).

Hidden Layer

- 10 neurons were used.
- The number of neurons is determined by the provided assignment scenario specifications in this instance.

Output Layer

- 1 neuron was used.
- This neuron is meant to perform classification through the output of a 1-dimensional vector.

A 1-dimensional vector is utilized to store the resultant output. The resultant output represents the probability prediction value corresponding to admission. This is a value ranging between 0 and 1, indicting the chance of getting admitted into a Master's Program. Other aspects relating to the model are detailed as follow:

Parameter Optimization

- Parameters are updated and optimized using gradient descent, in order to minimize the cost function.

Cost Function

- A function defined as the mean squared error (MSE) with L_2 weight regularization.

Assessment of the model performance during the model training process is done by plotting the following 2 performance metric values:

1. Training Error
2. Test Error

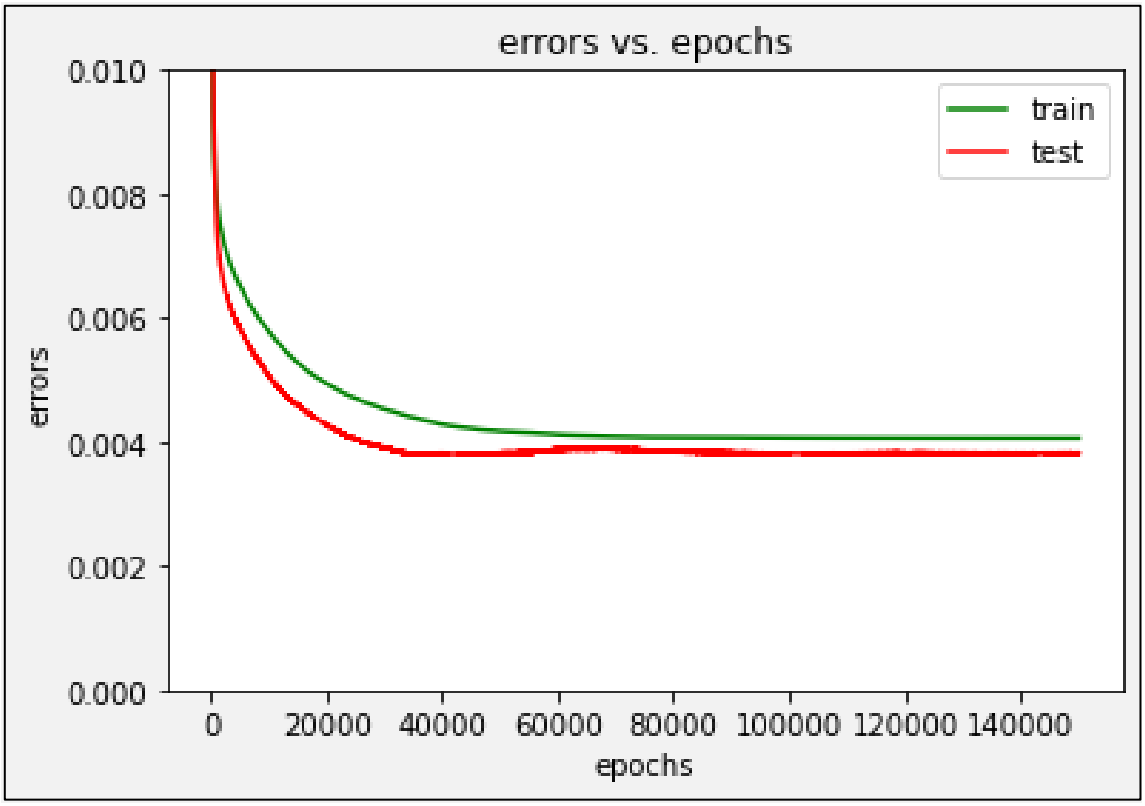
The parameters in the model are initialized to the following values as such:

NUM_FEATURES:	7
hidden_units:	10
learning_rate:	10^{-3}
beta:	10^{-3}
batch_size:	8
epochs:	150000

EXPERIMENTS & RESULTS

Question (1a)

We trained the model and plotted both train and test errors against epochs as graphed below:

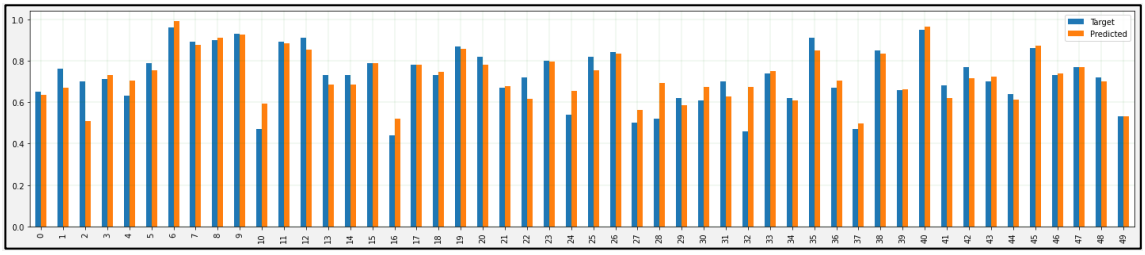


Question (1b)

We observe that the test error starts to increase at ~37,000 iterations. This observation is confirmed as the lowest test error throughout the training process is **0.003800114**. The optimal epoch iteration is at **36,511**.

Question (1c)

Using the previously obtained optima epoch iteration value, we re-train the neural network model for 36,511 epoch iterations. The re-trained model is then used to predict on the test set. In accordance with the assignment brief, we plotted a bar chart to showcase the target and predicted values for 50 test patterns as below:



Question (2)

RFE requires us to iteratively remove 1 input feature at a time, before re-training the neural network model after each feature removal. This removal should cause either change to occur:

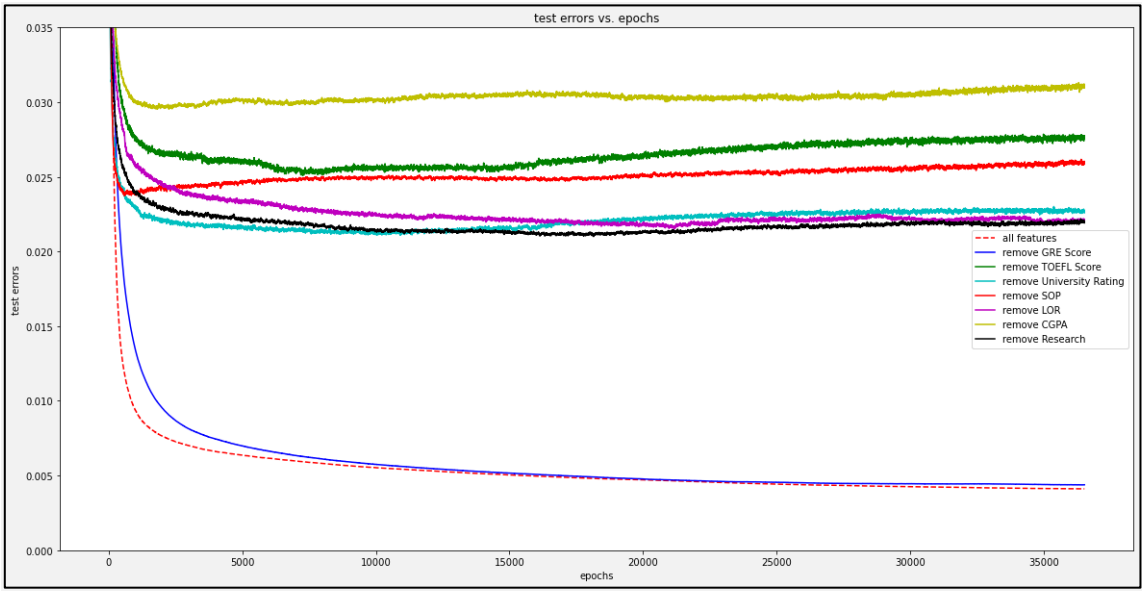
Minimal Decrease Occurrence

- A minimal decrease in the model's ability to predict the likelihood of Master's Program admission.

Maximum Increase Occurrence

- A maximal increase in the model's ability to predict the likelihood of Master's Program admission

We begin the RFE process by training different models, with a combination of inputs that involving the removal of a singular input feature, and then plotting the resultant test errors. In this instance, only 6 features will be used for each model's input combination. This is plotted as test errors against epoch iterations for each individual input feature, and is graphed as such:

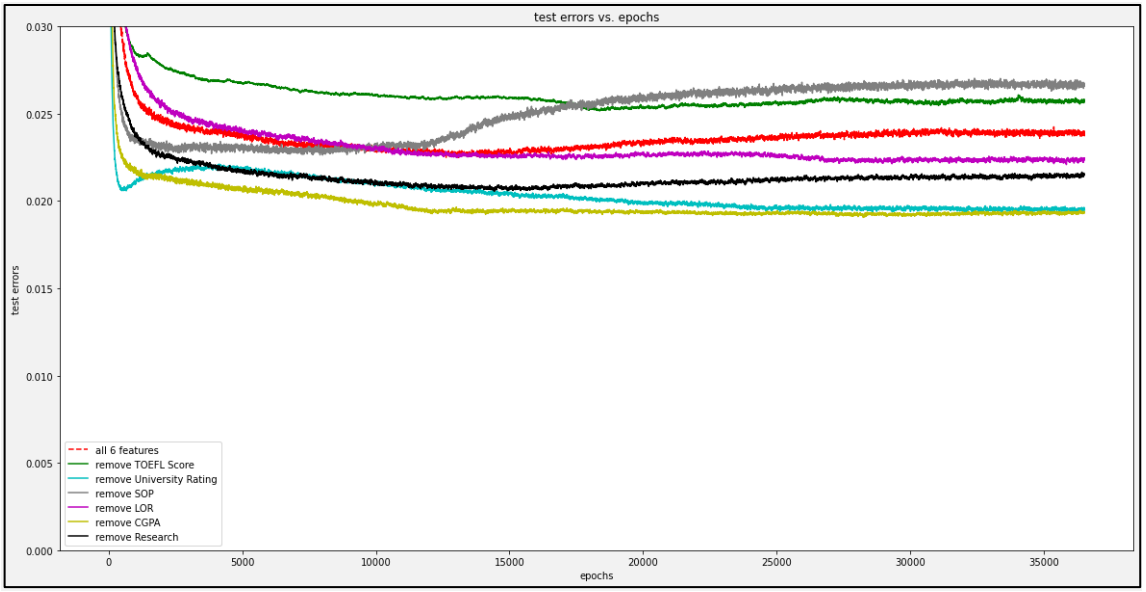


The plotting of the immediate graph above is meant to facilitate our decision flow in selecting which input feature is most suitable for removal, as part of the RFE process.

From the above graph, we observe that the removal of the *GRE Score* input feature results in a minimal increase in the test error values, as shown by the lowest curve in the graph in blue (right above the all features dashed curve in red). Owing to this observation, the first input feature that will be removed is the **GRE Score** feature.

As a result of removing the GRE Score feature, we are now left with only 6 input features. As such, we then proceed to repeat what we did previously – by training different model combinations involving the removal of a singular input feature, and then again – plotting the resultant errors.

In this iteration of the RFE process, only 5 features will be used for each model's input combination, and is graphed on the next page as follow:

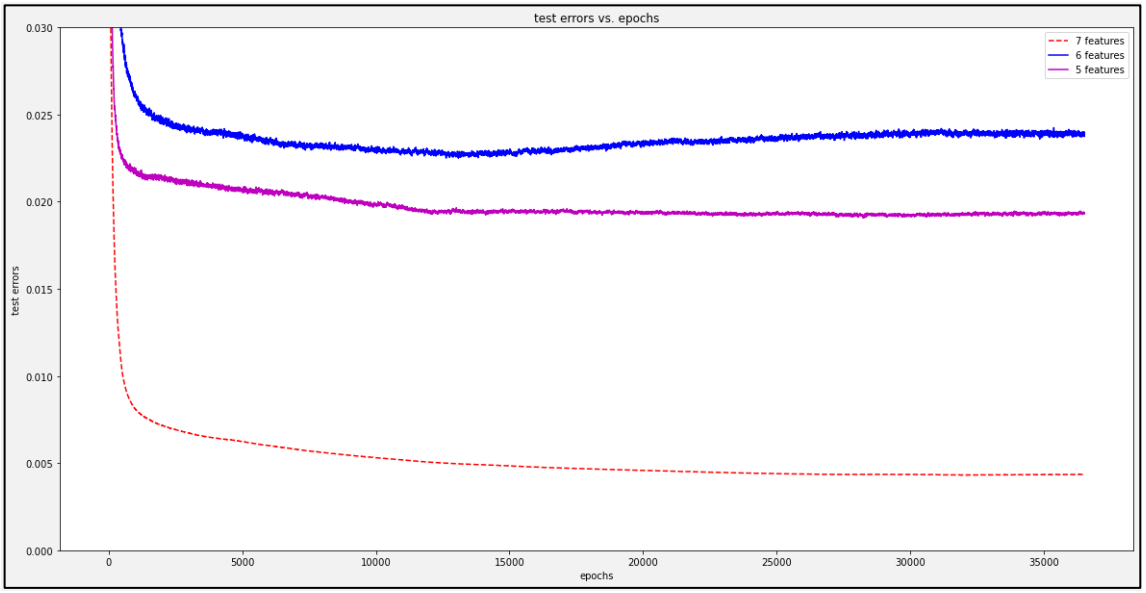


From the immediate above graph, we observe that the removal of the *CGPA* input feature results in the smallest overall increase in the test error, as shown by the lowest curve in yellow. Owing to this observation, the second input feature that will be removed is the **CGPA** feature.

After 2 iterative rounds of RFE has been finished, only *5 features* remain. They are noted on the right-hand side as such:

1. TOEFL Score
2. University Rating
3. SOP
4. LOR
5. Research

We then proceed to compare the model test errors produced as a result of using 7, 6 and 5 input features in the graph below as such:



From the immediate graph in the previous page, we comment on the following observations:

7 input features to 6 input features

The removal of the GRE Score input feature results in a ~3 times increase in test error. This suggests that GRE Score is actually an important feature for prediction, as it is known to be a significant contributor for admissions into a Master's Program, although this is domain-specific knowledge.

7 input features to 5 input features

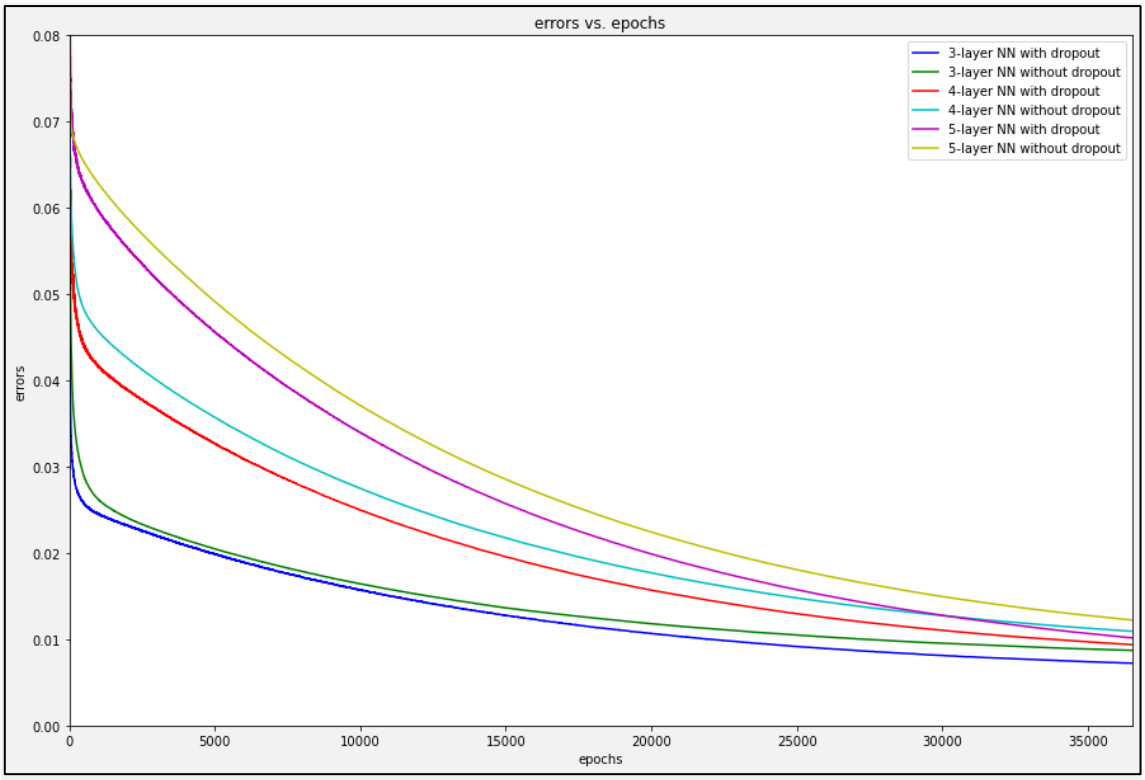
The removal of both GRE Score and CGPA input features results in a ~3.5 times increase in test error. This would mean that the model performs more poorly without CGPA as an input feature, and that like GRE Score, CGPA is also a good input feature for predicting the admission rate for a Master's Program.

In addition, there is an understanding that CGPA, GRE Score and TOEFL Scores tend to be strongly correlated. As such, the likelihood that these input features capture similar meta-data and patterns is also high. This multi-collinearity phenomena suggests that it might not be necessary to use all these 3 input features, as doing so would result in bias, due to how 1 input feature is essentially nearly repeated 3 times over, when these 3 input features are used together in this specific domain scenario.

Thus, by choosing only 1 of these 3 input features, this bias can be minimized. This results in a model that will be more accurately predict the likelihood of admission into a Master's Program.

Question (3)

Following the assignment brief, we have created and trained 3 neural network variants, in the form of four-layer, five-layer and three-layer neural networks. All 3 variants are also trained with and without dropouts. This is plotted as a graph of errors against epoch iterations as so:



From the graph above, we make the following observations:

Epoch Iterations
As the number of epochs increase, the differences in error amongst differing neural network variants for layer number decreases.

Fewer Layers
Neural networks that have lower errors tend to have fewer layers.

Dropout
Neural networks with dropout applied produce lower test errors than neural networks without dropout applied.

CONCLUSION

More is not always better.

- We observed that a complex neural network with more layers will take a longer time to properly train itself to recognize patterns in a dataset. This is contrary to the initial first thought that an increase in layers would lead to a model that can perform better predictions than models that use a lower number of layers.

Dropping out is good.

- We observed that the use of dropout can reduce model overfitting, which is helps improve model generalization. This means that the model would be able to better adapt to new, previously unseen data and still perform accurate prediction outputs.

RFE can be very, very time consuming.

- RFE is an automated feature selection that is used where domain knowledge is limited. While RFE in theory can identify the most optimal input features based on a set of input features, it is noted that it is a very exhaustive and time-consuming process.
- In our case, RFE for the provided dataset took 92.92 minutes.
- For pre-processing before the RFE process, there are more efficient ways to accomplish this.
    - We take reference from the field of statistics – where statistical methods can be used to identify strongly correlated features for feature removal.
    - Another method is the use of domain knowledge, by enlisting a domain expert to identify which are the candidate features for removal.
- The use of the above-mentioned 2 techniques can help significantly reduce the time taken for the RFE process to be carried out.