CE4042: Neural Network & Deep Learning

Individual Assignment 1

Name: Do Anh Tu

Matriculation No.: U1721947F

Date: 14th October 2020

1. Abstract

Neural networks are one of the many machine learning tools that are applied by machine learning engineers to solve real life problems. The process of building such networks are said to be iterative and are not as straight forward as it is believed by the layman. In this lab report, such demanding nature of building practical neural networks application is demonstrated with an attempt to solve 2 basics machine learning problem, classification, and regression, using simple multi-layer perceptron networks. Despite their rudimentary building blocks, the models that are being designed can have significant performance boost by making use of various regularization techniques, hyperparameter choices and model architectures. The process of designing the models, evaluating them, and improving their performance takes time and resource despite the lean design of the network, highlighting the challenging nature of this exciting field of machine learning.

2. Part A

2.1. Introduction of problem

In part A of this assignment, we are given the Cardiotocography dataset and tasked to design a multilayer perceptron network to predict one of the three fetal states (N: Normal, S: Suspect, and P: Pathologic) given a record of data. This dataset was reviewed and classified in consensus of three expert obstetricians.

Each data record has 23 fields. While there are multiple labels available in our dataset, the main interest of this assignment was placed on building a predictive model for the NSP label. Thus, we have 21 features/attributes and 1 label to work with in the designing and evaluation of this model.

Once a baseline model was designed, we also attempted to lift the performance of the models with various modifications to the architecture and hyperparameters of the model. Firstly, we experiment without modifying the architecture of the baseline model by experimenting with hyperparameters: batch size for training, number of neurons in the hidden layers and lastly the beta decay factor for the L2 weight regularization. Then we experimented with modifying the architecture of the baseline model and comparing the lift in performance between this and the earlier model with optimal hyperparameters.

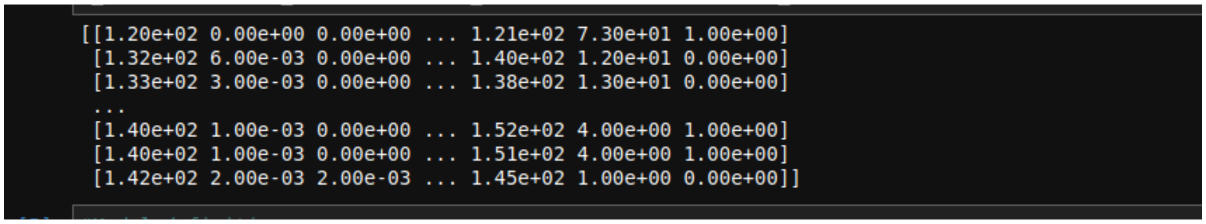
The results of this experiments shown how existing neural networks' performances may be improved by:

- Tuning the hyperparameters of the model
- Modifying the model architecture.

2.2. Methodology

2.2.1. Data preprocessing

There is clear evidence that the dataset has been cleaned so we do not have to deal with missing data or Nan values.



```
[[1.20e+02 0.00e+00 0.00e+00 ... 1.21e+02 7.30e+01 1.00e+00]
 [1.32e+02 6.00e-03 0.00e+00 ... 1.40e+02 1.20e+01 0.00e+00]
 [1.33e+02 3.00e-03 0.00e+00 ... 1.38e+02 1.30e+01 0.00e+00]
 ...
 [1.40e+02 1.00e-03 0.00e+00 ... 1.52e+02 4.00e+00 1.00e+00]
 [1.40e+02 1.00e-03 0.00e+00 ... 1.51e+02 4.00e+00 1.00e+00]
 [1.42e+02 2.00e-03 2.00e-03 ... 1.45e+02 1.00e+00 0.00e+00]]
```

Figure 1. Dataset attributes with varying range

However, it can be noted that the r**anges between the features are quite varied, from e-3 to e2 magnitude** (figure 1)**.** Therefore, perform normalization in terms of min-max values to ensure all the attributes for learning are of similar scale. This preprocessing step is important as some algorithms in machine learning will not work well with varying scale of input attributes.

Although it is common to convert the label of multiclass classification problem into a one hot encoding vector, this is not necessary since the Sparse Categorical Cross Entropy loss function rather than the Categorical Cross Entropy function is selected. The former also has advantage in term of memory and computation efficiency as it uses an integer to denote a class rather than a one-hot vector required by the later.

Train – test split of 70:30 ratio is applied on the original dataset. The training dataset (70%) will be use in model selection as discussed in model selection below, as part of a three-way data splits for model selection and training.

Another consideration for data preprocessing is that the normalization process shall be fitted only on the training set (min-max stats are from the 70% train set), and then applied on the train set and test set individually. This is to prevent the leaking of data from the test set to the train set, which can skew the results of our model during evaluation.

2.2.2. Model selection

To evaluate and compare the performance of different models (with different hyper parameters or model architecture), **cross validation with K fold and three-way data splits (train-validate-test set) will be applied.**

For this assignment, the number of splits is 5, thus we are t**raining a model with 4 partitions of train set and testing/evaluating it on the remaining 1 partition of the train set (called the validate set)**, then repeat this procedure 4 more times, ensuring that each partition of the train set gets to be evaluation set once. For interest of time and to prevent overfitting from training long epochs to affect our answer, early stopping with patience = 20 steps is applied to monitor the min validate loss in 5 folds experiments. **The choice of these parameters might need to be better quantified and qualified, which is not the scope of this report. This is left for future improvements.**

Once train and evaluated, the average accuracy of the 5 folds (max accuracy from each fold) will be calculated and represent the performance of the model being evaluated. **The model with the highest average accuracy amongst the 5 folds will be the optimal model with optimal architecture or hyperparameters to be selected.**

This optimal model is then trained on the complete train set (70% of original) and tested on the test set (30%). The evaluation is part of the train set and is only used in the 5-fold cross validation procedure.

K fold cross validation is important as we only have access to a finite set of examples, which is smaller than the population of observation we are trying to build the model for. This ensure the error rate that we estimated is not overly optimistic. We decided to do model selection on train set and test the optimal model (trained with train set on the test set) as the validation set is also used in the selection of that model, causing it to be biased. The test set on the other hand is independent and will be necessary to evaluate the estimate the performance of the selected model.

Lastly, it is important and hence we are emphasizing it here again that the model shall be evaluated only based on accuracy performance (not on any other factors like overfitting, time constraint, resource constraints, etc.). **Also, the evaluation will be based on mean values across the folds, no consideration on std of the values will be taken (disclaimer)**

2.2.3. Codebase organization

For each question, there might be multiple jupyter notebooks available to answer the various aspects of the question. While the results of these notebook have been pre-obtained, in order to replicate these results to answer the question, please follow the annex A, which explain what each of the notebooks in part A folders do and which order to run them.

2.3. Experiments and Results

2.3.1. Baseline 3-layer model (Qns A-1)

Code for this question is available in the notebook ***PartA/1A/1A.ipynb***

a) Use the training dataset to train the model and plot accuracies on training and testing data against training epochs:

The baseline model architecture is as following:

- 3 layers
- Hidden layer with 10 neurons and ReLU activation function, L2 regularization with Beta = 1e-6
- Training based on mini-batch gradient descent with batch size = 32 and learning rate of 0.01 (default, no need specifies)

The plot of model accuracies and model losses vs training epochs are shown in (figure 2) and (figure 3). We train the model with 1500 epochs and allows overfitting due to training long epochs to happens as shown in the plot (blue and orange diverges). The accuracy starts to overfit at about 1000 epochs. The test accuracy converges faster and earlier than that of the training accuracy.

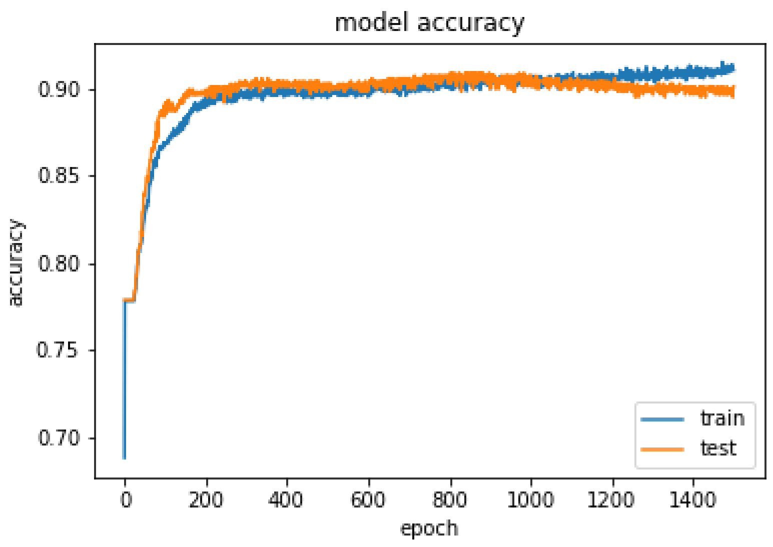


Figure 2 Train and Test accuracy of baseline 3-layer model

b) State the approximate number of epochs where the test error begins to converge

The approximate number of epochs where the test error begins to converge is about 500 epochs. As seen from (Figure 3), the decrease in test loss is very small and gradually and it takes longer to observe a decrease in loss level.

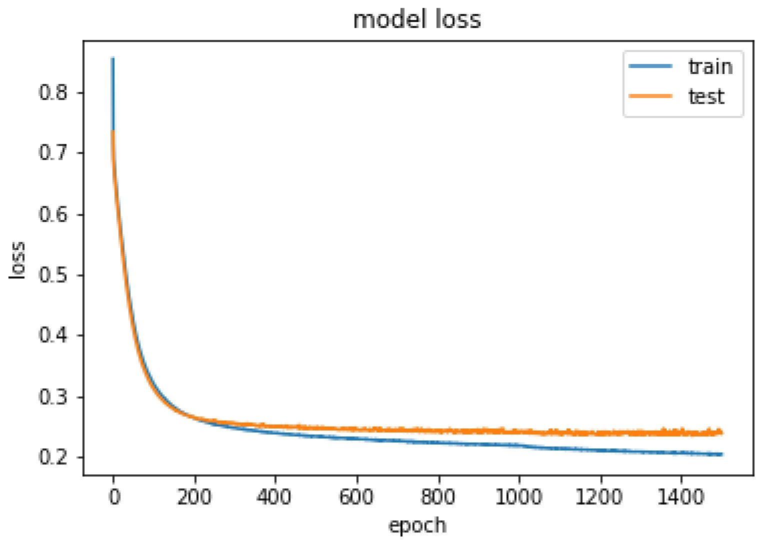


Figure 3. Train and test losses of baseline 3-layer model

The keras callbacks function EarlyStopping() were also used to test out this observation. With a patience of 20, monitoring the validation loss leads to the model early stopped at 521 epochs, meaning the convergence of validation loss (or test error) should starts at about 500 epochs. We provided the code in the same notebook but commented out in case the grader wants to try for himself.

2.3.2. Finding optimal batch size for 3-layer network (Qns A-2)

a) Plot the cross-validation accuracies against the number of epochs for different number of batch sizes [4,8,16,32,64]. Plot the time taken to train the network for one epoch against different batch sizes.

Cross-validation accuracy is defined as the mean accuracy of each batch sizes over the 5 folds. By logging down the training stats with a csv logger, the cross-validation accuracy curves below (Figure 4 to 8) are obtained by taking the average of 5 folds for each batch size. The detailed code and plot generation is shown in the notebook ***PartA/2A/2A-ave.ipynb***

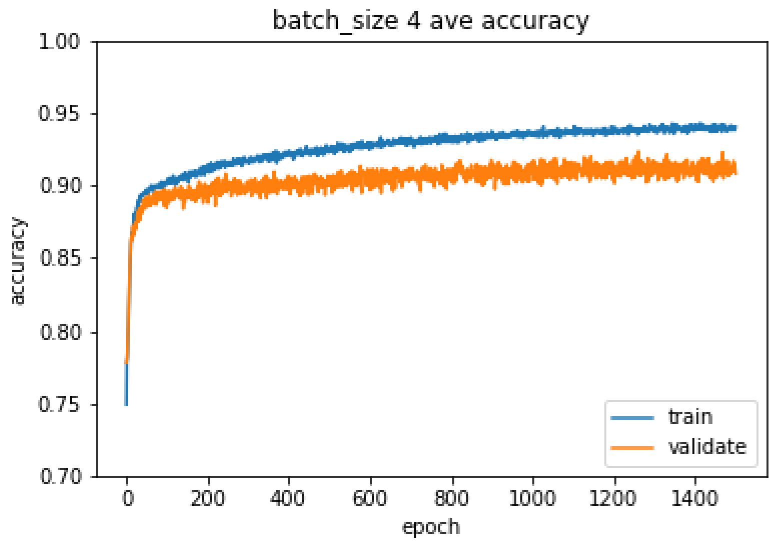


Figure 4. Average train and test/cross-validate accuracy vs epoch of batch_size = 4

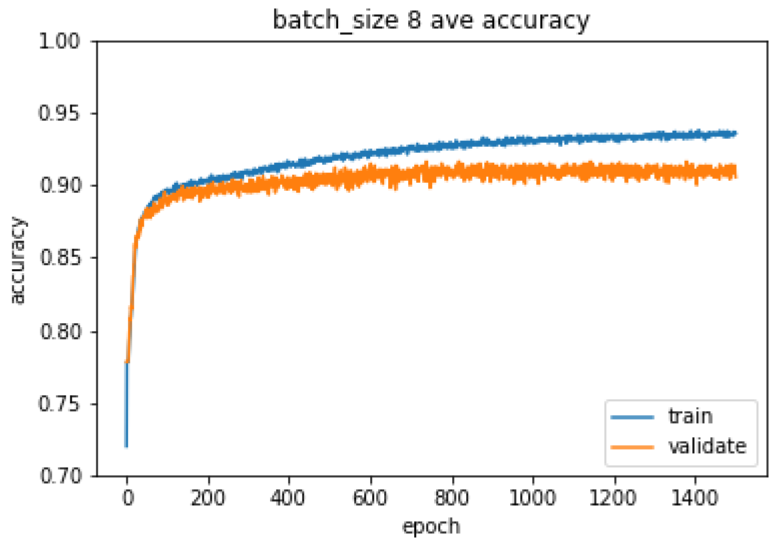


Figure 5. Average train and test/cross-validate accuracy vs epoch of batch_size = 8

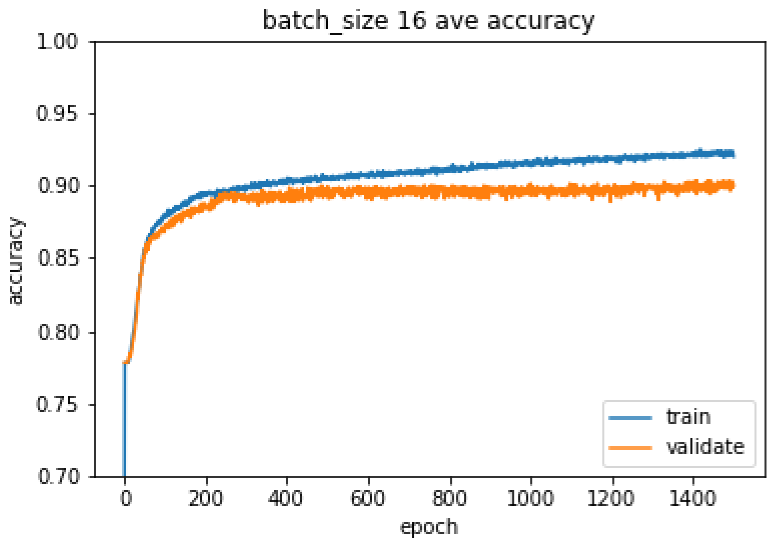


Figure 6. Average train and test/cross-validate accuracy vs epoch of batch_size = 16

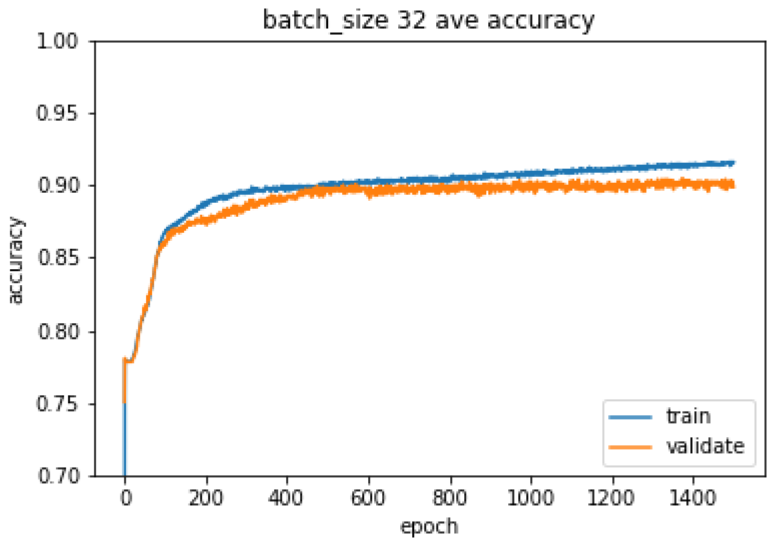


Figure 7. Average train and test/cross-validate accuracy of model with batch_size = 32

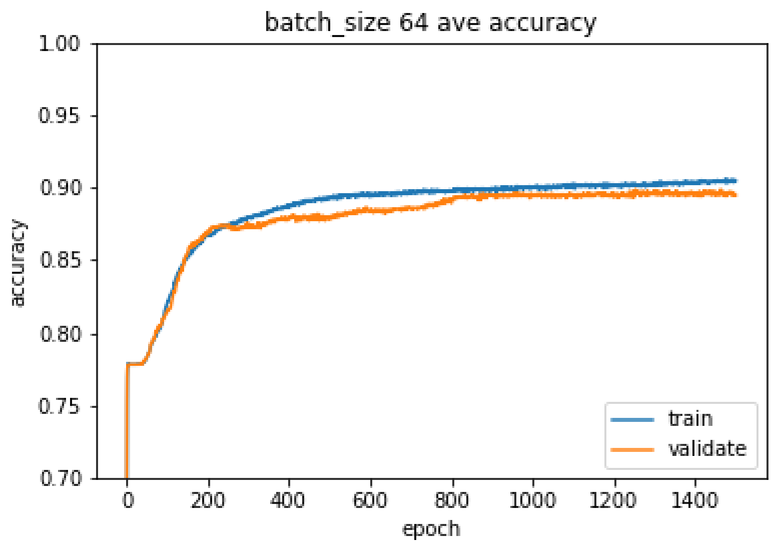


Figure 8. Average train and test/cross-validate accuracy of model with batch_size =64

Then, plotting only the cross-validation accuracy of these 5 models against each other yields the (figure 9) below:

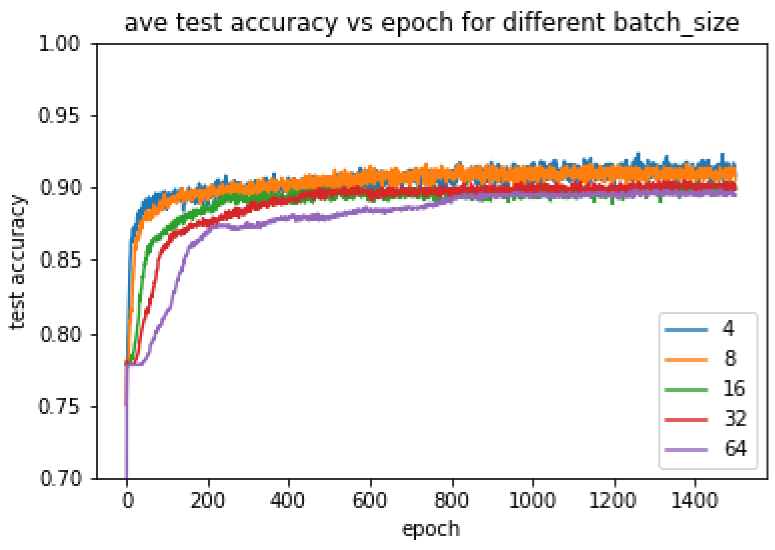


Figure 9. Average test/cross-validate accuracy vs epoch for different batch_sizez

There are also validate accuracies vs epochs plot for individual folds of different batch sizes available in the notebook ***PartA/2A/2A-overfit.ipynb***

The relationship between time taken to train the network for 1 epoch and the batch size used in training is shown in (figure 10). As time taken varies from epoch to epoch, we record the training time for 250 epochs per batch size and takes it average, s per We can see that as batch size increases, the average time required for 1 epoch decreases sharply. Training with a larger batch size will help to speed up the training process.

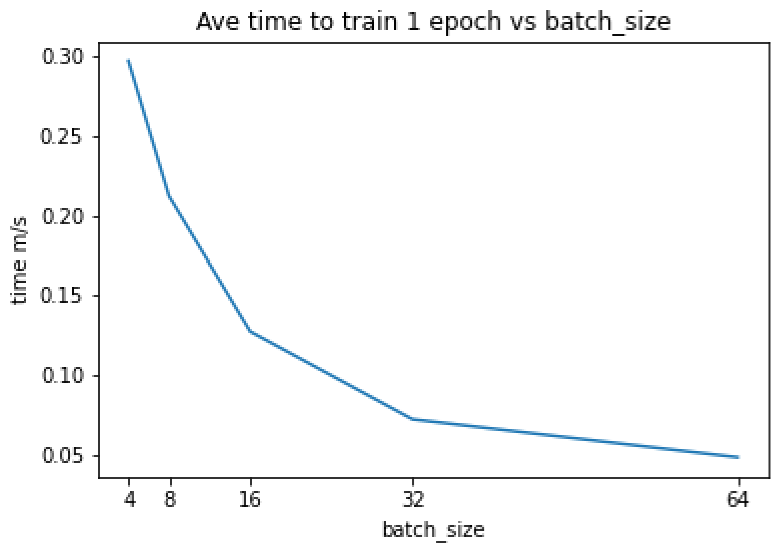


Figure 10. Average time taken for training 1 epoch for different batch_sizes

The average time per epoch experiment can be found in the notebook ***PartA/2A/2A-time-per-epoch.ipynb***

b) Select the optimal batch size and states your reasons

To obtain the optimal batch size, we build 5 models for each of the 5 batch-sizes in the search space [4,8,16,32,64] corresponding to the 5 folds of dataset for cross validation. **With early stopping of the training when overfitting starts to happen**, we then take the max validate accuracies of each model and use these to calculate the average accuracies of different batch sizes, as shown in (Table 1). The code for these experiments is available in ***PartA/2A/2a-find-optimal-batch.ipynb***

Table 1. Average test accuracy/cross-validate accuracy over 5 folds of different batch_size (with early stopping)

Batch Size	Kth fold / experiment	Validate Accuracy	Mean validate accuracy / cross-validate accuracy
4	0	0.9027	0.9073
	1	0.9329	

	2	0.8960	
	3	0.9091	
	4	0.8956	
8	0	0.9027	0.9126
	1	0.9430	
	2	0.8926	
	3	0.9057	
	4	0.9192	
16	0	0.8960	0.8999
	1	0.9329	
	2	0.8826	
	3	0.9057	
	4	0.8822	
32	0	0.8993	0.9032
	1	0.9463	
	2	0.8826	
	3	0.9024	
	4	0.8855	
64	0	0.8960	0.8945
	1	0.9228	
	2	0.8691	
	3	0.9024	
	4	0.8822	

A plot of cross-validate accuracies vs batch size is shown in (figure 11). From the (figure 11) and (table 1), **batch size = 8 has the highest accuracy performance** amongst all batch size in the search space. Thus, the optimal batch size is 8.

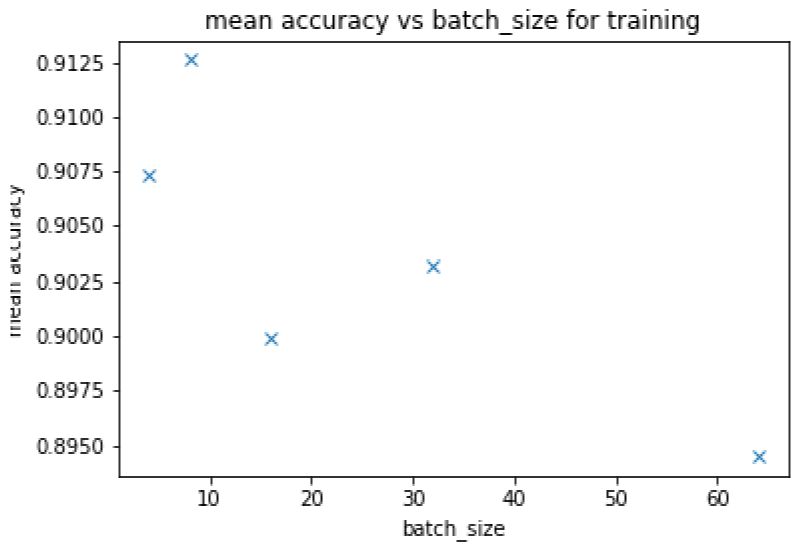


Figure 11. Plot of mean accuracy (over 5 folds) for different batch_size (with early stopping)

In the code notebook, there is also information about the standard deviation of the cross-validate accuracies for each batch size as well. However, we assume that optimal model shall be obtain my maximizing the average accuracy only without any consideration of standard deviation

c) *Plot the train and test accuracies against epochs for the optimal batch size*

The model with batch size = 8 is retrained on the full train set and then tested on the test set (70:30 of the original dataset). The learning curve is obtained as follow in (figure 12):

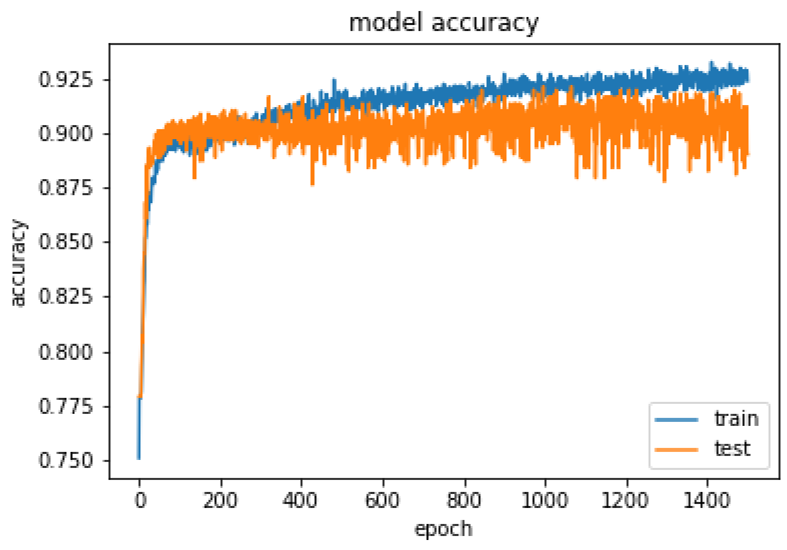


Figure 12. Train and test accuracy vs epoch for optimal batch_size=8

We can see that the test accuracy starts to converge very early (comparing to part 1A train and test accuracy plot), however, the overfitting will also happen earlier and the variations in test accuracy is noisier as we continue training above overfitting threshold point (about 300 epoch). On this end of overfitting, we can simply use early stopping to prevents further overfitting. The max model accuracy is also slightly higher than that in the baseline model (at convergence and before overfitting).

Note that due to stochastic nature of training models, it is recommended to repeat this optimal training model a few times and take the average of model accuracy to plot the curve. However, we do not have time for such luxury here, this will be an improvement area we can work on for the future. **This means that the results above is only specific to our seed and order of experiments we conducted (due to pseudo random generator)**

Code for this part is available in ***PartA/2A/2A-optimal.ipynb***

2.3.3. Finding optimal number of neurons for 3-layer network (Qns A-3)

a) Plot cross-validation accuracies against the number of epochs for number of neurons in the search space [5,10,15,20,25]

Cross-validation accuracy is defined as the mean accuracy of each number of neurons in the hidden layer over the 5 folds. By logging down the training stats with a csv logger, the cross-validation accuracy curves below (figure 13 to 17) are obtained. Code and detail plots are available in the notebook ***PartA/3A/3A-ave.ipynb***

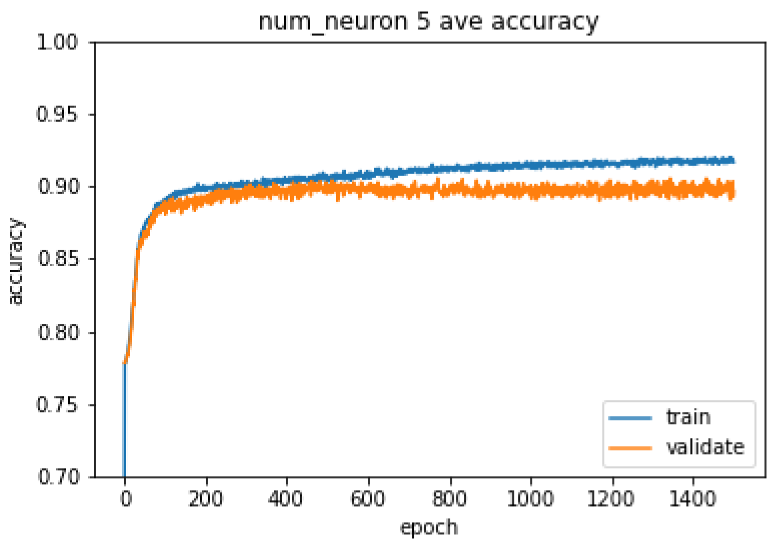


Figure 13. Average train and test/cross-validate accuracy for 5 neurons

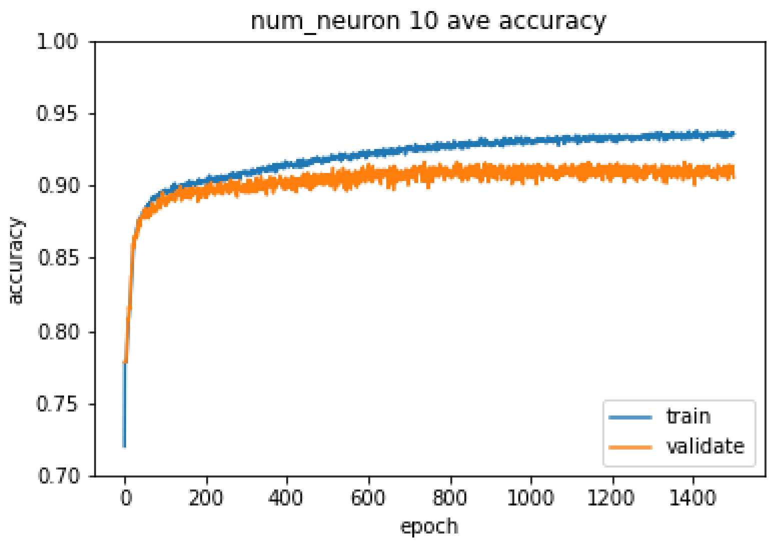


Figure 14. Average train and average test/cross-validate accuracy vs epoch for 10 neurons

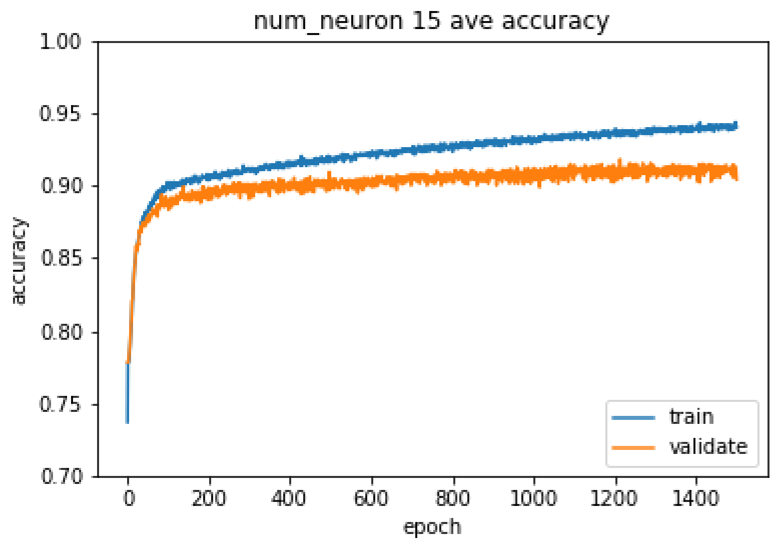


Figure 15. Average train and average test/cross-validate accuracy vs epoch for 15 neurons

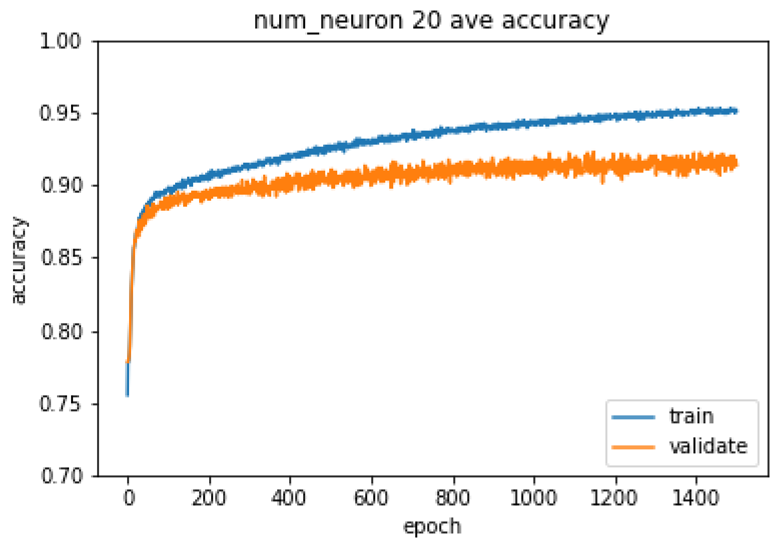


Figure 16. Average train and average test/cross-validate accuracy vs epoch for 20 neurons

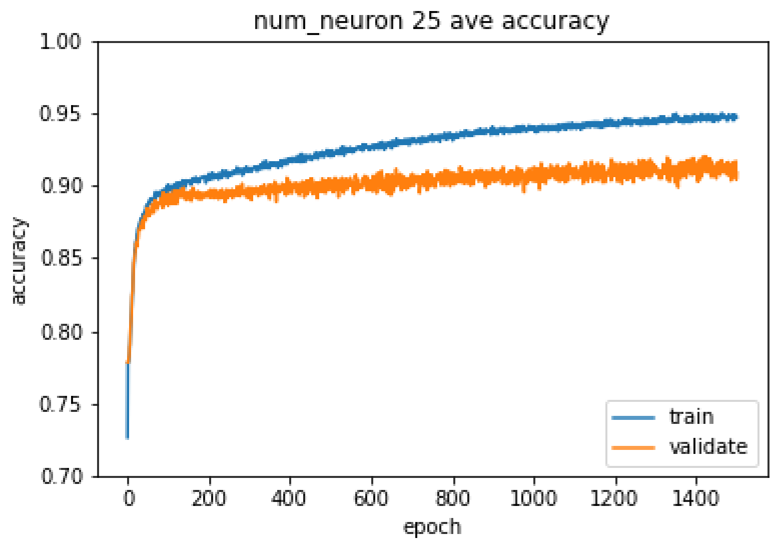


Figure 17. Average train and average test/cross-validate accuracy vs epoch for 25 neurons

Then, plotting only the cross-validation accuracy of these 5 models against each other yields the (figure 18) below:

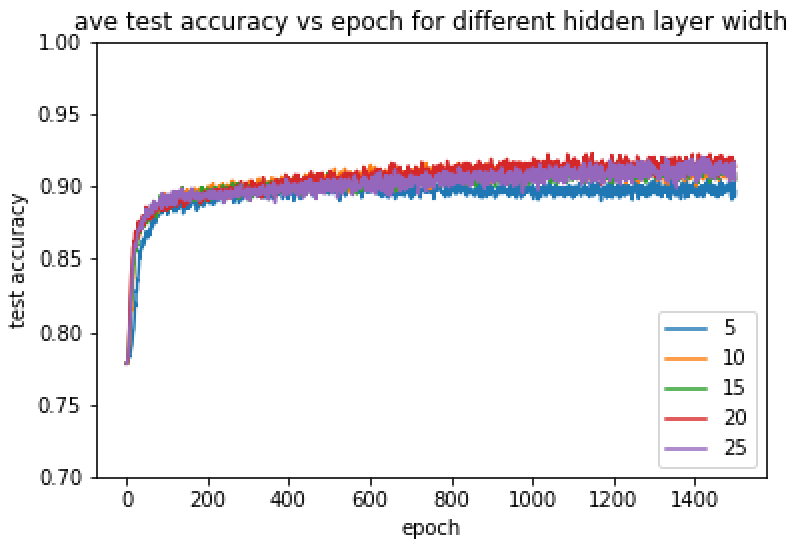


Figure 18. Average test / cross-validate accuracies vs epochs for different number of neurons

There are also validate accuracies vs epochs plot for individual folds of different number neurons available in the code book ***PartA/3A/3A-overfit.ipynb*** *b) Select the optimal number of neurons for the hidden layer. State the rationale*

To obtain the optimal number of neurons of the hidden layer for this 3-layer network, we build 5 models for each of the 5 number of neurons in the search space [5,10,15,20,25] corresponding to the 5 folds of dataset for cross validation. **With early stopping of the training when overfitting starts to happen**, we then take the max validate accuracies of each model and use these to calculate the average accuracies of different number of neurons (also known as the cross-validation accuracy), as shown in Table 2. The code for these experiments is available in the notebook ***PartA/3A/3A-find-optimal-neuron.ipynb***

Table 2. Average test accuracy/cross-validate accuracy over 5 folds of different number of neurons in hidden layer (with early stopping)

Num of neurons	Kth fold / experiment	Validate Accuracy	Mean validate accuracy / cross-validate accuracy
5	0	0.9060	0.9066
	1	0.9362	
	2	0.8926	
	3	0.9158	
	4	0.8822	
10	0	0.9027	0.9126
	1	0.9430	
	2	0.8926	
	3	0.9057	
	4	0.9192	
15	0	0.8993	0.9052
	1	0.9396	
	2	0.8893	
	3	0.9024	
	4	0.8956	
20	0	0.9161	0.9066
	1	0.9329	
	2	0.8859	
	3	0.9057	

	4	0.8923	
25	0	0.8960	0.9039
	1	0.9396	
	2	0.8826	
	3	0.9091	
	4	0.8923	

A plot of cross-validate accuracies vs number of neurons in hidden layer is then plot and is shown in (figure 19). From the (figure 19) and the (table 2), **number of neurons = 10 has the highest accuracy performance** amongst all number of neurons in the search space. Thus, the optimal number of neurons is 10 for the hidden layer

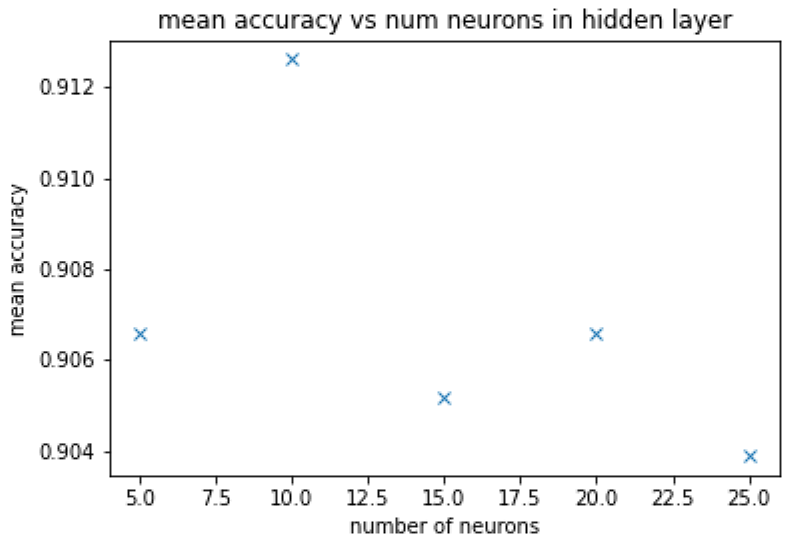


Figure 19. Plot of mean accuracy/cross-validate accuracy of different number of neurons in hidden layers (with early stopping)

In the code notebook, there is also information about the standard deviation of the cross-validate accuracies for each number of neurons of hidden layer as well. However, we assume that optimal model shall be obtain my maximizing the average accuracy only without any consideration of standard deviation. Hence the above plot is not a box plot.

c) Plot the train and test accuracies against epochs with the optimal number of neurons

The optimal number of neurons for hidden layer model will have similar plot (figure 20) to part c of 2.3.2 (where batch size = 8 and number of neurons = 10)

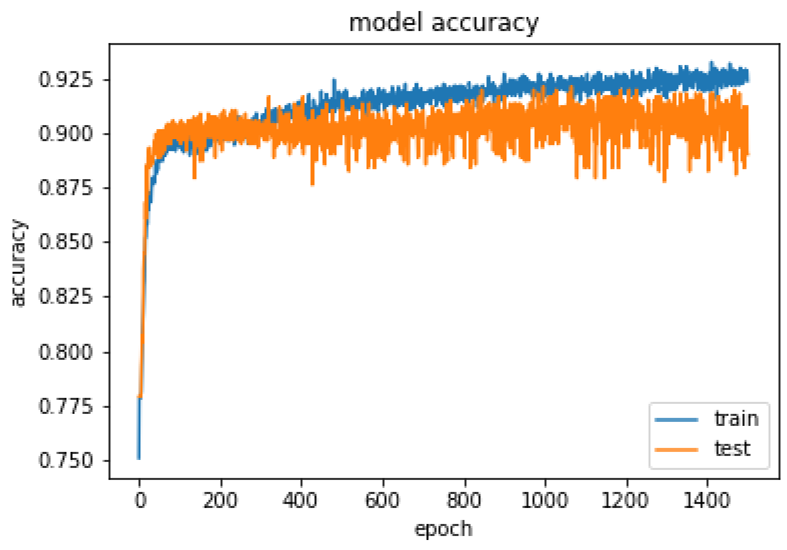


Figure 20. train and test accuracy of the optimal model (batch_size=8, 10 neurons)

Note that due to stochastic nature of training models, it is recommended to repeat this optimal training model a few times and take the average of model accuracy to plot the curve. However, we do not have time for such luxury here, this will be an improvement area we can work on for the future. **This means that the results above is only specific to our seed and order of experiments we conducted (due to pseudo random generator)**

The code for this part c is available in the notebook ***PartA/3A/3A-optimal.ipynb***

2.3.4. Finding optimal decay parameter (L2 regularization) for the 3-layer network (Qns A-4)

a) Plot the cross-validation accuracies against the number of epochs for the 3-layer network with decay parameter in the search space [0,1e-3,1e-6,1e-9,1e-12]

Cross-validation accuracy is defined as the mean accuracy of each beta values over the 5 folds. By logging down the training stats with a csv logger, the cross-validation accuracy curves below (figure 21 to 25) are obtained. Code for this averaging procedure and plotting is available in the notebook ***PartA/4A/4A-ave.ipynb***

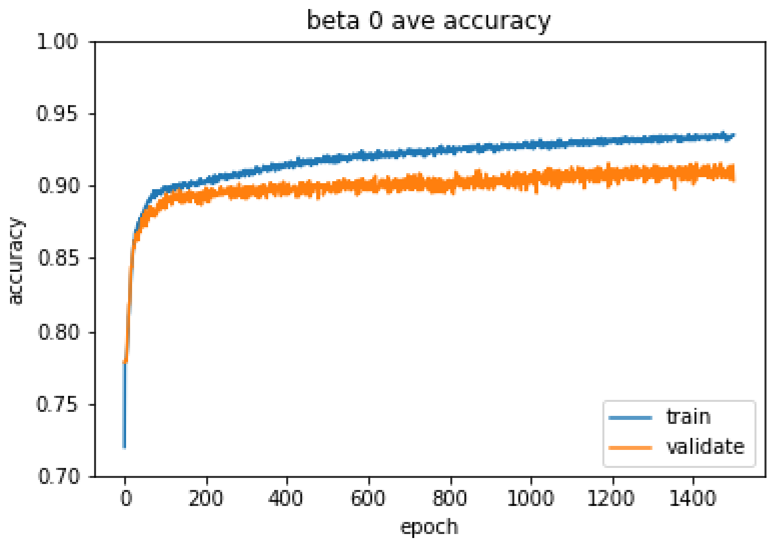


Figure 21. Average Train and average test (cross-validate) accuracies vs epochs for beta = 0

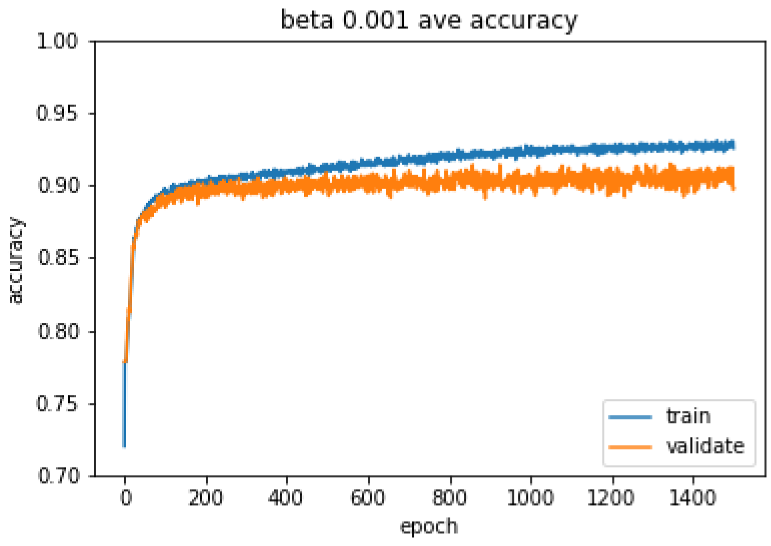


Figure 22. Average Train and average test (cross-validate) accuracies vs epochs for beta = 1e-3

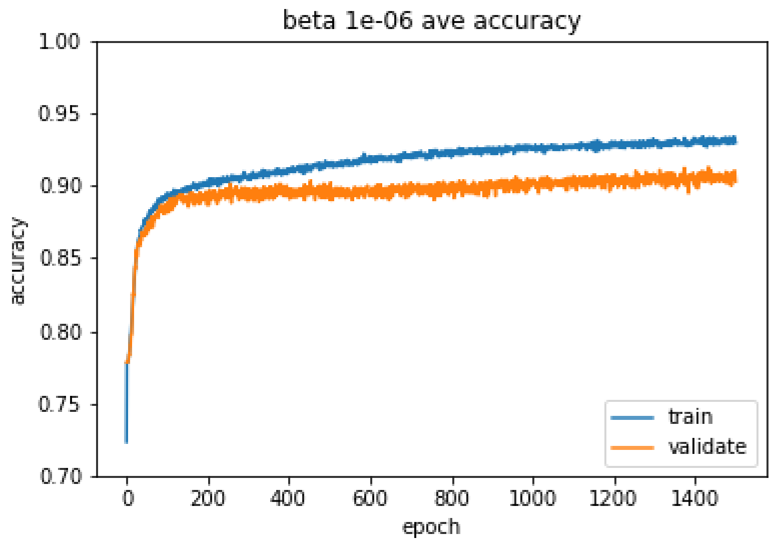


Figure 23.. Average Train and average test (cross-validate) accuracies vs epochs for beta = 1e-6

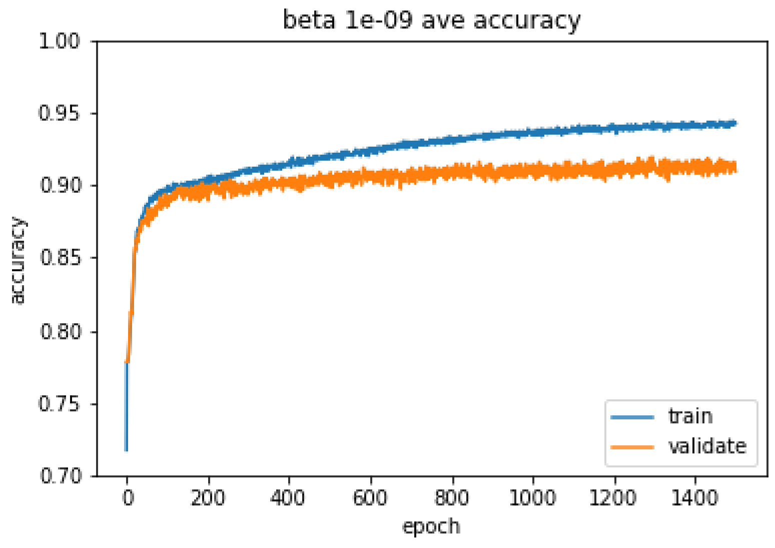


Figure 24. Average Train and average test (cross-validate) accuracies vs epochs for beta = 1e-9

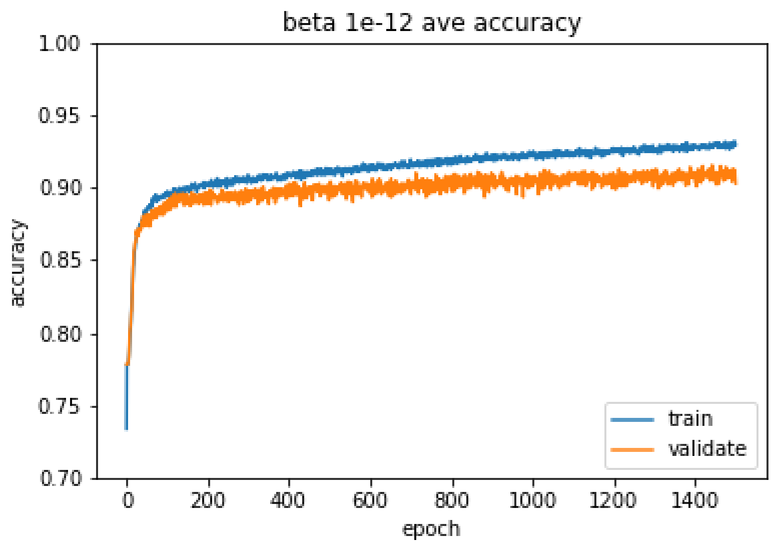


Figure 25 . Average Train and average test (cross-validate) accuracies vs epochs for beta = 1e-12

Then, plotting only the cross-validation accuracy of these 5 models against each other yields the (figure 26) below:

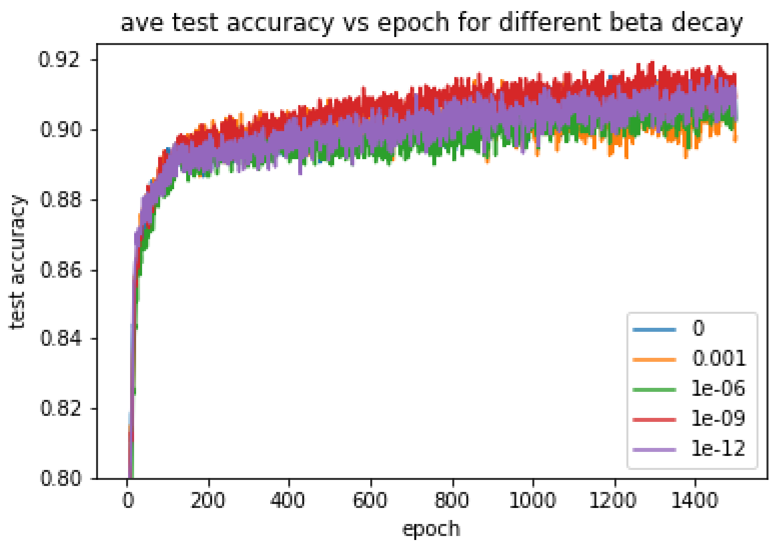


Figure 26. Average cross-validation accuracy of 5 different beta decay parameters

There are also validate accuracies vs epochs plot for individual folds of different number neurons available in the notebook ***PartA/4A/4A-overfit.ipynb***

b) Select the optimal decay parameter. Explain

To obtain the optimal decay parameter beta, we build 5 models for each of the 5 beta values in the search space [0,1e-3,1e-6,1e-9,1e-12] corresponding to the 5 folds of dataset for cross validation. **With early stopping of the training when overfitting starts to happen**, we then take the max validate accuracies of each model and use these to calculate the average accuracies of different batch sizes, as shown in Table 3. The code for these experiments is available at the notebook ***PartA/4A/4A-find-optimal-beta.ipynb***

Table 3. Average test accuracy/cross-validate accuracy over 5 folds of different beta decay parameters (with early stopping)

Beta Decay	Kth fold / experiment	Validate Accuracy	Mean validate accuracy / cross-validate accuracy
0	0	0.9060	0.9066
	1	0.9329	
	2	0.8926	
	3	0.9057	
	4	0.8956	
1e-3	0	0.9027	0.9086
	1	0.9463	
	2	0.8893	
	3	0.9057	
	4	0.8990	
1e-6	0	0.8960	0.9025
	1	0.9362	
	2	0.8893	
	3	0.9057	
	4	0.8855	
1e-9	0	0.9094	0.9059
	1	0.9396	
	2	0.8859	
	3	0.9091	
	4	0.8855	

1e-12	0	0.8993	0.9019
	1	0.9329	
	2	0.8859	
	3	0.9024	
	4	0.8889	

A plot of cross-validate accuracies vs beta value is then plotted and is shown in (figure 27). From the (figure 27) and (table 3), **beta = 1e-3 has the highest accuracy performance** amongst all beta values in the search space. Thus, the optimal beta decay parameters for L2 regularization is 1e-3

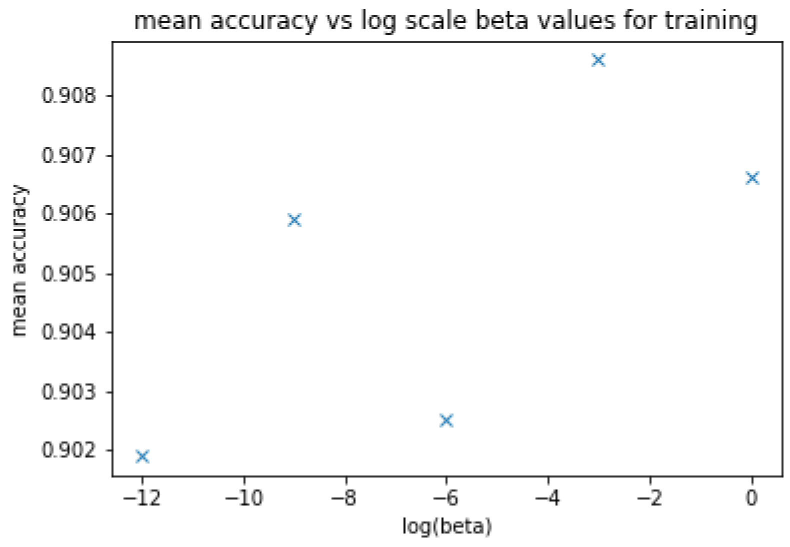


Figure 27. Cross-validation accuracy of different beta values (with early stopping)

In the code notebook, there is also information about the standard deviation of the cross-validate accuracies for each beta values. However, we assume that optimal model shall be obtain my maximizing the average accuracy only without any consideration of standard deviation. Hence the above plot is not a box plot.

c) Plot the train and test accuracies against epochs for the optimal decay parameter

The model with batch size = 8, number of neurons = 10 and optimal beta decay = 1e-3 is retrained on the full train set and then tested on the test set (70:30 of the original dataset). The learning curve is obtained in (figure 28) as follow:

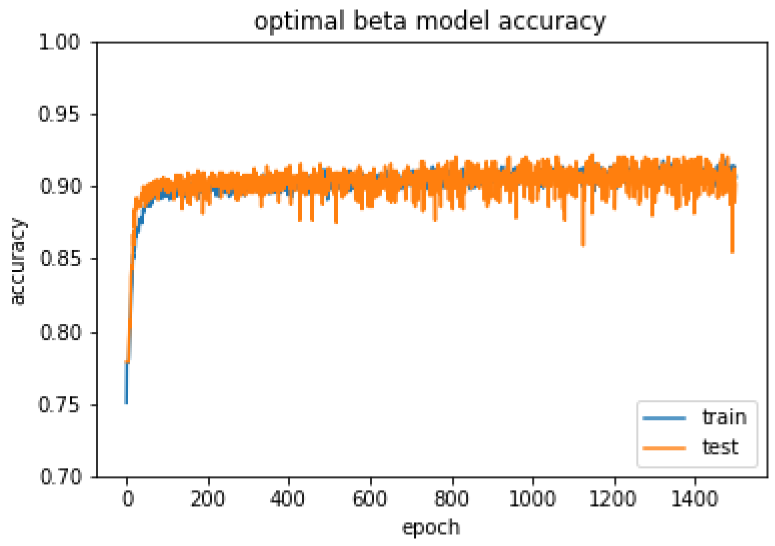


Figure 28. Train and test accuracies vs epochs for the optimal decay parameter

We can see that the overfitting as we train the model to larger epochs have been reduced with the optimal value for beta decay of L2 regularization. However, the datapoints on the test curve is still noisy. On the bright side, the convergence of test and train curves happen much earlier than comparing to the baseline model or the optimal batch size + optimal number of neuron model (in figure 20). This shows the power of L2 regularization in lifting performance, both by accuracy and by overfitting virtue.

Note that due to stochastic nature of training models, it is recommended to repeat this optimal training model a few times and take the average of model accuracy to plot the curve. However, we do not have time for such luxury here, this will be an improvement area we can work on for the future. **This means that the results above is only specific to our seed and order of experiments we conducted (due to pseudo random generator)**

The code for this part c is available in the notebook ***PartA/4A/4A-optimal.ipynb***

2.3.5. Modifying the architecture by adding more hidden layers (Qns A-5)

a) Plot the train and test accuracy vs epochs of the 4-layer network

The 4 layers neural network has the following hyperparameter specifications: 2 hidden layers, each of 10 neurons, batch size is 32 for training and L2 regularization beta decay = 1e-6.

We let training to continue even when overfitting happened, the plot of train and test accuracy against the number of epochs is shown below in figure 29.

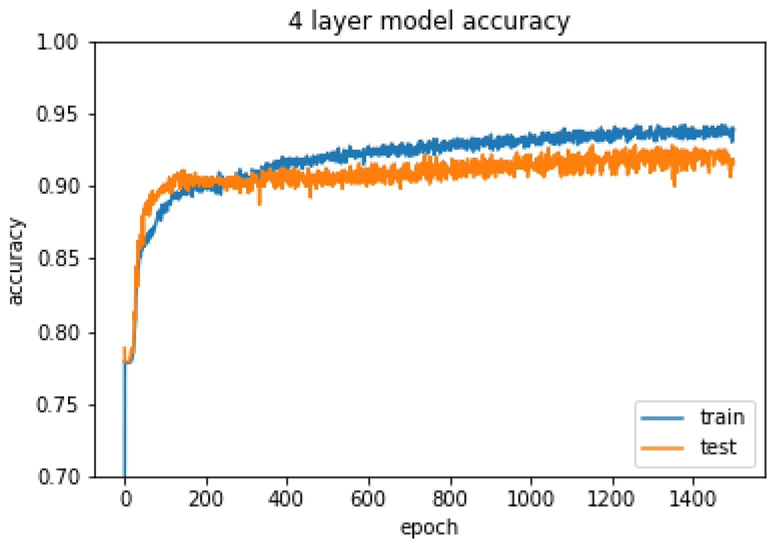


Figure 29. Train and test accuracy of 4-layer model (with overfitting)

The code for this part is available in the notebook ***PartA/5A/5A.ipynb***

b) Compare and comment on the performances of the optimal 3 layers and this model.

We carried out 5-fold cross evaluation procedure for this new 4-layer MLP. The cross-validation accuracy of this 4-layer model is obtained in similar fashion as shown in qns A-2, A-3 and A-4. We obtained the statistics of the optimal 3-layer from the previous part 2.3.4b where 1 hidden layer, beta = 1e-3, batch size is 8 and number of neurons in the hidden layer is 10. Combining these information yields the (table 4). The code for these experiments is available in the notebook ***PartA/5A/5A-find-optimal-model.ipynb***

Table 4. Cross-validation accuracy of models (with overfitting in 5 folds)

Architecture	Kth fold/ experiments	Validate accuracy	Cross-validation accuracy (Mean of validate accuracy)
Optimal 3 layers	0	0.9027	0.9086
	1	0.9463	
	2	0.8893	
	3	0.9057	
	4	0.8990	
4 layers	0	0.8993	0.9073
	1	0.9396	
	2	0.8859	
	3	0.9158	
	4	0.8956	

A plot of cross-validate accuracies for the optimal 3 layers and the 4 layers are shown below (figure 30). From the (figure 30) and table 4, the optimal 3 layers has higher cross-validation accuracy performance.

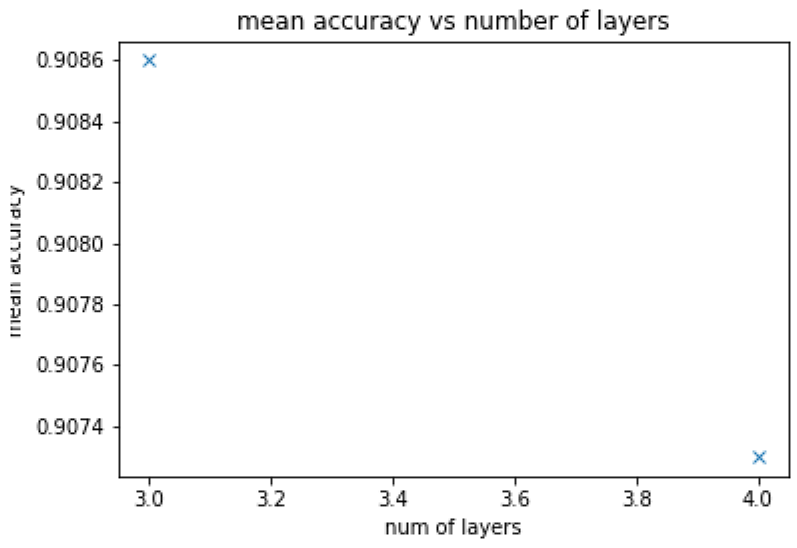


Figure 30. Mean accuracy/cross-validate accuracy of optimal 3-layer and 4-layer (with early stopping)

As the **mean accuracy (cross-validation accuracy) of the 4 layers model is lower than that of the optimal 3 layers model**, the optimal 3 layers is better than that of the 4 layers.

It is also noted that in (figure 28) of part 2.3.4.c that the plot of accuracy (train and test) for the optimal 3 layers model also shown little or no overfitting while the 4 layers model has overfitting if we continue training on (figure 29). In additionally, the time taken to converges for the test accuracy of the optimal 3 layers model (figure 28) is also shorter that of the 4 layers (figure 29). **Thus, it is deducible that the optimal 3 layers is also better due to less overfitting and faster convergence speed.**

Note that due to stochastic nature of training models, it is recommended to repeat this optimal training model a few times and take the average of model accuracy to plot the curve. However, we do not have time for such luxury here, this will be an improvement area we can work on for the future. **This means that the results above is only specific to our seed and order of experiments we conducted (due to pseudo random generator)**

2.4. Summary and Discussion

In conclusion, in this part A, we attempt to build a baseline MLP model with 3 layers to classify the data records of the Cardiotocography dataset. While this model performs generally well, we were interested in exploring how to improve the performance of the model with our choice of hyper-parameters and model structure.

We then experiment around with finding the optimal batch size in training the mode (=8), the width of the hidden layer (optimal is 10 neurons) and the optimal beta decay parameter for L2 regularization (=1e-3).

This optimal model is then compared to a modified architecture where we are extending the network depth to 4 layers (with 2 hidden layers instead of 1). It is shown that the increase in depth does not lift the performance of the model.

In selecting the best models (be it with optimal hyper-parameter or optimal model architecture), we attempt to use cross-validation techniques with k-fold, the state-of-the-art methods for model selection in machine learning. This process is very time-consuming but yields very reliable results to evaluate and judge the various models that we need to evaluate to decide our hyperparameters and model architecture.

Somethings that we can improve further, given more time or better computing resource is to address the stochastic nature of training models more substantially. This can be done by bagging the k fold (repeating each fold by n times and takes their average as the values for the fold experiments, on top of just doing 1 time per fold and take the average of k fold). This approach of average over average surely can give a better estimate of our model for selection, but it also requires more time and resource in computing. In short more iteration used in training or model evaluating is beneficial in selecting the correct hyperparameters and model architecture, thus our results here is somewhat correct, not fully correct unless more effort is channeled into the extra iterations and analysis. The results should be replicable due to the random seeding and specifying random state of many toolkit that we have used (tf, np, etc.). Where applicable, we have applied reshuffled of the dataset to ensure best performance.

We also attempt to use heuristics, early stopping to prevent models from overfitting during k-fold experiments. However, the choice of parameters (min_delta and patience step) for this early stopping algorithm is not yet justifiable or studied closely (based on the noisiness of the training curve, validate curve, etc). Given more time, this shall be evaluated to justify our choice better.

Lastly, a big assumption that we are basing on is the evaluation of model in selection is only based on the average testing error or average accuracy performance of the model (across 5 folds). This is for simplicity sake of school project, however, in real life, we would need to consider the standard deviation of the accuracy performance of all the folds as well. In additionally, we might want to consider the overfitting of the model in model selection as well, however that will make our problem even more complex and unable to be delivered in such a short time frame.

3. Part B

3.1. Introduction of problem

In part B, we are given the Graduate Admissions Prediction dataset with the task to design a multilayer perceptron networks to predict by regression the chances of being admitted to the Master program.

Each data records consists of 9 features. The first one is a serial number and should be ignored. The next 7 features are the input attributes used in training and the last feature is the label, which is the probability of getting an admission to the program. Since the label is continuous, this will be a regression problem.

Besides building a baseline model as mentioned in the methodology below, we will be experimenting with RFE feature selection to remove some features from the data records and build models from this new dataset to see the lift in performance of training. We also experiment with adding more layers to the network and making use of drop out regularization to observe the improvement in performances of the model.

3.2. Methodology

3.2.1. Data preprocessing

Serial number column (the left most column in the original dataset) is first removed since this feature has no meaning or significant in our model building.

Then, input standardization is conducted. This is done using the StandardScaler class of sklearn library, which essentially turn the features (except for label, which is the last columns in the dataset) into features having mean of 0 and standard deviation of 1. We carried out such standardization as the features included in this dataset are all assumed to be of Gaussian distribution in natural population. This is often the case for scoring of candidates in an examination, their CGPA, etc.

Then, data is spilt into 2 sets with ratio 70:30 (train to test). The train set will be used in training while the test set will act like the hold out, being used in evaluating the models. **It is assumed here that cross validation is not necessary since the question is not openly asking for it.** However, in practice, to have a more robust evaluation of the models, we are highly recommended to apply 3-way data split and cross evaluation with k-fold as shown in part A. This is the golden standard in machine learning practice for model evaluation.

Like part A, to ensure no data leakage of test set to training set, we shall apply the standardization based on training dataset's statistics (mean and std) on both the training set and the test set. This is to ensure no information from the test set is avail to the model once training happens (the statistics of test set is not contributed into the train set due to fitting only on train set).

3.2.2. Feature Selection with RFE

Feature selection is another crucial step in data preparation for machine learning. It can be used to speed up the modelling process and improving the model's performance to a problem. In our case, we are looking to do feature selection with Recursive Feature Elimination to find the optimal number of features and which features are more important in training our MLP network.

In RFE, we are to **remove the least important features** one by one until the optimal or target number of features in the dataset is obtained. There are many ways to judge the importance of features such as by **using another machine learning algorithm to rank them** (Logistic Regression, Ridge Regression…) and remove accordingly. In our experiments, we implement a **brute force algorithm** where we remove one by one a feature from the dataset, train the model on train set and evaluate the test m.s.e on test set, keeping the model with the lowest m.s.e (that is to remove those models with higher m.s.e). We continue doing so for the second, third, and so on features to be removed until the target number of features (6 and 5 in our case) is obtained. **The features that their removals caused the most drop in m.s.e (and so better improvement in performance of the model) is to be removed**.

Due to the heuristic nature of the algorithm and the stochastic nature of training models, we will **repeat the training of models for 5 times for each feature being removed, then take the average test m.s.e as the represented m.s.e** for judging whether to remove that feature or not. This ensure a more consistent performance level for models and can help us decide with more confidence.

The detailed results of this approach are shown in 3.3.2.a

3.2.3. Model implementation and evaluation

Like part A, models in part B are built and trained with the keras library which helps to abstract the linear algebra behinds the models and speed up developing and testing of models.

While cross validation is not implemented in this part B, we try to make up for that with **5 experiments repetition strategy**. This means each of the models to be evaluated shall be trained 5 times with an early stopping callback (to prevent overfitting). The **minimum m.s.e** of these models in each experiment are recorded and the **mean** will be used for comparisons of their performances.

At the same time, we also carry out training with overfitting by allowing training to run until 1500 epochs. This is only done once per model and only used to show the overall shape of overfitting or the behavior of m.s.e metrics during training which is easier to work with.

A variety of plot, such as learning curve and average test m.s.e for each model will be used to visualize and rationalize the decisions after each experiment. There will also be table of values to substantiate the answers where needed. Otherwise, the jupyter notebook or the annex contains information to support if need be. **Details on this will be mention in the experiments and results discussion.**

For simplicity sake, we shall take test m.s.e as the main metrics for comparisons. However, we also attempt to quantify the models based on overfitting, noises, etc. as well. The end decisions are to be formalized with test m.s.e only.

3.2.4. Dropout regularization

Drop out regularization is implemented by the keras.layers.Dropout class in our case. This is applied to both input layers and all hidden layers. **Dropout can significantly reduce overfitting of our model** in training by forcing our models to learn and make use of all features, inputs from the dataset instead of just memorizing some of them (over-reliance on a small subset). This in turn can makes the model to become more generalized to the problem too.

3.2.5. Codebase organization

For each question, there might be multiple jupyter notebooks available to answer the various aspects of the question. While the results of these notebook have been pre-obtained, in order to replicate these results to answer the question, please follow the annex B, which explain what each of the notebooks in part B folders do and which order to run them.

3.3. Experiments and Results

3.3.1. Baseline 3-layer feedforward model (Qns B-1)

Code for this question is available in the notebook ***1B.ipynb*** of folder ***PartB/1B*** of the submitted code

a) Train the model on train set and plot the train and test errors against epochs.

The architecture of this baseline, 3-layer MLP is as following:

- 10 neurons hidden layer with ReLU activation, L2 regularization with beta = 1e-3
- Output layer of linear activation
- Learning with SGD and learning rate alpha = 1e-3
- Input size is 7
- Label is the chance of admitted the master program.

We trained the model on the cleaned and standardized dataset **with early stopping callbacks and condition where test m.s.e is minimum for delta = 1e-5 and patience = 10 steps**. The following learning curve of m.s.e versus epochs is obtained (figure 31).

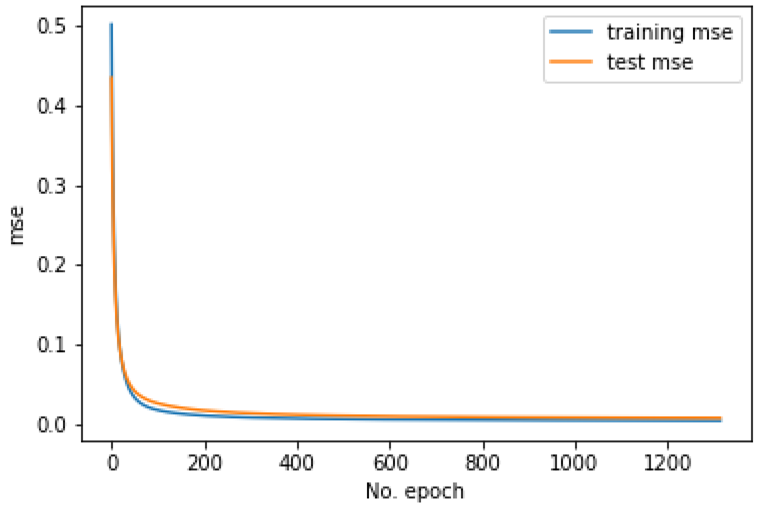


Figure 31. train and test m.s.e vs epoch for baseline 3-layer model

The test m.s.e is slightly above the training m.s.e in all epochs, suggesting there is overfitting. We shall address this problem in the next few experiments.

b) State the approximate number of epochs where the test error is minimum and use it to stop training

From the (figure 31) above, we can see that our model is early stopped at **about 1300 epochs.** The choice of s**mall patience steps** (10) should be sufficient because the learning curve looks very smooth.

c) Plot the predicted values and the target values for any 50 data points in the test set.

We first shuffle the index of the test set (30% of initial dataset) and used the model to predict the values. A plot of the target and predicted values based on the baseline model is shown in (figure 32).

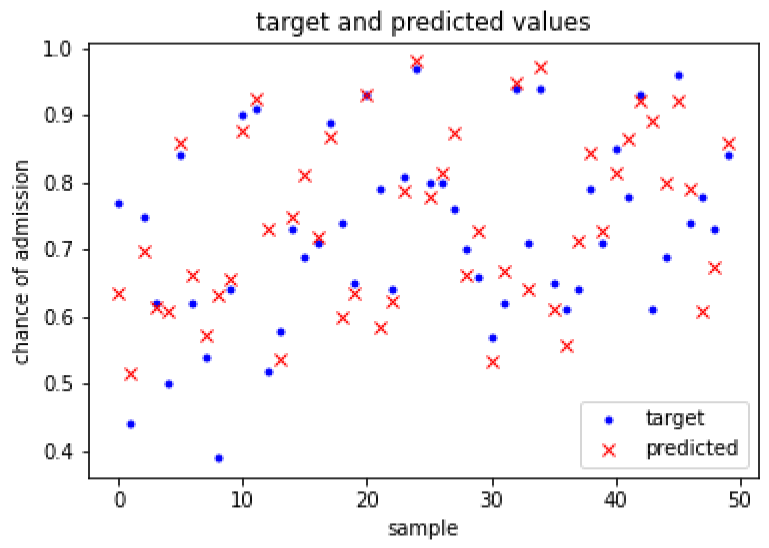


Figure 32. Target and predicted values (baseline 3-layer MLP)

We can notice that some of the target values (blue dot) and predicted values (red cross) are very close, however, many of them are not very well fitted together (far from each others). We shall attempt to close their distance (lifting the performance of the model) in the next part with RFE and with different model architecture.

3.3.2. Applying RFE to the baseline model (Qns B-2)

a) RFE feature removal

6 features

The brute force RFE approach for selecting 6 out of 7 features, leads to the following results for the 5 experiments, shown in (table 5) below.

Table 5. Average test m.s.e (of 5 experiments) for each feature removed (in order to get 6 features set)

Feature removed	Ave test m.s.e (amongst 5 experiments)
GRE	0.00700
TOEFL	0.00740
University Ranking	0.00664
SOP	0.00676
LOR	0.00742
CGPA	0.00763
Research	0.00641

The results for each of the 5 experiments and for each features removed is shown in notebook ***2B-brute-force-rfe.ipynb*** in folder ***PartB/2B*** of the code submitted.

The plot of ave test m.s.e and the feature removed is also shown in (figure 33).

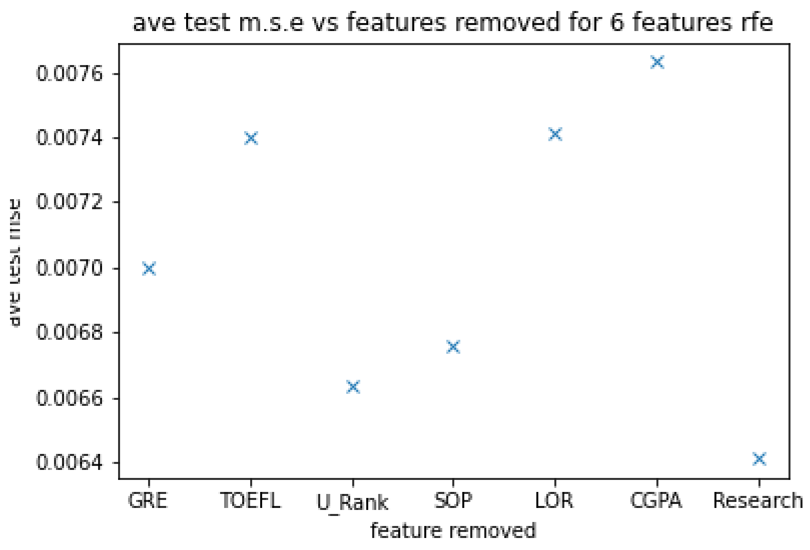


Figure 33. Average test m.s.e for each feature removed to form 6 features dataset

From the (figure 33) and the (table 5) above, removing "Research" give the best lift in performance. This is intuitive as for master candidates, most of them are either fresh graduates or employees who wish to upgrade their skills, thus most should not be judged based on their research as most would not be opened for research experience before.

5 features

We then remove more features from the 6 features dataset (with "Research" removed). The average results across 5 experiments is shown in the (table 6).

Table 6. Average test m.s.e (of 5 experiments) for each feature removed (in order to get 5 features set)

Feature removed	Ave test m.s.e (amongst 5 experiments)
GRE	0.00667
TOEFL	0.00682
University Ranking	0.00678
SOP	0.00632
LOR	0.00717
CGPA	0.00840

The results for each of the 5 experiments and for each features removed is shown in notebook ***2B-brute-force-rfe.ipynb*** in folder ***PartB/2B*** of the code submitted.

The plot of ave test m.s.e and the feature removed is shown in figure 34.

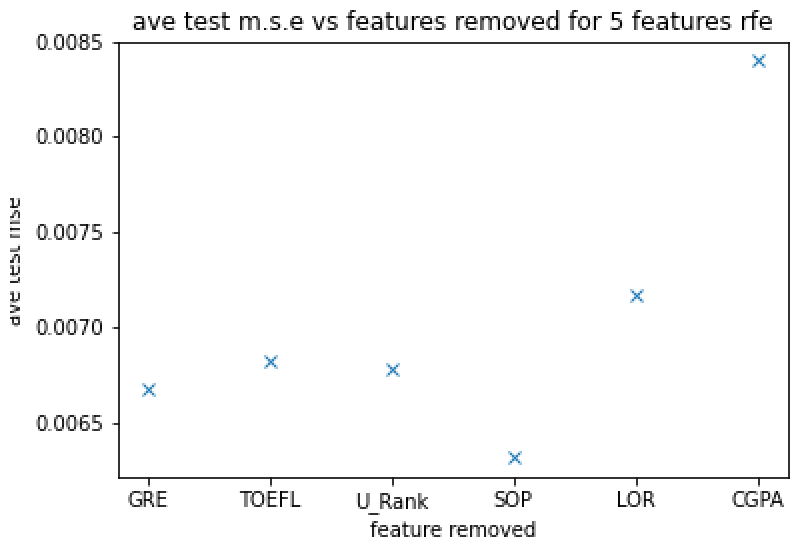


Figure 34. Average test m.s.e for each feature removed to form 5 features dataset

From the figure 34 and (table 6) above, it is obvious that the "SOP" feature gives rise to the best improvement in M.S.E, thus it should be removed to form the 5-feature dataset.

This might be the case as the dataset contains the admission probability of master candidates for different university, not just one. This can imply that in certain university and master courses, the performances of candidates (CGPA, GRE, TOEFL or even LOR from employer, professors) is more important that the SOP.

b) Accuracy of the models

Like RFE models, we did 5 experiments where 3 models of 7, 6, 5 features based on the result of a is trained and record their individual test m.s.e. The average test m.s.e across the 5 experiments for these 3 models are shown below in table 7.

Table 7. Average test m.s.e of the full, 6 features and 5 features models with early stopping

Num of Feature	Experiment	Test m.s.e	Ave m.s.e
Full	0	0.00757	0.00716
	1	0.00651	

	2	0.00620	
	3	0.00684	
	4	0.00866	
6	0	0.00764	0.00665
	1	0.00675	
	2	0.00655	
	3	0.00616	
	4	0.00615	
5	0	0.00605	0.00609
	1	0.00624	
	2	0.00594	
	3	0.00599	
	4	0.00622	

The plot of average test m.s.e of each models is shown in (figure 35). The 5-feature model has the lowest average m.s.e, following by the 6-feature model (based on table 7 and figure 35). Therefore, we can see that the performance of the model is lifted with less important features removed from the dataset in training. We will now choose our optimal 3-layer model to be the one with 5 features and use this model and this dataset for next question.

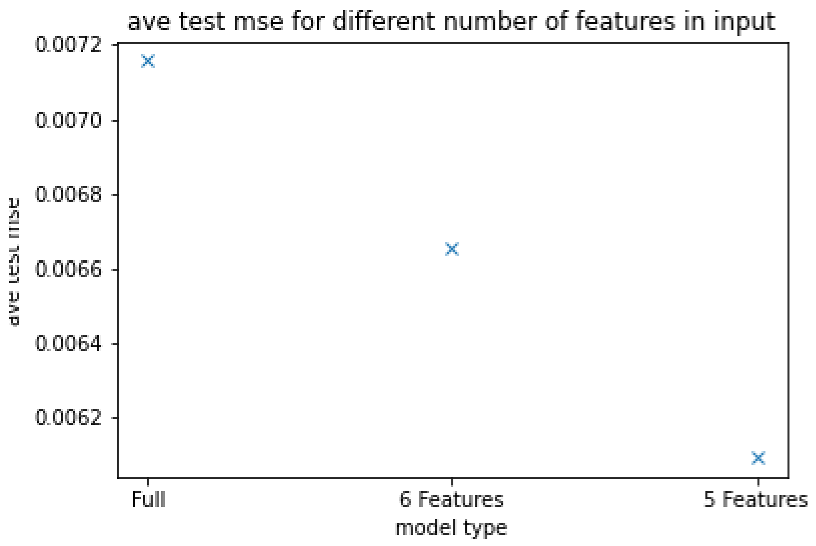


Figure 35. Average test m.s.e for full, 6-feature and 5-feature trained model (with early stopping)

The details result for these 5 experiments are shown in the notebook ***2B-find-optimal.ipynb*** located inside folder ***PartB/2B*** of the submitted code.

Something to add on is that the 5-feature (figure 38) has smaller overfitting than 6-feature (figure 37) model which is less than that of the full 7-feature model (figure 36). (details in ***2B-overfit-curve.ipynb***)

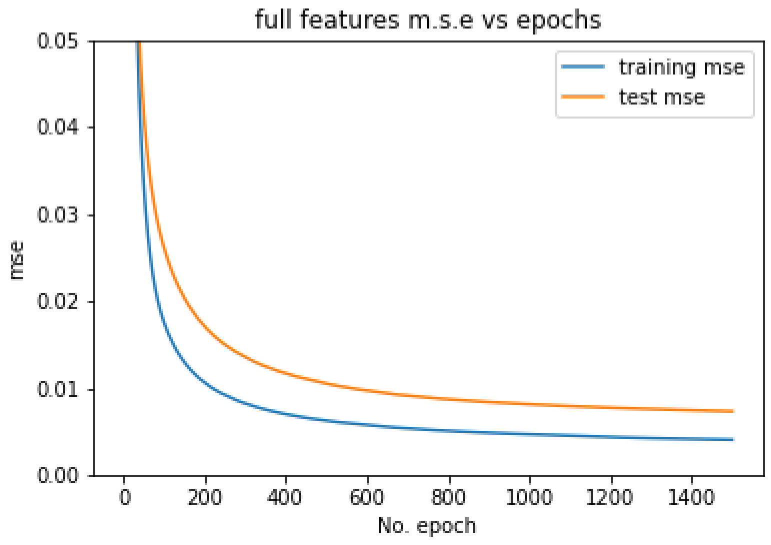


Figure 36. Train and Test m.s.e for model trained with 7 features (with overfitting)

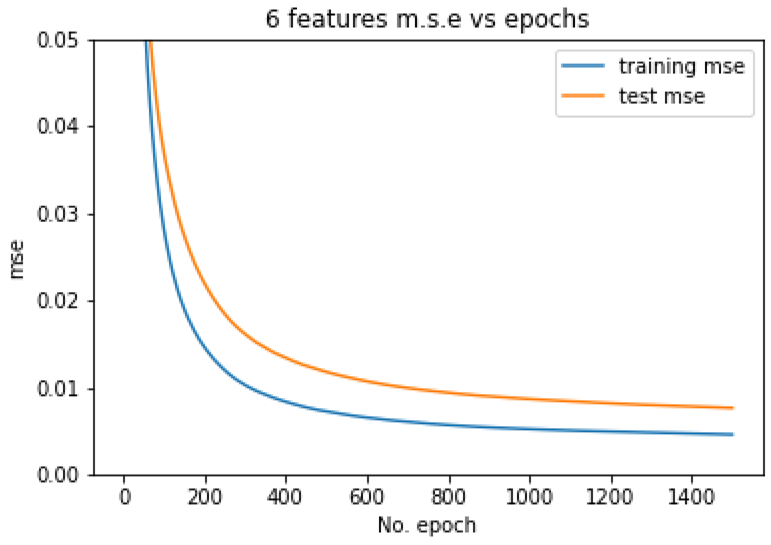


Figure 37. Train and Test m.s.e for model trained with 6 features (with overfitting)

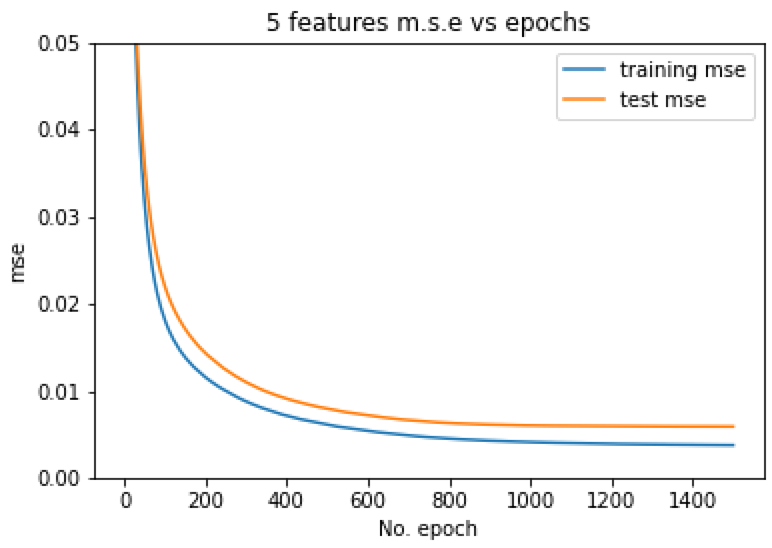


Figure 38. Train and Test m.s.e for model trained with 5 features (with overfitting)

The combined training curve (test m.s.e vs epochs) (figure 40) and the predicted vs target plot of these 3 models are also shown below (figure 39) for completeness. The source code and results for this part is shown in the notebook ***2B-overfit-curve.ipynb*** in folder ***PartB/2B*** of the submitted code.

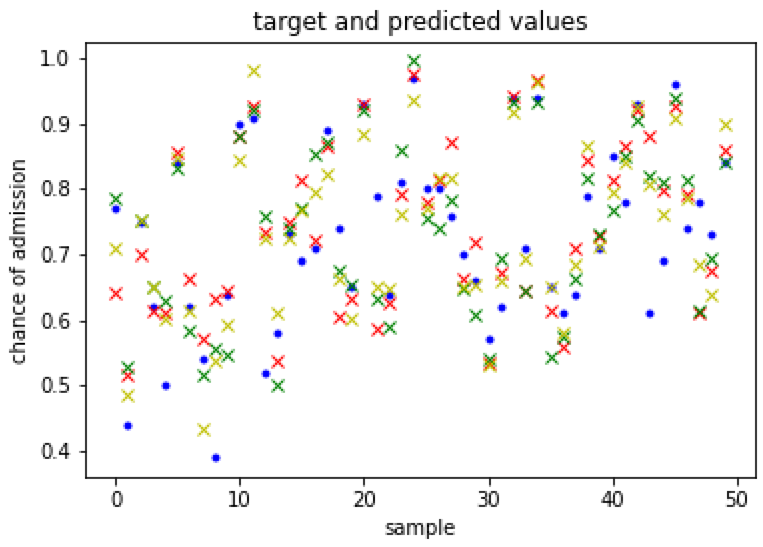


Figure 39. Target and Predicted plot for full, 6-feature and 5-feature trained models (with training no early stopped)

The target are the blue dots. The predicted value for baseline, 7-feature model are red crosses. The predicted values for 6-feature model are green crosses. The predicted values for 5-feature model are yellow crosses.

We can see that the green and yellow crosses in most cases are closer to the blue dot than that of the red dot, which address our problem in 3.3.1c

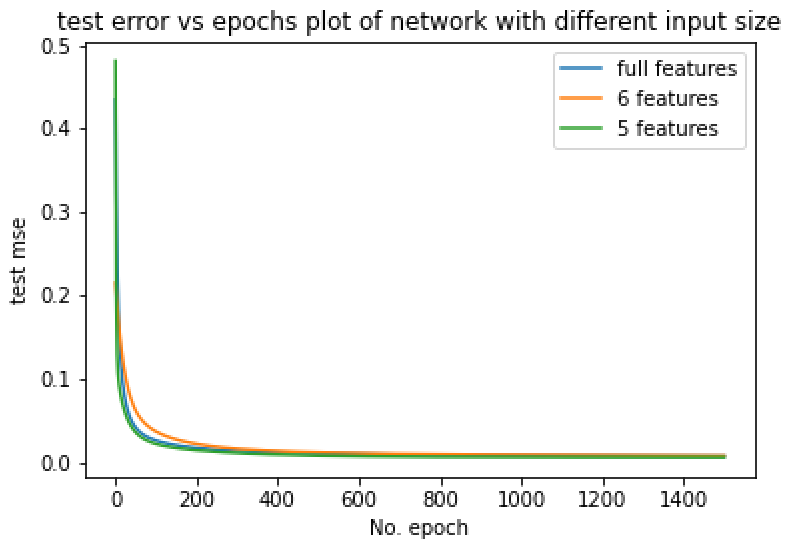


Figure 40. Test error vs epochs for full, 6-features and 5-features models (with overfitting)

The training curve for these 3 models also shows that the 6-feature curve has worse performance than the baseline (longer time to converge and converge at higher mse than baseline) (figure 40). This highlights the stochastic nature of model training and justify our 5 experiments approach earlier on to obtain a reliable result. As a rule of thumb, more experiments mean more accurate results.

3.3.3. Experiment with network depth and drop out regularization (Qns B-3)

Like other parts, 5 experiments for training of the 5 models following were conducted. **In this set of experiments, we did not use early stopping callbacks, reason can be explained from the noisy train and test curve of the dropout models shown in figure 45 and 46 below**. This is to ensure the uniformity of results at 1500 epochs for our evaluation rather than a fake early stopped models due to noise in data points of test m.s.e.

- Model 1: optimal model from 3.3.2 (5 features with "research" and "sop" removed)

- Model 2: 4-layer model with no dropouts
- Model 3: 4-layer model with dropouts at every level, including input layer
- Model 4: 5-layer model with no dropouts
- Model 5: 5-layer model with dropouts at every level, including input layer

The results of these experiments are shown below in Table 8.

Table 8. Results for 5 experiments in finding the optimal model architecture, with lowest average m.s.e (did not use early stopping)

Model	Experiments	Test M.S.E	Ave M.S.E
Optimal 3-layer	0	0.00759	0.00667
	1	0.00572	
	2	0.00619	
	3	0.00740	
	4	0.00643	
4-layer	0	0.00582	0.00584
	1	0.00557	
	2	0.00576	
	3	0.00586	
	4	0.00620	
4-layer + dropout	0	0.00611	0.00609
	1	0.00629	
	2	0.00622	
	3	0.00603	
	4	0.00581	
5-layer	0	0.00559	0.00571
	1	0.00554	
	2	0.00531	
	3	0.00625	
	4	0.00585	
5-layer + dropout	0	0.00705	0.00663
	1	0.00624	

	2	0.00655	
	3	0.00666	
	4	0.00664	

The plot of these average m.s.e from table 8 for different models are also shown in the (figure 41) below.

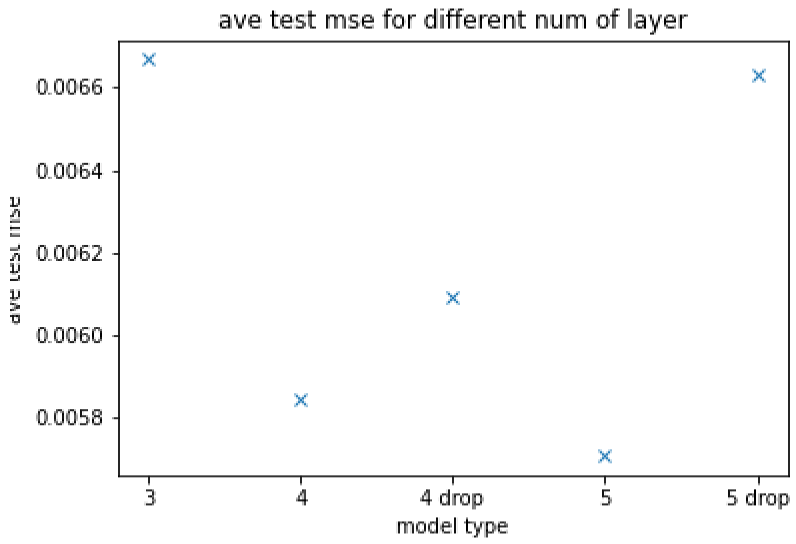


Figure 41. Plot of average test m.s.e (trained untill 1500) for different models

Codes and results for these experiments can be founded in the jupyter notebook ***3B-ave.ipynb*** in the folder ***PartB/3B*** of the submitted code zip file

Beside the 5 experiments above, we also trained the models and let them run with no early stopping (up until epoch=1500) for only once (to observe the training curve), the individual train and test mse and combined test mse vs epochs plots are shown below:

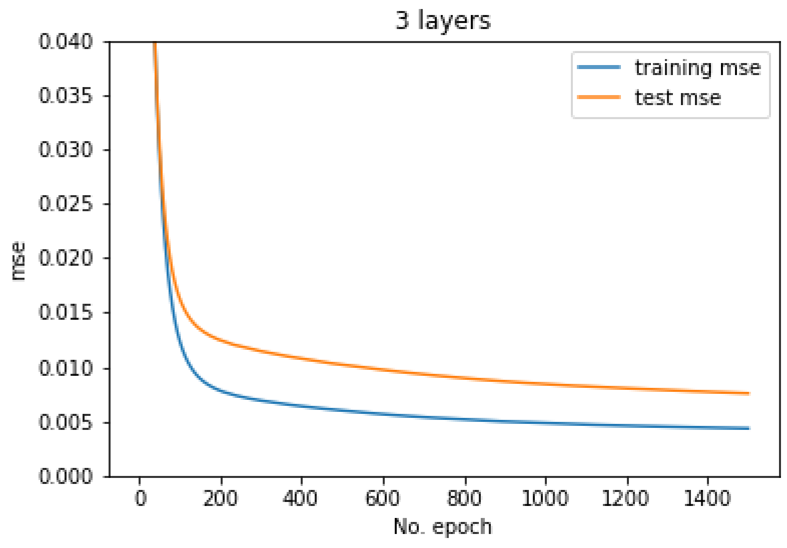


Figure 42. train and test m.s.e vs epochs for optimal 3-layer

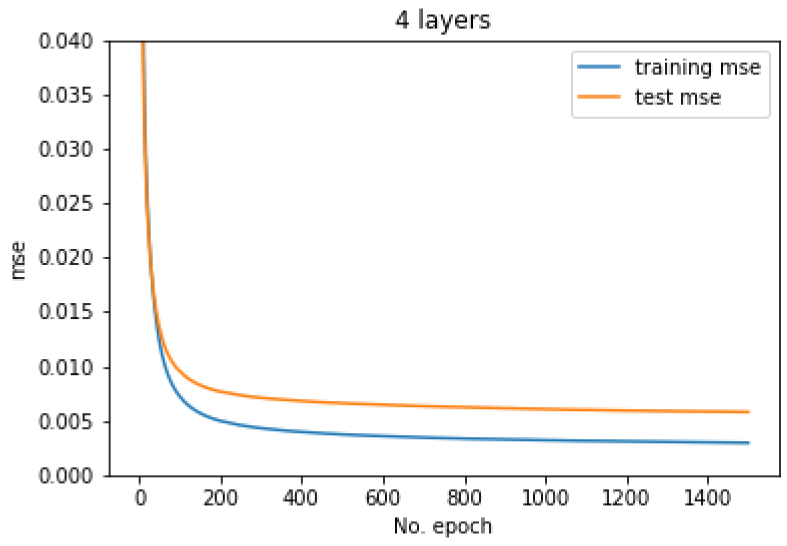


Figure 43. train and test m.s.e vs epochs for 4-layer

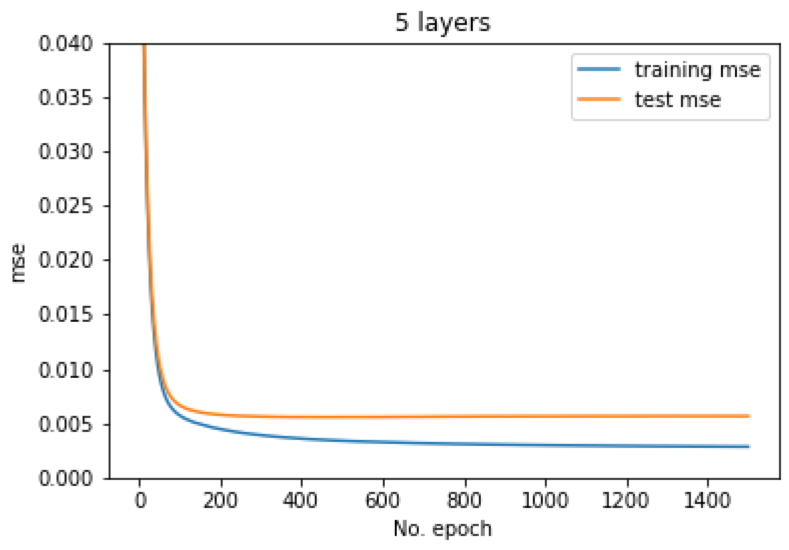


Figure 44. train and test m.s.e vs epochs for 5-layer

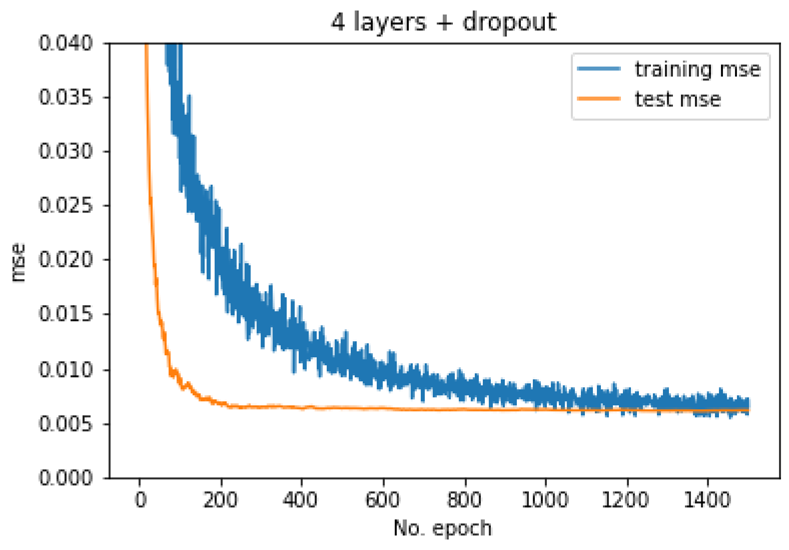


Figure 45. train and test m.s.e vs epochs for 4-layer with dropout regularization

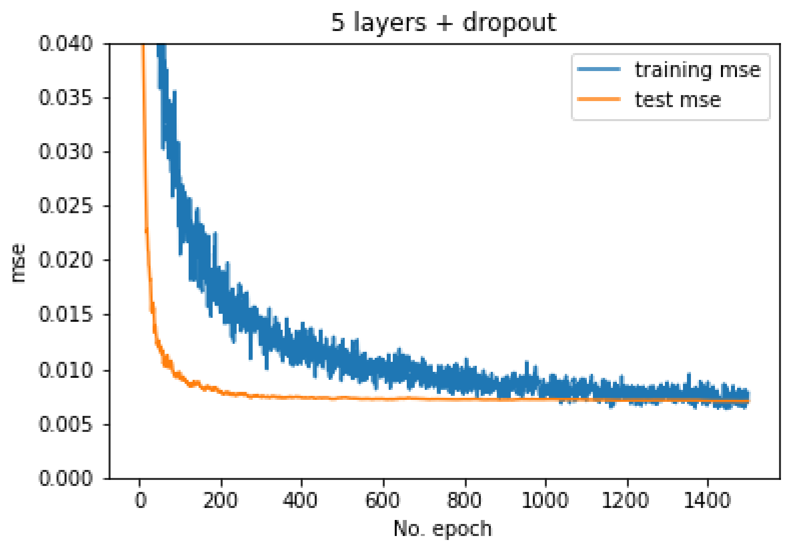


Figure 46. train and test m.s.e vs epochs for 5-layer with dropout regularization

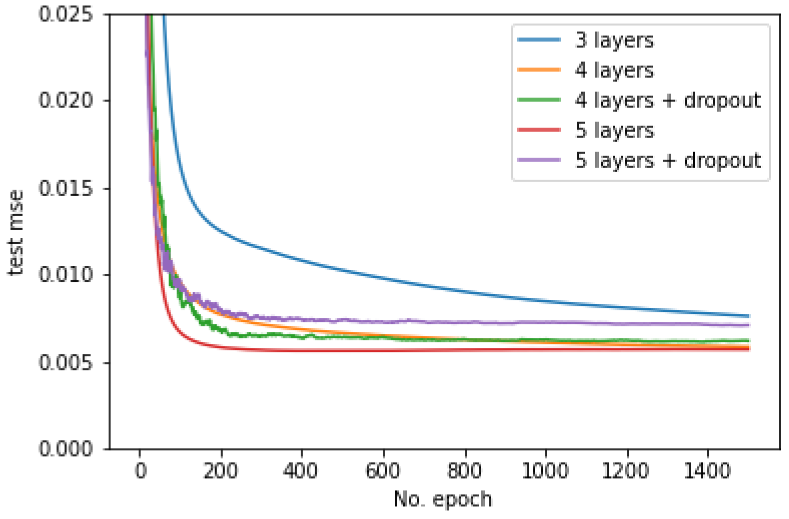


Figure 47. Test m.s.e for different models vs epochs

Code for running training once and obtaining the learning curves can be found in ***PartB/3B/3B.ipynb*** notebook.

a) Comparing performances between with and without dropout regularization

Code for obtaining the plots of target and predicted values, learning curves used below are all from notebook ***PartB/3B/3B.ipynb*** notebook

4-layer vs 4-layer with dropouts

From (figure 43) and (figure 45), there is evidence that the 4-layer with dropouts has lower overfitting or none comparing to the 4-layer with no dropouts. This is manifests in that the training m.s.e for the former has convergence at the same level of convergence as the test m.s.e, while this is not the case for the 4-layer with no dropouts. This also shown in the two plot (figure 48) and (figure 49), where the yellow crosses form closer shape to the shape of the blue target than the green crosses (better fitted).

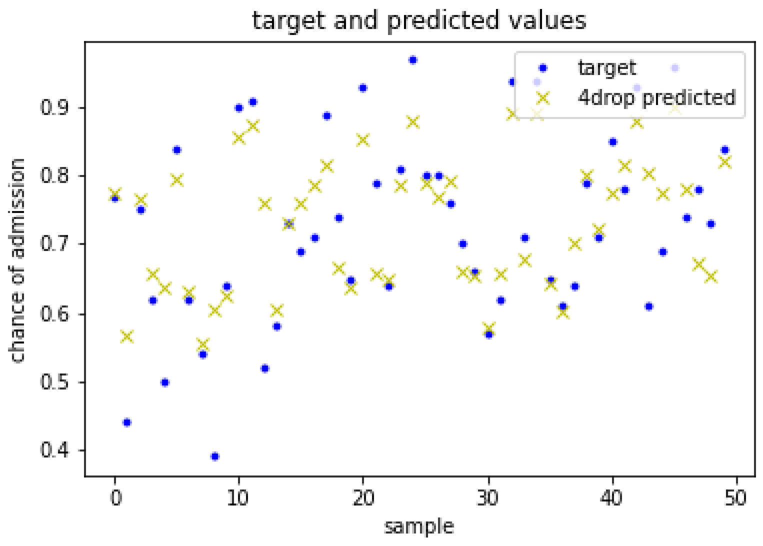


Figure 48. Target vs predicted values for 4-layer with dropouts

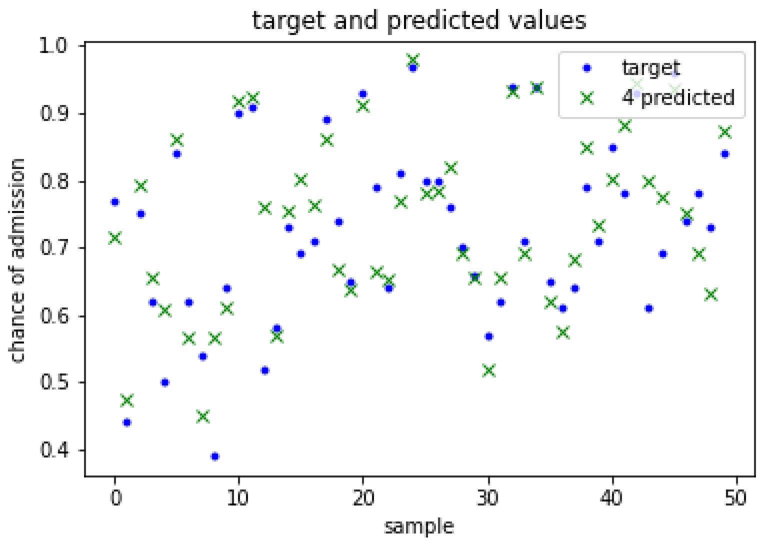


Figure 49. Target vs predicted values for 4-layer

However, the 4-layer with dropouts (figure 45) has noisier training and testing m.s.e compare to the one with no dropouts (figure 43).

From (figure 50) and (figure 47), it is noticeable that the testing m.s.e for the 4-layer with dropout converges faster and has more gradual decrement when it converges to that of the vanilla 4-layer.

However, the 4-layer network has lower test m.s.e (from table 8 and from figure 41) at 1500 epochs than the one with dropouts. Since we are evaluating model only on their accuracy, it can be said that the 4-layer with no dropouts has better performance than that of the one with dropout.

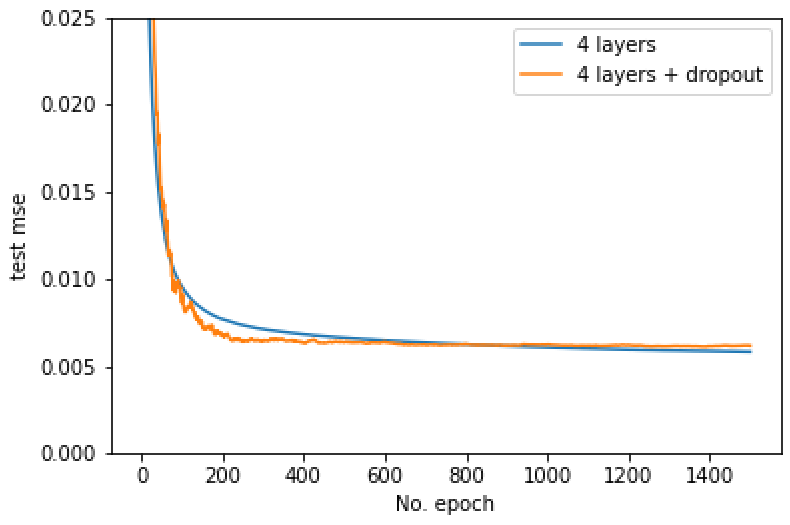


Figure 50. Testing m.s.e of 4-layer and 4-layer with dropout's vs epoch curve

5-layer vs 5-layer with dropouts

Like the observations between 4-layer and 4-layer with dropouts, the 5-layer with dropout (figure 46) has lower overfitting comparing to the one with no dropouts (figure 44). This is also shown in the two plots below (figure 51) and (figure 52), where the magenta crosses are shown to be forming shapes that are closer to the target points than the black crosses, indicating better fitting of the problem.

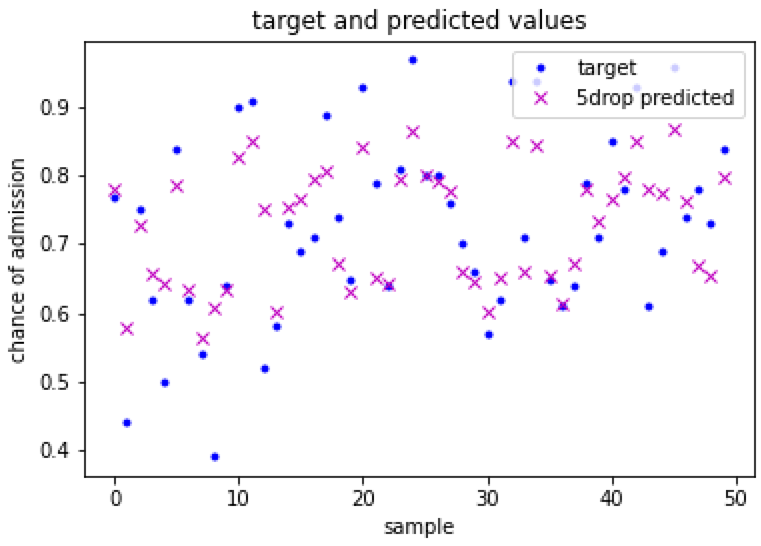


Figure 51. target and predicted for 5-layer + dropout

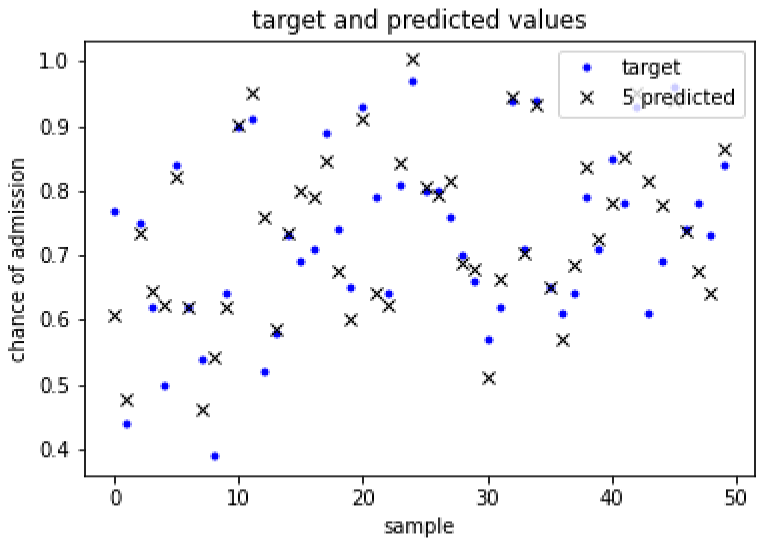


Figure 52. target and predicted for 5-layer, no dropout

Also, the 5-layer with dropouts has noisier training and testing m.s.e than the 5-layer one (figure 46 and 44)

From (figure 47) and (figure 53), it is clear evidence that the testing m.s.e of 5-layer with dropout model is converging slightly at the same number of epochs as that of the 5-layer.

The test m.s.e of the 5-layers with dropout is also said to be higher than that of the 5-layers, from (figure 53) and (figure 47). This is in-line with the average m.s.e observations from the table 8 of the 5 experiments conducted.

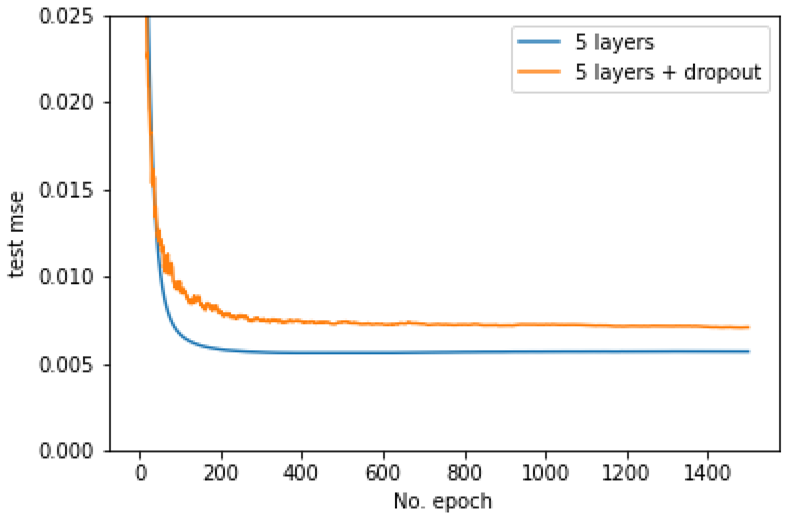


Figure 53. Testing m.s.e of 5-layer and 5-layer with dropout vs epochs

In short, using dropouts will lead to less overfitting of the model where the training and testing m.s.e convergence at the closer value than when no dropouts were used (figure 43 vs 45, figure 44 vs 46). However, this also introduce noisy data points to the training and test m.s.e curves. Also, the test m.s.e of the models with dropouts tends to be higher (poorer performances) than that of those with no dropouts (figure 50 and 53)

These observations are valid as introducing dropouts can prevent overfitting from the layers being over-reliant on the small dataset of input. Dropping the inputs at random rate of 20% will force the layers to learn and use all inputs in training, therefore improving its generalization.

As a result of reducing over-reliance of models on the few inputs available, we can observe the increase in m.s.e for those with dropouts and the noise in their curves (throwing themselves out from the assumptions/reliance on the few significant input features).

The verdict (if we were to ignore overfitting and early stopping of overfitting from training of longer epoch) is that the dropouts 4-layer and 5-layer has poorer performances (higher m.s.e) than that of the 4-layer and 5-layer without dropouts

b) Comparing performances between 3 layers, 4 layers and 5 layers network

Code for obtaining the plots of target and predicted values, learning curves used below are all from notebook ***PartB/3B/3B.ipynb*** notebook

From (figure 42,43, 44 and 47), we can see that the test m.s.e of 5-layer converges the fastest, following by 4-layer and 3-layer, respectively. The performance (based on lower m.s.e) also increases with more layers in the model (which can also be observed in table 8).

This can be accounted to the extra capacity of models with deeper number of layers as they have more neurons to learn and fit to the problem. This also allow models to learn and perform better in more complex problem.

This is demonstrated in the (figure 54), (figure 55) and (figure 56) below. As noticed, there are more black crosses nearer to blue dots (target) than green crosses, and red crosses. 5-layer is performing better than 4-layer and then 3-layer in this case.

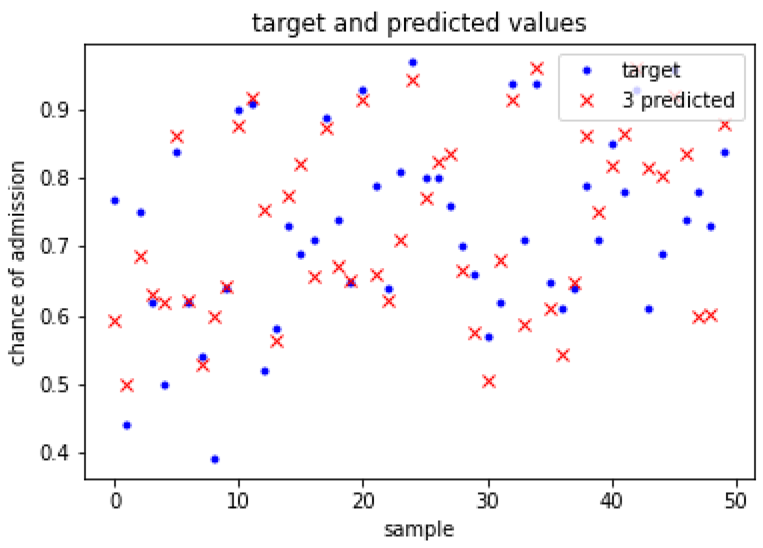


Figure 54. Target and predicted for 3-layer

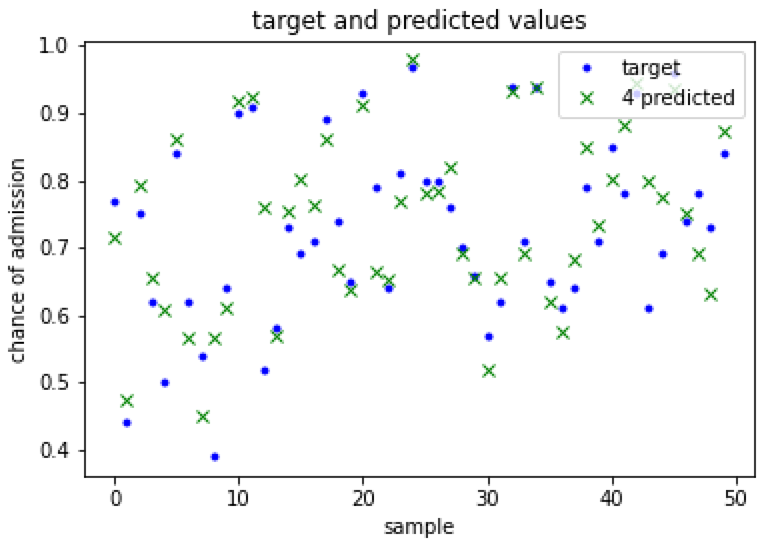


Figure 55. Target and predicted for 4-layer no dropout

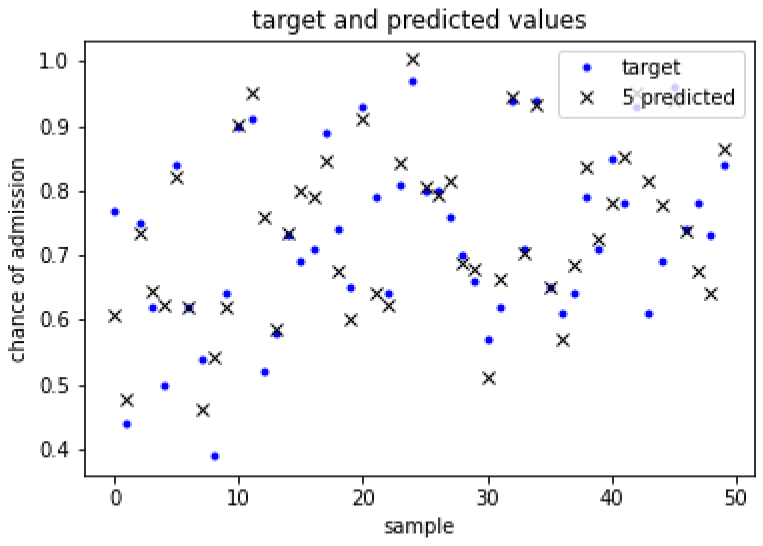


Figure 56. Target and predicted value for 5-layer with no dropout

For the 4-layer with dropout and 5-layer with dropout, this is not the trend (m.s.e of 5-layer + dropout is higher than that of the 4-layer + dropout) and we can only guess that this is because of reducing overfitting from their dropouts. This is a grey zone that we might consider working on for the future for the completeness of the project. This observation is shown in the figure 47.

c) Comparing performances between all of them

The dropout 4-layer and 5-layer will have better fitting to the problem (less overfitting) as discuss in part a, while worse performance (as the model does not perform so well like when it is forced to use all features in learning for its generalization).

We wish to highlight that in our case, for simplicity, **we shall take minimum m.s.e as the selection criteria for these 5 models (no concerns on overfitting of the model)**. From the ranking of the (table 8), where training to 1500 epochs were performed 5 times for each of the models listed**, model 5-layer with no dropouts has the best performance, followed by 4-layer with no dropouts, 4-layer with dropouts, 5-layer with dropouts and finally the optimal 3-layer**. This trend can also be observed in (figure 47) where the red curve (5-layer, no dropout) has the lowest test m.s.e and converges earliest amongst all models.

Should overfitting play a factor in selection, we might want to pick 4-layer + dropout, who has good fitting characteristics as shown in a, yet a ranking of third in terms of m.s.e performance (higher than 5-layer + dropout and 3-layer).

3.4. Summary and Discussion

To summarize, when given the task of predicting the chance of admitted to a master's degree program given 7 different features, we have built a good baseline model with 3-layer and make uses of the 7 features to regressing this probability. This is shown in 3.3.1

We then successfully enhance the performance of the model by removing less important features from the training of the model and managed to remove 2 features “research” and “sop” which lift the performances of the model up. This is shown in 3.3.2

Then in 3.3.3 we attempt to address the issue of overfitting (from using a smaller dataset of 5 features) by introducing dropouts and increasing the depth of our network. The former has proven to reduce the overfitting of our model to the dataset as shown in part 3.3.3a, while the later has proven to also lift the performance of the network as more hidden layer leads to better capacity of the model to fit with the problem as well. This is shown in 3.3.3b

We conclude that should overfitting is not the main problem, the lowest m.s.e model will be the 5-layer with no dropout, and this is the best performing model. When overfitting is an issue needs to be address, we reason why the 4-layer + dropout is a good model to make use of (in 3.3.3c)

In the experiments, we identified how stochastic nature of the training of the model can leads to different results of experiments and justify our approach of taking an average of 5 experiments to at least even out and give us a better estimate of our models' performance, this is shown in both the RFE process of 3.3.2 and the whole of 3.3.3. However, we may take this to another step of reshuffling the dataset after every experiment to generalize the experiments better, this is a room for future improvements.

We acknowledge that a better approach such as cross-evaluation with k-fold should be applied instead of just the 70:30 train-test split approach in model selection but this was not required by the assignment question. Lastly, we recognize the need to quantify the difference between the 4-layer + dropout and 5-layer + dropout better to explain why the increase in number of layer in this case does not yields a better m.s.e as shown in 3.3.3b.

4. Conclusion

Part A and B of this personal assignment has shown how time-consuming and resource-hungry deep learning and machine learning in general are. Given short time constraint and high stake (grades) for this project, we are only able to build acceptable models with a few tradeoffs. Our model might not be the global best, but they are the local best that we have for now. This simulate the process of training and building model in real life, time-consuming and very iterative.

5. Annex

Annex A: Part A codebase navigation

The PartA folder has 5 sub-folders (1A, 2A, 3A, 4A and 5A) corresponding to the different questions in A

Question A-1: There is only 1 notebook to run which gives all the answer to question A-1, as shown in part 2.3.1 of the report.

Question A-2, A-3, A-4:

These questions have the following notebooks in common, where X = 2,3,4 and hyperparameter = batch, neuron or beta.

- ***XA-overfit.ipynb:*** This obtain the train and test accuracy of each individual folds for each hyperparameter model. This needs to be run first before XA-ave.ipynb can be run
- ***XA-ave.ipynb:*** This notebook makes use of the history information from the XA-overfit.ipynb notebook and find the average training and testing accuracy curve with respect to epoch for each hyperparameter model. The plots of these average accuracy vs epoch are shown in part a of 2.3.2, 2.3.3, and 2.3.4. The average is neater to presents than those individual fold overfitted curve mentioned above. Requisite: XA-overfit.ipynb needs to be run first.
- ***XA-find-optimal-hyperparameter.ipynb:*** This notebook is run in part b for 2.3.2, 2.3.3 and 2.3.4. It carries out the experiments of 5 folds and obtain the average accuracy performance for the 5 folds of each hyperparameter model. We can based on these values to decide the optimal hyperparameter. It can also be run alone too and does not depends on another notebook.
- ***XA-optimal.ipynb:*** This notebook is run finally when the optimal hyperparameter has been decided. It can also be run alone too and does not depends on another notebook. This is used in the answer to part c of 2.3.2, 2.3.3 and 2.3.4.

In addition, question A-2 has an extra notebook called ***2A-time-per-epoch.ipynb.*** This calculate the average time taken to train 1 epoch for different batch size, as needed in answer of question 2A in 2.3.2.a

Question A-5:

This question has only 2 notebooks:

- ***5A.ipynb:*** run first to obtain the train and test accuracy vs epoch plot for part a of 2.3.5
- ***5A-find-optimal-model.ipynb:*** Run and obtain the average performance across 5 folds for the 3-layer model and the 4-layer model. This is used in part b of 2.3.5

Annex B: Part B codebase navigation

The PartB folder in the submitted code has 3 sub-folders (1B, 2B and 3B) corresponding to the answers to the 3 questions in part B.

Question B-1: All answers to this question and the part 3.3.1 is self-contained in the notebook 1B.ipynb.

Question B-2:

The following are the notebook of interest:

- ***2B-brute-force-rfe.ipynb:*** This show the process of RFE by brute force, the results are used in 3.3.2a
- ***2B-find-optimal.ipynb:*** This is used to carry out the experiments to decide which feature set (7, 6 or 5) gives the best performing model, as answered in 3.3.2b
- ***2B-overfit-curve.ipynb:*** This is to generate the individual train and test m.s.e vs epochs for each of the 7, 6 and 5 features model.

Question B-3

The following are the notebooks of interest in this sub folder:

- ***3B.ipynb:*** The train to overfit approach is done in this notebook. This results in the various train and test curve for the 5 models: 3-layer, 4-layer, 4-layer with dropout, 5-layer and 5-layer with dropout. This also include the predicted and target plots all shown in 3.3.3
- ***3B-ave.ipynb:*** The experiment to select the best model amongst the 5 models (repeated 5 times) is conducted here. Results are used in 3.3.3.