Assignment 1

CZ4042 Neural Network & Deep Learning

U1721316F

14th Oct 2020

Contents

Part A: Classification

Part A Introduction

This project aims at building neural networks to classify the Cardiotocography dataset containing measurements of fetal heart rate (FHR) and uterine contraction (UC) features on 2126 cardiotocograms classified by expert obstetricians.

Objective

The objective of the part A is to design a feedforward neural network that classifies the fetal cardiotocograms according to the fetal state class code (N= normal; S=suspect; P=pathologic).

Methodology

The process of predicting the class code is summarized in the next page.

Data Understanding

Exploratory Data Analysis

1. Explore the distribution of the data
2. What is NSP?
3. Which attribute is ignored?
4. How many labels are there in total?

Data Preparation

Data Pre-processing

1. Eliminate attribute - Class
2. Fix random state for reproducibility
3. Perform standard scaling for predictors
4. Split dataset into 70% train set, 30% test set

Experiment with scaling methods to get best baseline model

Modelling

Train and debug model

1. Train model using training set
2. Use model.summary() to verify model inputs
3. Plot loss and accuracy against epochs for train and test sets
4. Observe for anomalies
5. Debug errors

Find optimal no. of epochs

1. Set no. of epochs at a high number (i.e. 2000)
2. Note down no. of epochs where test loss is higher than train loss
3. Observe for anomalies

Sanity Checks

- Verify Kfold evenly distributes labels across each fold
- Rerun experiments to verify test results
- Test other hyperparameters such as learning rate & optimizers

Find optimal batch size/no. of neurons/decay

1. Perform 5-fold cross validation on training set (70:30 validation split)
2. Run a hyperparameter sweep to find the best CV accuracy
3. Record the runtime of the sweeps
4. Find balance between speed and accuracy of runs
5. Eliminate non-optimal hyperparameters
6. Tune next hyperparameter

Evaluation

Evaluate Test Scores

1. Predict using tuned parameters
2. Plot accuracy and loss against epochs for training and test sets

Exploratory Data Analysis

Before designing a neural network model, we must first understand the source of our data and the meaning of the attributes. The attribute information can be found on the UCI Machine Learning Repository website summarizes that the dataset is a classification task that can be used for either 10-class or 3-class experiments. Hence, this explains why we are to ignore the attribute 'CLASS'. From the pandas profiling report, we also know that there are no missing values in the dataset. With this understanding, we can proceed to pre-process the data.

Pandas Profiling Report: cardiotocography_report.html

Data Pre-processing

To ensure the reproducibility of results, we fix the random seed at 10 for both numpy and Tensorflow.

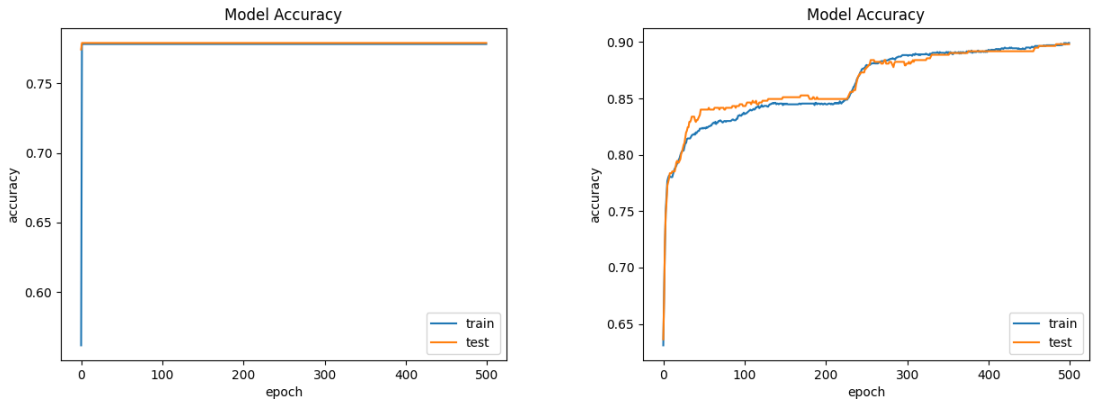


Fig 2. Comparison between MinMax Scaling (Left) and Standard Scaling (Right)

As observed above, the usage of min max scaling results in the model accuracy staying constant even after 300 epochs. The model has learned all the features within the first 2 epochs. Hence, we use standard scaling instead which is a much better pre-processing method.

After standardizing the features by removing the mean and scaling to the unit variance. The target feature is then set to 0, 1 and 2 instead of 1,2 and 3. The data is then split into 70% train data and 30% test data using scikit learn's train test split.

Files: classification_3L.py, Part A\Plots

Train and Debug Model

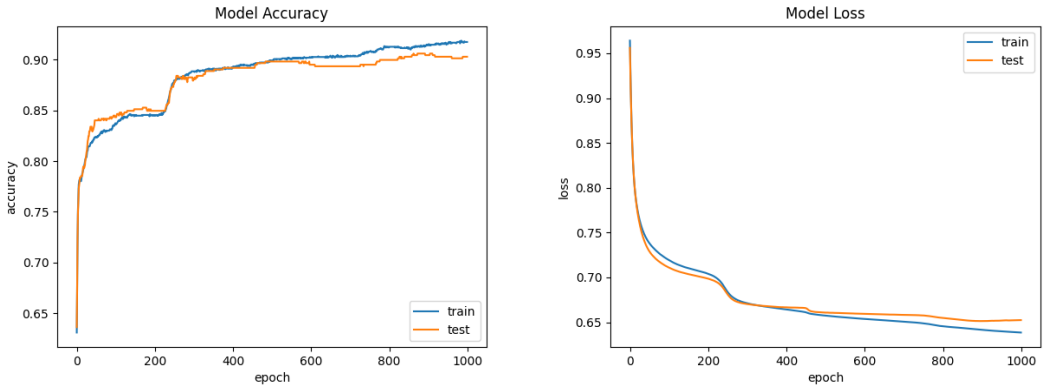


Figure 3: Model Accuracy and Loss against no. of epochs for train and test sets

The model is trained using the train set. The loss and accuracy are then plotted against the number of epochs for both train and test sets to determine the optimal no. of epochs. From Figure 3, we can observe that the model accuracy tapers off as number of epochs increases. Beyond 550 epochs, the model starts to overfit as the accuracy of the model becomes higher without an equivalent increase in test accuracy. This is similar observed for the loss plot as well. This firmly suggests that about 300 epochs should enable the network to learn the characteristics of dataset.

Files: classification_3L.py, Part A\Plots

Cross Validation

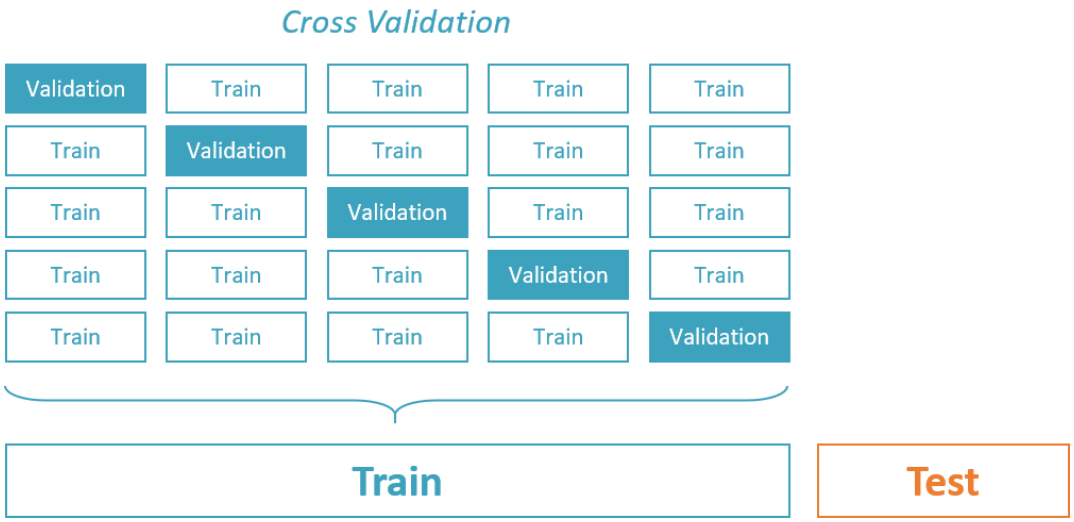


Figure 4: Train Test Split of Dataset

To ensure the validity of the results, a 5-fold cross validation is applied on the train set. This is visualized in Figure 4. We split the train set into 5 folds and use 1/5 of the data for validation. The cross validated accuracy is then derived from the average of the validation accuracy among the 5 folds.

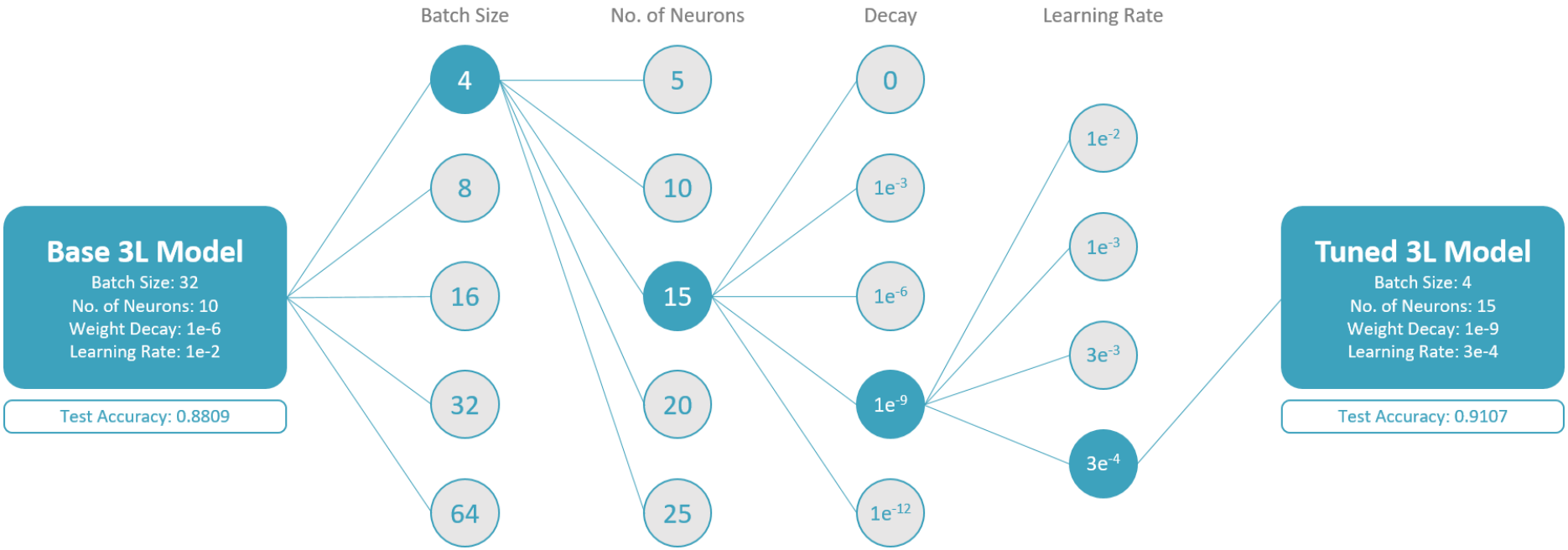


Figure 5: Overview of Hyperparameter Tuning for 3-layer network

Hyperparameter Tuning

For hyperparameter tuning, we start simple by creating a baseline model. Next, we select a hyperparameter to tune and choose the optimal parameter from a range of values based on the CV accuracy before moving on to the next parameter. Figure 5 shows the outcome of the parameter selection. The 4-Layer model is tuned in the same procedure as the 3-layer model, albeit with a smaller range of values.

Experiment and Results

Question 1

Design a feedforward neural network which consists of an input layer, one hidden layer of 10 neurons with ReLU activation function, and an output softmax layer. Assume a learning rate α=0.01, L2 regularization with weight decay parameter β=10−6, and batch size = 32. Use appropriate scaling of input features.

1a. Use the training dataset to train the model and plot accuracies on training and testing data against training epochs.

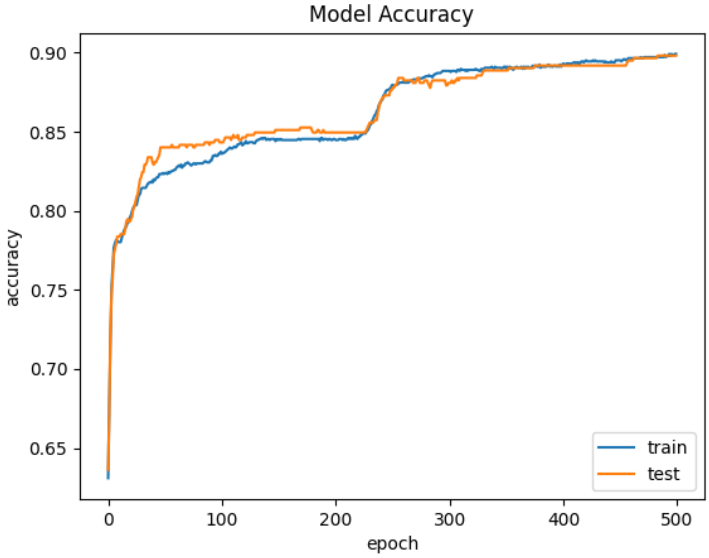


Figure 6: Model Accuracy against epochs for train and test sets

As seen in Figure 3, beyond 550 epochs, the model starts to overfit as the train accuracy increases without a subsequent increase in test accuracy. In Figure 6, we cut down the number of epochs to 500 to analyse the exact number of epochs to use which is about 300 epochs.

Files: classification_3L.py, Part A\Plots

1b. State the approximate number of epochs where the test error begins to converge.

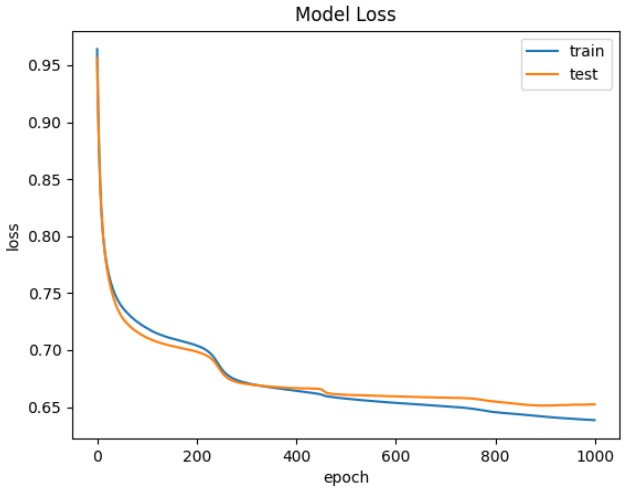


Figure 7: Model Loss against epochs for train and test sets

The approximate number of epochs is about 300 epochs. Beyond 300 epochs, the decrease in loss is minimal for both train and test sets.

Files: classification_3L.py, Part A\Plots

Question 2

2. Find the optimal batch size by training the neural network and evaluating the performances for different batch sizes.

2a. Plot cross-validation accuracies against the number of epochs for different batch sizes. Limit search space to batch sizes {4,8,16,32,64}. Plot the time taken to train the network for one epoch against different batch sizes.

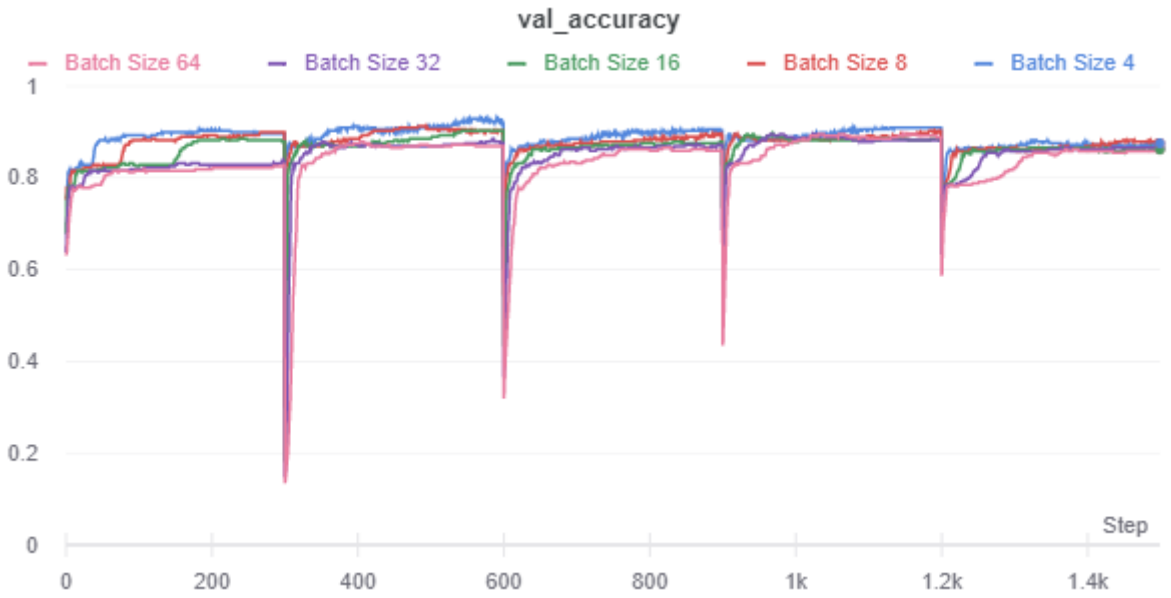


Figure 8: Cross Validated Accuracy against epochs for different batch sizes

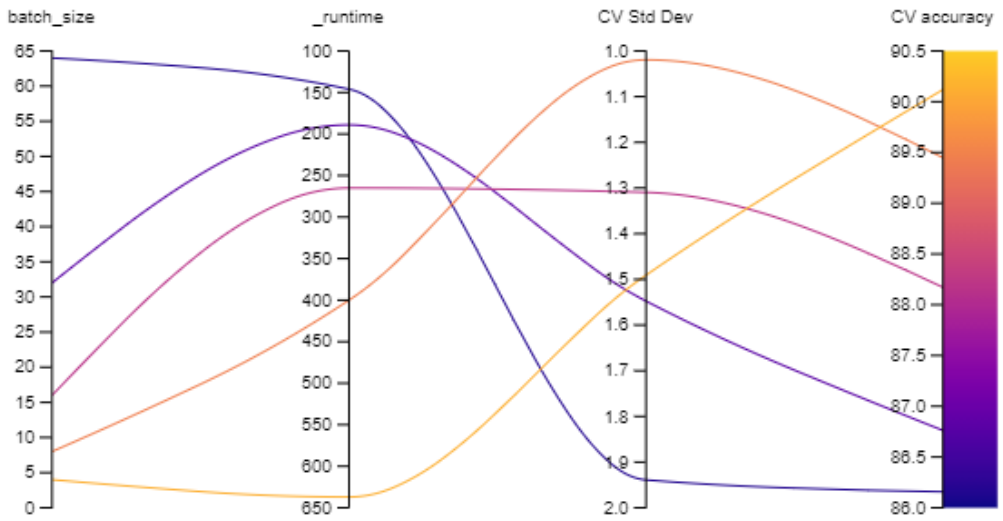


Figure 9: Parallel Coordinates Graph showing relevant metrics for batch size sweep

The time taken to train 1 epoch can be taken by the below formula.

$$Average\ Time\ to\ train\ 1\ epoch = \frac{runtime}{300\ epochs \times 5\ folds}$$

As number of epochs and folds remain the same for all 5 batch sizes, we can use runtime (in seconds) to compare the speed of using different batch sizes.

Files: classification_2_batch.py, sweep_2.yaml, wandb Batch Sweep

2b. Select the optimal batch size and state reasons for your selection.

From Figure 9, we can observe that the higher the batch size, the lower the cross validated accuracy of the model. Therefore, we have to consider the trade-off between speed and accuracy when tuning a neural network model.

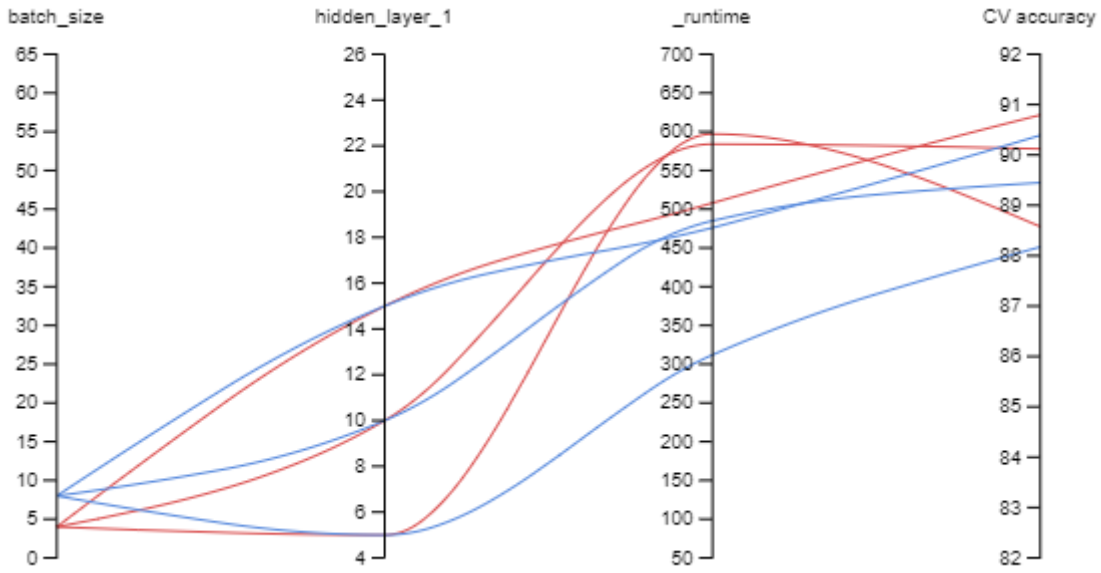


Figure 10: Effect of Batch Size on Runtime with different no. of neurons in hidden layer

[Red – Batch Size 4, Blue – Batch Size 8]

Further Verifications

To verify that the run time of the models is accurate, we do a hyperparameter sweep for 5, 10, 15 neurons for both batch sizes 4 and 8. From Figure 10, we can observe that the runtime for batch size 4 is at the maximum, about 600 seconds while runtime for batch size 8 is at the maximum, about 500 seconds. As we take cross validated accuracy to be the main metric, it makes sense to choose batch size 4.

Links: Batch Size 8, Batch Size 4

2c. Plot the train and test accuracies against epochs for the optimal batch size.

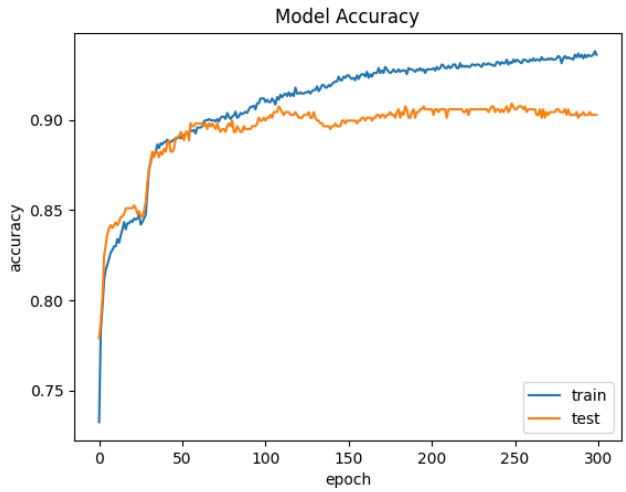


Figure 11: Train and Test Accuracy against epochs with optimal batch size

Question 3

Find the optimal number of hidden neurons for the 3-layer network designed in part (2).

3a. Plot the cross-validation accuracies against the number of epochs for different number of hidden-layer neurons. Limit the search space of number of neurons to {5,10,15,20,25}.

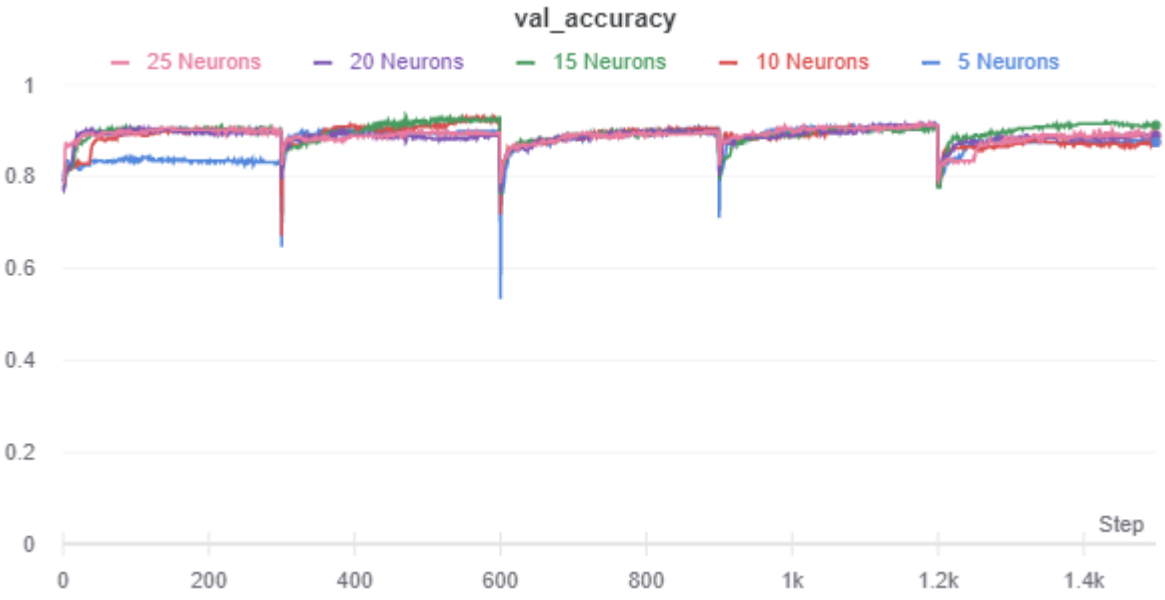


Figure 12: Cross Validated Accuracy against epochs for different no. of neurons

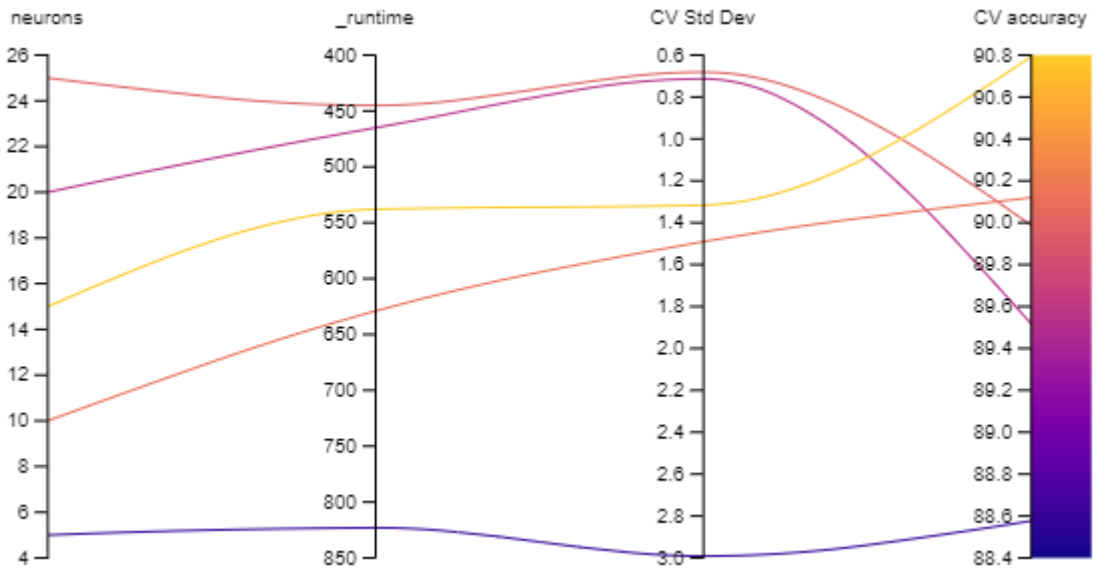


Figure 13: Parallel Coordinates Graph showing relevant metrics for hidden neurons sweep

Files: Neuron Sweep, classification_3_neuron.py, sweep_3.yaml

3b. Select the optimal number of neurons for the hidden layer. State the rationale for your selection.

Similar to question 2, we pick 15 neurons to be the optimal number of neurons for the hidden layer as it gives the highest CV accuracy.

3c. Plot the train and test accuracies against epochs with the optimal number of neurons.

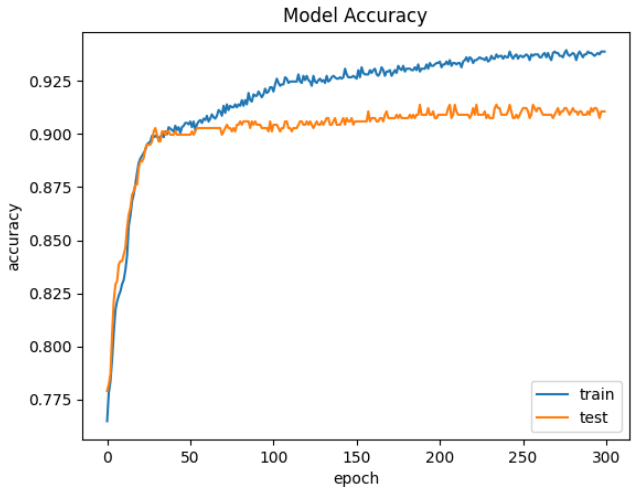


Figure 14: Train and Test Accuracy against epochs with optimal neurons

Question 4

4.Find the optimal decay parameter for the 3-layer network designed with optimal hidden neurons in part (3).

4a. Plot cross-validation accuracies against the number of epochs for the 3-layer network for different values of decay parameters. Limit the search space of decay parameters to $\{0,10^{-3},10^{-6},10^{-9},10^{-12}\}$.

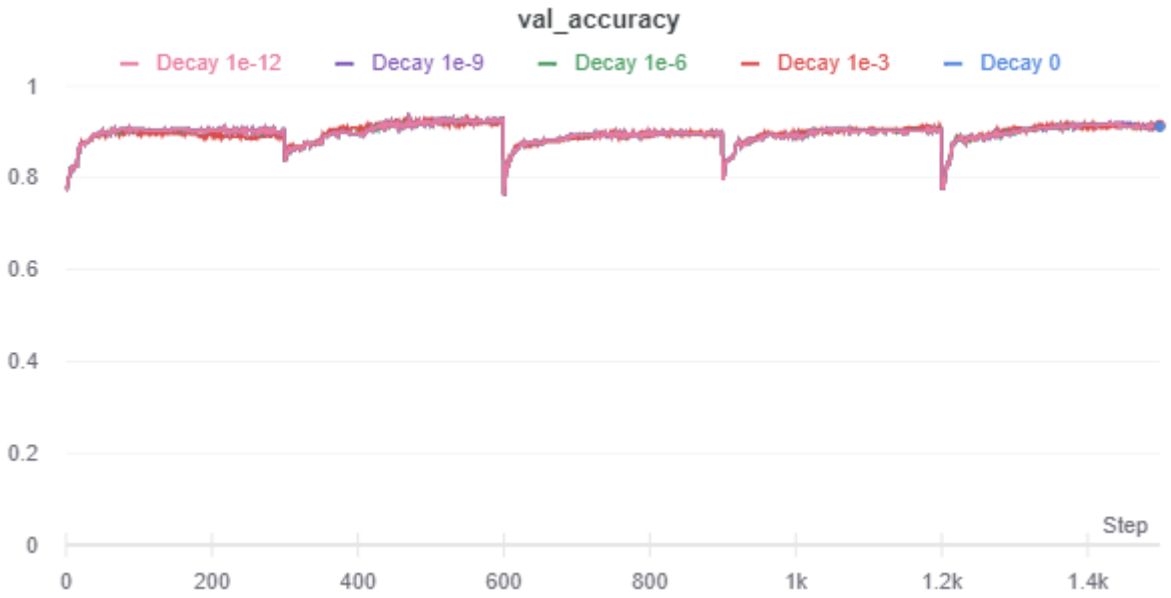


Figure 14: Cross Validated Accuracy against epochs for different decay values

4b. Select the optimal decay parameter. State the rationale for your selection.

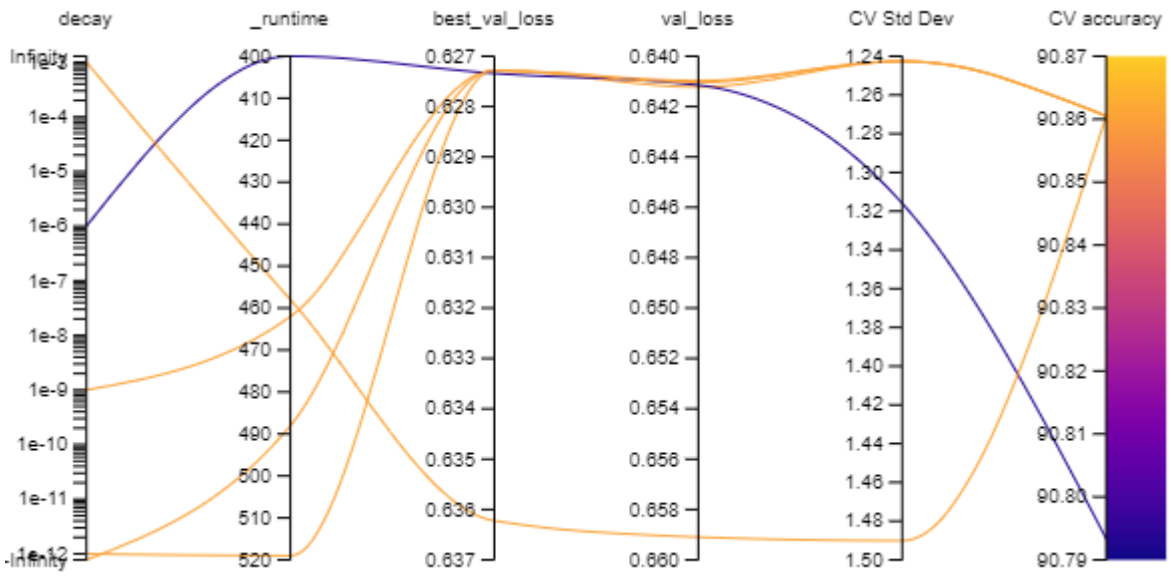


Figure 15: Parallel Coordinates Graph showing relevant metrics for decay sweep

Name (5 visualized)	decay	CV accuracy	Runtime	CV Std Dev	accuracy	best_val_lo	loss	val_accurac	val_loss
Decay 1e-12	1.000e-12	90.861	8m 47s	1.243	0.958	0.6273	0.603	0.9125	0.641
Decay 1e-9	1.000e-9	90.861	7m 52s	1.243	0.9572	0.6273	0.603	0.9125	0.6412
Decay 1e-6	0.000001	90.793	6m 46s	1.316	0.9572	0.6273	0.603	0.9125	0.6411
Decay 1e-3	0.001	90.861	7m 46s	1.49	0.9488	0.6362	0.6239	0.9158	0.6591
Decay 0	0	90.861	8m 15s	1.243	0.958	0.6273	0.603	0.9125	0.641

Figure 16: Table showing decay sweep results

The selection of the decay parameter cannot be decided based on just the CV accuracy of the model. In Figure 16, we can observe that using decay parameters $1e^{-3}$, $1e^{-9}$, $1e^{-12}$ or 0 results in the same CV accuracy of 90.861. Hence, the next two metrics we should consider is CV standard deviation and runtime. This leaves us with a decay rate of $1e^{-9}$ as the parameter with the lowest runtime and lowest CV standard deviation. Therefore, the decay rate of $1e^{-9}$ is selected.

Files: Decay Sweep, classification_4_decay.py, sweep_4.yaml

4c. Plot the train and test accuracies against epochs for the optimal decay parameter.

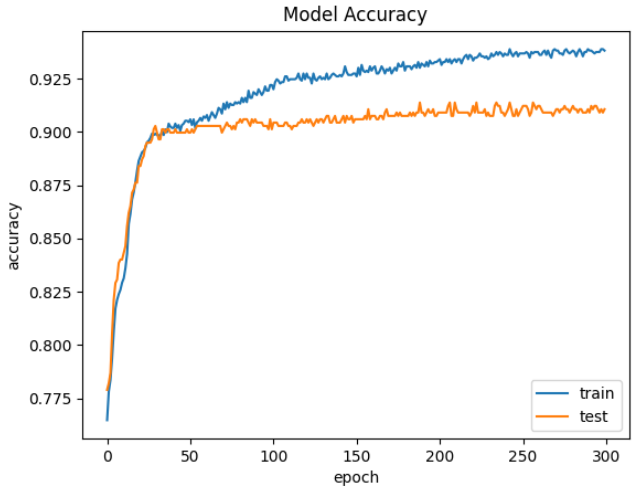


Figure 17: Train and Test Accuracy against epochs with optimal decay rate

Tuning Learning Rate

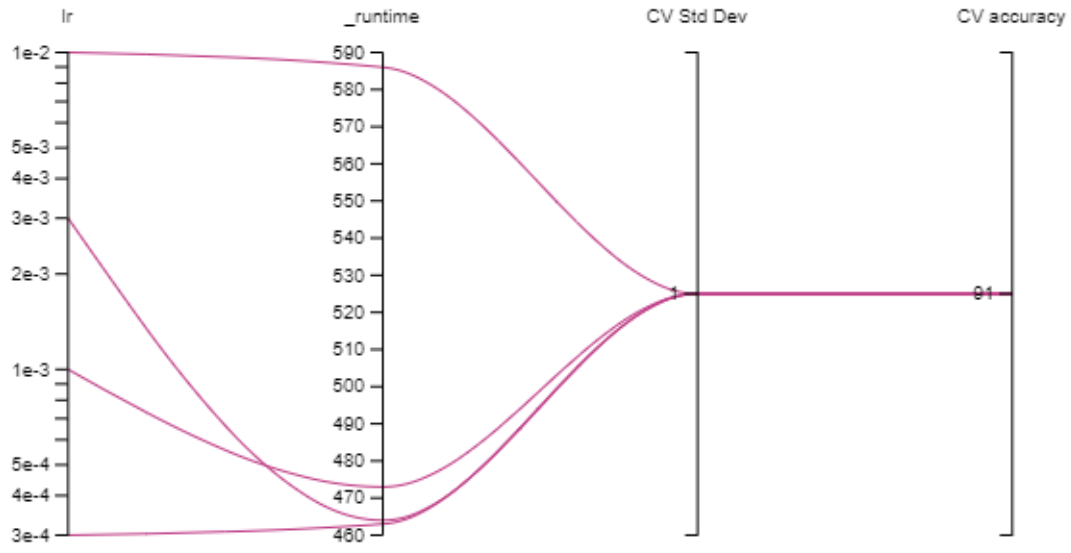


Figure 18: Parallel Coordinates Graph showing relevant metrics for learning rate sweep

By doing a hyperparameter sweep with learning rate of 1e-2, 1e-3, 3e-3 and 3e-4, we can deduce that a change in learning rate does not help to increase CV accuracy further. Choosing a learning rate of $3e^{-4}$ helps to reduce the time taken to train the model.

Files: classification_5_lr.py, sweep_5.yaml, learning rate sweep

Question 5

5. After you are done with the 3-layer network, design a 4-layer network with two hidden-layers, each consisting 10 neurons, and train it with a batch size of 32 and decay parameter 10^{-6}.

5a. Plot the train and test accuracy of the 4-layer network.

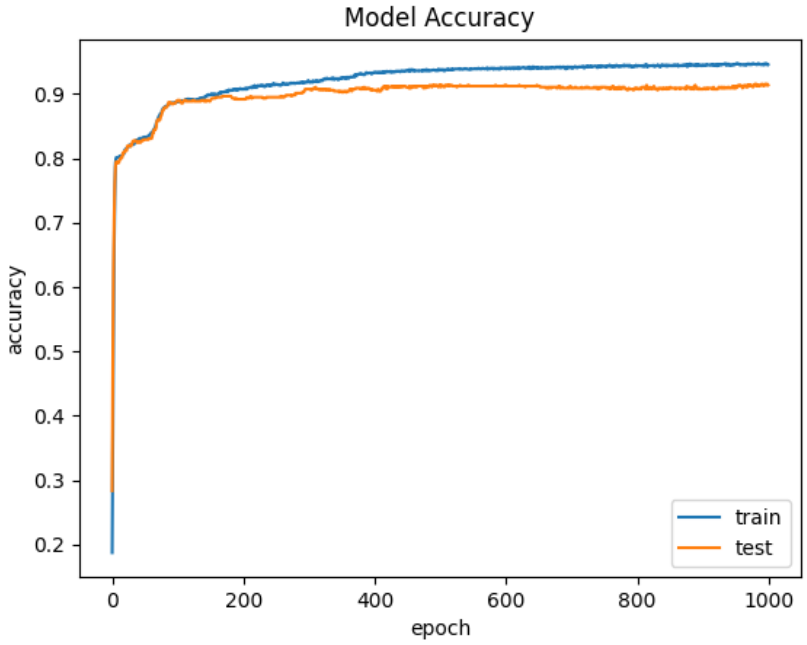


Figure 19: Train and Test Accuracy against epochs for 4 Layer Network

Files: classification_4L.py

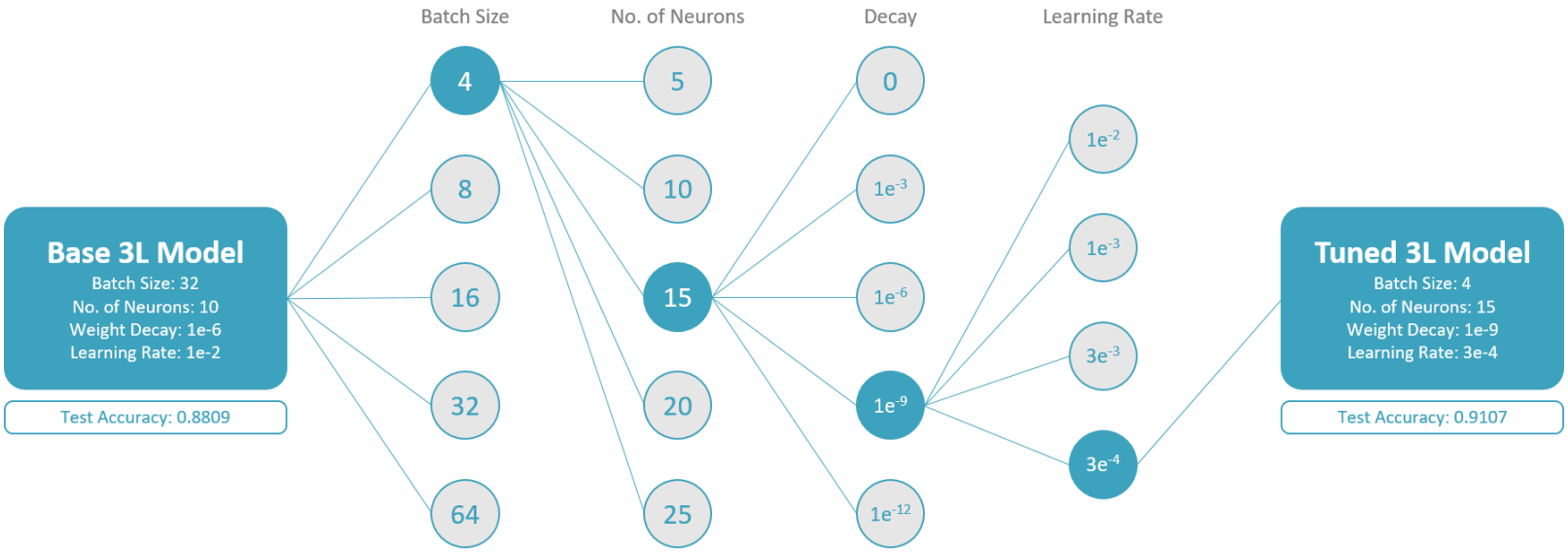


Figure 20: Overview of Tuning Process for 3L Network

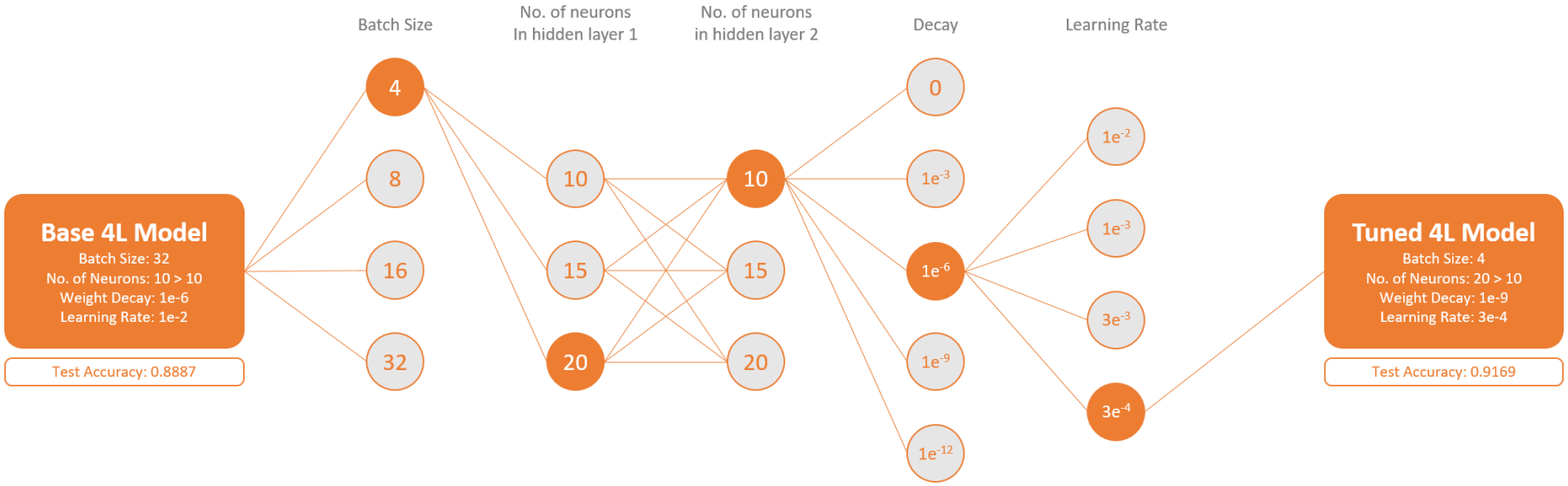


Figure 21: Overview of Tuning Process for 4L Network

5b. Compare and comment on the performances of the optimal 3-layer and 4-layer networks.

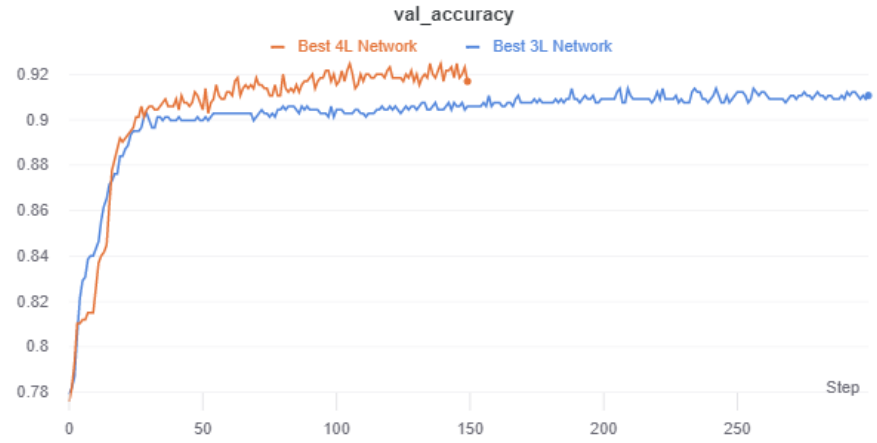


Figure 22: Test Accuracy against no. of epochs for 3L and 4L network

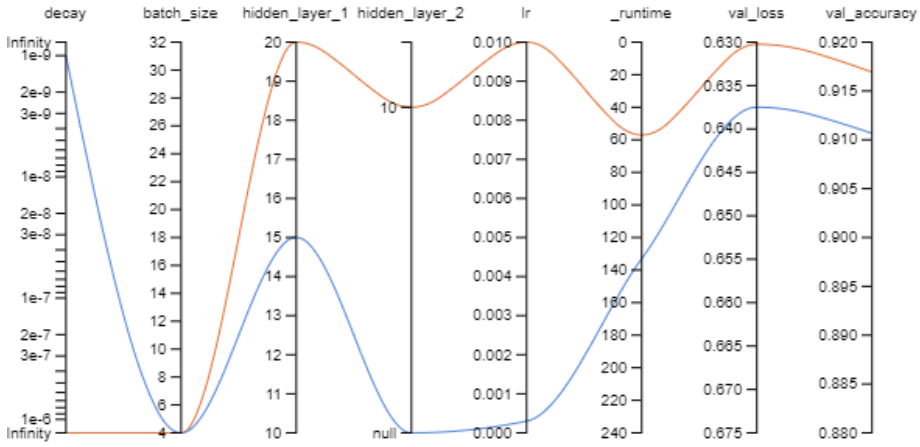


Figure 23: Parallel Coordinates Graph showing comparison between 3L and 4L network

From Figure 23, we can deduce that the 4-layer network (0.9169) performs better than the 3-layer network (0.9107) in both test accuracy and loss. Due to the no. of epochs set at 300 which is twice of what the 4-Layer network has (Figure 22), the time taken to train the model for the 3-layer network is significantly greater than the 4-layer network. If we were to compare between the average amount of time needed to train each epoch, the 3-layer network takes 0.44s (132/300) while the 4-layer network takes 0.367s (55/150). Thus, this proves that the increasing the number of layers not only improves the test results but allows the model to converge faster in this case.

4 Layer NN Optimization: classification_6.py, classification_7_neuron.py, classification_8_decay.py, classification_9_lr.py, sweep_6.yaml, sweep_7.yaml, sweep_8.yaml, sweep_9.yaml, 4L Sweeps

Test Results: Test Prediction Results, classification_part C.py, classification_part 5b.py

Additional Analysis

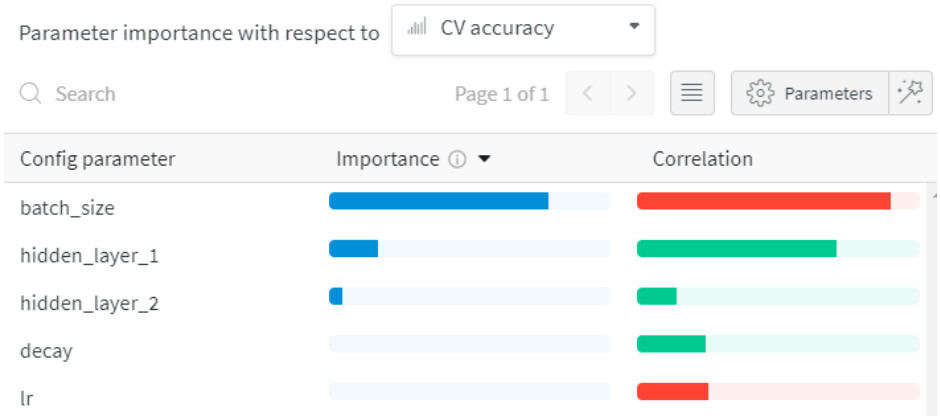


Figure 24: Parameter importance and correlation for both 3L and 4L network

Using Weights and Biases' parameter importance tool, the importance metric reports the feature importance values trained using a random forest. This highlights that choosing a batch size has a significant impact on the model's performance. Lowering the batch size potentially increases the performance of the model at the expense of a longer training time.

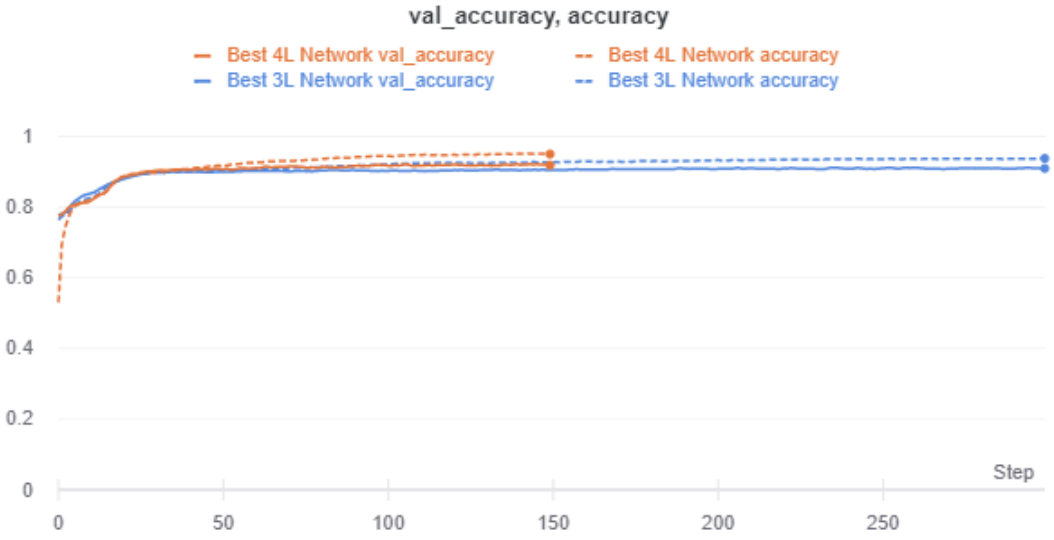


Figure 25: Train and Test Accuracy against no. of epochs for best 3L and 4L model

The figure above compares the test accuracy and the train accuracy for the 3-layer and 4-layer neural network. From the above, we can conclude that the model has overfitted to a slight extent. This can be mitigated by further tuning of the other hyperparameters in the model and introducing early stopping when there is little to no improvement in the model's performance.

Conclusion

The number of hidden layers, neurons, decay rate and batch size are not the only hyperparameters that can be tuned in a feedforward neural network. Tuning the learning rate, learning rate schedule or loss function can make a big difference to the model's performance and could be tuned first before other parameters.

To further optimize the model for speed, we can pick a model with more layers and neurons and use early stopping to prevent it from overfitting. Other regularization techniques such as adding dropout layers can also help to prevent the model from overfitting.

To further improve the model's performance, we can adopt a 'coarse to fine' approach by further reducing the range of hyperparameters and eliminating non-optimal hyperparameters.

Part B: Regression

Part B Introduction

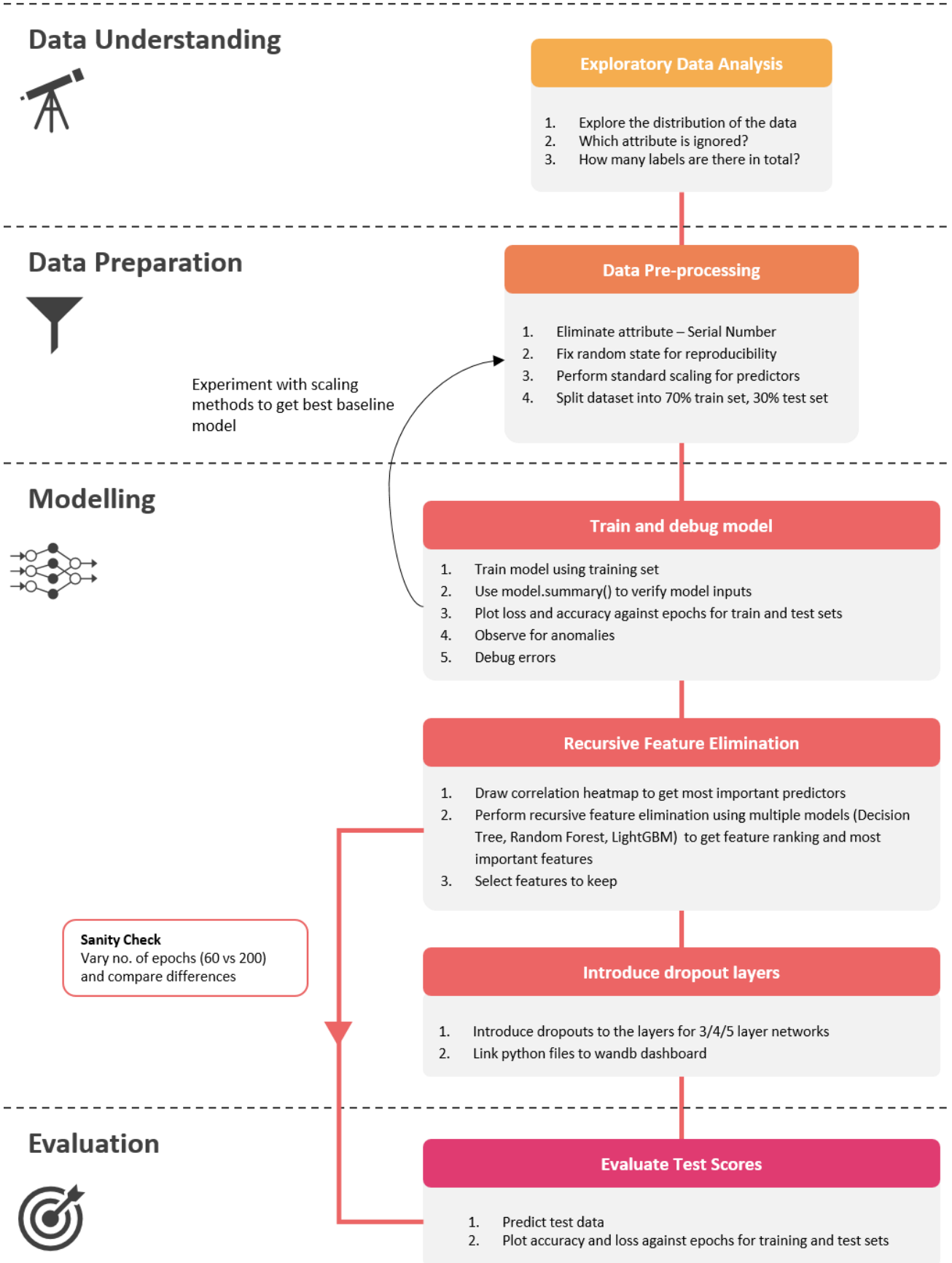
This assignment uses the data from the Graduate Admissions Predication. The dataset contains several parameters like GRE score (out of 340), TOEFL score (out of 120), university Rating (out of 5), strengths of Statement of Purpose and Letter of Recommendation (out of 5), undergraduate GPA (out of 10), research experience (either 0 or 1), that are considered important during the application for Master Programs.

Objective

The objective of the part A is to design a feedforward neural network that predicts the probability of a student getting admitted into the master's program. The predicted parameter is the chance of getting an admit (ranging from 0 to 1).

Methodology

The steps taken to complete part 2 is summarized in the next page.


Data Understanding
Exploratory Data Analysis
1. Explore the distribution of the data
2. Which attribute is ignored?
3. How many labels are there in total?
Data Preparation
Data Pre-processing
1. Eliminate attribute – Serial Number
2. Fix random state for reproducibility
3. Perform standard scaling for predictors
4. Split dataset into 70% train set, 30% test set
Experiment with scaling methods to get best baseline model
Modelling
Train and debug model
1. Train model using training set
2. Use model.summary() to verify model inputs
3. Plot loss and accuracy against epochs for train and test sets
4. Observe for anomalies
5. Debug errors
Recursive Feature Elimination
1. Draw correlation heatmap to get most important predictors
2. Perform recursive feature elimination using multiple models (Decision Tree, Random Forest, LightGBM) to get feature ranking and most important features
3. Select features to keep
Sanity Check
Vary no. of epochs (60 vs 200) and compare differences
Introduce dropout layers
1. Introduce dropouts to the layers for 3/4/5 layer networks
2. Link python files to wandb dashboard
Evaluation
Evaluate Test Scores
1. Predict test data
2. Plot accuracy and loss against epochs for training and test sets

Experiment and Results

Question 1

Design a 3-layer feedforward neural network consists of an input layer, a hidden-layer of 10 neurons having ReLU activation functions, and a linear output layer. Use mini-batch gradient descent with a batch size = 8, $L2$regularization at weight decay parameter β=10−3and a learning rate α=10−3to train the network.

1a. Use the train dataset to train the model and plot both the train and test errors against epochs.

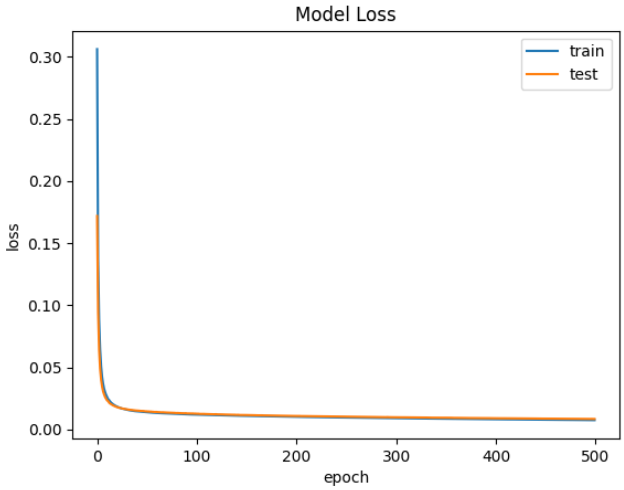


Figure 19: Train and Test Accuracy against epochs for 4 Layer Network

1b. State the approximate number of epochs where the test error is minimum and use it to stop training.

The approximate number of epochs where the test error is minimum is about 75 epochs.

1c. Plot the predicted values and target values for any 50 test samples.

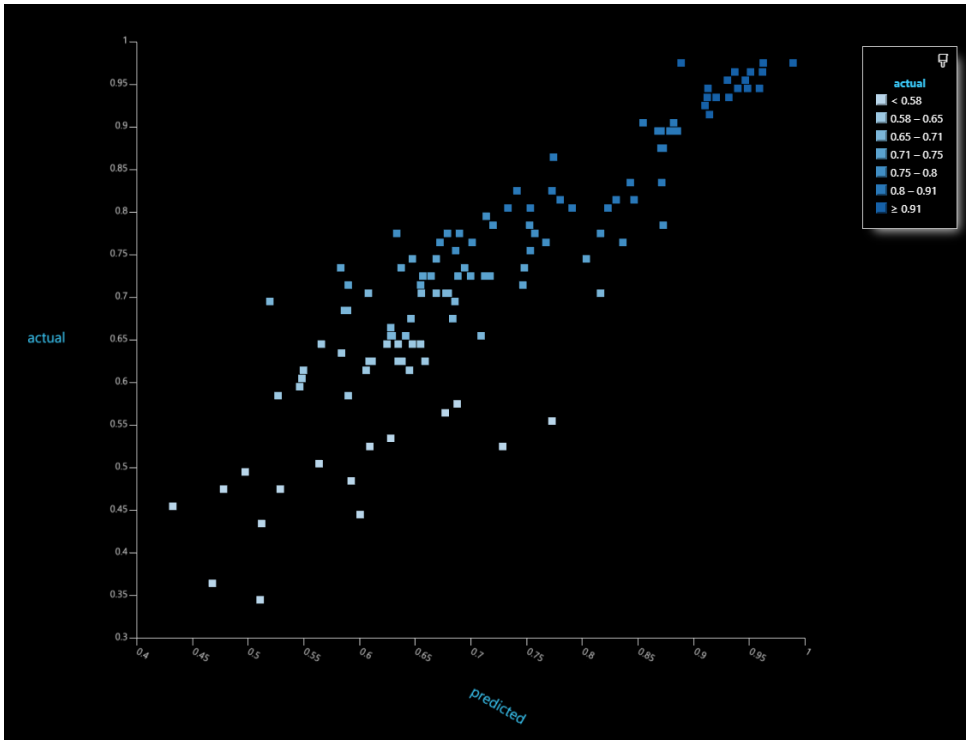


Figure 20: Actual values against predicted values plotted using vscode's SandDance

From Figure 20, we can observe that as the probability of getting an admit increases, the chances of getting a prediction closer to the actual value is greater.

Question 2

2. Recursive feature elimination (RFE) is a feature selection method that removes unnecessary features from the inputs. Start by removing one input feature that causes the minimum drop (or maximum improvement) in performance. Repeat the procedure recursively on the reduced input set until the optimal number of input features is reached. Remove the features one at a time. Compare the accuracy of the model with all input features, with models using 6 input features and 5 input features selected using RFE. Comment on the observations.

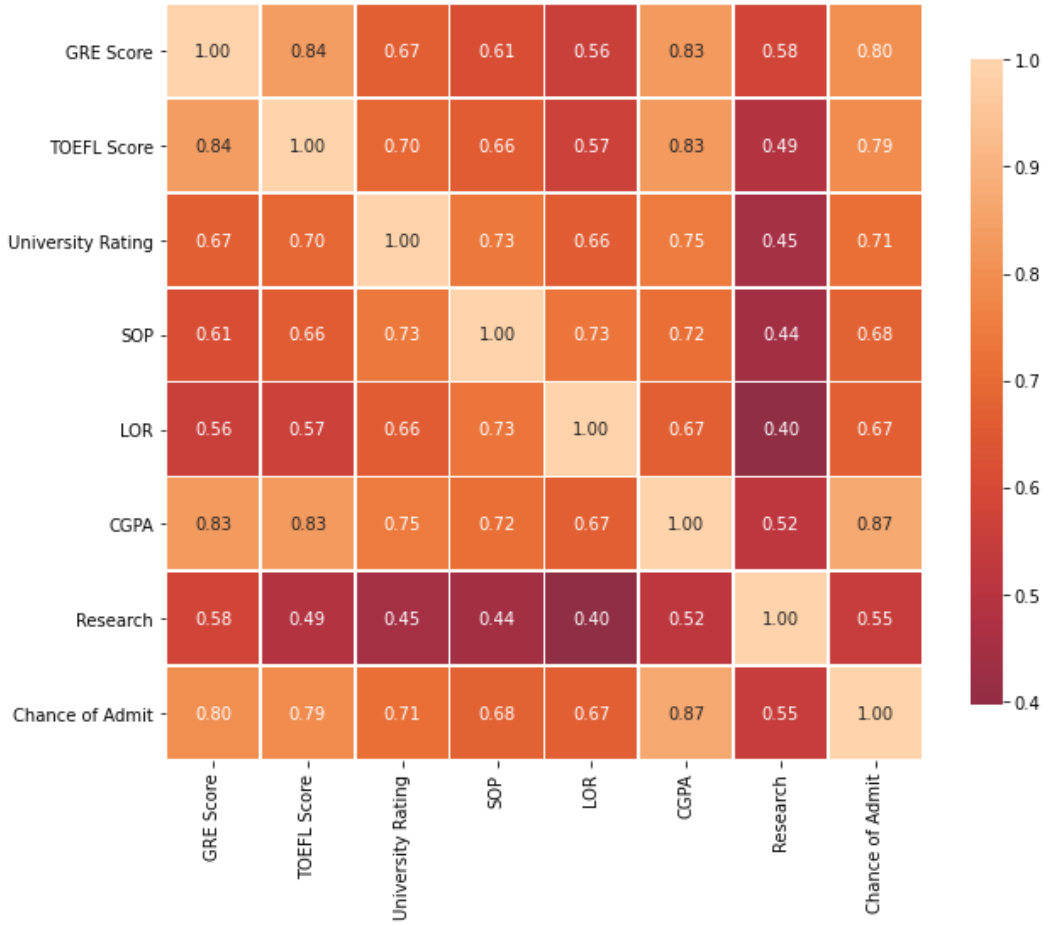


Figure 21: Correlation Plot

Before performing recursive feature elimination, from the correlation plot, we can observe that CGPA, TOEFL and GRE scores are highly correlated to the target variable: Chance of Admit. Therefore, these features should be kept.

Feature	Decision Tree	LGBM	Linear Regression	Random Forest
CGPA	1	1	1	1
GRE Score	2	2	2	1
TOEFL Score	3	3	1	1
SOP	4	4	1	1
LOR	5	5	1	1
Research	6	7	1	1
University Rating	7	6	1	1

Figure 22: Comparison between models for feature ranking after performing RFE

In Figure 22, we can observe that the features research and university ranking both rank the lowest among the 7 features. As such, they will be removed one at a time to be used to compare the accuracy of the model with lesser features. Research is chosen as the first feature to eliminate as it has the lowest correlation with the target attribute.

Files: RFE_preprocess.ipynb, RFE.ipynb

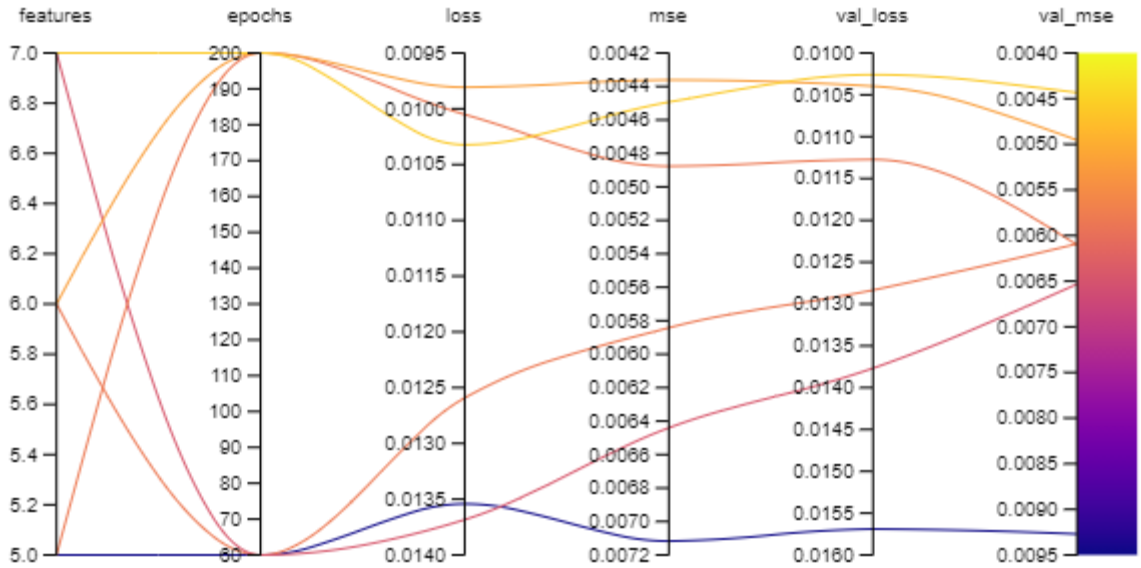


Figure 23: Parallel Coordinates Graph comparing between 60 and 200 epochs for 5/6/7 features

In question 1, we conclude that the test error is at a minimum at about 60 epochs. However, increasing the number of epochs to 200 changes the performance of the model for 7 features, resulting in the lowest test loss and test mse. To ensure the correctness of the test results, 200 epochs is used instead of 60 epochs.

The above figure also shows that removing features leads to a decrease in model performance. Therefore, it is recommended not to remove features unnecessarily unless we are dealing with a large number of features in a dataset.

Links: Test Results, regression_2_5f.py, regression_2_6f.py, regression_2_7f.py

Question 3

3. Design a four-layer neural network and a five-layer neural network, with the hidden layers having 50 neurons each. Use a learning rate of 10^{-3} for all layers and optimal feature set selected in part (3). Introduce dropouts (with a keep probability of 0.8) to the layers and report the accuracies. Compare the performances of all the networks (with and without dropouts) with each other and with the 3-layer network.

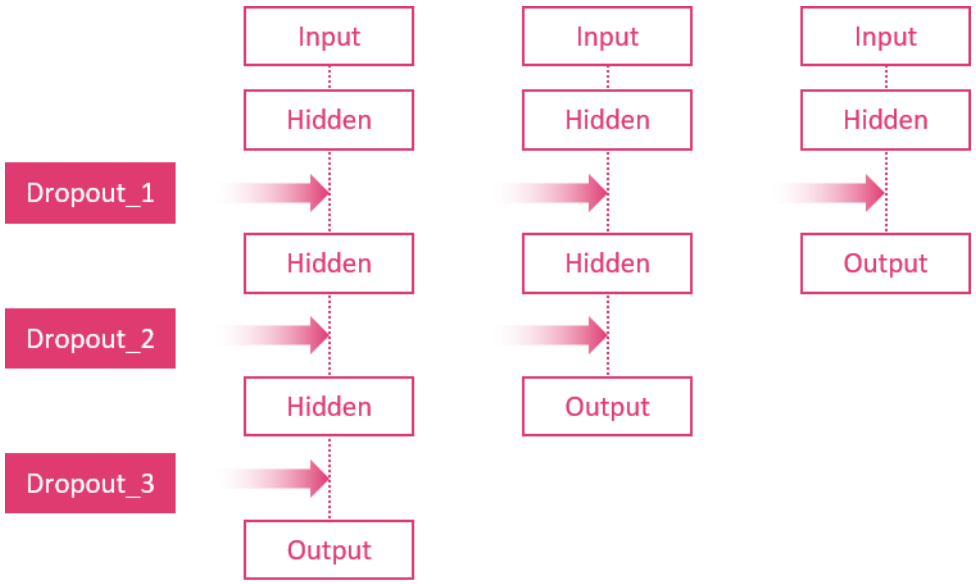


Figure 24: Illustration of where the dropout layer is inserted

We introduce a dropout layer after every hidden layer that is illustrated in Figure 24.

Name (14 visualized)	dropout_1	dropout_2	dropout_3	Runtime	hidden_lay	hidden_lay	hidden_lay	layers	loss	mse	val_loss	val_mse
4 Layer Model	No	Yes	No	28s	50	50	-	4	0.04906	0.004234	0.04832	0.003531
3 Layer Model	Yes	No	No	23s	10	-	-	3	0.01213	0.006602	0.009511	0.003986
4 Layer Model	Yes	Yes	No	24s	50	50	-	4	0.04995	0.005716	0.04828	0.004085
5 Layer Model	Yes	No	Yes	27s	50	50	50	5	0.08792	0.005233	0.08675	0.004118
5 Layer Model	No	No	Yes	29s	50	50	50	5	0.08699	0.003578	0.08756	0.004208
5 Layer Model	No	Yes	No	26s	50	50	50	5	0.08744	0.004107	0.08752	0.004242
5 Layer Model	Yes	No	No	27s	50	50	50	5	0.08761	0.004194	0.08766	0.004298
4 Layer Model	Yes	No	No	23s	50	50	-	4	0.04921	0.004389	0.04913	0.004337
3 Layer Model	No	No	No	26s	10	-	-	3	0.01032	0.004493	0.01026	0.004435
5 Layer Model	No	No	No	30s	50	50	50	5	0.087	0.002579	0.0888	0.004437
5 Layer Model	No	Yes	Yes	27s	50	50	50	5	0.08782	0.005197	0.08701	0.004443
4 Layer Model	No	No	No	28s	50	50	-	4	0.04865	0.002771	0.05029	0.00445
5 Layer Model	Yes	Yes	No	26s	50	50	50	5	0.08854	0.00586	0.08721	0.004582
5 Layer Model	Yes	Yes	Yes	31s	50	50	50	5	0.08909	0.006969	0.08713	0.005066

Figure 25: Table showing experiment results (sorted by lowest val_mse)

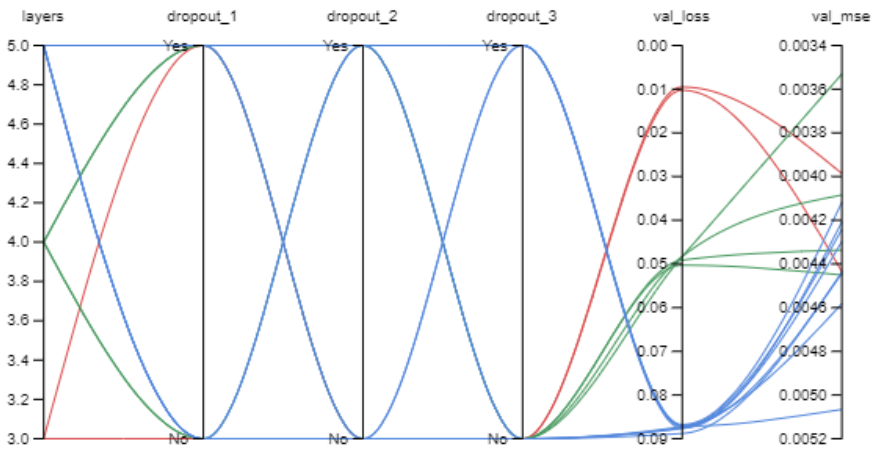


Figure 25: Comparison of layers against test mse metric

Comparison of No. of Layers

In general, the model performance decreases as the number of layers increases. This could be due to overfitting as the number of neurons per layer is 50. On the other hand, the 3-layer network produces above average results as it only has 10 neurons in its hidden layer.

Comparison of Dropout vs No Dropouts

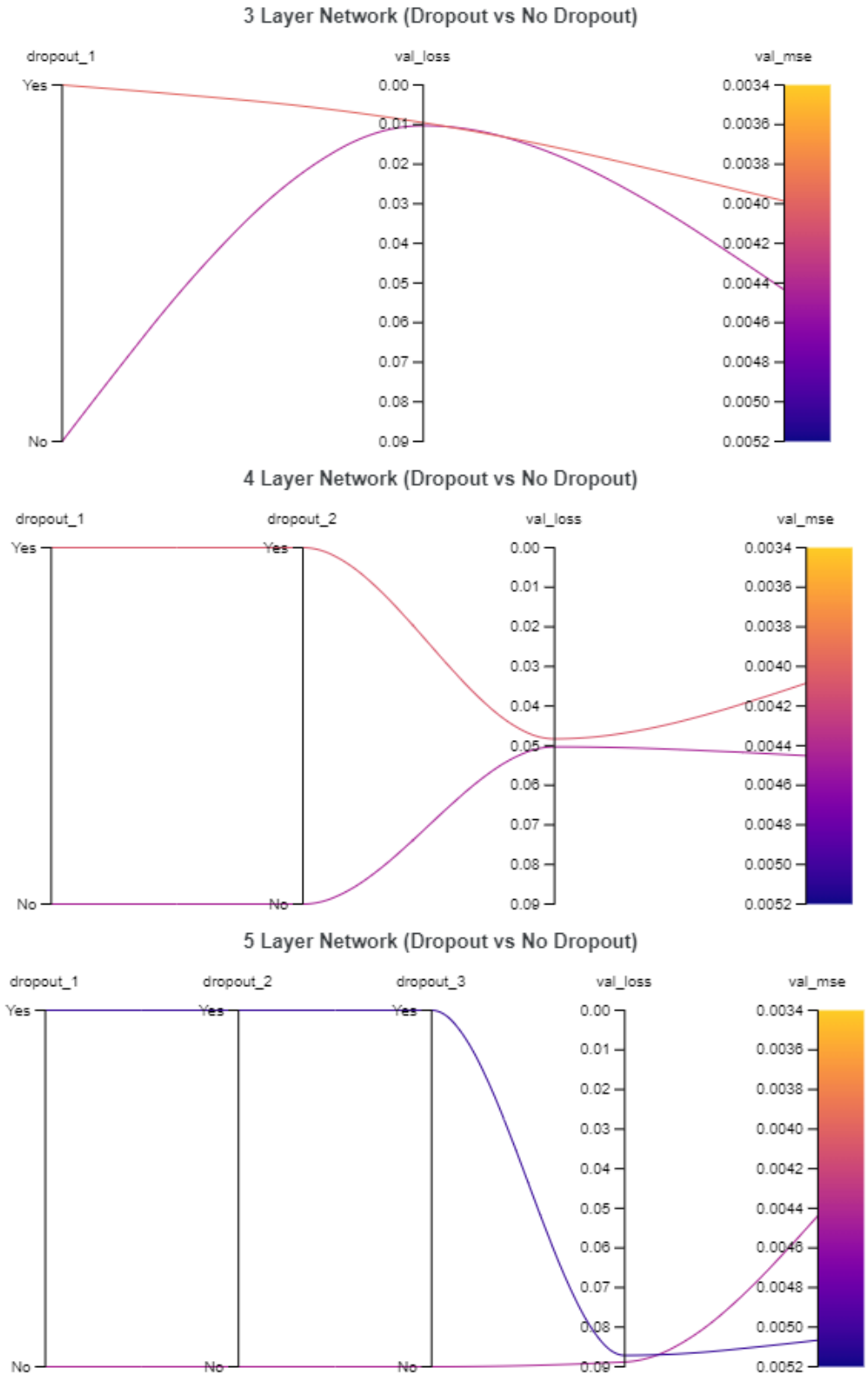


Figure 26: Comparison of layers for dropout and no dropout layer against test mse metric

For both the 3-layer and 4-layer network, it is observed that the model performance has improved after adding dropout layers to all hidden layers. However, the 5-layer network shows a decrease in model performance. To ensure the best use of dropout layers, we can vary the number of dropout layers. This is observed in figure 25 where the best performing model is the 4-layer model with only 1 dropout layer after the 2nd hidden layer. The top 5 performing models all use a varying number of dropout layers as having a full dropout is too strong and often leads to a poorer performance.

Links: Test Runs for Dropouts, regression_3_3L.py, regression_3_4L.py, regression_3_5L.py

Conclusion

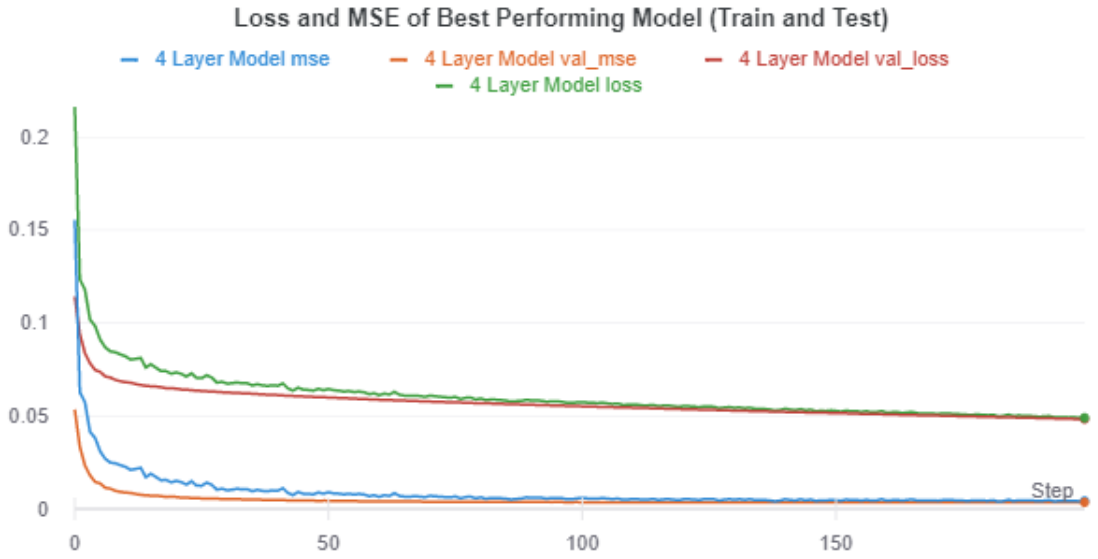
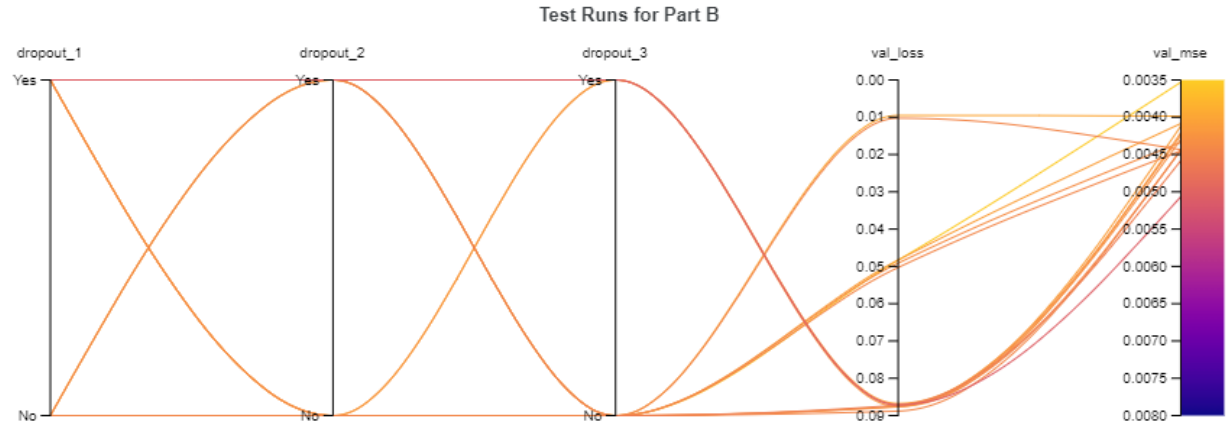
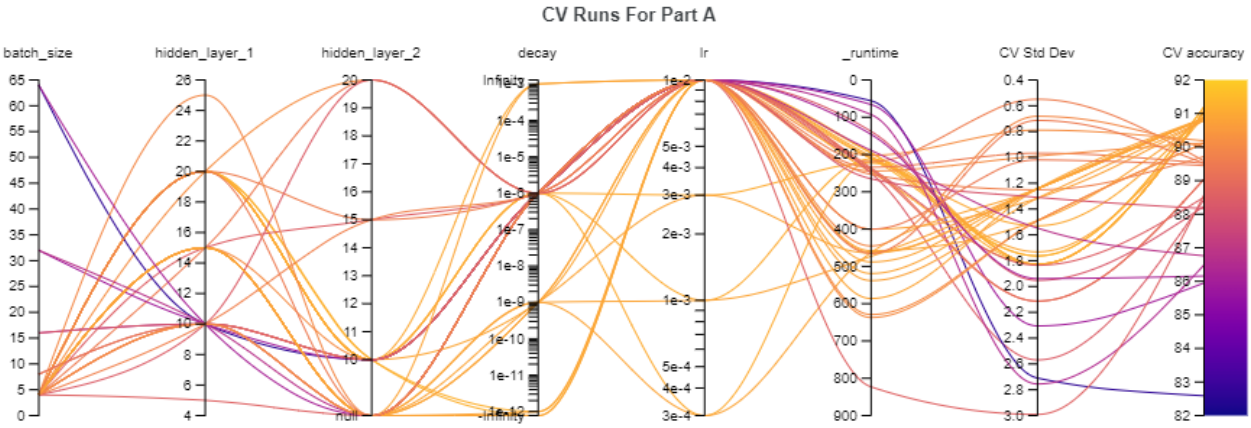


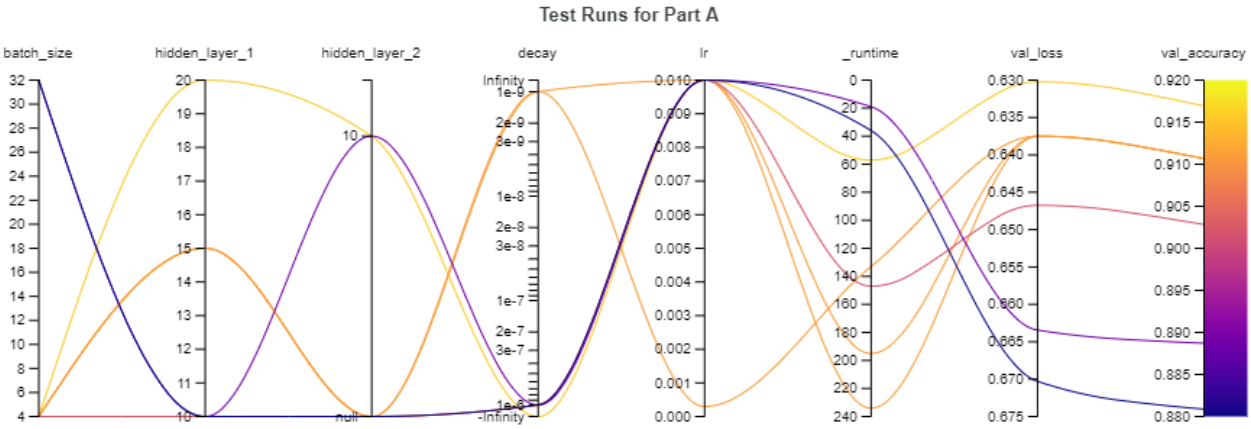
Figure 27: Best Performing Model for Question 3

Dropout is one of the most popular regularization techniques for deep neural networks. By including dropout layers during training, we are able to decrease the test errors and improve the model's performance as shown above in Figure 27 where the loss and mse are lower for the test set as compared to the train set. To improve the model performance further, we can experiment with increasing the dropout rate as increasing the dropout rate for large layers tend to improve performance if the model is overfitting.

Appendix


Test Runs for Part B
dropout_1
dropout_2
dropout_3
val_loss
val_mse
Yes
No


CV Runs For Part A
batch_size
hidden_layer_1
hidden_layer_2
decay
lr
_runtime
CV Std Dev
CV accuracy


Test Runs for Part A
batch_size
hidden_layer_1
hidden_layer_2
decay
lr
_runtime
val_loss
val_accuracy