CZ4042 Assignment 1 Report

Foo Chuan Sheng (U1820713C)

School of Computer Science and
Engineering (SCSE)
Nanyang Technological University
d180001@e.ntu.edu.sg

***Abstract*-** This report provides an overview of all experiment results as well as a summary of the findings for Part A: Classification Problem and Part B: Regression Problem.

***Index Terms*-** Deep Learning, Hyperparameter Tuning, Keras, Neural Networks, TensorFlow

1. INTRODUCTION

This report will examine Part A: Classification Problem and Part B: Regression Problem.

In Part A: Classification Problem, we will be using the GTZAN dataset. We will predict the genre of the corresponding audio files in the test dataset after training the neural network on the training dataset.

In Part B: Regression Problem, we will be using the HDB flat prices in Singapore dataset, obtained from data.gov.sg on 5th August 2021. We will perform retrospective prediction of HDB housing prices and identify the most important features that contributed to the prediction.

2. PART A: CLASSIFICATION PROBLEM

This part of the assignment will be conducted with the CSV file named features_30_sec.csv. Each data sample is a row of 60 columns, which consist of filename, length of audio, genre, and the 57 features which will be used.

A neural network is built to predict the genre of the corresponding audio files in the test dataset after training the neural network on the training dataset. The genres are blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae and rock.

The dataset is divided into a 70:30 ratio for training and testing and appropriate scaling of input features was applied.

A. Question 1

In Question 1, a feedforward deep neural network (DNN) was constructed with the following parameters:

- Architecture
  - 1 input layer of 57 neurons
  - 1 hidden layer of 16 neurons with ReLU activation
  - 1 dropout layer with probability 0.3
  - 1 output layer of 10 neurons with SoftMax activation
- Optimiser: Stochastic gradient descent with 'Adam' optimizer with default parameters
- Loss Function: Sparse categorical cross entropy
- Batch Size: 1
- Epochs: 50

The accuracies on training and test data against training epochs is plotted in Figure 1.

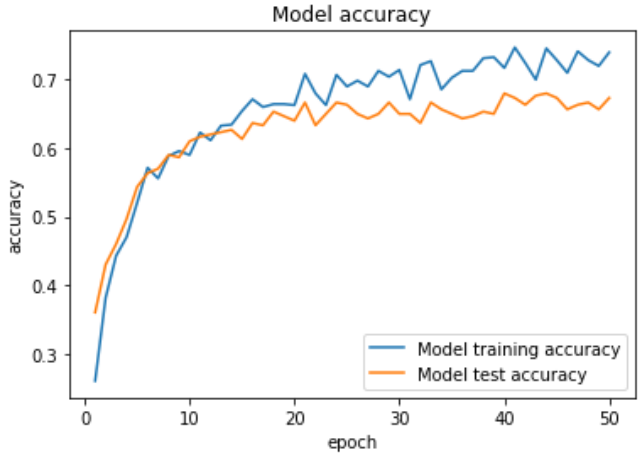

Figure 1: Model accuracy on training and test data against training epochs

As shown in Figure 1, the model's training and test accuracy trends upwards with the number of epochs passed.

The losses on training and test data against training epochs is plotted in Figure 2.

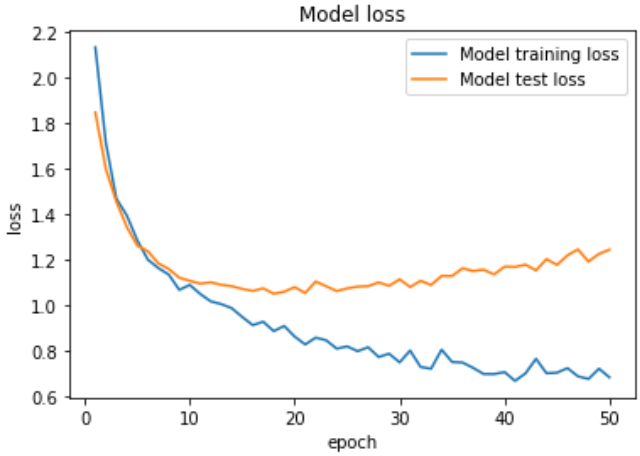

Figure 2: Model loss on training and test data against training epochs

As shown in Figure 2, the model's training loss trends downwards with the number of epochs passed. However, the model's test loss gradually decreases and converges to a minima at around 20 epochs before trending upwards with the number of epochs passed.

B. Question 2

In Question 2, the performance of the model using stochastic gradient descent and mini-batch gradient descent was compared. The optimal batch size for mini-batch gradient descent was determined by training the neural network and evaluating the performances for different batch sizes among {1, 4, 8, 16, 32, 64}.

The methodology used to determine the optimal batch size is by conducting 10 experiments of 3-fold cross-validation. 3-fold cross-validation only uses the training data and will not use the test data. For each experiment, the training data is shuffled before cross-validation partitions the shuffled training data into 3-folds.

The results of conducting 10 experiments of 3-fold cross-validation is first stored in a dictionary_A which is as follows:

```
dictionary_A maps experiment_key
-> batch_size_key
-> num_fold_key
-> epoch_key
-> val_accuracy
{
  "experiment: 0":{
    "batch_size: 1":{
      "num_fold: 0":{
        "epoch: 0":0.1,
        ...,
        "epoch: 99":0.8
      },
      "num_fold: 1":{...},
      "num_fold: 2":{...}
    },
    ...,
    "batch_size: 64":{...}
  },
  ...,
  "experiment: 9":{...}
}
```

Note: To obtain the first epochs' val_accuracy for first fold of cross-validation using batch size 1 in experiment 1 is obtained with the expression:

```
dictionary_A.get("experiment: 0") \
  .get("batch_size: 1") \
  .get("num_fold: 0") \
  .get("epoch: 99")
```

Afterwards, the results are aggregated in two steps. Firstly, to produce a dictionary_B which is as follows:

```
dictionary_B maps experiment_key
-> batch_size_key
-> epoch_key
-> mean_val_accuracy
{
  "experiment: 0":{
    "batch_size: 1":{
      "epoch: 0":0.1,
      ...,
      "epoch: 99":0.8
      }
    },
    ...,
    "batch_size: 64":{...}
  },
  ...,
  "experiment: 9":{...}
}
```

Note: mean_val_accuracy is mean of val_accuracy across the folds for a particular batch_size and particular experiment and particular epoch from dictionary_A.

Secondly, dictionary_B is converted to dictionary_C which is as follows:

```
dictionary_C maps batch_size_key
-> epoch_key
-> mean_of_mean_val_accuracy
{
  "batch_size: 1":{
    "epoch: 0":0.1,
    ...,
    "epoch: 99":0.8
    }
  },
  ...,
  "batch_size: 64":{...}
}
```

Note: mean_of_mean_val_accuracy is mean of mean_val_accuracy across the experiments for a particular batch_size and particular epoch from dictionary_B.

We will use the values inside dictionary_C for evaluating the performances for different batch sizes.

The mean cross-validation accuracies over the training epochs for different batch sizes (from epoch 1) is plotted in Figure 3.

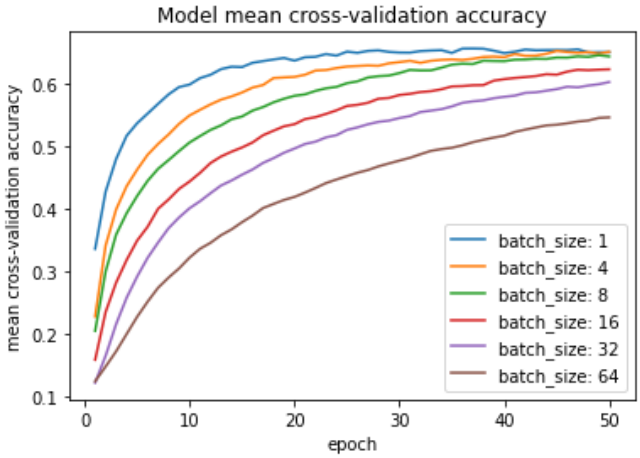

Figure 3: Mean cross-validation accuracies over the training epochs for different batch sizes (from epoch 1)

The mean cross-validation accuracies over the training epochs for different batch sizes (from epoch 40) is plotted in Figure 4.

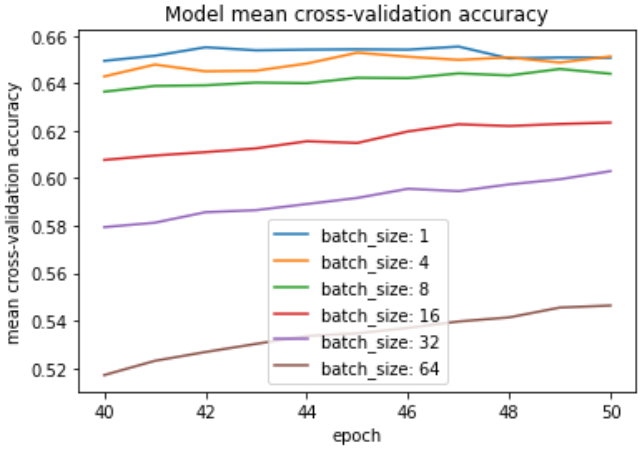

Figure 4: Mean cross-validation accuracies over the training epochs for different batch sizes (from epoch 40)

While conducting the 10 experiments of 3-fold cross-validation, the time taken to train the network for one epoch was also recorded in dictionary_D which is as follows:

```
dictionary_D maps experiment_key
-> batch_size_key
-> num_fold_key
-> time_taken_per_epoch_list
{
  "experiment: 0":{
    "batch_size: 1":{
      "num_fold: 0":[0.7,…,0.6]
      "num_fold: 1":[…],
      "num_fold: 2":[…]
    },
    …,
    "batch_size: 64":{…}
  },
  …,
  "experiment: 9":{…}
}
```

Afterwards, the results are aggregated in one step to produce a dictionary_E which is as follows:

```
dictionary_E maps batch_size_key
-> median_time_taken_per_epoch
{
  "batch_size: 1":0.7649,
  …,
  "batch_size: 64":0.0292
}
```

Note: median_time_taken_per_epoch is median of the values inside combination of time_taken_per_epoch_list across the folds and experiments for a particular batch_size from dictionary_D.

Using, the values inside dictionary_E, the table of median time taken to train the network for one epoch against different batch sizes is provided in Table 1.

Batch Size	Median Time Taken per Epoch (s)
1	0.7649
4	0.1954
8	0.1343
16	0.0663
32	0.0466
64	0.0292

Table 1: Median time taken to train the network for one epoch against different batch sizes

Based on Figure 3 and Figure 4, the optimal batch size selected is 4 because it has a highest mean cross-validation accuracy compared to the other batch sizes at epoch 50. It also has slightly better mean cross-validation accuracy compared to batch size 1 (**0.6514** vs 0.6509).

The main difference between mini-batch gradient descent (GD) and stochastic GD is summarised in Table 2.

Stochastic GD	Mini-Batch GD
Uses only one training data to update weights at each step in one epoch	Uses a fixed number of training examples (smaller than size of training dataset) to update weights at each step in one epoch

Table 2: Difference between Stochastic GD and Mini-Batch GD

As stochastic GD uses only one training data at each step in one epoch, we cannot take advantage of vectorized implementations that are available when using mini-batch GD. Hence, using mini-batch GD allows for faster computations. This observation is supported from results in Table 1 which shows that as batch size increases, the median time taken per epoch decreases.

After optimal batch size 4 is selected, a feedforward deep neural network (DNN) was constructed with the following parameters:

- Architecture
  - 1 input layer of 57 neurons
  - 1 hidden layer of 16 neurons with ReLU activation
  - 1 dropout layer with probability 0.3
  - 1 output layer of 10 neurons with SoftMax activation
- Optimiser: Stochastic gradient descent with 'Adam' optimizer with default parameters
- Loss Function: Sparse categorical cross entropy
- **Batch Size: 4**
- Epochs: 50

The accuracies on training and test data against training epochs is plotted in Figure 5.

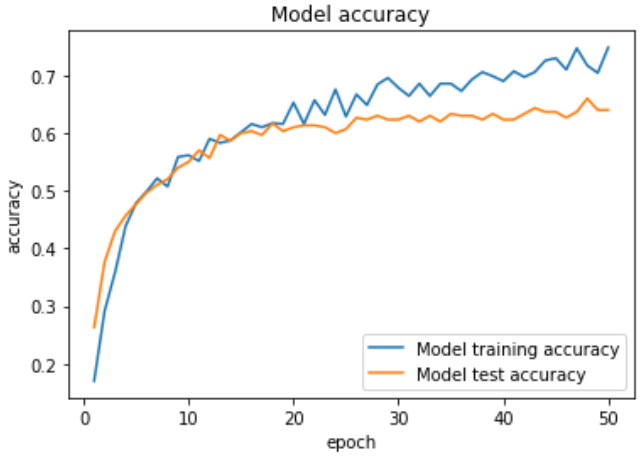

Figure 5: Model accuracy on training and test data against training epochs

C. Question 3

In Question 2, the optimal batch size was determined to be 4. In Question 3, the optimal number of hidden neurons for the 2-layer network was determined by training the neural network and evaluating the performances for different number of hidden neurons among {8, 16, 32, 64}.

The methodology used to determine the optimal number of hidden neurons is the same as the methodology in Question 2. We will obtain a slightly different dictionary_C from using the methodology in Question 2 which is as follows:

```
dictionary_C maps num_hidden_neurons_key
-> epoch_key
-> mean_of_mean_val_accuracy
{
  "num_hidden_neurons_key: 8":{
    "epoch: 0":0.1,
    …,
    "epoch: 99":0.8
    }
  },
  …,
  "num_hidden_neurons_key: 64":{…}
}
```

We will use the values inside dictionary_c for evaluating the performances for different number of hidden neurons.

The mean cross-validation accuracies over the training epochs for different number of hidden neurons (from epoch 1) is plotted in Figure 6.

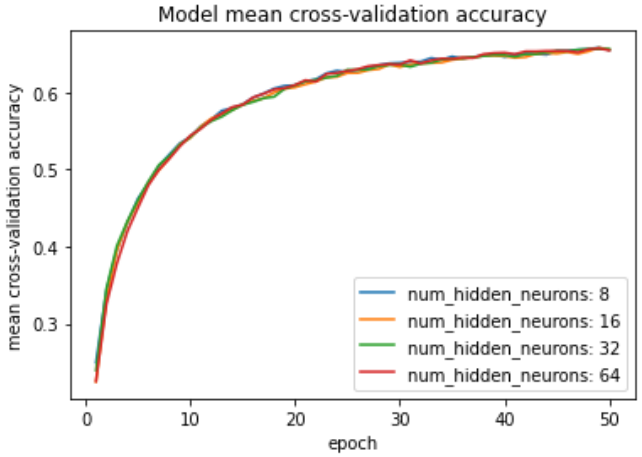

Figure 6: Mean cross-validation accuracies over the training epochs for different number of hidden neurons (from epoch 1)

The mean cross-validation accuracies over the training epochs for different number of hidden neurons (from epoch 40) is plotted in Figure 7.

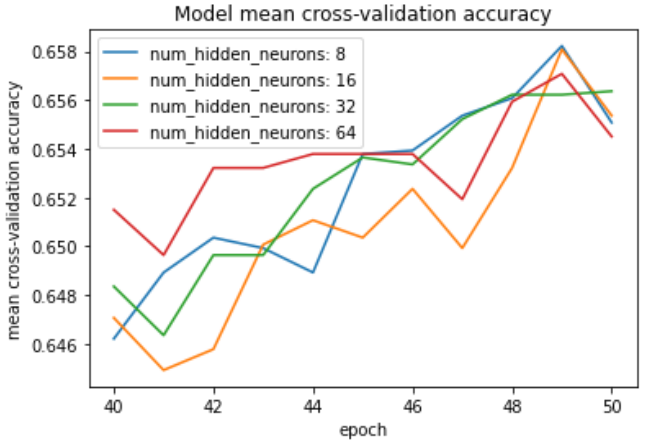

Figure 7: Mean cross-validation accuracies over the training epochs for different number of hidden neurons (from epoch 40)

Based on Figure 6 and Figure 7, the optimal number of hidden neurons selected is 32 because it has a highest mean cross-validation accuracy compared to the other number of hidden neurons at epoch 50.

After optimal number of hidden neurons 32 is selected, a feedforward deep neural network (DNN) was constructed with the following parameters:

- Architecture
  - 1 input layer of 57 neurons
  - **1 hidden layer of 32 neurons with ReLU activation**
  - 1 dropout layer with probability 0.3
  - 1 output layer of 10 neurons with SoftMax activation
- Optimiser: Stochastic gradient descent with 'Adam' optimizer with default parameters
- Loss Function: Sparse categorical cross entropy
- **Batch Size: 4**
- Epochs: 50

The accuracies on training and test data against training epochs is plotted in Figure 8.

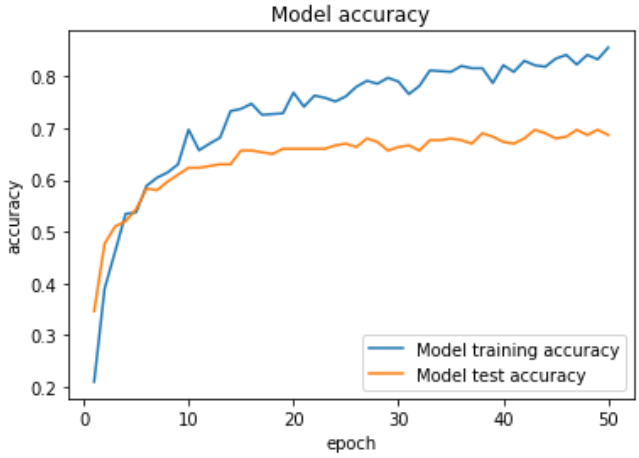

Figure 8: Model accuracy on training and test data against training epochs

Some possible parameters that can be considered for tunning are as follows:

- Dropout rate
- Feature selection to remove unnecessary features

D. Question 4

In Question 3, the optimal number of hidden neurons was determined to be 32. In Question 4, a feedforward deep neural network (DNN) was constructed with the following parameters:

- Architecture
  - 1 input layer of 57 neurons
  - **1 hidden layer of 32 neurons with ReLU activation**
  - 1 dropout layer with probability 0.3
  - **1 hidden layer of 32 neurons with ReLU activation**
  - 1 dropout layer with probability 0.3
  - 1 output layer of 10 neurons with SoftMax activation
- Optimiser: Stochastic gradient descent with 'Adam' optimizer with default parameters
- Loss Function: Sparse categorical cross entropy
- Batch Size: 1
- Epochs: 50

The accuracies on training and test data against training epochs is plotted in Figure 9.

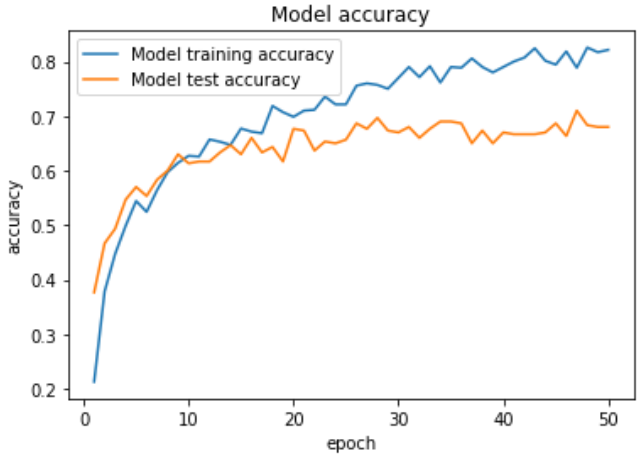

Figure 9: Model accuracy on training and test data against training epochs

The differences between optimal 2-layer network obtained from hyperparameter tuning in Question 2 and 3 and 3-layer network are summarised in Table 3.

Optimal 2-layer network	3-layer network
At epoch 50, the test accuracy is 0.6867	At epoch 50, the test accuracy is 0.6800
At epoch 50, the test loss is 0.9635	At epoch 50, the test loss is 1.1457

Table 3: Differences between optimal 2-layer network and 3-layer network

The optimal 2-layer network has a lower test loss and a higher test accuracy than 3-layer network. Hence, the optimal 2-layer network outperforms the 3-layer network.

One possible reason for the differences in test loss and test accuracy between the optimal 2-layer network and 3-layer network is that the introduction of an additional layer caused the 3-layer network to become overfitted to the training dataset.

E. Question 5

In Question 5, we will investigate the purpose of dropouts by removing dropouts from the original 2-layer network (before changing the batch size and number of neurons) in Question 1. A feedforward deep neural network (DNN) was constructed with the following parameters:

- Architecture
  - 1 input layer of 57 neurons
  - 1 hidden layer of 16 neurons with ReLU activation
  - 1 output layer of 10 neurons with SoftMax activation
- Optimiser: Stochastic gradient descent with 'Adam' optimizer with default parameters
- Loss Function: Sparse categorical cross entropy
- Batch Size: 1
- Epochs: 50

The accuracies on training and test data against training epochs is plotted in Figure 10.

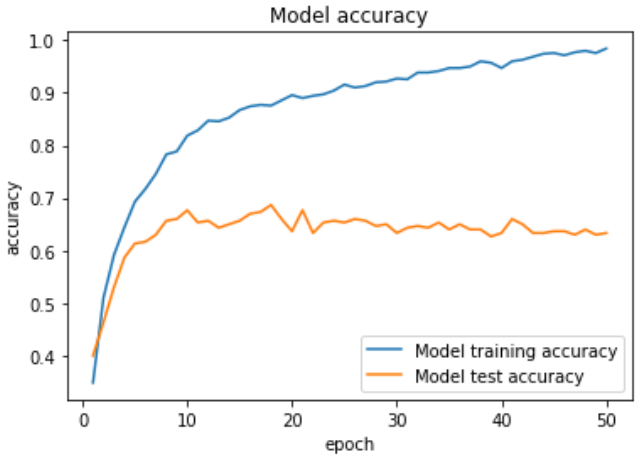

Figure 10: Model accuracy on training and test data against training epochs

The losses on training and test data against training epochs is plotted in Figure 11.

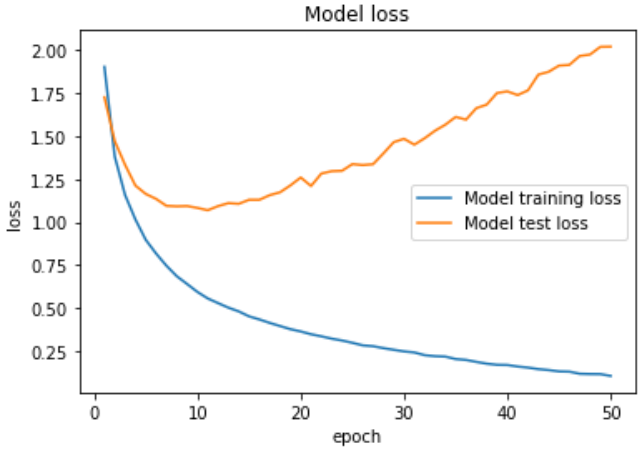

Figure 11: Model loss on training and test data against training epochs

As shown in Figure 10, the model's training accuracy trends upwards with the number of epochs passed. However, the model's test accuracy gradually increases and converges to a maxima at around 15 epochs before trending downwards with the number of epochs passed.

This indicates that the model has become overfitted to the training dataset and is supported by results in Figure 11 which shows that the test loss increases after epoch 10.

The benefits of dropout is that it can help a model reduce overfitting by randomly setting the output for a given neuron to 0. Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs. This conceptualization suggests that perhaps dropout breaks-up situations where network layers co-adapt to correct mistakes from prior layers, in turn making the model more robust. [1]

Another method that we can use to help a model reduce overfitting is by applying regularization. In L1 or L2 regularization, we can add a penalty term on the cost function to push the estimated coefficients towards zero (and not take more extreme values). L2 regularization allows weights to decay towards zero but not to zero, while L1 regularization allows weights to decay to zero.[2]

F. Conclusion

Although we have a classifier that predicts the genre of audio files based on features obtained from processing the audio tracks, there are some limitations of the current approach (using FFNs to model such engineered features). Namely, FFNs have a large number of parameters because of the dense layers which may cause overfitting. In addition, it causes longer convergence times for mode training, higher inference times and an overall larger model size compared to other neural network architectures.

Furthermore, FFNs are also prone to the vanishing and exploding gradient problem. Vanishing gradient refers to

[1] Brownlee, J. (2018, December 3). A Gentle Introduction to Dropout for Regularizing Deep Neural Networks. https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

[2] Chuan, D. (2020, June 7). 8 Simple Techniques to Prevent Overfitting. https://towardsdatascience.com/8-simple-techniques-to-prevent-overfitting-4d443da2ef7d#d178

gradients getting smaller and approaching zero when backpropagation advances from output layer to input layer, which leaves the weights of the initial layers nearly unchanged. Exploding gradient refers to gradients getting larger and larger when backpropagation advances from output layer to input layer, which causes very large weight updates and causes gradient descent to diverge.[3]

Out of the parameters that were tuned, the most impactful in terms of improving the model performance is batch size. This is because setting batch size too high can make the network take too long to achieve convergence (no more gain in accuracy). On the other hand, if it is too low, it will make the network bounce back and forth without achieving acceptable performance.

As that audio tracks are originally waveforms, an alternative approach to perform genre classification would be to use spectrograms (visual pictures of audio tracks) as an input instead. Convolutional Neural Network (CNN) is an image classification algorithm and will be able use spectrograms to perform genre classification.

3. PART B: REGRESSION PROBLEM

This part of the assignment will be conducted with the CSV file named HDB_price_prediction.csv. Each data sample is a row of 13 columns, which consist of year, full address, nearest station, resale price, and the 9 features which will be used.

A neural network is built to perform retrospective prediction of HDB housing prices (represented by the resale price column) after training the neural network on the training dataset. The most important features that contributed to the prediction will also be identified.

The dataset is divided into training and test datasets by using entries from year 2020 and before as training dataset (with the remaining data from year 2021 used as test dataset). The dataset is split in this manner because the resale price of a unit is dependent on year (although year is not an input feature for our model training).

Hence, we will use year 2020 and before training data to train a model to predict resale price. Our model's performance will be evaluated using 2021 data to check whether it can generalise well to datapoints in year 2021 and later.

A. Question 1

In Question 1, a feedforward deep neural network (DNN) was constructed with the following parameters:

- Architecture
  - 1 input layer of 81 neurons
  - 1 hidden layer of 10 neurons with ReLU activation
  - 1 output layer of 1 neuron with Linear activation
- Optimiser: Stochastic gradient descent with 'Adam' optimizer with learning rate 0.05
- Loss Function: Mean square error
- Batch Size: 128
- Epochs: 100

The input layer of 81 neurons is a concatenation of the following features:

- Categorical features:
  - month
  - flat_model_type
  - storey_range
- Numeric features:
  - dist_to_nearest_stn
  - dist_to_dhoby
  - degree_centrality
  - eigenvector_centrality
  - remaining_lease_years
  - floor_area_sqm

One-hot encoding was applied to all categorical features using the function encode_categorical_feature provided while all numeric features were standardised using the function encode_numerical_feature provided.

The model architecture of the resulting model is shown in Figure 13 on the next page.

The root mean square errors (RMSE) on training and test data against training epochs (from epoch 5) is plotted in Figure 12.

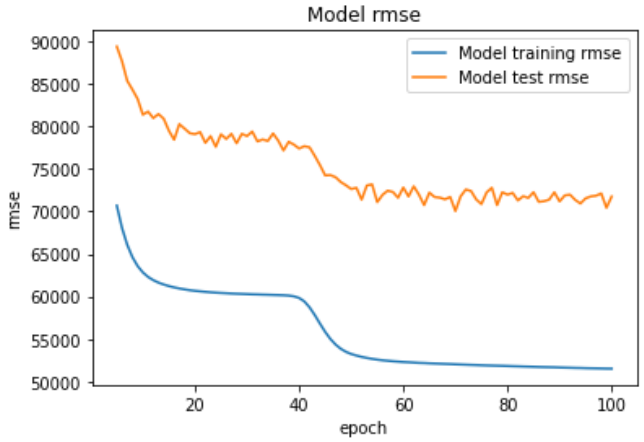

Figure 12: RMSE on training and test data against training epochs (from epoch 5)

The lowest test loss (mean squared error of 4905354240.0) occurs at epoch 70. The corresponding test R^2 value at epoch 70 is 0.8074.

By restoring the model weights from epoch 70 via a callback, the predicted values and target values for a batch of 128 test samples are plotted in Figure 14 on the next page.

[3] Bohra, Y. (2021, June 18). The Challenge of Vanishing/Exploding Gradients in Deep Neural Networks. https://www.analyticsvidhya.com/blog/2021/06/the-challenge-of-vanishing-exploding-gradients-in-deep-neural-networks/

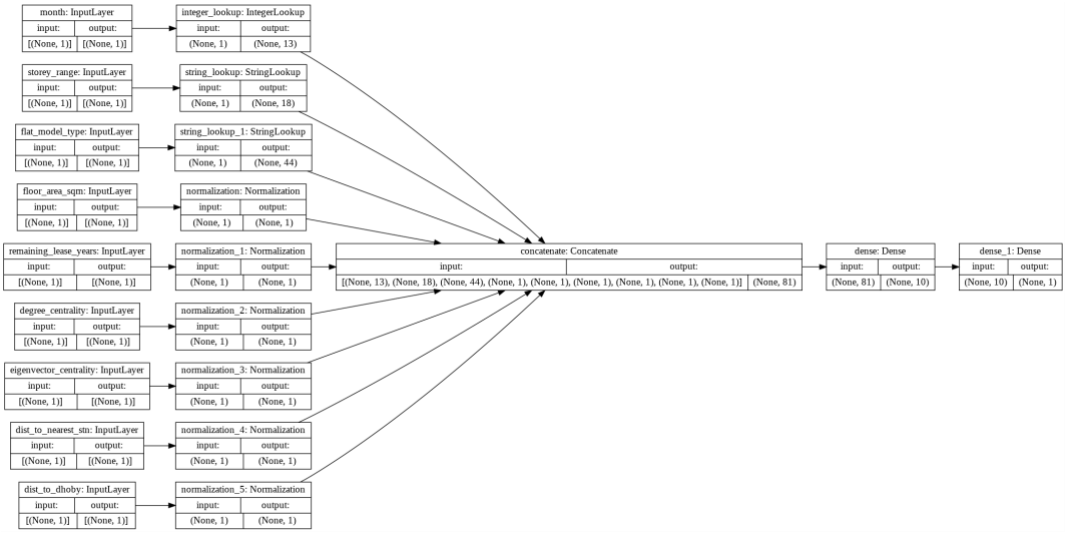

Figure 13: Model architecture

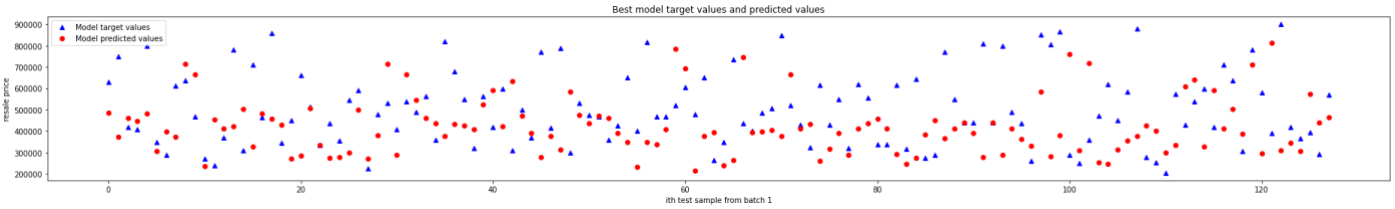

Figure 14: Best model target values and predicted values for a batch of 128 test samples

B. Question 2

In Question 2, instead of using one-hot encoding, an alternative approach of using embeddings to encode categorial variables was employed.

A new function modified_encode_categorical_feature was created which uses output_mode='int'. It converts inputs to their index in the vocabulary (which we shall call integer encoding). Integer encoding was applied to all categorical features using the function modified_encode_categorical_feature while all numeric features were standardised using the function encode_numerical_feature provided.

After applying integer encoding to the categorical features, each integer encoded categorical feature was passed to a separate Embedding layer. The output dimension of each Embedding layer is equal to the floor of num_categories/2.

As the Embedding layer produces a 2D output (3D, including batch), the output of Embedding layer is flattened to 1D output (2D, including batch) so that it can be concatenated with the numeric features. After concatenation of the categorical and numeric features, an input layer of 41 neurons was obtained.

A feedforward deep neural network (DNN) was constructed with the following parameters:

- Architecture
  - **1 input layer of 41 neurons**
  - 1 hidden layer of 10 neurons with ReLU activation
  - 1 output layer of 1 neuron with Linear activation
- Optimiser: Stochastic gradient descent with 'Adam' optimizer with learning rate 0.05
- Loss Function: Mean square error
- Batch Size: 128
- Epochs: 100

The model architecture of the new resulting model is shown in Figure 16 on the next page.

The root mean square errors (RMSE) on training and test data against training epochs (from epoch 5) is plotted in Figure 15.

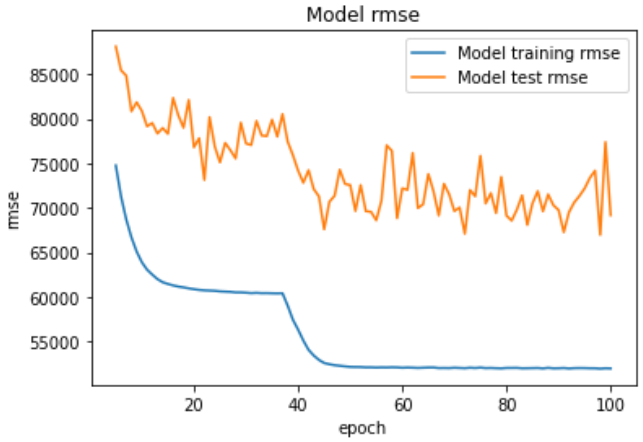

Figure 15: RMSE on training and test data against training epochs (from epoch 5)

The differences between the models from Question 1 and 2 are summarised in Table 4.

Question 1 Model	Question 2 Model
Lowest test MSE is 4905354240.0 at epoch 70	Lowest test MSE is 4484649472.0 at epoch 98
At epoch 70, the test RMSE is 70038.23	At epoch 98, the test RMSE is 66967.53
At epoch 70, the test R^2 value is 0.8074	At epoch 98, the test R^2 value is 0.8241

Table 4: Differences between models from Question 1 and 2

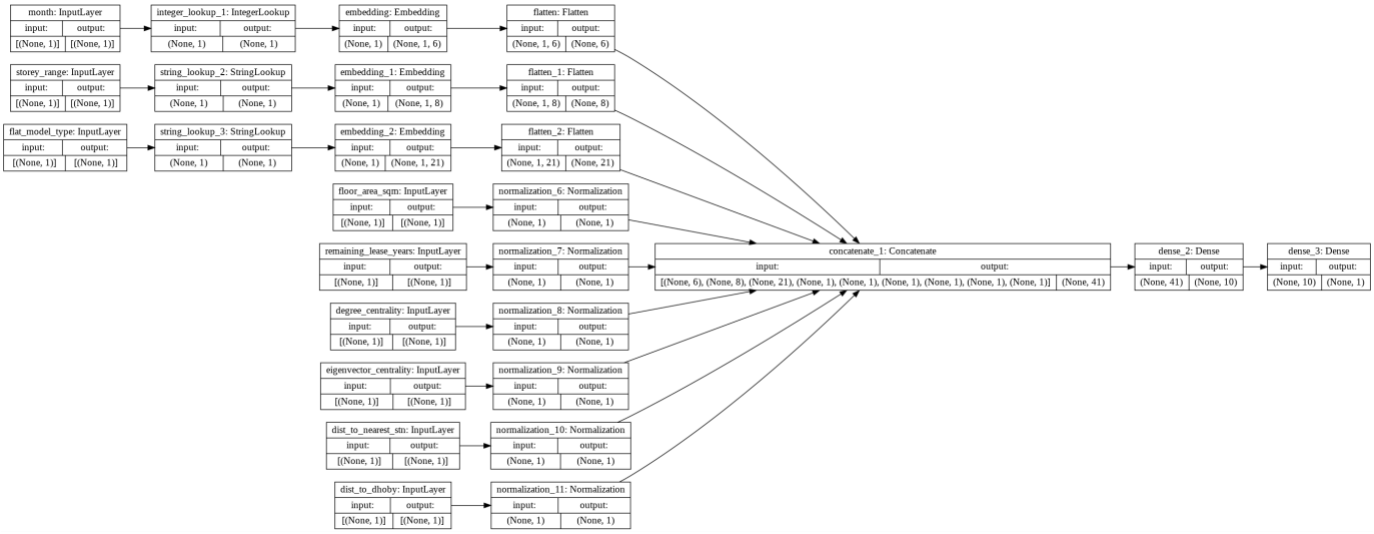

Figure 16: Model architecture

From the results in Table 4, we can see that Question 2 Model has a lower best test RMSE and a higher best test R^2. A possible reason for the difference in performance is that the embeddings present in Question 2 model are able to represent words as semantically-meaningful dense real-valued vectors.

The use of embeddings overcomes a primary problem that one-hot vector encodings have which is the vocabulary size issue. The vocabulary size issue states that an increase in vocabulary by n, will cause feature size vectors to also increase by length n. Hence, the increase in feature size vectors means that the model will contain more parameters that might lead to overfitting during training and not be able to generalise well.[4]

C. Question 3

In Question 3, recursive feature elimination (RFE) was used to remove unnecessary features from the inputs.

The model architecture from Question 2 was first improved by introducing early stopping (based on test loss) with patience of 10 epochs. A feedforward deep neural network (DNN) was constructed with the following parameters:

- Architecture
  - 1 input layer of 41 neurons
  - 1 hidden layer of 10 neurons with ReLU activation
  - 1 output layer of 1 neuron with Linear activation
- Optimiser: Stochastic gradient descent with 'Adam' optimizer with learning rate 0.05
- Loss Function: Mean square error
- Batch Size: 128
- Epochs: 100
- **Early Stopping Patience Epochs: 10**

The model was then trained to demonstrate the effectiveness of early stopping in reducing the training duration. The early stopping behaviour is shown in Figure 17.

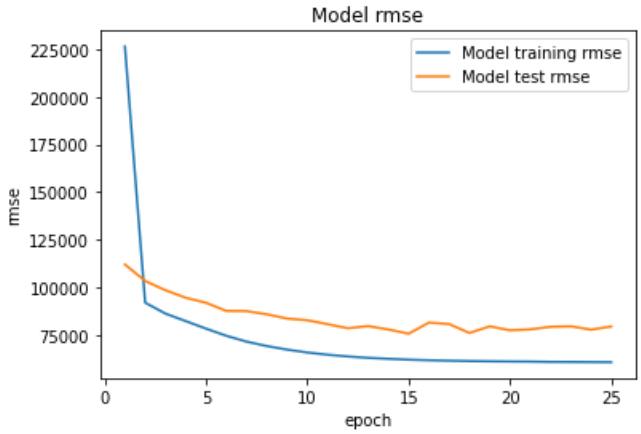

Figure 17: Early stopping behaviour

RFE was performed on the model by first removing one input feature whose removal provided the lowest test loss. The procedure was repeated recursively on the reduced input set until there is only one feature left. The RFE results are stored in a dictionary_A which is as follows:

```
dictionary_A maps num_features_key
-> features_mask_key
-> (test_loss,test_R2)
{
  "num_features_key: 1":{
    "features_mask_key: 001000000": (200.0, -0.065)}
  },
  ...,
  "num_features_key: 9":{...}
}
```

Note: The test_loss refers to the lowest test_loss among all training epochs. The test_R2 refers to the R2 value at the training epoch with lowest test_loss.

A features_mask_key was used to represent which feature was removed when running RFE. The index in features_mask_key is the feature it represents and the value is 0 if the feature is not used while 1 if the feature is used. The mapping between index and feature for features_mask_key is shown in Table 5 on the next page.

[4] Latysheva, N. (2019, September 10). Why do we use word embeddings in NLP? https://towardsdatascience.com/why-do-we-use-embeddings-in-nlp-2f20e1b632d2

Index	Feature
0	month
1	storey_range
2	flat_model_type
3	floor_area_sqm
4	remaining_lease_years
5	degree_centrality
6	eigenvector_centrality
7	dist_to_nearest_stn
8	dist_to_dhoby

Table 5: Mapping between index and input feature for features_mask_key

The RFE results obtained are summarised in Table 6.

From Table 6, we can see that the best model arrived at by RFE is features_mask_key 111111101. It has a RMSE of 66159.16 and R2 of 0.8281. Meanwhile, the baseline model with features_mask_key 111111111 has a RMSE of 75616.49 and R2 of 0.7779.

The order in which features were removed indicate the usefulness of each feature from least important to most important in predicting HDB resale prices. They are as follows:

1. dist_to_nearest_stn
2. eigenvector_centrality
3. month
4. degree_centrality
5. floor_area_sqm
6. storey_range
7. remaining_lease_years
8. dist_to_dhoby

The last feature not removed is flat_model_type and indicates that it is the most important feature for HDB resale price prediction.

D. Conclusion

From RFE, we have determined which features were (un)important when performing the prediction. We can make use of this information to find out what could be the factors that lead to the price increase by looking at similar datapoints with only a few input features different from each other. We then check if the datapoints vary significantly in price. If the price varies significantly, it means that the features are important factors that will lead to price increase.

num_features	features_mask_key	test_loss	test_R2
9	**111111111**	**5717853184.0**	**0.7779**
8	011111111	5197023232.0	0.7982
	101111111	5965175296.0	0.7678
	110111111	273647730688.0	-9.7067
	111011111	5546246656.0	0.7846
	111101111	6078417408.0	0.7628
	111110111	5050791936.0	0.8037
	111111011	4404904448.0	0.8285
	111111101	**4377034752.0**	**0.8281**
	111111110	8134453248.0	0.6830
7	011111101	5856390144.0	0.7711
	101111101	6566730752.0	0.7453
	110111101	6573117440.0	0.7445
	111011101	6080544256.0	0.7631
	111101101	6143792640.0	0.7603
	111110101	5955041280.0	0.7679
	111111001	5687175168.0	0.7782
	111111100	9053798400.0	0.6482
6	011111001	5718942208.0	0.7765
	101111001	6695179264.0	0.7401
	110111001	6978765312.0	0.7299
	111011001	5759996928.0	0.7768
	111101001	6356366336.0	0.7524
	111110001	6193245184.0	0.7584
	111111000	9926458368.0	0.6149
5	001111001	6302439424.0	0.7547
	010111001	6978765312.0	0.7395
	011011001	6057265152.0	0.7641
	011101001	5968362496.0	0.7679
	011110001	5348214272.0	0.7914
	011111000	9706788864.0	0.6244
4	001110001	6618103808.0	0.7437
	010110001	6273459712.0	0.7553
	011010001	6225031168.0	0.7583
	011100001	6579874816.0	0.7440
	011110000	10525547520.0	0.5907
3	001010001	5574002688.0	0.7833
	010010001	19766734848.0	0.2300
	011000001	5951908352.0	0.7675
	011010000	10191127552.0	0.6038
2	000010001	20246769664.0	0.2186
	001000001	7355895296.0	0.7132
	001010000	12112607232.0	0.5331
1	000000001	27541968896.0	-0.0650
	001000000	12734301184.0	0.5080

Table 6: RFE results (green highlight indicates best features_mask_key among same num_features)