

## CIS 662 HW4 Report: NN

### PART 1: Evaluation of different Learning Rates and Optimisers for Neural Network:

Dataset selected is **71-80.csv**

Data has also been split in the same way as the previous homework 80:20 to compare the results.

Seed for maintaining the same consistent and best results throughout the training is 3 and can be set by **keras.utils.set\_random\_seed(3)**

The requirement is to predict cit\_2022 values with the features values between cit\_2017 to cit\_2021 and the target features are quantitative. Hence, I have used a **simple neural network for regression task**.

Preferably, logistic regression is used for classification (binary classification) as the exact values are to be predicted, however due to continuous numerical values I have selected created a simple neural network for regression and used **linear activation function in the target layer as there is one target variable in the output layer** for the requirement mentioned above.

I created a Sequential model using Keras. This model is a feedforward neural network, and it's designed for supervised learning tasks, which involve backpropagation.

**Backpropagation** is implicitly included in the code. Backpropagation is a key component of training neural networks, and it's handled automatically when you use deep learning frameworks like Keras or TensorFlow.

However, for the hidden layer, the activation function I have selected is **RELU** as it introduces non-linearity in the hidden layer and can capture complex patterns within the data. This layer represents a set of weights and biases that will be learned during training. The initial weights are typically random, and backpropagation is used to adjust them iteratively to minimize the loss function during training.

**I have tried different learning rates for hyperparameter tuning such as**  
**learning\_rates = [0.001,0.0001,0.00001,0.000001,0.0000001]**

Also, I have experimented with 2 prominent optimizers such as **Adam** and **SGD** relevant for the neural network model task. The optimizer is responsible for updating the weights of the neural network during training. The optimization process, which includes backpropagation, will minimize the loss function defined for your task (not shown in the code) by iteratively adjusting the weights.

The output for the same is attached below:

**OUTPUT:****SGD (Based on different learning rates) Code [with SGD] ;**

```
# Create a Sequential model
model = Sequential()

# Add a hidden layer with 3 neurons and ReLU activation
model.add(Dense(3, input_dim=5, activation='relu', kernel_regularizer=l2(0.01)))

# Add the output layer with 1 neuron (for regression task)
model.add(Dense(1, activation='linear')) # Linear activation for regression

# Choose an optimizer with the current learning rate
custom_optimizer = SGD(learning_rate=learning_rate)

# Compile the model with MSE Loss and the selected optimizer
model.compile(loss='mean_squared_error', optimizer=custom_optimizer)
```

```
1/1 [=====] - 0s 20ms/step - loss: 0.3373
Learning Rate: 0.001, Test Loss: 0.3373
1/1 [=====] - 0s 22ms/step - loss: 0.1614
Learning Rate: 0.0001, Test Loss: 0.1614
1/1 [=====] - 0s 27ms/step - loss: 0.2765
Learning Rate: 1e-05, Test Loss: 0.2765
1/1 [=====] - 0s 34ms/step - loss: 0.1311
Learning Rate: 1e-06, Test Loss: 0.1311
1/1 [=====] - 0s 21ms/step - loss: 0.2102
Learning Rate: 1e-07, Test Loss: 0.2102
```

**Adam (Based on different learning rates):**

```
# Create a Sequential model
model = Sequential()

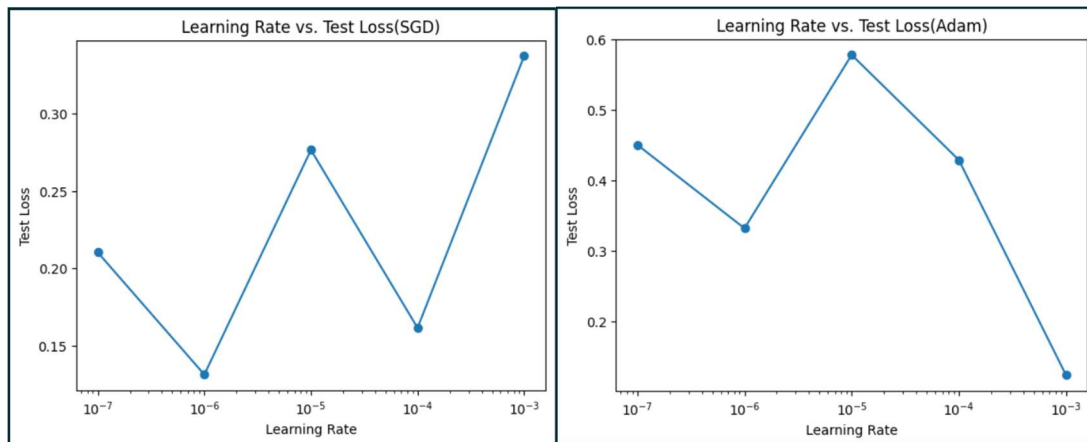
# Add a hidden layer with 3 neurons and ReLU activation
model.add(Dense(3, input_dim=5, activation='relu', kernel_regularizer=l2(0.01)))

# Add the output layer with 1 neuron (for regression task)
model.add(Dense(1, activation='linear')) # Linear activation for regression

# Choose an optimizer with the current learning rate
custom_optimizer = Adam(learning_rate=learning_rate)

# Compile the model with MSE Loss and the selected optimizer
model.compile(loss='mean_squared_error', optimizer=custom_optimizer)
```

```
1/1 [=====] - 0s 23ms/step - loss: 0.1240
Learning Rate: 0.001, Test Loss: 0.1240
1/1 [=====] - 0s 19ms/step - loss: 0.4290
Learning Rate: 0.0001, Test Loss: 0.4290
1/1 [=====] - 0s 20ms/step - loss: 0.5781
Learning Rate: 1e-05, Test Loss: 0.5781
1/1 [=====] - 0s 32ms/step - loss: 0.3322
Learning Rate: 1e-06, Test Loss: 0.3322
1/1 [=====] - 0s 18ms/step - loss: 0.4506
Learning Rate: 1e-07, Test Loss: 0.4506
```

**OUTPUT PLOT (using matplotlib):****CONCLUSION:**

As attached in the screenshots above, the lowest Test loss can be seen with the learning rate **0.001 for Adam optimizer with a loss of 0.1211 for 100 epochs and batch size of 32**. For learning rate of 0.001, using SGD optimizer model has a test loss of 0.3373 and has the lowest loss for 0.000001. However, we know that **SGD converges** slowly and might require more iterations to reach convergence. Since the model with **Adam optimizer has a lower loss with 100 epochs and it converges quickly and accurately, I have selected the Adam optimizer for training the model.**

**PART 2: Evaluation NN Results with the results of HW3 (Clustering Algorithm):**

As mentioned above, **0.001** has lowest mse (loss) using **Adam** optimizer hence, I have assigned the learning rate of 0.001 for model training.

5-3-1 model has 5 input nodes and 3 hidden nodes and 1 output node.

**Mean Square Error** is used to compute the loss for the regression neural network model. Since input dataset needs to be scaled to effectively compute the prediction results for neural network, I have scaled the X(input variable) as well as the Y(output variable) using Standard Scaler library.

On obtaining the predicted values, I have used the **inverse\_transform** library to revert a transformation that was previously applied to the data during preprocessing for calculating the difference between real and predicted values.

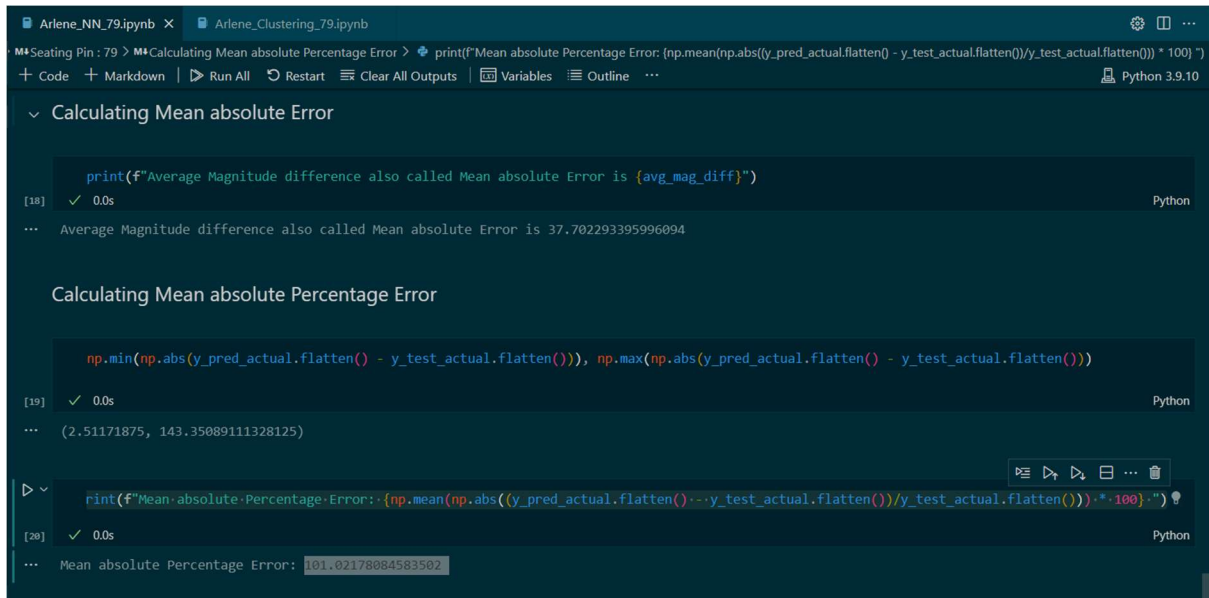
I have considered MAPE and MAE for analysis of the neural network results as well as the Clustering results.

$$\text{MAE} = (1/n) * \sum |(\text{Actual} - \text{Predicted})|$$

$$\text{MAPE} = (1/n) * \sum |(\text{Actual} - \text{Predicted}) / \text{Actual}| * 100\%$$

## OUTPUT:

### Code snippet for NN (Both MAE and MAPE):



```
Arlene_NN_79.ipynb x Arlene_Clustering_79.ipynb
M Seating Pin : 79 > M Calculating Mean absolute Percentage Error > print(f"Mean absolute Percentage Error: (np.mean(np.abs(y_pred_actual.flatten() - y_test_actual.flatten()))/y_test_actual.flatten()) * 100) ")
+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ... Python 3.9.10

v Calculating Mean absolute Error

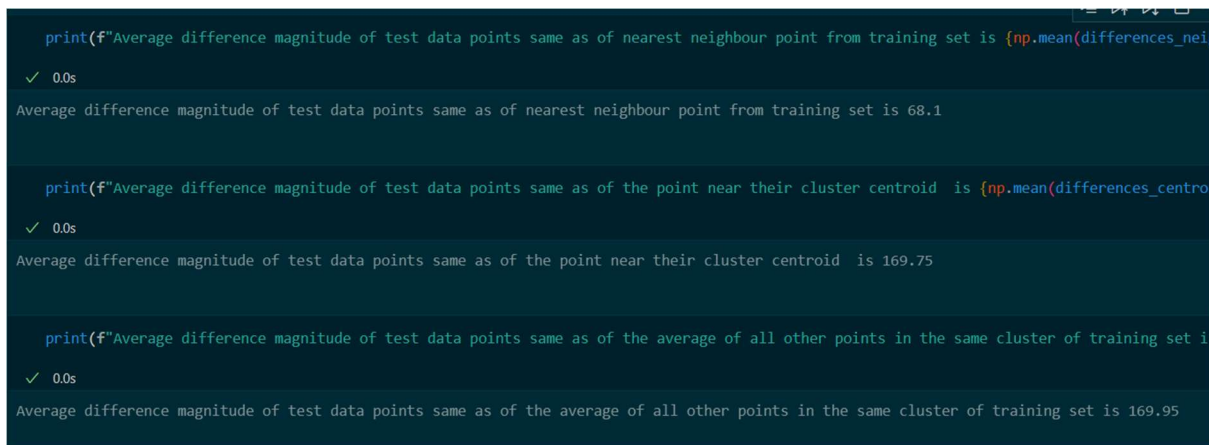
print(f"Average Magnitude difference also called Mean absolute Error is {avg_mag_diff}")
[18] ✓ 0.0s Python
... Average Magnitude difference also called Mean absolute Error is 37.702293395996094

Calculating Mean absolute Percentage Error

np.min(np.abs(y_pred_actual.flatten() - y_test_actual.flatten()), np.max(np.abs(y_pred_actual.flatten() - y_test_actual.flatten())))
[19] ✓ 0.0s Python
... (2.51171875, 143.35089111328125)

print(f"Mean absolute Percentage Error: {(np.mean(np.abs((y_pred_actual.flatten() - y_test_actual.flatten()))/y_test_actual.flatten())) * 100}")
[20] ✓ 0.0s Python
... Mean absolute Percentage Error: 101.02178084583502
```

### Code snippet for Kmeans (MAE):



```
print(f"Average difference magnitude of test data points same as of nearest neighbour point from training set is {np.mean(differences_nei)}")
✓ 0.0s
Average difference magnitude of test data points same as of nearest neighbour point from training set is 68.1

print(f"Average difference magnitude of test data points same as of the point near their cluster centroid is {np.mean(differences_centroid)}")
✓ 0.0s
Average difference magnitude of test data points same as of the point near their cluster centroid is 169.75

print(f"Average difference magnitude of test data points same as of the average of all other points in the same cluster of training set is {np.mean(differences_avg)}")
✓ 0.0s
Average difference magnitude of test data points same as of the average of all other points in the same cluster of training set is 169.95
```

**Code snippet for Kmeans (MAPE):**

```
print(f"Mean absolute Percentage Error: {np.mean(np.abs((predictions_neighbor - y_test_values)/y_test_values)) * 100} ")
✓ 0.0s
Mean absolute Percentage Error: 46.66168962301542

print(f"Mean absolute Percentage Error: {np.mean(np.abs((predictions_centroid - y_test_values)/y_test_values)) * 100} ")
✓ 0.0s
Mean absolute Percentage Error: 389.8026989193447

print(f"Mean absolute Percentage Error: {np.mean(np.abs((predictions_average - y_test_values)/y_test_values)) * 100} ")
✓ 0.0s
Mean absolute Percentage Error: 392.0996205101294
```

**MAE (Absolute Error) :**

Clustering Prediction		Simple NN for Regression
Nearest Neighbour pred	<b>68.1 units</b>	<b>37.70 units</b>
Centroid Values pred	<b>169.75 units</b>	
Avg of values in cluster	<b>169.95 units</b>	

**MAPE (Relative Error) :**

Clustering Prediction		Simple NN for Regression
Nearest Neighbour pred	<b>46.66 units</b>	<b>101.021 units</b>
Centroid Values pred	<b>389.80 units</b>	
Avg of values in cluster	<b>392.099 units</b>	

**CONCLUSION:**

As shown in the output, **MAE is lower in NN as compared to Clustering, but MAPE is comparatively higher than the nearest neighbour Clustering prediction.** This signifies that the predicted value is far from the actual value as a proportion of the actual value. However, **predictions are closer to the actual values without considering the direction of errors.**

This could be the case because my trained model tends to make relatively small absolute errors for large actual values but relatively large absolute errors for small actual values.

The dataset is relatively small and Neural Network captures **high variance(overfit)** in data as opposed to Clustering algorithms which capture **high bias** in data. Hence, for such data, there is a possibility that NN might not perform well in such limited data where data is distributed widely and contains outliers.

Also, data is locally clustered hence nearest neighbour Kmeans clustering algorithm performed well on the above dataset.

Name: Arlene Antony D'costa

Seating Pin: 79

SUID: 594303899

I have saved the model and can be accessed directly for the prediction as can be seen in my attached code.