

# Introduction to ML HW-04

## Introduction:

I tried a 1-hidden layer **neural network (5-3-1 architecture)** using the back-propagation algorithm and predicted the values of cit\_2022 based on all the 2017 to 2021 citations. I have used Tensor flow , keras and Relu activation functions on both the hidden and output nodes.  
Utilised Stochastic gradient descent (SGD) optimization technique of gradient descent.

## Steps:

1. Imported the important and useful libraries.
2. Loaded the data set into the python notebook using pandas.
3. Explored and analysed the data.
4. Splitting the data into training (80 %) and testing (20 %) set using sklearn.
5. Normalising both the train and test data.
6. Defining the neural network 5-3-1 architecture using TensorFlow.
7. Defining hyper parameters i.e learning rate and epochs.
8. Calculate the Mean Absolute Error for the predicted and tested value.

## Result:

Epoch	Learning rate	Output
1000	0.1	37.75960168838501
1000	0.01	50.229731702804564
1000	0.001	180.7489749908447
1000	0.6	122.49656753540039
2000	0.1	37.766401863098146

**Output:**

1. For epoch: 1000, learning rate = 0.1:

```

learning_rate = 0.1
model.compile( tf.keras.optimizers.SGD(learning_rate), loss = 'mean_absolute_error')
model.fit(x_train, y_train, epochs = 1000 , shuffle = False)

Epoch 127/1000
3/3 [=====] - 0s 863us/step - loss: 126.6463
Epoch 128/1000
3/3 [=====] - 0s 1ms/step - loss: 135.6426
Epoch 129/1000
3/3 [=====] - 0s 799us/step - loss: 131.5548
Epoch 130/1000
3/3 [=====] - 0s 858us/step - loss: 139.7656
Epoch 131/1000
3/3 [=====] - 0s 1ms/step - loss: 136.3538
Epoch 132/1000
3/3 [=====] - 0s 815us/step - loss: 126.3549
Epoch 133/1000
3/3 [=====] - 0s 743us/step - loss: 135.3015
Epoch 134/1000
3/3 [=====] - 0s 780us/step - loss: 128.6892
Epoch 135/1000
3/3 [=====] - 0s 862us/step - loss: 140.3808
Epoch 136/1000
3/3 [=====] - 0s 839us/step - loss: 136.6335
Epoch 137/1000

In [26]: # Calculate MAE

from sklearn.metrics import mean_absolute_error
y_pred = model.predict(x_test)
mean_absolute_error = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error (MAE): {mean_absolute_error}")

1/1 [=====] - 0s 25ms/step
Mean Absolute Error (MAE): 37.75960168838501

```

2. For epoch: 1000, learning rate = 0.01:

```

# defining hyper parameters:
learning_rate = 0.01
model.compile( tf.keras.optimizers.SGD(learning_rate), loss = 'mean_absolute_error')
model.fit(x_train, y_train, epochs = 1000 , shuffle = False)

Epoch 46/1000
3/3 [=====] - 0s 843us/step - loss: 403.8535
Epoch 47/1000
3/3 [=====] - 0s 787us/step - loss: 403.6339
Epoch 48/1000
3/3 [=====] - 0s 980us/step - loss: 403.4019
Epoch 49/1000
3/3 [=====] - 0s 2ms/step - loss: 403.1567
Epoch 50/1000
3/3 [=====] - 0s 824us/step - loss: 402.8976
Epoch 51/1000
3/3 [=====] - 0s 893us/step - loss: 402.6235
Epoch 52/1000
3/3 [=====] - 0s 2ms/step - loss: 402.3335
Epoch 53/1000
3/3 [=====] - 0s 873us/step - loss: 402.0266
Epoch 54/1000
3/3 [=====] - 0s 832us/step - loss: 401.7016
Epoch 55/1000
3/3 [=====] - 0s 768us/step - loss: 401.3574
Epoch 56/1000

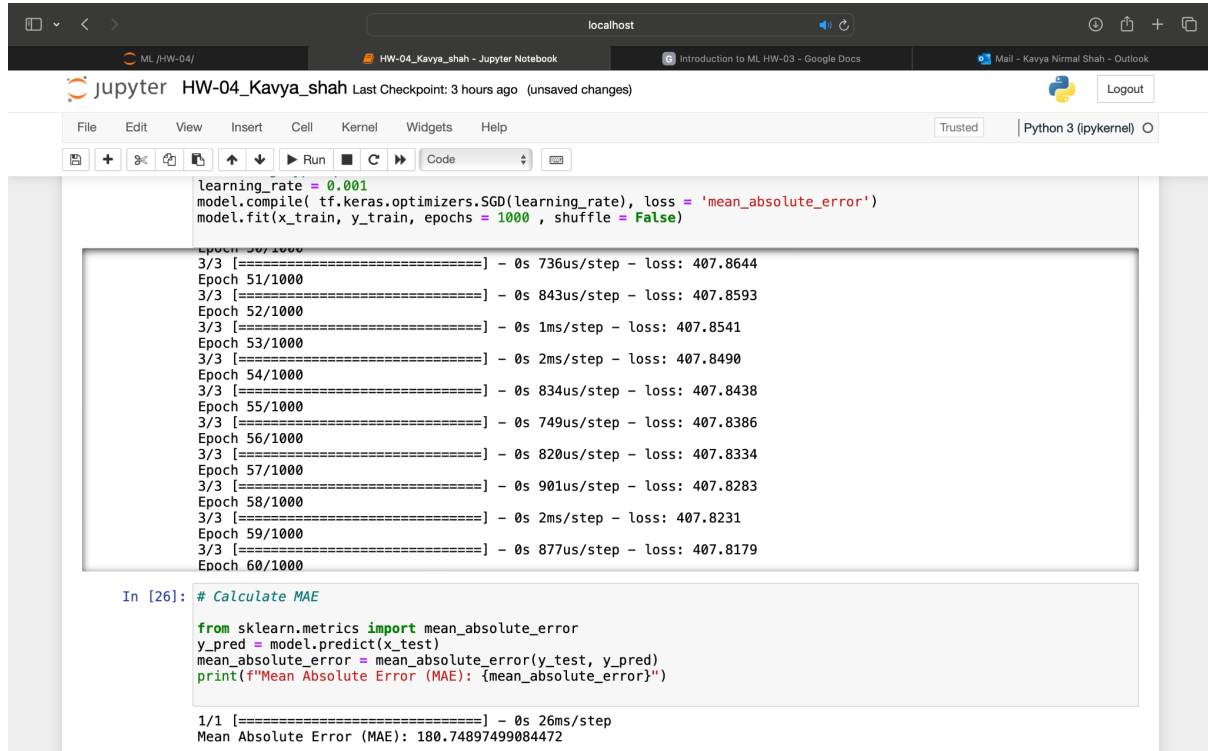
In [26]: # Calculate MAE

from sklearn.metrics import mean_absolute_error
y_pred = model.predict(x_test)
mean_absolute_error = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error (MAE): {mean_absolute_error}")

1/1 [=====] - 0s 27ms/step
Mean Absolute Error (MAE): 50.229731702804564

```

3. For epoch: 1000, learning rate = 0.001:



```

learning_rate = 0.001
model.compile( tf.keras.optimizers.SGD(learning_rate), loss = 'mean_absolute_error')
model.fit(x_train, y_train, epochs = 1000 , shuffle = False)

Epoch 50/1000
3/3 [=====] - 0s 736us/step - loss: 407.8644
Epoch 51/1000
3/3 [=====] - 0s 843us/step - loss: 407.8593
Epoch 52/1000
3/3 [=====] - 0s 1ms/step - loss: 407.8541
Epoch 53/1000
3/3 [=====] - 0s 2ms/step - loss: 407.8490
Epoch 54/1000
3/3 [=====] - 0s 834us/step - loss: 407.8438
Epoch 55/1000
3/3 [=====] - 0s 749us/step - loss: 407.8386
Epoch 56/1000
3/3 [=====] - 0s 820us/step - loss: 407.8334
Epoch 57/1000
3/3 [=====] - 0s 901us/step - loss: 407.8283
Epoch 58/1000
3/3 [=====] - 0s 2ms/step - loss: 407.8231
Epoch 59/1000
3/3 [=====] - 0s 877us/step - loss: 407.8179
Epoch 60/1000

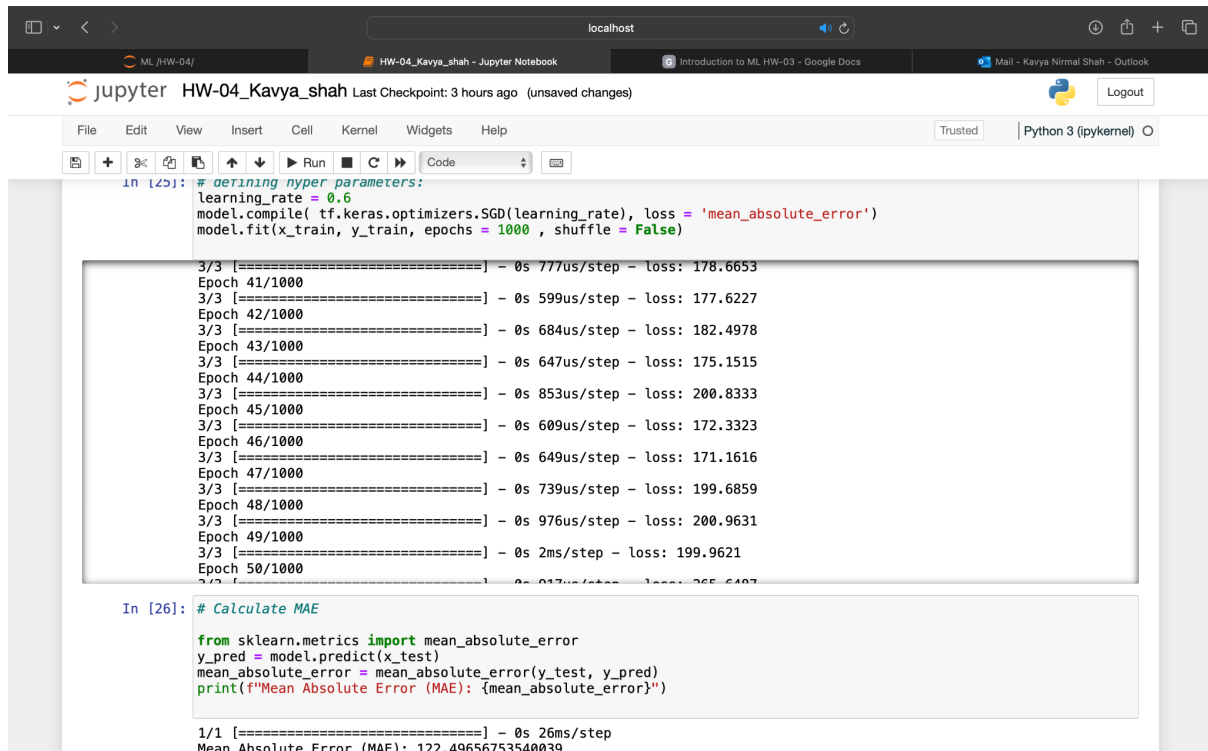
In [26]: # Calculate MAE

from sklearn.metrics import mean_absolute_error
y_pred = model.predict(x_test)
mean_absolute_error = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error (MAE): {mean_absolute_error}")

1/1 [=====] - 0s 26ms/step
Mean Absolute Error (MAE): 180.74897499084472

```

4. For epoch: 1000, learning rate = 0.6:



```

In [25]: # defining nyper parameters:
learning_rate = 0.6
model.compile( tf.keras.optimizers.SGD(learning_rate), loss = 'mean_absolute_error')
model.fit(x_train, y_train, epochs = 1000 , shuffle = False)

Epoch 41/1000
3/3 [=====] - 0s 777us/step - loss: 178.6653
Epoch 42/1000
3/3 [=====] - 0s 599us/step - loss: 177.6227
Epoch 43/1000
3/3 [=====] - 0s 684us/step - loss: 182.4978
Epoch 44/1000
3/3 [=====] - 0s 647us/step - loss: 175.1515
Epoch 45/1000
3/3 [=====] - 0s 853us/step - loss: 200.8333
Epoch 46/1000
3/3 [=====] - 0s 609us/step - loss: 172.3323
Epoch 47/1000
3/3 [=====] - 0s 649us/step - loss: 171.1616
Epoch 48/1000
3/3 [=====] - 0s 739us/step - loss: 199.6859
Epoch 49/1000
3/3 [=====] - 0s 976us/step - loss: 200.9631
Epoch 50/1000
3/3 [=====] - 0s 2ms/step - loss: 199.9621

In [26]: # Calculate MAE

from sklearn.metrics import mean_absolute_error
y_pred = model.predict(x_test)
mean_absolute_error = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error (MAE): {mean_absolute_error}")

1/1 [=====] - 0s 26ms/step
Mean Absolute Error (MAE): 122.49656753540039

```

5. For epoch: 2000, learning rate = 0.1:

```

learning_rate = 0.1
model.compile(tf.keras.optimizers.SGD(learning_rate), loss = 'mean_absolute_error')
model.fit(x_train, y_train, epochs = 2000, shuffle = False)

3/3 [=====] - 0s 1ms/step - loss: 152.8185
Epoch 41/2000
3/3 [=====] - 0s 827us/step - loss: 153.1856
Epoch 42/2000
3/3 [=====] - 0s 827us/step - loss: 153.8114
Epoch 43/2000
3/3 [=====] - 0s 841us/step - loss: 150.8393
Epoch 44/2000
3/3 [=====] - 0s 2ms/step - loss: 152.0636
Epoch 45/2000
3/3 [=====] - 0s 776us/step - loss: 152.6371
Epoch 46/2000
3/3 [=====] - 0s 831us/step - loss: 150.5703
Epoch 47/2000
3/3 [=====] - 0s 882us/step - loss: 150.3355
Epoch 48/2000
3/3 [=====] - 0s 1ms/step - loss: 150.4183
Epoch 49/2000
3/3 [=====] - 0s 2ms/step - loss: 150.7630
Epoch 50/2000
3/3 [=====] - 0s 2ms/step - loss: 150.7630

In [26]: # Calculate MAE

from sklearn.metrics import mean_absolute_error
y_pred = model.predict(x_test)
mean_absolute_error = mean_absolute_error(y_test, y_pred)
print(f'Mean Absolute Error (MAE): {mean_absolute_error}')

1/1 [=====] - 0s 27ms/step
Mean Absolute Error (MAE): 37.766401863098146

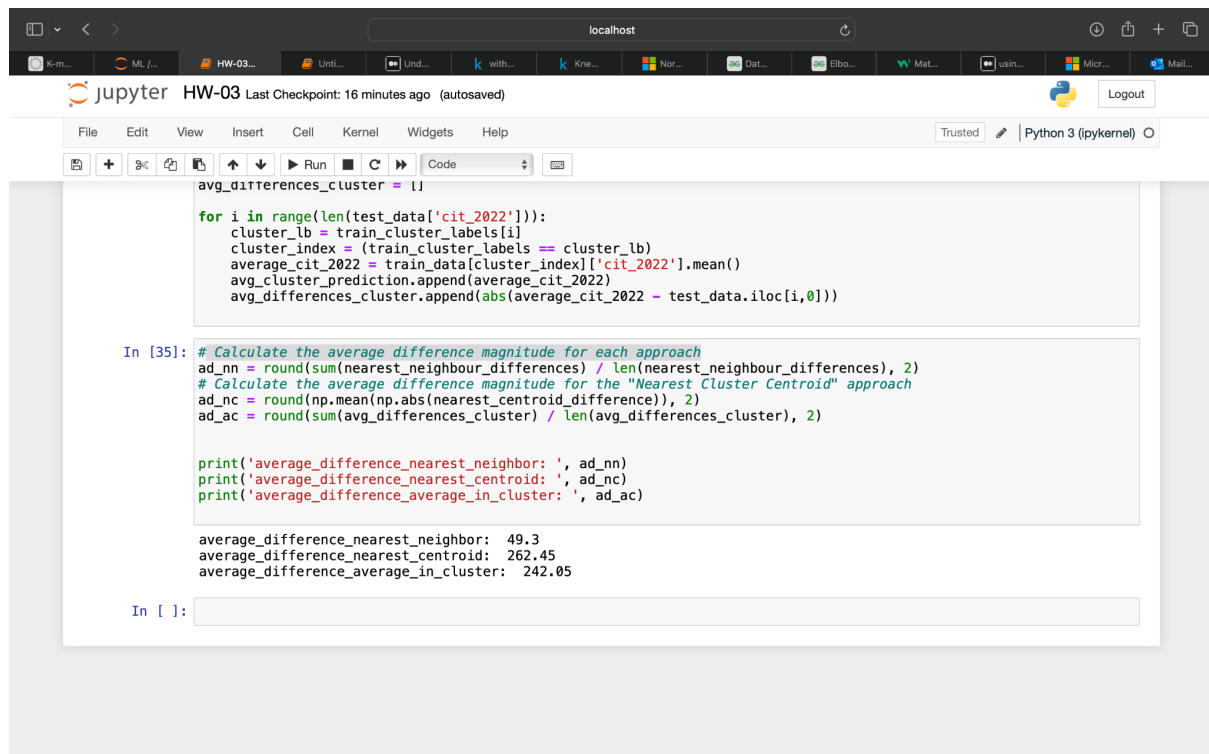
```

Now, In Hw-03,

Using  $k = 3$ , I labelled the normalised data into KMeans Clustering Model and then repeated the process for the normalised test data set

Questions:	Method used	Output
Question 1	Average difference magnitude for nearest neighbour	49.3
Question 2	Point nearest the cluster centroid	262.45
Question 3	Average of all others from the training set	242.05

## HW-03 output:



The screenshot shows a Jupyter Notebook titled "HW-03" with a last checkpoint of 16 minutes ago. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The notebook contains two code cells. The first cell defines a list `avg_differences_cluster` and a loop that iterates over `test_data['cit_2022']`, calculating the average difference for each cluster. The second cell, labeled "In [35]:", calculates the average difference magnitude for each approach, rounds the results, and prints them. The output shows the average difference for the nearest neighbor, nearest centroid, and average in cluster approaches.

```
avg_differences_cluster = []

for i in range(len(test_data['cit_2022'])):
    cluster_lb = train_cluster_labels[i]
    cluster_index = (train_cluster_labels == cluster_lb)
    average_cit_2022 = train_data[cluster_index]['cit_2022'].mean()
    avg_cluster_prediction.append(average_cit_2022)
    avg_differences_cluster.append(abs(average_cit_2022 - test_data.iloc[i,0]))

In [35]: # Calculate the average difference magnitude for each approach
ad_nn = round(sum(nearest_neighbour_differences) / len(nearest_neighbour_differences), 2)
# Calculate the average difference magnitude for the "Nearest Cluster Centroid" approach
ad_nc = round(np.mean(np.abs(nearest_centroid_difference)), 2)
ad_ac = round(sum(avg_differences_cluster) / len(avg_differences_cluster), 2)

print('average_difference_nearest_neighbor: ', ad_nn)
print('average_difference_nearest_centroid: ', ad_nc)
print('average_difference_average_in_cluster: ', ad_ac)

average_difference_nearest_neighbor: 49.3
average_difference_nearest_centroid: 262.45
average_difference_average_in_cluster: 242.05

In [ ]:
```

## Conclusion:

In this dataset, we can conclude that the predictions generated by a backpropagation neural network with a 5-3-1 architecture, trained with 1000 epochs and at 0.1 learning rate are more accurate and efficient compared to the predictions made by the nearest neighbour algorithm used in HW-03. The neural network's output, which is approximately 37.76, is a lower mean error compared to the nearest neighbour, where mean absolute error was 49.3.

- **Y- test and Y-predicted values using 1000 epochs and at 0.1 learning rate:**

Y - test:

Y -predicted:

```
In [28]: y_test
```

Out[28]:

	cit_2022
83	782
53	81
70	80
45	404
44	89
39	69
22	96
80	297
10	422
0	397
18	344
30	42
73	88
33	2
90	53
4	741
76	129
77	95
12	509
31	9

```
In [27]: y_pred
```

Out[27]: array([[692.8738 ],  
[ 91.59355 ],  
[ 66.454506],  
[401.33453 ],  
[122.34772 ],  
[110.199554],  
[108.72528 ],  
[259.09006 ],  
[335.62357 ],  
[458.32962 ],  
[319.7795 ],  
[ 20.921854],  
[ 58.752457],  
[ 19.690155],  
[ 33.98474 ],  
[628.07434 ],  
[104.128235],  
[106.52757 ],  
[600.3913 ],  
[ 23.404865]], dtype=float32)