**Assignment 1**

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import Sequential

from tensorflow.keras.layers import Dense,Flatten

from sklearn import preprocessing

tensorflow: open-source framework for building and training machine learning models.

scikit-learn library: provides various utilities for data preprocessing.

"keras" module: which is a high-level API for building and training neural networks.

A sequential model: is a linear stack of layers, where the output of each layer becomes the input for the next layer.

It adds two layers to the model: a dense layer and a flatten layer. The dense layer is a fully connected layer that performs a simple matrix multiplication on the input data, followed by a non-linear activation function. The flatten layer is used to convert the multi-dimensional output of the previous layer into a flat vector.

(X\_train,y\_train),(X\_test,y\_test)=keras.datasets.boston\_housing.load\_data()

Loading the dataset – Keras library

X – input features, y – target labels, X\_test, y\_test – unseen data

print("Training data shape:",X\_train.shape)

Training data shape: (404, 13)

404 – rows

13 – columns

print("Testing data shape:",X\_test.shape)

print("Train output data shape:",y\_train.shape)

Train output data shape: (404,)

a shape of (404,) indicates that the array has 404 elements in a single dimension.

print("Actual Test  output data shape:",y\_test.shape)

X\_train[0]

the code retrieves and displays the value of the first element from the X\_train array

y\_train[0]

y\_train

## Normalize the data

X\_train=preprocessing.normalize(X\_train)

Normalize(): scales the values in a dataset to a standard range. It ensures that all features have a similar scale and prevents any one feature from dominating the learning process.

X\_train

X\_train[0]

X\_train[1]

X\_test=preprocessing.normalize(X\_test)

X\_test

X\_test[0]

y\_test[0]

## Model Building

X\_train[0].shape

model=Sequential()

model.add(Dense(128,activation='relu',input\_shape= X\_train[0].shape))

model.add(Dense(64,activation='relu'))

model.add(Dense(32,activation='relu'))

model.add(Dense(1))

The above code creates a sequential model in TensorFlow using the Keras API and adds layers to it.

The first layer is added to the model using the model.add() method. It is a dense layer, defined by the Dense() function. The Dense() function specifies the number of neurons or units in the layer, which is set to 128 in this case. The activation function used in this layer is the Rectified Linear Unit (ReLU) activation function, denoted by 'relu'. The input shape of this layer is determined by X\_train[0].shape, which represents the shape or dimensions of the input features for a single training example.

Second layer, third layer

Last layer is output layer which has only 1 neuron, which is appropriate for regression tasks and it doesn’t contain any activation function which means it will produce a raw numerical output.

ReLU (Rectified Linear Unit) is an activation function commonly used in neural networks. It's a simple function that takes an input and returns the input itself if it's positive, and zero if it's negative.

model.summary()

model.compile(loss='mse',optimizer='rmsprop',metrics=['mae'])

compile() – configure model

mse – Mean Squared Error – common regression function - aims to minimize the average squared difference between the predicted and actual values.

Rmsprop – Root Mean Square Propagation Optimizer – optimization function commonly used for neural networks

Mae - Mean Absolute Error – represents the absolute squared difference between the predicted and actual values.

history= model.fit(X\_train,y\_train,epochs=100,batch\_size=1,verbose=1,validation\_data=(X\_test,y\_test))

The epochs parameter is set to 100, indicating the number of times the model will iterate over the entire training dataset during training.

The batch\_size parameter is set to 1, which means the model will update its weights after processing each individual sample.

The verbose parameter is set to 1, which means that during training, progress updates and metrics will be displayed on the screen.

#test\_input=[(0.02675675,0.00000000,0.026779,0.0000000,0.0010046,0.00951931,0.14795322,0.0027145,0.03550877,0.98536841,0.02988655,0.04031725,0.04298041)]

test\_input=[(0.0024119 , 0.        , 0.01592969, 0.        , 0.00105285,

       0.01201967, 0.17945359, 0.00778265, 0.00782786, 0.6007879 ,

       0.04109624, 0.77671895, 0.03663436)]

print("Actual output:15.2")

print("Predicted Output:",model.predict(test\_input))

#test\_input –X\_test[0] - 13 features for one data point

Test\_input – X\_train[0]

(values taken after normalization)

test\_input=[(4.07923050e-05, 1.54587284e-01, 3.80378407e-03, 0.00000000e+00,

       7.77620881e-04, 1.42595058e-02, 2.94184285e-02, 1.17486336e-02,

       3.74757051e-03, 6.52077269e-01, 2.75446433e-02, 7.40857215e-01,

       5.82747215e-03)]

print("Actual output:42.3")

print("Predicted Output:",model.predict(test\_input))

test\_input – X\_train[1]

y\_pred=model.predict(X\_test)

**Assignment 2**