LAB 3

```
import tensorflow as tf   # Load Tensor flow Library
import matplotlib.pyplot as plt
```

Load Boston Dataset .We will be implement Backpropagation learning on Traning data.so Test_spilt=0

```
(train_x,train_y),(_,_)=tf.keras.datasets.boston_housing.load_data(test_split=0)
#(train_x,train_y),(test_x,test_y)=tf.keras.datasets.boston_housing.load_data(test_split=0.4)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/boston_housing.npz
57026/57026 [==============================] - 0s 0us/step
```

```
y_actual=train_y
```

Size of input data along with feature size

+ Code + Text

train_x.shape

```
train_y.shape
```

```
(506,)
```

```
train_x.dtype
```

```
dtype('float64')
```

Initilaize weights and bias with zeros..............

y=w1x1+w2x2+w3x3..........+w13x13+b .....Linear Regression equation

```
w=tf.zeros(shape=(13,1))
b=tf.zeros(shape=(1))
```

Numpy works on 64bit floating numbers but Tensorflow works with 32 bit so mapping from 64 to 32 bit

```
train_x=train_x.astype('float32')
train_y=train_y.astype('float32')
```

```
train_x[0]
```

```
array([  1.23247,   0.     ,   8.14   ,   0.     ,   0.538  ,   6.142  ,
        91.7    ,   3.9769 ,   4.     , 307.     ,  21.     , 396.9    ,
        18.72   ], dtype=float32)
```

Data Normalization

```
from sklearn.preprocessing import Normalizer
```

```
transformer=Normalizer()
train_x=transformer.fit_transform(train_x)
```

```
train_x[0]
```

```
array([0.0024119 , 0.        , 0.01592969, 0.        , 0.00105285,
       0.01201967, 0.17945357, 0.00778265, 0.00782785, 0.6007879 ,
       0.04109624, 0.7767189 , 0.03663436], dtype=float32)
```

Define function to calculate Prediction

```
def prediction(train_x,w,b):
  xw_matmul=tf.matmul(train_x,w)
  y=tf.add(xw_matmul,b)
  y_predicted=y
  return y
```

Define function to calculate loss (MSE)

```
def loss(y_train,y_predicted):
  diff=y_train- y_predicted
  sqr=tf.square(diff)
  avg=tf.reduce_mean(sqr)
  return avg
```

Define function to train the model

```
def train(x,y_actual,w,b,learning_rate=0.01):
  #Record Mathematical operations on 'tape' to calculate loss
    with tf.GradientTape() as t:
      t.watch([w,b])
      current_prediction=prediction(x,w,b)
      current_loss=loss(y_actual,current_prediction)
  #Calculate gradients for loss w.r.t weights and bias
    dw,db=t.gradient(current_loss,[w,b])
  #Update weights and bias
    w=w-learning_rate*dw
    b=b-learning_rate*db
    return w,b
```

```
import matplotlib.pyplot as plt
```

Start Training

```
epochs=100
for i in range(100):
    w,b=train(train_x,train_y,w,b,learning_rate=0.01)
    avgloss=loss(train_y,prediction(train_x,w,b)).numpy()
    print('current loss on iteration ',i,loss(train_y,prediction(train_x,w,b)).numpy())
    #plt.plot(i,avgloss)
#plt.ylabel("Error")
#plt.xlabel("Epochs")
#plt.show()
```

```
current loss on iteration  0 553.7515
current loss on iteration  1 518.2617
current loss on iteration  2 485.45786
current loss on iteration  3 455.13654
current loss on iteration  4 427.10983
current loss on iteration  5 401.20413
current loss on iteration  6 377.259
current loss on iteration  7 355.12592
current loss on iteration  8 334.66785
current loss on iteration  9 315.758
current loss on iteration  10 298.27924
current loss on iteration  11 282.12317
current loss on iteration  12 267.1898
current loss on iteration  13 253.38652
current loss on iteration  14 240.62785
current loss on iteration  15 228.83473
current loss on iteration  16 217.93404
current loss on iteration  17 207.8583
current loss on iteration  18 198.54506
current loss on iteration  19 189.9366
current loss on iteration  20 181.9796
current loss on iteration  21 174.62473
current loss on iteration  22 167.82645
current loss on iteration  23 161.54263
current loss on iteration  24 155.73433
current loss on iteration  25 150.36557
current loss on iteration  26 145.40309
current loss on iteration  27 140.81613
current loss on iteration  28 136.57626
current loss on iteration  29 132.65727
current loss on iteration  30 129.03482
current loss on iteration  31 125.68646
current loss on iteration  32 122.59149
current loss on iteration  33 119.73073
current loss on iteration  34 117.086426
current loss on iteration  35 114.64222
current loss on iteration  36 112.38295
current loss on iteration  37 110.294624
current loss on iteration  38 108.36432
current loss on iteration  39 106.58006
```

```
current loss on iteration  40 104.93081
current loss on iteration  41 103.40634
current loss on iteration  42 101.99722
current loss on iteration  43 100.6947
current loss on iteration  44 99.49073
current loss on iteration  45 98.377846
current loss on iteration  46 97.349144
current loss on iteration  47 96.39828
current loss on iteration  48 95.51933
current loss on iteration  49 94.70689
current loss on iteration  50 93.9559
current loss on iteration  51 93.2617
current loss on iteration  52 92.620026
current loss on iteration  53 92.02687
current loss on iteration  54 91.478584
current loss on iteration  55 90.971756
current loss on iteration  56 90.50326
current loss on iteration  57 90.070175
```

```
print('Weights:\n',w.numpy())
print('bias:\n', b.numpy())
```

```
Weights:
 [[6.27857521e-02]
 [2.58572191e-01]
 [2.17821732e-01]
 [1.46146410e-03]
 [1.13587305e-02]
 [1.31812572e-01]
 [1.38818800e+00]
 [8.23873580e-02]
 [1.76484972e-01]
 [7.99328279e+00]
 [3.82081807e-01]
 [7.41107655e+00]
 [2.53885090e-01]]
bias:
 [11.476417]
```