Load tensorflow

```
import tensorflow as tf
```

Enable Eager Execution if you are using tensorflow 1.x

```
#Enable Eager Execution
#tf.enable_eager_execution()
```

Collect Data

```
import pandas as pd

data = pd.read_csv('./prices2.csv',encoding='utf-8')
```

Check all columns in the dataset

```
data.columns
```

```
Index(['date', 'symbol', 'open', 'close', 'low', 'high', 'volume'], dtype='object')
```

Drop columns `date` and `symbol`

```
data=data.drop(['date','symbol'],axis=1)

data.head()
```


0	123.430000	125.839996	122.309998	126.250000	2163600.0
1	125.239998	119.980003	119.940002	125.540001	2386400.0
2	116.379997	114.949997	114.930000	119.739998	2489500.0
3	115.480003	116.620003	113.500000	117.440002	2006300.0
4	117.010002	114.970001	114.089996	117.330002	1408600.0

Consider only first 1000 rows in the dataset for building feature set and target set

```
data.shape
```

```
(999, 5)
```

Convert Float64 to Float32

```
data.dtypes
```

```
open      float64
close     float64
low       float64
high      float64
volume    float64
dtype: object
```

Divide the data into train and test sets

```
X=data.drop('close', axis=1)
y=data["close"]
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_state = 1)
```

```
X_train = X_train.astype('float32')
y_train = y_train.astype('float32')
```

Normalize Train and Test Data

```
from sklearn.preprocessing import Normalizer
transformer = Normalizer()
X_train = transformer.fit_transform(X_train)
X_test = transformer.fit_transform(X_test)
```

```
X_train
```



```
[1.27343546e-05, 1.26587793e-05, 1.30163062e-05, 1.00000000e+00],
...,
[1.04472536e-04, 1.03533064e-04, 1.04582476e-04, 1.00000000e+00],
[1.26646424e-04, 1.23368736e-04, 1.27374806e-04, 9.99999940e-01],
[1.16344170e-04, 1.14913542e-04, 1.18731979e-04, 9.99999940e-01]],
dtype=float32)
```

Building the graph in tensorflow

2.Define Weights and Bias

```
#We are initializing weights and Bias with Zero
w = tf.zeros(shape=(4,1))
b = tf.zeros(shape=(1))
```

3.Prediction

```
def prediction(x, w, b):

    xw_matmul = tf.matmul(x, w)
    y = tf.add(xw_matmul, b)

    return y
```

4.Loss (Cost) Function [Mean square error]

```
def loss(y_actual, y_predicted):

    diff = y_actual - y_predicted
    sqr = tf.square(diff)
    avg = tf.reduce_mean(sqr)

    return avg
```

5.GradientDescent Optimizer to minimize Loss [GradientDescentOptimizer]

```
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 with respect to 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
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

Execute the Graph for 100 epochs and observe the loss

```
import numpy as np
y_train=np.array(y_train)
```

```
y_train.shape
```

```
#Train for 100 Steps
for i in range(100):
    w, b = train(X_train, y_train, w, b, learning_rate=0.01)
    print('Current Loss on iteration', i, loss(y_train, prediction(X_train, w, b)).numpy())
```

```
Current Loss on iteration 34 3538.1643
Current Loss on iteration 35 3518.946
Current Loss on iteration 36 3501.229
Current Loss on iteration 37 3484.902
Current Loss on iteration 38 3469.8586
Current Loss on iteration 39 3455.9915
Current Loss on iteration 40 3443.2107
Current Loss on iteration 41 3431.433
Current Loss on iteration 42 3420.58
Current Loss on iteration 43 3410.576
Current Loss on iteration 44 3401.3586
Current Loss on iteration 45 3392.8635
Current Loss on iteration 46 3385.0308
Current Loss on iteration 47 3377.815
Current Loss on iteration 48 3371.1655
Current Loss on iteration 49 3365.0364
Current Loss on iteration 50 3359.388
Current Loss on iteration 51 3354.1824
Current Loss on iteration 52 3349.3843
Current Loss on iteration 53 3344.9631
Current Loss on iteration 54 3340.8887
Current Loss on iteration 55 3337.1333
Current Loss on iteration 56 3333.671
Current Loss on iteration 57 3330.4824
Current Loss on iteration 58 3327.541
Current Loss on iteration 59 3324.834
Current Loss on iteration 60 3322.3364
Current Loss on iteration 61 3320.036
Current Loss on iteration 62 3317.9153
Current Loss on iteration 63 3315.963
Current Loss on iteration 64 3314.1594
Current Loss on iteration 65 3312.5012
Current Loss on iteration 66 3310.9712
Current Loss on iteration 67 3309.5603
Current Loss on iteration 68 3308.2627
Current Loss on iteration 69 3307.0647
Current Loss on iteration 70 3305.9602
Current Loss on iteration 71 3304.9453
Current Loss on iteration 72 3304.0063
Current Loss on iteration 73 3303.1414
```



```
Current Loss on iteration 77 3300.3147
Current Loss on iteration 78 3299.74
Current Loss on iteration 79 3299.2065
Current Loss on iteration 80 3298.7234
Current Loss on iteration 81 3298.273
Current Loss on iteration 82 3297.8586
Current Loss on iteration 83 3297.477
Current Loss on iteration 84 3297.124
Current Loss on iteration 85 3296.7998
Current Loss on iteration 86 3296.5015
Current Loss on iteration 87 3296.2258
Current Loss on iteration 88 3295.9734
Current Loss on iteration 89 3295.7388
Current Loss on iteration 90 3295.5222
Current Loss on iteration 91 3295.3237
Current Loss on iteration 92 3295.1418
Current Loss on iteration 93 3294.974
Current Loss on iteration 94 3294.8171
Current Loss on iteration 95 3294.6724
```

```
Current Loss on iteration 96 3294.5427
Current Loss on iteration 97 3294.4214
Current Loss on iteration 98 3294.3083
Current Loss on iteration 99 3294.204
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

Get the shapes and values of W and b

```
w.shape
```

```
TensorShape([Dimension(4), Dimension(1)])
```

```
b.shape
```

```
TensorShape([Dimension(1)])
```

Linear Classification using Keras

Building the simple Neural Network in Keras with one neuron in the dense hidden layer.

Use Mean square error as loss function and sgd as optimizer

```
#Initialize Sequential Graph (model)
model = tf.keras.Sequential()

#Add Dense layer for prediction - Keras declares weights and bias automatically
model.add(tf.keras.layers.Dense(1, input_shape=(4,)))

#Compile the model - add Loss and Gradient Descent optimizer
model.compile(optimizer='sgd', loss='mse')
```

Execute the model

```
model.fit(X_train, y_train, epochs=100)
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
699/699 [==============================] - 0s 32us/sample - loss: 3294.5402
Epoch 69/100
699/699 [==============================] - 0s 31us/sample - loss: 3296.1281
Epoch 70/100
699/699 [==============================] - 0s 34us/sample - loss: 3295.7881
Epoch 71/100
699/699 [==============================] - 0s 32us/sample - loss: 3298.6814
Epoch 72/100
699/699 [==============================] - 0s 35us/sample - loss: 3297.8016
Epoch 73/100
699/699 [==============================] - 0s 49us/sample - loss: 3297.4829
Epoch 74/100
699/699 [==============================] - 0s 36us/sample - loss: 3297.0508
Epoch 75/100
699/699 [==============================] - 0s 33us/sample - loss: 3295.5795
Epoch 76/100
699/699 [==============================] - 0s 41us/sample - loss: 3297.2934
Epoch 77/100
699/699 [==============================] - 0s 35us/sample - loss: 3294.9840
Epoch 78/100
699/699 [==============================] - 0s 41us/sample - loss: 3296.9732
Epoch 79/100
699/699 [==============================] - 0s 30us/sample - loss: 3297.1886
Epoch 80/100
699/699 [==============================] - 0s 32us/sample - loss: 3296.6504
Epoch 81/100
699/699 [==============================] - 0s 30us/sample - loss: 3294.6967
Epoch 82/100
699/699 [==============================] - 0s 35us/sample - loss: 3297.2123
Epoch 83/100
699/699 [==============================] - 0s 34us/sample - loss: 3295.9242
Epoch 84/100
699/699 [==============================] - 0s 30us/sample - loss: 3296.5942
Epoch 85/100
699/699 [==============================] - 0s 31us/sample - loss: 3297.4495
Epoch 86/100
699/699 [==============================] - 0s 37us/sample - loss: 3296.3899
Epoch 87/100
699/699 [==============================] - 0s 34us/sample - loss: 3295.3651
Epoch 88/100
```



```
Epoch 90/100
699/699 [==============================] - 0s 33us/sample - loss: 3294.8488
Epoch 91/100
699/699 [==============================] - 0s 30us/sample - loss: 3297.5118
Epoch 92/100
699/699 [==============================] - 0s 29us/sample - loss: 3297.4279
Epoch 93/100
699/699 [==============================] - 0s 29us/sample - loss: 3297.0927
Epoch 94/100
699/699 [==============================] - 0s 33us/sample - loss: 3295.6729
Epoch 95/100
699/699 [==============================] - 0s 35us/sample - loss: 3297.5157
Epoch 96/100
699/699 [==============================] - 0s 36us/sample - loss: 3296.5805
Epoch 97/100
699/699 [==============================] - 0s 42us/sample - loss: 3295.3373
Epoch 98/100
699/699 [==============================] - 0s 33us/sample - loss: 3298.4934
Epoch 99/100
```

```
699/699 [==============================] - 0s 31us/sample - loss: 3295.6498
Epoch 100/100
699/699 [==============================] - 0s 33us/sample - loss: 3296.6970
<tensorflow.python.keras.callbacks.History at 0x7f20a6199358>
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

Classification using Keras

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 1)                 5         
=================================================================
Non-trainable params: 0
_________________________________________________________________
```


Load the given Iris data using pandas (Iris.csv)

```
iris =  pd.read_csv('Iris.csv',encoding='utf-8')
```

```
iris.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa

```
iris=iris.drop('Id', axis=1)
```

4	5.0	3.6	1.4	0.2	Iris-setosa

Splitting the data into feature set and target set

```
X=iris.drop('Species', axis=1)
y=iris['Species']
```

```
y.unique()
```

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

Target set has different categories. So, Label encode them. And convert into one-hot vectors using get_dummies in pandas.

```
from sklearn import preprocessing
new_le = preprocessing.LabelEncoder()
y = new_le.fit_transform(y)
#
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```


Divide the dataset into Training and test (70:30)

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_state = 1)
y_train = tf.keras.utils.to_categorical(y_train, num_classes=3)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=3)
```

```
X_train.shape
```

```
(105, 4)
```

Model

Build the model with following layers:

1. First dense layer with 10 neurons with input shape 4 (according to the feature set)
2. Second Dense layer with 8 neurons
3. Output layer with 3 neurons with softmax activation (output layer, 3 neurons as we have 3 classes)
4. Use SGD and categorical_crossentropy loss

```
#Initialize Sequential model
model = tf.keras.models.Sequential()

#Reshape data from 2D to 1D -> 28x28 to 784
#model.add(tf.keras.layers.Reshape((16,),input_shape=(4,4,)))

#Normalize the data
#model.add(tf.keras.layers.BatchNormalization())
#Add 1st hidden layer
model.add(tf.keras.layers.Dense(10, input_shape=(4,)))
#Add 2st hidden layer
model.add(tf.keras.layers.Dense(8, input_shape=(4,)))

#Add Dense Layer which provides 3 Outputs after applying softmax
model.add(tf.keras.layers.Dense(3, activation='softmax'))

#Comile the model
model.compile(optimizer='sgd', loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Fitting the model and predicting

```
model.fit(X_train, y_train,
          validation_data=(X_test, y_test),
          epochs=60,
          batch_size=9)
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
Train on 105 samples, validate on 45 samples
Epoch 1/60
105/105 [==============================] - 0s 278us/sample - loss: 0.2802 - acc: 0.
Epoch 2/60
105/105 [==============================] - 0s 220us/sample - loss: 0.2715 - acc: 0.
Epoch 3/60
105/105 [==============================] - 0s 201us/sample - loss: 0.2569 - acc: 0.
Epoch 4/60
105/105 [==============================] - 0s 194us/sample - loss: 0.2581 - acc: 0.
Epoch 5/60
105/105 [==============================] - 0s 253us/sample - loss: 0.2461 - acc: 0.
Epoch 6/60
105/105 [==============================] - 0s 215us/sample - loss: 0.2388 - acc: 0.
Epoch 7/60
105/105 [==============================] - 0s 214us/sample - loss: 0.2503 - acc: 0.
Epoch 8/60
105/105 [==============================] - 0s 196us/sample - loss: 0.2280 - acc: 0.
Epoch 9/60
105/105 [==============================] - 0s 195us/sample - loss: 0.2260 - acc: 0.
Epoch 10/60
105/105 [==============================] - 0s 197us/sample - loss: 0.2126 - acc: 0.
Epoch 11/60
105/105 [==============================] - 0s 199us/sample - loss: 0.2111 - acc: 0.
Epoch 12/60
105/105 [==============================] - 0s 214us/sample - loss: 0.2136 - acc: 0.
Epoch 13/60
105/105 [==============================] - 0s 200us/sample - loss: 0.2037 - acc: 0.
Epoch 14/60
105/105 [==============================] - 0s 200us/sample - loss: 0.1926 - acc: 0.
Epoch 15/60
105/105 [==============================] - 0s 243us/sample - loss: 0.1873 - acc: 0.
Epoch 16/60
105/105 [==============================] - 0s 216us/sample - loss: 0.1989 - acc: 0.
Epoch 17/60
105/105 [==============================] - 0s 198us/sample - loss: 0.1896 - acc: 0.
Epoch 18/60
105/105 [==============================] - 0s 226us/sample - loss: 0.1721 - acc: 0.
Epoch 19/60
105/105 [==============================] - 0s 269us/sample - loss: 0.1817 - acc: 0.
Epoch 20/60
```



```
105/105 [==============================] - 0s 240us/sample - loss: 0.1515 - acc: 0.
Epoch 22/60
105/105 [==============================] - 0s 204us/sample - loss: 0.1827 - acc: 0.
Epoch 23/60
105/105 [==============================] - 0s 209us/sample - loss: 0.1584 - acc: 0.
Epoch 24/60
105/105 [==============================] - 0s 230us/sample - loss: 0.1754 - acc: 0.
Epoch 25/60
105/105 [==============================] - 0s 223us/sample - loss: 0.1562 - acc: 0.
Epoch 26/60
105/105 [==============================] - 0s 224us/sample - loss: 0.1532 - acc: 0.
Epoch 27/60
105/105 [==============================] - 0s 213us/sample - loss: 0.1438 - acc: 0.
Epoch 28/60
105/105 [==============================] - 0s 246us/sample - loss: 0.1477 - acc: 0.
Epoch 29/60
105/105 [==============================] - 0s 233us/sample - loss: 0.1665 - acc: 0.
Epoch 30/60
105/105 [==============================] - 0s 219us/sample - loss: 0.1545 - acc: 0.
```