Linear Classifier in TensorFlow

Using Low Level API in Eager Execution mode

Load tensorflow

```
import tensorflow as tf
```

```
#Enable Eager Execution if using tensflow version < 2.0
#From tensorflow v2.0 onwards, Eager Execution will be enabled by default
```

Collect Data

```
from google.colab import drive
drive.mount('/gdrive')
```

```
Drive already mounted at /gdrive; to attempt to forcibly remount, call drive.mount("/
```

```
import pandas as pd
```

Double-click (or enter) to edit

```
data = pd.read_csv('/gdrive/My Drive/3nov/prices.csv')
```

```
data
```

	date	symbol	open	close	low	high	v
0	2016-01-05 00:00:00	WLTW	123.430000	125.839996	122.309998	126.250000	216
1	2016-01-06 00:00:00	WLTW	125.239998	119.980003	119.940002	125.540001	238
2	2016-01-07 00:00:00	WLTW	116.379997	114.949997	114.930000	119.739998	248
3	2016-01-08 00:00:00	WLTW	115.480003	116.620003	113.500000	117.440002	200
4	2016-01-11 00:00:00	WLTW	117.010002	114.970001	114.089996	117.330002	140
5	2016-01-12 00:00:00	WLTW	115.510002	115.550003	114.500000	116.059998	109
6	2016-01-13 00:00:00	WLTW	116.459999	112.849998	112.589996	117.070000	94
7	2016-01-14 00:00:00	WLTW	113.510002	114.379997	110.050003	115.029999	78
8	2016-01-15 00:00:00	WLTW	113.330002	112.529999	111.919998	114.879997	109
9	2016-01-19 00:00:00	WLTW	113.660004	110.379997	109.870003	115.870003	152
10	2016-01-20 00:00:00	WLTW	109.059998	109.300003	108.320000	111.599998	165
11	2016-01-21 00:00:00	WLTW	109.730003	110.000000	108.320000	110.580002	94
12	2016-01-22 00:00:00	WLTW	111.879997	111.949997	110.190002	112.949997	74
13	2016-01-25 00:00:00	WLTW	111.320000	110.120003	110.000000	114.629997	70
14	2016-01-26 00:00:00	WLTW	110.419998	111.000000	107.300003	111.400002	56
15	2016-01-27 00:00:00	WLTW	110.769997	110.709999	109.019997	112.570000	89
16	2016-01-28 00:00:00	WLTW	110.900002	112.580002	109.900002	112.970001	68
17	2016-01-29 00:00:00	WLTW	113.349998	114.470001	111.669998	114.589996	74
18	2016-02-01 00:00:00	WLTW	114.000000	114.500000	112.900002	114.849998	57
19	2016-02-02 00:00:00	WLTW	113.250000	110.559998	109.750000	113.860001	69
20	2016-02-03 00:00:00	WLTW	113.379997	114.050003	109.639999	114.639999	89
21	2016-02-04 00:00:00	WLTW	114.080002	115.709999	114.080002	116.320000	95
22	2016-02-05 00:00:00	WLTW	115.120003	114.019997	109.709999	116.489998	99
23	2016-02-08 00:00:00	WLTW	113.300003	111.160004	110.459999	113.300003	120
24	2016-02-09 00:00:00	WLTW	111.169998	110.650002	109.639999	112.110001	172
25	2016-02-10 00:00:00	WLTW	106.730003	107.519997	106.360001	112.110001	194
26	2016-02-11 00:00:00	WLTW	105.629997	107.129997	104.110001	109.260002	131
27	2016-02-12 00:00:00	WLTW	108.559998	107.839996	107.070000	109.430000	92
28	2016-02-16 00:00:00	WLTW	109.110001	110.769997	107.010002	111.300003	118
29	2016-02-17 00:00:00	WLTW	110.830002	111.239998	107.970001	112.110001	92
...	...	...	...	...	...	...	
851234	2016-12-30	WAT	135.240005	134.389999	133.710007	135.300003	46
851235	2016-12-30	WBA	83.459999	82.760002	82.419998	83.620003	334

851235	2016-12-30	WBA	83.459999	82.760002	82.419998	83.620003	334
851236	2016-12-30	WDC	68.550003	67.949997	67.610001	69.400002	282
851237	2016-12-30	WEC	58.980000	58.650002	58.419998	59.119999	122
851238	2016-12-30	WFC	54.889999	55.110001	54.790001	55.360001	1509
851239	2016-12-30	WFM	31.059999	30.760000	30.670000	31.299999	270
851240	2016-12-30	WHR	183.800003	181.770004	180.869995	184.289993	45
851241	2016-12-30	WM	71.269997	70.910004	70.750000	71.500000	123
851242	2016-12-30	WMB	30.940001	31.139999	30.889999	31.650000	398
851243	2016-12-30	WMT	69.120003	69.120003	68.830002	69.430000	687
851244	2016-12-30	WRK	51.840000	50.770000	50.529999	51.840000	81
851245	2016-12-30	WU	21.840000	21.719999	21.600000	21.900000	253
851246	2016-12-30	WY	30.450001	30.090000	29.950001	30.450001	282
851247	2016-12-30	WYN	76.849998	76.370003	76.180000	76.970001	52
851248	2016-12-30	WYNN	87.099998	86.510002	85.570000	87.449997	188
851249	2016-12-30	XEC	136.520004	135.899994	135.309998	137.559998	46
851250	2016-12-30	XEL	41.000000	40.700001	40.560001	41.070000	188
851251	2016-12-30	XL	37.360001	37.259998	37.060001	37.419998	95
851252	2016-12-30	XLNX	61.090000	60.369999	60.020000	61.480000	211
851253	2016-12-30	XOM	90.029999	90.260002	90.010002	90.699997	911
851254	2016-12-30	XRAY	58.290001	57.730000	57.540001	58.360001	94
851255	2016-12-30	XRX	8.720000	8.730000	8.700000	8.800000	1125
851256	2016-12-30	XYL	49.980000	49.520000	49.360001	50.000000	64
851257	2016-12-30	YHOO	38.720001	38.669998	38.430000	39.000000	643
851258	2016-12-30	YUM	63.930000	63.330002	63.160000	63.939999	188
851259	2016-12-30	ZBH	103.309998	103.199997	102.849998	103.930000	97
851260	2016-12-30	ZION	43.070000	43.040001	42.689999	43.310001	193
851261	2016-12-30	ZTS	53.639999	53.529999	53.270000	53.740002	170
851262	2016-12-30 00:00:00	AIV	44.730000	45.450001	44.410000	45.590000	138

Check all columns in the dataset

```
data.describe()
```

	open	close	low	high	volume
count	851264.000000	851264.000000	851264.000000	851264.000000	8.512640e+05
mean	70.836986	70.857109	70.118414	71.543476	5.415113e+06
std	83.695876	83.689686	82.877294	84.465504	1.249468e+07
min	0.850000	0.860000	0.830000	0.880000	0.000000e+00
25%	33.840000	33.849998	33.480000	34.189999	1.221500e+06
50%	52.770000	52.799999	52.230000	53.310001	2.476250e+06
75%	79.879997	79.889999	79.110001	80.610001	5.222500e+06

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 851264 entries, 0 to 851263
Data columns (total 7 columns):
date      851264 non-null object
symbol    851264 non-null object
open      851264 non-null float64
close     851264 non-null float64
low       851264 non-null float64
high      851264 non-null float64
volume    851264 non-null float64
dtypes: float64(5), object(2)
memory usage: 45.5+ MB
```

Drop columns `date` and `symbol`

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

```
data
```

	open	close	low	high	volume
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
5	115.510002	115.550003	114.500000	116.059998	1098000.0
6	116.459999	112.849998	112.589996	117.070000	949600.0
7	113.510002	114.379997	110.050003	115.029999	785300.0
8	113.330002	112.529999	111.919998	114.879997	1093700.0
9	113.660004	110.379997	109.870003	115.870003	1523500.0
10	109.059998	109.300003	108.320000	111.599998	1653900.0
11	109.730003	110.000000	108.320000	110.580002	944300.0
12	111.879997	111.949997	110.190002	112.949997	744900.0
13	111.320000	110.120003	110.000000	114.629997	703800.0
14	110.419998	111.000000	107.300003	111.400002	563100.0
15	110.769997	110.709999	109.019997	112.570000	896100.0
16	110.900002	112.580002	109.900002	112.970001	680400.0
17	113.349998	114.470001	111.669998	114.589996	749900.0
18	114.000000	114.500000	112.900002	114.849998	574200.0
19	113.250000	110.559998	109.750000	113.860001	694800.0
20	113.379997	114.050003	109.639999	114.639999	896300.0
21	114.080002	115.709999	114.080002	116.320000	956300.0
22	115.120003	114.019997	109.709999	116.489998	997100.0
23	113.300003	111.160004	110.459999	113.300003	1200500.0
24	111.169998	110.650002	109.639999	112.110001	1725200.0
25	106.730003	107.519997	106.360001	112.110001	1946000.0
26	105.629997	107.129997	104.110001	109.260002	1319500.0
27	108.559998	107.839996	107.070000	109.430000	922400.0
28	109.110001	110.769997	107.010002	111.300003	1185100.0
29	110.830002	111.239998	107.970001	112.110001	921500.0
...	...	...	...	...	...
851234	135.240005	134.389999	133.710007	135.300003	464200.0
851235	83.459999	82.760002	82.419998	83.620003	3343200.0

851235	83.459999	82.760002	82.419998	83.620003	3343200.0
851236	68.550003	67.949997	67.610001	69.400002	2824100.0
851237	58.980000	58.650002	58.419998	59.119999	1221800.0
851238	54.889999	55.110001	54.790001	55.360001	15095500.0
851239	31.059999	30.760000	30.670000	31.299999	2707500.0
851240	183.800003	181.770004	180.869995	184.289993	458200.0
851241	71.269997	70.910004	70.750000	71.500000	1230600.0
851242	30.940001	31.139999	30.889999	31.650000	3980300.0
851243	69.120003	69.120003	68.830002	69.430000	6872000.0
851244	51.840000	50.770000	50.529999	51.840000	811200.0
851245	21.840000	21.719999	21.600000	21.900000	2538900.0
851246	30.450001	30.090000	29.950001	30.450001	2825300.0
851247	76.849998	76.370003	76.180000	76.970001	524600.0
851248	87.099998	86.510002	85.570000	87.449997	1888500.0
851249	136.520004	135.899994	135.309998	137.559998	466100.0
851250	41.000000	40.700001	40.560001	41.070000	1887600.0
851251	37.360001	37.259998	37.060001	37.419998	959200.0
851252	61.090000	60.369999	60.020000	61.480000	2111700.0
851253	90.029999	90.260002	90.010002	90.699997	9117800.0
851254	58.290001	57.730000	57.540001	58.360001	949200.0
851255	8.720000	8.730000	8.700000	8.800000	11250400.0
851256	49.980000	49.520000	49.360001	50.000000	646200.0
851257	38.720001	38.669998	38.430000	39.000000	6431600.0
851258	63.930000	63.330002	63.160000	63.939999	1887100.0
851259	103.309998	103.199997	102.849998	103.930000	973800.0
851260	43.070000	43.040001	42.689999	43.310001	1938100.0
851261	53.639999	53.529999	53.270000	53.740002	1701200.0
851262	44.730000	45.450001	44.410000	45.590000	1380900.0

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 851264 entries, 0 to 851263
Data columns (total 5 columns):
open      851264 non-null float64
```

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

Target 'Volume' has very high values. Divide 'Volume' by 1000,000

```
data['volume']=data['volume'].div(1000000)
```

```
data
```

Divide the data into train and test sets

```
from sklearn.model_selection import train_test_split
```

```
X =data[data.columns[1:-1]]
y=data["volume"]
```

```
train_x,test_x,train_y,test_y=train_test_split(X,y,test_size=0.30,random_state=1)
```

Convert Training and Test Data to numpy float32 arrays

```
train_x.shape
```

```
test_x.shape
```

```
train_y.shape
```

```
test_y.shape
```

```
import numpy as np
```

```
train_x=np.array(train_x.astype('float32'))
```

```
train_y=np.array(train_y.astype('float32'))
```

```
test_x=np.array(test_x.astype('float32'))
```

```
test_y=np.array(test_y.astype('float32'))
```

Normalize the data

You can use Normalizer from sklearn.preprocessing

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

```
transformer = Normalizer().fit(X)
```

```
transformer
```

Building the Model in tensorflow

1.Define Weights and Bias, use tf.zeros to initialize weights and Bias

```
w = tf.zeros(shape=(3,1))
b = tf.zeros(shape=(1))
```

2.Define a function to calculate prediction

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

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

    return y
```

3.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
```

4.Function to train the Model

1. Record all the mathematical steps to calculate Loss
2. Calculate Gradients of Loss w.r.t weights and bias
3. Update Weights and Bias based on gradients and learning rate to minimize loss

```
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
```

Train the model for 100 epochs

1. Observe the training loss at every iteration
2. Observe Train loss at every 5th iteration

```
import tensorflow as tf


for i in range(100):

    w, b = train(train_x, train_y, w, b)
    print('Current Loss on iteration', i, loss(train_y, prediction(train_x, w, b)))
```

Get the shapes and values of W and b

```
print('Weights:\n', w)
print('Bias:\n',b)
```

Model Prediction on 1st Examples in Test Dataset

```
prediction(test_x[0:1], w, b).numpy()
```

Classification using tf.Keras

In this exercise, we will build a Deep Neural Network using tf.Keras. We will use Iris Dataset for this

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

```
data1 = pd.read_csv('/gdrive/My Drive/3nov/Iris.csv')
```

```
data1
```

```
datadummy=pd.get_dummies(data=data1[("Species")])
```

```
datadummy
```

```
dataconc=pd.concat([data1,datadummy.iloc[:,0:3]],axis=1)
```

```
dataconc
```

Target set has different categories. So, Label encode them. And convert into o pandas.

```
dataconc.replace(to_replace="Iris-setosa", value=0,inplace=True)
dataconc.replace(to_replace="Iris-versicolor", value=1,inplace=True)
dataconc.replace(to_replace="Iris-virginica", value=2,inplace=True)
```

```
dataconc
```

Splitting the data into feature set and target set

```
features=dataconc[["SepalLengthCm","SepalWidthCm","PetalLengthCm","PetalWidthCm"]]
target=dataconc["Species"]
```

```
X_train,X_test,y_train,y_test=train_test_split(features,target,test_size=0.30)
```

Building Model in tf.keras

Build a Linear Classifier model

1. Use Dense Layer with input shape of 4 (according to the feature set) and number of outputs set to 3
2. Apply Softmax on Dense Layer outputs
3. Use SGD as Optimizer
4. Use categorical_crossentropy as loss function

```
model = tf.keras.Sequential()
```

```
model.add(tf.keras.layers.Dense(3, input_shape=(4,),activation="softmax"))
model.add(tf.keras.layers.BatchNormalization())
model.compile(optimizer='sgd', loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Model Training

```
model.summary()
```

Model Prediction

```
y_train = tf.keras.utils.to_categorical(y_train, num_classes=3)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=3)
```

```
model.fit(X_train,y_train,
          validation_data=(X_test,y_test),
          epochs=100,
          batch_size=32)
```