Linear Classifier in TensorFlow

Using Low Level API in Eager Execution mode

Double-click (or enter) to edit

Load tensorflow

```
import tensorflow as tf
```

```
tf.__version__
```

```
'1.15.0'
```

```
#Make sure tf 2.x is installed
!pip install -U tensorflow --quiet
```

```
False
```

```
#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')
```

```
Mounted at /gdrive
```

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

```
Mounted at /content/drive
```

```
import pandas as pd
```

```
data = pd.read_csv('/content/drive/My Drive/Residency 6 Lab/prices.csv')
```

Check all columns in the dataset

```
data.columns
```

Drop columns `date` and `symbol`

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

```
data.head()
```

	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

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

```
data1 = data.head(1000)
data1.shape
```

```
(1000, 5)
```

```
data1['volume'] = data1['volume'].div(1000000)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarnin
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/use
  """Entry point for launching an IPython kernel.
```

```
data1.head()
```

	open	close	low	high	volume
0	123.430000	125.839996	122.309998	126.250000	2.1636
1	125.239998	119.980003	119.940002	125.540001	2.3864
2	116.379997	114.949997	114.930000	119.739998	2.4895
3	115.480003	116.620003	113.500000	117.440002	2.0063
4	117.010002	114.970001	114.089996	117.330002	1.4086

Divide the data into train and test sets

```
from sklearn.model_selection import train_test_split

X = data1.copy()
y = data1['volume'].copy()

X_train,X_test,y_train,y_test= train_test_split(X,y,test_size = 0.30,random_state=324)



X_train.shape[0] == y_train.shape[0]
```

```
True
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(700, 5)
(300, 5)
(700,)
(300,)
```

Convert Training and Test Data to numpy float32 arrays

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

Normalize the data

You can use Normalizer from sklearn.preprocessing

```
from sklearn import preprocessing

X_train_scaled = preprocessing.scale(X_train)
X_train_scaled
```

```
array([[-3.02683250e-01, -3.11029505e-01, -3.02807150e-01,
        -3.03985463e-01, -1.52317386e-01],
       [-4.96933537e-01, -4.88134326e-01, -4.92534357e-01,
        -4.92576334e-01,  1.26143527e-01],
       [-8.73003563e-02, -7.53972951e-02, -8.58056193e-02,
        -7.25330483e-02, -1.29796540e-01],
       ...,
       [ 9.47999286e+00,  9.42922683e+00,  9.47402480e+00,
         9.42508312e+00,  5.01227637e-02],
       [-2.15358323e-02,  4.56127656e-05, -1.74493110e-02,
        -5.13111311e-03, -8.16190903e-02],
       [-6.80194941e-01, -6.85875988e-01, -6.79704569e-01,
        -6.80662908e-01,  3.50354973e-01]])
```

```
X_test_scaled = preprocessing.scale(X_test)
y_train_scaled = preprocessing.scale(y_train)
y_test_scaled = preprocessing.scale(y_test)

X_train_scaled = X_train_scaled.astype('float32')
X_test_scaled = X_test_scaled.astype('float32')
y_train_scaled = y_train_scaled.astype('float32')
y_test_scaled = y_test_scaled.astype('float32')
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/data.py:172: UserWarning
  warnings.warn("Numerical issues were encountered "
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/data.py:172: UserWarning
  warnings.warn("Numerical issues were encountered "
```

Building the Model in tensorflow

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

```
w = tf.zeros(shape=(5,1))
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

```
for i in range(100):

    w, b = train(X_train_scaled, y_train_scaled, w, b, learning_rate=0.01)
    print('Current Loss on iteration', i,
          loss(y_train_scaled, prediction(X_train_scaled, w, b)))
```

```
Current Loss on iteration 0 Tensor("Mean_609:0", shape=(), dtype=float32)
Current Loss on iteration 1 Tensor("Mean_611:0", shape=(), dtype=float32)
Current Loss on iteration 2 Tensor("Mean_613:0", shape=(), dtype=float32)
Current Loss on iteration 3 Tensor("Mean_615:0", shape=(), dtype=float32)
Current Loss on iteration 4 Tensor("Mean_617:0", shape=(), dtype=float32)
Current Loss on iteration 5 Tensor("Mean_619:0", shape=(), dtype=float32)
Current Loss on iteration 6 Tensor("Mean_621:0", shape=(), dtype=float32)
Current Loss on iteration 7 Tensor("Mean_623:0", shape=(), dtype=float32)
Current Loss on iteration 8 Tensor("Mean_625:0", shape=(), dtype=float32)
Current Loss on iteration 9 Tensor("Mean_627:0", shape=(), dtype=float32)
Current Loss on iteration 10 Tensor("Mean_629:0", shape=(), dtype=float32)
Current Loss on iteration 11 Tensor("Mean_631:0", shape=(), dtype=float32)
Current Loss on iteration 12 Tensor("Mean_633:0", shape=(), dtype=float32)
Current Loss on iteration 13 Tensor("Mean_635:0", shape=(), dtype=float32)
Current Loss on iteration 14 Tensor("Mean_637:0", shape=(), dtype=float32)
Current Loss on iteration 15 Tensor("Mean_639:0", shape=(), dtype=float32)
Current Loss on iteration 16 Tensor("Mean_641:0", shape=(), dtype=float32)
Current Loss on iteration 17 Tensor("Mean_643:0", shape=(), dtype=float32)
Current Loss on iteration 18 Tensor("Mean_645:0", shape=(), dtype=float32)
Current Loss on iteration 19 Tensor("Mean_647:0", shape=(), dtype=float32)
Current Loss on iteration 20 Tensor("Mean_649:0", shape=(), dtype=float32)
Current Loss on iteration 21 Tensor("Mean_651:0", shape=(), dtype=float32)
Current Loss on iteration 22 Tensor("Mean_653:0", shape=(), dtype=float32)
Current Loss on iteration 23 Tensor("Mean_655:0", shape=(), dtype=float32)
Current Loss on iteration 24 Tensor("Mean_657:0", shape=(), dtype=float32)
Current Loss on iteration 25 Tensor("Mean_659:0", shape=(), dtype=float32)
Current Loss on iteration 26 Tensor("Mean_661:0", shape=(), dtype=float32)
Current Loss on iteration 27 Tensor("Mean_663:0", shape=(), dtype=float32)
Current Loss on iteration 28 Tensor("Mean_665:0", shape=(), dtype=float32)
Current Loss on iteration 29 Tensor("Mean_667:0", shape=(), dtype=float32)
Current Loss on iteration 30 Tensor("Mean_669:0", shape=(), dtype=float32)
Current Loss on iteration 31 Tensor("Mean_671:0", shape=(), dtype=float32)
Current Loss on iteration 32 Tensor("Mean_673:0", shape=(), dtype=float32)
Current Loss on iteration 33 Tensor("Mean_675:0", shape=(), dtype=float32)
Current Loss on iteration 34 Tensor("Mean_677:0", shape=(), dtype=float32)
Current Loss on iteration 35 Tensor("Mean_679:0", shape=(), dtype=float32)
Current Loss on iteration 36 Tensor("Mean_681:0", shape=(), dtype=float32)
Current Loss on iteration 37 Tensor("Mean_683:0", shape=(), dtype=float32)
Current Loss on iteration 38 Tensor("Mean_685:0", shape=(), dtype=float32)
Current Loss on iteration 39 Tensor("Mean_687:0", shape=(), dtype=float32)
Current Loss on iteration 40 Tensor("Mean_689:0", shape=(), dtype=float32)
Current Loss on iteration 41 Tensor("Mean_691:0", shape=(), dtype=float32)
Current Loss on iteration 42 Tensor("Mean_693:0", shape=(), dtype=float32)
Current Loss on iteration 43 Tensor("Mean_695:0", shape=(), dtype=float32)
Current Loss on iteration 44 Tensor("Mean_697:0", shape=(), dtype=float32)
Current Loss on iteration 45 Tensor("Mean_699:0", shape=(), dtype=float32)
Current Loss on iteration 46 Tensor("Mean_701:0", shape=(), dtype=float32)
Current Loss on iteration 47 Tensor("Mean_703:0", shape=(), dtype=float32)
Current Loss on iteration 48 Tensor("Mean_705:0", shape=(), dtype=float32)
Current Loss on iteration 49 Tensor("Mean_707:0", shape=(), dtype=float32)
Current Loss on iteration 50 Tensor("Mean_709:0", shape=(), dtype=float32)
Current Loss on iteration 51 Tensor("Mean_711:0", shape=(), dtype=float32)
Current Loss on iteration 52 Tensor("Mean_713:0", shape=(), dtype=float32)
Current Loss on iteration 53 Tensor("Mean_715:0", shape=(), dtype=float32)
Current Loss on iteration 54 Tensor("Mean_717:0", shape=(), dtype=float32)
Current Loss on iteration 55 Tensor("Mean_719:0", shape=(), dtype=float32)
Current Loss on iteration 56 Tensor("Mean_721:0", shape=(), dtype=float32)
Current Loss on iteration 57 Tensor("Mean_723:0", shape=(), dtype=float32)
Current Loss on iteration 58 Tensor("Mean_725:0", shape=(), dtype=float32)
Current Loss on iteration 59 Tensor("Mean_727:0", shape=(), dtype=float32)
Current Loss on iteration 60 Tensor("Mean_729:0", shape=(), dtype=float32)
```

```
Current Loss on iteration 61 Tensor("Mean_731:0", shape=(), dtype=float32)
Current Loss on iteration 62 Tensor("Mean_733:0", shape=(), dtype=float32)
Current Loss on iteration 63 Tensor("Mean_735:0", shape=(), dtype=float32)
Current Loss on iteration 64 Tensor("Mean_737:0", shape=(), dtype=float32)
Current Loss on iteration 65 Tensor("Mean_739:0", shape=(), dtype=float32)
Current Loss on iteration 66 Tensor("Mean_741:0", shape=(), dtype=float32)
Current Loss on iteration 67 Tensor("Mean_743:0", shape=(), dtype=float32)
Current Loss on iteration 68 Tensor("Mean_745:0", shape=(), dtype=float32)
Current Loss on iteration 69 Tensor("Mean_747:0", shape=(), dtype=float32)
Current Loss on iteration 70 Tensor("Mean_749:0", shape=(), dtype=float32)
Current Loss on iteration 71 Tensor("Mean_751:0", shape=(), dtype=float32)
Current Loss on iteration 72 Tensor("Mean_753:0", shape=(), dtype=float32)
Current Loss on iteration 73 Tensor("Mean_755:0", shape=(), dtype=float32)
Current Loss on iteration 74 Tensor("Mean_757:0", shape=(), dtype=float32)
Current Loss on iteration 75 Tensor("Mean_759:0", shape=(), dtype=float32)
Current Loss on iteration 76 Tensor("Mean_761:0", shape=(), dtype=float32)
Current Loss on iteration 77 Tensor("Mean_763:0", shape=(), dtype=float32)
Current Loss on iteration 78 Tensor("Mean_765:0", shape=(), dtype=float32)
Current Loss on iteration 79 Tensor("Mean_767:0", shape=(), dtype=float32)
Current Loss on iteration 80 Tensor("Mean_769:0", shape=(), dtype=float32)
Current Loss on iteration 81 Tensor("Mean_771:0", shape=(), dtype=float32)
Current Loss on iteration 82 Tensor("Mean_773:0", shape=(), dtype=float32)
Current Loss on iteration 83 Tensor("Mean_775:0", shape=(), dtype=float32)
Current Loss on iteration 84 Tensor("Mean_777:0", shape=(), dtype=float32)
Current Loss on iteration 85 Tensor("Mean_779:0", shape=(), dtype=float32)
Current Loss on iteration 86 Tensor("Mean_781:0", shape=(), dtype=float32)
Current Loss on iteration 87 Tensor("Mean_783:0", shape=(), dtype=float32)
Current Loss on iteration 88 Tensor("Mean_785:0", shape=(), dtype=float32)
Current Loss on iteration 89 Tensor("Mean_787:0", shape=(), dtype=float32)
Current Loss on iteration 90 Tensor("Mean_789:0", shape=(), dtype=float32)
Current Loss on iteration 91 Tensor("Mean_791:0", shape=(), dtype=float32)
Current Loss on iteration 92 Tensor("Mean_793:0", shape=(), dtype=float32)
Current Loss on iteration 93 Tensor("Mean_795:0", shape=(), dtype=float32)
Current Loss on iteration 94 Tensor("Mean_797:0", shape=(), dtype=float32)
Current Loss on iteration 95 Tensor("Mean_799:0", shape=(), dtype=float32)
Current Loss on iteration 96 Tensor("Mean_801:0", shape=(), dtype=float32)
Current Loss on iteration 97 Tensor("Mean_803:0", shape=(), dtype=float32)
Current Loss on iteration 98 Tensor("Mean_805:0", shape=(), dtype=float32)
Current Loss on iteration 99 Tensor("Mean_807:0", shape=(), dtype=float32)
```

```
X_train_scaled
```

```
array([[-3.0268326e-01, -3.1102949e-01, -3.0280715e-01, -3.0398548e-01,
        -1.5231739e-01],
       [-4.9693355e-01, -4.8813432e-01, -4.9253437e-01, -4.9257633e-01,
         1.2614353e-01],
       [-8.7300353e-02, -7.5397298e-02, -8.5805617e-02, -7.2533049e-02,
        -1.2979653e-01],
       ...,
       [ 9.4799929e+00,  9.4292269e+00,  9.4740248e+00,  9.4250832e+00,
         5.0122764e-02],
       [-2.1535832e-02,  4.5612767e-05, -1.7449312e-02, -5.1311129e-03,
        -8.1619091e-02],
       [-6.8019491e-01, -6.8587601e-01, -6.7970455e-01, -6.8066293e-01,
         3.5035497e-01]], dtype=float32)
```

Get the shapes and values of W and b

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

```
Weights:
 Tensor("sub_401:0", shape=(5, 1), dtype=float32)
```

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

```
Bias:
 Tensor("sub_402:0", shape=(1,), dtype=float32)
```

Model Prediction on 1st Examples in Test Dataset

```
result = prediction(X_test_scaled, w, b)
result
```

```
<tf.Tensor 'Add_612:0' shape=(300, 1) dtype=float32>
```

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)

```
iris_data = pd.read_csv('/content/drive/My Drive/Residency 6 Lab/Iris.csv')
iris_data.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

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

```
iris_data = pd.get_dummies( iris_data, prefix=['Species'])
iris_data.drop(['Id'], axis = 1, inplace = True)
iris_data.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species_Iris-setosa	Spe
0	5.1	3.5	1.4	0.2	1	
1	4.9	3.0	1.4	0.2	1	
2	4.7	3.2	1.3	0.2	1	
3	4.6	3.1	1.5	0.2	1	
4	5.0	3.6	1.4	0.2	1	

Splitting the data into feature set and target set

```
target = iris_data[iris_data.columns[0:4]]
feature = iris_data[iris_data.columns[4:7]]
feature.head()
print(target.shape)
print(feature.shape)
```

```
(150, 4)
(150, 3)
```

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

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

#Normalize the data
model.add(tf.keras.layers.BatchNormalization())

#Add Dense Layer which provides 3 Outputs after applying softmax
model.add(tf.keras.layers.Dense(3, input_shape=(4,), activation='softmax'))

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

```
X1_train,X1_test,y1_train,y1_test= train_test_split(target,feature,test_size = 0.30,random
print(X1_train.shape)
print(X1_test.shape)
print(y1_train.shape)
print(y1_test.shape)
print(target.shape)
```

```
print(target.shape)
print(feature.shape)
```

```
(105, 4)
(45, 4)
(105, 3)
(45, 3)
(150, 4)
(150, 3)
```

Model Training

```
model.fit(X1_train, y1_train,
          validation_data=(X1_test, y1_test),
          epochs=100,
          batch_size=X1_train.shape[0])
```

Model Prediction

```
pred = model.predict(X1_test)
pred
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-206-5df4f5286b6d> in <module>()
----> 1 pred = model.predict(X1_test)
      2 pred

                                 4 frames
/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/keras/engine/training_u
   1147           'You cannot pass a dictionary as model {}.'.format(field_name))
   1148   elif not isinstance(inp, np.ndarray) and not tensor_util.is_tensor(inp):
-> 1149     raise ValueError(
   1150         'Please provide as model inputs either a single array or a list of '
   1151         'arrays. You passed: {}={}'.format(field_name, orig_inp))

ValueError: Please provide as model inputs either a single array or a list of arrays.
```

11	4.8	3.4	1.6	0.2
106	4.9	2.5	4.5	1.7
102	7.1	3.0	5.9	2.1
10	5.4	3.7	1.5	0.2
126	6.2	2.8	4.8	1.8
65	6.7	3.1	4.4	1.4
57	4.9	2.4	3.3	1.0
31	5.4	3.4	1.5	0.4
67	5.8	2.7	4.1	1.0
145	6.7	3.0	5.2	2.3
92	5.8	2.6	4.0	1.2
76	6.8	2.8	4.8	1.4
90	5.5	2.6	4.4	1.2
43	5.0	3.5	1.6	0.6
115	6.4	3.2	5.3	2.3
118	7.7	2.6	6.9	2.3
139	6.9	3.1	5.4	2.1
18	5.7	3.8	1.7	0.3
27	5.2	3.5	1.5	0.2
122	7.7	2.8	6.7	2.0
109	7.2	3.6	6.1	2.5
112	6.8	3.0	5.5	2.1
19	5.1	3.8	1.5	0.3
146	6.3	2.5	5.0	1.9
25	5.0	3.0	1.6	0.2
64	5.6	2.9	3.6	1.3
28	5.2	3.4	1.4	0.2
144	6.7	3.3	5.7	2.5
33	5.5	4.2	1.4	0.2
143	6.8	3.2	5.9	2.3
39	5.1	3.4	1.5	0.2
36	5.5	3.5	1.3	0.2
69	5.6	2.5	3.9	1.1
59	5.2	2.7	3.9	1.4
51	6.4	3.2	4.5	1.5
82	5.8	2.7	3.9	1.2
101	5.8	2.7	5.1	1.9
5	5.4	3.9	1.7	0.4
107	7.3	2.9	6.3	1.8
4	5.0	3.6	1.4	0.2
93	5.0	2.3	3.3	1.0
95	5.7	3.0	4.2	1.2
6	4.6	3.4	1.4	0.3
103	6.3	2.9	5.6	1.8
78	6.0	2.9	4.5	1.5

SEARCH STACK OVERFLOW

Save the Model

```
model.save('mnist_lc_v2.h5')
```

Build and Train a Deep Neural network with 2 hidden layer - Optional - For Pract

Does it perform better than Linear Classifier? What could be the reason for difference in performar