Linear Classifier in TensorFlow

Using Low Level API in Eager Execution mode

Load tensorflow

```
!pip3 install -U tensorflow --quiet
```

```
     |████████████████████████████████| 86.3MB 118kB/s
     |████████████████████████████████| 3.8MB 35.3MB/s
     |████████████████████████████████| 450kB 22.6MB/s
     |████████████████████████████████| 81kB 6.2MB/s
ERROR: tensorboard 2.0.1 has requirement grpcio>=1.24.3, but you'll have grpcio 1.15.0 w
ERROR: google-colab 1.0.0 has requirement google-auth~=1.4.0, but you'll have google-aut
```

```
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')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318

Enter your authorization code:
··········
Mounted at /gdrive
```

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318

Enter your authorization code:
··········
Mounted at /content/drive
```

```
import pandas as pd
```

```
data = pd.read_csv('/gdrive/My Drive/greatlakes/Residency6/InternalLab/prices.csv')
```

Check all columns in the dataset

```
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
```

```
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()
```

	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 se

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

```
data.shape
```

```
(851264, 5)
```

```
data_1000 = data.loc[0:1000]
```

```
data_1000['volume'] = data_1000['volume'] / 1000000
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
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/user_g
  """Entry point for launching an IPython kernel.
```

```
data_1000.sample(4)
```

	open	close	low	high	volume
229	125.500000	124.370003	124.120003	126.000000	0.6767
563	34.590000	34.189999	34.000000	34.869999	5.3201
401	44.490002	44.790001	44.490002	45.250000	0.7742
634	33.919998	33.610001	33.380001	33.919998	1.5722

Divide the data into train and test sets

```
from sklearn.model_selection import train_test_split

data_Y = data_1000["volume"]
data_X = data_1000.drop(["volume"], axis=1)
X_train, X_test, y_train, y_test = train_test_split(data_X,data_Y, test_size = 0.2, random_st
```

Convert Training and Test Data to numpy float32 arrays

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 800 entries, 194 to 265
Data columns (total 4 columns):
open     800 non-null float64
close    800 non-null float64
low      800 non-null float64
high     800 non-null float64
dtypes: float64(4)
memory usage: 31.2 KB
```

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

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

```
X_train[0:1]
```

```
array([[130.52, 128.73, 128.17, 131.42]], dtype=float32)
```

Normalize the data

You can use Normalizer from sklearn.preprocessing

```
#from sklearn.preprocessing import Normalizer
from sklearn import preprocessing
X_train_normalized = preprocessing.normalize(X_train)
X_test_normalized = preprocessing.normalize(X_test)

#transformer = Normalizer()
#train_x_1 = transformer.fit_transform(X_train)
```

```
X_train_normalized[0:1]
```

```
array([[0.5030965 , 0.49619684, 0.4940383 , 0.50656563]], dtype=float32)
```

```
X_train_normalized.shape
```

```
(800, 4)
```

Building the Model in tensorflow

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

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

```
print('Current Loss on iteration', i, loss(y_train, prediction(X_train_normalized, w, b))
```

```
Current Loss on iteration 1 276.0609
Current Loss on iteration 2 273.92584
Current Loss on iteration 3 271.9583
Current Loss on iteration 4 270.14493
Current Loss on iteration 5 268.47354
Current Loss on iteration 6 266.93332
Current Loss on iteration 7 265.51385
Current Loss on iteration 8 264.20557
Current Loss on iteration 9 262.9999
Current Loss on iteration 10 261.8888
Current Loss on iteration 11 260.86484
Current Loss on iteration 12 259.9211
Current Loss on iteration 13 259.05154
Current Loss on iteration 14 258.24994
Current Loss on iteration 15 257.51105
Current Loss on iteration 16 256.8304
Current Loss on iteration 17 256.20297
Current Loss on iteration 18 255.6247
Current Loss on iteration 19 255.09183
Current Loss on iteration 20 254.60065
Current Loss on iteration 21 254.14813
Current Loss on iteration 22 253.73096
Current Loss on iteration 23 253.34648
Current Loss on iteration 24 252.99222
Current Loss on iteration 25 252.6656
Current Loss on iteration 26 252.3647
Current Loss on iteration 27 252.08728
Current Loss on iteration 28 251.83185
Current Loss on iteration 29 251.59625
Current Loss on iteration 30 251.37912
Current Loss on iteration 31 251.17912
Current Loss on iteration 32 250.99472
Current Loss on iteration 33 250.82483
Current Loss on iteration 34 250.66823
Current Loss on iteration 35 250.52396
Current Loss on iteration 36 250.39085
Current Loss on iteration 37 250.2683
Current Loss on iteration 38 250.15523
Current Loss on iteration 39 250.0512
Current Loss on iteration 40 249.9553
Current Loss on iteration 41 249.86682
Current Loss on iteration 42 249.78528
Current Loss on iteration 43 249.7103
Current Loss on iteration 44 249.64098
Current Loss on iteration 45 249.5772
Current Loss on iteration 46 249.51843
Current Loss on iteration 47 249.4642
Current Loss on iteration 48 249.41425
Current Loss on iteration 49 249.3682
Current Loss on iteration 50 249.32582
Current Loss on iteration 51 249.28677
Current Loss on iteration 52 249.25075
Current Loss on iteration 53 249.2175
Current Loss on iteration 54 249.1868
Current Loss on iteration 55 249.15872
Current Loss on iteration 56 249.13284
Current Loss on iteration 57 249.1088
```

```
Current Loss on iteration 58 249.08673
Current Loss on iteration 59 249.06645
Current Loss on iteration 60 249.04765
Current Loss on iteration 61 249.03024
Current Loss on iteration 62 249.0144
Current Loss on iteration 63 248.9997
Current Loss on iteration 64 248.98615
Current Loss on iteration 65 248.97375
Current Loss on iteration 66 248.96214
Current Loss on iteration 67 248.95164
Current Loss on iteration 68 248.94196
Current Loss on iteration 69 248.93295
Current Loss on iteration 70 248.92473
Current Loss on iteration 71 248.917
Current Loss on iteration 72 248.90994
Current Loss on iteration 73 248.9036
Current Loss on iteration 74 248.89755
Current Loss on iteration 75 248.89215
Current Loss on iteration 76 248.88695
Current Loss on iteration 77 248.88225
Current Loss on iteration 78 248.878
Current Loss on iteration 79 248.874
Current Loss on iteration 80 248.87035
Current Loss on iteration 81 248.86694
Current Loss on iteration 82 248.864
Current Loss on iteration 83 248.86095
Current Loss on iteration 84 248.85835
Current Loss on iteration 85 248.85585
Current Loss on iteration 86 248.85362
Current Loss on iteration 87 248.8516
Current Loss on iteration 88 248.84962
Current Loss on iteration 89 248.84793
Current Loss on iteration 90 248.8463
Current Loss on iteration 91 248.84496
Current Loss on iteration 92 248.8435
Current Loss on iteration 93 248.8421
Current Loss on iteration 94 248.841
Current Loss on iteration 95 248.83984
Current Loss on iteration 96 248.8389
Current Loss on iteration 97 248.838
Current Loss on iteration 98 248.8372
Current Loss on iteration 99 248.83635
```

Get the shapes and values of W and b

```
#Check Weights and Bias
print('Weights:\n', w.numpy())
print('Bias:\n',b.numpy())
```

```
Weights:
```

```
X_test_normalized[0].shape
```

```
(4,)
```

```
import numpy as np
np.expand_dims(X_test_normalized[0], axis=0).shape
```

```
(1, 4)
```

Model Prediction on 1st Examples in Test Dataset

```
prediction(np.expand_dims(X_test_normalized[0], axis=0), w, b).numpy()
```

```
array([[5.567032]], dtype=float32)
```

```
y_test[0]
```

```
10.3077
```

```
loss(y_test[0],prediction(np.expand_dims(X_test_normalized[0], axis=0), w, b)).numpy()
```

```
22.473936
```

Classification using tf.Keras

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

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

```
iris_data = pd.read_csv('/gdrive/My Drive/greatlakes/Residency6/InternalLab/Iris.csv')
```

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

```
iris_data.head(2)
```

	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

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()

iris_data['Species'] = le.fit_transform(iris_data['Species'])
```

```
list(le.classes_)
```

```
['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
```

```
iris_data_one_hot_encoded = pd.get_dummies(iris_data, columns=['Species'])
```

```
iris_data_one_hot_encoded.sample(4)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species_0	Species_
33	34	5.5	4.2	1.4	0.2	1	
111	112	6.4	2.7	5.3	1.9	0	
5	6	5.4	3.9	1.7	0.4	1	
126	127	6.2	2.8	4.8	1.8	0	

Splitting the data into feature set and target set

```
target = iris_data_one_hot_encoded[["Species_0","Species_1","Species_2"]]
features = iris_data_one_hot_encoded.drop(["Id","Species_0","Species_1","Species_2"], axis=1)
```

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.Dense(4))
#model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(3,input_shape=(4,), activation='softmax'))


#Compile the model
```

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

```
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(features,target, test
```

Model Training

```
model.fit(X_train_iris,y_train_iris,
          validation_data=(X_test_iris,y_test_iris),
          epochs=100,
          batch_size=10)
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapt
Train on 120 samples, validate on 30 samples
Epoch 1/100
120/120 [==============================] - 0s 211us/sample - loss: 0.5129 - accuracy: 0.
Epoch 2/100
120/120 [==============================] - 0s 219us/sample - loss: 0.4993 - accuracy: 0.
Epoch 3/100
120/120 [==============================] - 0s 183us/sample - loss: 0.5111 - accuracy: 0.
Epoch 4/100
120/120 [==============================] - 0s 188us/sample - loss: 0.4929 - accuracy: 0.
Epoch 5/100
120/120 [==============================] - 0s 196us/sample - loss: 0.4987 - accuracy: 0.
Epoch 6/100
120/120 [==============================] - 0s 194us/sample - loss: 0.4889 - accuracy: 0.
Epoch 7/100
120/120 [==============================] - 0s 190us/sample - loss: 0.4821 - accuracy: 0.
Epoch 8/100
120/120 [==============================] - 0s 191us/sample - loss: 0.4802 - accuracy: 0.
Epoch 9/100
120/120 [==============================] - 0s 197us/sample - loss: 0.4756 - accuracy: 0.
Epoch 10/100
120/120 [==============================] - 0s 188us/sample - loss: 0.4662 - accuracy: 0.
Epoch 11/100
120/120 [==============================] - 0s 193us/sample - loss: 0.4688 - accuracy: 0.
Epoch 12/100
120/120 [==============================] - 0s 196us/sample - loss: 0.4653 - accuracy: 0.
Epoch 13/100
120/120 [==============================] - 0s 192us/sample - loss: 0.4605 - accuracy: 0.
Epoch 14/100
120/120 [==============================] - 0s 178us/sample - loss: 0.4573 - accuracy: 0.
Epoch 15/100
120/120 [==============================] - 0s 174us/sample - loss: 0.4547 - accuracy: 0.
Epoch 16/100
120/120 [==============================] - 0s 194us/sample - loss: 0.4536 - accuracy: 0.
Epoch 17/100
120/120 [==============================] - 0s 178us/sample - loss: 0.4430 - accuracy: 0.
Epoch 18/100
120/120 [==============================] - 0s 194us/sample - loss: 0.4435 - accuracy: 0.
Epoch 19/100
120/120 [==============================] - 0s 185us/sample - loss: 0.4409 - accuracy: 0.
Epoch 20/100
120/120 [==============================] - 0s 206us/sample - loss: 0.4399 - accuracy: 0.
Epoch 21/100
120/120 [==============================] - 0s 187us/sample - loss: 0.4320 - accuracy: 0.
Epoch 22/100
120/120 [==============================] - 0s 210us/sample - loss: 0.4311 - accuracy: 0.
Epoch 23/100
120/120 [==============================] - 0s 196us/sample - loss: 0.4248 - accuracy: 0.
Epoch 24/100
120/120 [==============================] - 0s 185us/sample - loss: 0.4248 - accuracy: 0.
Epoch 25/100
120/120 [==============================] - 0s 233us/sample - loss: 0.4215 - accuracy: 0.
Epoch 26/100
120/120 [==============================] - 0s 172us/sample - loss: 0.4146 - accuracy: 0.
Epoch 27/100
120/120 [==============================] - 0s 179us/sample - loss: 0.4186 - accuracy: 0.
Epoch 28/100
```

```
120/120 [==============================] - 0s 189us/sample - loss: 0.4141 - accuracy: 0.
Epoch 29/100
120/120 [==============================] - 0s 193us/sample - loss: 0.4160 - accuracy: 0.
Epoch 30/100
120/120 [==============================] - 0s 189us/sample - loss: 0.4106 - accuracy: 0.
Epoch 31/100
120/120 [==============================] - 0s 219us/sample - loss: 0.4070 - accuracy: 0.
Epoch 32/100
120/120 [==============================] - 0s 227us/sample - loss: 0.4085 - accuracy: 0.
Epoch 33/100
120/120 [==============================] - 0s 190us/sample - loss: 0.4021 - accuracy: 0.
Epoch 34/100
120/120 [==============================] - 0s 195us/sample - loss: 0.4009 - accuracy: 0.
Epoch 35/100
120/120 [==============================] - 0s 195us/sample - loss: 0.3965 - accuracy: 0.
Epoch 36/100
120/120 [==============================] - 0s 206us/sample - loss: 0.3962 - accuracy: 0.
Epoch 37/100
120/120 [==============================] - 0s 210us/sample - loss: 0.3936 - accuracy: 0.
Epoch 38/100
120/120 [==============================] - 0s 264us/sample - loss: 0.3891 - accuracy: 0.
Epoch 39/100
120/120 [==============================] - 0s 194us/sample - loss: 0.3899 - accuracy: 0.
Epoch 40/100
120/120 [==============================] - 0s 226us/sample - loss: 0.3923 - accuracy: 0.
Epoch 41/100
120/120 [==============================] - 0s 189us/sample - loss: 0.3799 - accuracy: 0.
Epoch 42/100
120/120 [==============================] - 0s 185us/sample - loss: 0.3871 - accuracy: 0.
Epoch 43/100
120/120 [==============================] - 0s 178us/sample - loss: 0.3841 - accuracy: 0.
Epoch 44/100
120/120 [==============================] - 0s 219us/sample - loss: 0.3755 - accuracy: 0.
Epoch 45/100
120/120 [==============================] - 0s 194us/sample - loss: 0.3933 - accuracy: 0.
Epoch 46/100
120/120 [==============================] - 0s 206us/sample - loss: 0.3737 - accuracy: 0.
Epoch 47/100
120/120 [==============================] - 0s 193us/sample - loss: 0.3726 - accuracy: 0.
Epoch 48/100
120/120 [==============================] - 0s 218us/sample - loss: 0.3746 - accuracy: 0.
Epoch 49/100
120/120 [==============================] - 0s 201us/sample - loss: 0.3734 - accuracy: 0.
Epoch 50/100
120/120 [==============================] - 0s 212us/sample - loss: 0.3746 - accuracy: 0.
Epoch 51/100
120/120 [==============================] - 0s 207us/sample - loss: 0.3710 - accuracy: 0.
Epoch 52/100
120/120 [==============================] - 0s 221us/sample - loss: 0.3647 - accuracy: 0.
Epoch 53/100
120/120 [==============================] - 0s 229us/sample - loss: 0.3657 - accuracy: 0.
Epoch 54/100
120/120 [==============================] - 0s 213us/sample - loss: 0.3587 - accuracy: 0.
Epoch 55/100
120/120 [==============================] - 0s 191us/sample - loss: 0.3632 - accuracy: 0.
Epoch 56/100
120/120 [==============================] - 0s 226us/sample - loss: 0.3633 - accuracy: 0.
Epoch 57/100
```

```
120/120 [==============================] - 0s 177us/sample - loss: 0.3583 - accuracy: 0.
Epoch 58/100
120/120 [==============================] - 0s 202us/sample - loss: 0.3538 - accuracy: 0.
Epoch 59/100
120/120 [==============================] - 0s 206us/sample - loss: 0.3542 - accuracy: 0.
Epoch 60/100
120/120 [==============================] - 0s 214us/sample - loss: 0.3564 - accuracy: 0.
Epoch 61/100
120/120 [==============================] - 0s 259us/sample - loss: 0.3495 - accuracy: 0.
Epoch 62/100
120/120 [==============================] - 0s 176us/sample - loss: 0.3516 - accuracy: 0.
Epoch 63/100
120/120 [==============================] - 0s 198us/sample - loss: 0.3489 - accuracy: 0.
Epoch 64/100
120/120 [==============================] - 0s 203us/sample - loss: 0.3443 - accuracy: 0.
Epoch 65/100
120/120 [==============================] - 0s 198us/sample - loss: 0.3449 - accuracy: 0.
Epoch 66/100
120/120 [==============================] - 0s 241us/sample - loss: 0.3400 - accuracy: 0.
Epoch 67/100
120/120 [==============================] - 0s 211us/sample - loss: 0.3401 - accuracy: 0.
Epoch 68/100
120/120 [==============================] - 0s 188us/sample - loss: 0.3422 - accuracy: 0.
Epoch 69/100
120/120 [==============================] - 0s 190us/sample - loss: 0.3400 - accuracy: 0.
Epoch 70/100
120/120 [==============================] - 0s 226us/sample - loss: 0.3342 - accuracy: 0.
Epoch 71/100
120/120 [==============================] - 0s 219us/sample - loss: 0.3368 - accuracy: 0.
Epoch 72/100
120/120 [==============================] - 0s 240us/sample - loss: 0.3340 - accuracy: 0.
Epoch 73/100
120/120 [==============================] - 0s 230us/sample - loss: 0.3332 - accuracy: 0.
Epoch 74/100
120/120 [==============================] - 0s 217us/sample - loss: 0.3384 - accuracy: 0.
Epoch 75/100
120/120 [==============================] - 0s 240us/sample - loss: 0.3323 - accuracy: 0.
Epoch 76/100
120/120 [==============================] - 0s 198us/sample - loss: 0.3307 - accuracy: 0.
Epoch 77/100
120/120 [==============================] - 0s 186us/sample - loss: 0.3315 - accuracy: 0.
Epoch 78/100
120/120 [==============================] - 0s 202us/sample - loss: 0.3263 - accuracy: 0.
Epoch 79/100
120/120 [==============================] - 0s 248us/sample - loss: 0.3266 - accuracy: 0.
Epoch 80/100
120/120 [==============================] - 0s 197us/sample - loss: 0.3286 - accuracy: 0.
Epoch 81/100
120/120 [==============================] - 0s 225us/sample - loss: 0.3227 - accuracy: 0.
Epoch 82/100
120/120 [==============================] - 0s 226us/sample - loss: 0.3209 - accuracy: 0.
Epoch 83/100
120/120 [==============================] - 0s 181us/sample - loss: 0.3222 - accuracy: 0.
Epoch 84/100
120/120 [==============================] - 0s 187us/sample - loss: 0.3201 - accuracy: 0.
Epoch 85/100
120/120 [==============================] - 0s 204us/sample - loss: 0.3200 - accuracy: 0.
Epoch 86/100
```

```
Epoch 86/100
120/120 [==============================] - 0s 199us/sample - loss: 0.3194 - accuracy: 0.
Epoch 87/100
120/120 [==============================] - 0s 213us/sample - loss: 0.3143 - accuracy: 0.
Epoch 88/100
120/120 [==============================] - 0s 233us/sample - loss: 0.3151 - accuracy: 0.
Epoch 89/100
120/120 [==============================] - 0s 180us/sample - loss: 0.3148 - accuracy: 0.
Epoch 90/100
120/120 [==============================] - 0s 201us/sample - loss: 0.3127 - accuracy: 0.
Epoch 91/100
120/120 [==============================] - 0s 204us/sample - loss: 0.3135 - accuracy: 0.
Epoch 92/100
120/120 [==============================] - 0s 185us/sample - loss: 0.3119 - accuracy: 0.
```

Model Prediction

```
120/120 [==============================] - 0s 226us/sample - loss: 0.3072 - accuracy: 0.
```

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 3)                 15
=================================================================
Total params: 15
Trainable params: 15
Non-trainable params: 0
_________________________________________________________________
```

```
X_test_iris.shape
```

```
(30, 4)
```

```
prediction = model.predict(X_test_iris)
```

```
#Print prediction
print(prediction)
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapt
[[2.2566225e-02 6.8323326e-01 2.9420057e-01]
 [2.6325923e-03 3.1936601e-01 6.7800146e-01]
 [9.5510685e-01 4.4568695e-02 3.2445331e-04]
 [3.0539937e-02 5.5706555e-01 4.1239455e-01]
 [9.3341446e-01 6.5898858e-02 6.8661763e-04]
 [5.7799432e-02 6.5318578e-01 2.8901473e-01]
 [9.0625184e-03 4.9180940e-01 4.9912807e-01]
 [7.8766443e-02 6.6258359e-01 2.5864992e-01]
 [9.3507057e-01 6.4246647e-02 6.8277476e-04]
 [2.7088685e-02 4.9396789e-01 4.7894341e-01]
 [2.2149922e-02 4.3659803e-01 5.4125208e-01]
 [2.3446213e-03 2.3663661e-01 7.6101875e-01]
 [2.9035807e-02 6.0057396e-01 3.7039027e-01]
 [9.0537530e-01 9.3501136e-02 1.1235600e-03]
 [9.5287615e-01 4.6608739e-02 5.1512092e-04]
 [2.3850501e-03 3.3260635e-01 6.6500866e-01]
```

```
predicted_num = np.argmax(prediction[0])

#Print the number
print(predicted_num)
```

```
1
```

```
y_test_iris[0:2]
```

	Species_0	Species_1	Species_2
87	0	1	0
111	0	0	1

```
predicted_num = np.argmax(prediction[1])

#Print the number
print(predicted_num)
```

```
2
```

Save the Model

```
model.save('Iris_v1.h5')
```

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

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

```
#Initialize Sequential model
```

```
model_2 = tf.keras.models.Sequential()

#Normalize the data
model_2.add(tf.keras.layers.Dense(2,input_shape=(4,)))
#model_2.add(tf.keras.layers.BatchNormalization())
model_2.add(tf.keras.layers.Dense(3, activation='softmax'))
#Compile the model
model_2.compile(optimizer='sgd', loss='categorical_crossentropy',
              metrics=['accuracy'])


model_2.fit(X_train_iris,y_train_iris,
          validation_data=(X_test_iris,y_test_iris),
          epochs=100,
          batch_size=10)
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapt
Train on 120 samples, validate on 30 samples
Epoch 1/100
120/120 [==============================] - 0s 1ms/sample - loss: 2.4487 - accuracy: 0.42
Epoch 2/100
120/120 [==============================] - 0s 200us/sample - loss: 1.0792 - accuracy: 0.
Epoch 3/100
120/120 [==============================] - 0s 194us/sample - loss: 0.9728 - accuracy: 0.
Epoch 4/100
120/120 [==============================] - 0s 198us/sample - loss: 0.9513 - accuracy: 0.
Epoch 5/100
120/120 [==============================] - 0s 199us/sample - loss: 0.9298 - accuracy: 0.
Epoch 6/100
120/120 [==============================] - 0s 189us/sample - loss: 0.9028 - accuracy: 0.
Epoch 7/100
120/120 [==============================] - 0s 192us/sample - loss: 0.8963 - accuracy: 0.
Epoch 8/100
120/120 [==============================] - 0s 187us/sample - loss: 0.8669 - accuracy: 0.
Epoch 9/100
120/120 [==============================] - 0s 184us/sample - loss: 0.8448 - accuracy: 0.
Epoch 10/100
120/120 [==============================] - 0s 191us/sample - loss: 0.8204 - accuracy: 0.
Epoch 11/100
120/120 [==============================] - 0s 207us/sample - loss: 0.7993 - accuracy: 0.
Epoch 12/100
120/120 [==============================] - 0s 192us/sample - loss: 0.7702 - accuracy: 0.
Epoch 13/100
120/120 [==============================] - 0s 208us/sample - loss: 0.7521 - accuracy: 0.
Epoch 14/100
120/120 [==============================] - 0s 213us/sample - loss: 0.7301 - accuracy: 0.
Epoch 15/100
120/120 [==============================] - 0s 195us/sample - loss: 0.7181 - accuracy: 0.
Epoch 16/100
120/120 [==============================] - 0s 223us/sample - loss: 0.6885 - accuracy: 0.
Epoch 17/100
120/120 [==============================] - 0s 239us/sample - loss: 0.6728 - accuracy: 0.
Epoch 18/100
120/120 [==============================] - 0s 211us/sample - loss: 0.6550 - accuracy: 0.
Epoch 19/100
120/120 [==============================] - 0s 216us/sample - loss: 0.6446 - accuracy: 0.
Epoch 20/100
120/120 [==============================] - 0s 190us/sample - loss: 0.6270 - accuracy: 0.
Epoch 21/100
120/120 [==============================] - 0s 175us/sample - loss: 0.6115 - accuracy: 0.
Epoch 22/100
120/120 [==============================] - 0s 189us/sample - loss: 0.5980 - accuracy: 0.
Epoch 23/100
120/120 [==============================] - 0s 189us/sample - loss: 0.5901 - accuracy: 0.
Epoch 24/100
120/120 [==============================] - 0s 210us/sample - loss: 0.5768 - accuracy: 0.
Epoch 25/100
120/120 [==============================] - 0s 254us/sample - loss: 0.5678 - accuracy: 0.
Epoch 26/100
120/120 [==============================] - 0s 196us/sample - loss: 0.5558 - accuracy: 0.
Epoch 27/100
120/120 [==============================] - 0s 193us/sample - loss: 0.5495 - accuracy: 0.
Epoch 28/100
```

```
120/120 [==============================] - 0s 181us/sample - loss: 0.5418 - accuracy: 0.
Epoch 29/100
120/120 [==============================] - 0s 186us/sample - loss: 0.5326 - accuracy: 0.
Epoch 30/100
120/120 [==============================] - 0s 183us/sample - loss: 0.5282 - accuracy: 0.
Epoch 31/100
120/120 [==============================] - 0s 178us/sample - loss: 0.5204 - accuracy: 0.
Epoch 32/100
120/120 [==============================] - 0s 181us/sample - loss: 0.5121 - accuracy: 0.
Epoch 33/100
120/120 [==============================] - 0s 188us/sample - loss: 0.5114 - accuracy: 0.
Epoch 34/100
120/120 [==============================] - 0s 182us/sample - loss: 0.4993 - accuracy: 0.
Epoch 35/100
120/120 [==============================] - 0s 253us/sample - loss: 0.4943 - accuracy: 0.
Epoch 36/100
120/120 [==============================] - 0s 235us/sample - loss: 0.4875 - accuracy: 0.
Epoch 37/100
120/120 [==============================] - 0s 253us/sample - loss: 0.4895 - accuracy: 0.
Epoch 38/100
120/120 [==============================] - 0s 219us/sample - loss: 0.4804 - accuracy: 0.
Epoch 39/100
120/120 [==============================] - 0s 170us/sample - loss: 0.4754 - accuracy: 0.
Epoch 40/100
120/120 [==============================] - 0s 181us/sample - loss: 0.4719 - accuracy: 0.
Epoch 41/100
120/120 [==============================] - 0s 187us/sample - loss: 0.4639 - accuracy: 0.
Epoch 42/100
120/120 [==============================] - 0s 208us/sample - loss: 0.4586 - accuracy: 0.
Epoch 43/100
120/120 [==============================] - 0s 230us/sample - loss: 0.4525 - accuracy: 0.
Epoch 44/100
120/120 [==============================] - 0s 231us/sample - loss: 0.4514 - accuracy: 0.
Epoch 45/100
120/120 [==============================] - 0s 189us/sample - loss: 0.4487 - accuracy: 0.
Epoch 46/100
120/120 [==============================] - 0s 201us/sample - loss: 0.4412 - accuracy: 0.
Epoch 47/100
120/120 [==============================] - 0s 207us/sample - loss: 0.4386 - accuracy: 0.
Epoch 48/100
120/120 [==============================] - 0s 256us/sample - loss: 0.4326 - accuracy: 0.
Epoch 49/100
120/120 [==============================] - 0s 233us/sample - loss: 0.4308 - accuracy: 0.
Epoch 50/100
120/120 [==============================] - 0s 198us/sample - loss: 0.4237 - accuracy: 0.
Epoch 51/100
120/120 [==============================] - 0s 199us/sample - loss: 0.4194 - accuracy: 0.
Epoch 52/100
120/120 [==============================] - 0s 209us/sample - loss: 0.4206 - accuracy: 0.
Epoch 53/100
120/120 [==============================] - 0s 183us/sample - loss: 0.4140 - accuracy: 0.
Epoch 54/100
120/120 [==============================] - 0s 179us/sample - loss: 0.4142 - accuracy: 0.
Epoch 55/100
120/120 [==============================] - 0s 201us/sample - loss: 0.4060 - accuracy: 0.
Epoch 56/100
120/120 [==============================] - 0s 225us/sample - loss: 0.4021 - accuracy: 0.
Epoch 57/100
```

```
120/120 [==============================] - 0s 204us/sample - loss: 0.3986 - accuracy: 0.
Epoch 58/100
120/120 [==============================] - 0s 169us/sample - loss: 0.3973 - accuracy: 0.
Epoch 59/100
120/120 [==============================] - 0s 193us/sample - loss: 0.3890 - accuracy: 0.
Epoch 60/100
120/120 [==============================] - 0s 200us/sample - loss: 0.3895 - accuracy: 0.
Epoch 61/100
120/120 [==============================] - 0s 252us/sample - loss: 0.3834 - accuracy: 0.
Epoch 62/100
120/120 [==============================] - 0s 233us/sample - loss: 0.3793 - accuracy: 0.
Epoch 63/100
120/120 [==============================] - 0s 212us/sample - loss: 0.3737 - accuracy: 0.
Epoch 64/100
120/120 [==============================] - 0s 180us/sample - loss: 0.3712 - accuracy: 0.
Epoch 65/100
120/120 [==============================] - 0s 210us/sample - loss: 0.3684 - accuracy: 0.
Epoch 66/100
120/120 [==============================] - 0s 202us/sample - loss: 0.3639 - accuracy: 0.
Epoch 67/100
120/120 [==============================] - 0s 188us/sample - loss: 0.3634 - accuracy: 0.
Epoch 68/100
120/120 [==============================] - 0s 189us/sample - loss: 0.3569 - accuracy: 0.
Epoch 69/100
120/120 [==============================] - 0s 211us/sample - loss: 0.3547 - accuracy: 0.
Epoch 70/100
120/120 [==============================] - 0s 186us/sample - loss: 0.3523 - accuracy: 0.
Epoch 71/100
120/120 [==============================] - 0s 191us/sample - loss: 0.3440 - accuracy: 0.
Epoch 72/100
120/120 [==============================] - 0s 179us/sample - loss: 0.3402 - accuracy: 0.
Epoch 73/100
120/120 [==============================] - 0s 189us/sample - loss: 0.3362 - accuracy: 0.
Epoch 74/100
120/120 [==============================] - 0s 182us/sample - loss: 0.3378 - accuracy: 0.
Epoch 75/100
120/120 [==============================] - 0s 188us/sample - loss: 0.3304 - accuracy: 0.
Epoch 76/100
120/120 [==============================] - 0s 218us/sample - loss: 0.3275 - accuracy: 0.
Epoch 77/100
120/120 [==============================] - 0s 180us/sample - loss: 0.3210 - accuracy: 0.
Epoch 78/100
120/120 [==============================] - 0s 193us/sample - loss: 0.3227 - accuracy: 0.
Epoch 79/100
120/120 [==============================] - 0s 187us/sample - loss: 0.3163 - accuracy: 0.
Epoch 80/100
120/120 [==============================] - 0s 182us/sample - loss: 0.3105 - accuracy: 0.
Epoch 81/100
120/120 [==============================] - 0s 190us/sample - loss: 0.3059 - accuracy: 0.
Epoch 82/100
120/120 [==============================] - 0s 185us/sample - loss: 0.3065 - accuracy: 0.
Epoch 83/100
120/120 [==============================] - 0s 185us/sample - loss: 0.3068 - accuracy: 0.
Epoch 84/100
120/120 [==============================] - 0s 189us/sample - loss: 0.2969 - accuracy: 0.
Epoch 85/100
120/120 [==============================] - 0s 189us/sample - loss: 0.2913 - accuracy: 0.
Epoch 86/100
```

```
Epoch 86/100
120/120 [==============================] - 0s 182us/sample - loss: 0.2906 - accuracy: 0.
Epoch 87/100
120/120 [==============================] - 0s 178us/sample - loss: 0.2842 - accuracy: 0.
Epoch 88/100
120/120 [==============================] - 0s 185us/sample - loss: 0.2868 - accuracy: 0.
Epoch 89/100
120/120 [==============================] - 0s 249us/sample - loss: 0.2803 - accuracy: 0.
Epoch 90/100
120/120 [==============================] - 0s 196us/sample - loss: 0.2782 - accuracy: 0.
Epoch 91/100
120/120 [==============================] - 0s 173us/sample - loss: 0.2731 - accuracy: 0.
Epoch 92/100
120/120 [==============================] - 0s 184us/sample - loss: 0.2743 - accuracy: 0.
```

```
model_2.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 2)                 10
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 9
=================================================================
Total params: 19
Trainable params: 19
Non-trainable params: 0
_________________________________________________________________
120/120 [==============================] - 0s 265us/sample - loss: 0.2437 - accuracy: 0.
```