IBM PHASE 3 PROJECT

Tensorflow and Keras – ANN

TEAM MEMBERS (GROUP 1)

- DHARANI G
- GOPIKA SHREE G
- ILAKIYA V
- KEERTHIGA DEVI P
- KARTHIKA
- KOWSALYA R

Introduction to tensorflow

- TensorFlow is an open-source platform for machine learning and a symbolic math library that is used for machine learning applications

Introduction to Keras

- It is an Open Source Neural Network library that runs on top of Theano or Tensorflow. It is designed to be fast and easy for the user to use. It is a useful library to construct any deep learning algorithm of whatever choice we want.

Layers

- a basic Artificial neural network will be in a form of,
- Input layer – To get the data from the user or a client or a server to analyze and give the result.
- Hidden layers – This layer can be in any number and these layers will analyze the inputs with passing through them with different biases, weights, and activation functions to provide an output
- Output Layer – This is where we can get the result from a neural network

Syntax: tf.keras.layers.Dense()

The first stage of our model building is:

```
# Defining the model

model = keras.Sequential([

    keras.layers.Dense(32, input_shape=(2,), activation=‘relu’

    keras.layers.Dense(16, activation = ‘relu’),

    keras.layers.Dense(2, activation = ‘sigmoid’)
])
```

How to Train a Neural Network with TensorFlow :

Step 1: Importing the libraries

We are going to import the required libraries.

```
# Importing the libraries

import pandas as pd

import numpy as np

from tensorflow import keras

from tensorflow.keras import layers

from sklearn.model_selection import train_test_split
```

Step 2: Importing the data

The data we used for this example are generated randomly with Numpy. In this data x and y are the point of coordinates and the color feature is the target value that was generated randomly which is in binary representing Red – 1 , Blue – 0.

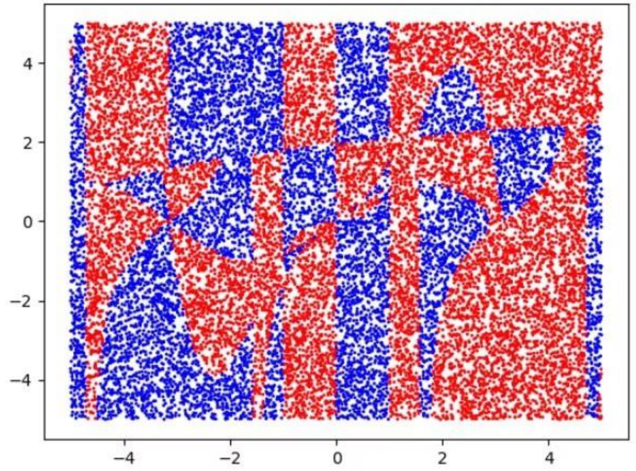
 # Importing the data

```
df = pd.read_csv('data.txt')
```

Out[14]:

	x	y	color
0	2.375386	-2.151675	0.0
1	0.155175	-3.939919	1.0
2	0.580631	-2.425793	1.0
3	2.045291	-2.755232	0.0
4	0.637783	-1.396165	1.0
...	...	...	...
19995	4.063291	-4.249531	0.0
19996	2.259423	1.168821	0.0
19997	-3.316201	3.792158	0.0
19998	0.378630	-4.874205	1.0
19999	0.488818	[illegible]	0.0

The data looks like



Step 3: Splitting the data

Now we are going to split the dataset into train and test splits to evaluate the model with the unseen data and check its accuracy.

```
# split the data into train and test set

train, test = train_test_split(

    df, test_size=0.2, random_state=42, shuffle=True)
```

Step 4: Constructing the input

In this step, we are going to construct the input we need to feed into a network. I
simplicity and for the model's sake we are going to stack the two features of the
data into x and the target variable as y. We use numpy.column_stack() to stack th

```
# Constructing the input

x = np.column_stack((train.x.values, train.y.values))

y = train.color.values
```

Step 5: Building a model

Now we are going to build a simple neural network to classify the color of the point with two in
nodes and a hidden layer and an output layer with relu and sigmoid activation functions, and sp
categorical cross-entropy loss function and this is going to be a fully connected feed-forward
network.

```
# Defining the model

model = keras.Sequential([

    keras.layers.Dense(4, input_shape=(2,), activation='relu'),

    keras.layers.Dense(2, activation='sigmoid')
])

# Compiling the model

model.compile(optimizer='adam'
.         loss=keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

# fitting the model

model.fit(x, y, epochs=10, batch_size=8)
```

```
Epoch 1/10
2000/2000 [==============================] - 5s 2ms/step - loss: 0.683
Epoch 2/10
2000/2000 [==============================] - 5s 2ms/step - loss: 0.643
Epoch 3/10
2000/2000 [==============================] - 5s 2ms/step - loss: 0.633
Epoch 4/10
2000/2000 [==============================] - 5s 2ms/step - loss: 0.624
Epoch 5/10
2000/2000 [==============================] - 5s 2ms/step - loss: 0.615
Epoch 6/10
2000/2000 [==============================] - 5s 2ms/step - loss: 0.611
Epoch 7/10
2000/2000 [==============================] - 5s 2ms/step - loss: 0.608
Epoch 8/10
2000/2000 [==============================] - 5s 2ms/step - loss: 0.606
Epoch 9/10
2000/2000 [==============================] - 5s 2ms/step - loss: 0.605
Epoch 10/10
2000/2000 [==============================] - 5s 2ms/step - loss: 0.605
```

If we evaluate the model with unseen data it will give a very low amount of accuracy,

```
# Evaluating the model

x = np.column_stack((test.x.values, test.y.values))

y = test.color.values

model.evaluate(x, y, batch_size=8)
```

```
500/500 [==============================] - 1s 2ms/step - loss: 0.6190 - accuracy:
Out[13]: [0.6190415620803833, 0.6669999957084656]
```

Step 6: Building a better model

Now we are going to improve the model with a few extra hidden layers and a better activation function 'softmax' in the output layer and built a better neural network.

Python

Defining the model

```
model_better = keras.Sequential([
    keras.layers.Dense(16, input_shape=(2,), activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(2, activation='softmax')
])

# Compiling the model
model_better.compile(optimizer='adam'
  loss=keras.losses.SparseCategoricalCrossentropy(),
                metrics=['accuracy'])

# Constructing the input
x = np. Column_stack((train.x.values, train.y.values))
y = train.color.values

# fitting the model
model_better.fit(x, y, epochs=10, batch_size=8)
```

```
Epoch 1/10
2000/2000 [==============================] - 3s 1ms/step - loss: 0.5867 - accuracy: 0.6747
Epoch 2/10
2000/2000 [==============================] - 3s 1ms/step - loss: 0.5334 - accuracy: 0.7393: 0s - loss: 0.5347 - accuracy: 0.
Epoch 3/10
2000/2000 [==============================] - 3s 2ms/step - loss: 0.5015 - accuracy: 0.7558
Epoch 4/10
2000/2000 [==============================] - 3s 1ms/step - loss: 0.4773 - accuracy: 0.7740
Epoch 5/10
2000/2000 [==============================] - 3s 1ms/step - loss: 0.4506 - accuracy: 0.7914
Epoch 6/10
2000/2000 [==============================] - 3s 1ms/step - loss: 0.4255 - accuracy: 0.8039
Epoch 7/10
2000/2000 [==============================] - 3s 2ms/step - loss: 0.4015 - accuracy: 0.8187
Epoch 8/10
2000/2000 [==============================] - 3s 2ms/step - loss: 0.3792 - accuracy: 0.8307
Epoch 9/10
2000/2000 [==============================] - 3s 1ms/step - loss: 0.3600 - accuracy: 0.8428 - ETA: 0s - loss: 0.3607 - accu
Epoch 10/10
2000/2000 [==============================] - 3s 2ms/step - loss: 0.3416 - accuracy: 0.8498: 1s
```

Step 7: Evaluating the model

Finally, if we evaluate the model we can clearly see that the accuracy of the model on unseen data has been improved from 66 -> 85. So we built an efficient model

```
500/500 [==============================] - 1s 1ms/step - loss: 0.3104 - accuracy: 0.8550
[0.310432106256485, 0.8550000190734863]
```

Thank you