```
try:
 from google.colab import drive
 %tensorflow_version 2.x
 COLAB = True
 print("Assignment 8")
 print("Note: using Google CoLab")
except:
 print("Assignment 8")
 print("Note: not using Google CoLab")
 COLAB = False

# Print your name and Roll No.
print('rohit byas ')
print('181210043')
# Print the curent time
import datetime
print(datetime.datetime.now())
```

```
Assignment 8
Note: using Google CoLab
rohit byas
181210043
2021-03-15 09:34:42.847342
```

```
# Import required libraries :
import numpy as np
```

Double-click (or enter) to edit

PART 10.1: Neural Network

CASE STUDY 1: Predicting Virus Contraction with a Artifical Neural Net

Summarizing an Artificial Neural Network:

1. Take inputs
2. Add bias (if required)
3. Assign random weights to input features
4. Run the code for training.
5. Find the error in prediction.
6. Update the weight by gradient descent algorithm.
7. Repeat the training phase with updated weights.
8. Make predictions

```
# Define input features :
input_features =np.array([
                          [1,0,0,1],
                          [1,0,0,0],
```

```
                              [0,0,1,1],
                              [0,1,0,0],
                              [1,1,0,0],
                              [0,0,1,1],
                              [0,0,0,1],
                              [0,0,1,0]
                              ])
print (input_features.shape)
print (input_features)
```

```
(8, 4)
[[1 0 0 1]
 [1 0 0 0]
 [0 0 1 1]
 [0 1 0 0]
 [1 1 0 0]
 [0 0 1 1]
 [0 0 0 1]
 [0 0 1 0]]
```

```
# Define target output :
target_output = np.array([1,1,0,0,1,1,0,0])# WRITE YOUR CODE HERE

# Reshaping our target output into vector :
target_output = target_output.reshape(8,1)
print(target_output.shape)
print (target_output)
```

```
(8, 1)
[[1]
 [1]
 [0]
 [0]
 [1]
 [1]
 [0]
 [0]]
```

```
# Define weights :
weights = np.array([[0.1],[0.2],[0.3],[0.4]])# WRITE YOUR CODE HERE


print(weights.shape)
print (weights)
```

```
(4, 1)
[[0.1]
 [0.2]
 [0.3]
 [0.4]]
```

```
# Bias weight :
bias = 0.3
# Learning Rate :
lr = 0.05
```

```
# Sigmoid function :
def sigmoid(x):
  return 1/(1+np.exp(-x))# WRITE YOUR CODE HERE


# Derivative of sigmoid function :
def sigmoid_der(x):
  return sigmoid(x)*(1-sigmoid(x))


# Main logic for neural network :
# Running our code 10000 times :
for epoch in range(10000):
  inputs = input_features
  #Feedforward input :
  pred_in = np.dot(inputs, weights) + bias
  #Feedforward output :
  pred_out = sigmoid(pred_in)
  #Backpropogation
  #Calculating error
  error = pred_out-target_output
  #Going with the formula :
  x = error.sum()
  print(x)
  #Calculating derivative :
  dcost_dpred = error
  dpred_dz = sigmoid_der(pred_out)
  #Multiplying individual derivatives :
  z_delta = dcost_dpred * dpred_dz
  #Multiplying with the 3rd individual derivative :
  inputs = input_features.T
  weights -= lr * np.dot(inputs, z_delta)
  #Updating the bias weight value :


for i in z_delta:
  bias -= lr * i
```

```
Streaming output truncated to the last 5000 lines.
0.4435029631688819
0.4435018248112544
0.4435006868777969
0.4434995493682664
0.4434984122824193
0.4434972756200135
0.4434961393808057
0.4434950035645542
0.44349386817101644
0.44349273319995075
0.44349159865111476
0.443490464524267
0.4434893308191656
0.44348819753557017
0.4434870646732397
0.44348593223193183
0.4434848002114077
```

```
0.44348366861142496
0.4434825374317449
0.4434814066721268
0.4434802763323305
0.44347914641211583
0.4434780169112439
0.44347688782947525
0.44347575916657056
0.4434746309222909
0.443473503096397
0.4434723756886502
0.44347124869881305
0.4434701221266458
0.44346899597191186
0.44346787023437195
0.4434667449137889
0.44346562000992557
0.44346449552254397
0.44346337145140746
0.4434622477962786
0.44346112455692066
0.44346000173309735
0.44345887932457145
0.4434577573311078
0.44345663575246963
0.4434555145884206
0.4434543938387266
0.44345327350315067
0.4434521535814577
0.443451034073413
0.44344991497878145
0.4434487962973274
0.44344767802881774
0.443446560173017
0.44344544272969055
0.44344432569860565
0.44344320907952745
0.44344209287222247
0.44344097707645774
0.4434398616919986
0.4434387467186131
```

```
#Taking inputs :
single_point = np.array([1,0,0,1])#1st step :
result1 = np.dot(single_point, weights) + bias#2nd step :
result2 = sigmoid(result1)
#Print final result
print(result2)
```

```
[0.99943686]
```

```
#Taking inputs :
single_point = np.array([0,0,1,0])#1st step :
result1 = np.dot(single_point, weights) + bias#2nd step :
result2 = sigmoid(result1)#Print final result
print(result2)
```

```
[0.42555372]
```

```
#Taking inputs :
single_point = np.array([1,0,1,0])#1st step :
result1 = np.dot(single_point, weights) + bias#2nd step :
result2 = sigmoid(result1)#Print final result
print(result2)
```

```
[0.9994356]
```

```
#Printing final weights:
print (weights)
print ("\n\n")
print (bias)
```

```
[[ 7.77919291]
 [-4.0898514 ]
 [-0.59482263]
 [-0.59258707]]



[0.29480724]
```

observation

The input feature “loss of smell” influences the output the most. If it is true, then there is high chance that person tests positive for the virus. The input feature “Weight loss” is not affecting the output much so we can remove that.

+ Code + Text