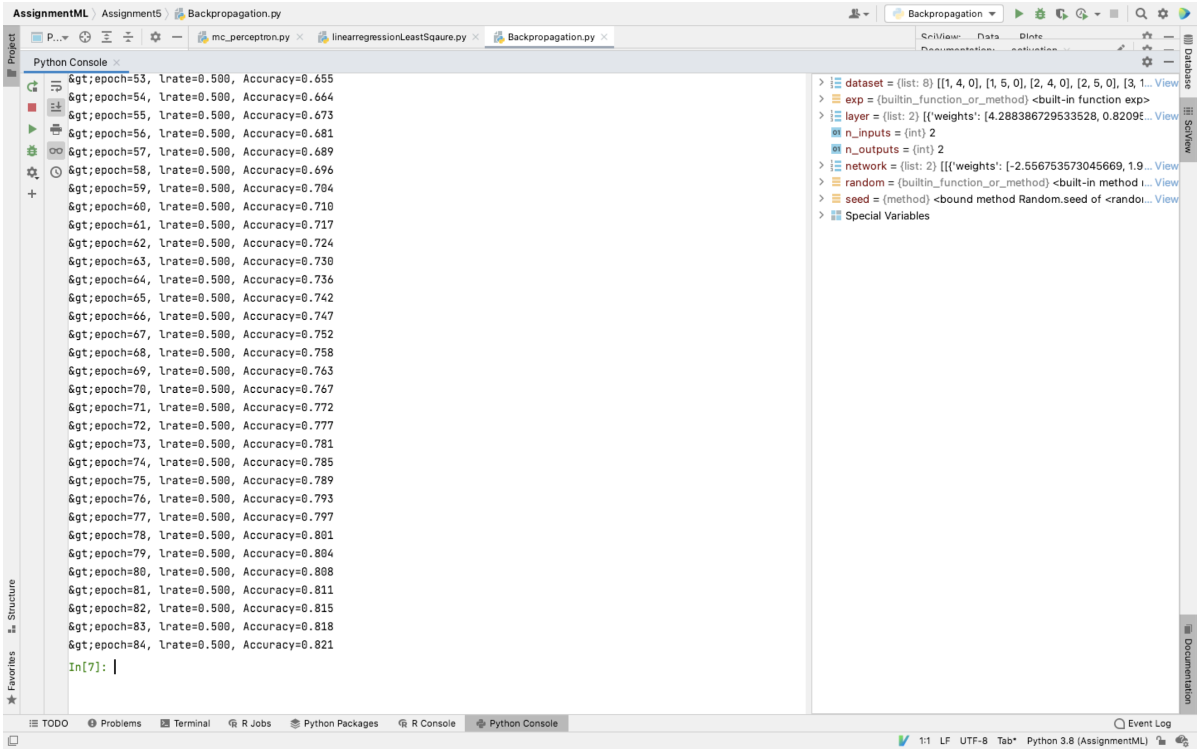
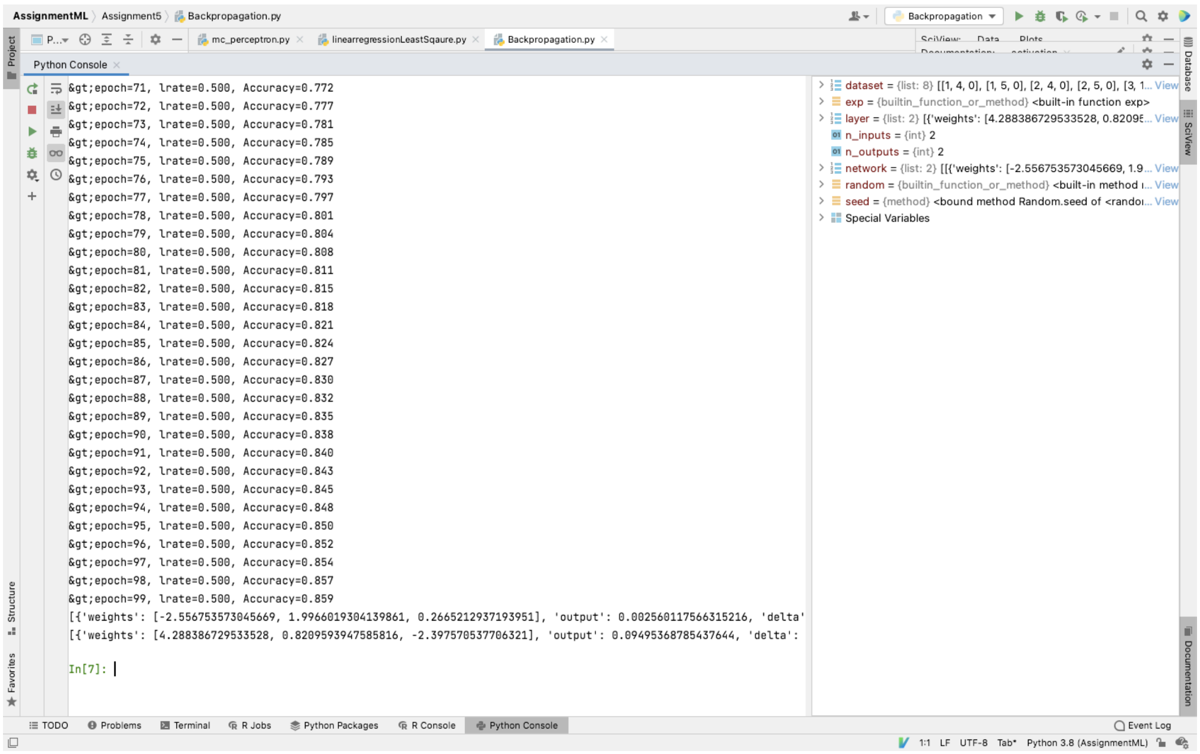
ASSIGNMENT 5

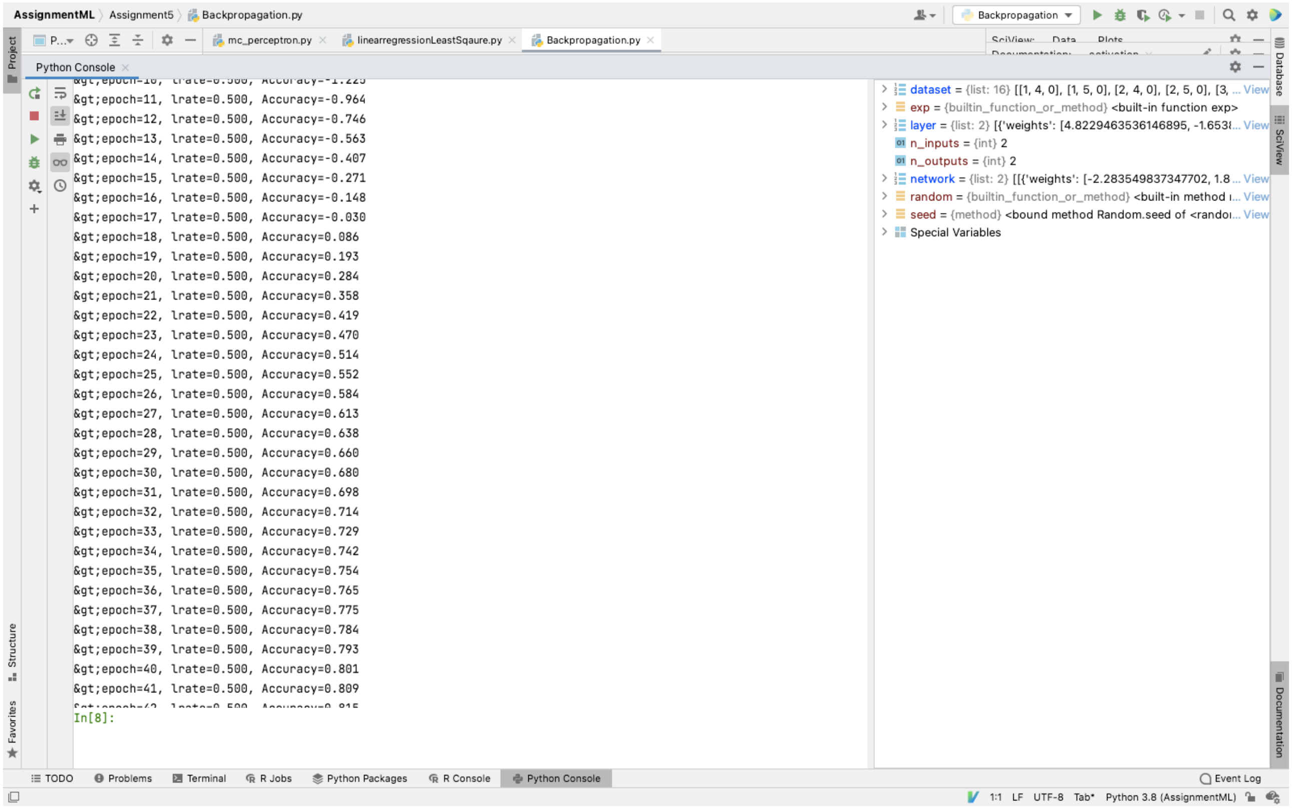
Question 1:

a) Initial Accuracy of the code with given inputs





b) Initial Accuracy of the code with some more inputs



c) When we increases the Learning rate from 0.05 to 0.5, the sum_error increase too much and results in degradation of Accuracy of the system.

```
from math import exp
from random import seed
from random import random

# Step 1 Initialise the network
def initialise_net(no_input, no_hidden, no_output):
    net = list()
    hid_layer = [{'weights':[random() for i in range(no_input + 1)]} for i in range(no_hidden)]
    net.append(hid_layer)
```

```
    out_layer = [{'weights':[random() for i in range(no_hidden + 1)]} for i in range(no_output)]
    net.append(out_layer)
    return net

# Step2 Calculate Activation Function
def activate(weight, input):
    activation = weight[-1]
    for i in range(len(weight)-1):
        activation += weight[i] * input[i]
    return activation

# Step 3 we have to transfer the activation of neutron
def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))

#Step 4 Forward propagated input to the neural outpur
def forward_propagate(net, r):
    input = r
    for layer in net:
        new_input = []
        for neuron in layer:
            activation = activate(neuron['weight'], input)
            neuron['output'] = transfer(activation)
            new_input.append(neuron['output'])
        input = new_input
    return input

# Step 5 Calculate the derivative of an neuron output
def transfer_derivative(output):
    return output * (1.0 - output)

#Step 6 Backpropagate error and store in neurons
def back_pro_err(net, expect):
```

```
    for i in reversed(range(len(net))):
        layer = net[i]
        err = list()
        if i != len(net)-1:
            for j in range(len(layer)):
                err1= 0.0
                for neuron in network[i + 1]:
                    err1 += (neuron['weights'][j] * neuron['delta'])
                err.append(err1)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])

# Step 7
def update_weights(net, r, learn_rate):
    for i in range(len(net)):
        input = r[:-1]
        if i != 0:
            input = [neuron['output'] for neuron in network[i - 1]]
        for neuron in net[i]:
            for j in range(len(input)):
                neuron['weights'][j] += learn_rate * neuron['delta'] * input[j]
            neuron['weights'][-1] += l_rate * neuron['delta']

# Train a network for a fixed number of epochs
def train_network(net, train, learn_rate, no_epochs, no_output):
    for epoch in range(no_epochs):
        sum_err = 0
        for r in train:
```

```
                out = forward_propagate(net, r)
                expected = [0 for i in range(no_output)]
                expected[r[-1]] = 1
                sum_error += sum([(expected[i]-outputs[i])**2 for i in
range(len(expected))])
                back_pro_err(net, expected)
                update_weights(net, r, learn_rate)
          print('epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))

# Test training backprop algorithm
seed(1)
dataset = [[1,4,0],
    [1,5,0],
    [2,4,0],
    [2,5,0],
    [3,1,1],
    [3,2,1],
    [4,1,1],
    [4,2,1]]
no_input = len(dataset[0]) - 1
no_output = len(set([row[-1] for row in dataset]))
net = initialise_net(no_input, 2, no_output)
train_network(network, dataset, 0.5, 100, no_outputs)
for layer in network:
    print(layer)
```

Question 2)

```
-1.0 1.0 -2.0 -1.0 1.0 -4.0 1 -1
-2.0 -2.0 -2.0 -1.0 0.0 -3.0 1 -1
1.0 1.0 -2.0 -1.0 0.0 -3.0 1 -1
1.0 2.0 -2.0 -1.0 0.0 -3.0 1 -1
2.0 -1.0 -2.0 -1.0 0.0 -3.0 1 -1
2.0 0.0 -2.0 -1.0 0.0 -3.0 1 -1
-1.0 2.0 -2.0 -1.0 2.0 -3.0 1 -1
-2.0 1.0 -2.0 -1.0 2.0 -3.0 1 -1
-1.0 1.0 -2.0 -1.0 2.0 -3.0 1 -1
-2.0 -2.0 -2.0 -1.0 1.0 -2.0 1 -1
1.0 1.0 -2.0 -1.0 1.0 -2.0 1 -1
1.0 2.0 -2.0 -1.0 1.0 -2.0 1 -1
2.0 -1.0 -2.0 -1.0 1.0 -2.0 1 -1
2.0 0.0 -2.0 -1.0 1.0 -2.0 1 -1
-1.0 2.0 -2.0 -1.0 1.0 -2.0 1 -1
-2.0 1.0 -2.0 -1.0 1.0 -2.0 1 -1
-1.0 1.0 -2.0 -1.0 1.0 -2.0 1 -1
-2.0 -2.0 -2.0 -1.0 0.0 -1.0 1 -1
1.0 1.0 -2.0 -1.0 0.0 -1.0 1 -1
1.0 2.0 -2.0 -1.0 0.0 -1.0 1 -1
2.0 -1.0 -2.0 -1.0 0.0 -1.0 1 -1
2.0 0.0 -2.0 -1.0 2.0 -2.0 1 -1
-1.0 2.0 -2.0 -1.0 2.0 -2.0 1 -1
-2.0 1.0 -2.0 -1.0 2.0 -2.0 1 -1
-1.0 1.0 -2.0 -1.0 2.0 -2.0 1 -1
-2.0 -2.0 -2.0 -1.0 1.0 -1.0 1 -1
1.0 1.0 -2.0 -1.0 -1.0 -3.0 1 -1
1.0 2.0 -2.0 -1.0 -1.0 -3.0 1 -1
2.0 -1.0 -2.0 -1.0 -1.0 -3.0 1 -1
2.0 0.0 -2.0 -1.0 1.0 -4.0 1 -1
-1.0 2.0 -2.0 -1.0 1.0 -4.0 1 -1
-2.0 1.0 -2.0 -1.0 1.0 -4.0 1 -1
-1.0 1.0 -2.0 -1.0 1.0 -4.0 1 -1
-2.0 -2.0 -2.0 -1.0 0.0 -3.0 1 -1
```

+ Code + Text

```
[18] 1.0 1.0 -0.5200000000000002 -1.6699999999999997 1.8599999999999999 -1.9999999999999998 1 1
     1.0 2.0 -0.5200000000000002 -1.6699999999999997 1.8499999999999999 -2.01 1 1
     2.0 -1.0 -0.5200000000000002 -1.6699999999999997 1.8499999999999999 -2.01 1 1
```

best parameters:

```
[19] max_accuracy, w1_max, w2_max, w3_max, w4_max, b1_max, b2_max, lr_max
```

```
(0.875, -3, -2, -2, -1, -1, -1, 1)
```

On the Test Data:

```
[24] # test dataset
     X_test = [[1,1],[1,2],[2,-1],[2,0],[-1,2],[-2,1],[-1,1],[-2,-2], [-3,-2],[-4,-2]]
     y_test = [1, 1, 1, 1, 3, 3, 3, 4, 4, 4]
```

```
def test_function(w1, w2, w3, w4, b1, b2):
  # SET ALL PARAMETERS AND VARIABLES
  x = X_test # input
  target = y_test # output

  # threshold
  threshold1 = 0.5
  threshold2 = 0.5

  # BASIC PERCEPTRON LEARNING PROCESS
  out_test = []
  for i in range(len(x)): # loop for each row in x
    x1 = float(x[i][0])
    x2 = float(x[i][1])
```

✓ 0s completed at 22:10

+ Code + Text

RAM Disk Editing

```
test_function(w1_max, w2_max, w3_max, w4_max, b1_max, b2_max)
```

```
1.0 1.0 -3 -2 -2 -1 -1 -1
1.0 2.0 -3 -2 -2 -1 -1 -1
2.0 -1.0 -3 -2 -2 -1 -1 -1
2.0 0.0 -3 -2 -2 -1 -1 -1
-1.0 2.0 -3 -2 -2 -1 -1 -1
-2.0 1.0 -3 -2 -2 -1 -1 -1
-1.0 1.0 -3 -2 -2 -1 -1 -1
-2.0 -2.0 -3 -2 -2 -1 -1 -1
-3.0 -2.0 -3 -2 -2 -1 -1 -1
-4.0 -2.0 -3 -2 -2 -1 -1 -1

 TEST RESULTS:

Final weights:-3,-2,-2,-1
Final predicted output: [1, 1, 1, 1, 1, 4, 1, 4, 4, 4]
accuracy: 0.7
0.7
```

✓ 0s completed at 22:10

```
#import required packages
%matplotlib inline
import os
import numpy as np
import pandas as pd
from sklearn.metrics import precision_score,
recall_score, accuracy_score, confusion_matrix
import seaborn as sns
from sklearn import preprocessing
# Inputs and labels
X_data = [[1,1],[1,2],[2,-1],[2,0],[-1,2],[-2,1],[-1,1],
[-2,-2]]
Y_data = [1,1,2,2,3,3,4,4]
def calculate_y_in(x_1, x_2, w_1, w_2, w_3, w_4, b_1,
b_2):
  y_in_1 = w_1 * x_1 + w_2 * x_2  + b_1
  y_in_2 = w_3 * x_1 + w_4 * x_2  + b_2
  print(x_1, x_2, w_1, w_2, w_3, w_4, b_1, b_2)
  return y_in_1, y_in_2

#Update Weight
def update_w(w_1, x_1, w_2, x_2, w_3, w_4, l_r, z):
  w_1_new = w_1 + l_r * z[0] * x_1
  w_2_new = w_2 + l_r * z[0] * x_2
  w_3_new = w_3 + l_r * z[1] * x_1
  w_4_new = w_4 + l_r * z[1] * x_2

  return w_1_new, w_2_new, w_3_new, w_4_new

def class_to_category(y):
  if y == 1: return np.array([0, 0])
  if y == 2: return np.array([0, 1])
  if y == 3: return np.array([1, 0])
  return np.array([1, 1])

# Activation_Function
def activation_function(y_in):
  sigmoid = (1/(1 + np.exp(-y_in)))
  return sigmoid
def train_function(w_1, w_2, w_3, w_4, b_1, b_2, l_r):
  x = X_data
  target = Y_data
```

```
  threshold1 = 0.5
  threshold2 = 0.5
  epoch = 20

  for e in range(epoch): # loop for each epoch
    out = []
    for i in range(len(x)): # loop for each row in x
      x_1 = float(x[i][0])
      x_2 = float(x[i][1])

      y_in_1, y_in_2 = calculate_y_in(x_1, x_2, w_1, w_2,
w_3, w_4, b_1, b_2) # calculate yin
      y_act_1 = activation_function(y_in_1)
      y_act_2 = activation_function(y_in_2)

      if y_act_1 > threshold1 and y_act_2 > threshold2:
        y = 4
      elif  y_act_1 > threshold1 and y_act_2 <
threshold2:
        y = 3
      elif y_act_1 < threshold1 and y_act_2 > threshold2:
        y = 2
      else:
        y = 1

      if y != target[i]:
        loss = class_to_category(target[i]) -
class_to_category(y)
        w_1, w_2, w_3, w_4 = update_w(w_1, x_1, w_2, x_2,
w_3, w_4, l_r, loss)
      out.append(y) # append all prediction in 'out'

  # performance evaluation
  accuracy = accuracy_score(target,out)
  print("Weights:[{:.2f},{:.2f},{:.2f},{:.2f}] and
Accuracy:{:.2f}".format(w_1,w_2,w_3,w_4,accuracy))
  return accuracy
w1_a = [-3, -2, 0, 1]
w2_a = [-2, -1, 1, 2]
w3_a = [-2, -1, 1, 2]
w4_a = [-2, -1, 0, 2]
```

```
b1_a = [-1, 1]
b2_a = [-1, 1]
lr_a = [0.001, 0.005, 0.05, 1,]

maximum_accuracy = 0
w_1_maximum = 0
w_2_maximum = 0
w_3_maximum = 0
w_4_maximum = 0
b_1_maximum = 0
b_2_maximum = 0
l_r_maximum = 0

for w_1 in w_1_a:
  for w_2 in w_2_a:
    for w_3 in w_3_a:
      for w_4 in w_4_a:
        for b_1 in b_1_a:
          for b_2 in b_2_a:
            for l_r in l_r_a:
              accuracy = train_function(w_1, w_2, w_3,
w_4, b_1, b_2, lr)
              if accuracy > max_accuracy:
                max_accuracy = accuracy
                w_1_maximum = w_1
                w_2_maximum = w_2
                w_3_maximum = w_3
                w_4_maximum = w_4
                b_1_maximum = b_1
                b_2_maximum = b_2
                l_r_maximum = l_r
maximum_accuracy, w_1_maximum, w_2_maximum, w_3_maximum,
w_4_maximum, b_1_maximum, b_2_maximum, l_r_maximum



#Testing
X_test = [[1,1],[1,2],[2,-1],[2,0],[-1,2],[-2,1],[-1,1],
[-2,-2], [-3,-2],[-4,-2]]
y_test = [1, 1, 2, 2, 3, 3, 3, 4, 4, 4]
```

```
print(maximum_accuracy, w_1_maximum, w_2_maximum,
w_3_maximum, w_4_maximum, b_1_maximum, b_2_maximum,
l_r_maximum)

def test_function(w_1, w_2, w_3, w_4, b_1, b_2):
  # SET ALL PARAMETERS AND VARIABLES
  x = X_test # input
  target = y_test # output

  # threshold
  threshold1 = 0.5
  threshold2 = 0.5

  # BASIC PERCEPTRON LEARNING PROCESS
  out_test = []
  for i in range(len(x)): # loop for each row in x
    x_1 = float(x[i][0])
    x_2 = float(x[i][1])

    y_in_1, y_in_2 = calculate_y_in(x_1, x_2, w_1, w_2,
w_3, w_4, b_1, b_2) # calculate yin
    y_act_1 = activation_function(yin1)
    y_act_2 = activation_function(yin2)

    if y_act_1 > threshold1 and y_act_2 > threshold2:
      y = 4
    elif  y_act_1 > threshold1 and y_act_2 < threshold2:
      y = 3
    elif y_act_1 < threshold1 and y_act_2 > threshold2:
      y = 2
    else:
      y = 1
    out_test.append(y) # append all prediction in 'out'

  print("\n TEST RESULTS:\n")
  print("Final weights:{},{},{},{}".format(w1,w2,w3,w4))
  print("Final predicted output:",out_test)

  # performance evaluation
  accuracy = accuracy_score(y_test,out_test)
  print("accuracy:",accuracy)
```

```
    return accuracy

test_function(w_1_maximum, w_2_maximum, w_3_maximum,
w_4_maximum, b_1_maximum, b2_maximum)
```

Question 3)

Insert Runtime Tools Help

+ Code + Text

RAM Disk Editing

l
son
using_test.csv
using_train.csv
sv
small.csv

61.54 GB available

```
Train_y
```

```
array([6., 5., 5., 5., 5., 6., 6., 6., 6., 6., 5., 6., 6., 5., 6., 6., 5.,
       5., 5., 5., 5., 6., 6., 6., 6., 5., 6., 6., 5., 5., 5., 5., 5., 6.,
       6., 5., 5., 5., 5., 6., 5., 6., 5., 6., 6., 6., 6., 5., 5., 5., 6.,
       6., 5., 5., 5., 6., 5., 6., 6., 5., 5., 6., 6., 5., 6., 5., 5., 6.,
       5., 5., 6., 5., 5., 6., 5., 6., 6., 5., 5., 5., 6., 5., 5., 6., 5.,
       5., 5., 6., 6., 6., 5., 5., 6., 5., 5., 5., 6., 6., 6., 6., 5., 6.,
       5., 5., 5., 6., 5., 6., 5., 6., 6., 6., 5., 6., 5., 6., 6., 6., 6.,
       6., 6., 5., 6., 6., 5., 6., 6., 6., 5., 6., 6., 5., 5., 5., 5., 5.,
       6., 5., 5., 5., 6., 6., 6., 5., 6., 6., 5., 6., 6., 6., 6., 6., 6.,
       5., 5., 6., 5., 6., 5., 6., 6., 5., 6., 6., 5., 5., 5., 6., 6., 6.,
       5., 5., 6., 6., 5., 6., 6., 5., 5., 5., 5., 5., 6., 6., 6., 6., 6.,
       5., 6., 5., 6., 5., 5., 5., 6., 5., 6., 5., 5., 6., 5., 6., 6., 5.,
       5., 5., 5., 5., 6., 5., 5., 5., 5., 6., 6., 5., 5., 5., 6., 6., 6.,
       5., 6., 5., 6., 6., 5., 5., 5., 6., 6., 5., 6., 6., 5., 5., 5., 6.,
       5., 5., 5., 5., 5., 6., 6., 5., 5., 5., 6., 5., 5., 5., 5., 5., 6.,
       5., 5., 6., 5., 5., 5., 5., 6., 5., 6., 5., 6., 6., 5., 5., 6., 6.,
       6., 6., 6., 6., 5., 5., 5., 6., 5., 6., 5., 6., 6., 6., 5., 6., 6.,
       5., 6., 6., 5., 6., 6., 6., 6., 6., 5., 5., 6., 6., 6., 6., 5., 6.,
       6., 5., 6., 6., 6., 5., 6., 6., 6., 6., 5., 6., 5., 5., 5., 6., 5.,
       6., 5., 6., 5., 5., 5., 6., 6., 5., 6., 6., 5., 6., 6., 6., 6., 6.,
       6., 5., 5., 6., 6., 6., 5., 6., 5., 6., 6., 5., 6., 5., 5., 6., 5.,
       5., 6., 6., 5., 5., 5., 6., 5., 6., 6., 6., 6., 6., 6., 6., 5., 6.,
       5., 5., 6., 6., 5., 5., 5., 5., 5., 6., 6., 6., 5., 5., 5., 5., 5.,
       5., 6., 6., 5., 6., 5., 5., 6., 6., 5., 6., 5., 5., 6., 6., 6., 6.,
       5., 5., 6., 5., 5., 6., 5., 6., 5., 6., 6., 6., 5., 5., 5., 6., 5.,
       6., 6., 6., 6., 5., 5., 5., 5., 6., 5., 5., 5., 6., 5., 5., 5., 6.,
       5., 6., 6., 5., 6., 5., 5., 5., 6., 6., 5., 5., 6., 6., 5., 6., 5.,
       6., 5., 6., 5., 5., 6., 5., 5., 6., 6., 5., 6., 6., 5., 5., 6., 6.,
       6., 5., 6., 6., 6., 5., 5., 5., 6., 5., 5., 6., 5., 6., 5., 6., 6.,
       6., 5., 5., 5., 6., 6., 6., 6., 6., 5., 5., 6., 6., 5., 6., 5., 6.,
       6., 5., 5., 5., 5., 5., 6., 6., 5., 6., 5., 5., 6., 6., 5., 5., 5.,
       5., 5., 5., 5., 5., 6., 5., 5., 5., 5., 5., 5., 5., 6., 6., 6., 6.,
       6., 5., 5., 5., 5., 5., 5., 5., 6., 6., 6., 6., 5., 5., 5., 5., 6.,
       6., 5., 5., 5., 6., 5., 6., 6., 5., 6., 6., 5., 5., 5., 6., 5., 5.,
       5., 5., 6., 6., 6., 5., 6., 6., 6., 6., 6., 5., 5., 5., 5., 6., 6.,
       5., 6., 6., 6., 6., 5., 5., 6., 6., 5., 6., 6., 6., 5., 6., 5., 6.,
       6., 5., 6., 6., 5., 5., 5., 6., 6., 5., 6., 5., 6., 6., 5., 6., 6.,
```

0s completed at 20:01

```
from tensorflow.keras import optimizers, losses, metrics
model = Sequential()
from keras.utils.vis_utils import plot_model
model.add(Dense(12, input_dim=64, activation='sigmoid'))
model.add(Dense(5, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.Adam(learning_rate=0.001),loss='binary_crossentropy')
model.fit(Train_X,Train_y,batch_size=64,epochs=1000,verbose=2)
model.evaluate(Train_X, Train_y)
model.summary()
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

```
Epoch 1/1000
12/12 - 0s - loss: 1.2357
Epoch 2/1000
12/12 - 0s - loss: 0.2882
Epoch 3/1000
12/12 - 0s - loss: -6.1252e-01
Epoch 4/1000
12/12 - 0s - loss: -1.3817e+00
Epoch 5/1000
12/12 - 0s - loss: -1.9045e+00
Epoch 6/1000
12/12 - 0s - loss: -2.2934e+00
Epoch 7/1000
12/12 - 0s - loss: -2.6417e+00
Epoch 8/1000
12/12 - 0s - loss: -2.9803e+00
Epoch 9/1000
12/12 - 0s - loss: -3.3090e+00
Epoch 10/1000
12/12 - 0s - loss: -3.6271e+00
Epoch 11/1000
```

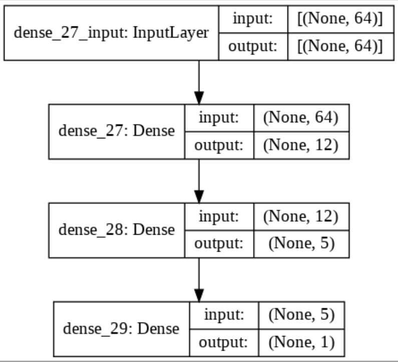
Executing (23s) Cell > fit() > __call__() > _call() > __call__() > _call_flat() > call() > quick_execute()

+ Code + Text

RAM
Disk

Editing

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_27 (Dense)             (None, 12)                780
_________________________________________________________________
dense_28 (Dense)             (None, 5)                 65
_________________________________________________________________
dense_29 (Dense)             (None, 1)                 6
=================================================================
Total params: 851
Trainable params: 851
Non-trainable params: 0
_________________________________________________________________
```



25s completed at 20:03

```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
 #Importing the dataset
Train_X = dataset = loadtxt('TrainX.csv', delimiter=',')
Train_y = dataset = loadtxt('TrainY.csv', delimiter=',')
Test_X = dataset = loadtxt('TestX.csv', delimiter=',')
Test_y = dataset = loadtxt('TestY.csv', delimiter=',')
Train_X.shape
model = Sequential()
model.add(Dense(128, input_dim=64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
from tensorflow.keras import optimizers, losses, metrics
model = Sequential()
from keras.utils.vis_utils import plot_model
model.add(Dense(12, input_dim=64, activation='sigmoid'))
model.add(Dense(4, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.Adam(learning_rate=0.001),loss='binary_crossentropy',metric
s=['accuracy'])
model.fit(Train_X,Train_y,batch_size=64,epochs=10,verbose=2)
model.evaluate(Train_X, Train_y)
model.summary()
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

```
#required libraries
import numpy as np
import math
import matplotlib.pyplot as plt
import matplotlib.colors
import time
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,
mean_squared_error, log_loss
from tqdm import tqdm_notebook

from IPython.display import HTML
import warnings
from sklearn.preprocessing import OneHotEncoder
from sklearn.datasets import make_blobs

import torch
import torch.nn.functional as F
warnings.filterwarnings('ignore')


#converting the numpy array to torch tensors
#Train_y --> (5,6) -- (0,1)
#Test_y --> (5,6) -- (0,1)
for x in range(len(Train_y)):
  if Train_y[x,] == 5:
    Train_y[x,] = 0.0
  else:
    Train_y[x,] = 1.0

for x in range(len(Test_y)):
  if Test_y[x,] == 5:
    Test_y[x,] = 0.0
  else:
    Test_y[x,] = 1.0


X_train = torch.tensor(X_train).float()
Y_train = torch.tensor(Y_train).long()
X_val = torch.tensor(X_val).float()
Y_val = torch.tensor(Y_val).long()
print(X_train.shape, Y_train.shape)


#function for computing forward pass in the network
def model(x):
    A1 = torch.matmul(x, weights1) + bias1 # 753 * 5
```

```
    H1 = A1.sigmoid() # 753 * 5
    A2 = torch.matmul(H1, weights2) + bias2 # (N, 2) x
(2, 4) -> (N, 4)
    H2 = A2.exp()/A2.exp().sum(-1).unsqueeze(-1) # (N, 4)
#applying softmax at output layer.
    return H2


#function to calculate loss of a function.
#y_hat -> predicted & y -> actual
def loss_fn(y_hat, y):
     return -(y_hat[range(y.shape[0]), y].log()).mean()

#function to calculate accuracy of model
def accuracy(y_hat, y):
     pred = torch.argmax(y_hat, dim=1)
     return (pred == y).float().mean()

torch.manual_seed(0)
weights1 = (torch.randn(64, 5) / math.sqrt(2)).float()
weights1.requires_grad_()
bias1 = torch.zeros(5, requires_grad=True)
weights2 = torch.randn(5, 2) / math.sqrt(2)
weights2.requires_grad_()
bias2 = torch.zeros(2, requires_grad=True)

learning_rate = 0.2
epochs = 30
loss_arr = []
acc_arr = []

for epoch in range(epochs):
    y_hat = model(X_train)
    loss = F.cross_entropy(y_hat, Y_train)
    loss.backward()
    loss_arr.append(loss.item())
    acc_arr.append(accuracy(y_hat, Y_train))

    with torch.no_grad():
        weights1 -= weights1.grad * learning_rate
        bias1 -= bias1.grad * learning_rate
        weights2 -= weights2.grad * learning_rate
```

```
bias2 -= bias2.grad * learning_rate
weights1.grad.zero_()
bias1.grad.zero_()
weights2.grad.zero_()
bias2.grad.zero_()
```

```
loss_arr, acc_arr
```

```
([0.863050103187561,
  0.7774642109870911,
  0.6932287216186523,
  0.6296347379684448,
  0.5727447271347046,
  0.525789201259613,
  0.49190765619277954,
  0.4662516415119171,
  0.445613831281662,
  0.4296336770057678,
  0.4181179404258728,
  0.40935662388801575,
  0.40208104252815247,
  0.39547044038772583,
  0.389049232006073,
  0.38291528820991516,
  0.3774395287036896,
  0.37271174788475037,
  0.3687872290611267,
  0.36553463339805603,
  0.36272817850112915,
  0.36021319031715393,
  0.3579065203666687,
  0.355755478143692,
  0.3537191152572632,
  0.3517727553844452,
  0.349911533164978,
  0.3481464385986328,
  0.34649723768234253,
  0.3449666202068329],
```

3NN.ipynb

Show all


```
  0.3481464385986328,
  0.34649723768234253,
  0.3449666202068329],
```


```
 [tensor(0.2736),
  tensor(0.3639),
  tensor(0.5671),
  tensor(0.6321),
  tensor(0.7437),
  tensor(0.8566),
  tensor(0.8938),
  tensor(0.9177),
  tensor(0.9336),
  tensor(0.9548),
  tensor(0.9575),
  tensor(0.9588),
  tensor(0.9602),
  tensor(0.9655),
  tensor(0.9695),
  tensor(0.9734),
  tensor(0.9761),
  tensor(0.9774),
  tensor(0.9774),
  tensor(0.9774),
  tensor(0.9788),
  tensor(0.9801),
  tensor(0.9827),
  tensor(0.9841),
  tensor(0.9854),
  tensor(0.9867),
  tensor(0.9894),
  tensor(0.9987),
  tensor(0.9987),
  tensor(0.9987)])
```