**Artificial Intelligence (CS-323)  
Assignment 01  
  
 Syed Abdul Kabir Qadri   
 CS-22039**

**Question 1:  
  
Code:**import numpy as np

def get\_user\_parameters():

initial\_weights = np.array([float(x) for x in input("Provide the initial weights

// (e.g., 0.5 0.5):").split()])

alpha = float(input("Provide the learning rate: "))

offset = float(input("Provide the bias: "))

num\_samples = int(input("Provide the number of training samples: "))

data\_samples = [ ]

for i in range(num\_samples):

sample = input(f"Input values and expected output for sample {i+1} (e.g., 1 0 1): ").split()

data\_samples.append(([int(sample[0]), int(sample[1])], int(sample[2])))

return initial\_weights, alpha, offset, data\_samples

def train\_perceptron(initial\_weights, alpha, offset, data\_samples):

for iteration in range(100):

for input\_vals, target\_output in data\_samples:

weighted\_sum = np.dot(input\_vals, initial\_weights) + offset

prediction = 1 if weighted\_sum >= 0 else 0

error = target\_output - prediction

initial\_weights += alpha \* error \* np.array(input\_vals)

offset += alpha \* error

return initial\_weights, offset

def test\_perceptron(trained\_weights, offset):

while True:

test\_vals = input("Provide test inputs (e.g., 1 0) or type 'exit' to finish: ")

if test\_vals == 'exit':

break

input\_vector = np.array([int(x) for x in test\_vals.split()])

weighted\_sum = np.dot(input\_vector, trained\_weights) + offset

prediction = 1 if weighted\_sum >= 0 else 0

print(f"Predicted output: {prediction}")

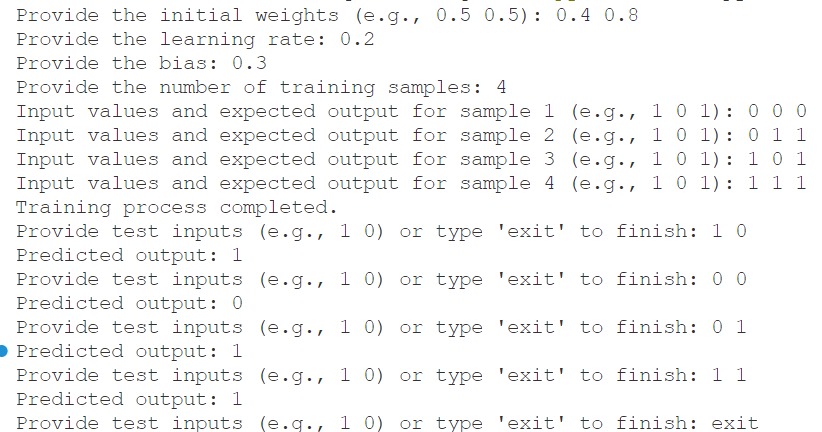
#Execution

initial\_weights, alpha, offset, data\_samples = get\_user\_parameters()

trained\_weights, offset = train\_perceptron(initial\_weights, alpha, offset, data\_samples)

print("Training process completed.")

test\_perceptron(trained\_weights, offset)  
  
  
**OUTPUT:**



**QUESTION 2:**

**Code:**

import numpy as np

def activation\_function(x):

return 1 / (1 + np.exp(-x))

def activation\_derivative(x):

return x \* (1 - x)

def get\_user\_settings():

alpha = float(input("Please input the learning rate: "))

offset = float(input("Please input the bias: "))

num\_epochs = int(input("Please input the number of iterations: "))

print("Specify the number of training samples: ")

num\_samples = int(input())

data\_samples = []

for i in range(num\_samples):

sample = input(f"Provide the input values and target output for sample {i+1}

//(e.g., 1 0 1):").split()

data\_samples.append(([int(sample[0]), int(sample[1])], int(sample[2])))

return alpha, offset, num\_epochs, data\_samples

def init\_weights(input\_dim, hidden\_dim, output\_dim):

weights\_input\_hidden = np.random.rand(input\_dim, hidden\_dim)

weights\_hidden\_output = np.random.rand(hidden\_dim, output\_dim)

return weights\_input\_hidden, weights\_hidden\_output

def train\_neural\_net(weights\_input\_hidden, weights\_hidden\_output, offset, alpha, num\_epochs, data\_samples):

for epoch in range(num\_epochs):

for input\_vals, target\_output in data\_samples:

input\_vals = np.array(input\_vals)

target\_output = np.array(target\_output)

# Forward propagation

hidden\_input = np.dot(input\_vals, weights\_input\_hidden) + offset

hidden\_output = activation\_function(hidden\_input)

final\_input = np.dot(hidden\_output, weights\_hidden\_output) + offset

final\_output = activation\_function(final\_input)

# Calculate error

output\_error = target\_output - final\_output

# Backpropagation

delta\_output = output\_error \* activation\_derivative(final\_output)

hidden\_layer\_error = delta\_output.dot(weights\_hidden\_output.T)

delta\_hidden\_layer = hidden\_layer\_error \* activation\_derivative(hidden\_output)

# Update weights

weights\_hidden\_output += hidden\_output.reshape(-1, 1).dot(delta\_output.reshape(1, -1)) \* alpha

weights\_input\_hidden += input\_vals.reshape(-1, 1).dot(delta\_hidden\_layer.reshape(1, -1)) \* alpha

return weights\_input\_hidden, weights\_hidden\_output

def evaluate\_neural\_net(weights\_input\_hidden, weights\_hidden\_output, offset):

while True:

test\_data = input("Enter test inputs (e.g., 1 0) or 'exit' to stop: ")

if test\_data == 'exit':

break

input\_vals = np.array([int(x) for x in test\_data.split()])

hidden\_input = np.dot(input\_vals, weights\_input\_hidden) + offset

hidden\_output = activation\_function(hidden\_input)

final\_input = np.dot(hidden\_output, weights\_hidden\_output) + offset

final\_output = activation\_function(final\_input)

prediction = 1 if final\_output >= 0.5 else 0

print(f"Predicted output: {prediction}")

# Execution

alpha, offset, num\_epochs, data\_samples = get\_user\_settings()

input\_dim = 2

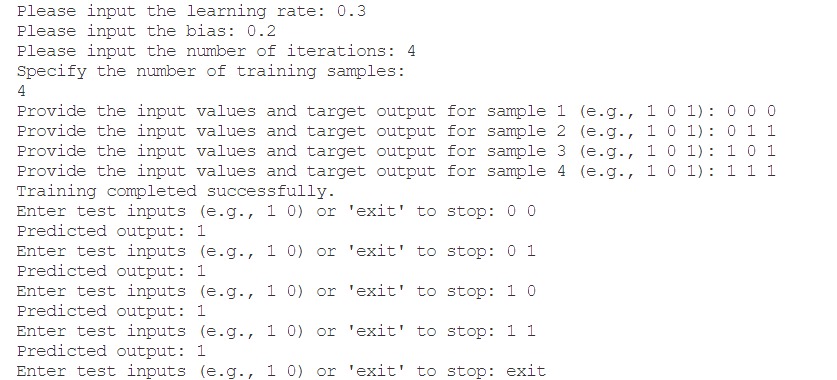
hidden\_dim = 2

output\_dim = 1

weights\_input\_hidden, weights\_hidden\_output = init\_weights(input\_dim, hidden\_dim, output\_dim)

weights\_input\_hidden, weights\_hidden\_output = train\_neural\_net(weights\_input\_hidden, weights\_hidden\_output, offset, alpha, num\_epochs, data\_samples)

print("Training completed successfully.")

evaluate\_neural\_net(weights\_input\_hidden, weights\_hidden\_output, offset)  
  
**OUTPUT:**