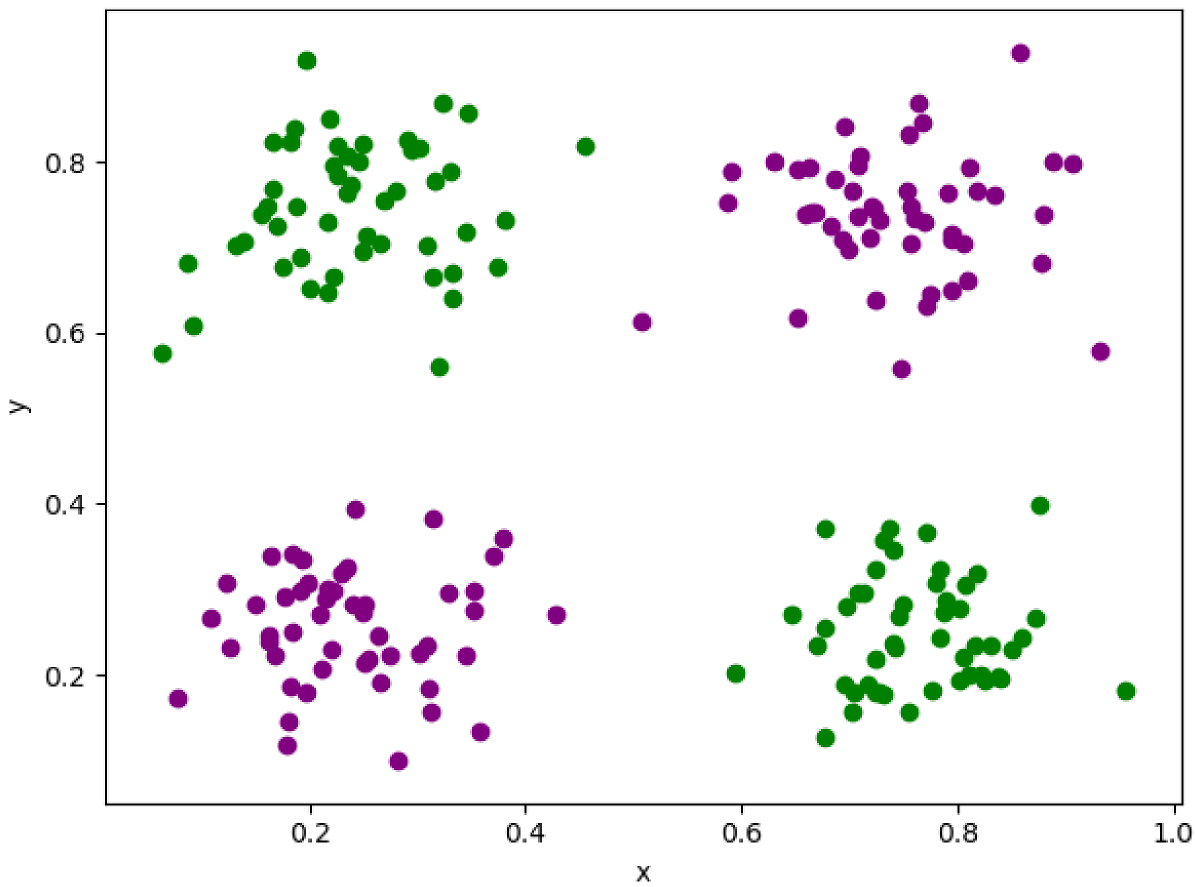
1)

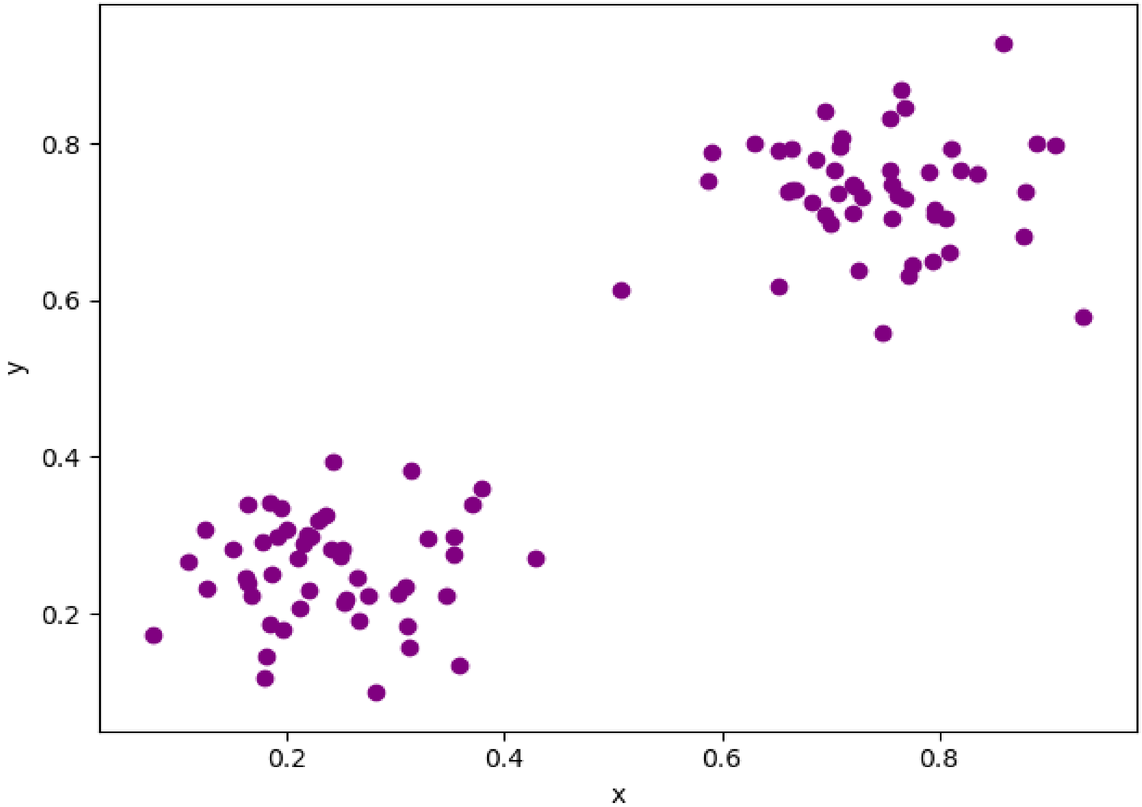
**A.Read and scatter**

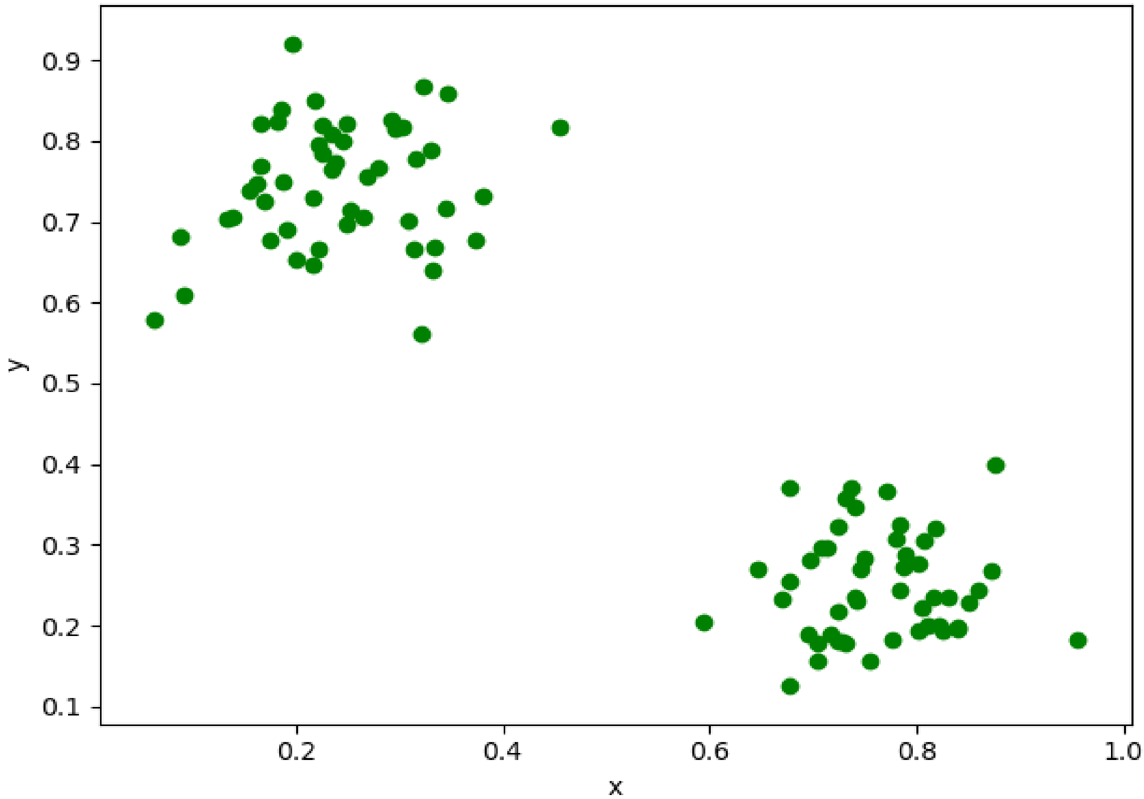
```
data = pd.read_csv('data.csv', header=None)

# print(data)
x = data[0]
y = data[1]
label = data[2]

fig, ax = plt.subplots()
ax.scatter(x, y, c=label)
plt.xlabel('x')
plt.ylabel('y')
```




Purple points: class 1(0)

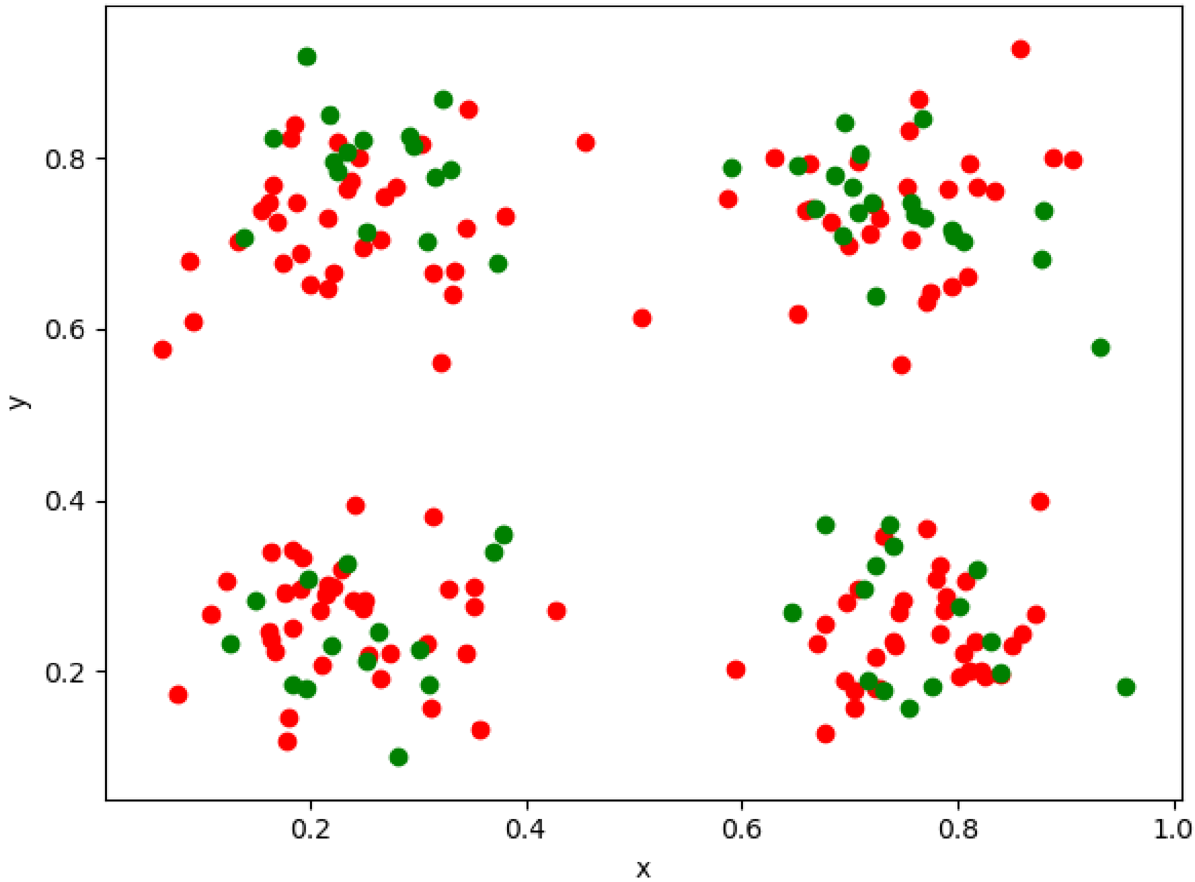
Green points: class 2(1)


0.8
0.6
0.4
0.2
y
0.2
0.4
0.6
0.8
x


0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
y
0.2
0.4
0.6
0.8
1.0
x

**B.Suffle and split**

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42, shuffle=True)
```



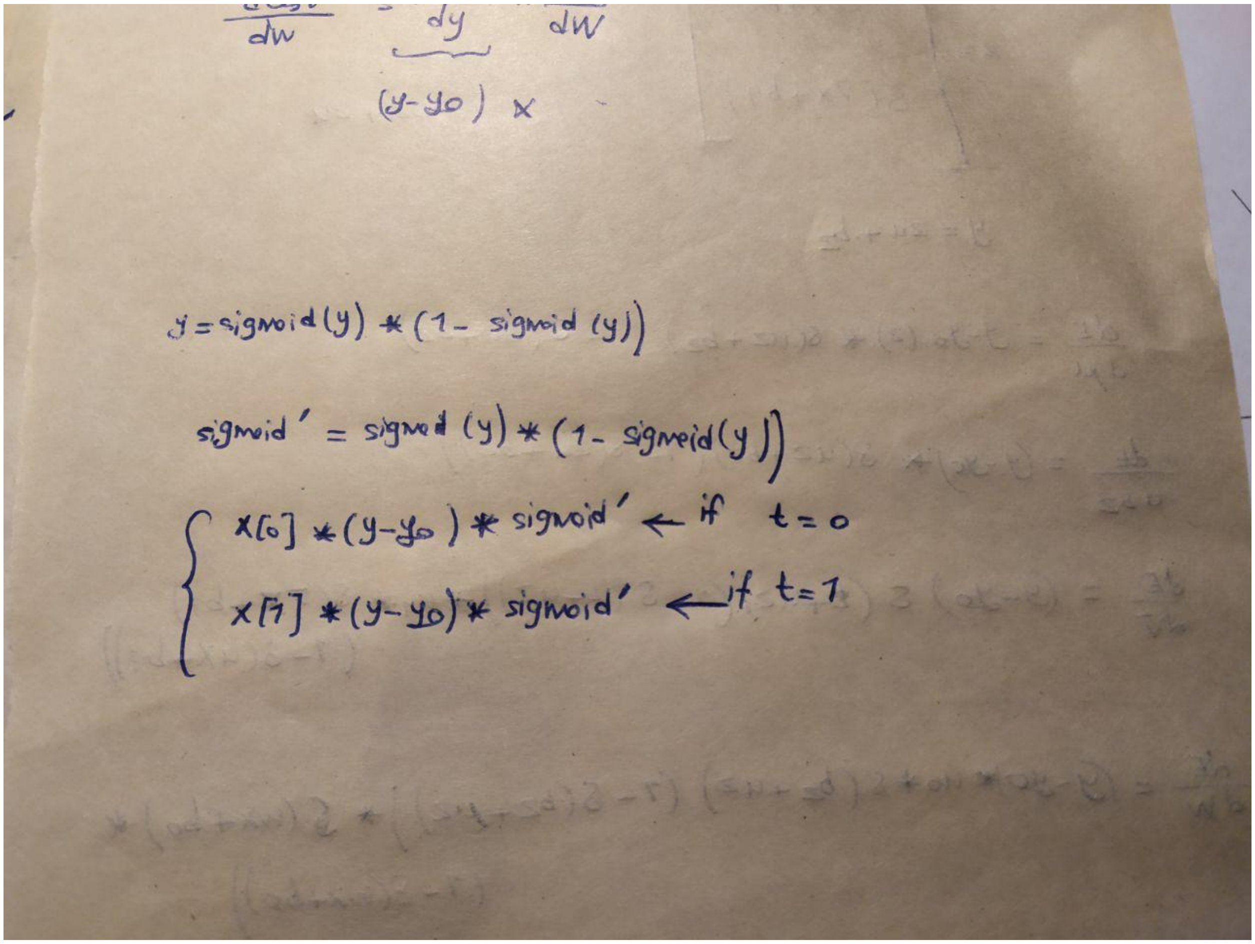

Red: train

Green: test

2)

dcost /dw

dcost/db




3)

```
def sigmoid(x):
    return (1 / (1 + math.exp(-x)))


def compute_gradient(W, X, b, y0, weight):
    y = compute_y(W, X, b)
    sigmoid_new = y * (1 - y)
    if weight == 0:
        return X[0] * (y - y0) * sigmoid_new
    elif weight == 1:
        return X[1] * (y - y0) * sigmoid_new
    else:
        return (y - y0) * sigmoid_new


def compute_y(W, X, b):
    return sigmoid(np.dot(X, W) + b)


def get_labels():
```

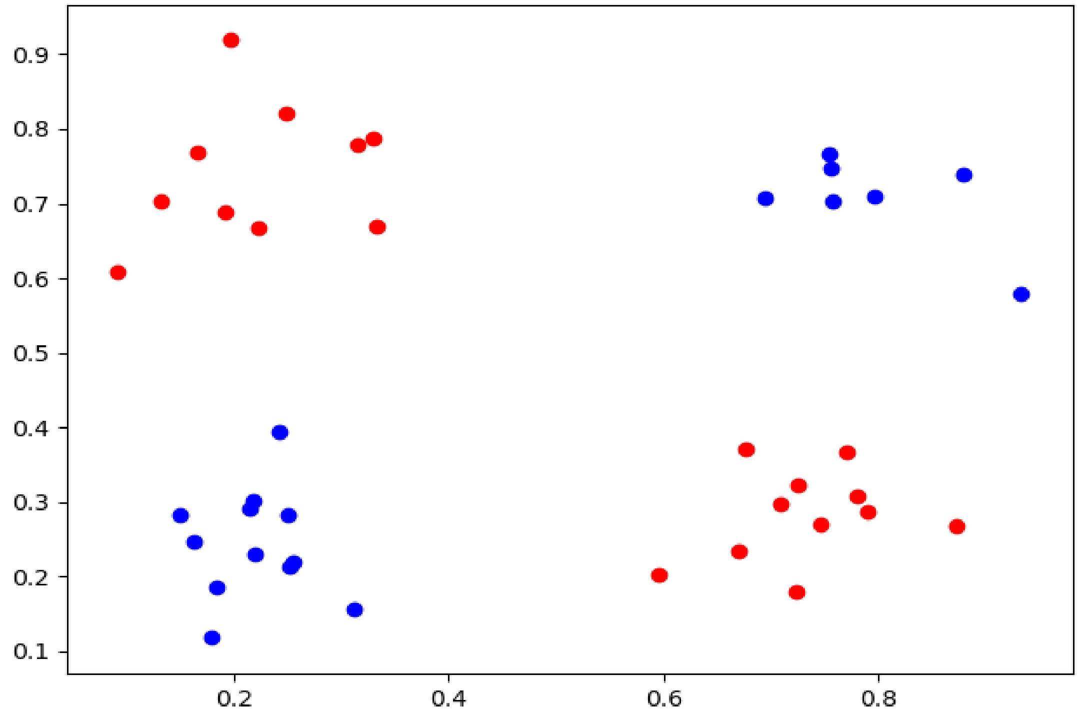
```
    dataset = csv.reader(open("data.csv"))
    dataset_label = []

    x0 = []
    x1 = []
    y0 = []
    y1 = []

    for col in dataset:
        dataset_label.append(
            [[float(col[0].replace('"', "")), float(col[1].replace('"', ""))], int(col[2].replace('"', ""))])
        if int(col[2].replace('"', "")) == 0:
            x0.append(float(col[0].replace('"', "")))
            y0.append(float(col[1].replace('"', "")))
        else:
            x1.append(float(col[0].replace('"', "")))
            y1.append(float(col[1].replace('"', "")))
    return dataset_label


def train(train_label):
    n_epoch = 3000
    lr = 3 / len(train_label)  # learning_rate
    gradient = [0, 0, 0]
    W = []
    b = np.random.normal(0, 1)
    W.append(np.random.normal(0, 1))
    W.append(np.random.normal(0, 1))

    for i in range(0, n_epoch):
        for w in range(0, 3):
            gradient[w] = 0
            for X in train_label:
                gradient[w] += compute_gradient(W, X[0], b, X[1], w)
        for w in range(0, 2):
            W[w] -= lr * gradient[w]
            b -= lr * gradient[2]
    return [W, b]
```

4)
Evaluating the network:

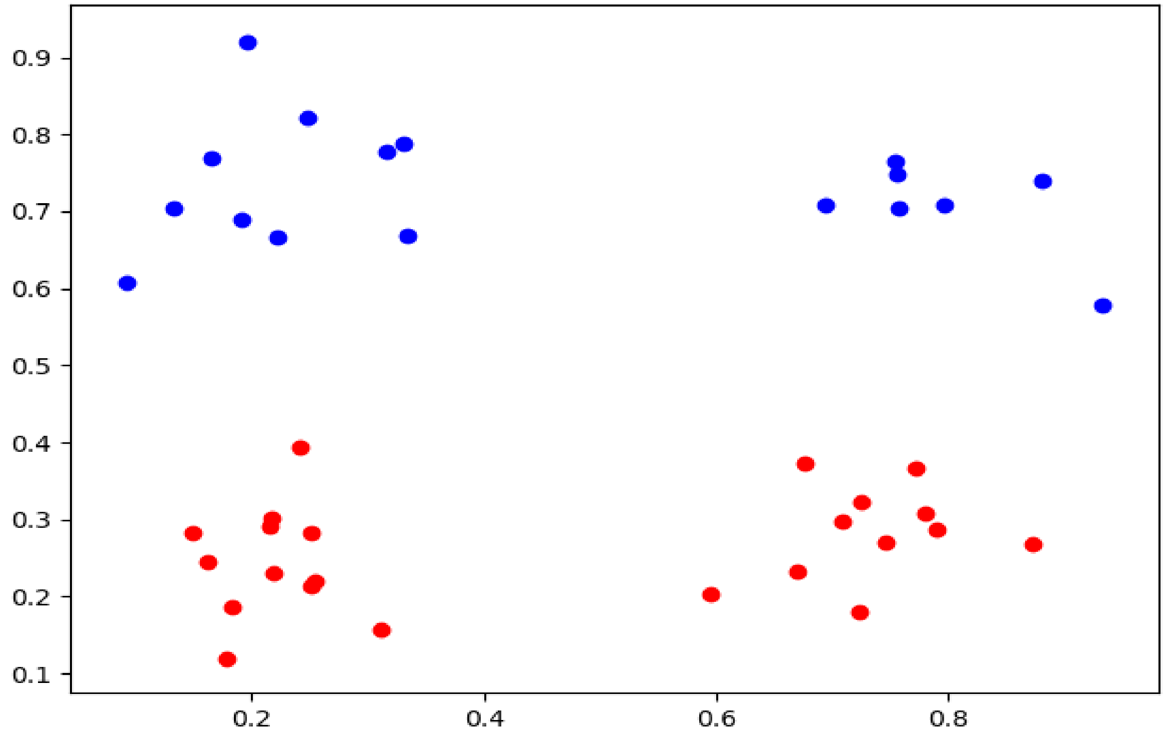
```
def test(W, b, test_label):
    for X in test_label:
        # Y = sigmoid(np.dot(W, X[0]) + b)
        Y = compute_y(W, X[0], b)
        if Y >= 0.5:
            X[1] = 1
        else:
            X[1] = 0
    return test_label


x0 = []
x1 = []
y0 = []
y1 = []

x0_predicted = []
```

```
x1_predicted = []
y0_predicted = []
y1_predicted = []

dataset = get_labels()
np.random.shuffle(dataset)

train_value = []
test_value = []

# plot test data
for i in range(0, len(dataset)):
    # split
    if i < np.round(0.8 * len(dataset)):
        train_value.append([[dataset[i][0][0], dataset[i][0][1]], dataset[i][1]])
    else:
        test_value.append([[dataset[i][0][0], dataset[i][0][1]], dataset[i][1]])

for i in range(0, len(test_value)):
    if test_value[i][1] == 0:
        x0.append(test_value[i][0][0])
        y0.append(test_value[i][0][1])
    else:
        x1.append(test_value[i][0][0])
        y1.append(test_value[i][0][1])

plt.scatter(x0, y0, color="blue")
plt.scatter(x1, y1, color="red")
plt.show()
# print(len(test_value))
# print(len(dataset))

W, b = train(train_value)
predicted_label_test = test(W, b, test_value)

for i in range(0, len(predicted_label_test)):
    if predicted_label_test[i][1] == 0:
        x0_predicted.append(predicted_label_test[i][0][0])
        y0_predicted.append(predicted_label_test[i][0][1])
    else:
        x1_predicted.append(predicted_label_test[i][0][0])
        y1_predicted.append(predicted_label_test[i][0][1])

plt.scatter(x0_predicted, y0_predicted, color="blue")
plt.scatter(x1_predicted, y1_predicted, color="red")
plt.show()

correct_pred = 0

for i in range(0, len(predicted_label_test)):
    if predicted_label_test[i][1] == test_value[i][1]:
        correct_pred += 1

accuracy = correct_pred / len(test_value)
print(accuracy)
```

Actual test data is:




The predicted Labels are:




Accuracy: True prediction / Total Prediction = 18/40 = 45%

New Design of network:

$$z = \begin{bmatrix} S(wx+b_0) \\ S(vx+b_1) \end{bmatrix} \qquad b_2 + zu$$

$$y = zu + b_2$$

$$\frac{dE}{d\mu} = y - y_0\,(z) * S(uz+b_2)\left(1 - S(\mu z + b_2)\right)$$

$$\frac{dE}{db_2} = (y - y_0) * S(uz+b_2)\left(1 - S(\mu z + b_2)\right)$$

$$\frac{dE}{dv} = (y - y_0)\, S(b+uz)\left(1 - S(b+uz)\right) * u_1 * S(v\cancel{x}+b_1)\left(1 - S(ux+b_1)\right)$$

$$\frac{dE}{dw} = (y - y_0) * u_0 * S(b_2+uz)\left(1 - S(b_2+\mu z)\right) * S(wx+b_0) * \left(1 - S(wx+b_0)\right)$$

$$\frac{dE}{db_0} = (y - y_0) * u_0\, S(b_2+uz)\left(1 - S(b_2+uz)\right) S(wx+b_0) * \left(1 - S(wx+b_0)\right)$$

$$\frac{dE}{db_1} = (y - y_0) * u_1 * S(b_2+uz)\left(1 - S(b_2+uz)\right) * S(vx+b_1) * \left(1 - S(ux+b_1)\right)$$

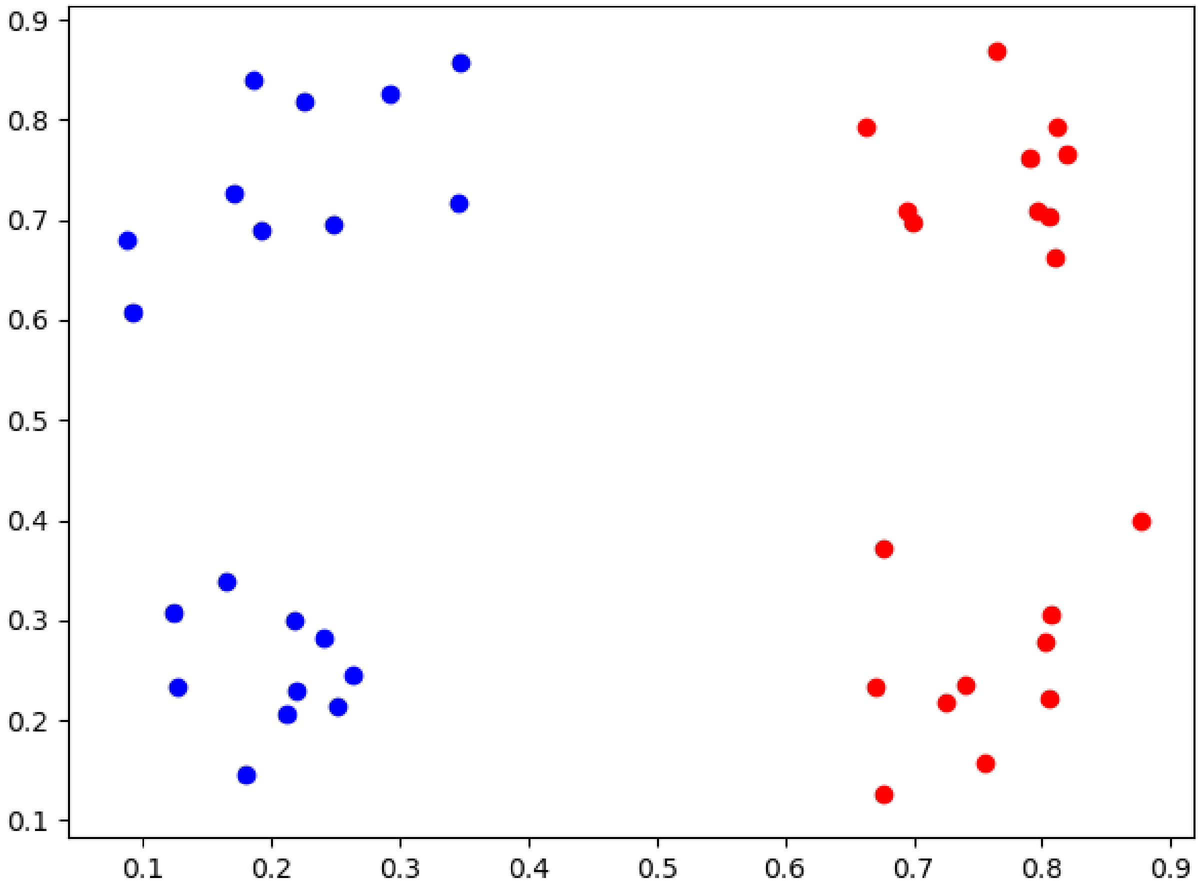
$$\frac{dE}{db_2} = (y - y_0) * u_2 * S(b_2+\mu z)\left(1 - S(b_2+uz)\right) * S(vx+b_2) * \left(1 - S(ux+b_2)\right)$$

5)

This new design finds the weights better and has better accuracy.

But Why?

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. A MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.




Accuracy: 36/40 = 90%

One perceptron is not able to classify a non-linearly separable data, unlike Multilayer peceptron which has a very high accuracy for this kind of data.