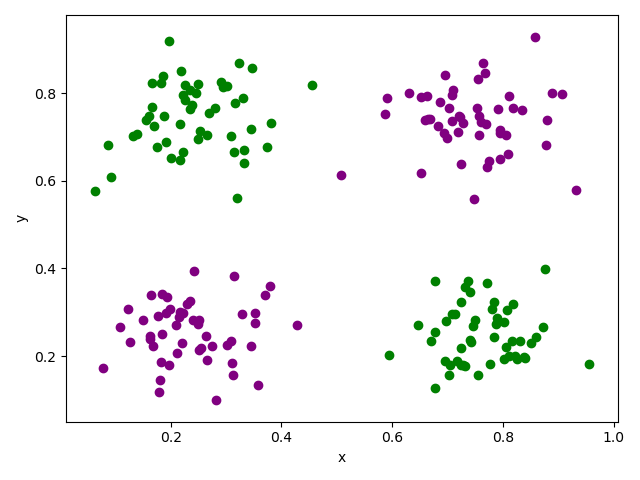
1)

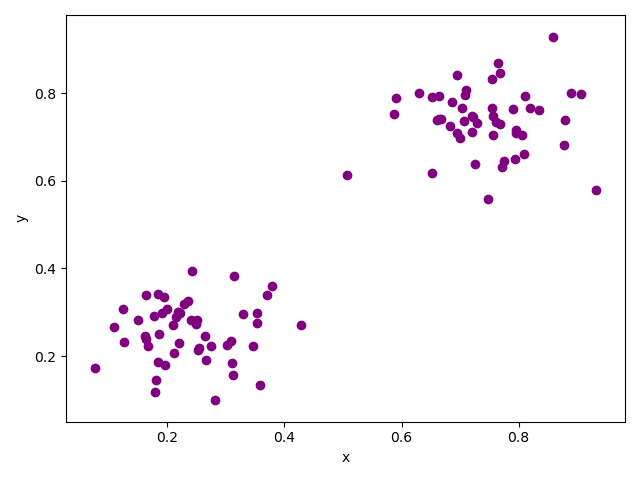
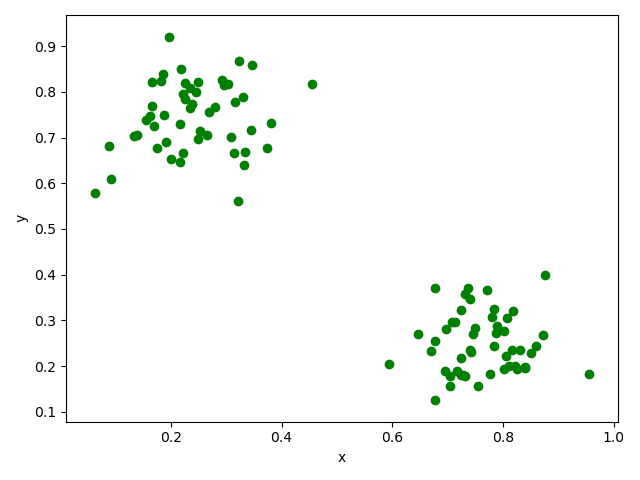
**A.Read and scatter**

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')



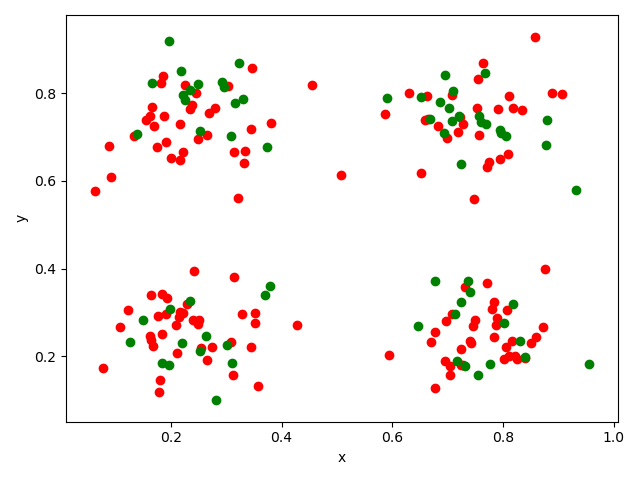
Purple points: class 1(0)

Green points: class 2(1)

**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)



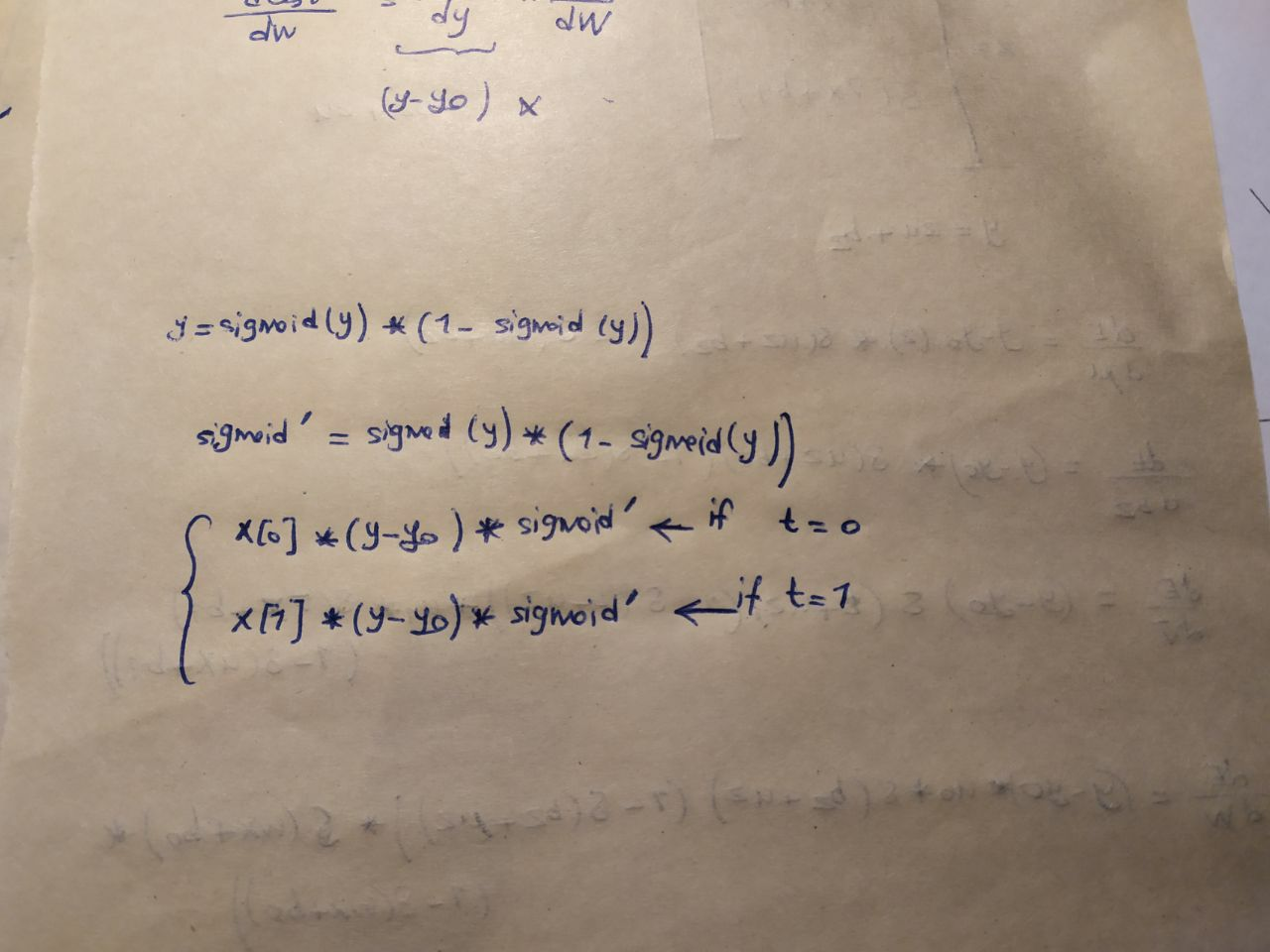
Red: train

Green: test

2)

dcost /dw

dcost/db



3)

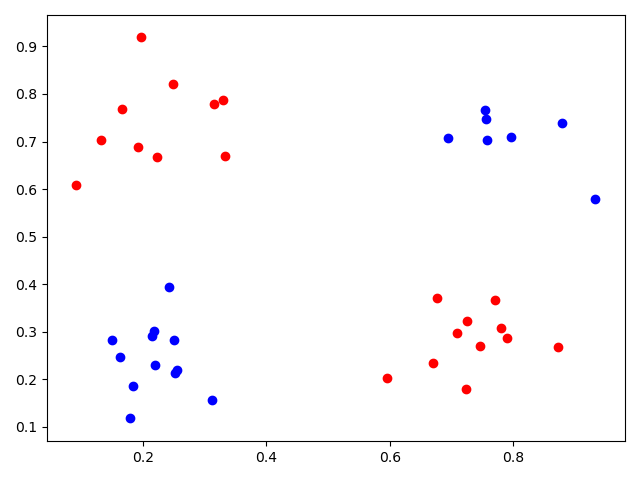
def sigmoid(x):  
 return (1 / (1 + math.exp(-x)))  
  
  
def compute\_gradient(W, X, b, y0, weight):  
 y = compute\_y(W, X, b)  
 sigmoid\_new = y \* (1 - y)  
 if weight == 0:  
 return X[0] \* (y - y0) \* sigmoid\_new  
 elif weight == 1:  
 return X[1] \* (y - y0) \* sigmoid\_new  
 else:  
 return (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  
 Ir = 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] -= Ir \* gradient[w]  
 b -= Ir \* gradient[2]  
 return [W, b]

4)

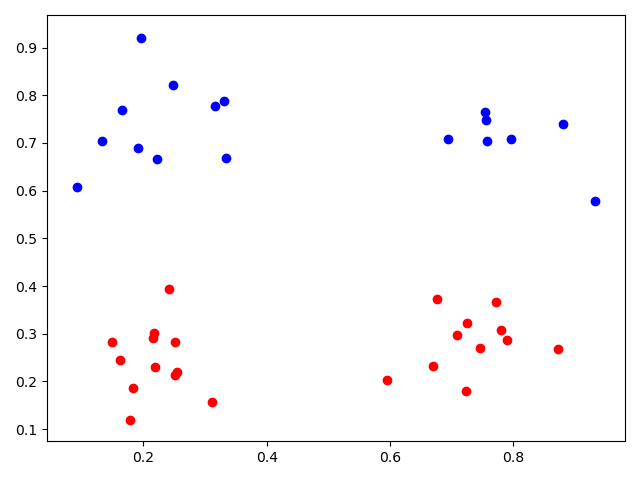
Evaluating the network:

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:

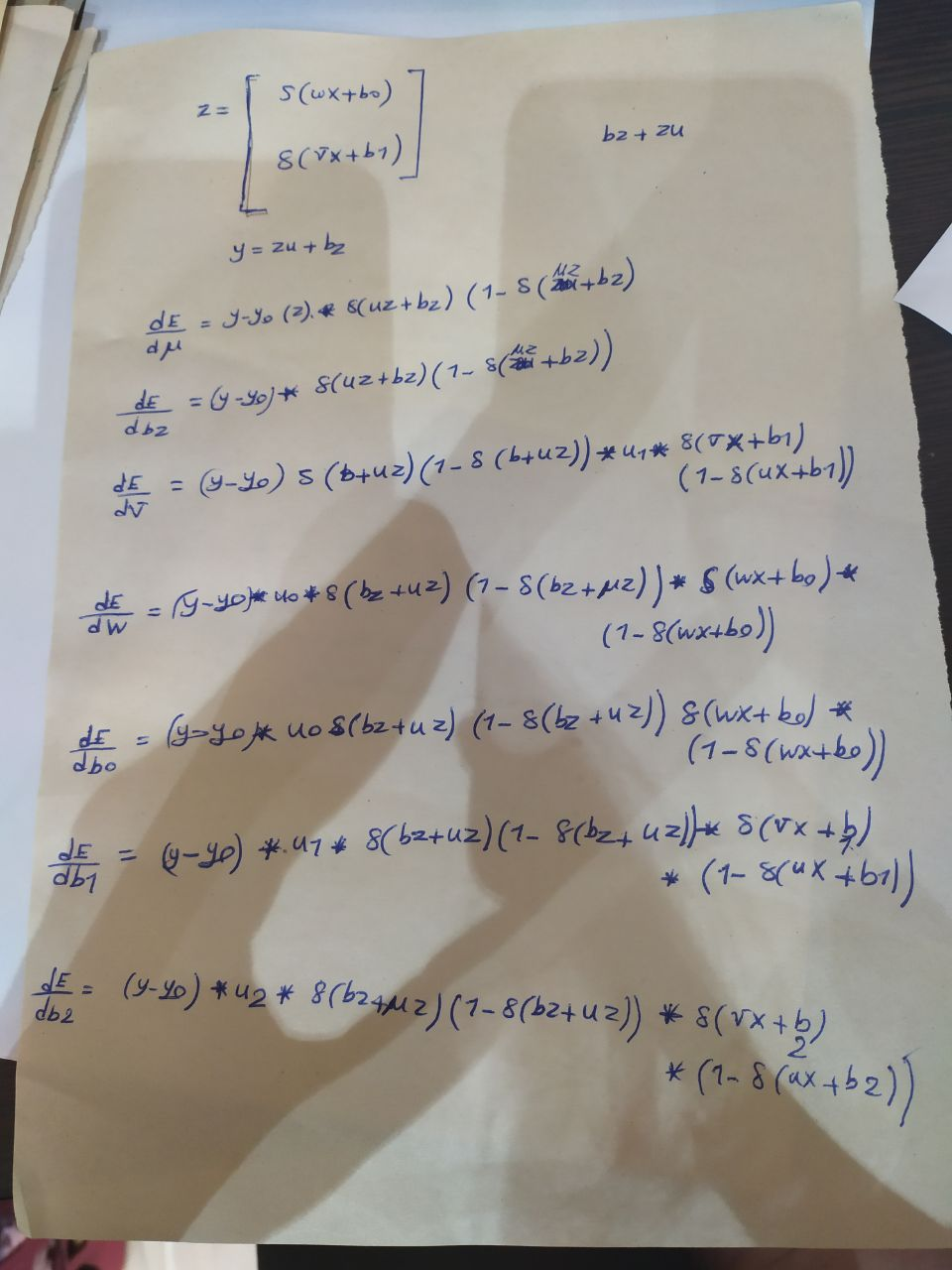


The predicted Labels are:



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

New Design of network:

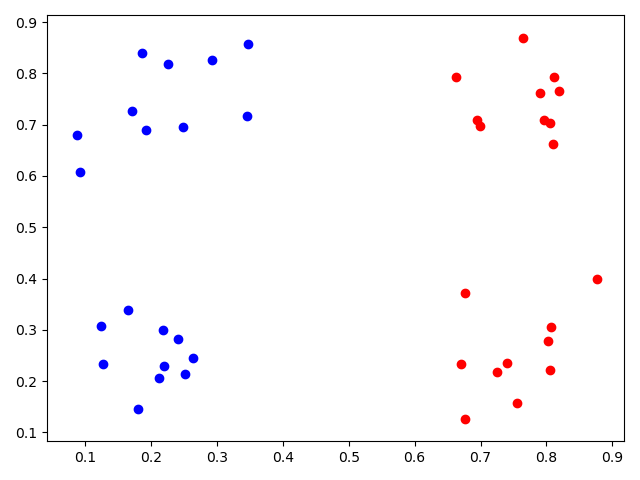


5)

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.