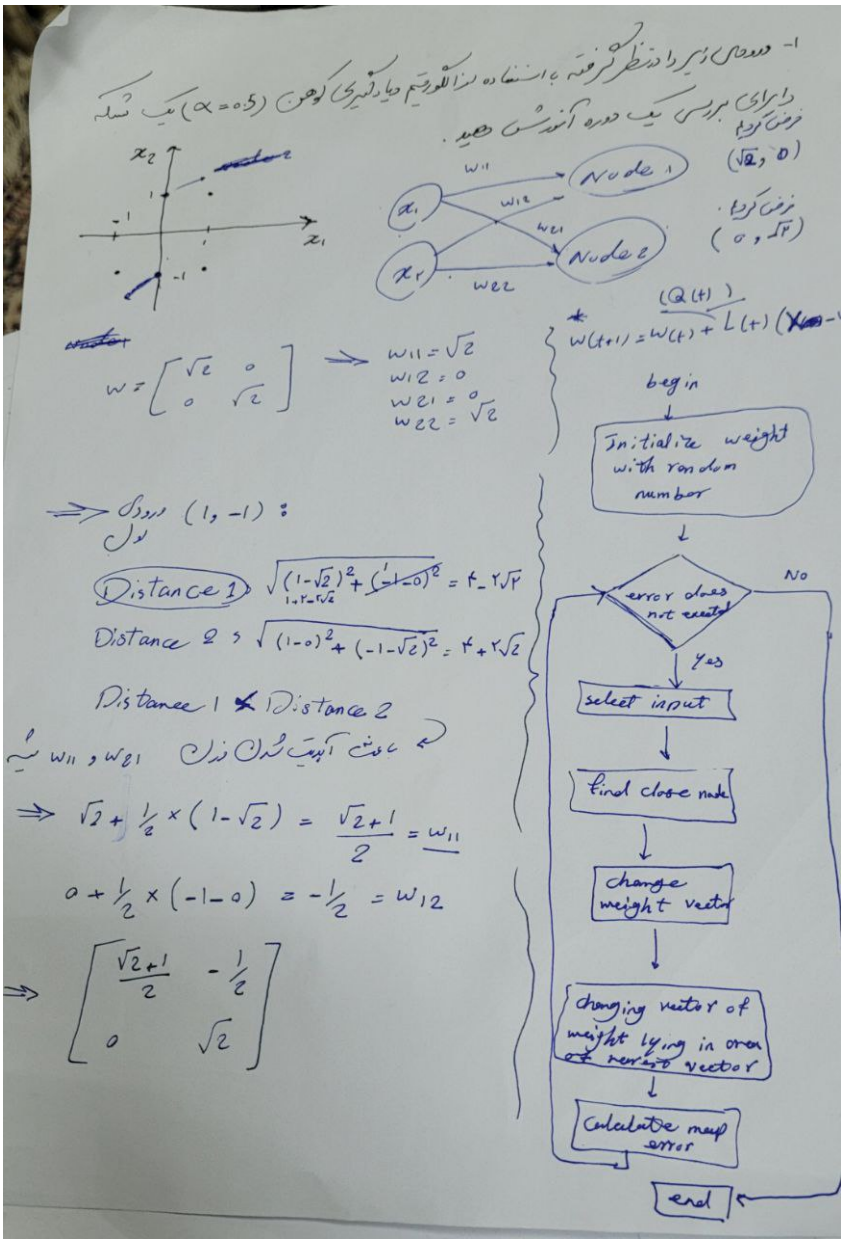
تمرین شماره ۳:

مهدی فقهی

4017221360

سوال اول :



$\Rightarrow$ نزدیک‌ترین نقطه به $(1,1)$

$$\begin{bmatrix} \frac{\sqrt{2}+1}{2} & -\frac{1}{2} \\ 0 & \sqrt{2} \end{bmatrix}$$

Distance 1: $\sqrt{(1-\frac{\sqrt{2}+1}{2})^2+(1+\frac{1}{2})^2} = \sqrt{3-\frac{\sqrt{2}}{2}}$

Distance 2: $\sqrt{(0-1)^2+(\sqrt{2}-\sqrt{2})^2} = \sqrt{4-2\sqrt{2}}$ ✓

Distance 2 < Distance 1

$\Rightarrow$ $0+\frac{1}{2}\times(1-0) = \frac{1}{2} = w_{21}$

$\sqrt{2}+\frac{1}{2}(\sqrt{2}-\sqrt{2}) = \sqrt{2}$ w_{22}

$\Rightarrow \begin{bmatrix} \frac{\sqrt{2}+1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \sqrt{2} \end{bmatrix}$

نزدیک‌ترین نقطه به $(-1,-1)$

Distance 1: $\sqrt{(-1-\frac{\sqrt{2}+1}{2})^2+(-1+\frac{1}{2})^2} = \sqrt{\frac{4+2\sqrt{2}}{4}}$

Distance 2: $\sqrt{(\frac{1}{2}+1)^2+(\sqrt{2}+1)^2} = \sqrt{\frac{21+8\sqrt{2}}{4}}$

Distance 1 < Distance 2

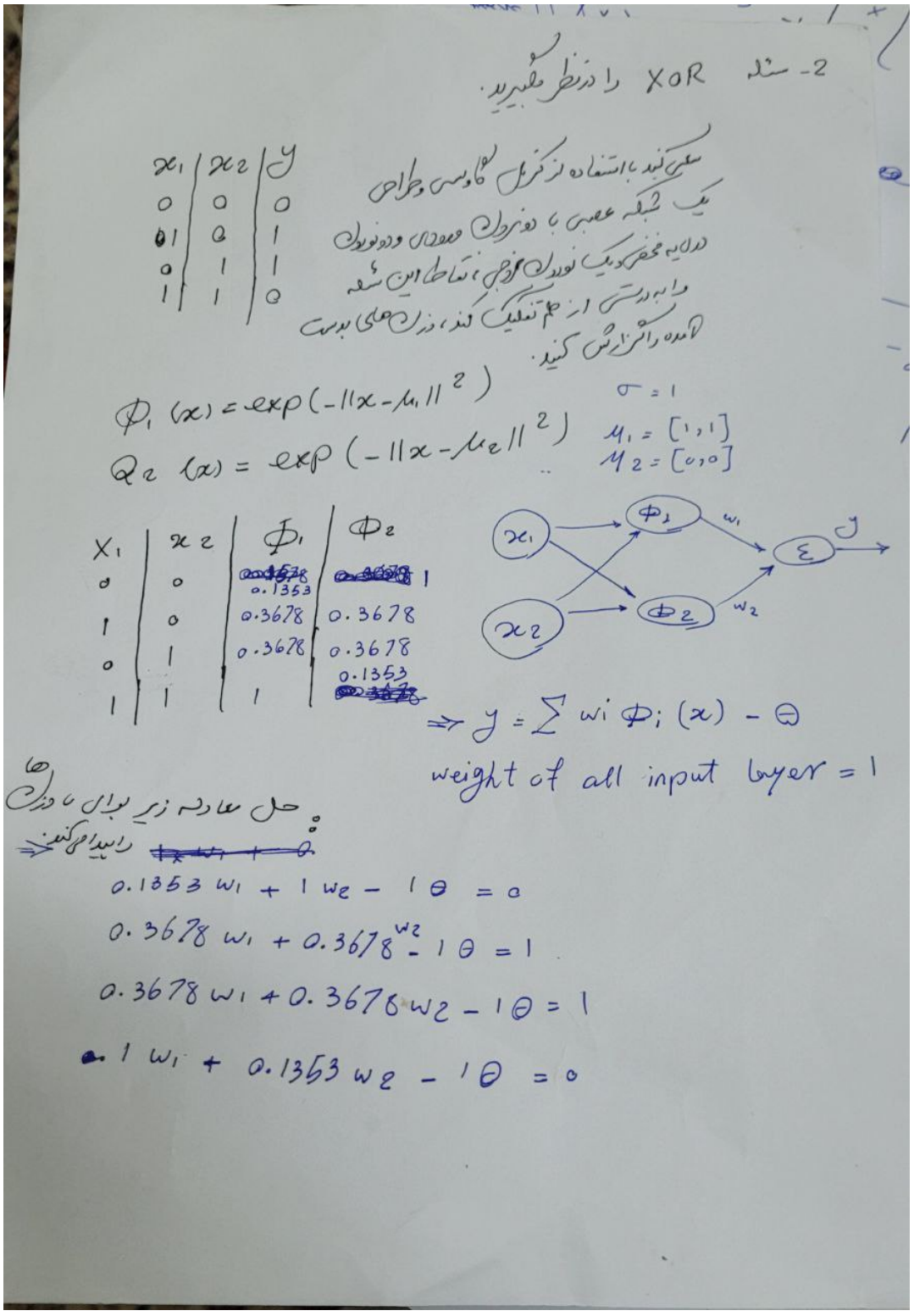
$\Rightarrow$ $\frac{\sqrt{2}+1}{2}+\frac{1}{2}(\frac{-1-\sqrt{2}}{2}-1-\frac{\sqrt{2}+1}{2}) = \frac{2\sqrt{2}+2-1-\sqrt{2}}{4} = \frac{1+\sqrt{2}}{4} = w_{11}$

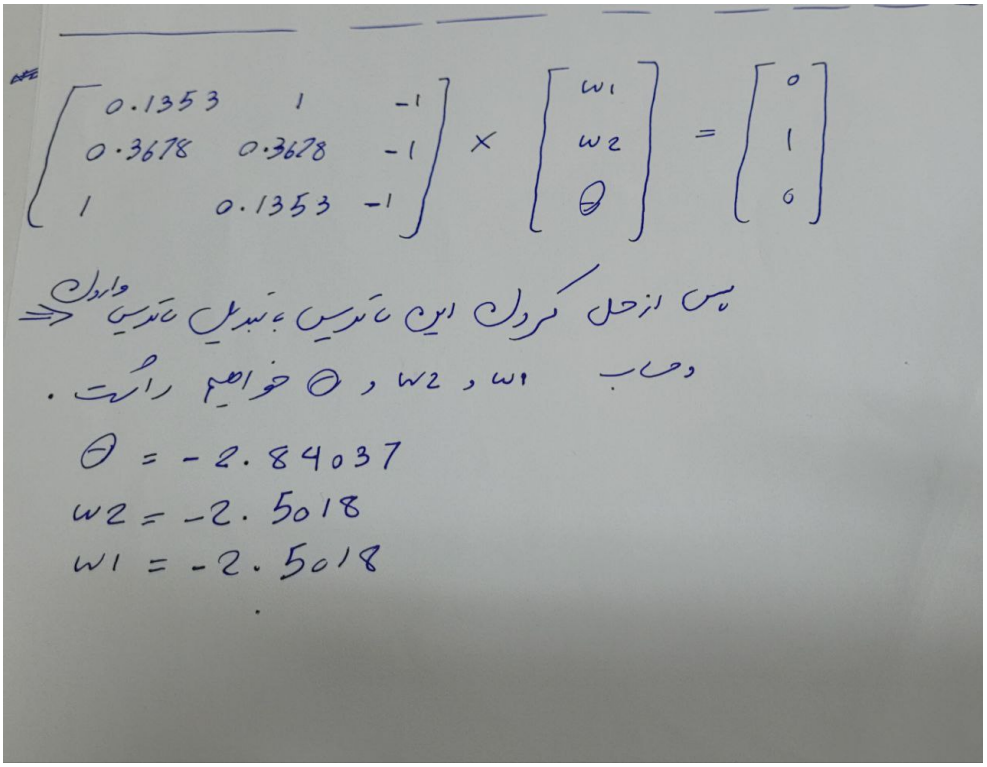
$-\frac{1}{2}+\frac{1}{2}(-1+\frac{1}{2}) = -\frac{1}{2}-\frac{1}{4} = -\frac{3}{4}$

$\Rightarrow \begin{bmatrix} \frac{1+\sqrt{2}}{4} & -\frac{3}{4} \\ \frac{1}{2} & \sqrt{2} \end{bmatrix}$ $\Rightarrow$ ذیل اپدیت شده بعد از سه مرحله

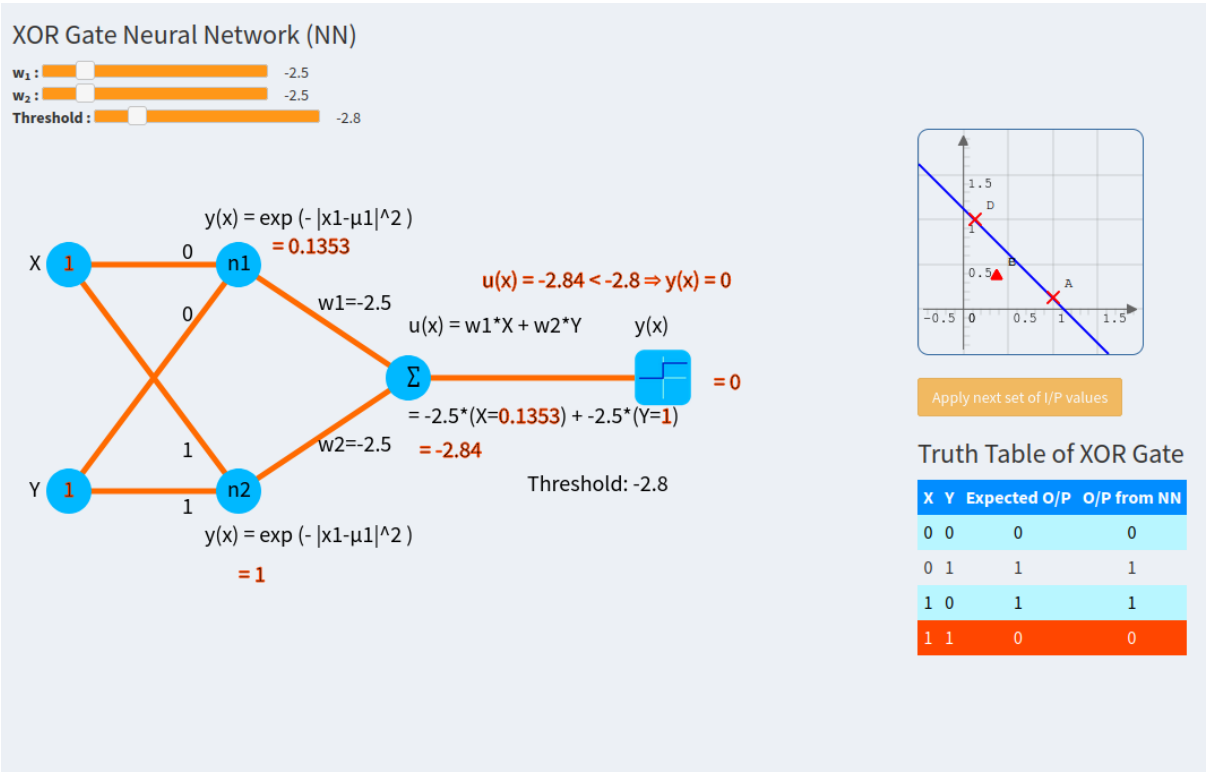
سوال دوم :

قسمت الف :




0.1353 1 -1
0.3678 0.3678 -1
1 0.1353 -1
w1
w2
θ
0
1
0
پس از حل کردن این ماتریس و تبدیل ماتریس داریم
و در نهایت w1 و w2 و θ خواهیم داشت.
θ = -2.84037
w2 = -2.5018
w1 = -2.5018

قسمت ب :


XOR Gate Neural Network (NN)
w1 : -2.5
w2 : -2.5
Threshold : -2.8
y(x) = exp (- |x1-μ1|^2)
= 0.1353
X
1
0
n1
0
w1=-2.5
u(x) = -2.84 < -2.8 ⇒ y(x) = 0
u(x) = w1*X + w2*Y
y(x)
Σ
= 0
= -2.5*(X=0.1353) + -2.5*(Y=1)
= -2.84
1
w2=-2.5
Threshold: -2.8
Y
1
n2
1
y(x) = exp (- |x1-μ1|^2)
= 1
1.5
D
1
0.5
B
A
-0.5
0
0.5
1
1.5
Apply next set of I/P values
Truth Table of XOR Gate
X Y Expected O/P O/P from NN
0 0 0 0
0 1 1 1
1 0 1 1
1 1 0 0

همانطور که می‌بینیم به کمک تابع خطی زیر می‌توانیم توی feature space جدید بدست آمده با دقت ۱۰۰ درصد حاصل را به درستی بدست آوریم .

$\varphi 1 * (-2.5) + \varphi 2 * (-2.5) + 2.8 = Y$

به گونه‌ای که اگر حاصل بزرگتر مساوی از صفر باشد یک و اگر کوچکتر از صفر باشد 0 در نظر می گیریم

قسمت ج :

```python
def gaussian_rbf(x, landmark, gamma=1):
    return np.exp(-gamma * np.linalg.norm(x - landmark)**2)
```

که در واقع حاصل تابع زیر را برمی گرداند.

$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$$

که در این تابع مقدار Default متغیر gamma برابر با یک در نظر گرفته شده است .

و حاصل نرم به توان دو بدست می‌آید .

به کمک np.linalg.norm فرم دوم تفاضل دو متغیر x و landmark را بدست می‌آوریم .

The following norms can be calculated:

ord	norm for matrices	norm for vectors
None	Frobenius norm	2-norm

$$||A||_F = [\sum_{i,j} abs(a_{i,j})^2]^{1/2}$$

در قسمت اصلی بدنه برنامه برای پیاده سازی xor دو بردار ویژگی X1, X2 که ورودی عملگر ما هستند را مقدار دهی کردیم و همچنین در ys خروجی هر دو ورودی خود را مشخص کردیم .

```
[5]  # points
     x1 = np.array([0, 0, 1, 1])
     x2 = np.array([0, 1, 0, 1])
     ys = np.array([0, 1, 1, 0])
```

در قسمت بعد برای هر کدام از node موجود یک مرکزیت تعریف کردیم و از آنجا که حاصل ما دو کلاس هست، دو مرکزیت (0,1) , (1,0) را مشخص می‌کنیم تا حول آنان مقدار gussain_rbf خود را بدست آوریم و در نهایت به کمک تابع end_to_end مقدار وزن‌های نهایی را پیدا کنیم .

```
# centers
mu1 = np.array([0, 1])
mu2 = np.array([1, 0])

w = end_to_end(x1, x2, ys, mu1, mu2)
```

در تابع end_to_end ابتدا مقدار gussain_rbf را به ازای هر ورودی از هر کدام یک از نقطه‌های mu2 , mu1 پیدا می‌کنیم .

در قدم بعد به کمک کتابخانه matplotlib.pyplot یک نمودار از وضعیت اولیه نقطه‌ها در یک فضای دو بعدی براساس feature x1, x2 بوجود می‌آوریم و به کمک رنگ نارنجی و آبی label هر نقطه را براساس همان دو کلاس تقسیم بندی مشخص می‌کنیم سپس یک نمودار دیگه نیز بوجود می‌آوریم که به کمک حاصل جدیدی به ازای خروجی تابع gussain_rbf بدست آورده‌ایم در دو بعد جدید from_1 و from_2 و به کمک label نشان خواهیم داد.

```
from_1 = [gaussian_rbf(i, mu1) for i in zip(X1, X2)]
from_2 = [gaussian_rbf(i, mu2) for i in zip(X1, X2)]
# plot

plt.figure(figsize=(13, 5))
plt.subplot(1, 2, 1)
plt.scatter((x1[0], x1[3]), (x2[0], x2[3]), label="Class_0")
plt.scatter((x1[1], x1[2]), (x2[1], x2[2]), label="Class_1")
plt.xlabel("$X1$", fontsize=15)
plt.ylabel("$X2$", fontsize=15)
plt.title("Xor: Linearly Inseparable", fontsize=15)
plt.legend()

plt.subplot(1, 2, 2)
plt.scatter(from_1[0], from_2[0], label="Class_0")
plt.scatter(from_1[1], from_2[1], label="Class_1")
plt.scatter(from_1[2], from_2[2], label="Class_1")
plt.scatter(from_1[3], from_2[3], label="Class_0")
plt.plot([0, 0.95], [0.95, 0], "k--")
plt.annotate("Seperating hyperplane", xy=(0.4, 0.55), xytext=(0.55, 0.66),
             arrowprops=dict(facecolor='black', shrink=0.05))
plt.xlabel(f"$mu1$: {(mu1)}", fontsize=15)
plt.ylabel(f"$mu2$: {(mu2)}", fontsize=15)
plt.title("Transformed Inputs: Linearly Seperable", fontsize=15)
plt.legend()
```

در مرحله بعدی باید مقدار وزن خروجی از لایه میانی به سمت لایه خارجی را حساب کنیم .

برای حل این قضیه به جای استفاده از روش کاهشی گرادیان برای وزن‌ها برای پیدا کردن بهترین وزن از رابطه زیر استفاده می‌کنیم.

$$\vec{\phi}.\vec{W} = \vec{d} \qquad \Rightarrow W = \phi^{-1} d$$

که از حاصل ضرب وارون ماتریس ادغامی form2 , form1 در حاصل بدست می‌آید .((همچنین ابتدا برای اینکه برای این یک بایاس در نظر بگیریم یک ستون تمام یک به ماتریس اضافه کنیم که با این ستون تماما یک ماتریس ما ماتریس 3*4 حاصل می‌شود که همانطور که می‌دانیم همچین ماتریسی به دلیل اینکه یک غیر مربعی است قابلیت وارون شدن ندارد به همین دلیل وارون ماتریس فوق را در ماتریس مذکور ضرب می‌کنیم تا یک ماتریس 3 * 3 بدست آوریم سپس وارون ماتریس را حساب کرده و سپس در وارون ماتریس A ضرب می‌کنیم و یک ماتریس 4*3 بدست می‌آید))

که درنهایت با ضرب در بردار حاصل که یک بردار ۱ * 4 است بردار وزن که برابر با یک بردار 3 * 1 بدست می‌آید که ستون سوم مشخص کننده ضریب biasاست .

```
A = []

for i, j in zip(from_1, from_2):
    temp = []
    temp.append(i)
    temp.append(j)
    temp.append(1)
    A.append(temp)

A = np.array(A)
print(A.T.dot(A))
print(np.linalg.inv(A.T.dot(A)))
print(np.linalg.inv(A.T.dot(A)).dot(A.T))
W = np.linalg.inv(A.T.dot(A)).dot(A.T).dot(ys)
print(np.round(A.dot(W)))
print(ys)
print(f"Weights: {W}")
return W
```

حال که وزن‌ها و bias را داریم به کمک تابع predict_matrix توانایی پیش‌بینی اینکه هر تابع متعلق به چه دسته‌ای است را داریم به کمک همان فرمول :

$$\vec{\phi}.\vec{W} = \vec{d} \qquad \Rightarrow W = \phi^{-1} d$$

سوال ۳)

در ابتدای کار فایل text به نام data_banknote_authentication را می‌خوانیم و ستون ۱ تا ۴ را به عنوان فیچر در data_x قرار می‌دهیم و و ستون پنجم را به عنوان label در data_y قرار می‌دهیم و از با مشاهده داده‌های ستون چهارم متوجه می‌شویم که یک classification باینری داریم چون داده‌های این ستون فقط یک و صفر است .

سپس ۲۰ درصد داده‌ها را برای test از داده‌ها جدا می‌کنیم و از بقیه به عنوان تست استفاده می‌نماییم و ابعاد مسئله را مشاهده می‌نماییم.

```
# banknote authentication Data Set
data_file = "data_banknote_authentication.txt"
data_x = np.loadtxt(data_file, delimiter=",", skiprows=0, usecols=range(0,4) ,dtype=np.float64)
data_y = np.loadtxt(data_file, delimiter=",", skiprows=0, usecols=(4,),dtype=np.int64)
```

```
[4] # train and test split
    train_x, test_x, train_y, test_y = train_test_split(data_x, data_y, test_size=0.2, random_state=42)
    print(train_x.shape, train_y.shape, test_x.shape, test_y.shape) # check the shapes

    (1097, 4) (1097,) (275, 4) (275,)
```

در قدم بعد فیچرهای train را به کمک تابع minmax_scaler نرمال می‌کنیم و تمامی اعداد بین مقیاس صفر تا ۱ قرار میگیرند .

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

در قدم بعد یک ماتریس که شامل ۱۰ عدد ماتریس ده در چهار است با عددهای رندوم بین صفر تا یک می‌سازیم به نام som در اینجا som همان شبکه‌ای از نرون‌های ما است . در واقع گویا ما 100 تا نرون داریم که به هر نرون چهار تا ورودی وارد می‌شود .

```
som = np.random.random_sample(size=(num_rows, num_cols, num_dims)) # map construction
som
```

به تعداد 7500 بار رویدادهای زیر را انجام می‌دهیم :

۱. به کمک تابع decay آیتم‌های learning rate, neighbourhood range را آپدیت می‌نماییم .
با فرض در نظر گرفتن max learning rate به مقدار ۱/2 و max distance به مقدار 4 تابع decay در هر بار اجرا مقدار را به شرح زیر آپدیت می‌کند :

۱.مقداری را به عنوان coefficient در نظر می‌گیریم برابر است با یک منهای تعداد بارهای که حلقه اجرا شده به تعداد کل یعنی 7500 بار .

۲. مقدار learning rate برابر خواهد شد با coefficient*max learning rate

۳. مقدار neighbour range برابر خواهد شد با جز صحیح coefficient * maxman distance

```
# Learning rate and neighbourhood range calculation
def decay(step, max_steps,max_learning_rate,max_m_dsitance):
  coefficient = 1.0 - (np.float64(step)/max_steps)
  learning_rate = coefficient*max_learning_rate
  neighbourhood_range = ceil(coefficient * max_m_dsitance)
  return learning_rate, neighbourhood_range
```

۲. در قدم بعدی نرون برنده را پیدا می‌کنیم :
برای انجام این کار یکی یکی نرون‌های موجود در som را بررسی می‌کنیم و فاصله اقلیدسی آنان را تا یکی از داده‌ها به که به صورت شانسی انتخاب شده است پیدا کنیم و آن که کمترین فاصله را دارد به عنوان نرون برنده انتخاب کنیم .

```
def winning_neuron(data, t, som, num_rows, num_cols):
  winner = [0,0]
  shortest_distance = np.sqrt(data.shape[1]) # initialise with max distance
  input_data = data[t]
  for row in range(num_rows):
    for col in range(num_cols):
      distance = e_distance(som[row][col], data[t])
      if distance < shortest_distance:
        shortest_distance = distance
        winner = [row,col]
  return winner
```

۳. سپس در قدم بعد هر نرونی در شعاع همسایگی کمتری از neighbour range از نرون برنده قرار داشت با فرمول زیر وزن آن را آپدیت کنیم

$$som[row][col] \mathrel{+}= learning\_rate*(train\_x\_norm[t]-som[row][col])1$$

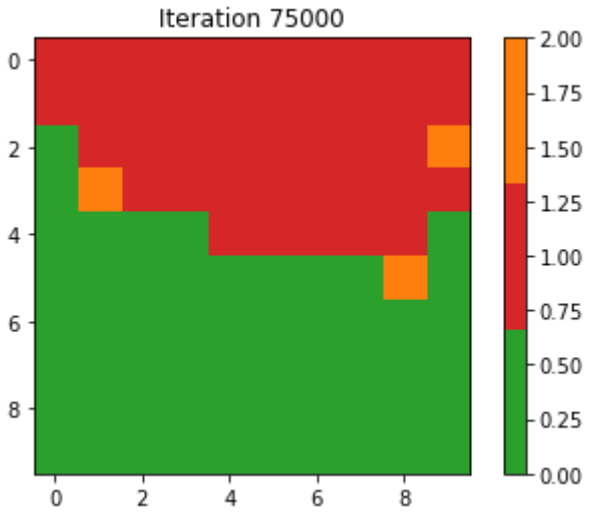
پس از نهایی شدن وزن هر کدام از نرون‌ها در مرحله بعد یکی یکی داده‌های train را می‌بینیم و برای آن از som یک winnig_neurn پیدا می‌کنیم و براساس مختصات نرون مربوطه و براساس label که داده مربوطه دارد ،به label های نرون مربوط در آرایه جدیدی به نام map اضافه می‌شود در واقع لیستی از label هایی که مربوط به این نرون می‌شود را در نهایت می‌سازیم .

```
label_data = train_y
map = np.empty(shape=(num_rows, num_cols), dtype=object)

for row in range(num_rows):
  for col in range(num_cols):
    map[row][col] = [] # empty list to store the label

for t in range(train_x_norm.shape[0]):
  if (t+1) % 1000 == 0:
    print("sample data: ", t+1)
  winner = winning_neuron(train_x_norm, t, som, num_rows, num_cols)
  map[winner[0]][winner[1]].append(label_data[t]) # label of winning neuron
```

به هر حال ممکن است بعد از این اقدام بعضی از داریه‌های ماتریس map یک لیست خالی باشند که در این صورت label نرون مربوطه برابر با عدد ۲ خواهد بود یعنی حالتی که برای ورودی در صورت نزدیکی به این عضو قابلیت آن که آن داده را به یکی از موارد گفته شده یک و صفر اختصاص بدهد نیست و در حالت دیگر براساس ما کزیمم label که در لیست map موجود است label آن نرون خاص برابر با اکثریت خواهد شد یعنی اگر در لیست به تعداد ۱۰ تا صفر و ۵ عدد یک باشد یعنی به ۱۰ تا صفر نزدیک است و به پنج تا یک نزدیک است که در نهایت دارای label صفر خواهد شد چون به تعداد بیشتری از یک نزدیک است .

```
[ ]  # construct label map
     label_map = np.zeros(shape=(num_rows, num_cols),dtype=np.int64)
     for row in range(num_rows):
       for col in range(num_cols):
         label_list = map[row][col]
         if len(label_list)==0:
           label = 2
         else:
           label = max(label_list, key=label_list.count)
         label_map[row][col] = label
```

در شکل زیر ناحیه‌ها هر کدام مشخص می‌کند که اگر داده ورودی به یکی از نرون‌های یکی از این مناطق نزدیکتر بود کدام یک از label قرمز سبز و نارنجی که می‌توانند همان ۰ و ۱ و ۲ باشند را به خود بگیرند .



برای predict داده‌های جدید مثلا داده‌های test خود در این سوال یکی‌یکی داده‌های مذکور را به می‌بینیم و از بین نرون‌های موجود نگاه می‌کنیم نزدیک‌ترین به داده فعلی کدام یک از نرون‌ها است و براساس همان نرون که نزدیکترین نزدیکی را به داده‌موجود داشت و براساس label که در ماتریس map برای آن در نظر گرفتیم پیش‌بینی می‌کنیم که داده داده شده متعلق به کدام کلاس است که با توجه به نتیجه‌ای که در این تست بدست آوردیم تمامی موارد را به درستی پیش‌بینی کرد .

```
]  # test data

   # using the trained som, search the winning node of corresponding to the test data
   # get the label of the winning node

   data = minmax_scaler(test_x) # normalisation

   winner_labels = []

   for t in range(data.shape[0]):
    winner = winning_neuron(data, t, som, num_rows, num_cols)
    row = winner[0]
    col = winner[1]
    predicted = label_map[row][col]
    winner_labels.append(predicted)

   print("Accuracy: ",accuracy_score(test_y, np.array(winner_labels)))
```

```
Accuracy:  1.0
```