Kohonen algorithm

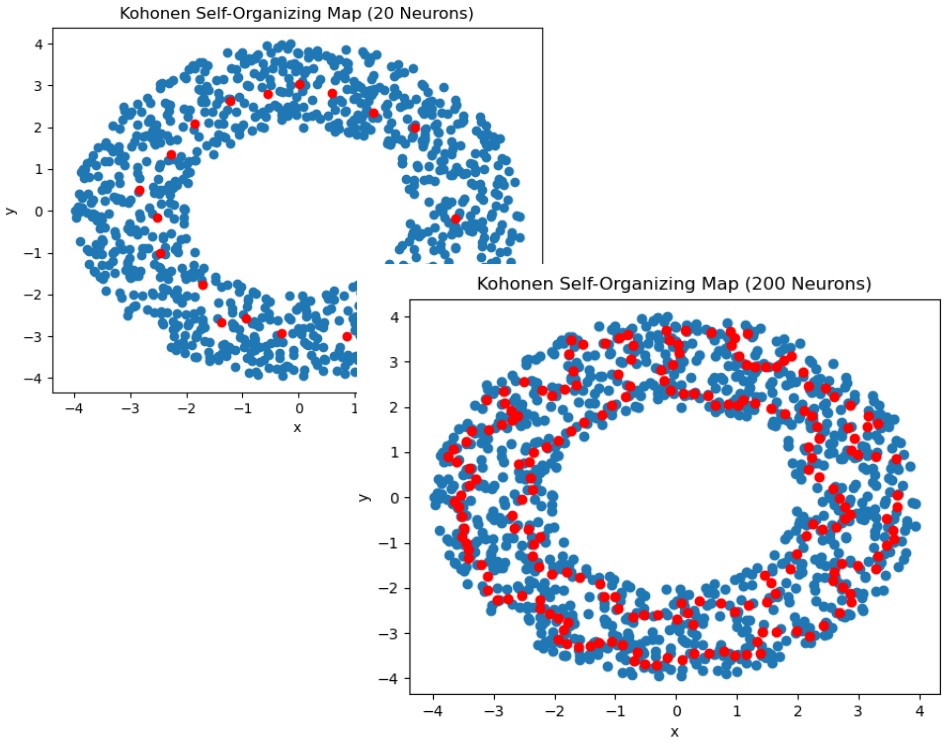
1. Submitters' names:

Abigail Isaacs - 212457535

Yeela Citron - 209914050

Talor Langnes - 204240477

Ohad Wolfman – 316552496



Link to the repository on Github:

https://github.com/ohadwolfman/neural_computation_Ex2_Kohonen_algorithm.git

2. introduction:

In this project, we implemented the Kohonen (SOM) algorithm and explore its behavior under different scenarios. The Kohonen algorithm, also known as the Self-Organizing Map algorithm, is a type of artificial neural network used for unsupervised learning and data visualization.

Part A:

The first part of the project involves fitting a line of random neurons in a data set consisting of points within the square range of $0 <= x <= 1$ and $0 <= y <= 1$. The distribution of the data points is uniform, while the Kohonen level is linearly ordered.The implementation of the algorithm will be tested with two different numbers of neurons: a small number (20) and a large number (200).

The aim is to observe the effect of varying the number of neurons on the algorithm's performance. Additionally, the evolution of the Kohonen map will be examined as the number of iterations increases.

The second part of the Part focuses on fitting a circle of neurons on a "donut" shape. The data set consists of points within the range $4 <= x^2+y^2 <= 16$.

A circle of 300 neurons will be used for this task.

Part B: (Did not implemented yet – to June 30th)

Part B of the project involves replicating an experiment known as the "monkey hand" problem. The data set is defined within a subset of the range $0 <= x <= 1$ and $0 <= y <= 1$, representing the shape of a hand. The Kohonen space consists of 400 neurons arranged in a 20 x 20 mesh. Initially, the entire hand is considered, and the evolution of the mesh is observed over iterations. Then, one of the fingers is "cut off," and the mesh is rearranged accordingly. The aim is to demonstrate how the Kohonen algorithm adapts to changes in the input data distribution.

3. Code review:
    a. Square shape:
        1) The Kohonen class contains the fields:
            - neurons_amount,
            - learning_rate,
            - Radius,
            - weights.

            Except the init function the class contains the following functions:

```python
def initialize_weights(self, input_dim):
    self.weights = np.random.rand(self.neurons_amount, input_dim)

def find_winner(self, input_data):
    distances = np.linalg.norm(input_data - self.weights, axis=1)
    return np.argmin(distances)

def update_weights(self, input_data, winner_idx):
    distance_to_winner = np.abs(np.arange(self.neurons_amount) - winner_idx)
    influence = np.exp(-distance_to_winner / self.radius)
    self.weights += self.learning_rate * influence[:, np.newaxis] * (input_data - self.weights)
```

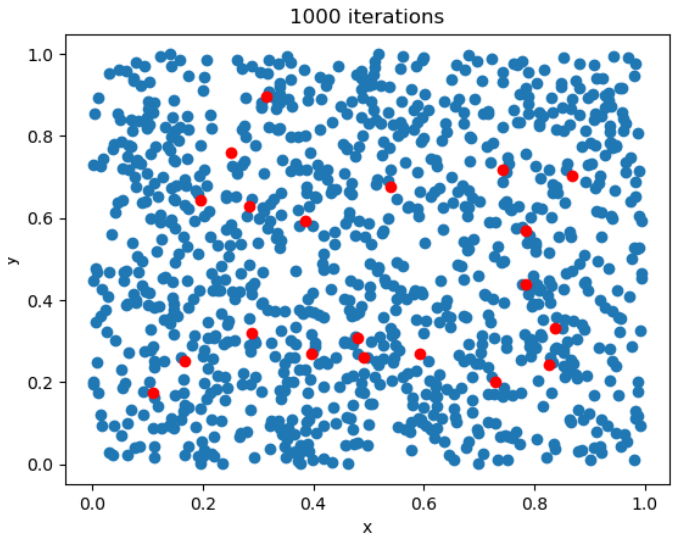
            - initialize_weights – that create samples of weights to the neurons
            - find_winner – for every point from the data, the winner is the neuron with the shortest distance from the that point.
            - update_weights – after we found the winner neuron we updating the weights of the neurons.
            - fit – training the model by updating the weights K times with the previous functions following the number of epochs required
              Into the fit function we added a boolean attribute 'toPlt' – if we want that during the fitting we will present the status of the model in a few possessions (1,30,70,100,300 etc.):

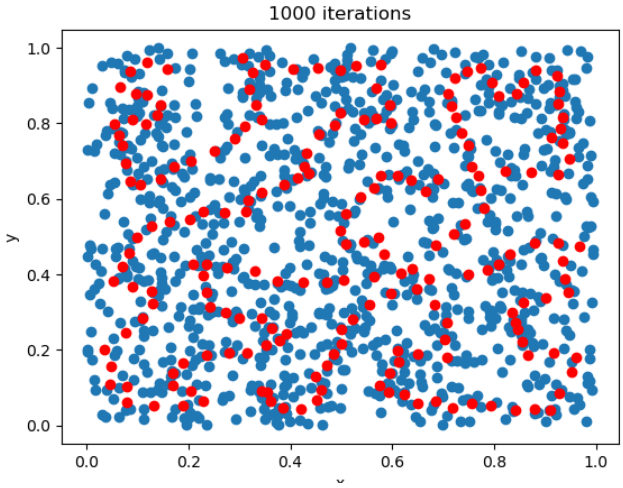
```python
for epoch in range(num_epochs):
    np.random.shuffle(data)

    if toPlt:
        if epoch == 0:
            plot_res_return(axes, 0, 0, data, self.weights)
            axes[0, 0].set_title('0 iterations')
        if epoch == 1:
            plot_res_return(axes, 0, 1, data, self.weights)
            axes[0, 1].set_title('1 iterations')
        if epoch == 30:
            plot_res_return(axes, 0, 2, data, self.weights)
            axes[0, 2].set_title('30 iterations')
        if epoch == 70:
```

2) First we generated the dataset – 1000 points (0<x,y<1) in 2D space X,Y. the we trained the model:

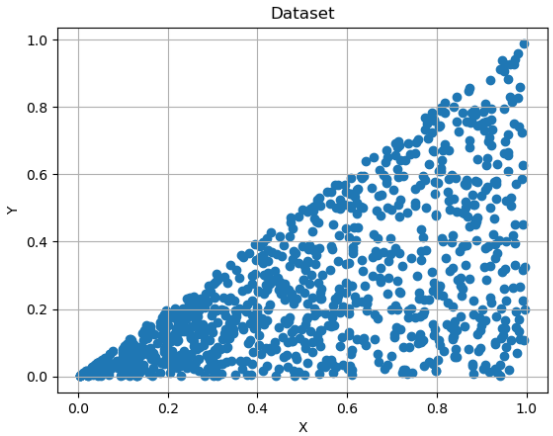
```python
# creating a square uniform dataset
data_square = create_random_dataset(1000)
create_data_plot(data_square)

# activate the kohonen algorithm with 20 neurons and plotting the results
Create_Kohonen_network(data_square, output_dim=20, learning_rate=0.5, num_epochs=1000, toPlt=False)

# activate the kohonen algorithm with 20 neurons and plotting the results
Create_Kohonen_network(data_square, output_dim=200, learning_rate=0.5, num_epochs=1000, toPlt=False)

# creating the first square un-uniform dataset
data_unUni_1 = create_unUni_1_dataset()
create_data_plot(data_unUni_1)
# same as above
Create_Kohonen_network(data_unUni_1, output_dim=20, learning_rate=0.5, num_epochs=1000, toPlt=False)
Create_Kohonen_network(data_unUni_1, output_dim=200, learning_rate=0.5, num_epochs=1000, toPlt=False)
```

3) Create a Kohonen network with 20 and 200 neurons and trained the model on these points:
(We used 1000 epochs and learning rate of 0.5 in all of the trainings below)



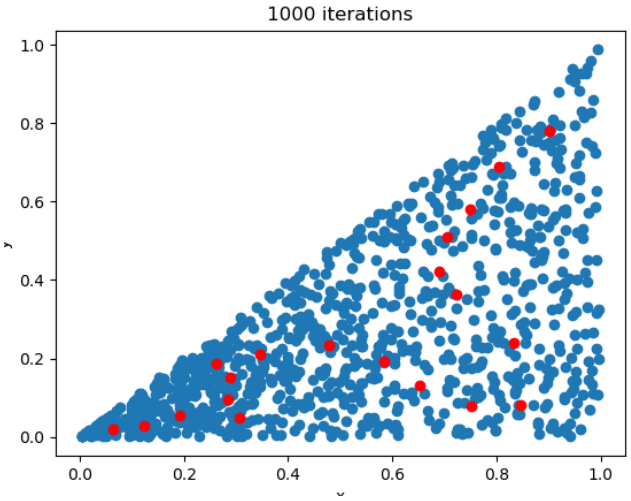


4) creating the first square un-uniform dataset:

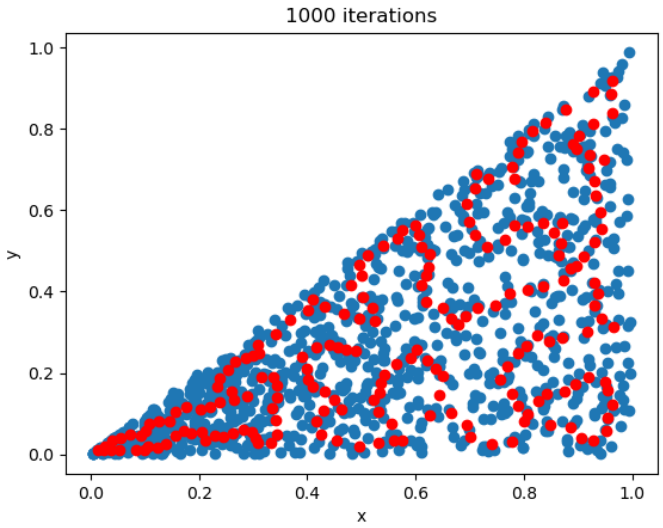


In this code, the data array is generated with non-uniform distributions based on the likelihood proportional to the size of x and uniform to the size of y. The x values are generated uniformly between 0 and 1, and the y values are generated uniformly between 0 and the corresponding x value.

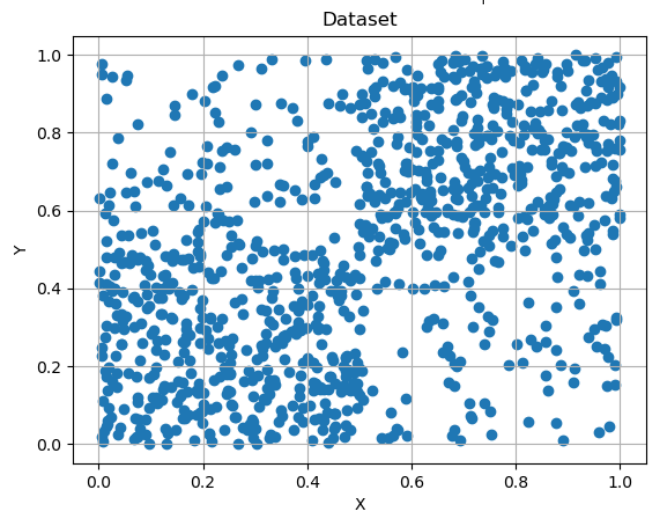
Then we train the model again with 20 and 200 points:



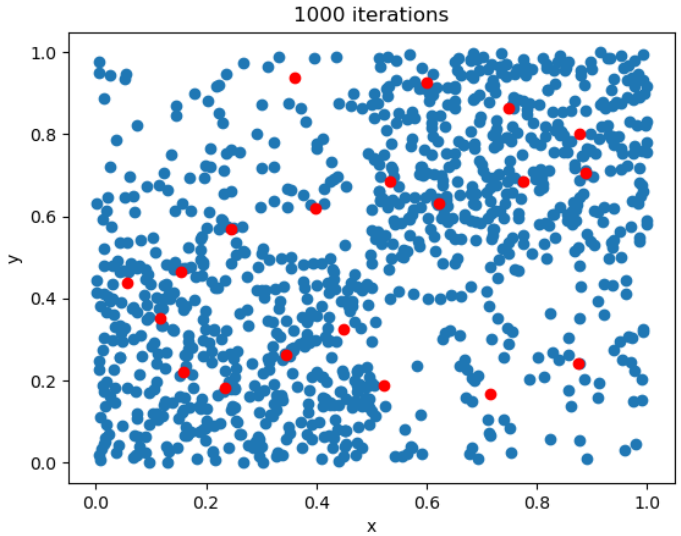
And 200 points:

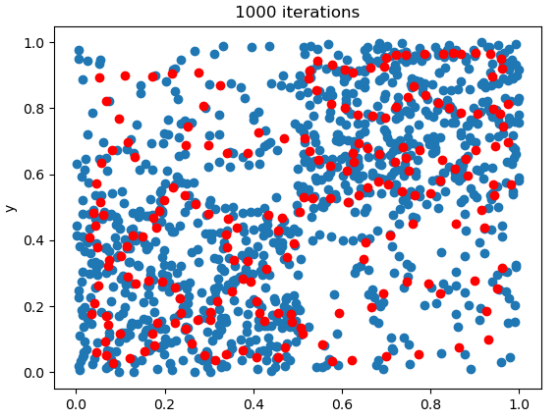


5) Then we created the second square un-uniform dataset:

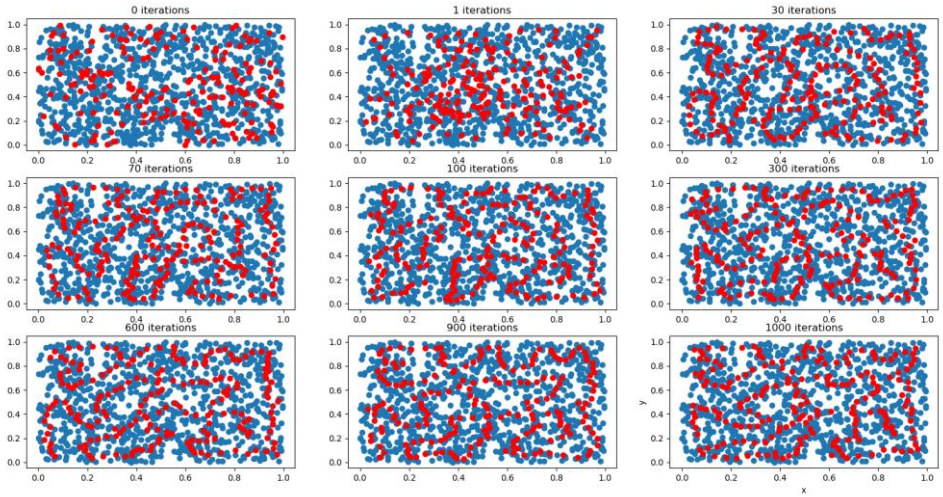


And we trained again with 20 and 200 points:

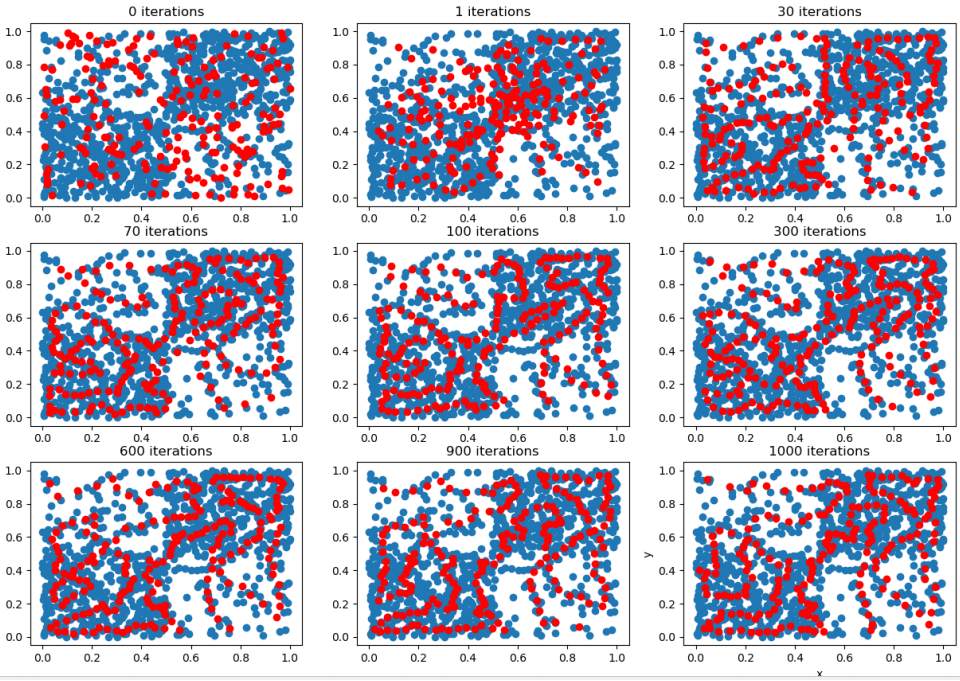




6) In the end we showed a fig containing 9 plots from 9 steps in the Kohonen algorithm the regular square uniform data (by passing to fit 'toPlt=True'):



And the second un-uniform square data:



b. Circle shape

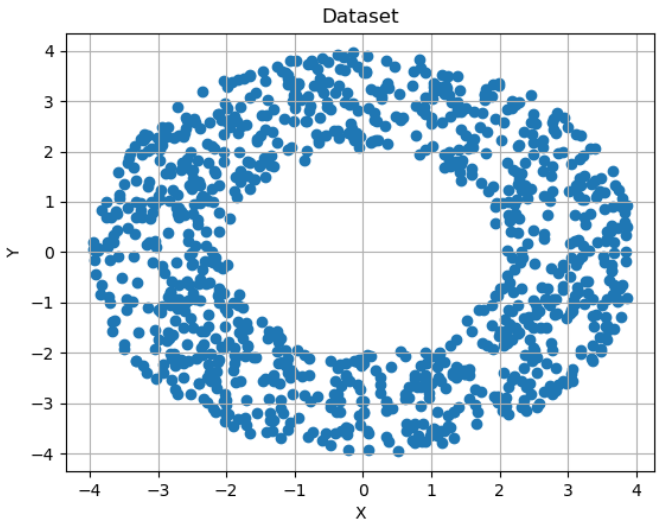
1) In the very same way we solved the circle problem:

```
# creating a donut uniform dataset
data_donut = create_circle_dataset()
create_data_plot(data_donut)
# activate the kohonen algorithm with 20 neurons and plotting the results
Create_Kohonen_network(data_donut, output_dim=20, learning_rate=0.5, num_epochs=1000, toPlt=False)
# activate the kohonen algorithm with 20 neurons and plotting the results
Create_Kohonen_network(data_donut, output_dim=200, learning_rate=0.5, num_epochs=1000, toPlt=False)

# creating the first donut un-uniform dataset
data_donut_unUni_1 = create_unUni_1_circle_dataset()
create_data_plot(data_donut_unUni_1)
# same as above
Create_Kohonen_network(data_donut_unUni_1, output_dim=20, learning_rate=0.5, num_epochs=1000, toPlt=False)
Create_Kohonen_network(data_donut_unUni_1, output_dim=200, learning_rate=0.5, num_epochs=1000, toPlt=False)
```

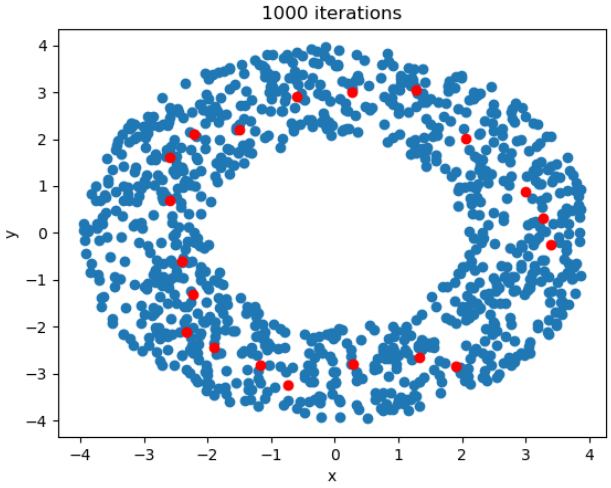
2) The create_circle_dataset function generates a synthetic circular dataset containing 100 points within a specified range. This dataset will be used to train the SOM.

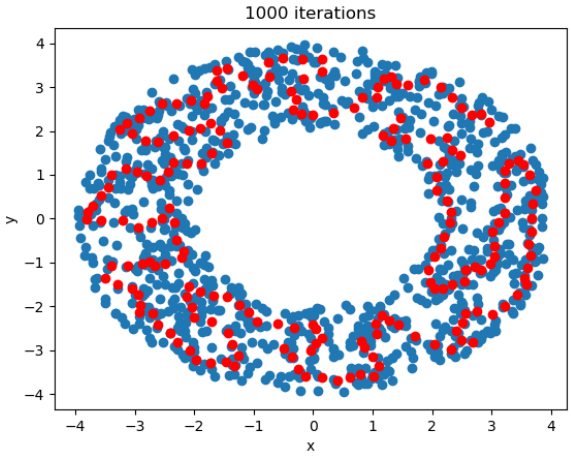
```
40  def create_circle_dataset():
41      dataset = []
42      while len(dataset) < 1000:  # Generate 100 points
43          x = np.random.uniform(-4, 4)
44          y = np.random.uniform(-4, 4)
45          distance_squared = x ** 2 + y ** 2
46          if 4 <= distance_squared <= 16:
47              dataset.append([x, y])
48      return np.array(dataset)
```

In this part of the code, a circular dataset is created using the create_circle_dataset function, and then a plot of the dataset is generated using the create_plot function.

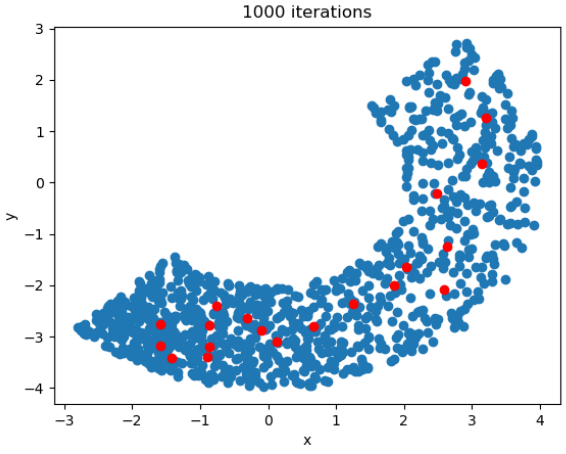


3) Next, we activated the kohonen algorithm with 20 and 200 neurons and plotted the result:

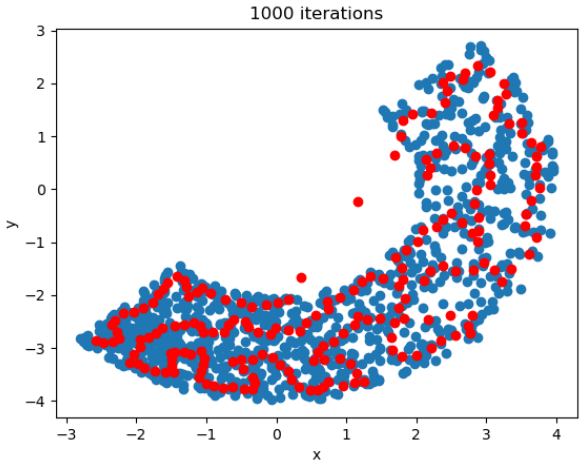




4) After creating the first donut un-uniform dataset the code creates two Kohonen networks with different numbers (20 and 200) of neurons, trains them using the dataset, and stores the resulting weights of the networks. The purpose is to compare the performance of the networks with different numbers of neurons in mapping and clustering the circular dataset.
20 neurons:

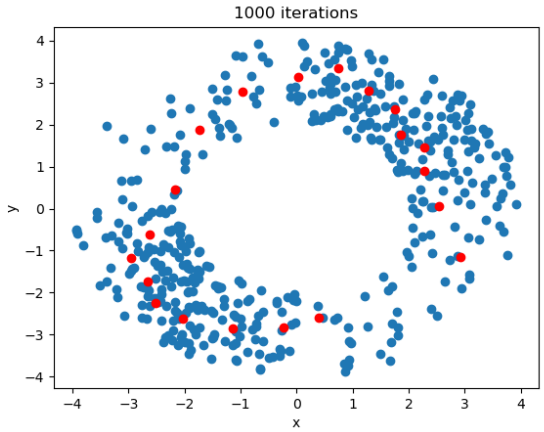


And 200 neurons:

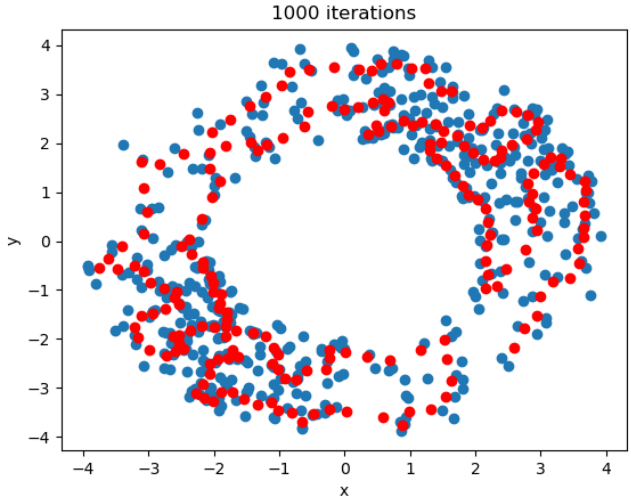


The resulting dataset will consist of 1000 data points that are distributed in a circular pattern within the specified radius range. The points will be closer to the origin and sparser as the distance from the origin increases.

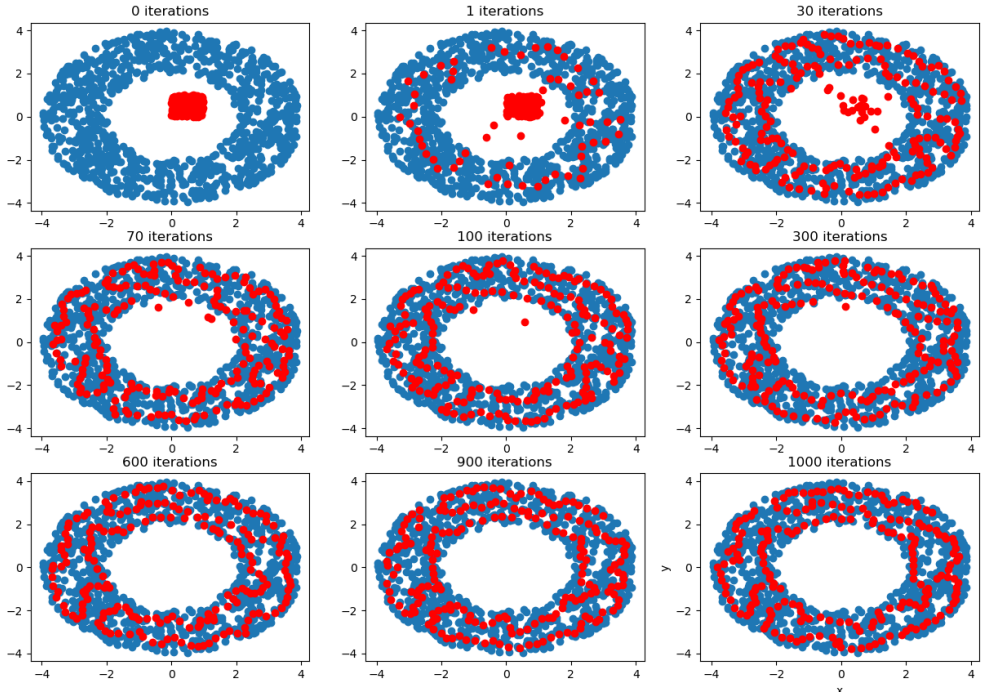
5) Creating the second non-uniform dataset on a circle (donut) and trining 20 neurons:



And 200 neurons:



6) In the end we showed a fig containing 9 plots from 9 steps in the Kohonen algorithm the regular square uniform data (by passing to fit 'toPlt=True'):



And the second un-uniform square data