Project 2

Project 2: Kohonen (SOM) Algorithm

Names & ID's :

Lara Abu Hamad [redacted]

[redacted]

Project 2

Part A

1. In this part we were asked to implement the Kohonen algorithm and use it to fit a set of 100 neurons.
   First we trained it on a set of 100 neurons arranged in a one dimensional.
   Here is an example:
   Create data:

```
data = create_data_a1(1000)
```

Figure 1: Data set of 1000 neurons

	0	1
0	0.684	0.559
1	0.629	0.192
2	0.835	0.763
3	0.707	0.359
4	0.009	0.723
5	0.277	0.754
6	0.804	0.599
7	0.070	0.472

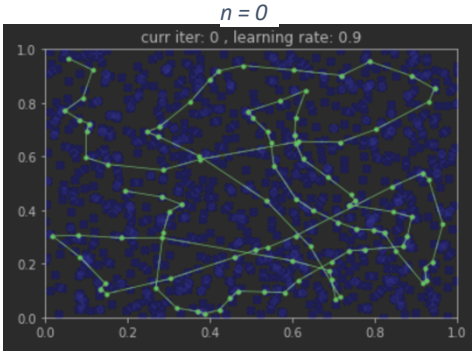
1,000 rows × 2 columns Open in new tab

Train data and plotting the results:

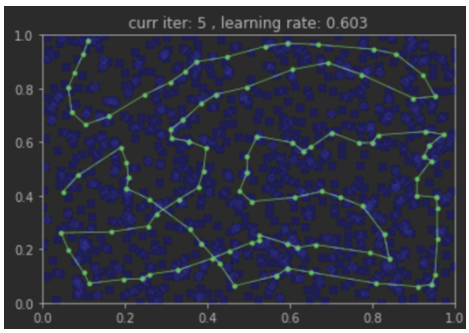
```
model = kohonen_1D(data, 100)
model.train()
```

Training this data set will get us these results:

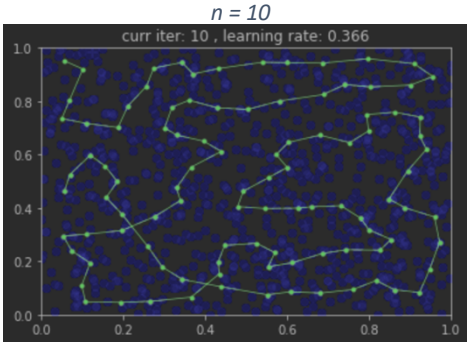
n = 0



n = 5

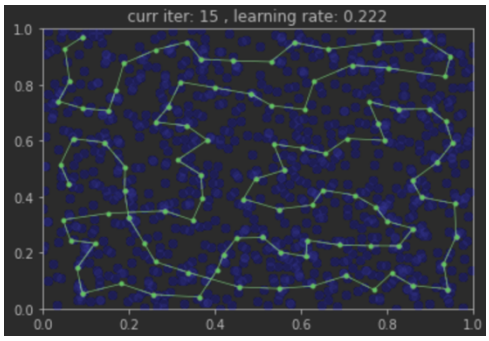


n = 10

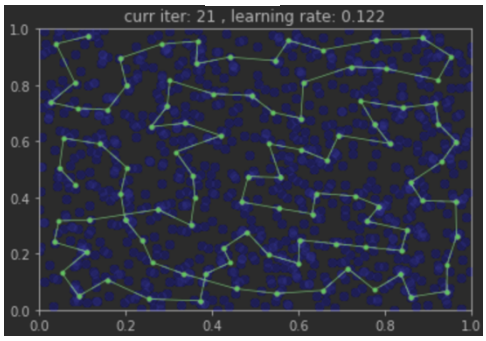


Project 2

n = 15



n = 21



We were also asked to do the same thing but now using a two dimensional array of 100 neurons

Here are some results of the same data set used before (in the one dimensional example):

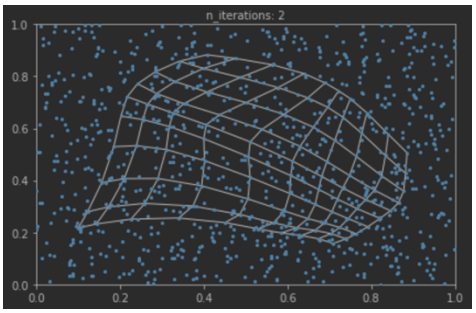
We used this functions to get these results and plotting them:

```
som = data
n = 10
eps = .5
de = 3
nr = 1
rep = 100
ste = 0
#%%
def gaussian_proximity(ix, iy, kx, ky, d):
    return np.exp(-((ix - kx) ** 2 + (iy - ky) ** 2) / (d ** 2))

def point():
    x = np.random.random()
    y = np.random.random()
    return np.array([x, y])

def euclidean(p1, p2):
    return (p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2

def plot(iter):
    plt.title("n_iterations: " + str(iter), fontsize=10)
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.scatter(som[:, 0], som[:, 1], s=5, zorder=3)
    for j in range(n):
        for i in range(n - 1):
            plt.plot([som[i + n * j, 0], som[i + 1 + n * j, 0]], [som[i + n *
```

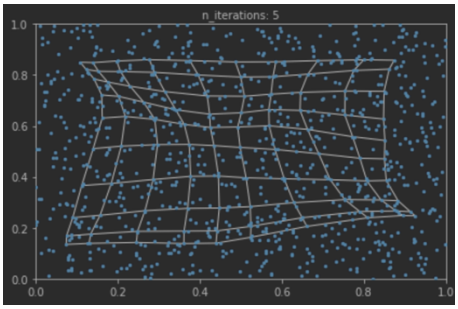
Project 2

```
j, 1], som[i + 1 + n * j, 1]],
                    linewidth=1, c='black')
    for j in range(n - 1):
        for i in range(n):
            plt.plot([som[i + n * j, 0], som[i + n * (j + 1), 0]], [som[i + n
* j, 1], som[i + n * (j + 1), 1]],
                    linewidth=1, c='black')
    plt.tight_layout()
    plt.show()

for nr in range(1, 15):
    for _ in range(nr):
        eps = eps * .97
        de = de * .98
        for _ in range(rep):
            ste = ste + 1
            p = point()
            dist = [euclidean(p, som[l]) for l in range(n * n)]
            ind_min = np.argmin(dist)
            ind_i = ind_min % n
            ind_j = ind_min // n

            for j in range(n):
                for i in range(n):
                    som[i + n * j] += eps * gaussian_proximity(ind_i, ind_j,
i, j, de) * (p - som[i + n * j])
    plot(nr)
```

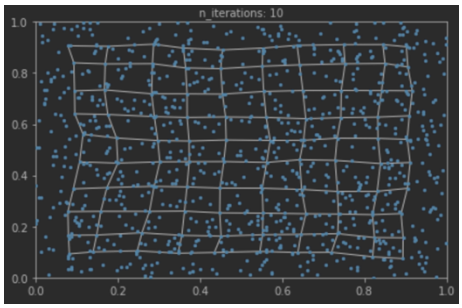
n = 2



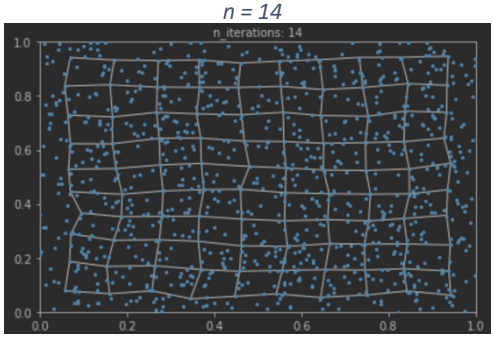
n = 5



n = 10



n = 14



Project 2

As we can see, when the number of iterations increases, it affects the positions of the neurons, and that's how we get more uniform data.(uniform distribution)

2. In this question we were asked to do the same thing as in 1.a, but here we need to create at least two non uniform distribution
   An example of when 80% of the data is: {(x,y)|0<=x,y<=0.7} and the rest of the 20% is: {(x,y)|0.7<x,y<=1}

```
Create the data: data_2a = create_data_a2(1000)
```

Figure 2: Data set

	0	1
0	0.866	0.964
1	0.065	0.068
2	0.376	0.347
3	0.169	0.166
4	0.110	0.644
5	0.413	0.645
6	0.210	0.519
7	0.665	0.100

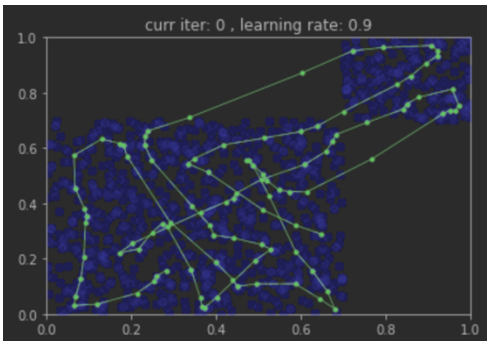
1,000 rows × 2 columns Open in new tab

Train the data and plotting the results:

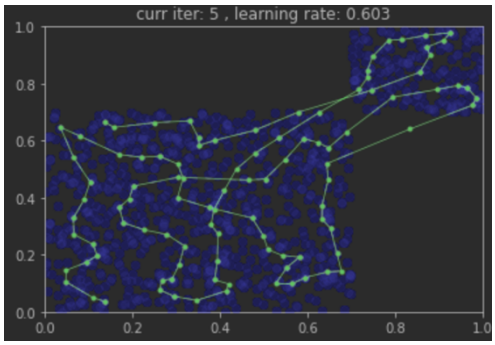
```
model_2a = kohonen_1D(data_2a, 100)
model_2a.train()
```

The results we got:

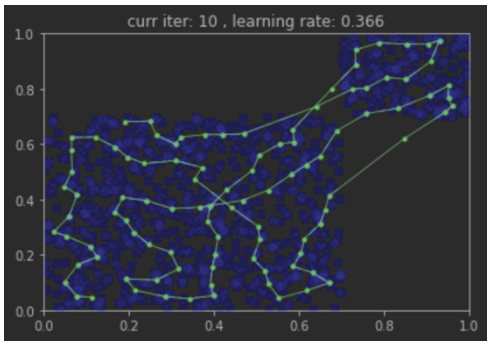
n = 0



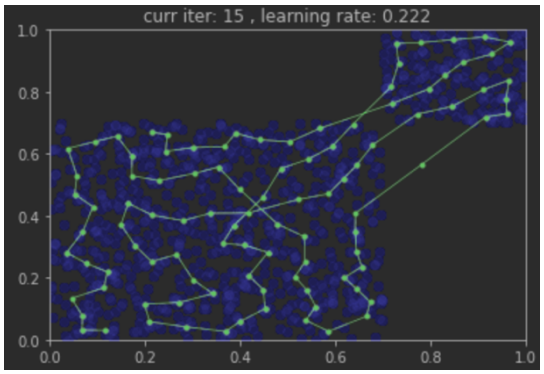
n = 5



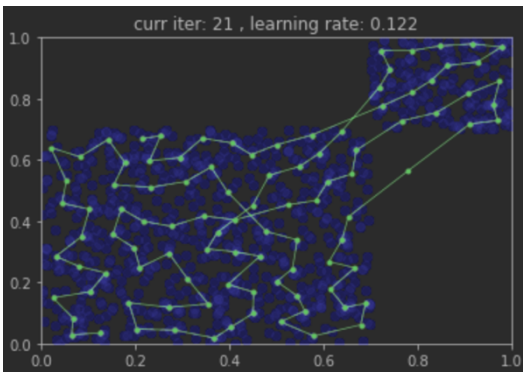
n = 10



n = 15



n = 21



Now let's see an example when the data set is build in this way:
10% :{(x,y)| 0<=x,y<=200}, 80%: {(x,y)|100<x,y<=800}, 10%: {(x,y)|800<x,y<=1000}
Create the data:

```
data_2a2 = create_data_a2_2(1000)
```

Figure 3: Data set

	0	1
0	0.451	0.753
1	0.563	0.424
2	0.674	0.638
3	0.354	0.554
4	0.451	0.593
5	0.009	0.185
6	0.883	0.980
7	0.069	0.169

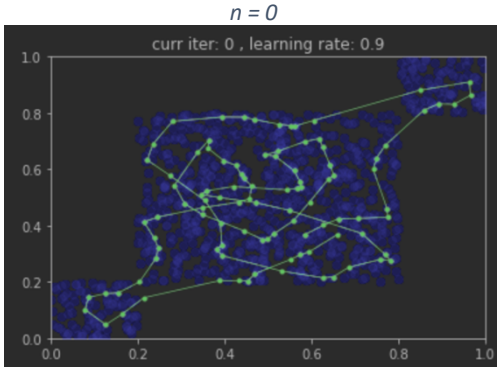
1,000 rows × 2 columns Open in new tab

Train the data and plotting the results:

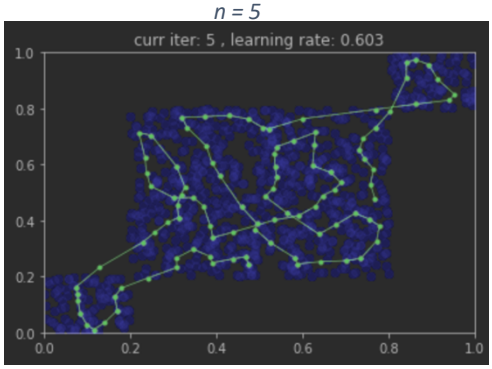
```
model_2a2 = kohonen_1D(data_2a2, 100)
model_2a2.train()
```

The results we got:

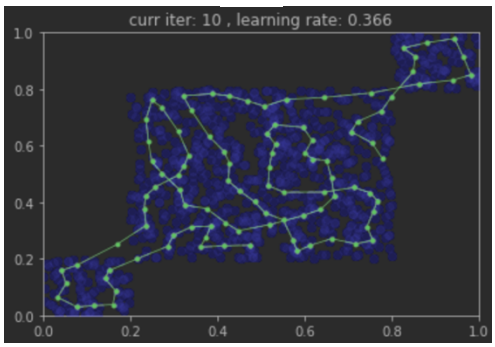
n = 0



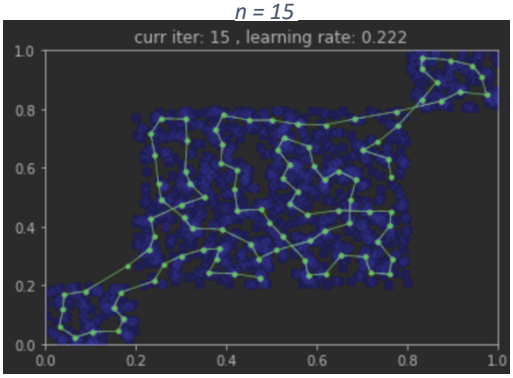
n = 5



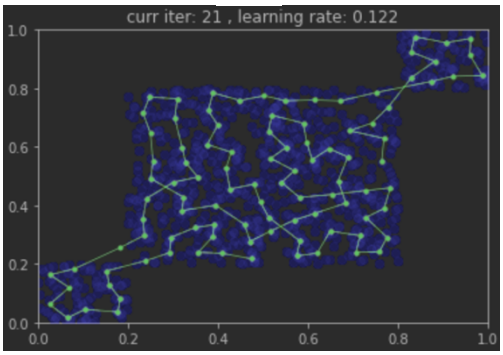
n = 10



n = 15



n = 21



3. Now we should do the same experiment but we need to fit the neurons on a “donut” shape, the data should be from the form {(x,y)|2<= x^2+y^2 <= 4}
   Create data :

```
data_3a = create_data_a3(1000)
```

For example:

Figure 4: data set

	0	1
0	-0.20	1.44
1	1.75	0.19
2	-0.78	1.57
3	-1.51	1.01
4	-0.51	1.48
5	1.05	1.62
6	0.01	1.95
7	-1.03	-1.30

1,000 rows × 2 columns Open in new tab

Train and plot the data:

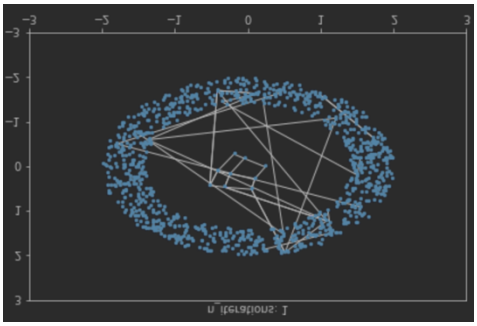
```
n = 3
eps = .5
de = 3
nr = 1
rep = 100
ste = 0
som = data_3a

def gaussian_proximity(ix, iy, kx, ky, d):
    return np.exp(-((ix - kx) ** 2 + (iy - ky) ** 2) / (d ** 2))


def point():
    while(1):
        x = random.randint(-1000, 1000) / 100
        y = random.randint(-1000, 1000) / 100
        if 2 <= x**2 + y**2 <= 4:
            break

    return np.array([x, y])


def euclidean(p1, p2):
    return (p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2


def plot(iter):
    plt.title("n_iterations: " + str(iter), fontsize=10)
    plt.xlim(-3, 3)
    plt.ylim(-3, 3)
    plt.scatter(som[:, 0], som[:, 1], s=5, zorder=3)
    for j in range(10):
        for i in range(2):
            plt.plot([som[i + n * j, 0], som[i + 1 + n * j, 0]], [som[i + n *
j, 1], som[i + 1 + n * j, 1]],
                     linewidth=1, c='black')
    for j in range(2):
        for i in range(10):
            plt.plot([som[i + n * j, 0], som[i + n * (j + 1), 0]], [som[i + n
* j, 1], som[i + n * (j + 1), 1]],
                     linewidth=1, c='black')
    plt.tight_layout()
    plt.show()


for nr in range(1, 15):
    for _ in range(nr):
        eps = eps * .97
        de = de * .98
        for _ in range(rep):
            ste = ste + 1
```

Project 2

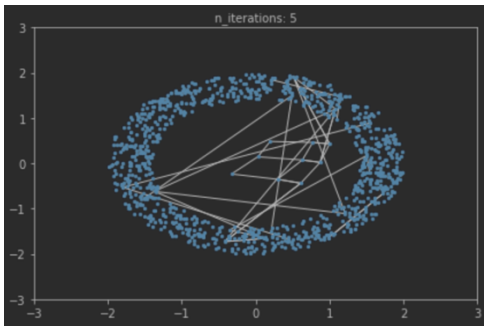
```
            p = point()
            dist = [euclidean(p, som[l]) for l in range(n * n)]
            ind_min = np.argmin(dist)
            ind_i = ind_min % n
            ind_j = ind_min // n

            for j in range(n):
                for i in range(n):
                    som[i + n * j] += eps * gaussian_proximity(ind_i, ind_j,
i, j, de) * (p - som[i + n * j])
    plot(nr)
```

The results we got:

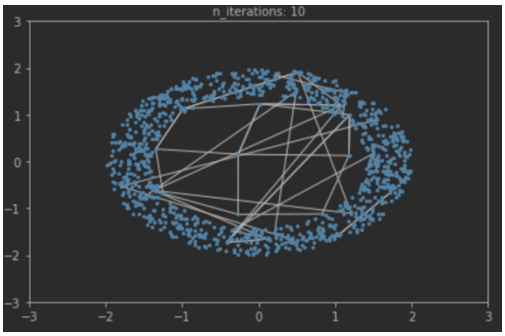
n = 1



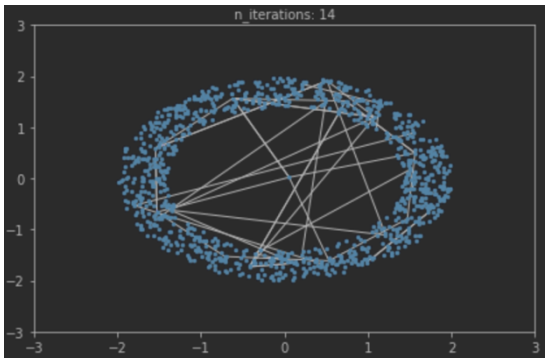
n = 5



n = 10



n = 14



Project 2

Part B

1. In this part we were asked to recreate the experiment “monkey hand” from class.
   4 fingers:
   $\{(x,y) \mid 0 <= x <= 1,\ 0 <= y <= 1\}$
   Create data:

```
data_1b = create_data_b1()
```

Example:

Figure 5: Data set

	0	1
0	0.093860	0.028347
1	0.835765	0.432767
2	0.762280	0.002106
3	0.901427	0.030590
4	0.939149	0.381204
5	0.437888	0.495812
6	0.289782	0.021490
7	0.837578	0.556454

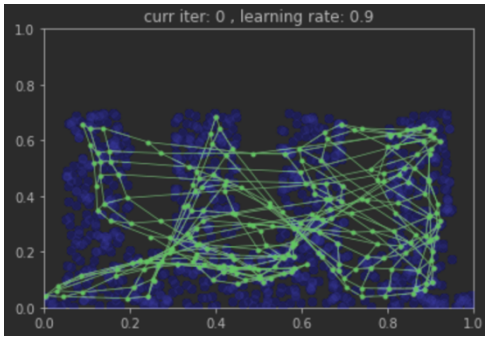
1,000 rows × 2 columns Open in new tab

Train and plot the data:

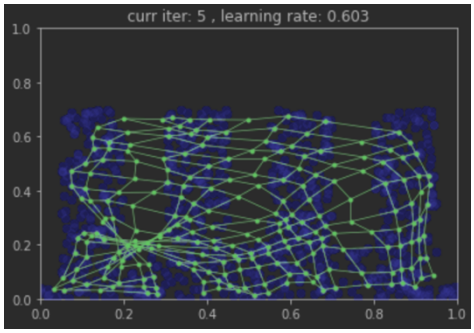
```
model_1b = kohonen_1D(data_1b, 225)
model_1b.train()
```

Results :

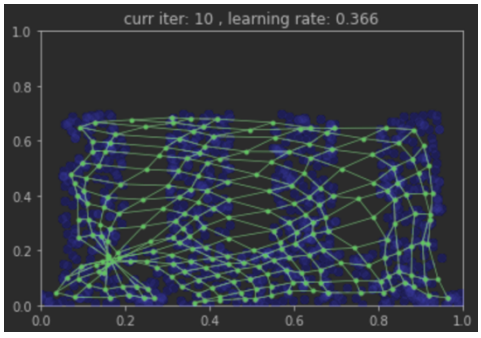
n = 0



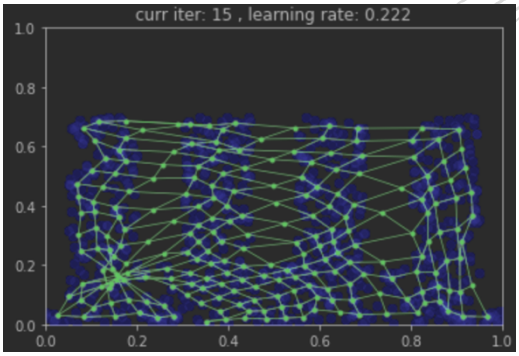
n = 5



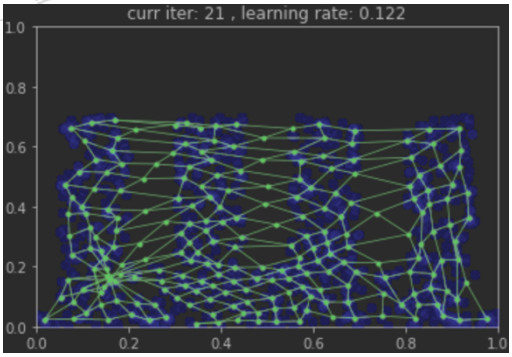
n = 10



n = 15



n = 21



2. Now the same thing but we “cut a finger”, 3 fingers hand:
   Create data:

```
data_2b = create_data_b2()
```

Example:

Figure 6: Data set

	0	1
0	0.651593	0.788723
1	0.093860	0.028347
2	0.835765	0.432767
3	0.762280	0.002106
4	0.901427	0.030590
5	0.939149	0.381204
6	0.289782	0.021490
7	0.837578	0.556454

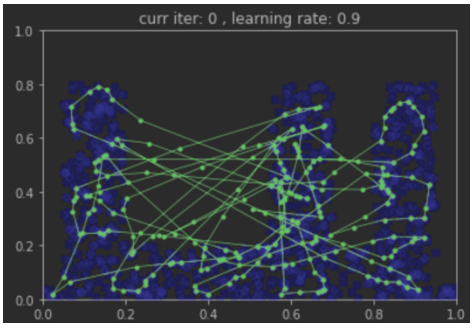
1,000 rows × 2 columns Open in new tab

Train and plot the data:

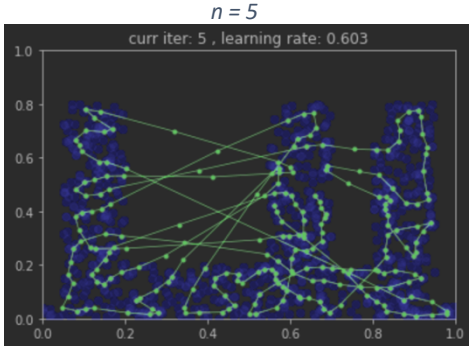
```
model_2b = kohonen_1D(data_2b, 225)
model_2b.train()
```

Results:

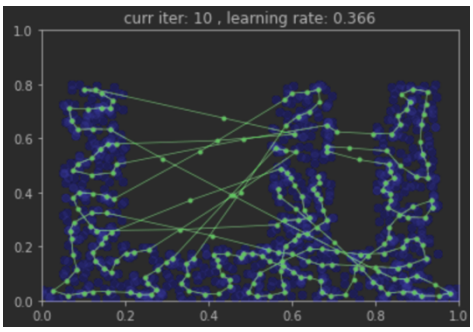
n = 0



n = 5

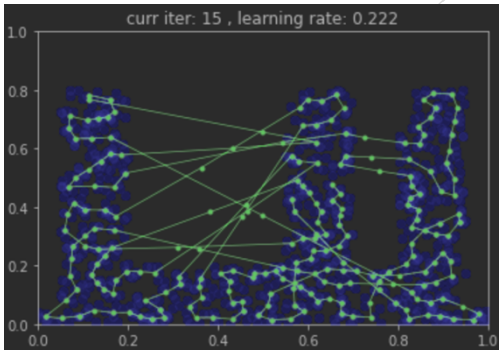


n = 10

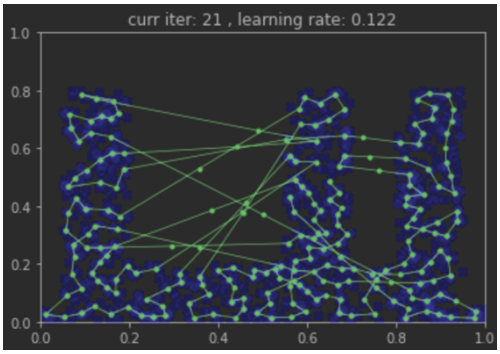


Project 2

n = 15



n = 21



In [4]:

```
import numpy as np
import random
import matplotlib.pyplot as plt
```

Part A + B :

create data :

In [5]:

```
def create_data_a1(size):
    r = np.random.RandomState(0)
    data = r.randint(0, 1000, (size, 2)) / 1000
    return data
```

In [6]:

```
def create_data_a2(size):
    r = np.random.RandomState(0)
    d1 = r.randint(0, 700, (int(size * 0.8), 2)) / 1000
    d2 = r.randint(701, 1000, (int(size * 0.2), 2)) / 1000
    data = np.vstack((d1, d2))
    r.shuffle(data)
    return data
```

In [7]:

```
def create_data_a2_2(size):
    r = np.random.RandomState(0)
    d1 = r.randint(0, 200, (int(size * 0.1), 2)) / 1000
    d2 = r.randint(201, 800, (int(size * 0.8), 2)) / 1000
    d3 = r.randint(801, 1000, (int(size * 0.1), 2)) / 1000
    data = np.vstack((d1, d2))
    data = np.vstack((data, d3))
    r.shuffle(data)
    return data
```

In [26]:

```
def create_data_a3(size):
    data = np.ndarray((size, 2))
    i = 0
    while i < size:
        x = random.randint(-1000, 1000) / 100
        y = random.randint(-1000, 1000) / 100
        if 2 <= (x ** 2) + (y ** 2) <= 4:
            data[i, 0] = x
            data[i, 1] = y
            i += 1
    return data
```

In [9]:

```
def create_data_b1():
    data = np.zeros((1000, 2))
    random.seed(11)
    mask = np.array([
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

```
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
        [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
        [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    ])
    mask = np.flip(mask).T
    n = 0
    random.seed(1)
    while n < 1000:
        x = random.uniform(0, 20)
        y = random.uniform(0, 20)
        i = int(x)
        j = int(y)
        if mask[i, j] == 1:
            data[n, 0] = x / 20
            data[n, 1] = y / 20
            n += 1
    return data
```

In [10]:

```
def create_data_b2():
    data = np.zeros((1000, 2))
    random.seed(11)
    mask = np.array([
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
        [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
        [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
        [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    ])
    mask = np.flip(mask).T
    n = 0
    random.seed(1)
    while n < 1000:
        x = random.uniform(0, 20)
        y = random.uniform(0, 20)
        i = int(x)
        j = int(y)
        if mask[i, j] == 1:
            data[n, 0] = x / 20
            data[n, 1] = y / 20
            n += 1
    return data
```

In [11]:

```
class kohonen_1D:
    def __init__(self, train_data: np.ndarray, num_of_neurons: int):
        r = np.random.RandomState(0)
        self.data = train_data
        self.num_of_neurons = num_of_neurons
        self.som = r.randint(0, 1000, (num_of_neurons, 2)).astype(float) / 1000

    def closest_neuron(self, sample):
        dist = ((self.som - sample) ** 2).sum(axis=1)
        return np.unravel_index(np.argmin(dist, axis=None), dist.shape)

    def plot(self, t):
        x = self.som[:, 0]
        y = self.som[:, 1]
        fig, ax = plt.subplots()
        ax.set_xlim(0, 1)
        ax.set_ylim(0, 1)
        xs = []
        ys = []
        for i in range(self.som.shape[0]):
            xs.append(self.som[i, 0])
            ys.append(self.som[i, 1])

        ax.plot(xs, ys, 'g-', markersize=0, linewidth=0.7)
        ax.plot(x, y, color='g', marker='o', linewidth=0, markersize=3)
        ax.scatter(self.data[:, 0], self.data[:, 1], c="b", alpha=0.2)
        plt.title(t)
        plt.show()

    def update_weights(self, sample, learning_rate, radius, closest_neuron, step=3):
        x = closest_neuron[0]
        if radius < 1e-2:
            self.som[x, :] += learning_rate * (sample - self.som[x, :])
            return self.som
        for i in range(max(0, x - step), min(self.som.shape[0], x + step)):
            dist = (i - x) ** 2
            func = np.exp(-dist / 2 / radius)
            self.som[i, :] += learning_rate * func * (sample - self.som[i, :])
        return self.som

    def train(self, learning_rate=0.9, radius=1, lr_decay=0.1, rad_decay=0.1, epochs=22)
:
        r = np.random.RandomState(0)
        lr = learning_rate
        r0 = radius
        for epoch in np.arange(0, epochs):
            r.shuffle(self.data)
            for sample in self.data:
                x = self.closest_neuron(sample)
                self.som = self.update_weights(sample, learning_rate, radius, x)
            self.plot("curr iter: " + str(epoch) + " , learning rate: " + str(round(lear
ning_rate, 3)))
            learning_rate = lr * np.exp(-epoch * lr_decay)
            radius = r0 * np.exp(-epoch * rad_decay)
        return self.som
```

In [ ]:

```
data_1a = create_data_a1(1000)
```

In [ ]:

```
n = 10
eps = .5
de = 3
nr = 1
rep = 100
ste = 0
som = data_1a

def gaussian_proximity(ix, iy, kx, ky, d):
```

```
    return np.exp(-((ix - kx) ** 2 + (iy - ky) ** 2) / (d ** 2))


def point():
    x = np.random.random()
    y = np.random.random()
    return np.array([x, y])


def euclidean(p1, p2):
    return (p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2


def plot(iter):
    plt.title("n_iterations: " + str(iter), fontsize=10)
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.scatter(som[:, 0], som[:, 1], s=5, zorder=3)
    for j in range(n):
        for i in range(n - 1):
            plt.plot([som[i + n * j, 0], som[i + 1 + n * j, 0]], [som[i + n * j, 1], so
m[i + 1 + n * j, 1]],
                     linewidth=1, c='black')
    for j in range(n - 1):
        for i in range(n):
            plt.plot([som[i + n * j, 0], som[i + n * (j + 1), 0]], [som[i + n * j, 1],
som[i + n * (j + 1), 1]],
                     linewidth=1, c='black')
    plt.tight_layout()
    plt.show()


for nr in range(1, 15):
    for _ in range(nr):
        eps = eps * .97
        de = de * .98
        for _ in range(rep):
            ste = ste + 1
            p = point()
            dist = [euclidean(p, som[l]) for l in range(n * n)]
            ind_min = np.argmin(dist)
            ind_i = ind_min % n
            ind_j = ind_min // n

            for j in range(n):
                for i in range(n):
                    som[i + n * j] += eps * gaussian_proximity(ind_i, ind_j, i, j, de) *
(p - som[i + n * j])
    plot(nr)
```

In [ ]:

```
data_3a = create_data_a3(1000)
```

In [ ]:

```
n = 3
eps = .5
de = 3
nr = 1
rep = 100
ste = 0
som = data_3a

def gaussian_proximity(ix, iy, kx, ky, d):
    return np.exp(-((ix - kx) ** 2 + (iy - ky) ** 2) / (d ** 2))


def point():
    while(1):
        x = random.randint(-1000, 1000) / 100
```

```
        y = random.randint(-1000, 1000) / 100
        if 2 <= x**2 + y**2 <= 4:
            break

    return np.array([x, y])


def euclidean(p1, p2):
    return (p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2


def plot(iter):
    plt.title("n_iterations: " + str(iter), fontsize=10)
    plt.xlim(-3, 3)
    plt.ylim(-3, 3)
    plt.scatter(som[:, 0], som[:, 1], s=5, zorder=3)
    for j in range(10):
        for i in range(2):
            plt.plot([som[i + n * j, 0], som[i + 1 + n * j, 0]], [som[i + n * j, 1], so
m[i + 1 + n * j, 1]],
                     linewidth=1, c='black')
    for j in range(2):
        for i in range(10):
            plt.plot([som[i + n * j, 0], som[i + n * (j + 1), 0]], [som[i + n * j, 1],
som[i + n * (j + 1), 1]],
                     linewidth=1, c='black')
    plt.tight_layout()
    plt.show()


for nr in range(1, 15):
    for _ in range(nr):
        eps = eps * .97
        de = de * .98
        for _ in range(rep):
            ste = ste + 1
            p = point()
            dist = [euclidean(p, som[l]) for l in range(n * n)]
            ind_min = np.argmin(dist)
            ind_i = ind_min % n
            ind_j = ind_min // n

            for j in range(n):
                for i in range(n):
                    som[i + n * j] += eps * gaussian_proximity(ind_i, ind_j, i, j, de) *
(p - som[i + n * j])
    plot(nr)
```