Project II

Due date: June 8th

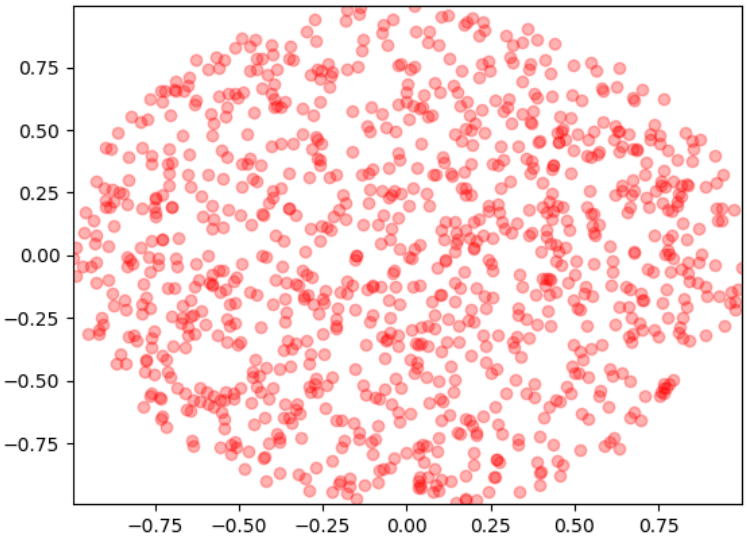
(Names, ID's): (Joshua Gordon , 332307073), (Arieh Norton, 315074963), (Natali Djavarov, 208419523), (Liron Asaraf, 206955841)

Part 1:

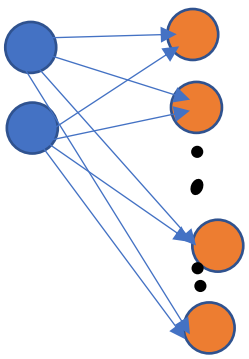
Q1)
Implement the Kohonen algorithm and use it to fit 1) a set of 100 neurons in a topology of a line to a disk.
(That is, the data set is {(x,y) | 0 <= x <= 1, 0<=y<=1} for which the distribution is uniform while the Kohonen level is linearly ordered.) You can obtain data by sampling the data set.

Answer:

First we created our data set under the constraints {(x,y) I 0 <= x <=1, 0 <= y <=1} as shown in the image below with 1000 data points:



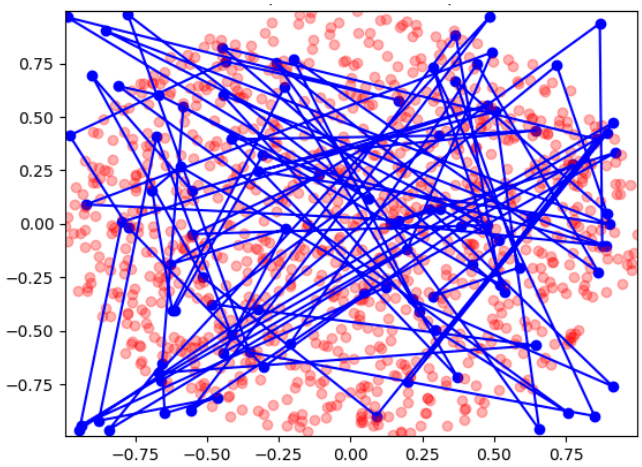
we then designed the Kohonen self organizing map neural network such that the network would have two inputs (x, y) , and 100 clusters:



we initialized random weights for the network and started the Kohonen algorithm and will present step by step what happened over a certain amount of iterations.

0 iterations aka starting point:

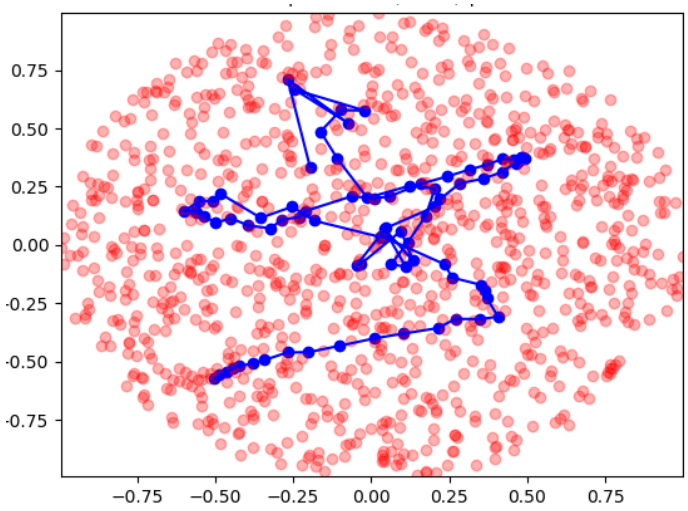
We can see the "**spaghetti**" map. This is not surprising and is due to the fact the we initialized **random weights.** it would be quite spectacular from random generated weights to receive any thing other than random points on our map, therefor we expected this outcome.



we then started the algorithm as taught in class:

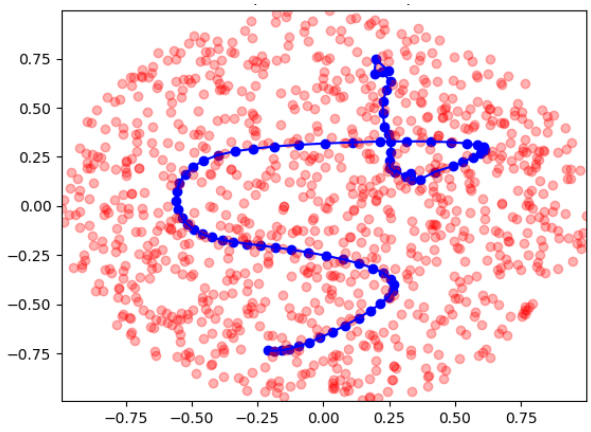
Measuring the Euclidean distance and finding the minimum j that of Wij and then updating the weights and weights neighbors accordingly. With a learning rate of 0.4 and radius of neighbors starting at 60.

After 50 iterations I received the following:

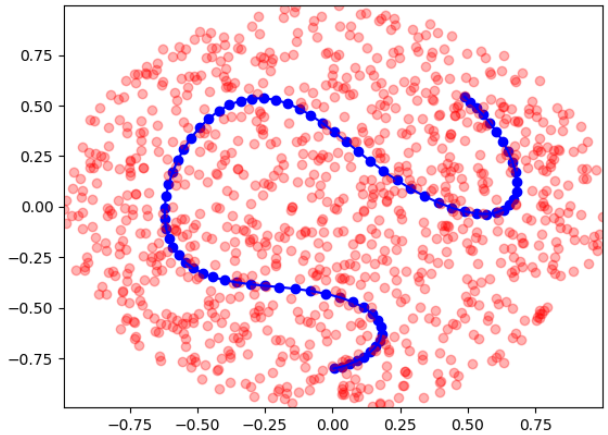


As we can see the map Is starting to "unravel" and form itself although its far from perfect.

After an additional 50 iterations I received the following:



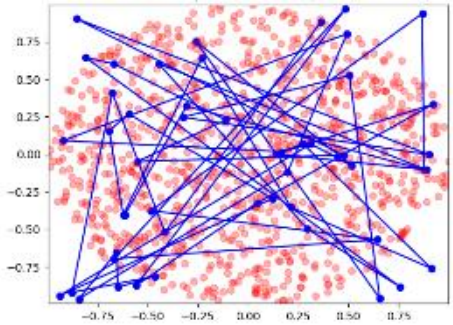
And after total of 1500 iterations we received:



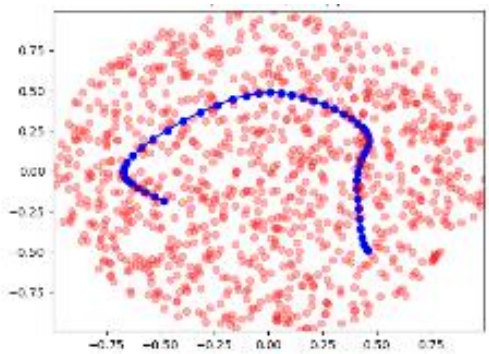
we then tried with a smaller number of clusters as we felt the ratio between 1000 points to 100 clusters might be too high:

Below are the results of the same data set of 1000 points with 50 clusters instead of 100:

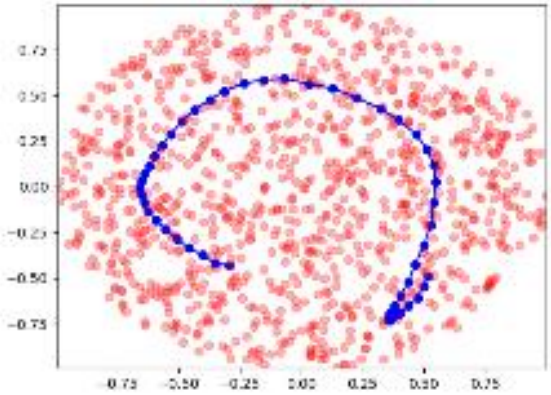
As expected like previously we start we our “pasta” starting state:



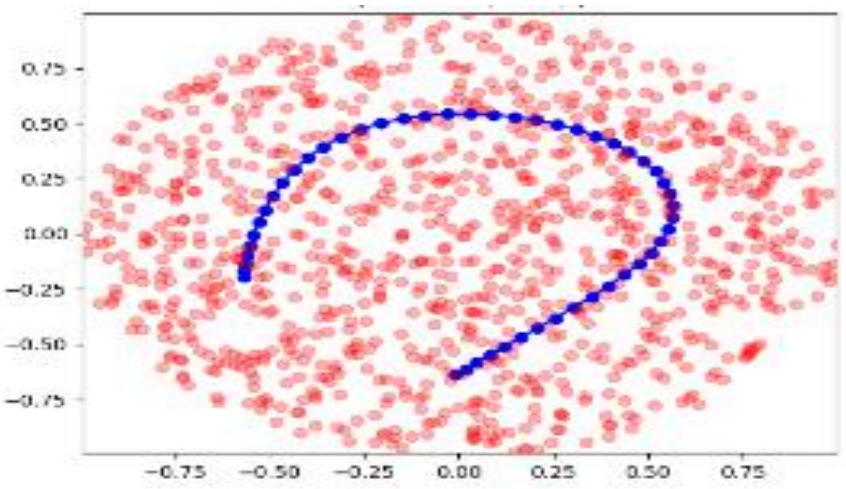
We then after 500 iterations recived the follwing:



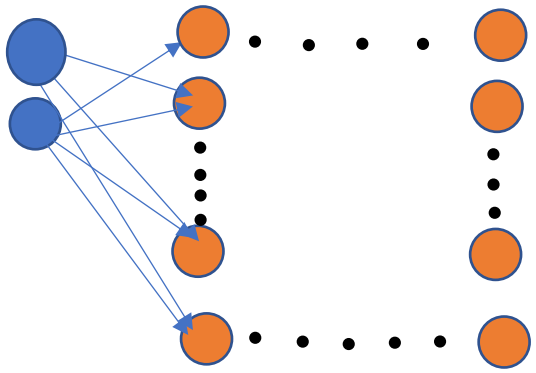
Then after 1500 iterations we received the following:



Finally after a few more thousand iterations we received the following

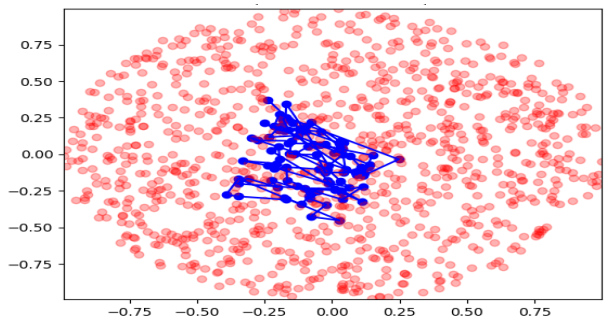


For the 100 neurons as a 10 X 10 2D array we rearranged the networks architecture to the following:

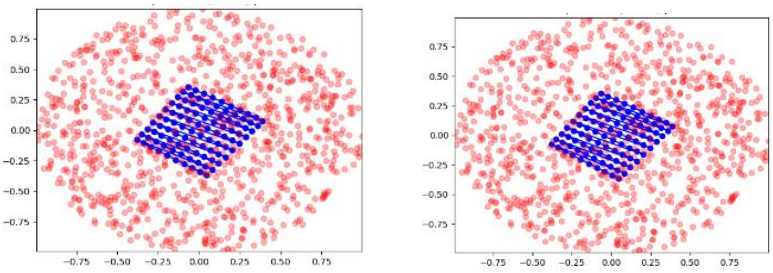


Also updated the number of neighbors to be in pairs of four and not two as previously with the line we had degrees of neighbors by pairs of two where in an array like topology we have pairs of 4.

We received our starting spaghetti random state as expected:



we noticed that the array like topology takes more iterations to see significant differences. And that the neighbors seem to pull each other and don't let the map expand across the data points, as seen in the image below: we believed this was because of the distribution of the points:



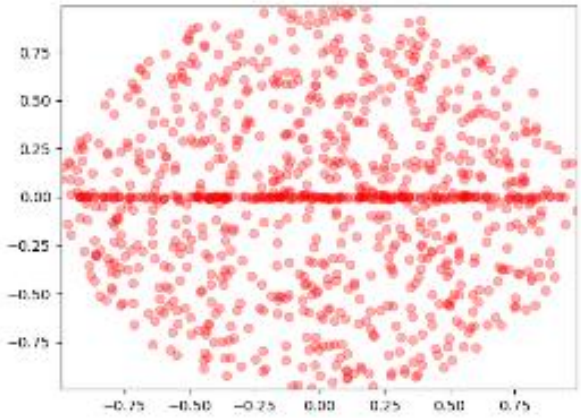
Q2:

. Do the same with at least **two non-uniform distributions** on the disk. (For example, when the likelihood of picking a point in the data set is proportional to the size of x, but uniform to the size of y. A second example could be where the likelihood of a point being chosen as a data point is proportional to the distance from the center of the disk. You can use other non-uniform distributions if you wish.)

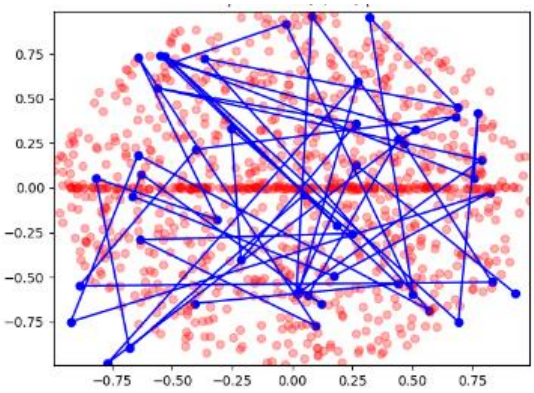
You should report with snapshots of the space as the Kohonen map evolves as the number of iterations grows.

Answer:

First we created our data set under the constraints {(x,y) I 0 <= x <=1, 0 <= y <=1} as shown in the image below with 1000 data points I also implied that 80 % of the data would be concentrated near the **y axis** as shown below:

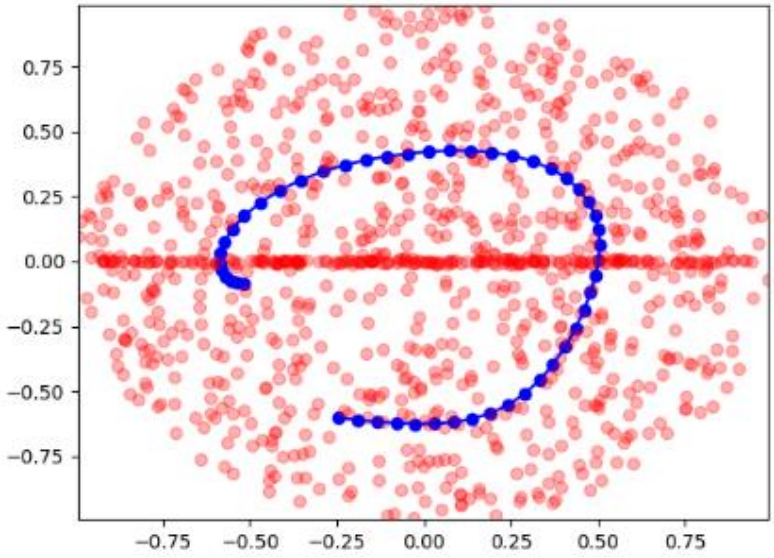


As mentioned before 100 clusters was not a great ratio to 1000 data points there for I used 50 clusters:

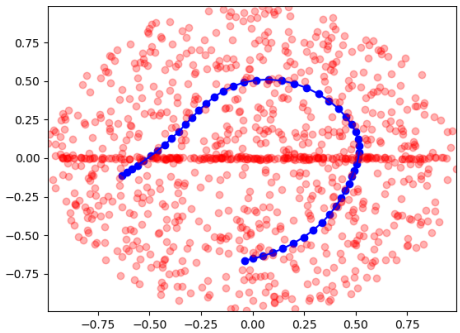


Above we can see our starting state where all clusters have been randomly maped by intislizing random weights.

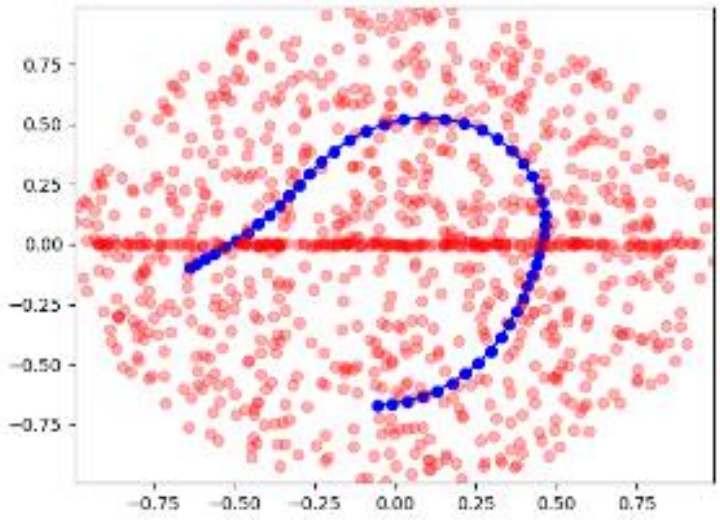
Below we can see what happends after 500 iterlations:



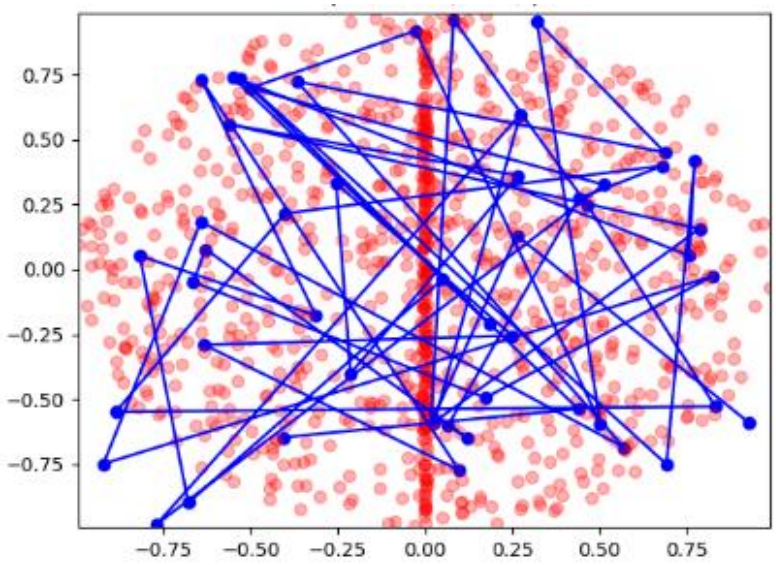
And after a few more thousand iterations we receive the following:

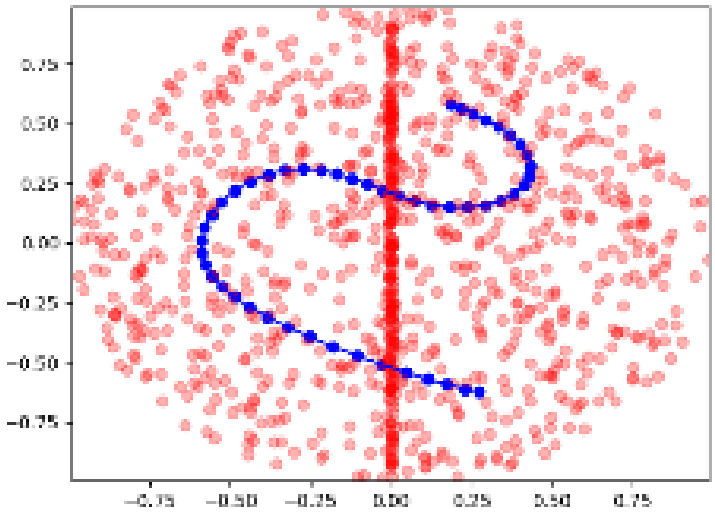


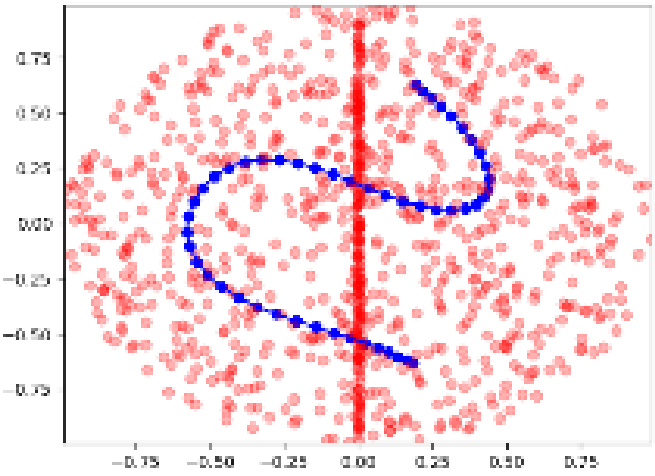
If we zoom in we can see what's interesting is the density of the map at the y = 0 base line, this is not surprising as our data was made artificially in that manner and we should expect the map to have a density bias towards where most of the data is clustered.

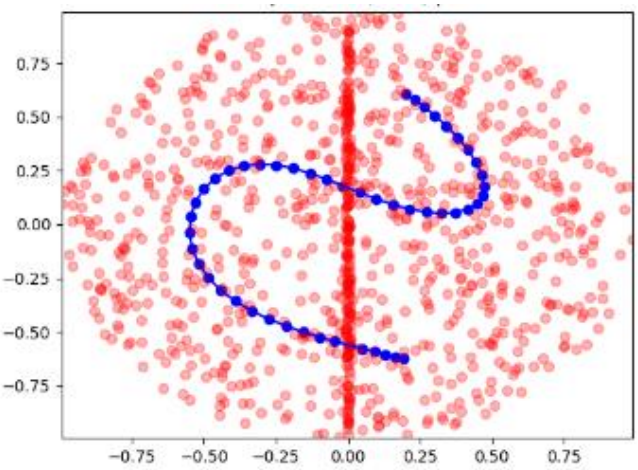


Then we created our data set under the constraints {(x,y) I 0 <= x <=1, 0 <= y <=1} as shown in the image below with 1000 data points I also implied that 80 % of the data would be concentrated nere the the **x axix** as shown below:



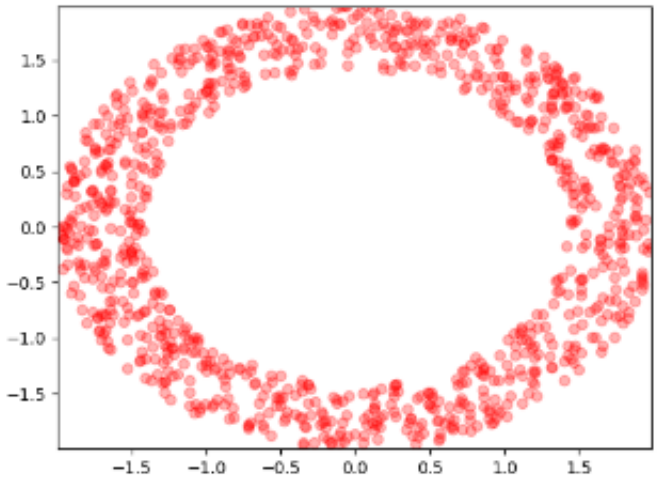

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75


0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75

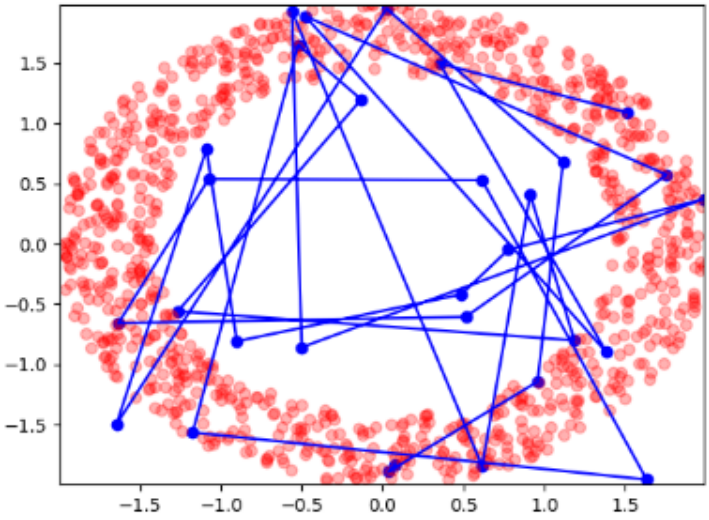

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75

Donut:

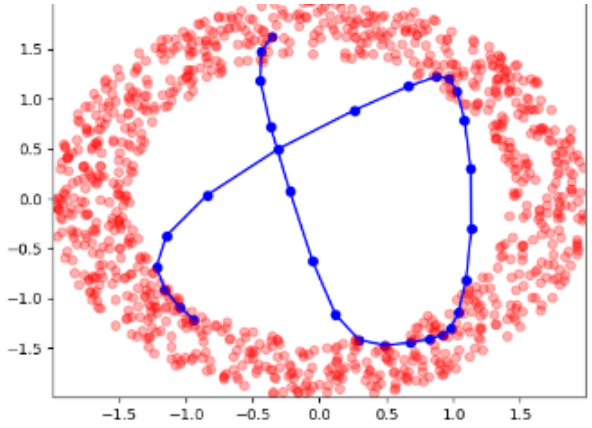
For the donut data set we generated the following data:



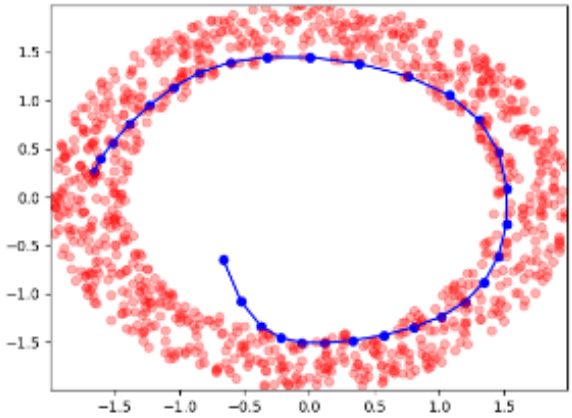
We again randomly initalsied weights for the SOM and recived our starting points:



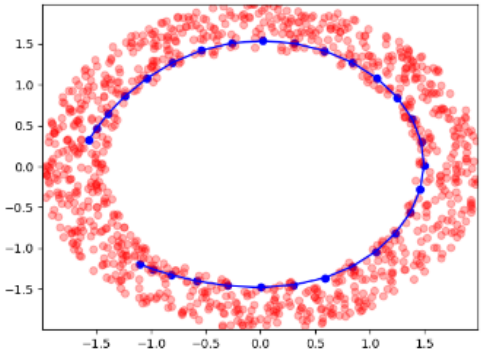
After our first few iterations the map started to unravel from randomless meaning weight values to a more map lie shape in regards to the donut:



After more iterations:



And finally:



Conclusions:

We concluded the uniform data the map would have no bias as seen above in the first example. Our next two examples had a bias towards the x and y axis accordingly with the non uniform data setes.

We concluded that although not all points were in the donut dataset that the map managed after a certain amount of iterations to get all points in the data set.

Codes:

Making disk shape data:

```
    data = np.empty((d_size, 2), dtype=object)
    random.seed(11)
    if condition == 1:
        n = 0
        while n < d_size:
            x = random.uniform(-2, 2)
            y = random.uniform(-2, 2)
            if x ** 2 + y ** 2 <= 1:
                data[n, 0] = x
                data[n, 1] = y
                n += 1
```

Making un- uniform disk data for y and x axis bias:

```
n = 0
while n < d_size:
    x = random.uniform(-2, 2)
    y = random.uniform(-2, 2)
    thresh = random.randint(0, 100)
    if x ** 2 + y ** 2 <= 1:
        if thresh < 80:
            data[n, 0] = x
            data[n, 1] = y
            n += 1
        else:
            data[n, 0] = x
            data[n, 1] = y / 100
            n += 1
```

Euclidean distance for finding min distance

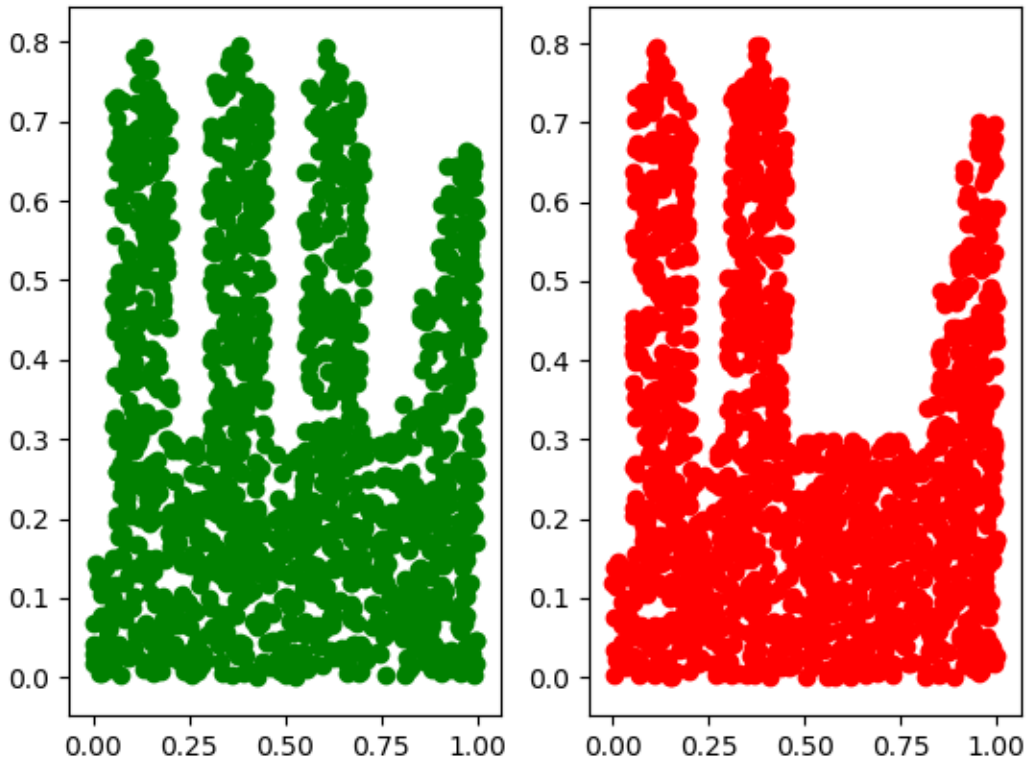
```
        min_neuron_dest = np.inf
        for x in range(self.shape[0]):
            for y in range(self.shape[1]):
                euclid_dist = np.linalg.norm(vector - self.map[x, y])
                if euclid_dist < min_neuron_dest:
                    min_neuron_dest = euclid_dist
                    ans = (x, y)
```

Updating map:

```
        for x in range(self.shape[0]):
            for y in range(self.shape[1]):
                dist_from_bmu = np.linalg.norm(np.array(bmu) - np.array([x, y]))
                alpha = self.alpha_start * np.exp(-t / 300)  # update alpha
                radius = np.exp(-np.power(dist_from_bmu, 2) / self.radius_strength)  # update radius
                self.map[(x, y)] += alpha * radius * (X_i - self.map[(x, y)])
```

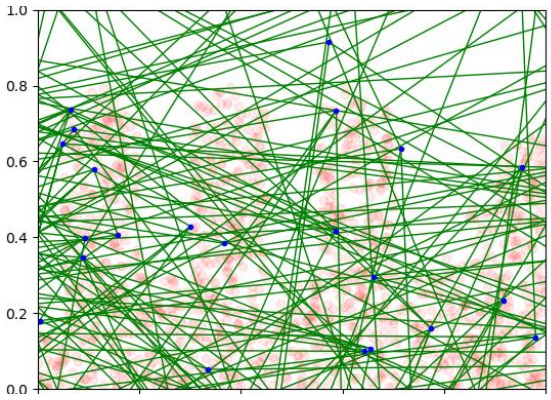
Monkey hand:

We first made an 25x25 array of zeros and manually inserted 1's in the array to "Draw" a hand with four and three fingers below were our results:

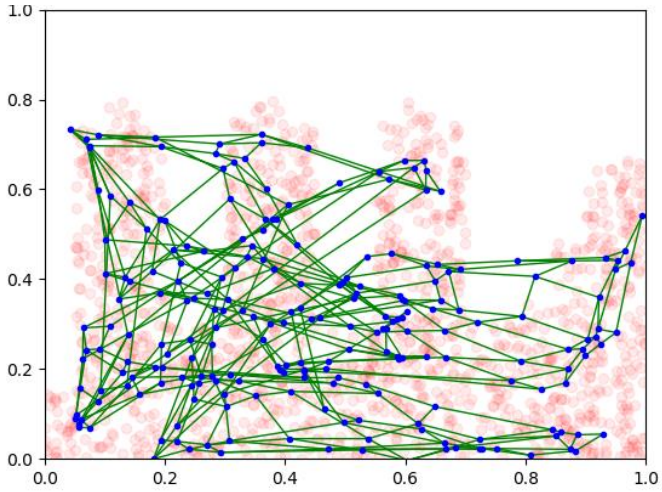


We then initialized Kohonen SOM algorithm with the hand with **four fingers** and received our starting random state:

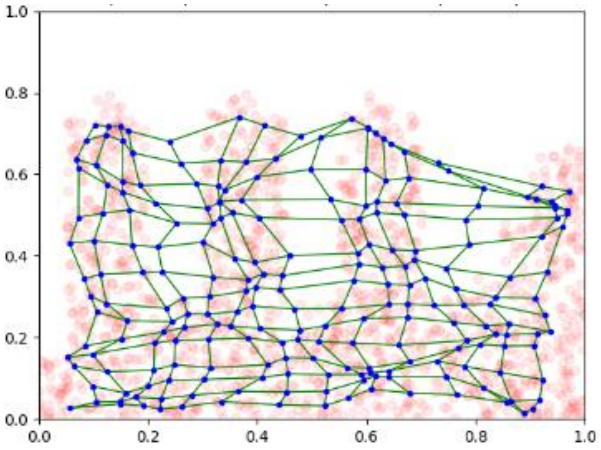
Spaghetti state



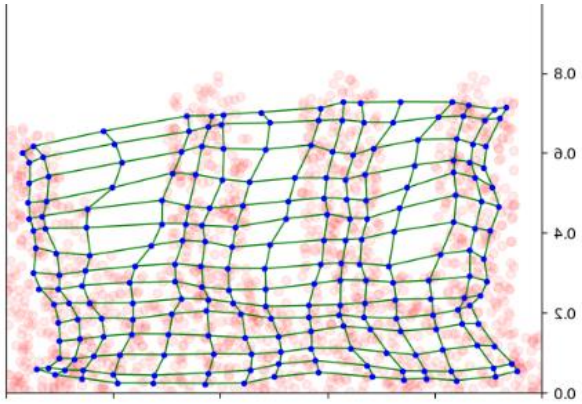
500 loops later our map is starting to look less random but still a bit of a mess:



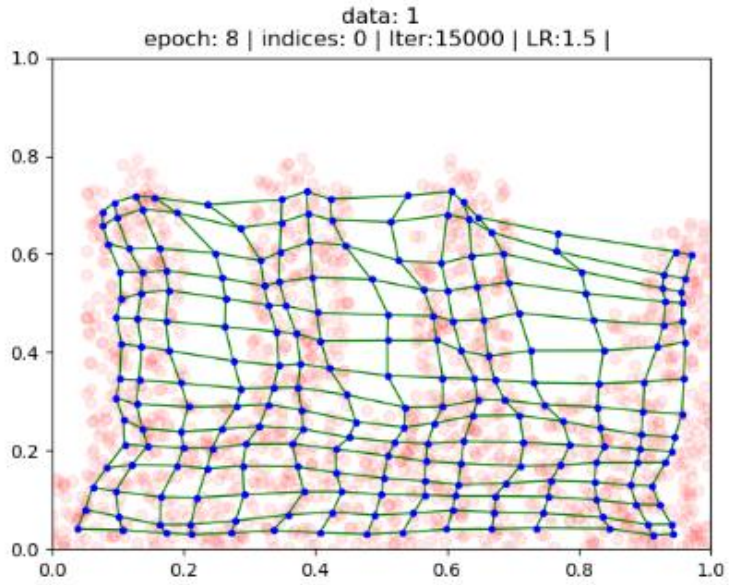
After 2000 we can clearly see the map as mapped out the hand and fingers clearly and is making much smaller adjustments moving forward:



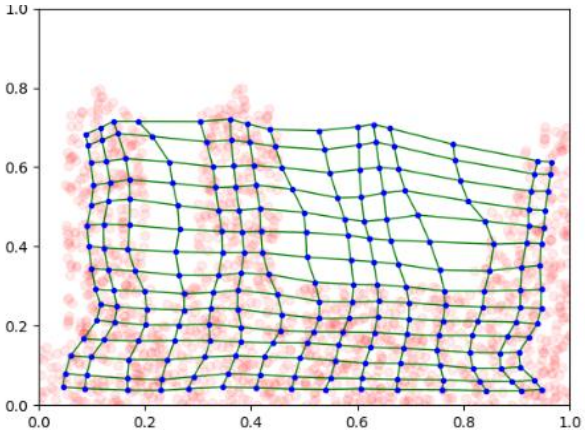
After 5000 iterations we can see the hand mapped and can differentiate between the fingers and space between the fingers.



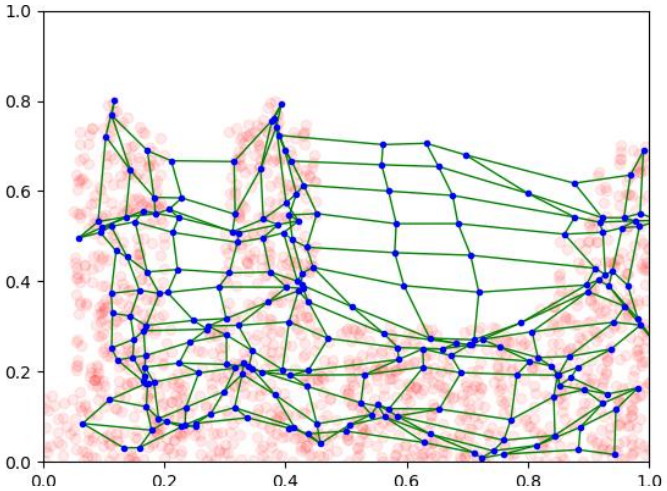
Finally after 6000 iterations we cut the monkeys third finger off (DON'T TRY THIS AT HOME!)



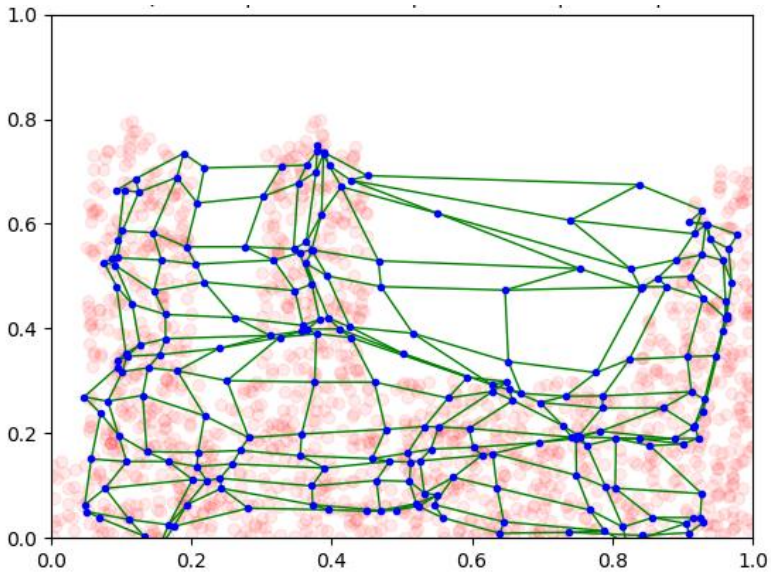
Our now three fingered monkey has the following starting point with the current SOM starting point:



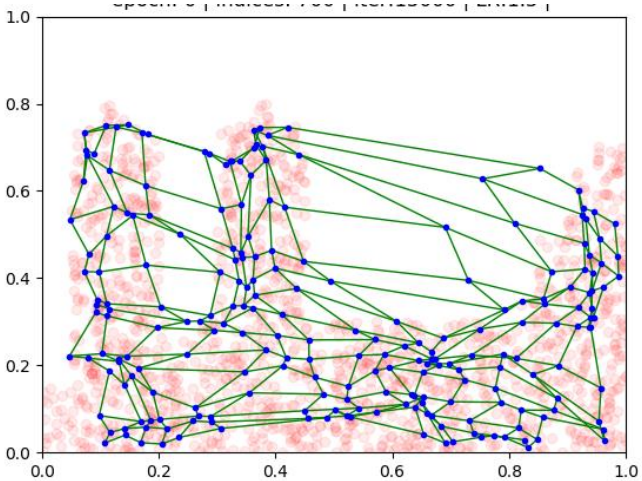
After 1500 loops we can see the map is starting to gravitate towards the other fingers and better fit the current monkeys hand with three fingers:



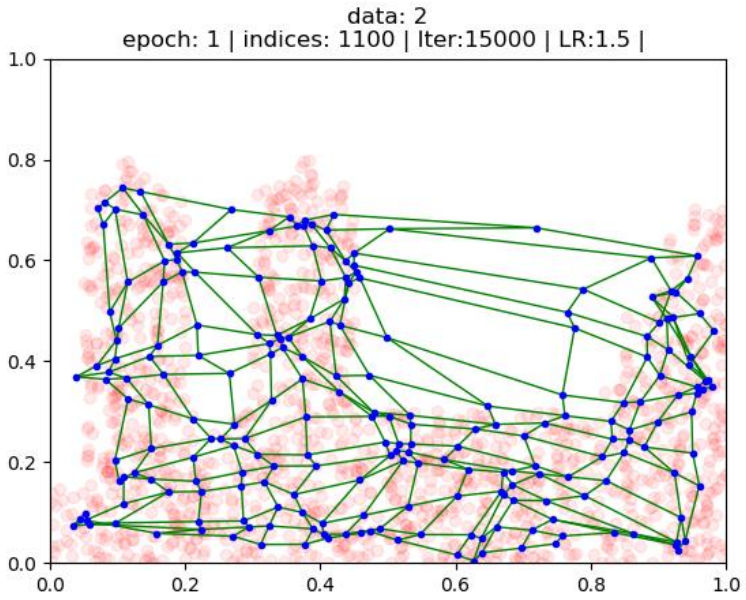
After 3000 loops:



After 4000 loops:



Finally after 5000 loops we can see the map has been re done to much better represent the current monkeys the fingered hand:



PLEASE NOTE: no actual monkeys were harmed in this experiment ☺

Conclusions and notes:

We came to the conclusion that we were not surprised that th SOM managed to map out the monkeys 4 fingered hand at the beginning. We were however surprised how well the map managed to rearrange itself suddenly after we “cut off” the monkeys finger.

We concluded had we started off with the data set of the monkey’s 3 fingers the map would look better , the reason being that we changed the amount of neighbors being changed hence once the map had a low radius of neighbors only then did we cut off the monkeys fingers and had we started with more neighbors we would have gotten similar results.

Below are some of our codes:

Codes :

monkey 4 fingers code:

```python
def getdata():
    """
    Make 2 random data sets at "monkey hand" shape using 20X20 matrix's.
    :return: data1 , data2
    """
    data1 = np.zeros((1500, 2))
    data2 = np.zeros((1500, 2))

    random.seed(11)
    Mask1 = 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, 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, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 1, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0],
        [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
        [0, 1, 1, 1, 1, 1, 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, 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, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    ])
```

monkey three finger code:

```python
    Mask2 = 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, 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, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 1, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0],
        [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
        [0, 1, 1, 1, 1, 1, 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, 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, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    ])
```