Problem 2

This “The Capacity of the Hopfield Associative Memory” paper starts by introducing the basic neural network. From the first part, I can get that the when we think of the neural networks with the state perspective, it is similar to the DFA, especially when the state transition occurs. And in this part it introduces the nn structure which is this paper based on, e.g. the neuron is bistable, dense connection, the connection matrix is also assumed to be symmetric with zero diagonal in almost all this paper, and the sign activation at each neuron with 0 threshold; Furthermore, this paper discusses two modes of changing states, synchronous and asynchronous(which is like the ECE657 slides example).

In the second part, this paper presents the outer product construction for the connection matrix based on given fundamental memory, and this paper uses this method to build connection matrix. Compared with the second part, the third part provides readers other methods of building weights matrix, which could lead to a higher capacity, which is the focus on this paper, but not as easy as the outer product construction method. The method mentioned in this part requires the eigenvector of the connection matrix with positive eigenvalues.

In the IV section, the author gives us an example of constructing the connection matrix, and how a probe vector converges to one of the fundamental memory. And after discussing the process of this probe vector converging, we also find a problem that the probe may not converge to the nearest memory, and the author also mentions that the probe vector also could not converge to one of the memories. With these consideration, we go into the V section, this section tells us the concept of associative memory. When you are given everything needed once, it cannot be considered as associative, and this section introduces the forced choice model in which the some components are not known, thus introducing the probability correctly recovered, which is used in the following part. In this section, we also know three ways of convergence, and prove the convergence in the energy perspective with asynchronous model.

Then we come to the VI part, which I think is the most important part for us to understand the paper. The author by choosing the fundamental memory randomly to explain the concept of asymptotic capacity. Only two of the concepts of the asymptotic capacity are based in this paper. The capacity is the rate of growth, and choose m = m(n) memories randomly, where n is the length of each fundamental memory. We are given fixed p, where p is between [0,1/2], and ask for the largest rate of growth m(n) as n -> ∞ so that we still can recover the fundamental memory within p*n of a prob. If we are allowed to fail with a small probability, then “recover” means with probability approaching 1 as n -> ∞. Otherwise, if the rate of growth is exceeded by any fraction 1+ E with E > 0, then instead of having what we want happen with probability approaching 1, it happens with probability approaching 0.

“Two cases are distinguished in this paper. First, with high probability, every one of the m fundamental memories may be fixed, almost its entire pn - sphere being directly attracted. Second, and this is a weaker concept, with high probability almost every memory is good, as above, but not necessarily every memory. It turns out that this weakening doubles capacity". This part I cite from the paper is both important for us to understand the paper but also a contribution discovery.

In the VII part, the author provides us the needed knowledge and lemmas need in part VIII for hard lemmas, and the two lemmas are the "large deviation" version of the central limit theorem and the quantitative form of truncated inclusion and exclusion needed to prove the Poisson distribution conjecture which will be mentioned in the VIII part.

VIII part gives us an important conclusion that "the number of ΣTijxi(a) sums which fail to be correct (with appropriate m,n) obeys a Poisson distribution. Here x(a) is one of the m fundamental or original memories used to construct the sum-of-outer products connection matrix Tij. "Correct" means that the sum equals xi(a)". This conjecture is also proved in this part. As above mentioned, there are two hard lemmas in this part. The second lemma derives an asymptotic independence result for row sum failures, needed to prove the Poisson result.

Given the lemmas in VII and VIII part, the paper also give a short proof for the Big theory in IV part. Based on the theory, the author derives a very important capacity results, that is " The theorem derives the capacity (corresponding to a coding theorem and its converse) when we want what amounts to immediate (one-step) convergence in the synchronous model, starting from any probe vector no more than pn away from a fundamental memory, 0 <= p <1/2. Two possible capacity definitions result in capacities differing. by a factor of two. The larger capacity is obtained when we are allowed to fail to converge for a small fraction (approaching 0 as the memory length n approaches ∞ of the m fundamental memories.". And this result is extended in part X.

At last, we get the following:

We have seen that the (asymptotic) capacity m of a Hopfield associative memory of length n when it is to be presented with a number m of random independent ± 1 probability $1/2$ fundamental memories to store and when probing with a probe n-tuple at most ρn away from a fundamental memory $(0 \le \rho < 1/2)$ is

1) $$\frac{(1-2\rho)^2}{2}\frac{n}{\log n}$$

if with high probability the unique fundamental memory is to be recovered by direct convergence to the fundamental memory, except for a vanishingly small fraction of the fundamental memories;

2) $$\frac{(1-2\rho)^2}{4}\frac{n}{\log n}$$

if, in the above scenario, *no* fundamental memory can be exceptional;

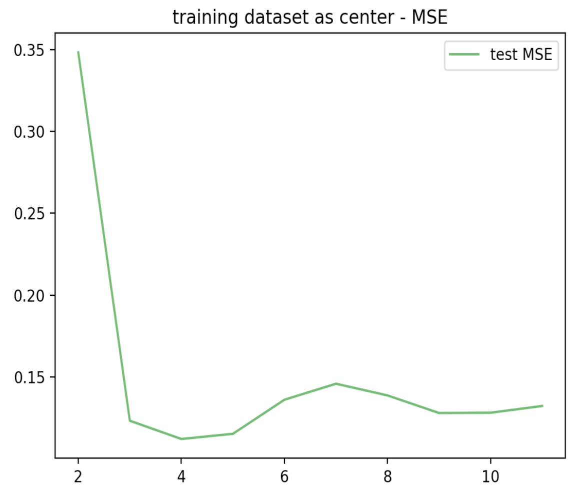
3) $$\frac{n}{2\log n}$$

if $0 \le \rho < 1/2$, ρ given, where some wrong moves are permitted (although two steps suffice), and we can have as above a small fraction of exceptional fundamental memories;

4) $$\frac{n}{4\log n}$$

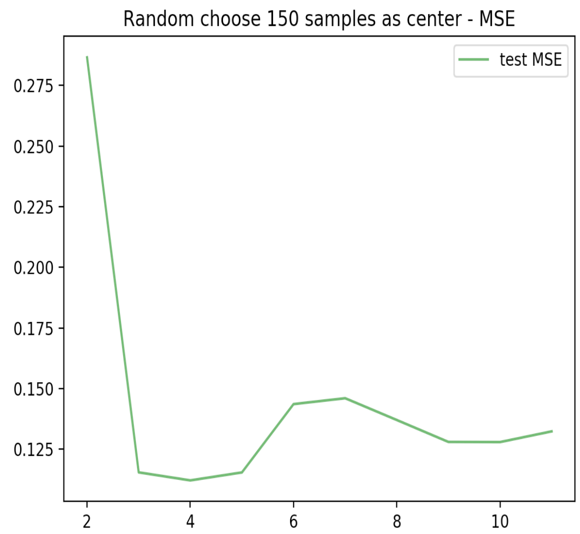
if as above some wrong moves are permitted (although two synchronous moves suffice) but no fundamental memory can be exceptional. [3) and 4) were not rigorously proven.]

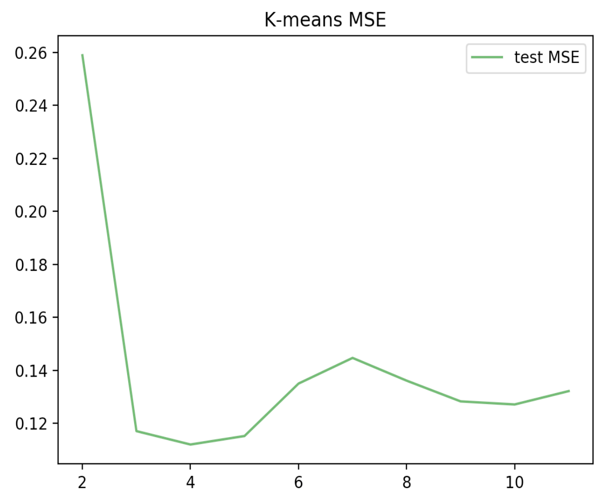
Problem 3

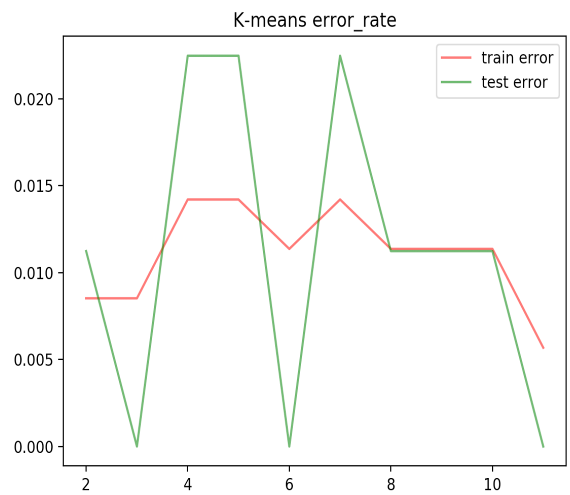


$$MSE(y, y') = \frac{\sum_{i=1}^{n} (y_i - y_i')^2}{n}$$

1) We can see that as I vary spared parameter from 2 to 12, the MSE experiences a rapidly decrease between sigma = 2 to sigma = 3, after that he MSE almost keeps stable. As for the error rate, we can see that at most time, the train error rate is lower than the test error rate, e.g. when sigma = 2, 4, 5, 7. And sometimes they are equal.





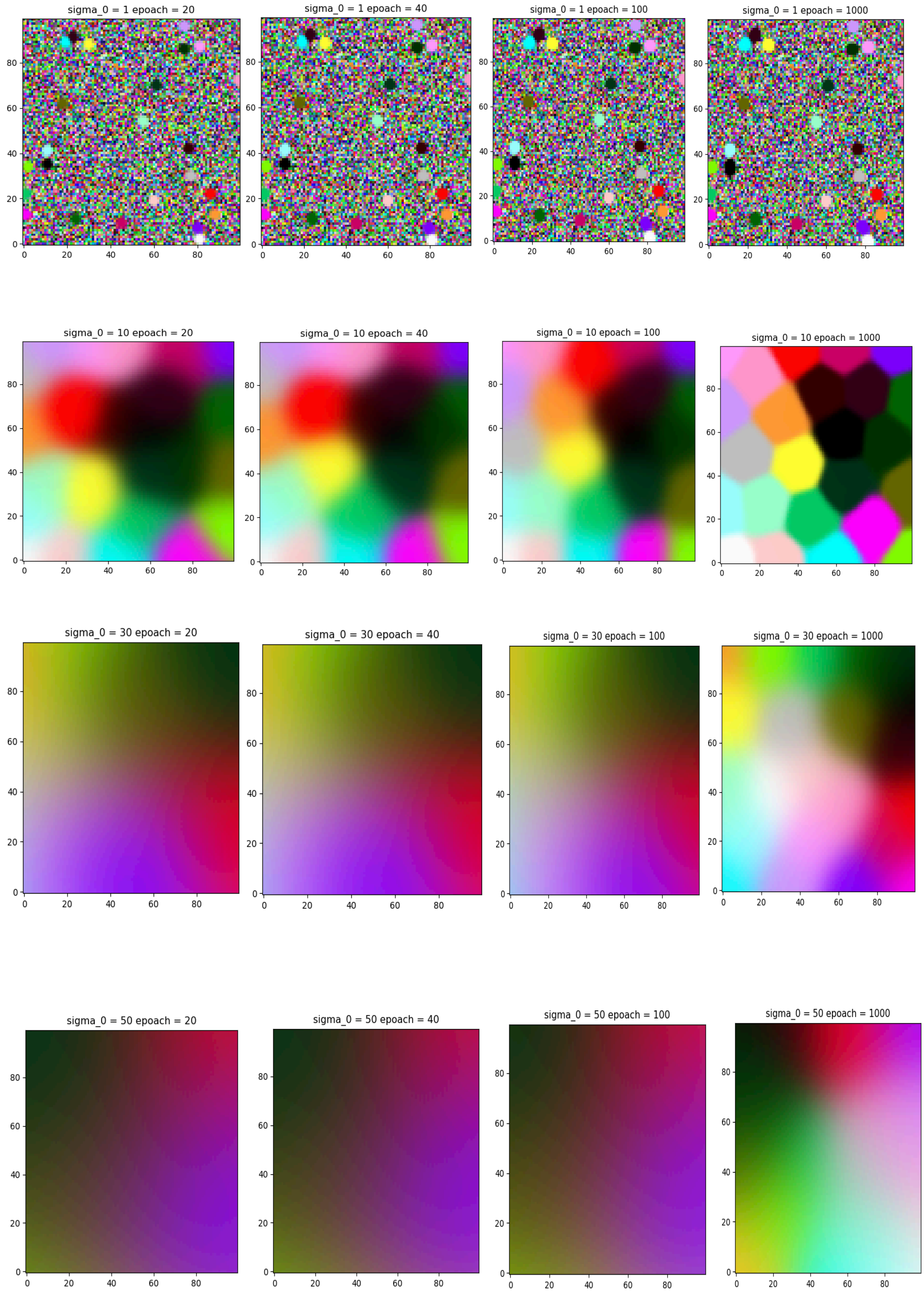


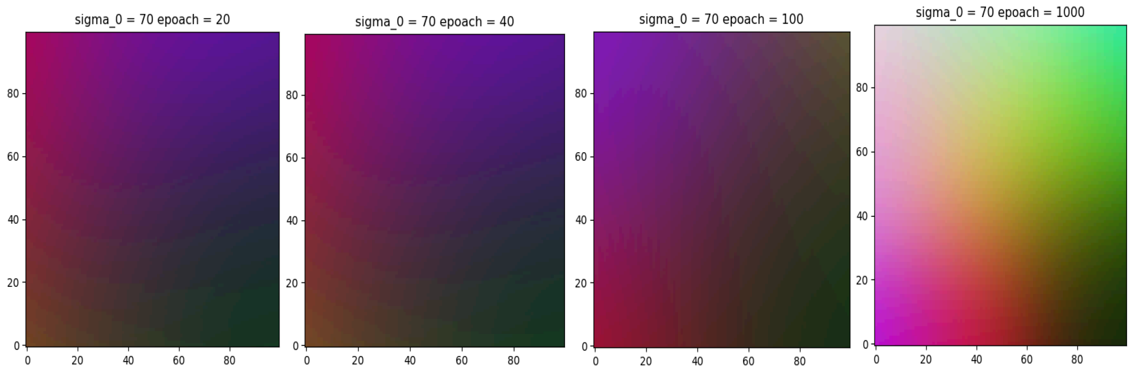
2) This is the MSE equation I use in my calculation. The results I generate based on different methods of getting center of RB function, i.e. use all the points in the training set, randomly select 150 points from input dataset, and using K-means algorithm to get 150 centers. From the results, we can see that all these methods get a decreasing trend with the increase of the spread parameter from 2 to 12, with step 1. And we can see that the K-means get the lowest MSE compared with the other two methods. The MSE change from around 0.26 to below 0.12. And the method that randomly select 150 points changes from 0.28 to below 0.125, while the first method starts from 0.35 and decrease to below 0.15.

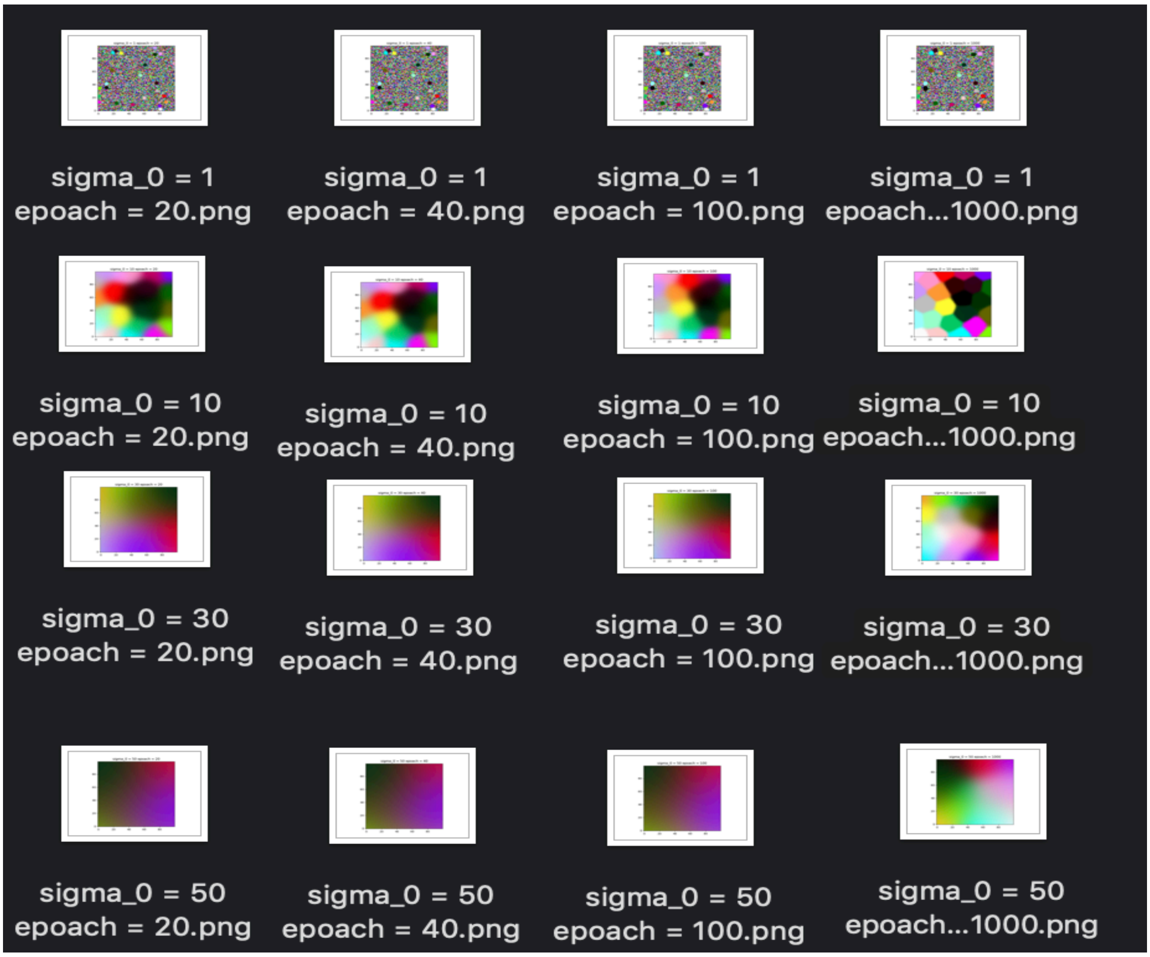
As for the error rate for these three methods, they almost have the same trend, except that when the spread parameter is 6, the last two methods have the test error rate at 0, while the first method test error rate is around 0.125. And we can see that the test error rate has a larger fluctuation compared with the train error rate. In other words, the train error rate is more stable than the test error rate.

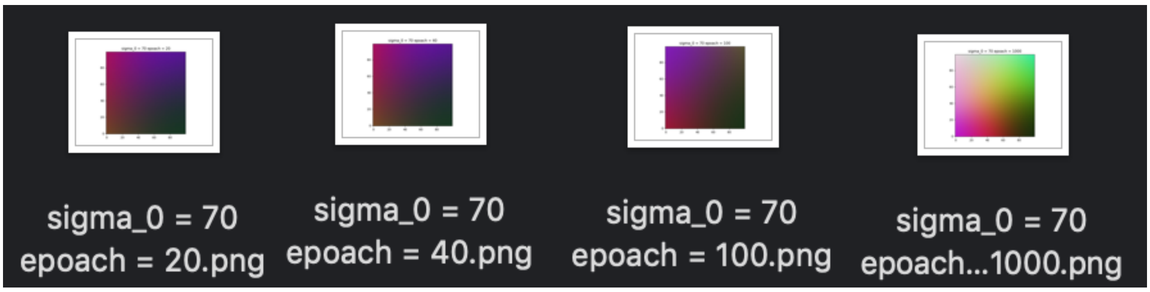
Problem 4

1)


sigma_0 = 1 epoach = 20
sigma_0 = 1 epoach = 40
sigma_0 = 1 epoach = 100
sigma_0 = 1 epoach = 1000
sigma_0 = 10 epoach = 20
sigma_0 = 10 epoach = 40
sigma_0 = 10 epoach = 100
sigma_0 = 10 epoach = 1000
sigma_0 = 30 epoach = 20
sigma_0 = 30 epoach = 40
sigma_0 = 30 epoach = 100
sigma_0 = 30 epoach = 1000
sigma_0 = 50 epoach = 20
sigma_0 = 50 epoach = 40
sigma_0 = 50 epoach = 100
sigma_0 = 50 epoach = 1000


sigma_0 = 70 epoach = 20
sigma_0 = 70 epoach = 40
sigma_0 = 70 epoach = 100
sigma_0 = 70 epoach = 1000

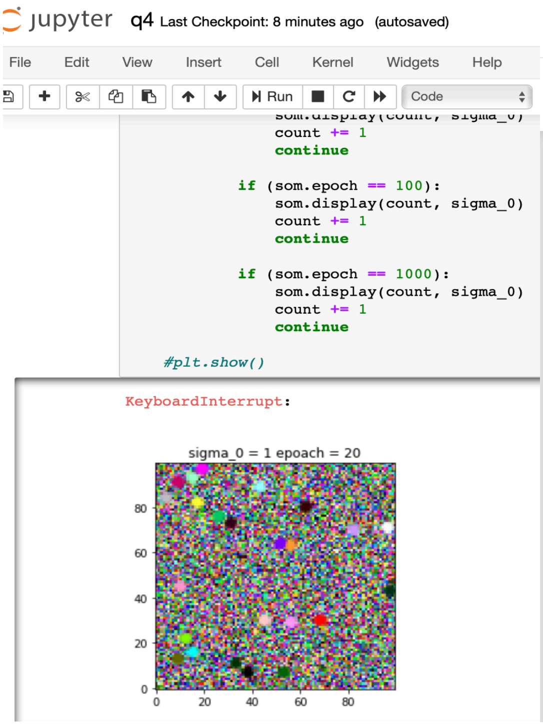

sigma_0 = 1 epoach = 20.png
sigma_0 = 1 epoach = 40.png
sigma_0 = 1 epoach = 100.png
sigma_0 = 1 epoach...1000.png
sigma_0 = 10 epoach = 20.png
sigma_0 = 10 epoach = 40.png
sigma_0 = 10 epoach = 100.png
sigma_0 = 10 epoach...1000.png
sigma_0 = 30 epoach = 20.png
sigma_0 = 30 epoach = 40.png
sigma_0 = 30 epoach = 100.png
sigma_0 = 30 epoach...1000.png
sigma_0 = 50 epoach = 20.png
sigma_0 = 50 epoach = 40.png
sigma_0 = 50 epoach = 100.png
sigma_0 = 50 epoach...1000.png

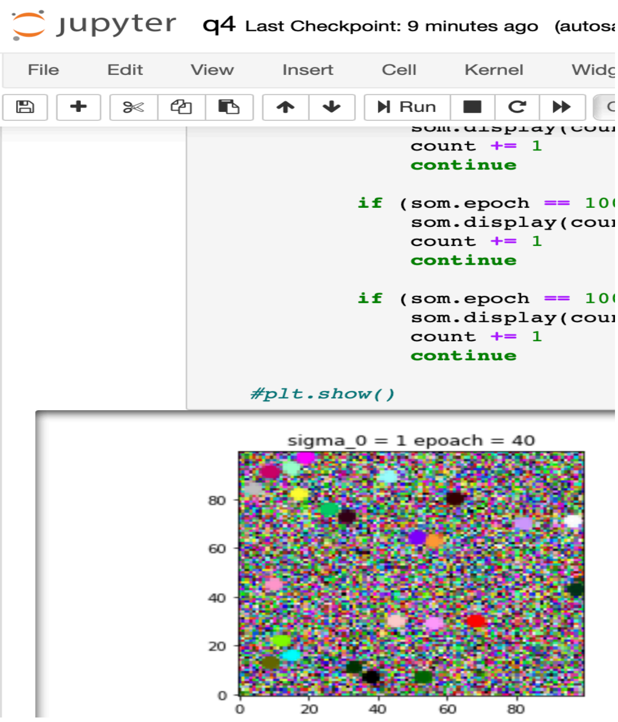

sigma_0 = 70 epoach = 20.png
sigma_0 = 70 epoach = 40.png
sigma_0 = 70 epoach = 100.png
sigma_0 = 70 epoach...1000.png

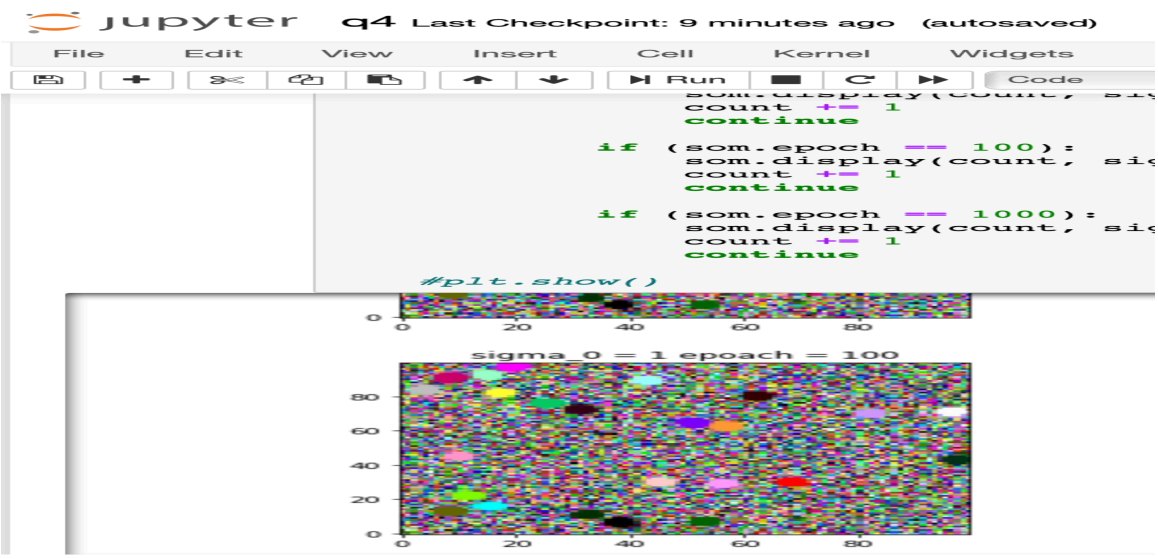
2)

From the output we can get that when the sigma_0 is smaller, the stronger the graininess in the picture. Furthermore, we can see that when the epoch increases, the boundary between different color areas is more obvious. And we can see that the results when sigma is equal to 10, the output result is best.

I give the example result for sigma_0 = 1, and epoch = 20, 40, 100 in the jupyter note book file.







```
sigma_0_list = [1, 10, 30, 50, 70]

#print(input_dataset[0])
epoch = 1000
for sigma_0 in sigma_0_list:
    som = Som(100, 24, sigma_0)
    count = 1
    for i in range(epoch):
        #print("epoach: ",i)
        for sample in input_dataset:
            som. train(sample[0], sample[1], sample[2
            #print(som.map[0][0].get_R())
            #print(sample[2])
        """
        if (som.epoch == 1):
            som.display(count, sigma_0)
            count += 1
            continue
        """


        if (som.epoch == 20):
            som.display(count, sigma_0)
            count += 1
            continue
```

If you want to change sigma_0 to run my code, you need to change the sigma_0_list, and use som.train for each sample input.