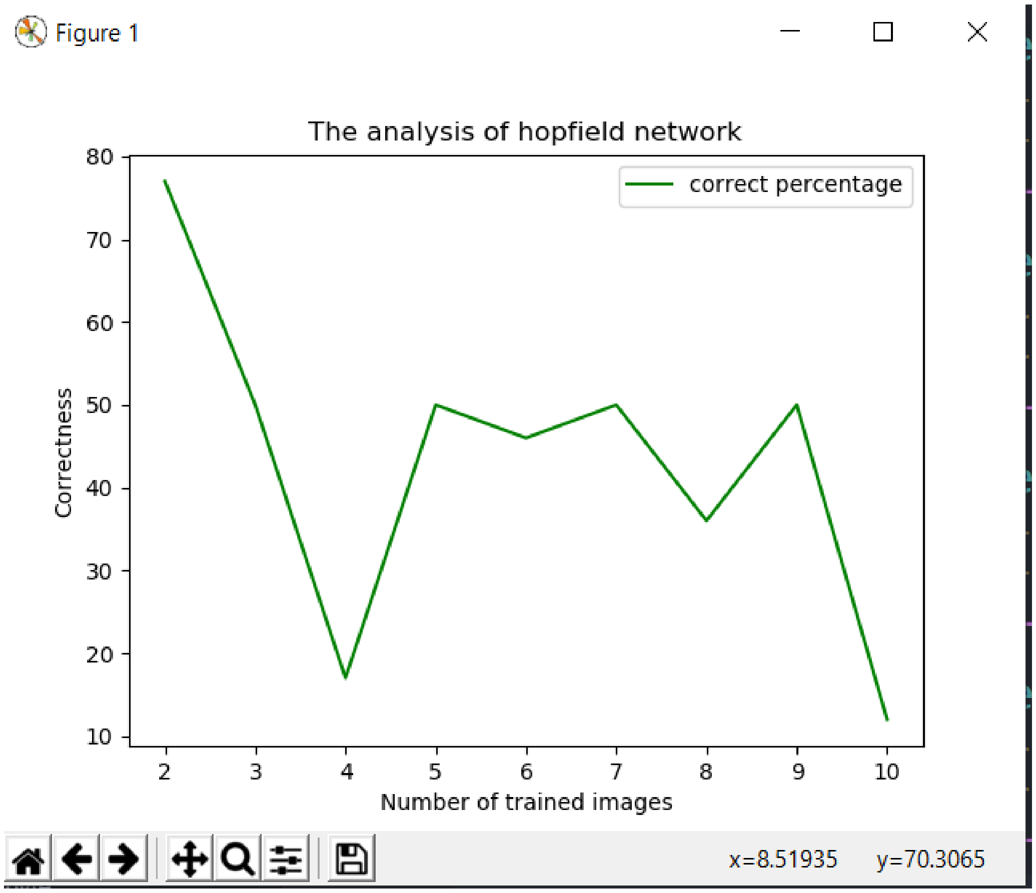
Question 1:

The program q1.py use scikit-learn utilities to load MNIST data, then the data is used to train in the Hopfield network so that I can classify the image of 1 and 5.

The sample for testing is 100 images and the training data starts from 2 to 10 images.



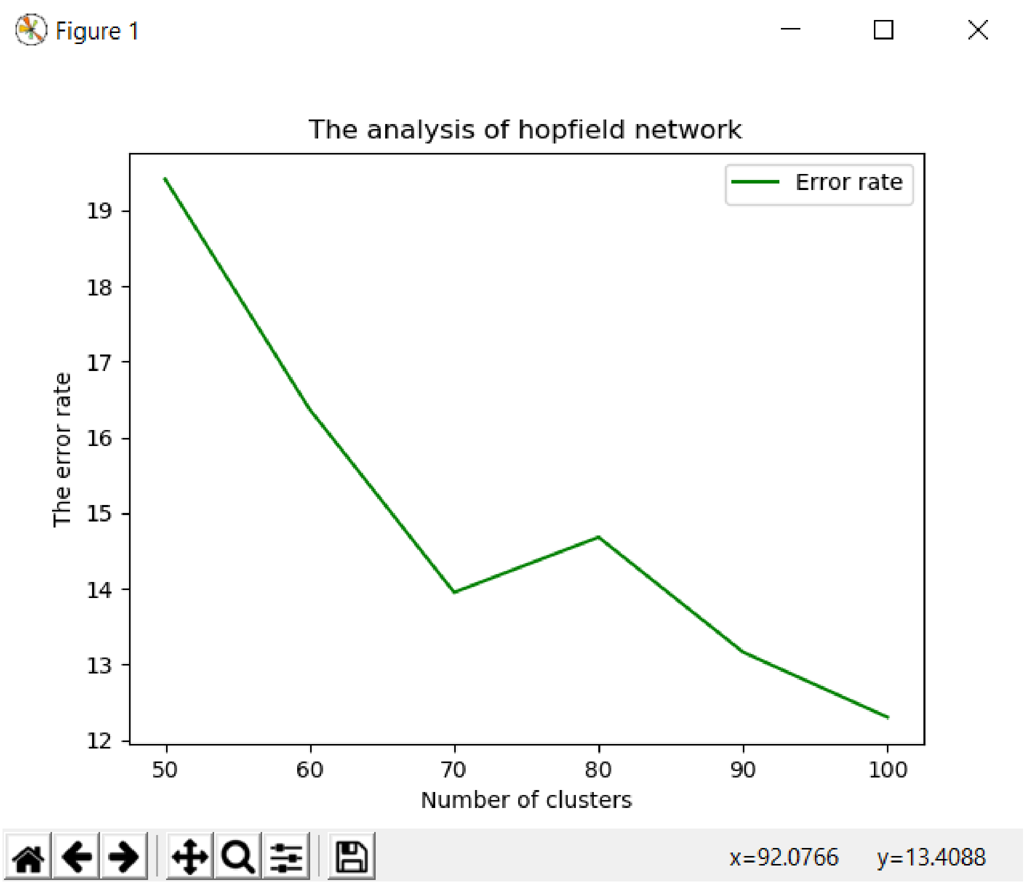
From the graph, we can see that with the train of 2 images, the correctness is up to 78%. However, when we train the Hopfield network with more than 2 images, the problem of unrecognition is popped up.

The experiment can also demonstrate the classification issue when the network is fitted with large amount of data.

Question 2:

a.

The q2.py program identify the number of hidden layers in the network by using K-means. Thankfully, the library sklearn provides a significant tools and commands to load data MNIST and efficiently implement kmeans. Here is the output of the first part:



```
Error rate 19.41

Error rate 16.37

Error rate 13.950000000000001

Error rate 14.680000000000001

Error rate 13.16

Error rate 12.3
```

With 50 clusters, the error is up to 19.41% but with 100 clusters the error rate is decrease down to 12.3%. That means as long as we increase the number of clusters the more accuracy the model will achieve. But for the efficiency goal, the appropriate number of clusters I would like to choose is 70 instead 100, because it provides an acceptable error rate which is around 14%.

b. c.

Once again, the scikit-learn also provides the model selection so that, K-fold cross correlation can be performed easily, the chosen value for k is 5. In details, the chosen activation for the RBF neural network is gaussian functions with the radius (sigma) is calculated as presented in the lecture slide:
First the beta value is calculated as:

RBF Beta Values

$$\sigma = \frac{1}{m}\sum_{i=1}^{m} \|x_i - \mu\|$$

If you use k-means clustering to select your prototypes, then one simple method for specifying the beta coefficients is to set sigma equal to the average distance between all points in the cluster and the cluster center.

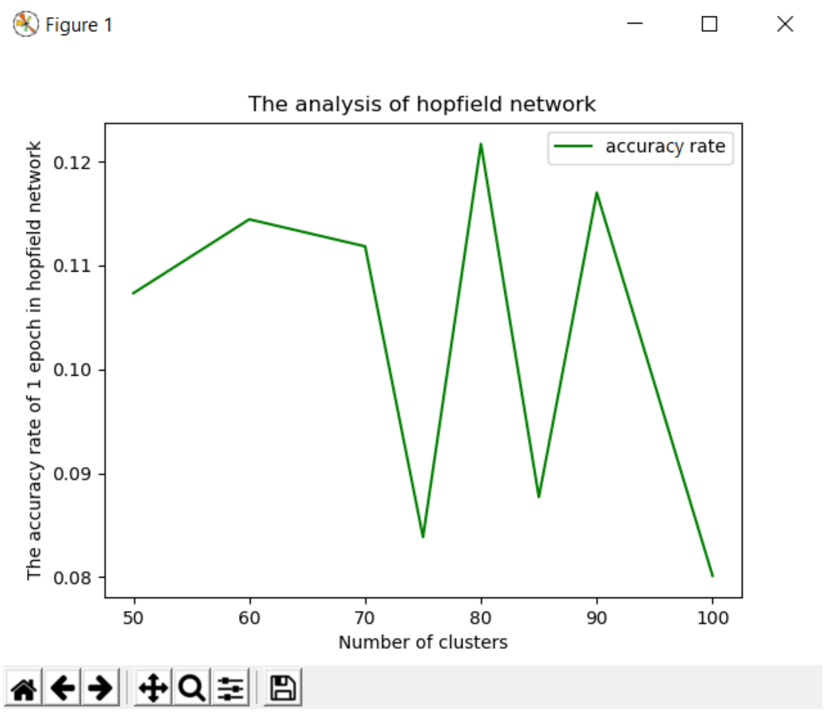
$$\beta = \frac{1}{2\sigma^2}$$

Then, we have:

RBF Activation Function

$$\varphi(x) = e^{-\beta\|x-\mu\|^2}$$

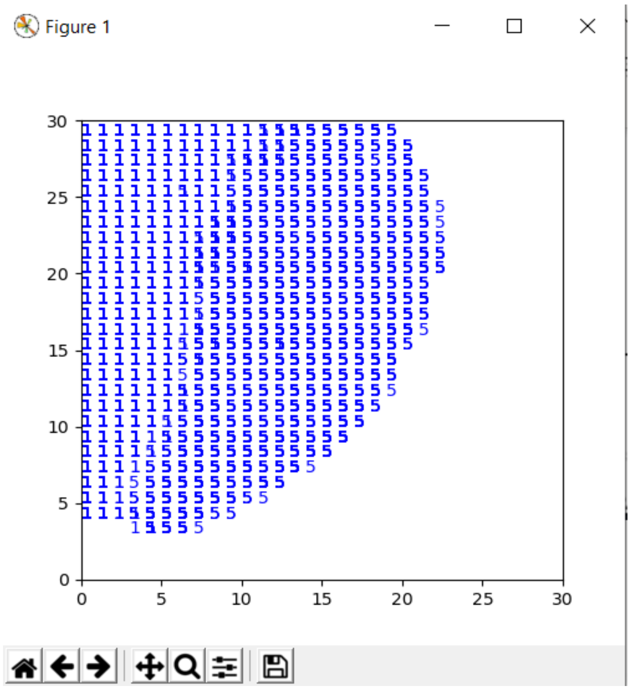
The learning rate is 0.05, with simple gradient descent.
Here is the result with 1 epoch of 5-fold cross correlation, each point is the mean of K fold cross correlation.



The 3 best accuracy is 70, 80 and 90. The best accuracy in the experiment is 80 clusters.

Question 3:

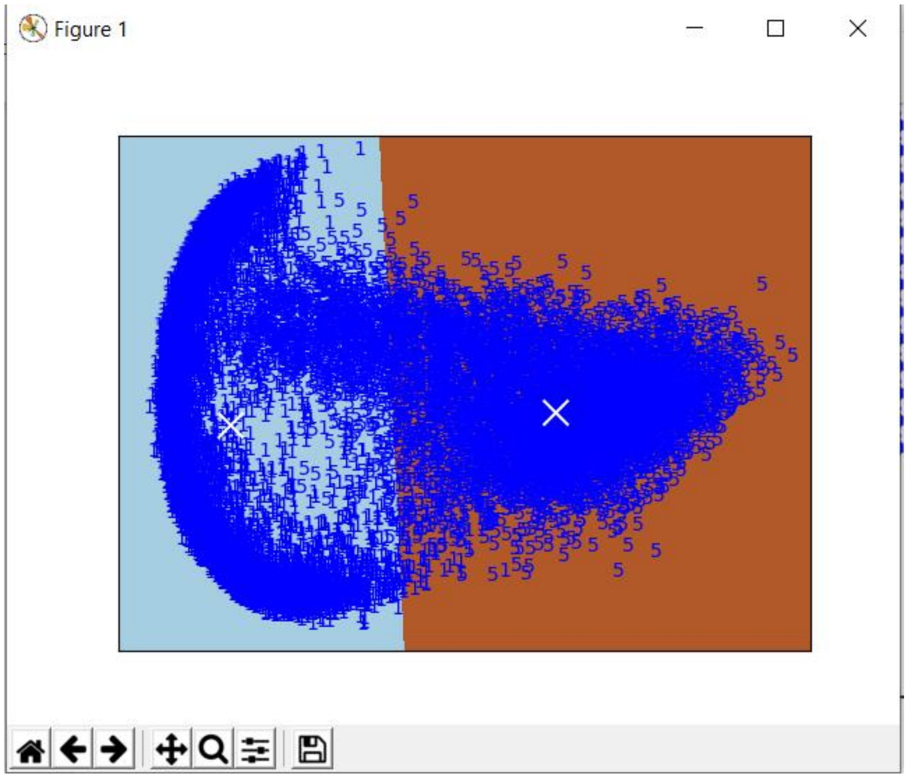
Similar to question 1, data is loaded by scikit learn. Thanks to minisom library, self-organizing map can be easily perform. In the experiment, the map has the size of 30 x 30 and the radius is 2 with the learning rate of 0.1 The data is trained with 4999 iterations, here is the result when SOM is trained:



For display purposes, instead of using SVD to display the solution of K-means, I would like to use PCA instead. Since there is a useful source helps to produce the result of K-means easier, the source can be found here:

https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html

Here is the result:



Question4:

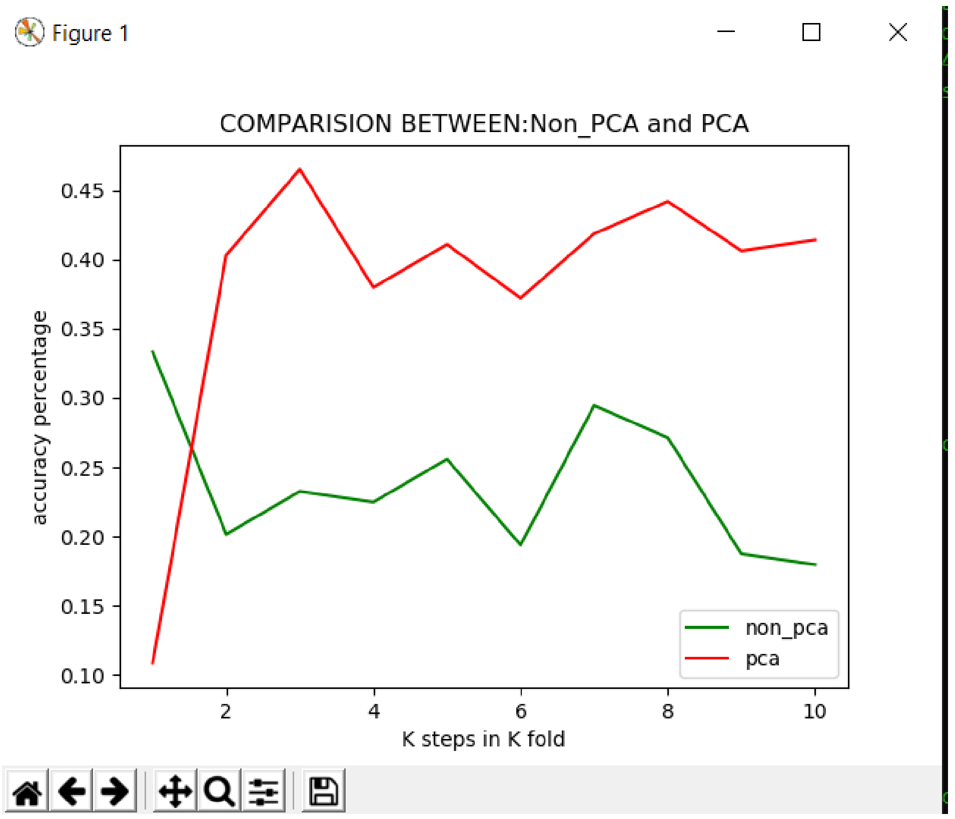
The program uses skicit learn to load data and convert data into the form, which is similar to MNIST data.

There are 2 methods be used in the experiment, that is K fold and PCA on the multiple layer perceptron networks. In details, there will be 2 layers which size are respectively 625, 300 (similar to the mlp.py standard). The activation function in the hidden layer is sigmoid and the activation function in the output layer is softmax.

The chosen value of k in the experiment is 10 and the components for PCA is 75. To implement PCA, the program uses scikit learn library and call the same function as question 3 to convert data.

For both cases, the number of epochs is 100 with the batch size 10, and the learning rate is 0.01. However, the standard deviation for raw data is higher than orthonormal basis because of the difference in dimensionality of both data.

Here is the result:



From the experiment, we see that the implementation of PCA is much better than non_pca, along with that, the running time of PCA is also faster than the running time of non_pca. As the result, we can conclude that for the given problem, PCA is a very useful and strong technique to help efficiently solve the problem.

Here is the result of the experiment:

0.3333333333333333

0.20155038759689922

0.23255813953488372

0.2248062015503876

0.2558139534883721

0.1937984496124031

0.29457364341085274

0.2713178294573643

```
0.1875

0.1796875

Average accuracy of non_pca 0.23749394379844962

USING PCA

0.10852713178294573

0.40310077519379844

0.46511627906976744

0.3798449612403101

0.4108527131782946

0.37209302325581395

0.4186046511627907

0.4418604651162791

0.40625

0.4140625

Average accuracy of non_pca 0.38203125
```