ME 5404 Neuron Networks Homework #3

A0263252L Shen Xiaoting

Q1:

a) RBFN for function approximation

We have the function of

$$y = 1.2sin(\pi x) - cos(2.4\pi x), for\ x \in [-1,1]$$

The training samples range from [-1,1] with a uniform step length of 0.05 containing 41 points and testing samples range from [-1,1] with a uniform step length of 0.01. The outputs of training samples have random noise. RBF function is Gaussian function with standard deviation $\sigma = 0.1$

$$\varphi_i(x) = e^{-\frac{\|x-x_i\|^2}{0.02}}$$

Then we calculate the exact interpolation matrix ϕ with $\varphi_i(x)$ and get the result of weight vector by $w = \phi^{-1}d$. We use the weight vector calculated above to get the prediction of test samples and plot with the green line in figure 1.1. The average error of RBFN method is 0.1662. It is obvious that the exact interpolation network is over-fitting. The network's outputs pass through all the data points. The training data is noisy and it is generally a high oscillatory function which will not provide good generalization.

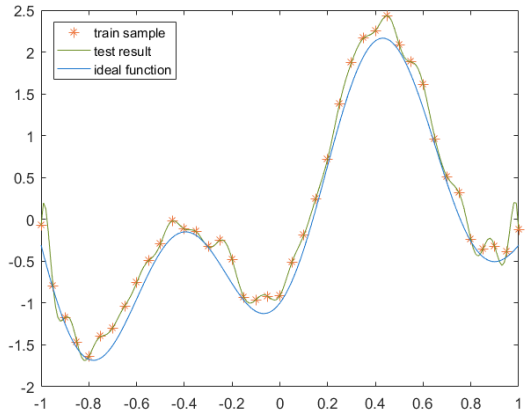


Figure 1.1 RBFN for function approximation

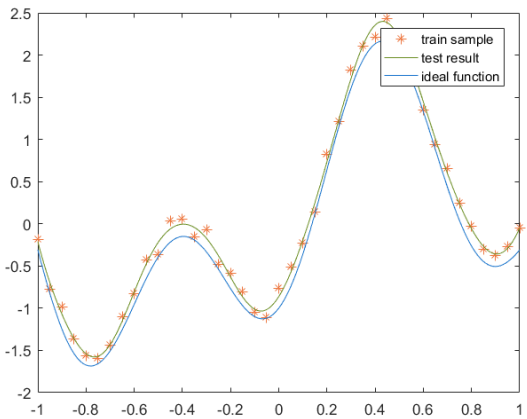


Figure 1.2 RBFN with 15 centers among sampling points

b) Fixed Centers Selected at Random for setting RBF parameters

We use the strategy of "Fixed centers selected at random" to determine the weights of RBFN, randomly selecting 15 centers among sampling points. We can get the maximum distance between the chosen centers and calculate the spreads which ensures that individual

$$\sigma_i = \frac{d_{max}}{\sqrt{2 * 15}}$$

The radial basis function centered at $\{\mu_i\}$ defined by

$$\varphi_i(x) = e^{-\frac{15}{{d_{max}}^2}\|x-\mu_i\|^2}$$

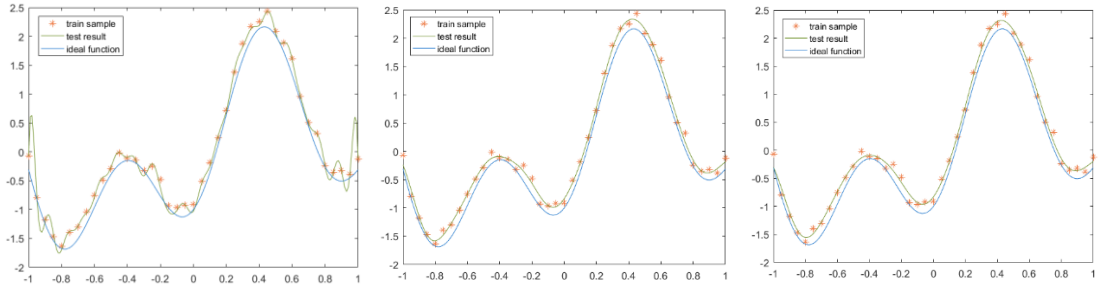
Then we calculate the exact interpolation matrix ϕ with $\varphi_i(x)$ and get the result of weight vector by $w = \phi^{-1}d$. The matrix ϕ is not a square matrix and we generalize the inverse for non-

square matrices with pseudoinverse. We use the weight vector calculated above to get the prediction of test samples and plot with the green line in figure 1.2. The result is much smoother and less oscillation than the original one and we calculate the average error of RBFN method with 15 centers is 0.1303 which is more accurate.

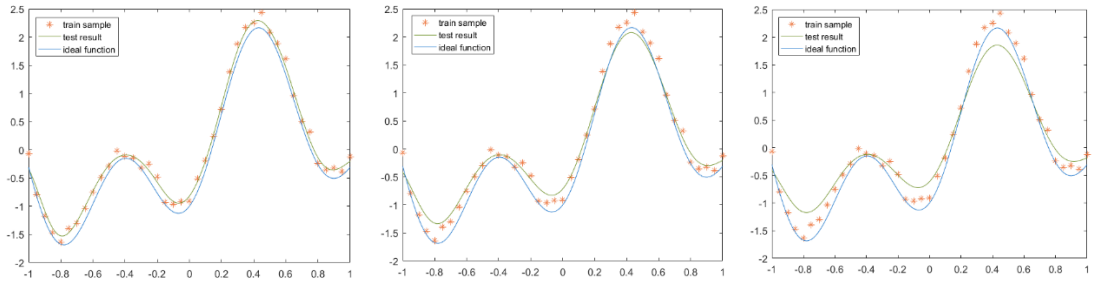
c) Regularization method

We use the same centers and widths as determined in part a. We calculate the weight vector by the equation of $w = (\phi^T\phi + \lambda I)^{-1}\phi^T d$. We regulate the parameter λ and get the regularized RBF networks as shown in figure 3.

If $\lambda = 0$, the network is unconstrained and not smoothness. When the parameter is large enough, the error will larger and the λ controls the balance between a smooth mapping and fitting the data points exactly. Through the tuning of parameters, λ between 0.1 to 0.5 will have small error and the details are shown in figure 1.3.



$\lambda = 0\,, error = 0.1808$ $\lambda = 0.1\,, error = 0.1495$ $\lambda = 0.3\,, error = 0.1497$



$\lambda = 0.5\,, error = 0.1500$ $\lambda = 2.5\,, error = 0.1698$ $\lambda = 5\,, error = 0.2394$

Figure 1.3. RBFN for function approximation with regularization method

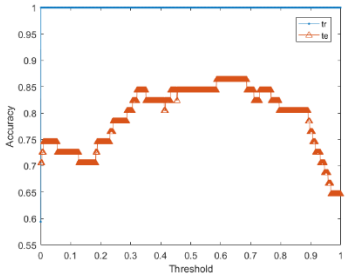
Q2:

a) RBFN for handwritten digits classification

We implement the exact interpolation method and apply regularization for digits classification. According to the matric number, we should classify the number 5 and 2. We set the label information of 5 to 0 and 2 to 1. There are 197 training samples and 51 testing samples. RBFN is Gaussian function with standard deviation $\sigma = 100$. We calculate the weight vector by the equation of $w = (\phi^T\phi + \lambda I)^{-1}\phi^T d$ and get the training and testing accuracy via threshold.

The first picture of figure 4 is the training and testing performance without regularization. The training accuracy is always 100% and testing accuracy is 86% (with the proper threshold). It is clearly that the result is over-fitting. Then we do regularization with different parameter λ and get

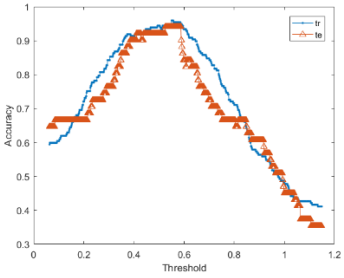
the results as the other 5 pictures in figure 2.1. We can see that the training accuracy and testing accuracy will be relative high with proper regulation with $\lambda = 0.01$, training accuracy of 96% and testing accuracy of 94%. If the parameter $\lambda\ too\ big$, the training and testing accuracy would be worse which is clear in the last picture in figure 2.1.



$\lambda = 0$

$train\_acc(max) = 1$

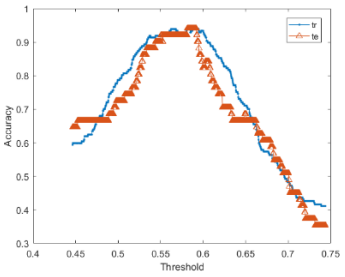
$test\_acc(max) = 0.86$



$\lambda = 0.01$

$train\_acc(max) = 0.96$

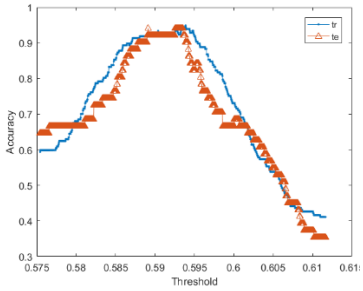
$test\_acc(max) = 0.94$



$\lambda = 0.1$

$train\_acc(max) = 0.95$

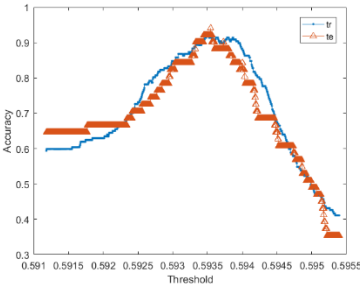
$test\_acc(max) = 0.94$



$\lambda = 1$

$train\_acc(max) = 0.95$

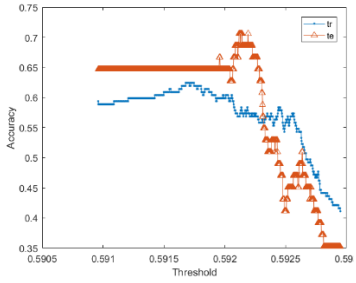
$test\_acc(max) = 0.94$



$\lambda = 10$

$train\_acc(max) = 0.92$

$test\_acc(max) = 0.94$



$\lambda = 100$

$train\_acc(max) = 0.62$

$test\_acc(max) = 0.71$

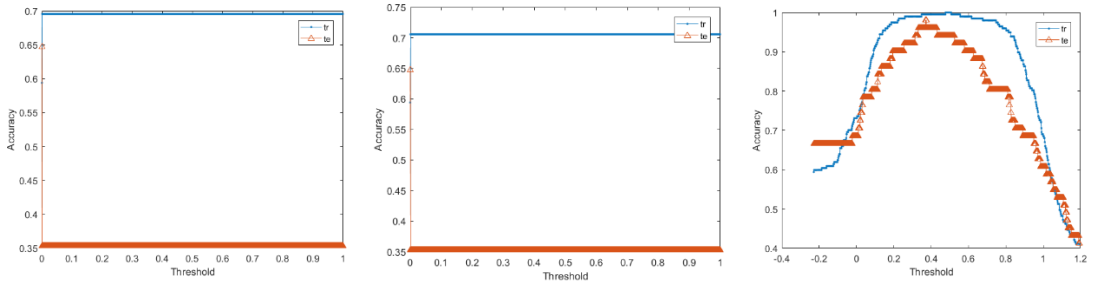
Figure 2.1 Training and testing accuracy via threshold with and without the regularization

b) RBFN performance with width regularization

We randomly choose 100 centers among the training samples and fix the width 100 same as a) and get the result of picture 4 in figure 2.2. The training accuracy is not always exactly 100% and testing accuracy is around 90% with the proper threshold. The testing accuracy is higher than a) and the over-fitting problem will be solved to some degree.

Then we change the width form 0.1 to 10000 and get the results as follows. They indicate that if the width is too small ($\sigma = 0.1$ or $\sigma = 1$), the training and testing accuracy will be low. If the standard deviation is properly set, the training and testing accuracy will be both high like $\sigma = 10$. It has a wide range that the training accuracy is close to 100% and maximum testing accuracy is 98%.

If the width is too large or too small, it can't achieve the best balance between the training accuracy and testing accuracy. For this handwritten classification problem, we can adapt the width to 10 and get the best performance.



$\sigma = 0.1$

$train\_acc = 0.69$

$test\_acc = 0.35$

$\sigma = 1$

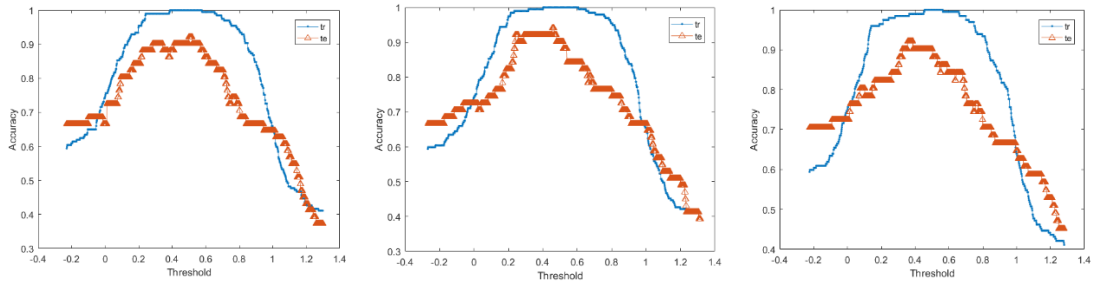
$train\_acc = 0.706$

$test\_acc = 0.35$

$\sigma = 10$

$train\_acc(max) = 1$

$test\_acc(max) = 0.98$



$\sigma = 100$

$train\_acc(max) = 1$

$test\_acc(max) = 0.92$

$\sigma = 1000$

$train\_acc(max) = 1$

$test\_acc(max) = 0.94$

$\sigma = 10000$

$train\_acc(max) = 1$

$test\_acc(max) = 0.92$

Figure 2.2 Training and testing accuracy via threshold with width regularization

c) K-Mean Clustering

We assume the number of the centers for RBFN is 2 and choose random values between 0 to 1 for the initial centers and form a matrix with the dimension of 784*2. Then we calculate the distance from the training samples and the center. We choose the center which is closest to the sample and assign the vector to the cluster which is represented by that center. We can update the centers by the means of the samples for each cluster until the cluster not change.

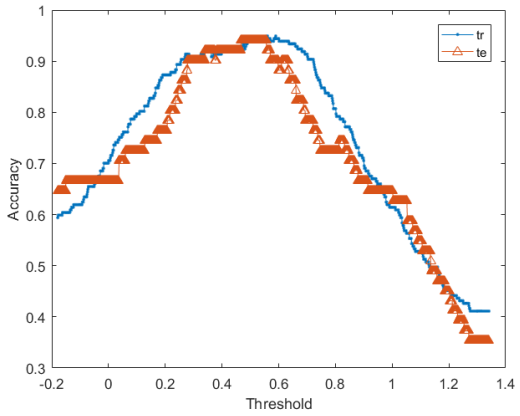
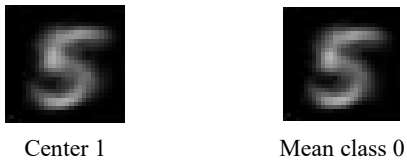


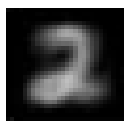
Figure 2.3 Training and testing accuracy via threshold with 2 centers of K Means clustering algorithm



Center 1

Mean class 0

Center 2



Mean class 1

Figure 2.4 Visualization of obtained centers and mean of training images

After the implementation of the K-means clustering algorithm, we get the training and testing

accuracy shown in figure 2.3. With proper thresholding setting, we can get the training accuracy to 94.9% and testing accuracy to be 94.1%. The process only needs 8 iterations to converge. The speed is fast and high efficiency. Then we do centers visualization and get the images in figure 2.4. We also calculate the mean of training images of two classes and get the images in figure 2.4. It is clear that the images of center and images of mean of training images are very close which can do high accuracy classification with K means algorithm.

Q3:

a) SOM implementation with 1-dimensional output layer of 25 neurons

We implement the SOM algorithm with the training points of "heart curve". First, we randomly initialize the weight vector with the dimension of 2*25 due to the dimension of training set is 2 and the number of neurons is 25. Then we choose an input vector from the training set and calculate the Euclidean distance between the training set and the weight vector. We select the index of minimum distance in the weight vector as the winner neuron. Then we update all weight vectors of all neurons in the neighborhood of the winning neuron with the equation

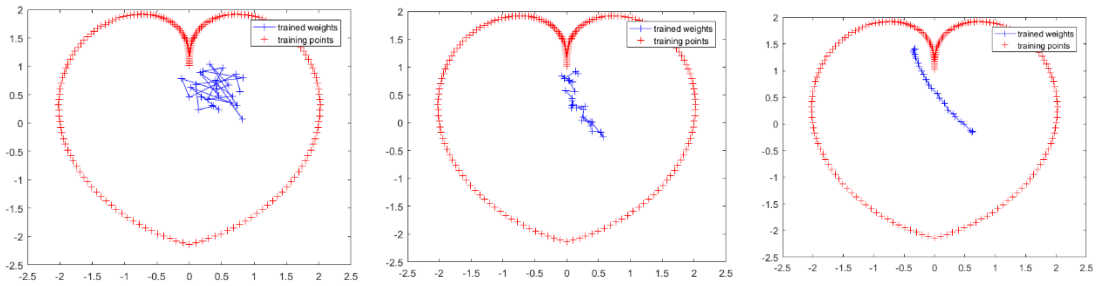
$$w_j(n+1) = w_j(n) + \eta(n) h_{j,i(x)}(n)(x - w_j(n))$$

In the update equation, we can get the learning rate $\eta(n)$ and time varying neighborhood function $h_{j,i}(n)$

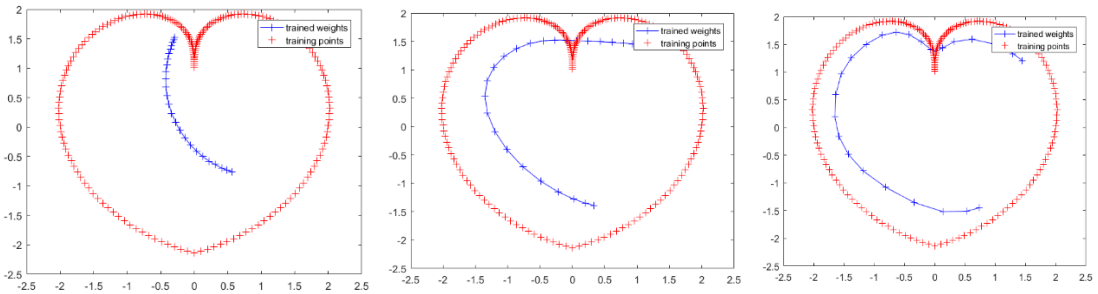
$$\eta(n) = \eta_0 e^{-\frac{n}{\tau}}$$

$$h_{j,i(x)}(n) = e^{-\frac{{d_{j,i}}^2}{2\sigma(n)^2}}, \qquad \sigma(n) = \sigma_0 e^{-\frac{n}{\tau}}$$

After the implementation, we can get the results in figure 3.1 with different iteration times.



Iteration = 1 Iteration = 20 Iteration = 50



Iteration = 100 Iteration = 300 Iteration = 500

Figure 3.1 Performance of 1-dimension SOM with different iteration times

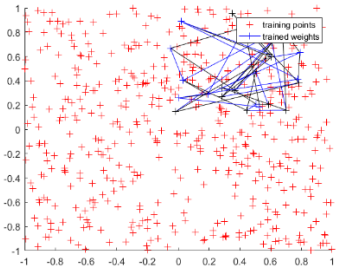
During the early stages of training, the SOM may learn the basic features of the input data. The data gradually smoothed out from aggregation to dispersion. As the number of iterations increases, the SOM will continue to refine its representation of the input data, and will be more close to the "heart curve".

b) SOM implementation with 2-dimensional of 5*5 output layer of 25 neurons

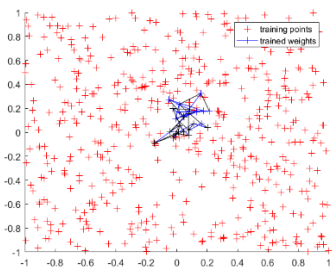
We change the neurons with a 2-dimensional output layer. Compared with the task a), we need to change the way to calculate the distance of neighbors by a 2 dimensional coordinate. The distance from the neuron at position (l, n) to the one at (n, k) can be calculated by

$$d_{i,j} = \sqrt{(l-n)^2 + (m-k)^2}$$

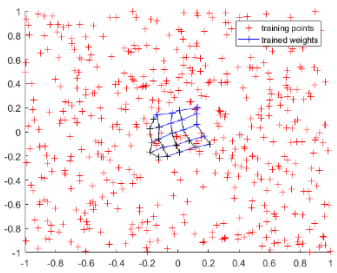
Then we get the results as shown in figure 3.2. At the beginning, the neurons are concentrated. With increasing iteration times, it will follow the distribution of the inputs. The inputs are uniformly distributed in this square, and they will eventually fill up the space of the inputs.



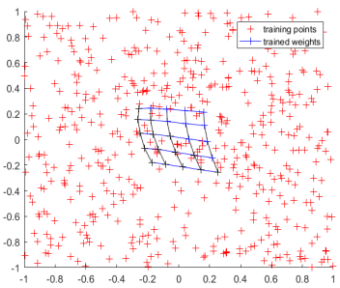
Iteration = 1



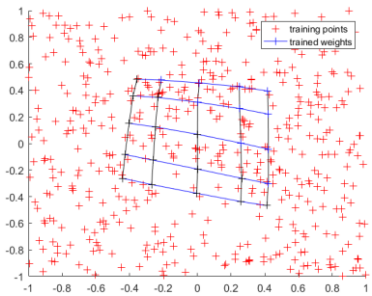
Iteration = 20



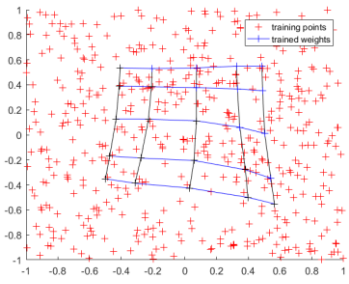
Iteration = 50



Iteration = 100



Iteration = 300



Iteration = 500

Figure 3.2 Performance of 2-dimension SOM with different iteration times

c) Implementation of SOM for classification of handwritten digits

c-1) Conceptual map of trained SOM and trained weights visualization

We use the SOM to do handwritten digits classification. We ignore the class 5 and 2 to do the classification of the other 8 numbers. The output neuron is a 10*10 map and we print out the corresponding class of each neuron as shown in figure 3.3. We can also visualize the trained weights of each output neuron on a 10*10 map and reshape into a 28*28 matrix as shown in figure 3.4. It is clear that the neighbors of images are similar to each other. The number with similar shape will show in close area. The far the distance of two neurons, the bigger difference of the shape of digits.

3 3 3 8 0 0 0 0 0 0
3 8 3 3 3 0 0 0 0 0
3 3 3 3 6 6 6 6 0 0
3 3 3 8 6 6 6 6 4 0
3 3 0 8 8 6 4 4 4 4
1 1 1 1 3 7 4 4 4 4
1 1 1 1 7 7 9 9 4 4
1 1 8 8 7 9 9 9 4 4
1 1 8 8 8 9 7 7 0 9
1 1 8 8 8 7 7 7 7 7

Figure 3.3 Conceptual map of trained SOM

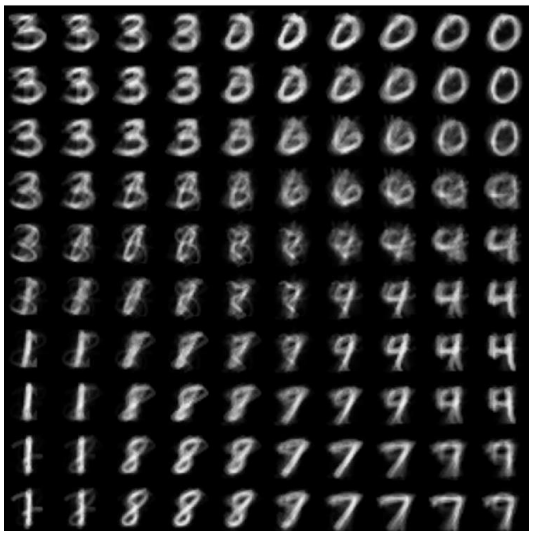


Figure 3.4 Visualization of trained weights

c-2) Training and classification analysis of trained SOM

Table 3.1 The training and testing accuracy of SOM via iterations

Iteration	1	50	100	300	600	800	1000
Training accuracy	31.76%	41.59%	28.39%	37.11%	60.77%	75.72%	80.70%
Testing accuracy	32.16%	33.67%	26.13%	37.11%	48.24%	67.34%	70.35%

We can calculate the accuracy of training and testing accuracy with increasing iterations. When we iterate for 1000 times, the training accuracy will be 80.7% and testing accuracy will be 70%. It has an oscillation between 50-300 iterations. The performance will be not that good of SOM with a few iterations. We should implement enough iterations and the accuracy will be relative stable.