4/16/2018

UNSUPERVISED LEARNING ON HANDWRITING DIGITS

USING PRINCIPAL COMPONENT ANALYSIS AND COMPETITIVE LEARNING

Junjin Chen
THE UNIVERSITY OF SHEFFIELD

Abstract

Unsupervised learning is a machine learning method that can detect the structure inside a dataset without manual labelling the data. This article is split into 2 parts and will demonstrate how unsupervised learning can be used in training images and how well is the performance. In the experiment, we will use images of hand-writing digits as training data and apply two unsupervised learning method in training. First part describes the training process using Principle Component Analysis (PCA). At the end of the training process, the result will be clustered using K-mean clustering method and 3 most valuable principle components will be determined. The function of PCA is to reduce the dimension of the data. The second part uses Competitive learning method for training. Conscience method is added in the process in order to remove the dead units. At the end, correlation matrix is computed to determine the similarity between different units. Competitive learning develops distinguishable structure from the input data. In the experiment, it is able to learn all kinds of digits by their shape. In conclusion, PCA and Competitive learning are efficient unsupervised learning methods in developing structure from data. Results will be shown to illustrate the performance of above methods.

Introduction

Erkki Oja (1982) developed a learning rule to simulate how our brain learns over time by strengthening the neurons. It is based on Herbb's Rule (1949) which can be simply explained as "Cells that fire together wire together". Differ from Herbb's rule, Oja's rule goes through the process of multiplicative normalization. It also sorts out the steadiness issues and shows the connection between neuron network and the algorithm of Principle Component Analysis. Oja's rule now plays an important role in image processing and speech processing since it can deal with high dimension data.

Principle Component Analysis was developed in 1901 by Karl Pearson. It is a statistical method that converts a set of data from high dimensional space to a set of Principle Components that are linearly uncorrelated using orthogonal transformation. Thus, it can reduce the dimensions by choosing the most significant principle components as predictive model to test other data. An example application of PCA can be to see the connection in the genetic distance and relatedness among populations.

Stuart Lloyd proposed the algorithm of K-means clustering in 1957. K-means clustering partitions the data into k clusters and calculates the nearest mean and set the mean as new centre for every clusters in one iteration. Then the step is repeated until data converges. This algorithm is relatively simple to perform on large data sets. It is often used in computer vision and astronomy.

In the first part, we aim to reduce the dimension of image data from 784 to 3 using PCA method. K-mean clustering will be used to determine which 3 principle components are the best as model.

Early model of competitive learning was developed by Von der Malsburg in 1973 when he studied visual cortex. The model was able to describe the self-organization. Competitive learning usually contains 3 elements, the neuron net, the bound to the value in the net and the algorithm to determine which one of hidden units in competitive layer should be updated. (Duane, n.d.) The algorithm to choose one to update usually uses the Euclidean distance between the input and the weights of the hidden units. However, this can lead to some dead units since some units can never be updated and its distance will never become closer than other units. There are many ways to cope with this problem and reduce number of dead units. An example algorithm can be that update also the one with far distance, but with smaller learning rate. In the experiment, we will add a conscience method to our competitive learning, which should perform well in removing dead units without confusing other units.

Conscience algorithm keep tracking the number of times that each unit is updated. Then bias is added to balance the times that units should be updated.

In Conclusion, we will use PCA and Competitive learning to evaluate how well is unsupervised learning doing in classifying image data.

1.1 PCA

Principal component analysis (PCA) basically uses an orthogonal transformation transform high dimensional data into a new coordinate system, such that we have new axis on which data points lie with greater variance. Each of the new axis is called principle component (PC). Covariance matrix, Eigen-value and Eigen-vector can be computed to find the axis with maximum variance where data lies.

In our case, MNIST data (hand-writing image data) was used in training. We sampled 60000 (all) data from database. The data uses 1-byte unsigned integer to represent the value of pixels in an image. Therefore, the data was initially normalised and centred to having the feature vector length of 1. Then, the covariance matrix was computed using the formula:

$$C_{ij} = E[(X_i - \mu_i)(X_j - \mu_j)]$$

Where $\mu_i = E(X_i)$, and X_i is the vector of the normalised data.
Next, the eigen-value and eigen-vector can be calculated according to the formula:

$$C^0 \cdot P^n = \lambda^n \cdot P^n$$

Where C^0 is the covariance matrix, λ^n is the eigen-value and P^n is the principle component, as known as eigen-vector.
After that, we sort the eigen-value from largest to smallest and then we picked 6 eigen-vectors with largest eigen-values as our Principle component vectors.

Next, we (once) randomly sampled 10000 data from previous results, we keep the sample data, each time select 3 principle components (vectors) from data, cluster the data using the K-means clustering method with 10 clusters and keep the result. (since there are 10 digits). Figure 11 in Appendix C is the python code for K-means clustering. Finally, we plot the clustered data and determine with different colours and determine which principle components are the most effective in classification.

We select following combination of PCs as model to plot 3D graphs:

a. 1st, 2nd and 3rd
b. 2nd, 3rd and 4th
c. 1st, 3rd and 4th
d. 2nd, 3rd and 5th
e. 3rd, 4th and 5th

The result graphs can be found in Appendix A.

The numbers of clusters are 10 for all graphics since we use K-means clustering. Instead judging from the number of clusters, we can see how unambiguous between clusters to determine which 3 PC are most effective. From the figures in Appendix A, PC2, 3 and 4 seems to have the clearest borders between clusters. Therefore, we can conclude that the 2nd, 3rd and 4th principle components are most useful.

1.2 Competitive learning

We implemented Competitive learning in a neuron network that contains 784 (total number of features in data) input neurons and 15 competitive neurons. Below is the training process:

Initially, the weights in the network was randomly generated and normalized. The visual form of the weights would be a mess. Then, in each of 40000 training steps, an image data was randomly sampled from 60000 images in data set to feed the network. The sample data was normalized such that the network learns with the right pace from the data. Next, the algorithm of conscience was used to decide the one competitive neuron to be updated. The detail of how this algorithm work will be explained later. After the winner neuron was chosen, its weight vector was updated simply by the formula:

$$\Delta w = \alpha(X_i - W_i)$$

Where α is the learning rate, X_i is the input signal and W_i is the previous weight vector.

This is an online learning. Differ from batch learning, this rule uses the updated weights to determine the amount to be updated in the future, where in batch learning, the order of training does not really matter since the current training step in batch learning will not interfere the future steps.

We mention that the weight vectors and the data needs to be normalised. This process is very important because the lengths of the weight vectors and raw data vector are quite different. The above formula for updating weights indicates that we need to normalise it to the same size in order to have the network learning in correct pace. If we did not normalise the weight or the data, the result = can go extreme, e.g. only learns in one unit or it never learns.

The mechanism of conscience has 2 parts, first part is the base algorithm of competitive learning, i.e. generate the outputs by calculating Euclidean distance; second part is to adjust the outputs by adding some bias related to the frequency that a unit is updated. The output for competitive layer is either 1 or 0 depending on whether it is the closest to the input, output y_i can be described as:

$$y_i = 1 \; if \; |W_i - X| < |W_j - X| \quad \forall j \neq i$$

$$else \; y_i = 0$$

Where W_i is the weight vector and X is the input. If there are more than 1 weight vectors closest to the input, we pick only one weight vector to have output value 1.

Then, we have the eligibility trace e_i to trace the frequency of update as:

$$e_i' = e_i + B(y_i - e_i)$$

Where B represents the rate of trace and

$$0 < B \ll 1$$

In our case, $B = 0.001$

Now, we can compute the bias b_i by:

$$b_i = C(\frac{1}{N} - e_i)$$

Where C is the bias factor and N is the number of competitive neurons.

In our case, $C = 7$

Finally, we can determine the winner unit by the following algorithm:

$$W_i \; wins$$

$$if \; |W_i - X| - b_i < |W_j - X| - b_j \quad \forall j \neq i,$$

$$else \; W_i \; lose$$

This algorithm cares about those units that is barely visited, thus can reduce or remove the dead units.

With only the base competitive algorithm, there are 2-3 dead units usually. However, this method removes all dead units. Each scenario is run 12 times. Figure 12 in Appendix C shows the python code for conscience algorithm.

A way to detect dead units can be tracking the frequency of each unit being updated. From the frequencies, we can derive the probability of a unit being updated. We can then compare to the expected probability to check if the unit is barely visit or not. For example,

suppose there are 15 units. To be fair, each unit is expected to have the probability of 1/15 to get updated. Then we can say if the actual probability of a unit is even lower than 100th of the expected probability, this unit would be regarded as dead unit.

In order to see when the network learns the data sufficiently, a semi-log axis of average weight change is plotted as Figure 6 and Figure 7 in Appendix B. The curve is smoothed with Gaussian sliding window; and the size of the window is 401. Figure 6 shows the average weight change without removing dead units. The time network learns sufficiently is less than a thousand as shown in Figure 6. However, it leaves 2 dead units. Figure 7 shows the average weight change when removing the dead units. As we can see, there are gaps in the curve and it becomes stable after 20 thousand iterations. The gaps indicate the time when conscience method cares about the dead units.

Figure 8 and Figure 9 in Appendix B show the prototypes that the network learns, as shown, they both contains all 10 prototypes of digits. Figure 8 represents the one without removing dead units, there are 2 dead units and the image of prototypes are blurry and ambiguous. Figure 9 represents the one that removes dead units. The prototypes are clearer than the ones in Figure 8.

Here are the links to see the animated weight changes over time:

https://drive.google.com/file/d/1jY_1SUGS6VJGhiBWDuqjX7upnO6SLfGf/view?usp=sharing

adding conscience method:

https://drive.google.com/file/d/1dALrS-bTV0Xgp8PAIZOKgtEgefiEhCSl/view?usp=sharing

Figure 10 in Appendix B shows the visualised correlation matrix with the prototypes. The matrix indicates the correlations among data features. Similar prototypes will have similar correlation matrix; we can pick a few pairs of features from the correlation matrix that are most representative to determine the similarities between prototypes.

Discussion

The performance of unsupervised learning in image data was good. The implementation of PCA shows that the image data with high dimension can be effectively represented with a few principle components, together with the K-means clustering method. While competitive learning with conscience method successfully learns the weights that represents all different digits. In fact, the K-means clustering in some way is equivalent to Competitive learning. K-means clustering assigns each data point to the nearest cluster and competitive learning updates each input to the nearest competitive neurons. In this way, we combined the PCA with the Competitive learning. If we implement the PCA alone, the result will not be informative as there is no clusters and all data looks the same, with competitive learning, we can distinguish the data points that lie in principle components.

Reference

Duane DeSieno, (n.d.), “ADDING A CONSCIENCE TO COMPETITIVE LEARNING“, Available: http://www.inf.ufrgs.br/~engel/data/media/file/cmp121/conscience%20competitive%20learning.pdf

Appendix A

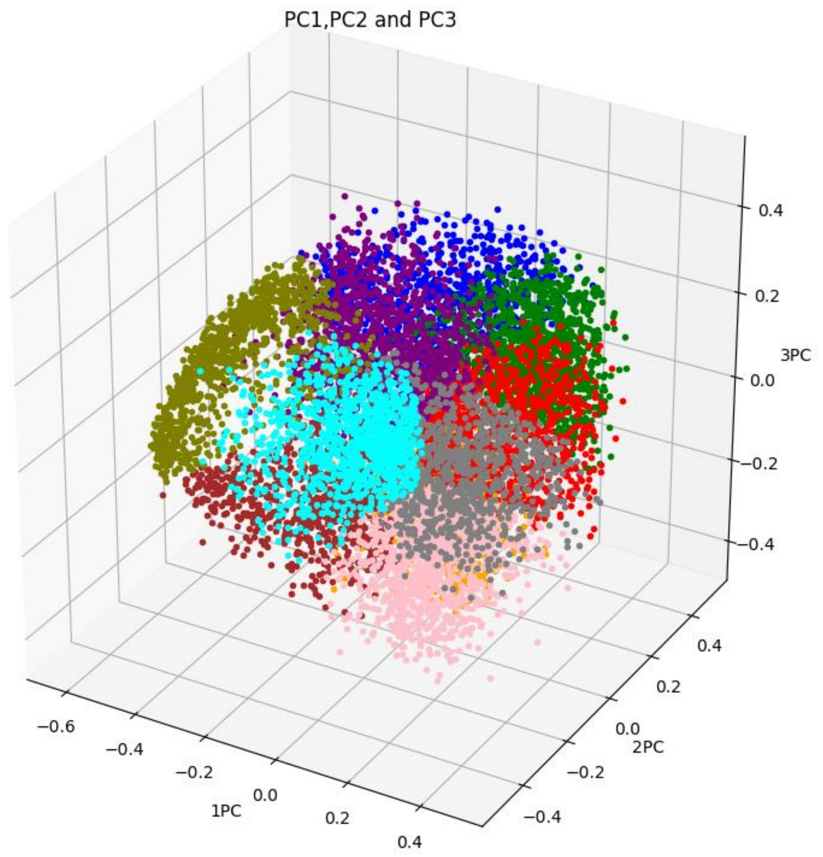


Figure 1: Principle Component 1,2 and 3 for the hand-writing image data

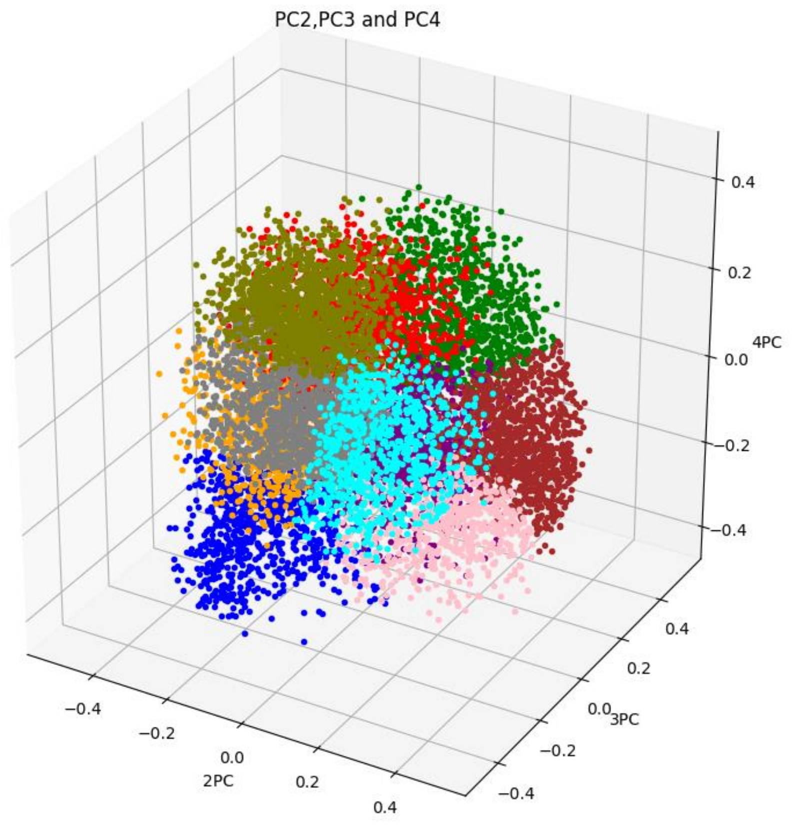


Figure 2: Principle Component 2,3 and 4 for the hand-writing image data

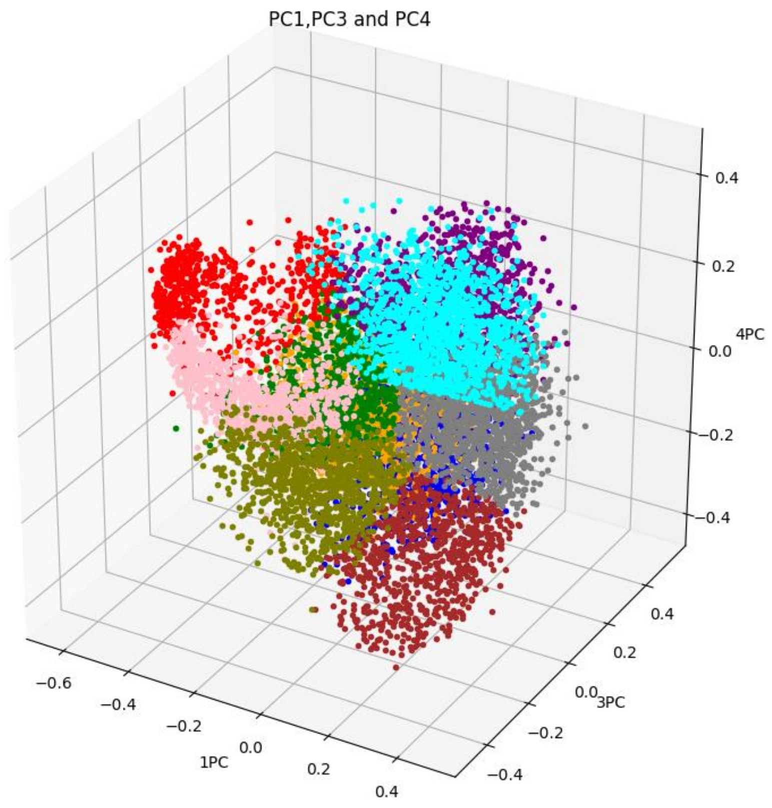


Figure 3: Principle Component 1,3 and 4 for the hand-writing image data

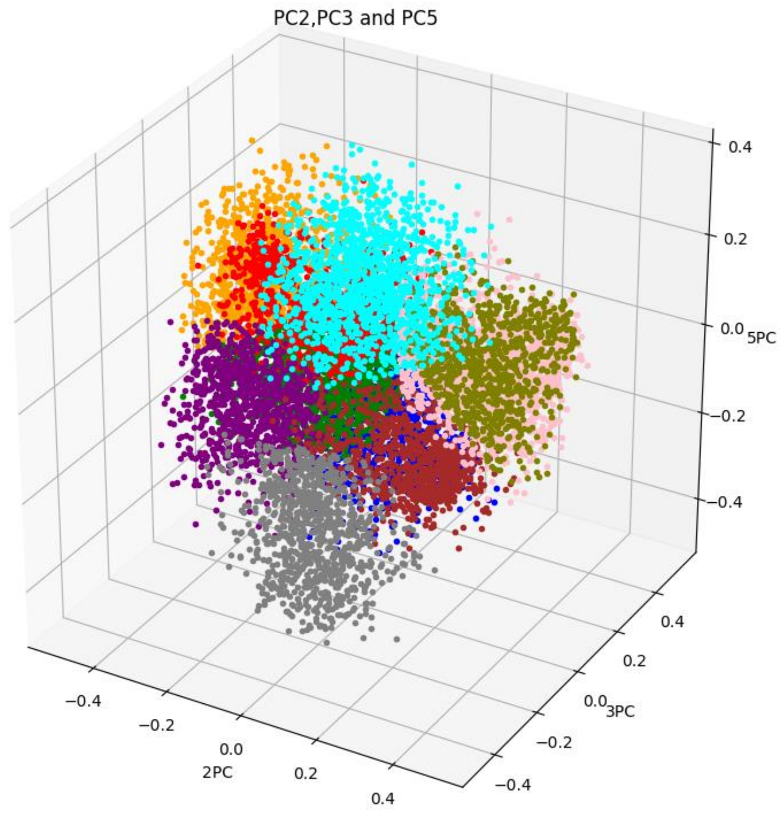


Figure 4: Principle Component 2,3 and 5 for the hand-writing image data

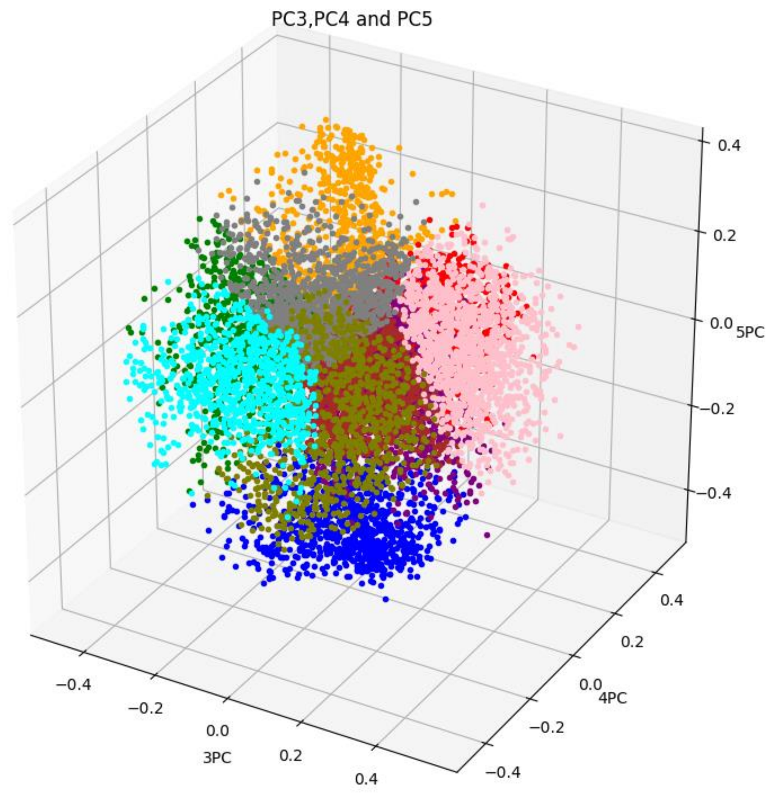


Figure 5: Principle Component 3,4 and 5 for the hand-writing image data

Appendix B

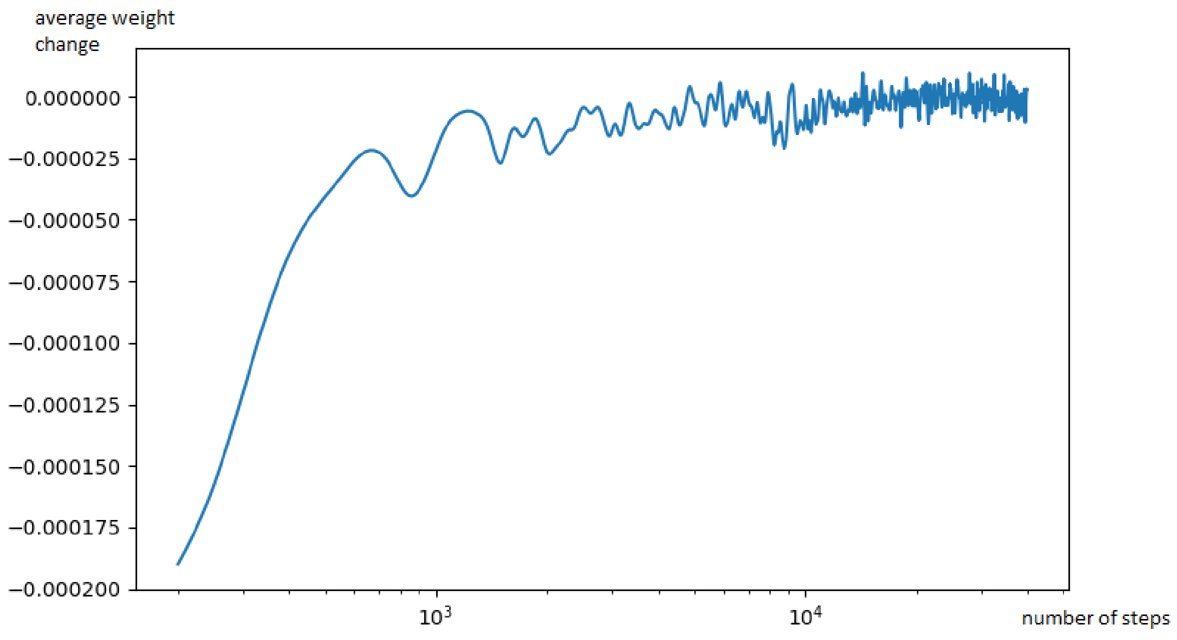


Figure 6: Average Weight changes in Competitive learning without removing dead units

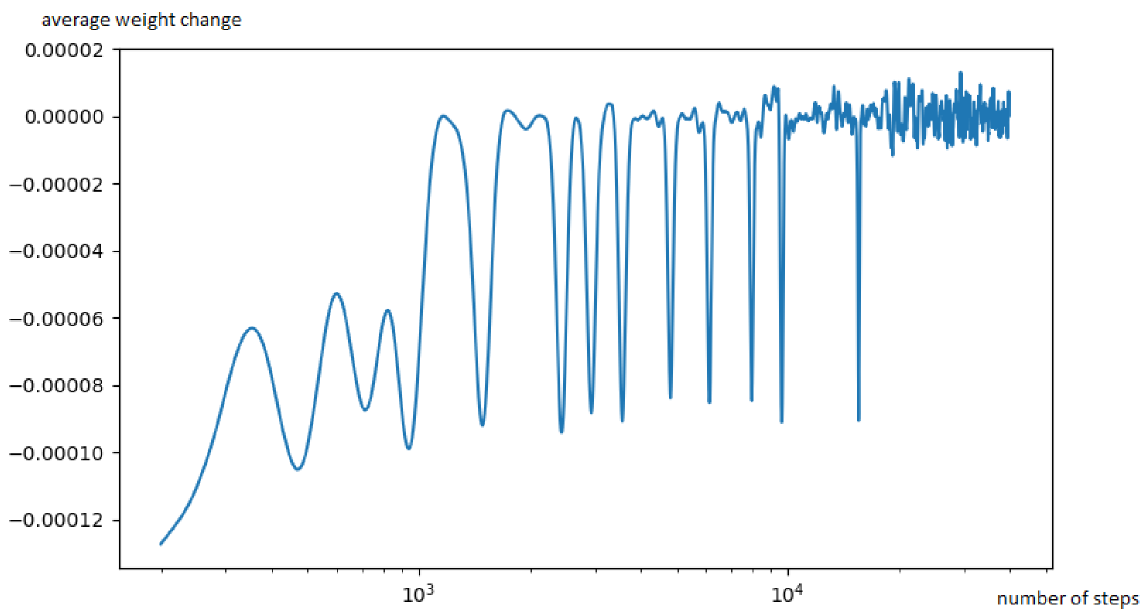


Figure 7: Average Weight changes in Competitive learning removing dead units

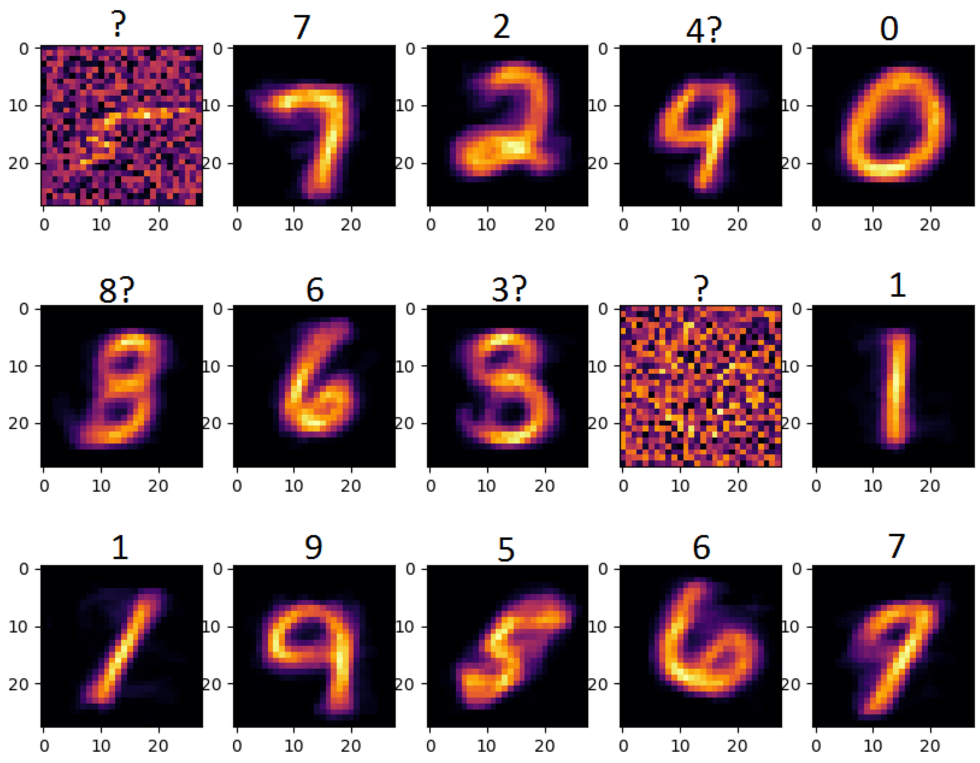


Figure 8: Prototypes of Competitive learning without removing dead units

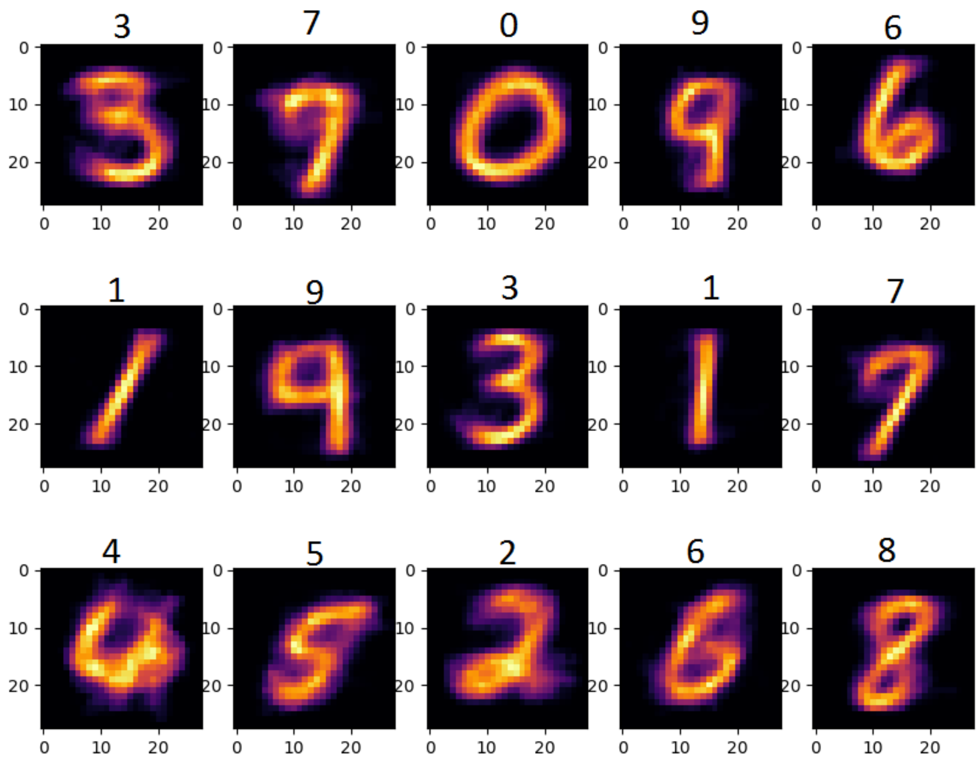


Figure 9: Prototypes of Competitive learning removing dead units

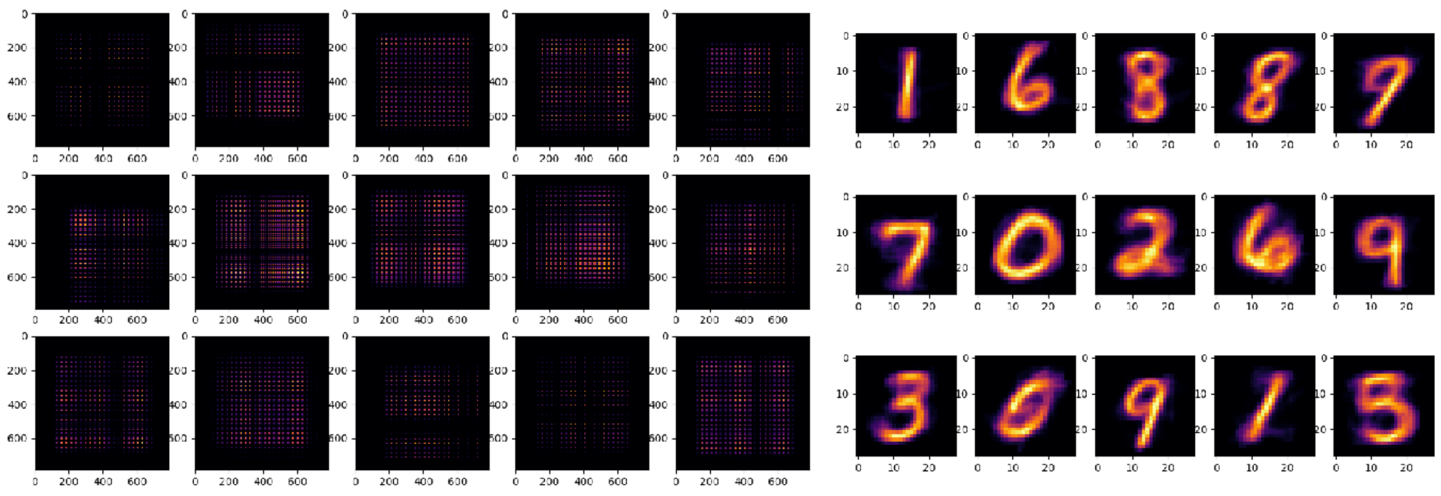
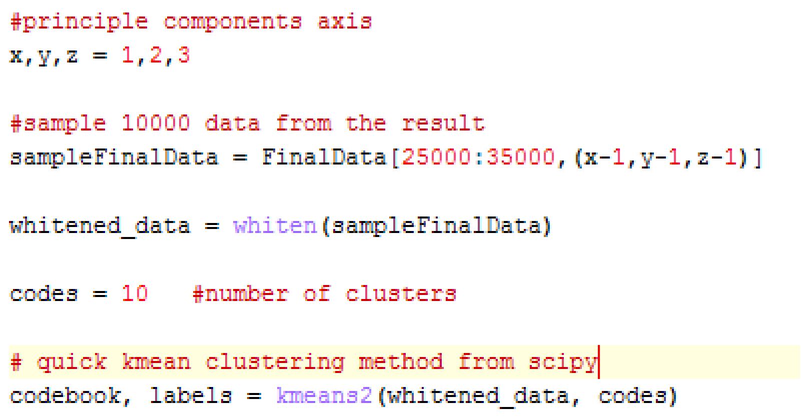


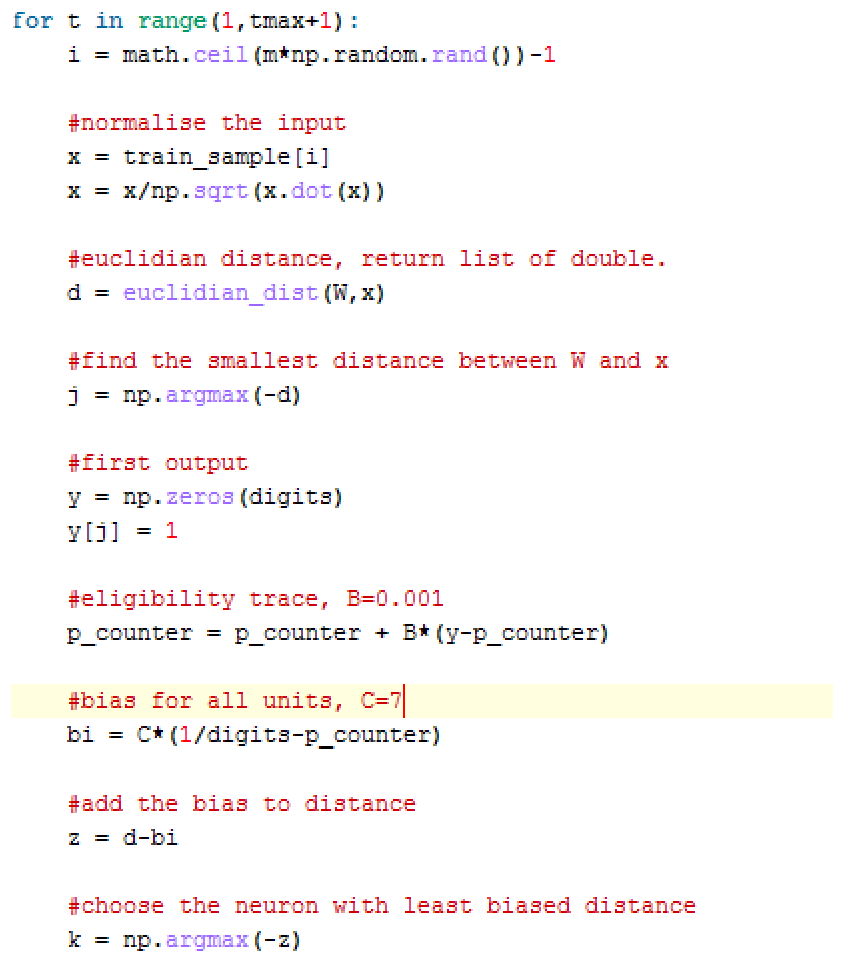
Figure 10: visualise correlation matrix with its prototypes.

Appendix C



```
#principle components axis
x,y,z = 1,2,3

#sample 10000 data from the result
sampleFinalData = FinalData[25000:35000,(x-1,y-1,z-1)]

whitened_data = whiten(sampleFinalData)

codes = 10   #number of clusters

# quick kmean clustering method from scipy
codebook, labels = kmeans2(whitened_data, codes)
```

Figure 11. K-means clustering python code



```
for t in range(1,tmax+1):
    i = math.ceil(m*np.random.rand())-1

    #normalise the input
    x = train_sample[i]
    x = x/np.sqrt(x.dot(x))

    #euclidian distance, return list of double.
    d = euclidian_dist(W,x)

    #find the smallest distance between W and x
    j = np.argmax(-d)

    #first output
    y = np.zeros(digits)
    y[j] = 1

    #eligibility trace, B=0.001
    p_counter = p_counter + B*(y-p_counter)

    #bias for all units, C=7
    bi = C*(1/digits-p_counter)

    #add the bias to distance
    z = d-bi

    #choose the neuron with least biased distance
    k = np.argmax(-z)
```

Figure 12. Competitive learning Conscience algorithm python code