In the Name of God

Neuroscience of Learning, Memory, Cognition

Homework_4 Report

- Hand Written Digit Recognition
- PCA Algorithm

Armin Panjehpour – 98101288

1- Hand Written Digit Recognition:

1.1 – **visualize some numbers of the dataset**:

۱.۲ – **make train and test data**:

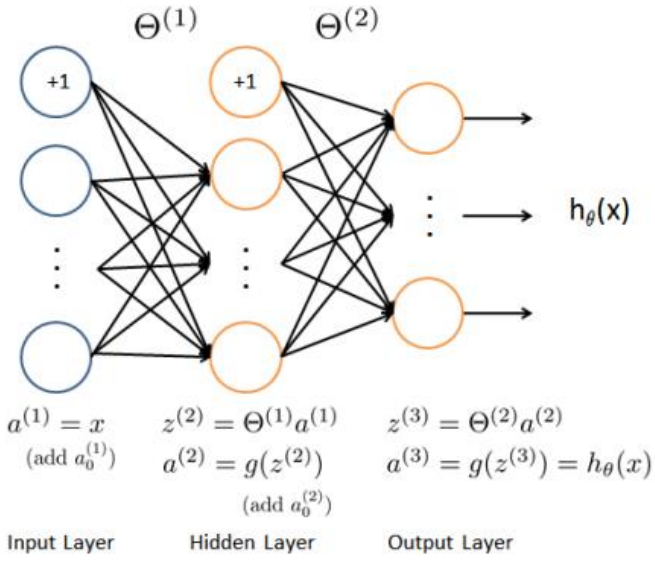
In the dataset, we have 500 samples for each number and there is 10 digits, so we have a 5000×400 dataset called X. Also, there is a y vector which contains the digit of each these 5000 samples, So y is a vector with length of 5000. Now we take out each 300 samples of each number for the **train dataset** and we would have a 3000×400 dataset for training the network and the other 200 samples for each number will be stored in the **test dataset** for the testing the network at the end which is a 2000×400 dataset. This clustering will be applied to the y dataset, too.

test	2000x400 double	y_test	2000x1 double
train	3000x400 double	y_train	3000x1 double

Careful: "0" digit is labeled as "10" in the y dataset, while the digits "1" to "9" are labeled as "1" to "9" in their natural order.

1.3 – **Structure of the network**:

Here, We want to implement a 3 layer neural network, an input layer, a hidden layer and an output layer:



Since our digits are 20×20 images and each of the pixels has a value which it will be given to input layer neurons, The input layer contains 400+1 neurons. The hidden layer contains 25+1 neurons and the output layer has 10 neurons, each of them containing a value which represents the probability of the related digit being predicted. With this formula, we`ve calculate the initial values for the layers` weights:

$$W = rand(L_{out}, 1 + L_{in}) * 2 * \epsilon - \epsilon, \epsilon = 0.12$$

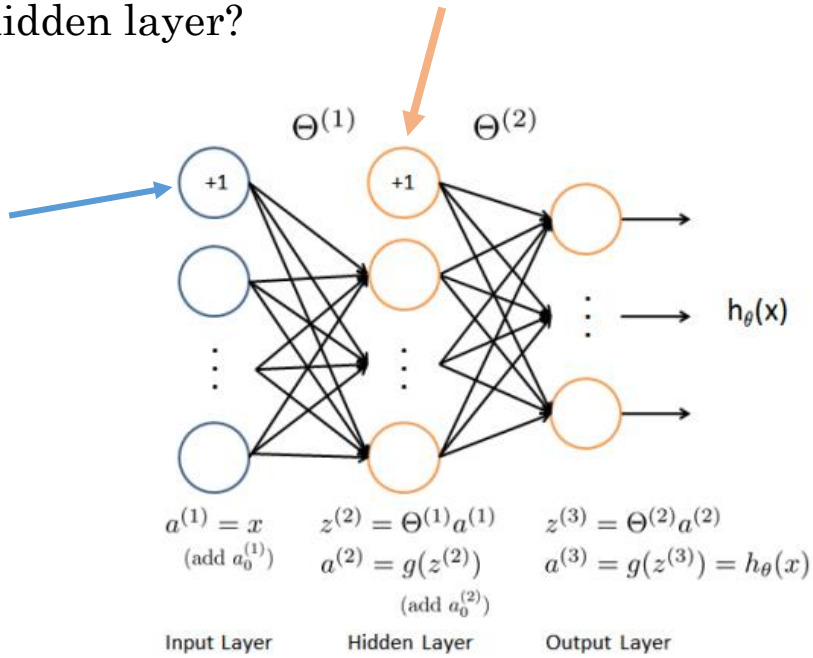
Which Lout and Lin are the number of output and

input neurons for that particular weight

We call the weights between the first and the second layer, W12 or θ_1 and the weights of the second and the third layer, W23 or θ_2.

```
W12 = (rand(25,401)).*2*0.12 - 0.12; % first layer weights
W23 = (rand(10,26)).*2*0.12 - 0.12; % second layer weights
```

By the way, what are the added neurons doing in the input/hidden layer?



These two neurons added to the 2 first layers are the bias neurons. They have been added to help us to control the behavior of the layers. This bias neuron will increase the flexibility of the model to fit the data. So having bias neurons in our network make our model performance as good as possible.

1.4 – **Cost Function and It`s derivative**:

We use sigmoid cost function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[-y_k^{(i)}log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})log(1-(h_\theta(x^{(i)}))_k)]$$
$$\frac{\lambda}{2m}[\sum_{j=1}^{25}\sum_{k=1}^{400}(\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}(\Theta_{j,k}^{(2)})^2]$$

The derivative of J(θ) is:

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta) = \frac{1}{m}\Delta_{ij}^{(l)} + \frac{\lambda}{m}\Theta_{ij}^{(l)} : for : j > 0$$

$$\frac{1}{m}\Delta_{ij}^{(l)} : for : j = 0$$

And sigmoid and sigmoid derivative functions are implemented as you can see below:

$$\text{sigmoid}(z) = g(z) = \frac{1}{1+e^{-z}}.$$

```
function output = sigmoid_calculator(input)
    output = 1./(1+exp(-input));
end
```

$$g'(z) = \frac{d}{dz}g(z) = g(z)(1-g(z))$$

```
function output = sigmoid_derivative_calculator(input)
    output = exp(-input)./((1+exp(-input)).^2);
end
```

1.5/1.6 – **BackPropagation**:

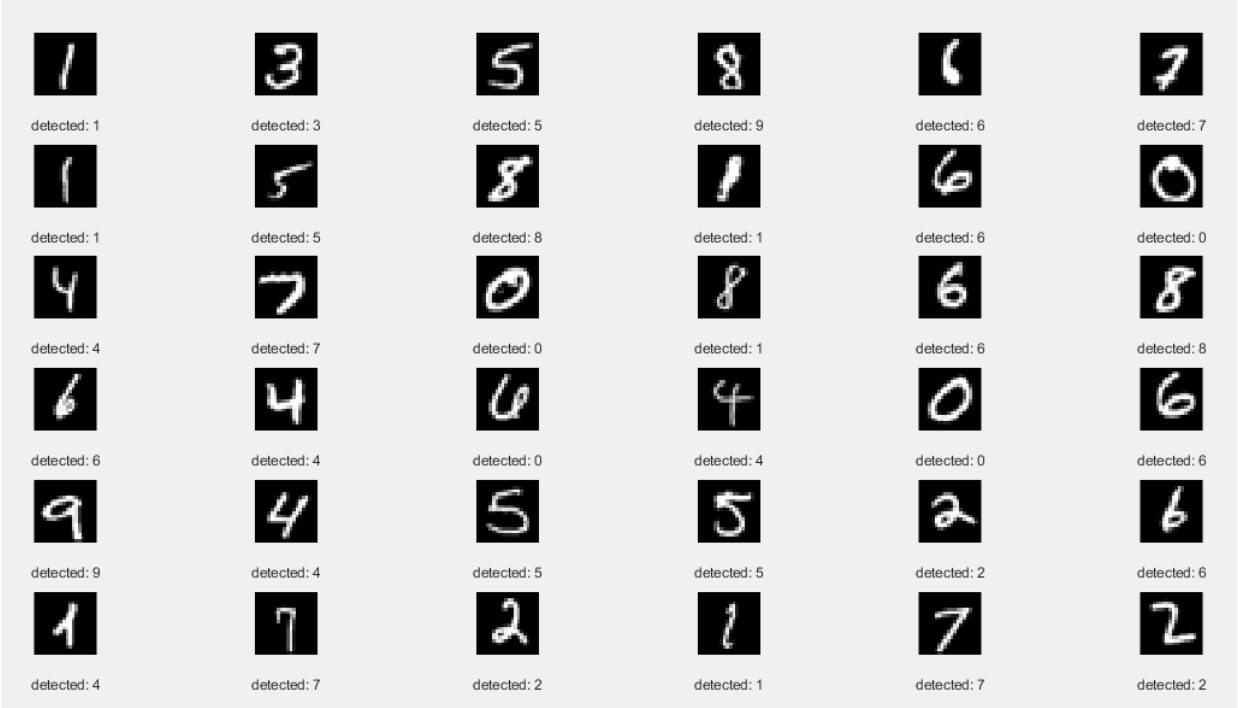
Now a nnCostFunction is implemented to do the backpropagation and calculate the delta`s:

$$\delta_k^{(3)} = a_k^{(3)} - y_k, \delta^{(2)} = (\Theta^{(2)T})\delta^{(3)}. * g'(z^{(2)}) \qquad \Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$$

And this function will be given to the fmincg which is an optimization function to update our weights and gives us the final weights.

۱,۷ – Visualize the output layer neuron:

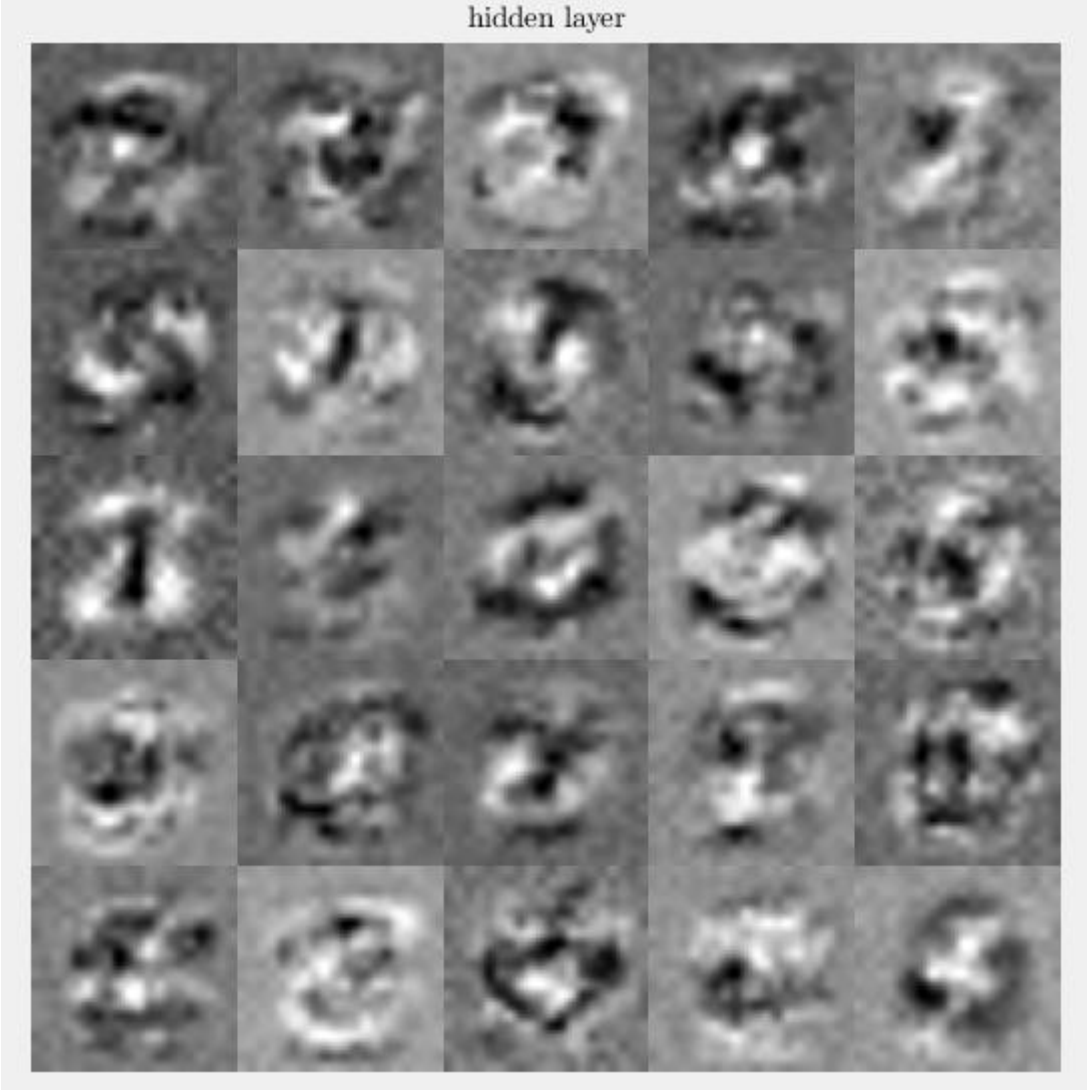
Now values of the 10 output neurons are calculated by the final weights with the test set and after that we have a 2000×10 output. Index of maximum of each row shows the predicted digit. Randomly 36 test samples are selected and compared to the detected numbers:



1.7 – Accuracy of the implemented neural network and visualized final W12:

My network accuracy after 50 times optimization is something between 95.9 to 96.3:

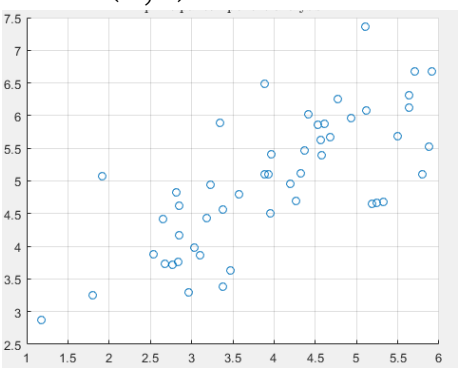
```
accuracy =          accuracy =

   95.9000             96.3000
```

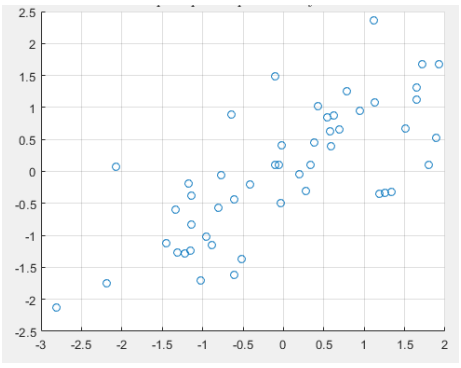


2- PCA Algorithm:

2,1 – **PCA Algorithm on a set of points:**

At first we subtract the mean of points and bring the mean of them to (0,0).





Then the covariance matrix is calculated:

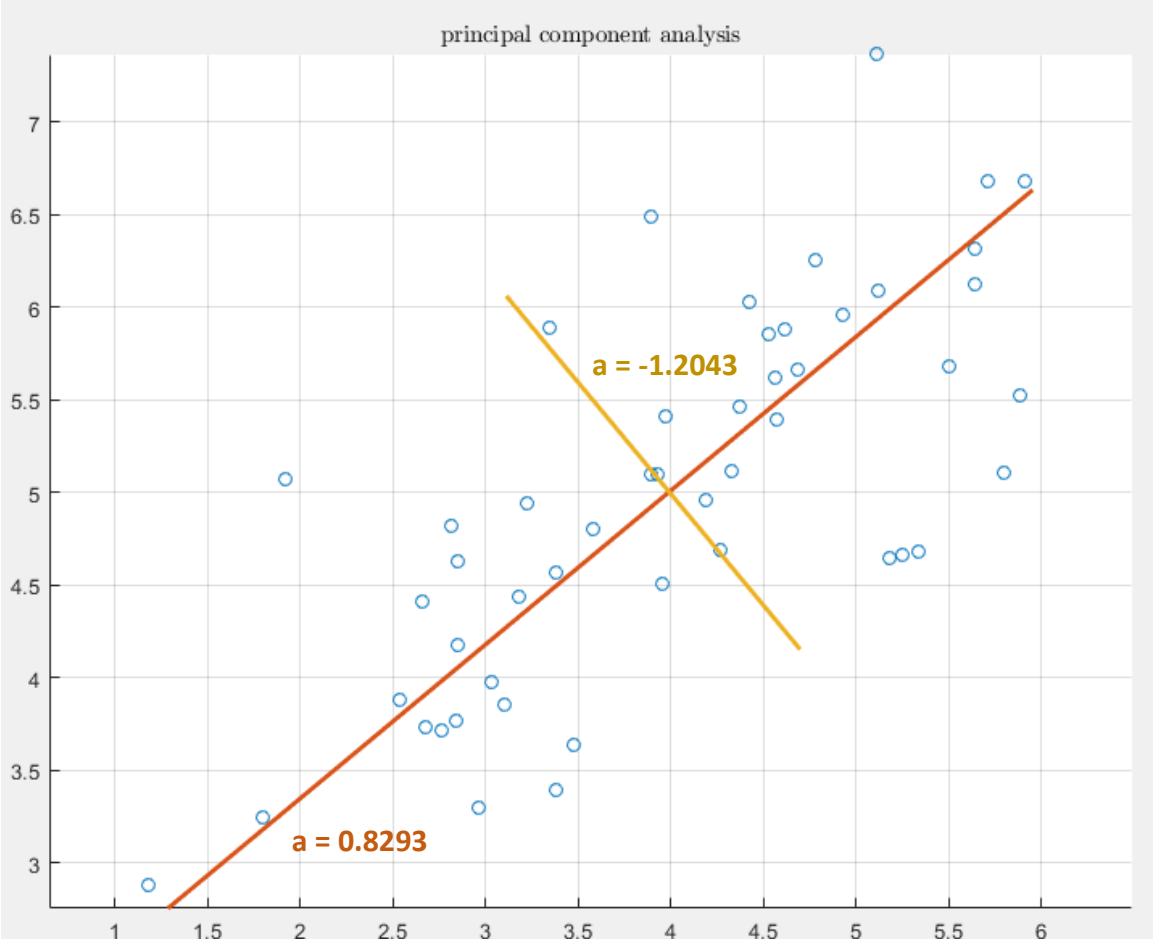
```
covMatrix =

    1.3760    0.8830
    0.8830    1.0474
```

Now eigenvectors/eigenvalues are calculated:

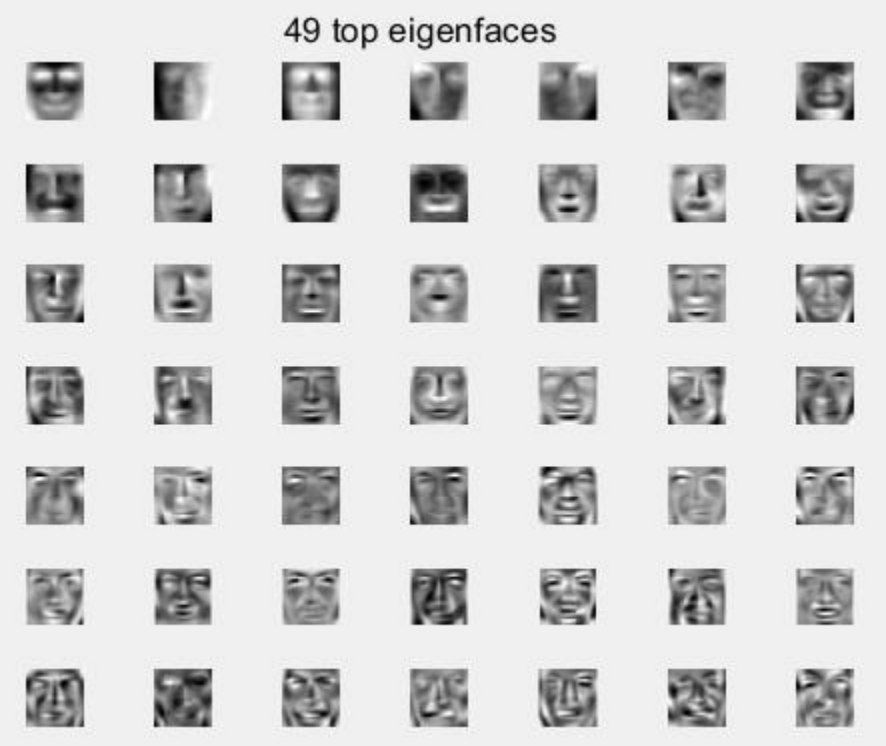
```
e =

    0.6392   -0.7691
   -0.7691   -0.6392
```

So the principal eigenvector is e(:,2) and this is the direction of the maximum vairance of the input data. And e(:,1) is the less principal eigenvector.

```
landa =

    0.3135         0
         0    2.1099
```



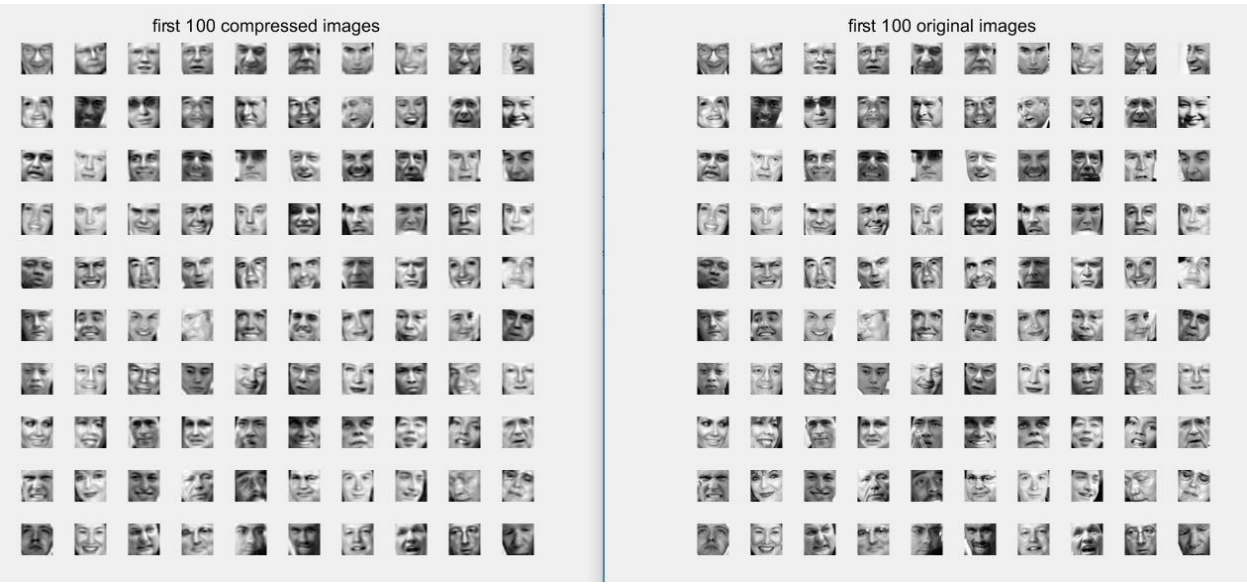
2,2/3 – **Eigenvalues and Eigenfaces:**



The 5 biggest eigenvalues:

0	0	0	0	0
2.7978e+08	0	0	0	0
0	3.1708e+08	0	0	0
0	0	5.5213e+08	0	0
0	0	0	1.1541e+09	0
0	0	0	0	1.8428e+09

Now we reconstruct the images(Compress them) using the 100 most principal eigenfaces:



As we expected, the difference can`t be relize by the human eye.