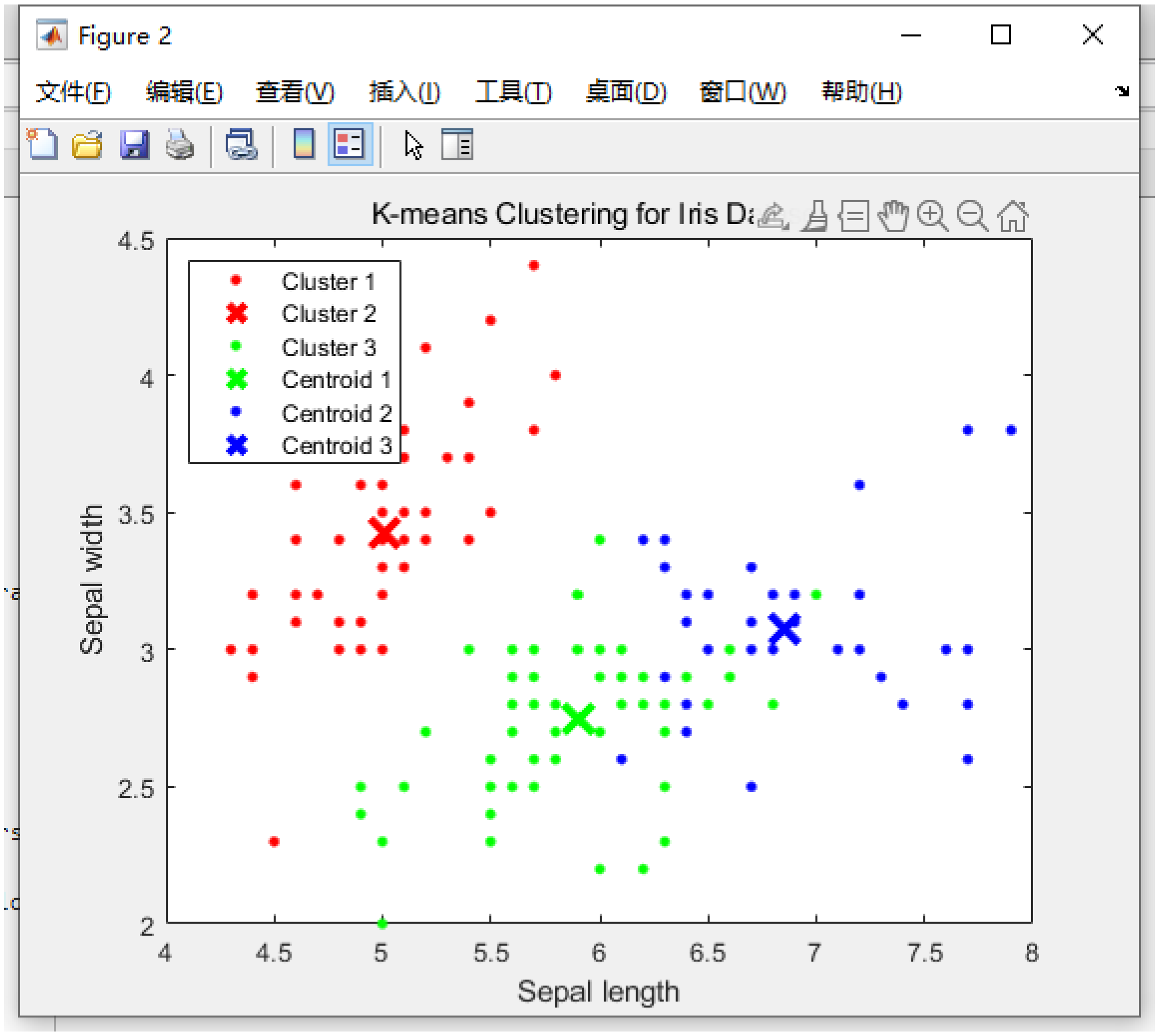
(P.s) because MATLAB already has an iris dataset which is called `fisheriris`. `It will be easier for me to input the dataset. So, the following code I use that API to set the data.`

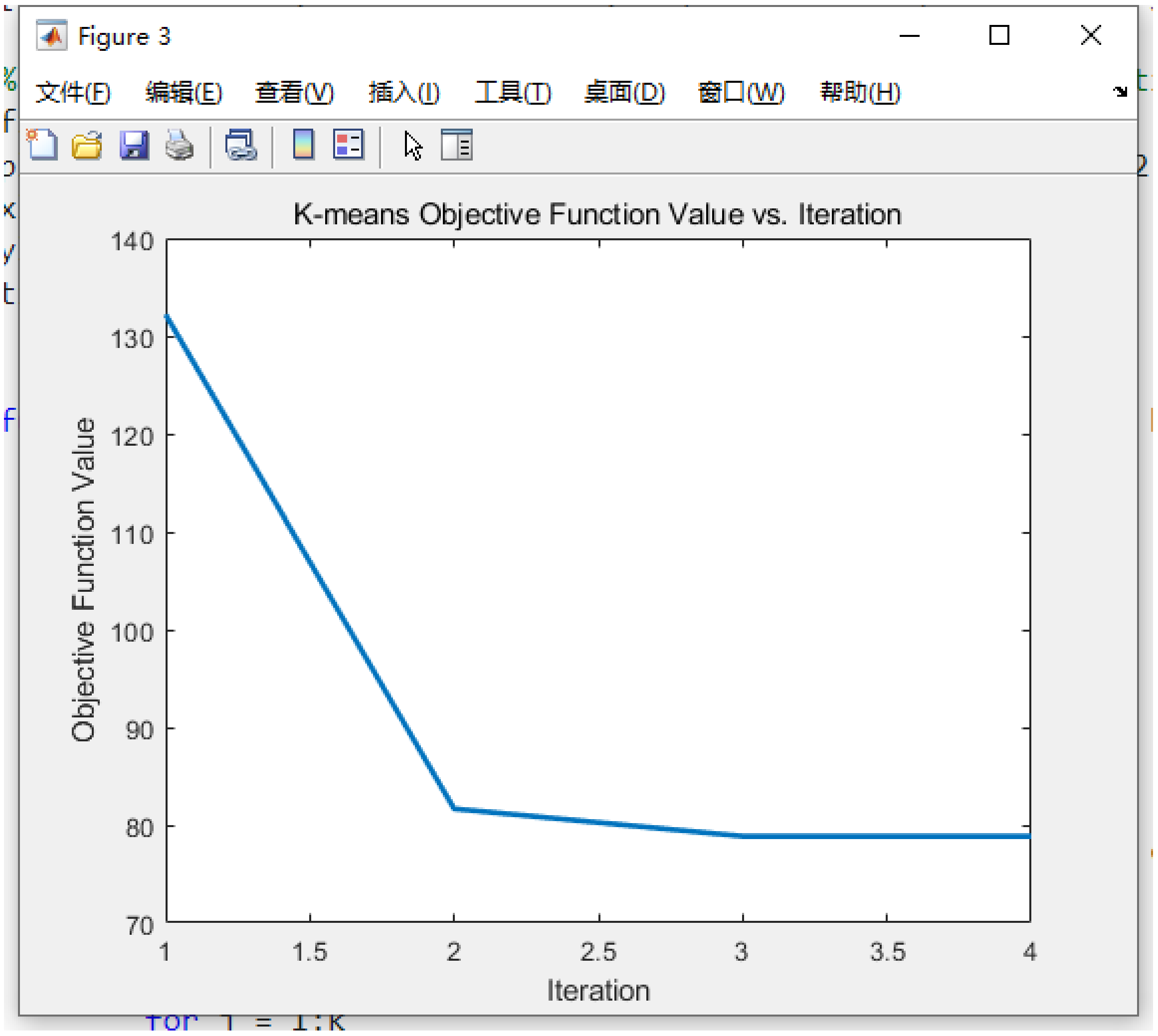
Clustering

a) Here is the plot shows as MATLAB.

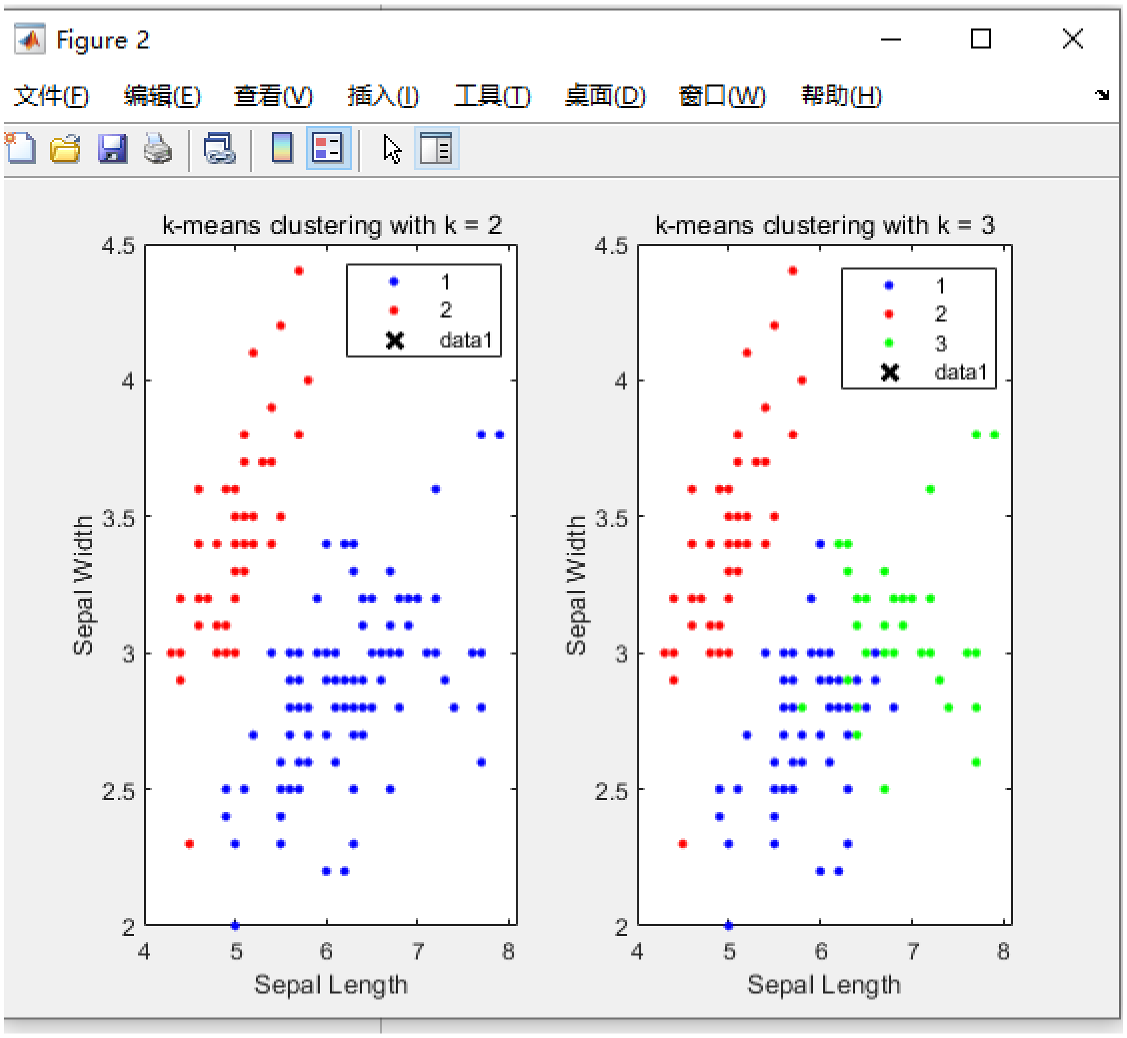


- MATLAB has a built-in Iris dataset which can be loaded by fisherriris.
- The Iris dataset consists of 4 features and 3 species. So we need to extract these variables by
- I set the k = 3 here since there are 3 species in the dataset. And the max iteration number is 10000. A larger max iteration number will make the graph looks better.
- "X" is the cluster centers and "." Is the following points distinguished by color.

b) Here is to calculate the distance between Xn and μk. I initialize the cluster centers and indices. Then using iteration to get the function value by assigning data points to the closest cluster center and keeping updating it. Last, checking the convergence of these value and plotting the final data on the graph. Shows as below:

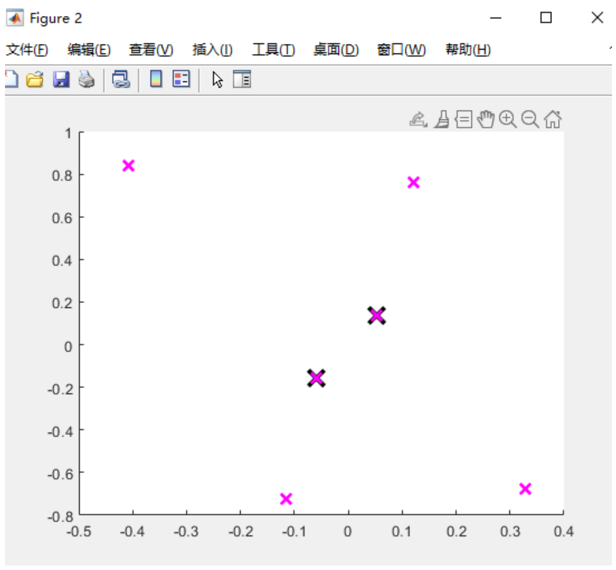


c) The plotting graph shows as below:
The function here is similar to part b.



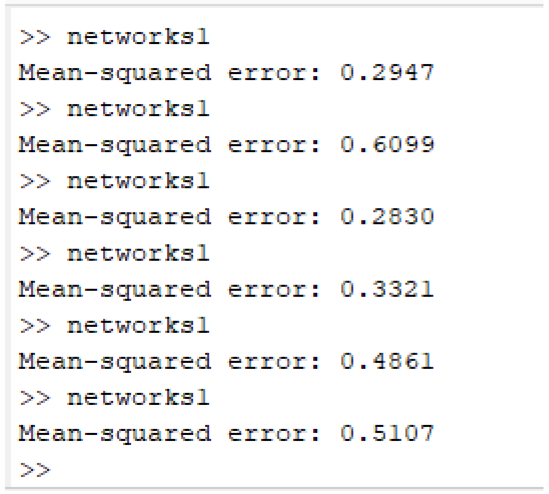
d)

I create a fine grid or meshgrid that spans the entire feature space (the first two features of the Iris dataset). The purpose of the meshgrid is to visualize the decision boundaries at a high resolution, so we can see how the algorithm classifies points in different regions of the feature space. Then, assign each meshgrid point to the closest centroid. The graph shows:



Neural networks

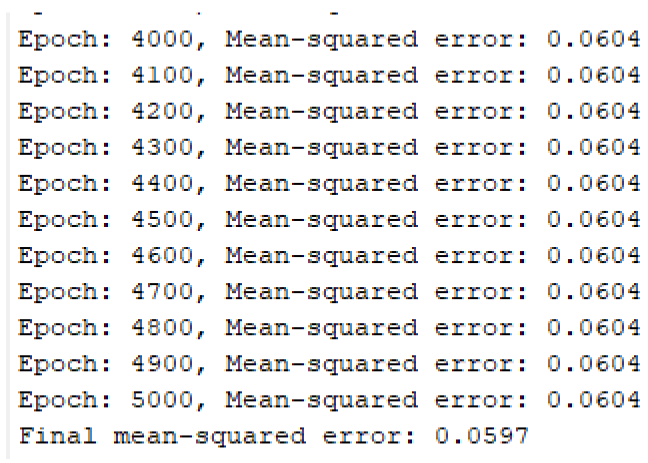
a) This program loads the Iris dataset, normalizes the data, defines the sigmoid function and the neural network parameters, calculates the neural network predictions, and computes the mean-squared error.
   By several times of calculation, we can see the error is really large here:



```
>> networks1
Mean-squared error: 0.2947
>> networks1
Mean-squared error: 0.6099
>> networks1
Mean-squared error: 0.2830
>> networks1
Mean-squared error: 0.3321
>> networks1
Mean-squared error: 0.4861
>> networks1
Mean-squared error: 0.5107
>>
```

   To improve it, the code is in networks1_5.m. By implements a one-layer neural network with a sigmoid non-linearity. A training loop using gradient descent to adjust the weights and biases, and a learning rate to control the speed of the training process.

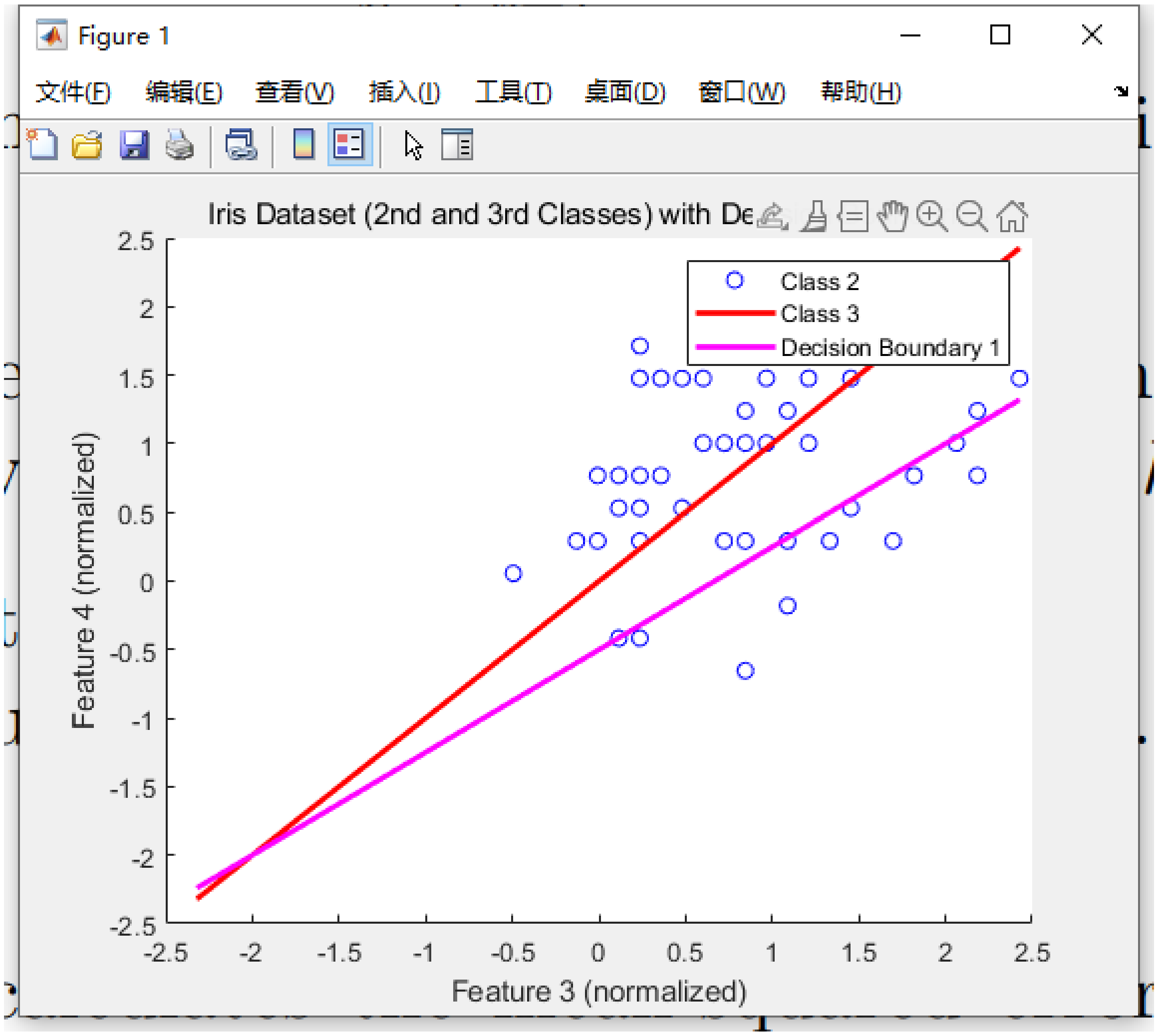
   The result shows here:



```
Epoch: 4000, Mean-squared error: 0.0604
Epoch: 4100, Mean-squared error: 0.0604
Epoch: 4200, Mean-squared error: 0.0604
Epoch: 4300, Mean-squared error: 0.0604
Epoch: 4400, Mean-squared error: 0.0604
Epoch: 4500, Mean-squared error: 0.0604
Epoch: 4600, Mean-squared error: 0.0604
Epoch: 4700, Mean-squared error: 0.0604
Epoch: 4800, Mean-squared error: 0.0604
Epoch: 4900, Mean-squared error: 0.0604
Epoch: 5000, Mean-squared error: 0.0604
Final mean-squared error: 0.0597
```

Which improved a lot.

b) The graph shows as follows:



```
>> networks2
Mean-squared error for the first setting: 0.2740
Mean-squared error for the second setting: 0.4743
```

In the code network2, we first load the iris dataset, then we load the Iris dataset and select only the 2nd and 3rd Iris classes (rows 51 to 150) and the 3rd and 4th features (columns 3 and 4) for simplicity. Then, we convert the categorical class labels to numerical labels and normalize the data to have a mean of 0 and standard deviation of 1. This helps the neural network converge faster during training. At last, we define the sigmoid function and neural network function and define two different weights and biases for the decision boundary.

c)

First, let the input data be X, weights be W (with dimensions n x 1), and the bias term w0. Define the following functions:

Net input: $net = X * W + w0$ Sigmoid function: $\sigma(net) = 1 / (1 + \exp(-net))$

Neural network output: $Y\_pred = \sigma(net)$

Mean squared error: $MSE = 1/N * \Sigma(Y\_t - Y\_pred)^2$, where N is the number of samples

Then, we use chain rule to get $\partial MSE/\partial W$:

$\partial MSE/\partial W = \partial MSE/\partial Y\_pred * \partial Y\_pred/\partial net * \partial net/\partial W$

$= -2/N * \Sigma(Y\_true - Y\_pred) * \sigma(net) * (1 - \sigma(net)) * X$

$= -2/N * \Sigma(Y\_true - Y\_pred) * Y\_pred * (1 - Y\_pred) * X$

This is the gradient of the mean squared error objective function with respect to the neural network weights.

d) For the scalar form, we will compute the gradient for each weight w_i independently. Let x_i be the corresponding input feature, y_i_true be the true output, and y_i_pred be the predicted output for the i-th sample.

Then, the gradient of MSE with respect to a single weight w_i can be written as:
$\partial MSE/\partial w\_i = (-2/N) * \Sigma(y\_i\_true - y\_i\_pred) * (y\_i\_pred * (1 - y\_i\_pred)) * x\_i$

Thus,

$\partial MSE/\partial W0 = (-2/N * \Sigma(Y\_true - Y\_pred)) * (Y\_pred * (1 - Y\_pred))$

$\partial MSE/\partial W1 = (-2/N * \Sigma(Y\_true - Y\_pred)) * (Y\_pred * (1 - Y\_pred)) * x1$

$\partial MSE/\partial W2 = (-2/N * \Sigma(Y\_true - Y\_pred)) * (Y\_pred * (1 - Y\_pred)) * x2$

$\partial MSE/\partial W3 = (-2/N * \Sigma(Y\_true - Y\_pred)) * (Y\_pred * (1 - Y\_pred)) * x3$

For the vector form, we will write the gradient as a column vector with the same dimensions as the weight vector W. Let Y_true be the column vector of true outputs and Y_pred be the column vector of predicted outputs.

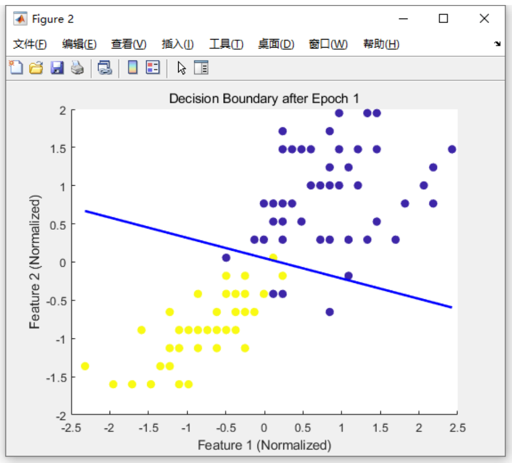
Then, the gradient of MSE with respect to the weight vector W can be written as:
$\nabla MSE = (-2/N) * X' * ((Y\_true - Y\_pred) .* (Y\_pred .* (1 - Y\_pred)))$
we can conclude that X' is the transpose of the input data matrix X and delta becomes the error term. Thus, we can conclude W0,W1,W2,W3 into:

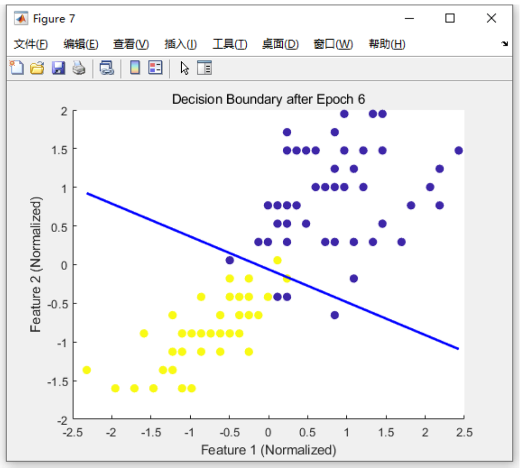
$\partial MSE/\partial W = [\partial MSE/\partial W0; \partial MSE/\partial W1; \partial MSE/\partial W2; \partial MSE/\partial W3] = X' * delta$

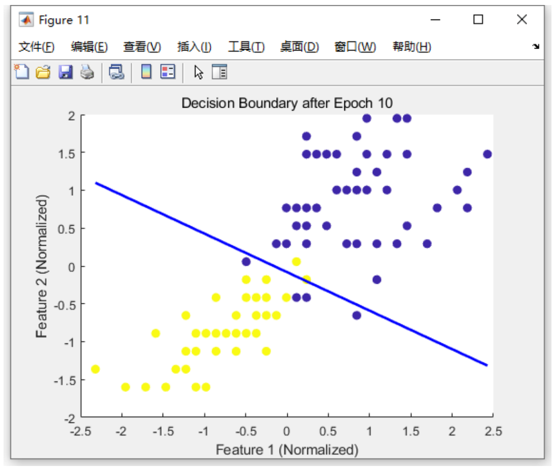
e) Here is the result of it:

```
>> network5
Initial mean-squared error: 0.2634
Epoch 1: mean-squared error = 0.0480
Epoch 2: mean-squared error = 0.0453
Epoch 3: mean-squared error = 0.0436
Epoch 4: mean-squared error = 0.0424
Epoch 5: mean-squared error = 0.0415
Epoch 6: mean-squared error = 0.0408
Epoch 7: mean-squared error = 0.0402
Epoch 8: mean-squared error = 0.0396
Epoch 9: mean-squared error = 0.0392
Epoch 10: mean-squared error = 0.0388
```

And we can see the change by the plotting graph.







As the weights are updated, the decision boundary shifts and rotates to better separate the positive and negative class labels in the input space.

In the code part named *network5.m.* It will run 11 graph immediately so you can check the changes easier.