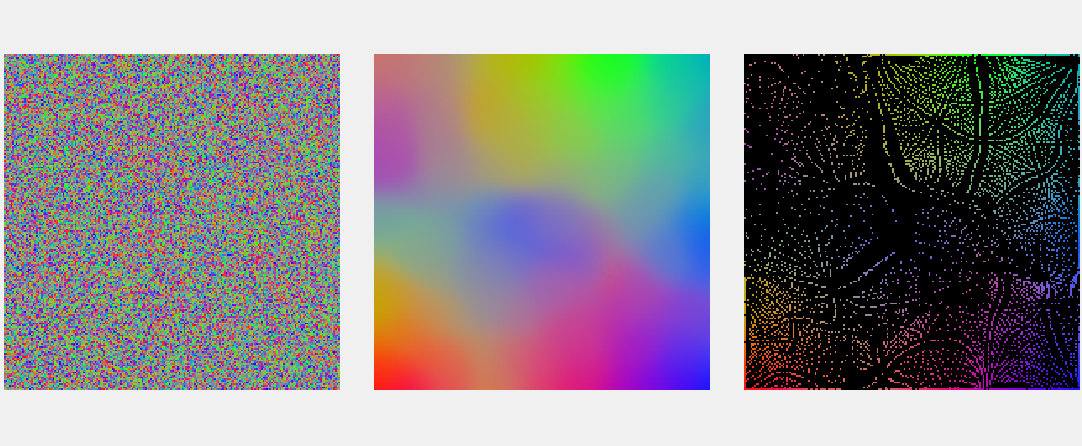
# Report——Chapter 19

**Problem1**:

Output:



There are three images in the output of the code, representing the results of the three stages of the SOM algorithm.

* The first image shows the initialized weight matrix W, where each pixel represents a weight vector. This image shows the random initialization state of the weight matrix at the beginning of the SOM algorithm.
* The second image shows the weight matrix W after training. After the training is completed, the SOM algorithm adjusts the weight matrix to better represent the low dimensional representation of the input data. This image shows the state of the weight matrix after training. (so it is colorful, because after clustering, all pixels are well divided into different color blocks)
* The third image shows the mapping of input data on SOM. For each input vector, the SOM algorithm maps it to the nearest neuron and sets the weight vector of that neuron to that input vector. This image shows the distribution of input data on SOM, where adjacent neurons represent similar colors.

Process of color learning:

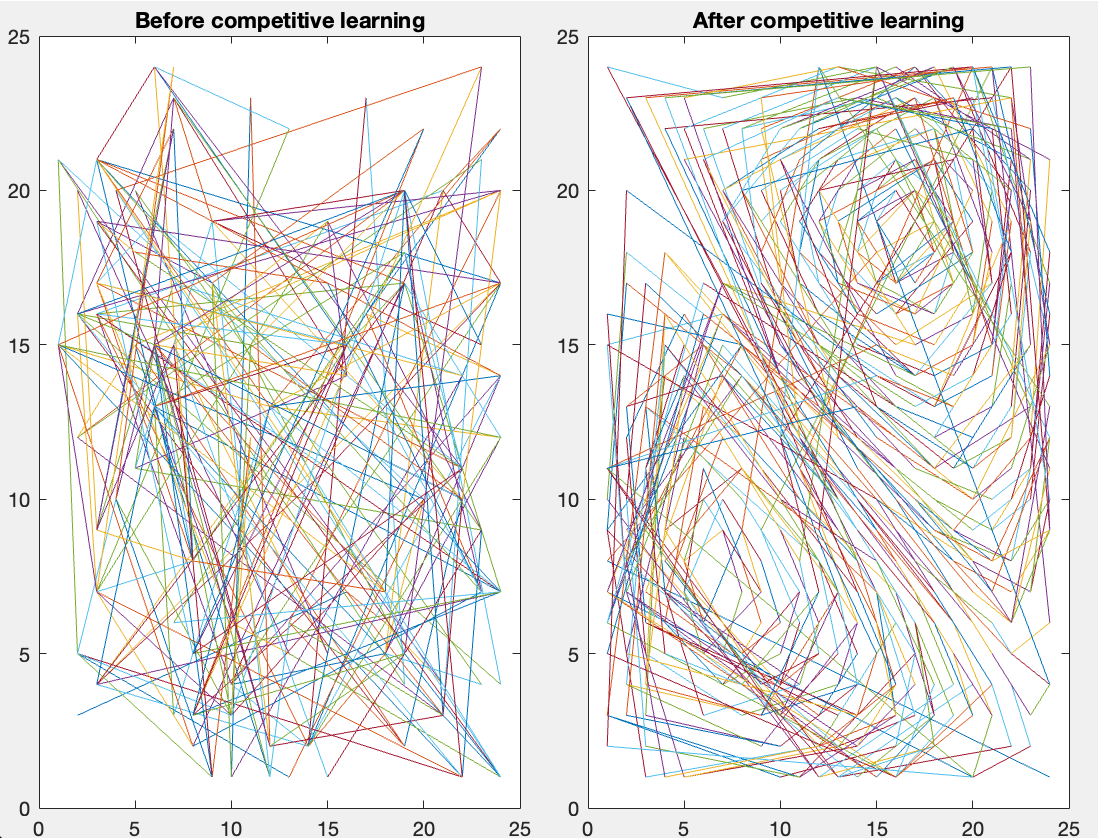
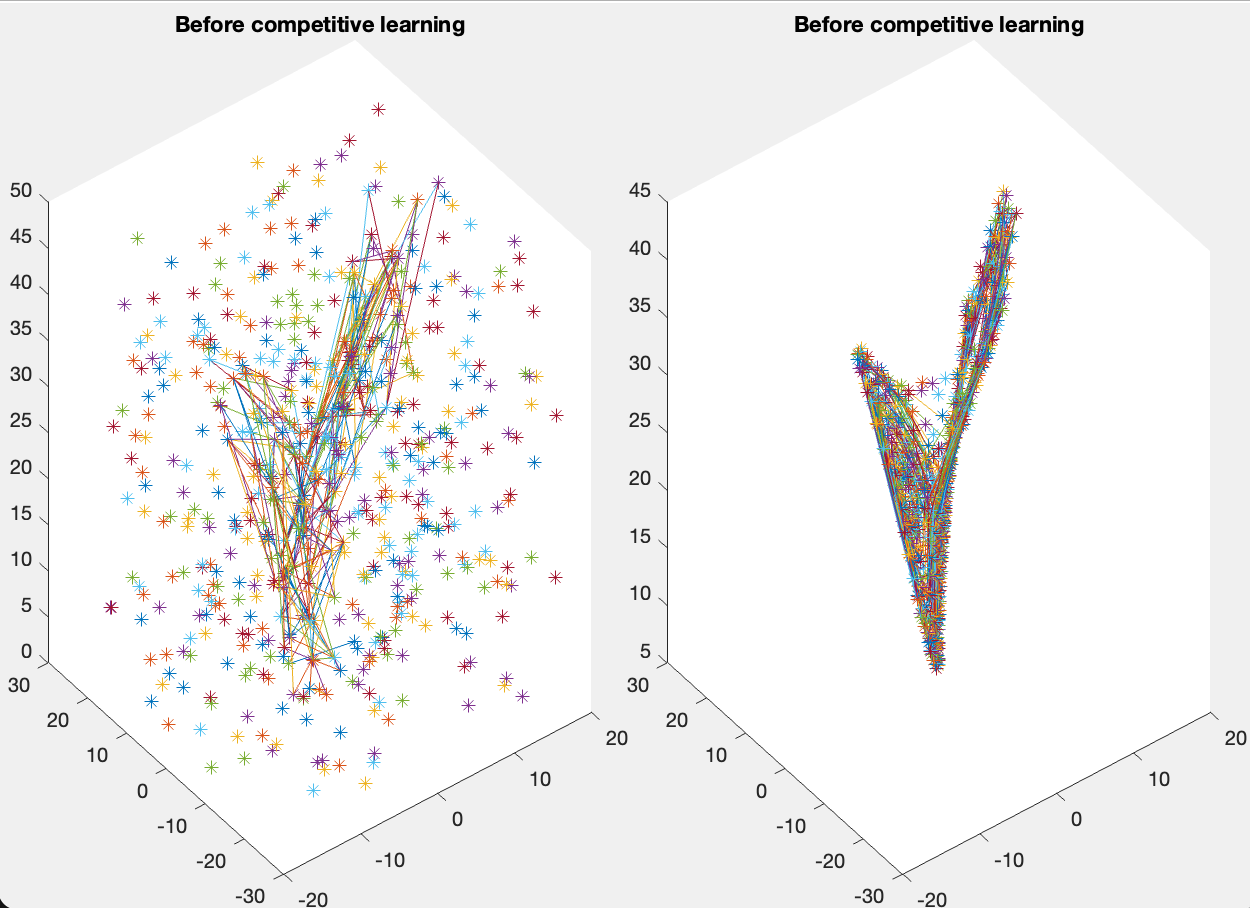
The process of color learning by SOM involves using the SOM algorithm to map the high-dimensional RGB color space onto a lower-dimensional space, while preserving the relative relationships between colors.

In the SOM algorithm, a set of input vectors (representing the colors in the RGB space) is presented to the network. The network consists of a 2D grid of neurons, each of which has a weight vector that represents a point in the lower-dimensional space. During training, the weights of the neurons are adjusted to be closer to the input vectors that are presented to the network.

The SOM algorithm uses a neighborhood function to determine which neurons in the grid should be updated in response to each input vector. Initially, the neighborhood function is large, so that many neurons in the grid are updated in response to each input vector. As training progresses, the neighborhood function becomes smaller, so that fewer neurons are updated in response to each input vector. This process of reducing the neighborhood function is called the "learning rate decay".

In the case of color learning, the SOM algorithm can be used to create a 2D map of colors that preserves the relative relationships between colors. For example, neighboring neurons in the SOM grid will have weight vectors that are similar to each other, and therefore represent similar colors. This means that colors that are similar to each other in the RGB space will be mapped to nearby neurons in the SOM grid, while colors that are dissimilar will be mapped to more distant neurons. This can be used for color visualization or clustering purposes, since colors that are visually similar will be mapped to nearby neurons in the SOM grid.

**Problem 2**:

Output:  


* The first image shows the weight matrix state before the start of the SOM algorithm, where each pixel represents a weight vector.
* The second figure shows the weight matrix state of the SOM algorithm after training. This figure shows how the SOM algorithm maps input data into a low dimensional space.
* The third figure shows the mapping of input data in the weight matrix after using the SOM algorithm. It demonstrates how the SOM algorithm maps input data to low dimensional spaces and can be used for clustering analysis and visualizing relationships between datasets.
* The fourth figure shows the weight matrix state of every 100 iterations during the training process of the SOM algorithm. It can help us understand how SOM gradually adjusts the weight matrix to adapt to input data, and observe the trend of weight matrix changes during the training process of SOM algorithm.

The process:  
1. Initialize the weight matrix: The code initializes the weight matrix W with random values, where each pixel represents a weight vector. In this example, the weight matrix has a size of 24x24.

2. Find the best matching unit (BMU) for input vectors in the weight matrix: For each input vector, the SOM algorithm computes the distance between the vector and all weight vectors in the weight matrix, finding the BMU as the weight vector closest to the input vector.

3. Update the weight matrix: For each input vector, the SOM algorithm updates its BMU and the weight vectors in its neighborhood within a certain range, gradually adjusting the weight matrix to preserve similarity between data points in the lower-dimensional space.

4. Repeat steps 2 and 3 until the weight vectors in the weight matrix no longer change significantly, or until a preset number of iterations is reached.

In this code, each pixel in the weight matrix represents a weight vector, which is randomly initialized. During the SOM algorithm's iteration, the weight vectors in the weight matrix are adjusted to preserve similarity between data points in the lower-dimensional space. Finally, we can observe the mapping of input data to the weight matrix after the SOM algorithm has learned, which can be used for clustering analysis and visualizing relationships between data sets, achieving dimensionality reduction.