Project4

1.Team member

- **m5268101 Liu Jiahe**
- **m5251140 Ken Sato**
- **m5271051 Keisuke Utsumi**

2.Team Project IV

- Using the Winner-Take-All algorithm to cluster the Iris dataset (http://www.ics.uci.edu/~mlearn/MLRepository.html).

3.Mathematical formulas

WTA (Winner-Take-All) is a competitive learning algorithm that follows the principle of updating only the winner (the neuron with the highest or lowest output) during each iteration while keeping the other neurons unchanged.

Assuming we have an input vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and a weight matrix $\mathbf{W}$, where each row of $\mathbf{W}$ corresponds to the weight vector of a neuron.

1.Number of Neurons

The number of neurons is equal to the number of clusters.

2.Weight Matrix Initialization

At the beginning of the WTA algorithm, we need to initialize the weight matrix $\mathbf{W}$.One common approach is to initialize the weights with small random values. This helps the algorithm explore the solution space better in the initial stages.

Here is an example of the weight initialization process:

$$W_{ij} = \text{rand}(n, m)$$

W_{ij} represents the element at the $i-th$ row and $j-th$ column of $\mathbf{W}$, $\text{rand}(n, m)$ 表generates a random matrix of size n rows adn m columns, where n is the number of neurons and , m is the dimension of the input vector $\mathbf{x}$.

3.Weight Matrix Normalization

Normalize the weight matrix $\mathbf{W}$, so that each row has a length of 1 (Euclidean norm or L2 norm). The purpose of this step is to ensure that all neuron weights are in the same order of magnitude in the initial state, avoiding learning instability caused by some weights being too large or too small.

Below is the process of weight matrix normalization:

$$\mathbf{W}_{i,:} = \frac{\mathbf{W}_{i,:}}{\|\mathbf{W}_{i,:}\|}$$

$\|\mathbf{W} * i, :\|$ represents the length (or norm) of the weight vector $\mathbf{W} * i, :$
For an n-dimensional vector x , its Euclidean norm can be represented as:

$$\|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \quad or \quad \|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{1}$$

This process involves operating on each row of the weight matrix $\mathbf{W}$. $\|\mathbf{W}\|_2$ denotes the calculation of the length of each row of the weight matrix $\mathbf{W}$, resulting in a vector.

$$\mathbf{W} = \frac{\mathbf{W}}{\|\mathbf{W}\|_2}$$

4.Updating the Weight Matrix

In each iteration, we need to compute the dot product of the input vector $\mathbf{x}$ and each row of the weight matrix $\mathbf{W}$, and then select the neuron with the largest dot product for update.

Here is an example of the weight matrix update process:

$$i^* = \arg\max_i \mathbf{x} \cdot \mathbf{W}_{i,:}$$
$$\mathbf{W}_{i^*,:} = \mathbf{W}_{i^*,:} + \alpha(\mathbf{x} - \mathbf{W}_{i^*,:})$$

i^* represents the index of the neuron with the largest dot product. Mathematically, the largest dot product represents the most consistent direction, which means the weight and the input pattern are most similar.

$\mathbf{W}_{i,:}$ represents the i-th row of the weight matrix $\mathbf{W}$, and α is the learning rate, which controls the step size of the weight update.

The Winner-Take-All algorithm only updates the weights of the neuron corresponding to i^*, while the weights of other neurons remain unchanged.

It's important to note that before updating the matrix, the training set needs to be normalized to make the range of its elements consistent with the range of the elements in the weight matrix. After updating the matrix, continue to use the L2 norm to normalize the weight matrix again.

When preprocessing the dataset, columns need to be normalized because the same feature exists across a column.

5.The strusture of Iris dataset

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
....
....
```

4.Implement code

project4.py

```python
def main():
    # load and pre_process data
    data = load_data('./iris.data')
    # initialize weights
    n_clusters = 8
    weights = initialize_weights(n_clusters, data.shape[1]) # data.shape =
(150, 4)
    # train WTA
    weights = train_wta(data, weights, n_epochs=400)
    # test WTA
    labels = test_wta(data, weights)
    # print(labels)
    visualize_3d(data, labels, n_clusters)

if __name__ == '__main__':
    main()

```

utils.py

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import csv
import random
```

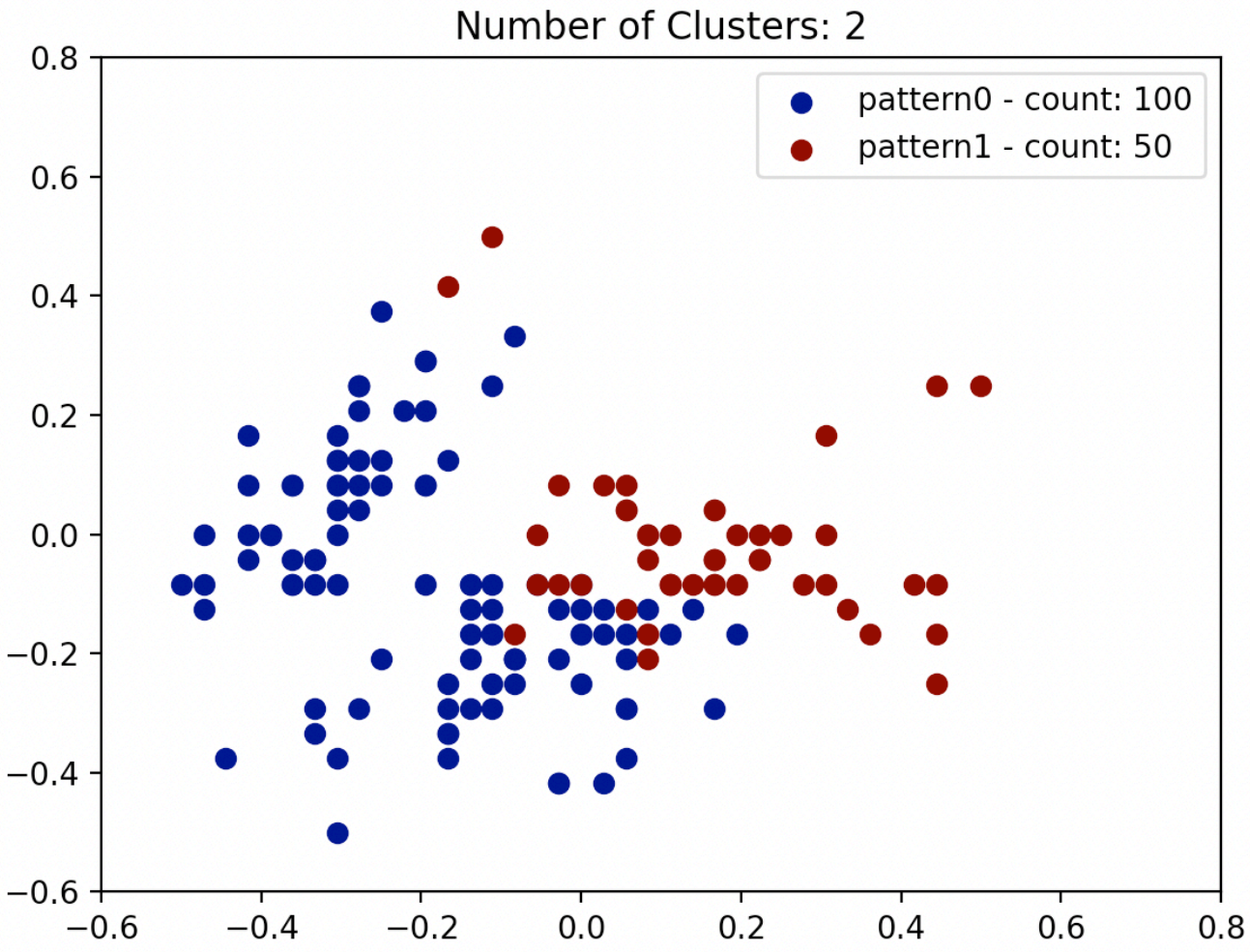
```
6
7  def load_data(filename):
8      with open(filename, 'r') as csvfile:
9          lines = csv.reader(csvfile)
10         dataset = list(lines)
11     dataset = [row for row in dataset if len(row) == 5]
12     for i in range(len(dataset)):
13         dataset[i] = dataset[i][:-1]
14     dataset = np.array(dataset, dtype=float)
15
16     # normalization
17     min_values = dataset.min(axis=0)
18     max_values = dataset.max(axis=0)
19     norm_dataset = (dataset - min_values) / (max_values - min_values) - 0.5
20
21     return norm_dataset
22
23
24
25 def initialize_weights(n_clusters, n_features):
26     weights = np.random.rand(n_clusters, n_features) - 0.5
27     # normalization
28     norm = np.linalg.norm(weights, axis=1, keepdims=True) # norm.shape = (3,
   1)
29     weights /= norm # weights.shape = (3, 4)
30     return weights
31
32 # train WTA
33 def train_wta(data, weights, alpha=0.5, n_epochs=20):
34     for epoch in range(n_epochs):
35         for x in data:
36             outputs = np.dot(weights, x) # (3, 4) inner product (4,
   ),outputs.shape = (3, )
37             winner = np.argmax(outputs)
38             # update weights
39             weights[winner] += alpha * (x - weights[winner])
40             # normalization
41             weights[winner] /= np.linalg.norm(weights[winner])
42     return weights
43
44 # test WTA, and return labels of each pattern
45 def test_wta(data, weights):
46     labels = []
47     for x in data:
48         outputs = np.dot(weights, x)
49         winner = np.argmax(outputs)
50         labels.append(winner)
51     return labels
52
53 def cluster_summary(data, labels, n_clusters):
54     cluster_counts = []
55     for i in range(n_clusters):
```

```
        cluster_data = data[np.array(labels) == i]
        count = len(cluster_data)
        cluster_counts.append(count)
        print(f'Pattern {i} includes {count} data points.')
    return cluster_counts

# visualization
def visualize_2d(data, labels, n_clusters):
    plt.figure()
    cmap_list = ['cool', 'hot', 'jet', 'viridis', 'rainbow', 'gray']
    cmap = plt.cm.get_cmap(random.choice(cmap_list), n_clusters)  # get color map
    cluster_counts = cluster_summary(data, labels, n_clusters)
    for i in range(n_clusters):
        cluster_data = data[np.array(labels) == i]
        plt.scatter(cluster_data[:, 0], cluster_data[:, 1], color=cmap(i), label=f'pattern{i} - count: {cluster_counts[i]}')
    plt.legend(loc='upper right')
    plt.xlim(-0.6, 0.8)
    plt.ylim(-0.6, 0.8)
    plt.title('Number of Clusters: {}'.format(n_clusters))
    plt.show()


def visualize_3d(data, labels, n_clusters):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    cmap_list = ['cool', 'hot', 'jet', 'viridis', 'rainbow', 'gray']
    cmap = plt.cm.get_cmap(random.choice(cmap_list), n_clusters)  # get color map
    cluster_counts = cluster_summary(data, labels, n_clusters)
    for i in range(n_clusters):
        cluster_data = data[np.array(labels) == i]
        ax.scatter(cluster_data[:, 0], cluster_data[:, 1], cluster_data[:, 2], color=cmap(i), label=f'pattern{i}: {cluster_counts[i]}')

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_xlim(-0.5, 0.6)
    ax.set_ylim(-0.5, 0.6)
    ax.set_zlim(-0.5, 0.6)
    ax.legend(loc='upper left', bbox_to_anchor=(1,1)) # set the uppder left cornor in (1, 1)
    ax.set_title('Number of Clusters: {}'.format(n_clusters))
    plt.show()

```

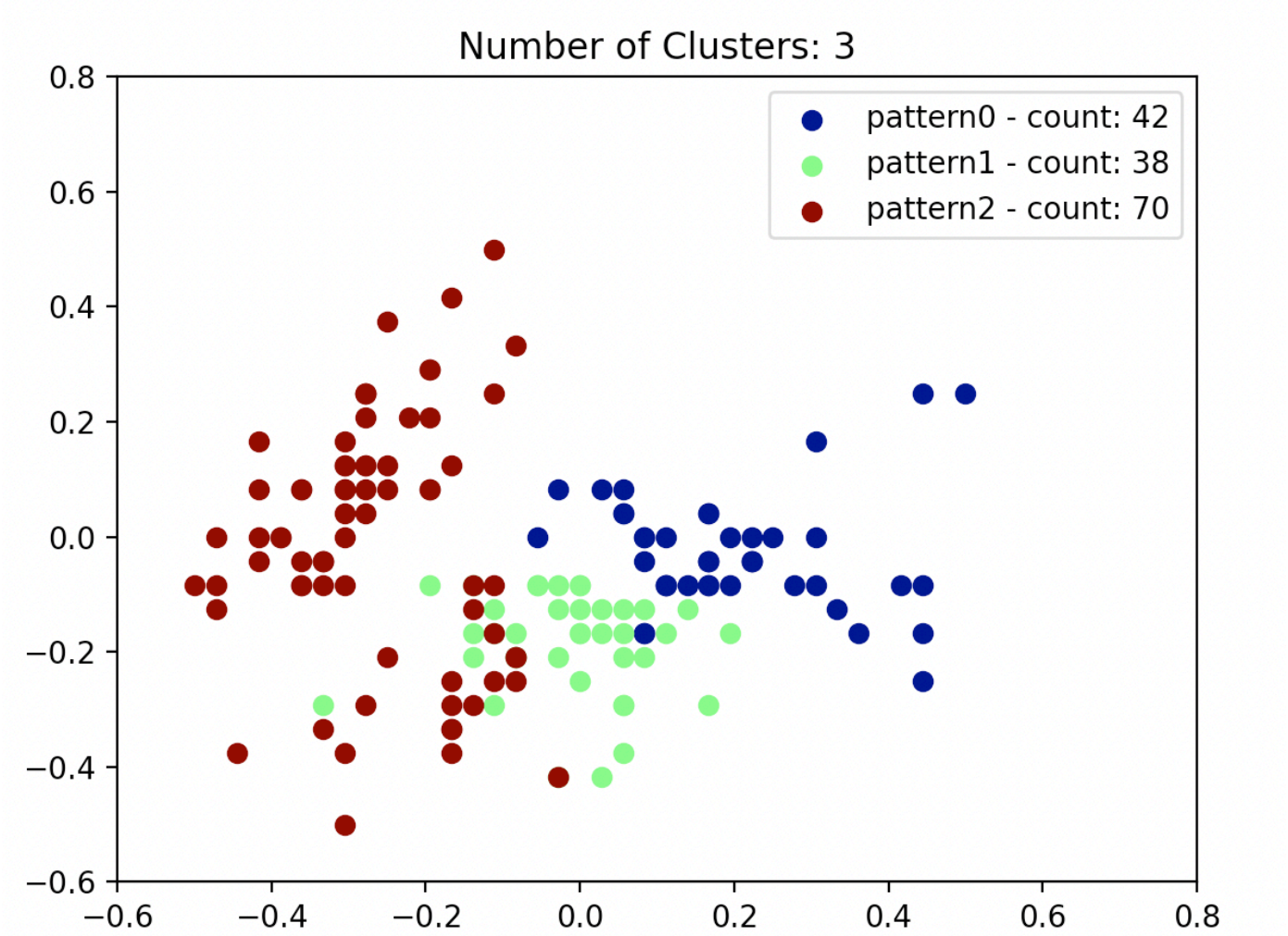
5.Results

Try setting different numbers of clusters to observe the effect of the neural network. When the number of clusters is small, a two-dimensional image is sufficient. However, when the number of clusters is large, different classes may overlap in two-dimensional space. In such cases, use a three-dimensional image.

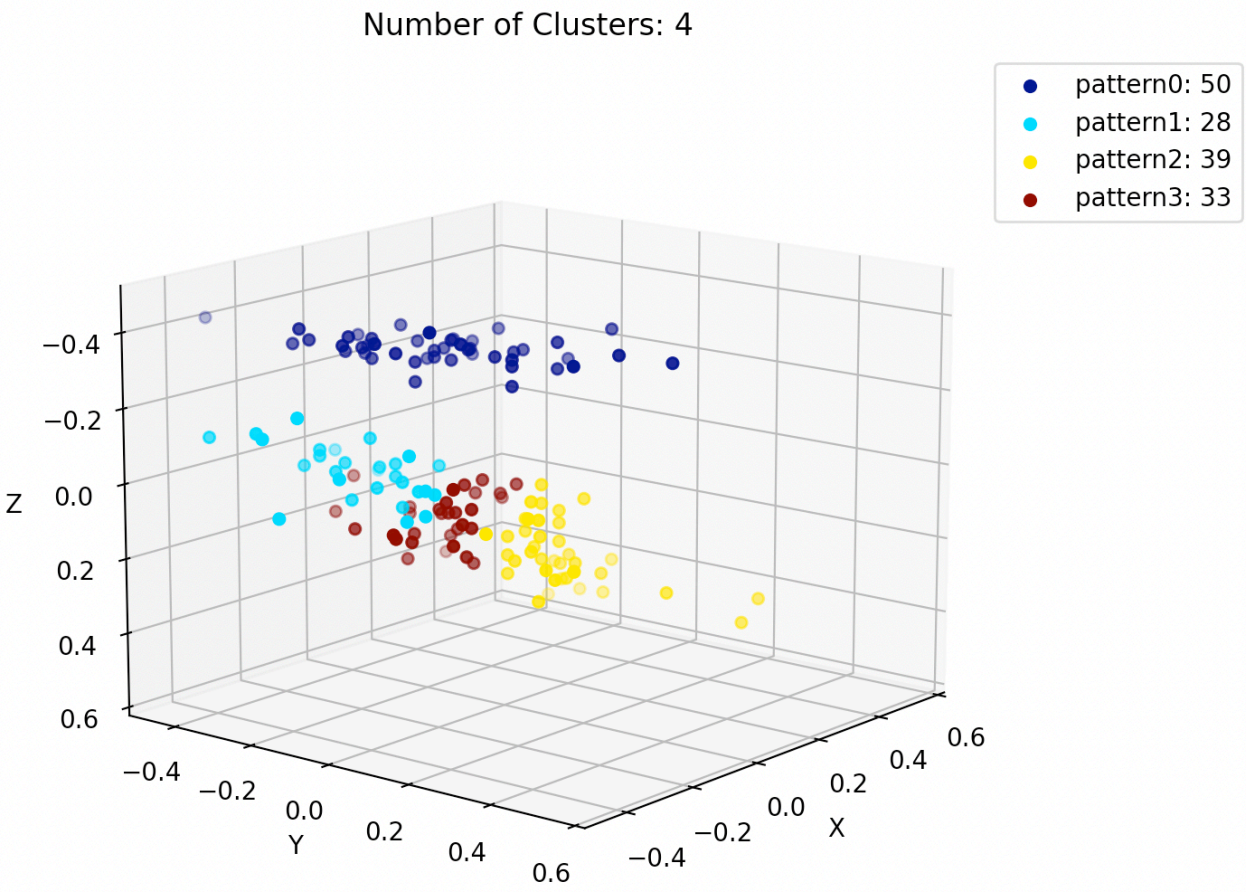
1.n_clusters = 2



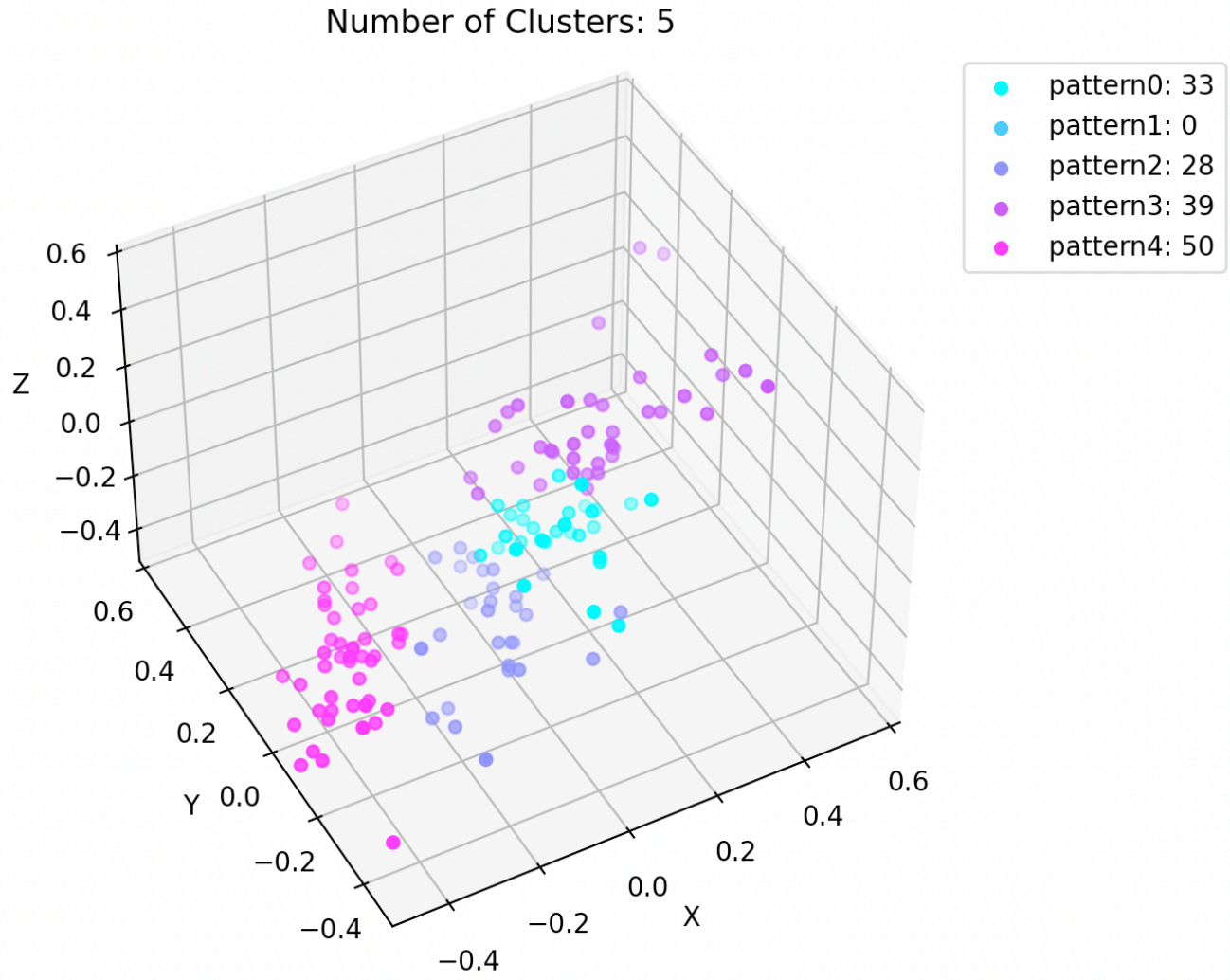
2.n_clusters = 3



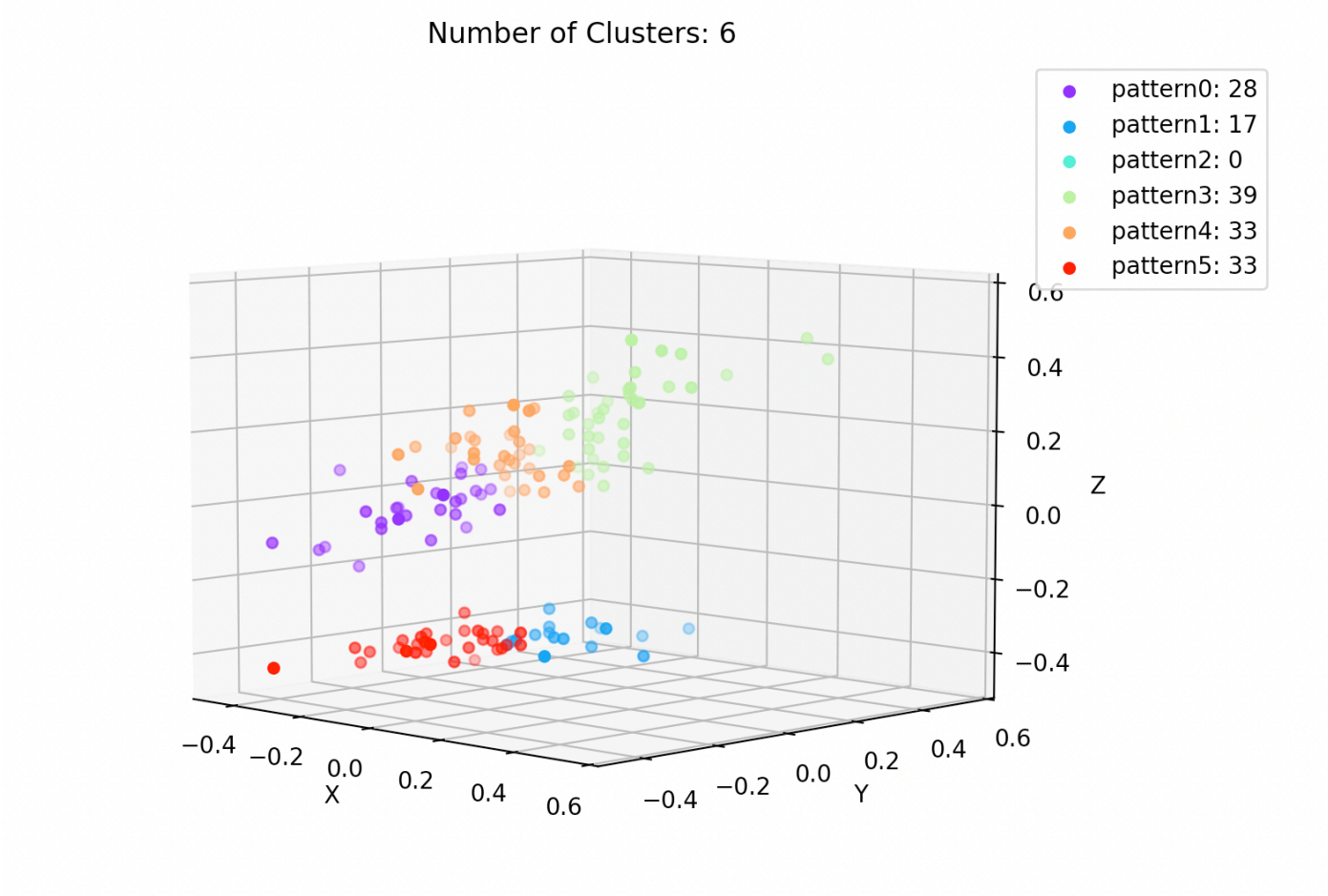
3.n_clusters = 4



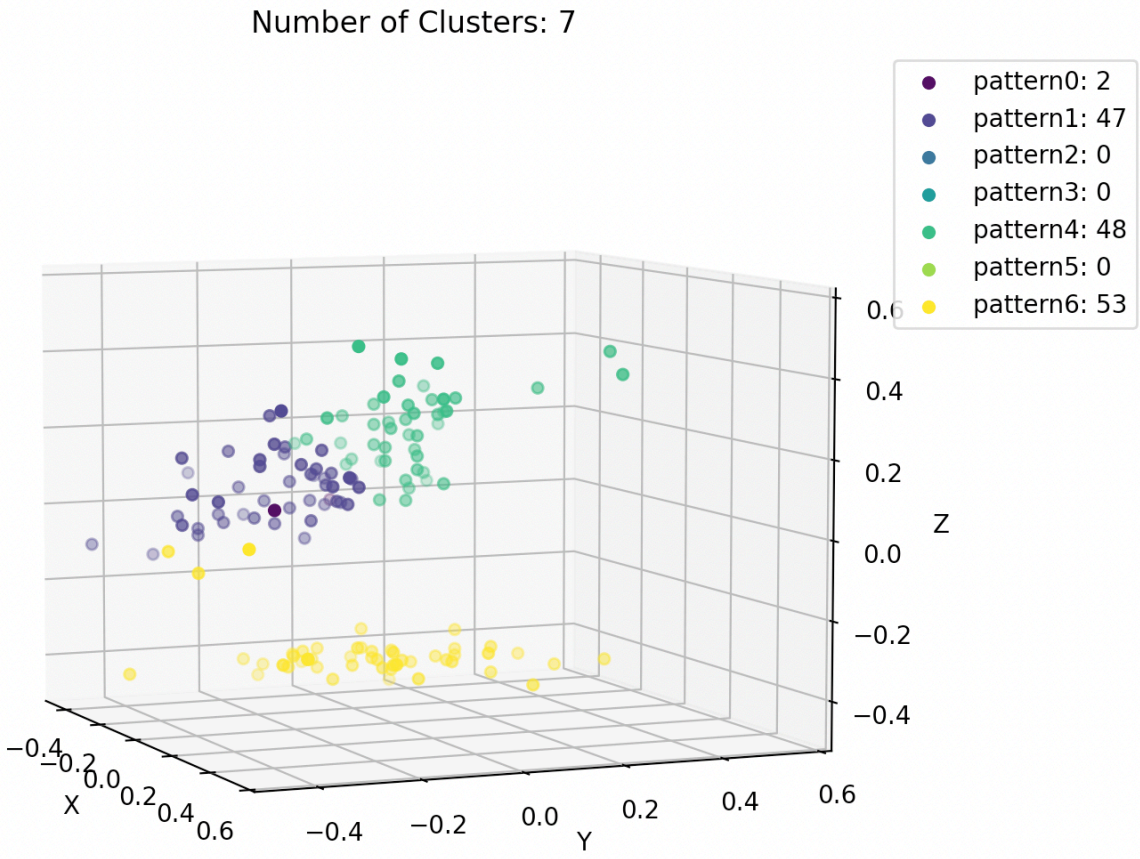
4.n_clusters = 5



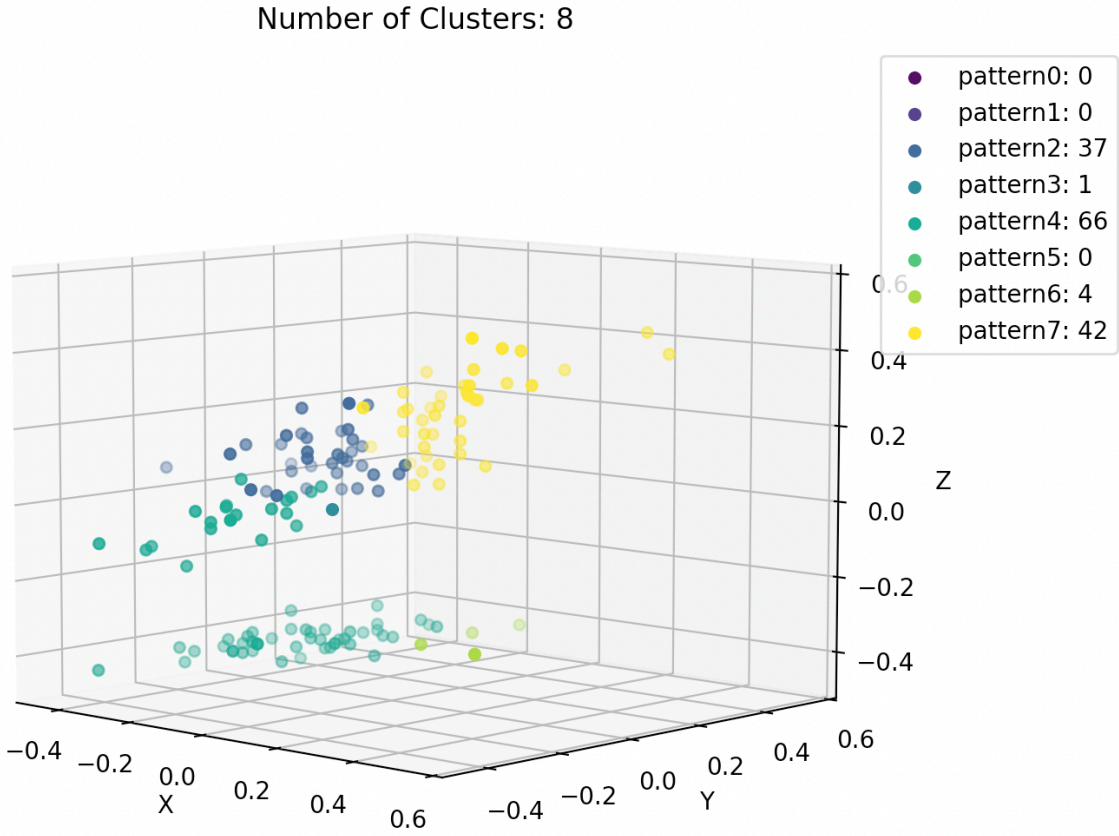
5.n_clusters = 6



6.n_clusters = 7



7.n_clusters = 8



8.Conclusion

From the above images, it can be seen that for the Iris dataset, the WTA (Winner-Take-All) neural network has a good clustering capability. It's also noticed that even as the number of 'n_clusters' increases, the actual number of types of classes doesn't necessarily increase. That is, the real number of clusters in the data may be less than the number of clusters we set. This indicates that the number of intrinsic distribution patterns in the Iris dataset itself is limited. Judging intuitively from the above images, the number of classes determined by the intrinsic data structure of Iris is probably between 2-5.