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.数学原理

WTA（Winner-Take-All）是一种竞争学习算法，其主要思想是在一次迭代中，只有赢家（即输出最大或者最小的那个）被更新，其他神经元保持不变。

假设我们有一个输入向量 $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ 和一个权重矩阵 $\mathbf{W}$，其中 $\mathbf{W}$ 的每一行对应一个神经元的权重向量。

1.神经元个数

神经元的个数等于聚类的个数

2.权重矩阵初始化

在 WTA 算法的开始，我们需要初始化权重矩阵 $\mathbf{W}$。一种常用的方法是将权重初始化为小的随机值。这样可以帮助算法在初始阶段更好地探索解空间。

以下是权重初始化过程的示例：

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

在这里，W_{ij} 表示权重矩阵 $\mathbf{W}$ 中的第 i 行第 j 列的元素，$\text{rand}(n, m)$ 表示生成一个 n 行 m 列的随机矩阵，其中 n 是神经元的数量，m 是输入向量 $\mathbf{x}$ 的维度。

3.权重矩阵归一化

对权重矩阵 $\mathbf{W}$ 进行归一化，使得每一行的长度为 1（欧几里得范数，L2范数）。这样做的目的主要是为了让所有神经元的权重在初始状态下处于同样的量级，避免某些权重过大或过小导致的学习过程不稳定。

以下是权重矩阵归一化的过程：

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

在这里，$\|\mathbf{W} * i, :\|$ 表示权重向量 $\mathbf{W} * i, :$ 的长度（或范数）。

对于一个n维向量x，其欧几里得范数可以表示为：

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

这个过程需要对权重矩阵 $\mathbf{W}$ 的每一行进行，在这里，$\|\mathbf{W}\|_2$ 表示对权重矩阵 $\mathbf{W}$ 的每一行计算长度，结果是一个向量。

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

4.权重矩阵更新

在每次迭代中，我们需要计算输入向量 $\mathbf{x}$ 和权重矩阵 $\mathbf{W}$ 的每一行的点积，然后选择点积最大的那个神经元进行更新。

以下是权重矩阵更新过程的示例：

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

在这里，i^* 表示点积最大的那个神经元的索引，从数学上看，点积最大代表着方向最一致，也就是权重和输入的模式最相似

$\mathbf{W}_{i,:}$ 表示权重矩阵 $\mathbf{W}$ 中的第 i 行，α 是学习率，它控制了权重更新的步长。

Winner-Take-All算法只有 i^* 对应的那个神经元的权重被更新，其他神经元的权重保持不变。

需要注意的是，在更新矩阵前需要对训练集进行归一化使其元素的范围和权重矩阵元素的范围一致，更新矩阵后继续使用L2范数对权重矩阵再次归一化。

对数据集预处理的时候，要对列进行正则化，因为同一列具有相同的特征

5.Iris数据集结构

```
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.代码实现

project4.py

```
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

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

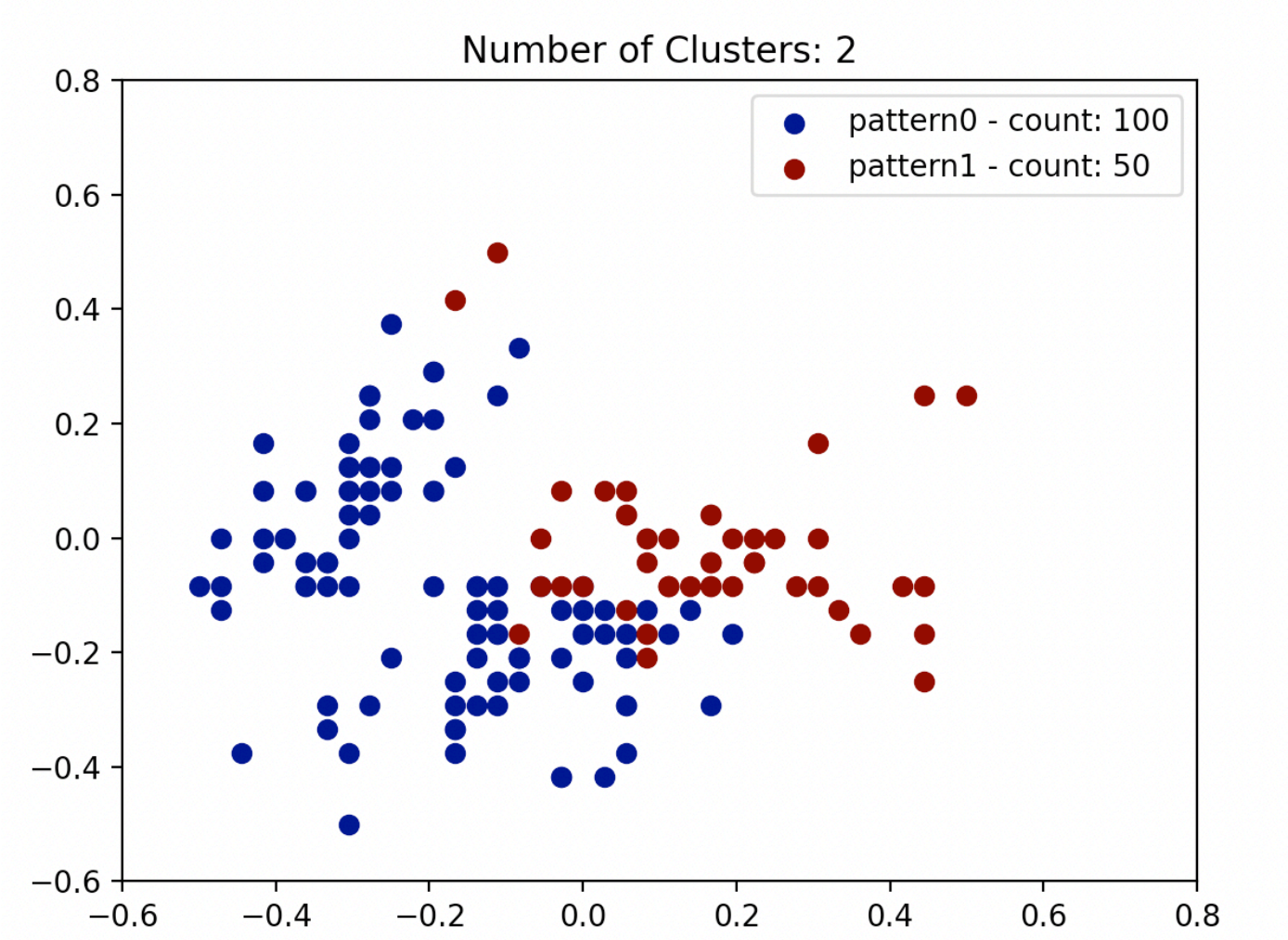
```
import csv
import random

def load_data(filename):
    with open(filename, 'r') as csvfile:
        lines = csv.reader(csvfile)
        dataset = list(lines)
    dataset = [row for row in dataset if len(row) == 5]
    for i in range(len(dataset)):
        dataset[i] = dataset[i][:-1]
    dataset = np.array(dataset, dtype=float)

    # normalization
    min_values = dataset.min(axis=0)
    max_values = dataset.max(axis=0)
    norm_dataset = (dataset - min_values) / (max_values - min_values) - 0.5

    return norm_dataset



def initialize_weights(n_clusters, n_features):
    weights = np.random.rand(n_clusters, n_features) - 0.5
    # normalization
    norm = np.linalg.norm(weights, axis=1, keepdims=True) # norm.shape = (3,
1)
    weights /= norm # weights.shape = (3, 4)
    return weights

# train WTA
def train_wta(data, weights, alpha=0.5, n_epochs=20):
    for epoch in range(n_epochs):
        for x in data:
            outputs = np.dot(weights, x) # (3, 4) inner product (4,
),outputs.shape = (3, )
            winner = np.argmax(outputs)
            # update weights
            weights[winner] += alpha * (x - weights[winner])
            # normalization
            weights[winner] /= np.linalg.norm(weights[winner])
    return weights

# test WTA, and return labels of each pattern
def test_wta(data, weights):
    labels = []
    for x in data:
        outputs = np.dot(weights, x)
        winner = np.argmax(outputs)
        labels.append(winner)
    return labels

def cluster_summary(data, labels, n_clusters):
```

```
    cluster_counts = []
    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()

```

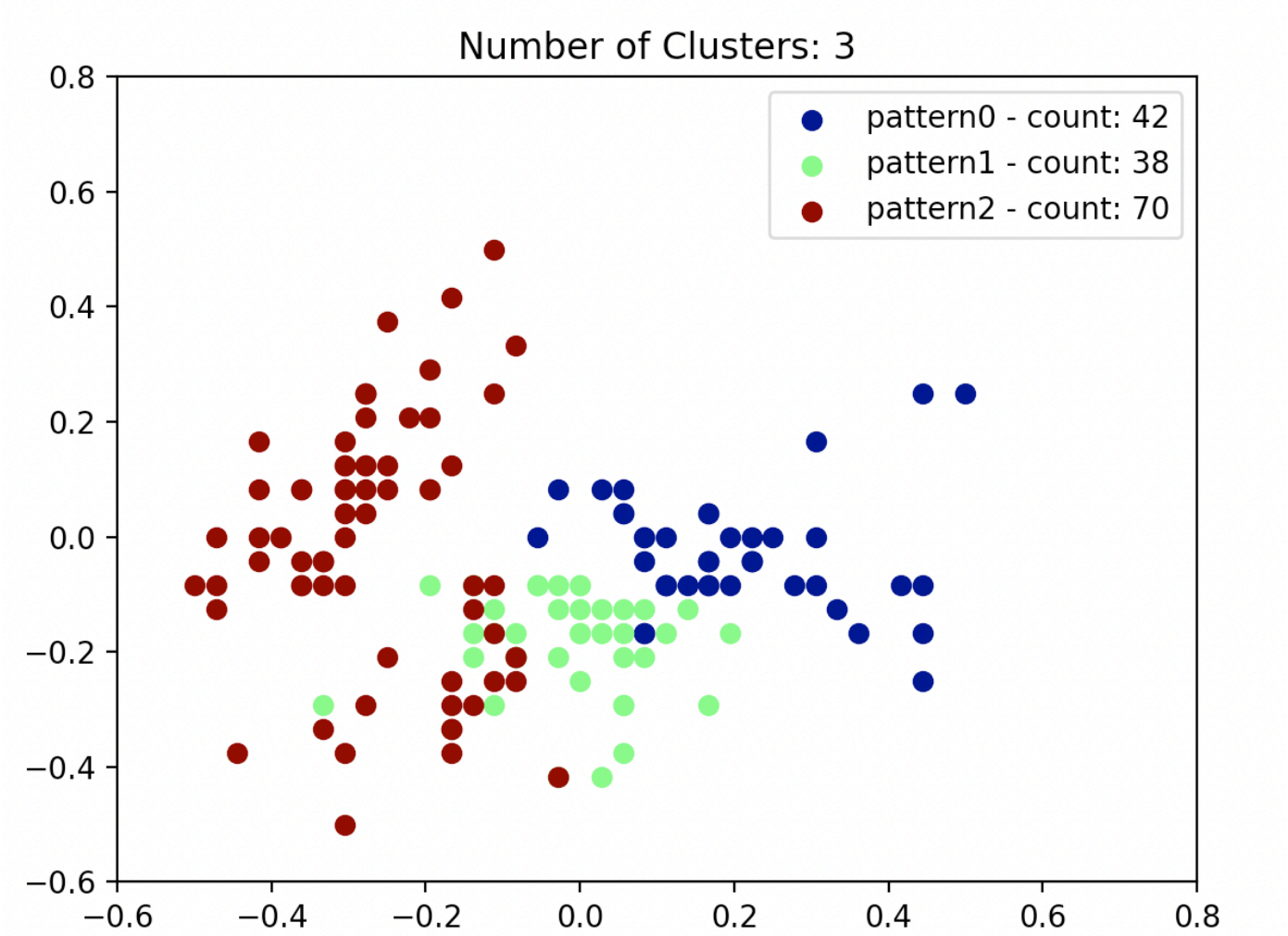
5.结果讨论

尝试取不同聚类数量观察神经网络的效果，当聚类数量较少时二维图像足够，当聚类数量较多不同的类可能会在二维空间重叠，此时采用三维图像

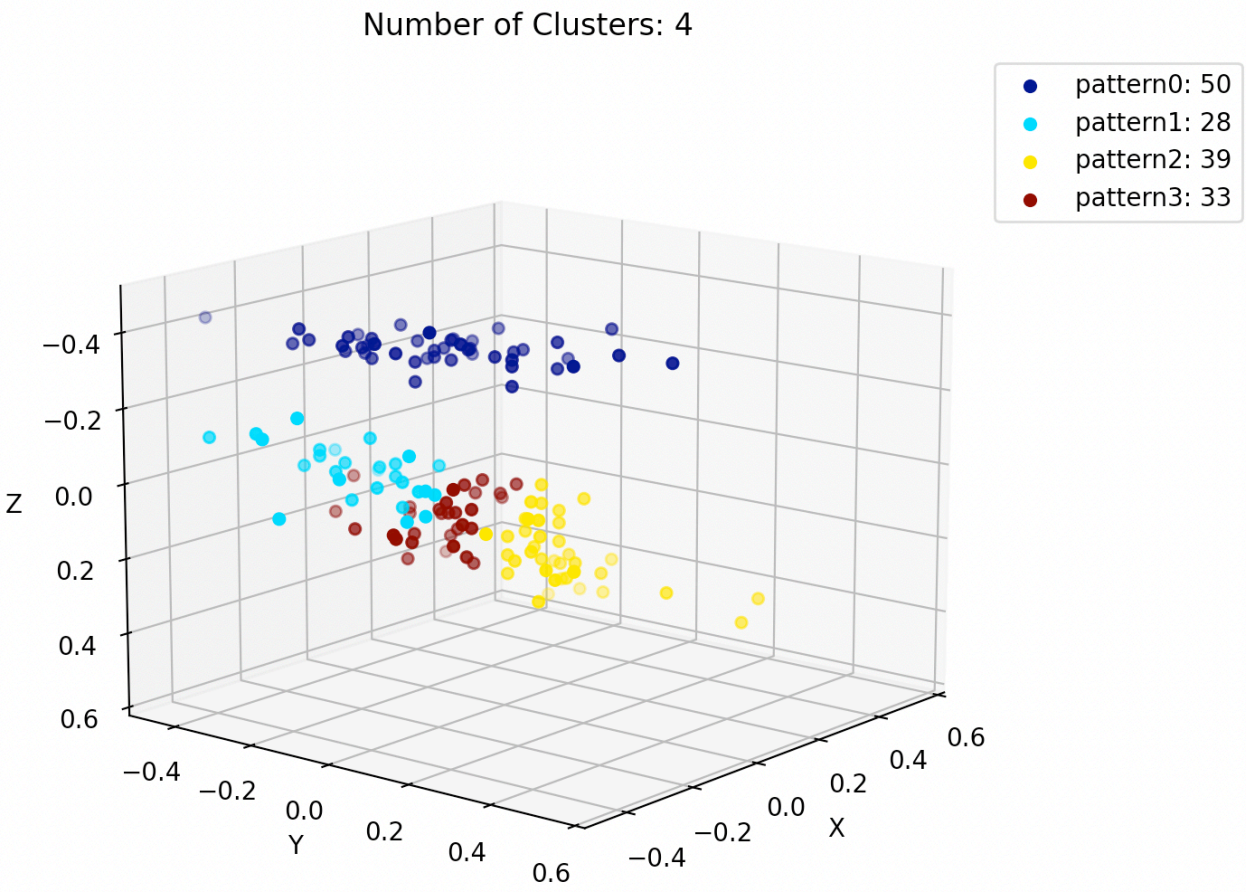
1.n_clusters = 2



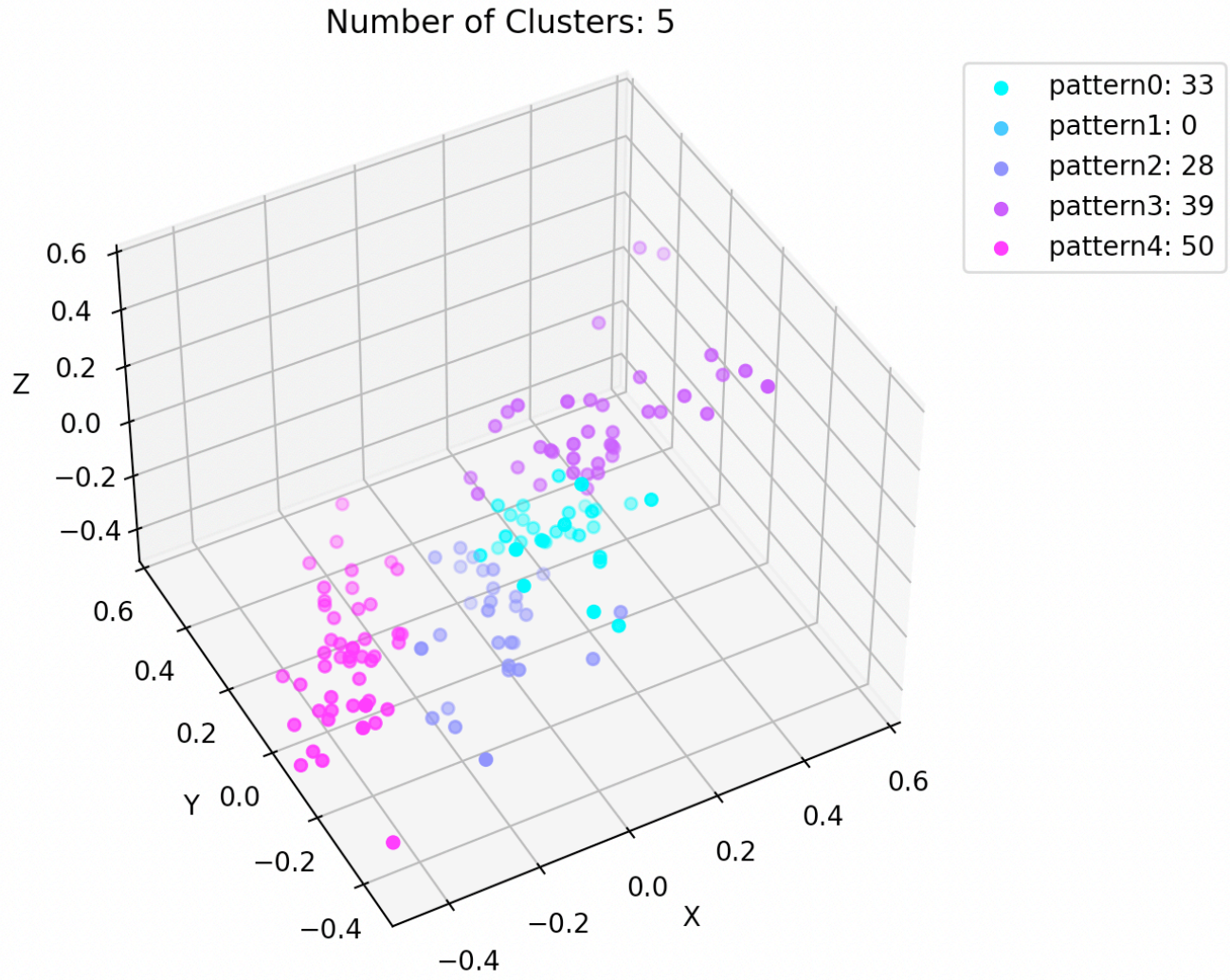
2.n_clusters = 3



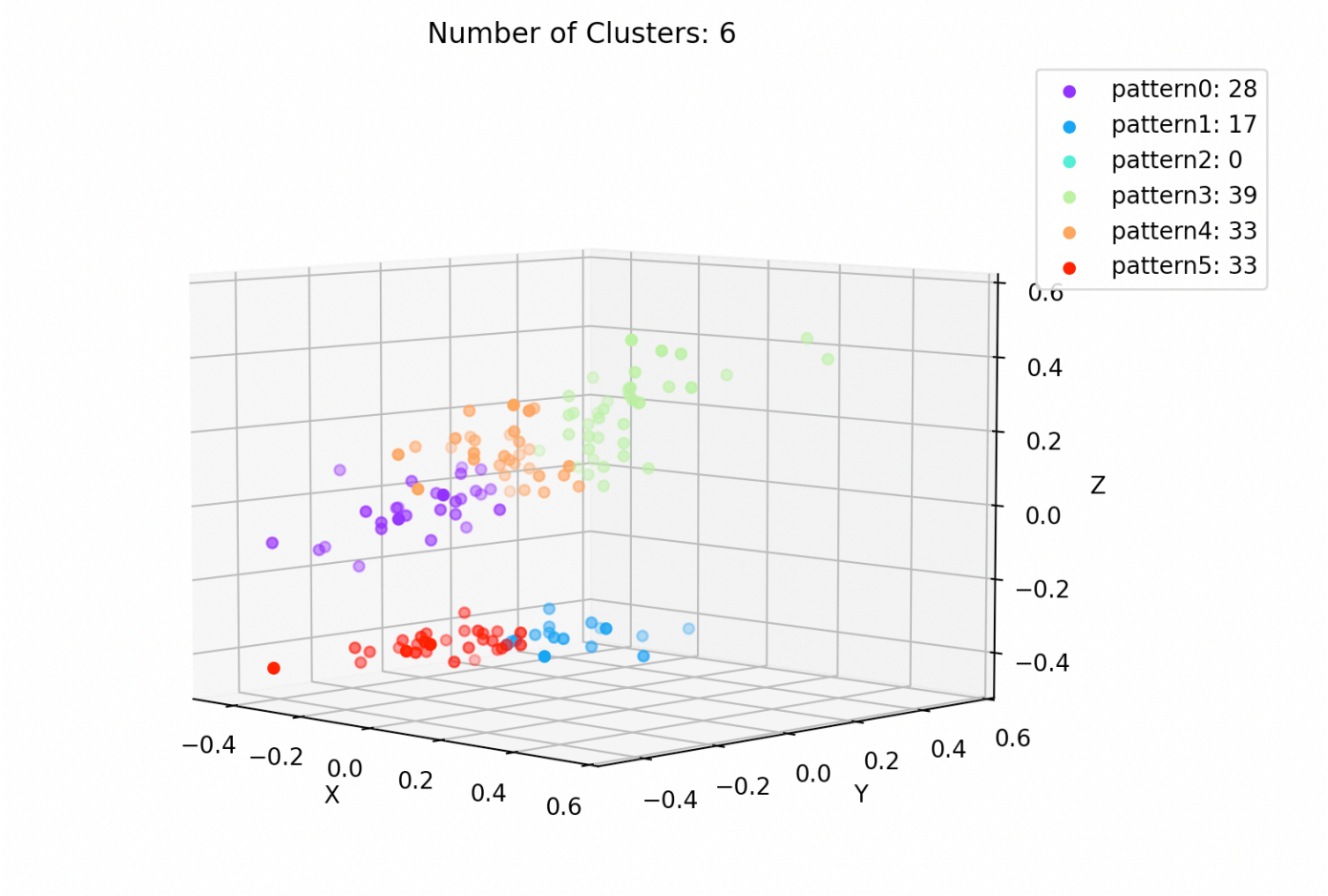
3.n_clusters = 4



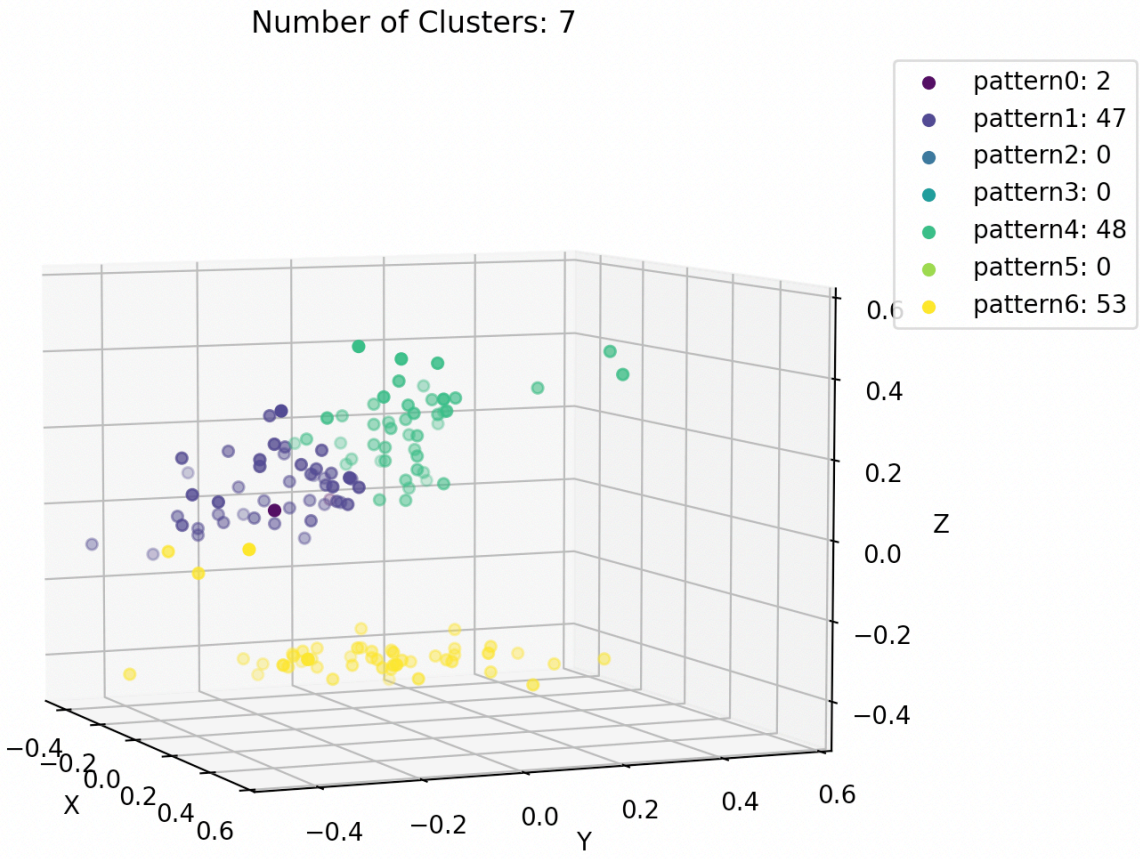
4.n_clusters = 5



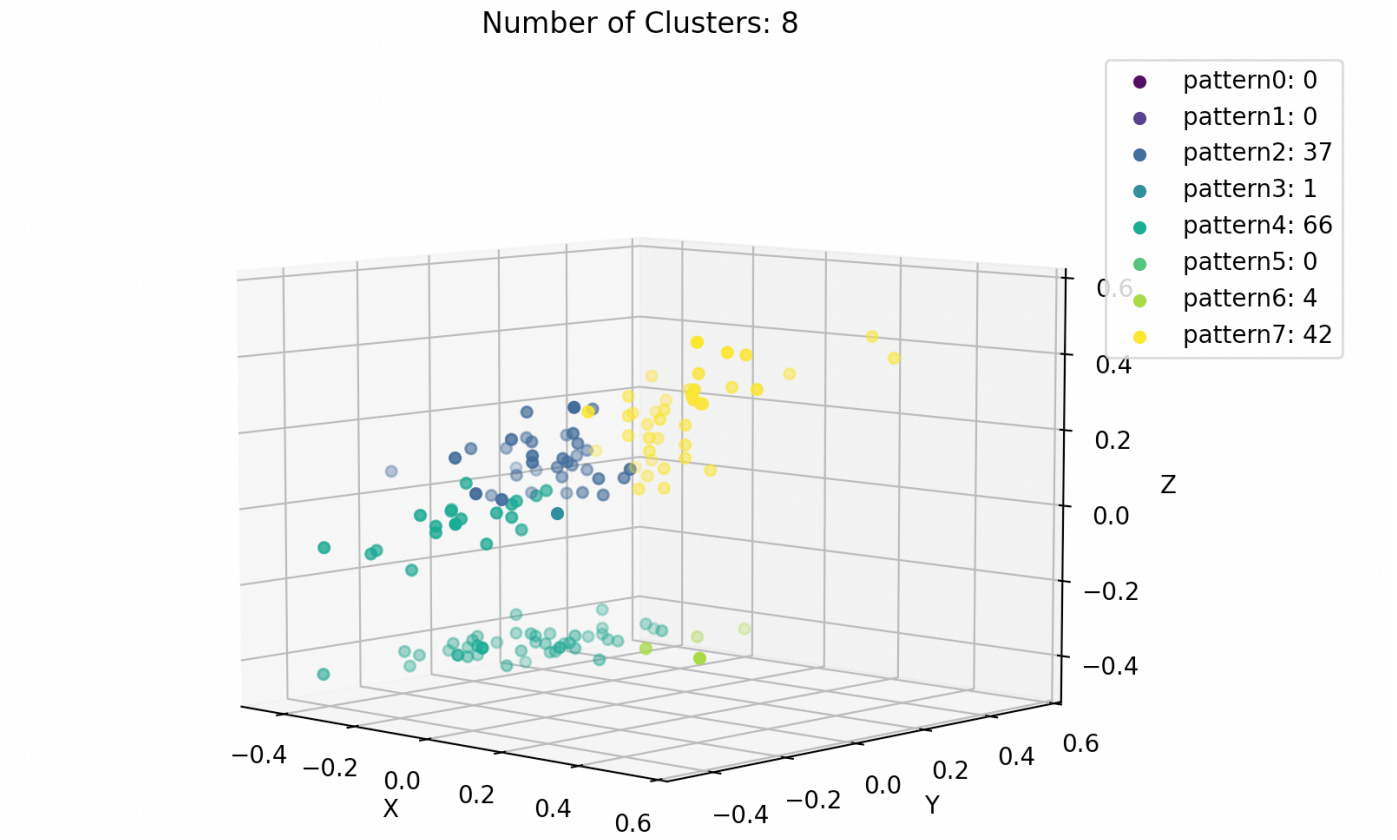
5.n_clusters = 6



6.n_clusters = 7



7.n_clusters = 8



8.结论

从上面的图像可以看出，对于Iris数据集，WTA神经网络具有很好的聚类能力。同时也注意到，即使n_clusters增多，但是实际上类别的种类也不一定增多，即数据中的真实聚类数目可能小于我们预设的聚类数目。这说明Iris数据集本身的内在分布模式的数量是有限的。根据上面的图像直观判断，Iris内在数据结构决定的类别数量大概为2-5类。