**2022 – 2023 第一学期 实验报告**

课程编号： 2801000060， 课程名称：机器人学导论， 主讲教师： 邱国平

|  |  |  |
| --- | --- | --- |
| 学号：2022110131 | 姓名：廖祖颐 | 专业年级：电子信息工程 |

**Coursework 1 （20%）**

|  |
| --- |
| 教师评语/成绩 |

Catalogue

[Brief Introduction Of Experimental Principle 2](#_Toc155436775)

[Experimental Process 5](#_Toc155436776)

[Experiment Summary 17](#_Toc155436777)

# Brief Introduction Of Experimental Principle

The perceptron is a simple linear binary classification algorithm and forms the foundation of artificial neural networks. The goal of the perceptron is to find a decision boundary that can divide the input data into two parts.

**Components**

1. **Input Weights**: Each input node is associated with a weight. These weights are the parameters that are adjusted during the learning process.
2. **Weighted Sum**: The sum of the products of input values and their corresponding weights.
3. **Activation Function**: A function that determines whether a neuron should be activated. In the perceptron, a step function is typically used. (Due to insufficient time in this experiment, this was not further optimized.)
4. **Output**: If the weighted sum exceeds a certain threshold, the output is 1; otherwise, it is 0.

**Working Principle**

1. **Initialize Weights**: Initially, weights are set to small random numbers.
2. **Compute Output**: For each input, calculate the weighted sum and produce an output through the activation function.
3. **Update Weights**: If the output is incorrect, adjust the weights based on a predefined learning rate. The adjustment of weights is based on the perceptron rule:



*η* is the learning rate.

1. **Repeat Process**: This process is repeated over multiple iterations or until the error rate is acceptable.

**Limitations**

* **Linear Separability**: The perceptron can only deal with linearly separable data sets. For nonlinear problems, it cannot converge to a solution and thus cannot successfully train a "black box".
* **Single-Layer Structure**: As it is a single-layer structure, it can learn simple patterns such as the AND operation, but the perceptron cannot learn complex patterns, such as the XOR problem.

**Applications**

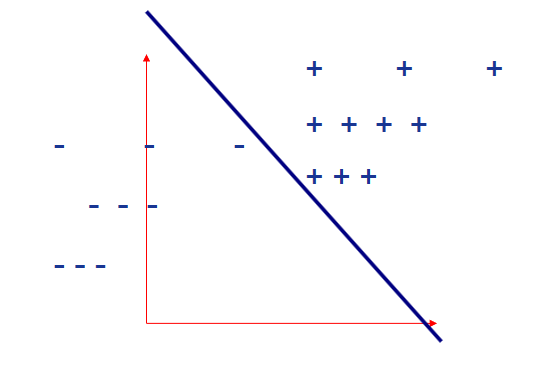
Despite its limitations, the perceptron is very important for understanding the basic concepts of neural networks and is still effective for some simple linearly separable problems.

In these more advanced networks, by adding multiple layers and using different activation functions, it is possible to solve nonlinear problems that the perceptron cannot, such as dividing different numbers later in this task. In fact, multiple single-layer perceptrons can be used to divide, you just need to write a single-layer perceptron and then call it repeatedly.

# Experimental Process

Before applying the perceptron, it is necessary to train with a large dataset. This training process involves labeling the data in the training set and continually adjusting the weights based on the characteristics of the data to more accurately fit the dataset's features. As for the features of the dataset, a feature can be understood as a combination of numerical values that can completely and uniquely represent a data point. For example, in the MNIST dataset of handwritten digits, each image is composed of 28x28 pixels, totaling 784 pixels. This combination of 784 pixel values provides a unique representation for each image, constituting the features that distinguish the MNIST handwritten digits.

One challenge we may face is distinguishing the number 6 from other numbers, or identifying the numbers 2 and 5. It is important to understand that single-layer perceptrons are best applied to datasets that can be divided by a straight line (or in high-dimensional space, a hyperplane), which are called linearly separable data.

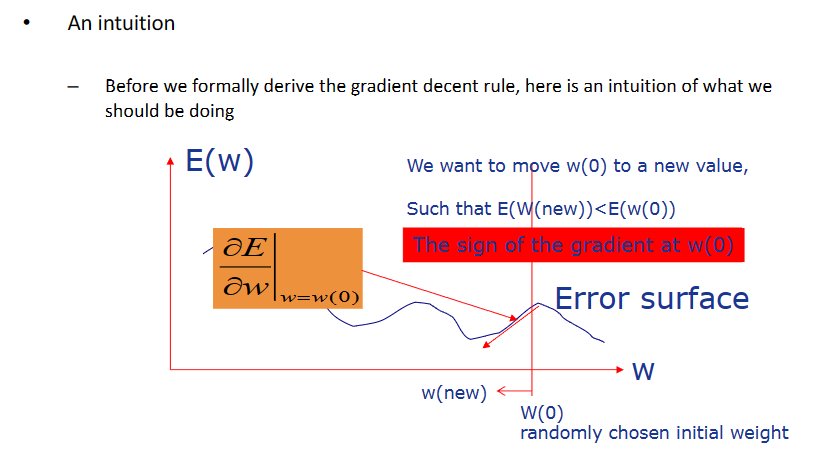


The process of training the perceptron for effective classification involves four key steps:

1. **Weight Initialization:** We start with small random numbers, assigning an initial weight to each feature.
2. **Computation and Classification:** For each data point, we calculate the weighted sum of its feature values and use a sign function to predict the category. If the weighted sum is positive, we predict one class (for example, +1); if negative, another class (for example, -1).
3. **Weight Adjustment:** We adjust the weights according to the model's prediction results and the actual labels. This is done through a simple mathematical formula that includes the learning rate, a crucial parameter that determines the speed and extent of our weight adjustments.
4. **Iterative Optimization:** By repeating the steps above, we optimize the weights until the model's predictions are as close as possible to the actual labels.

Ideally, we want to find a set of weights that results in the lowest classification error rate. This can be achieved by observing the gradient of the loss function. The loss function is a mathematical function that describes the size of the prediction errors, and we adjust the weights by calculating the gradient to reduce these errors.

To visualize this concept, we can imagine a three-dimensional space where the x and y axes represent the weights and the z-axis represents the value of the loss function. Our goal is to find the global minimum of the loss function, which is like finding the lowest point in a mountain range. By using the gradient descent algorithm, we "walk down the hill" along the steepest downhill route, getting closer to the lowest point with each step.



This process requires careful adjustment because a learning rate that is too high might cause us to "step over" the lowest point, while a learning rate that is too low can slow down the training process to an unacceptable degree. Through continuous iteration and adjustment, we ultimately find a line (or in high-dimensional space, a hyperplane) that can accurately divide the dataset into two classes.

In the performance evaluation of classification models, the confusion matrix is a key tool. It shows in a tabular form the comparison between the model's predicted results and the actual situations. The confusion matrix is composed of four parts, each reflecting different types of prediction situations: the number of true positives (TP) shows how many cases of the positive class the model predicted correctly; the number of false positives (FP) indicates how many cases the model incorrectly marked as the positive class; the number of false negatives (FN) is how many cases of the positive class the model incorrectly marked as the negative class; and the number of true negatives (TN) is how many cases of the negative class the model predicted correctly.

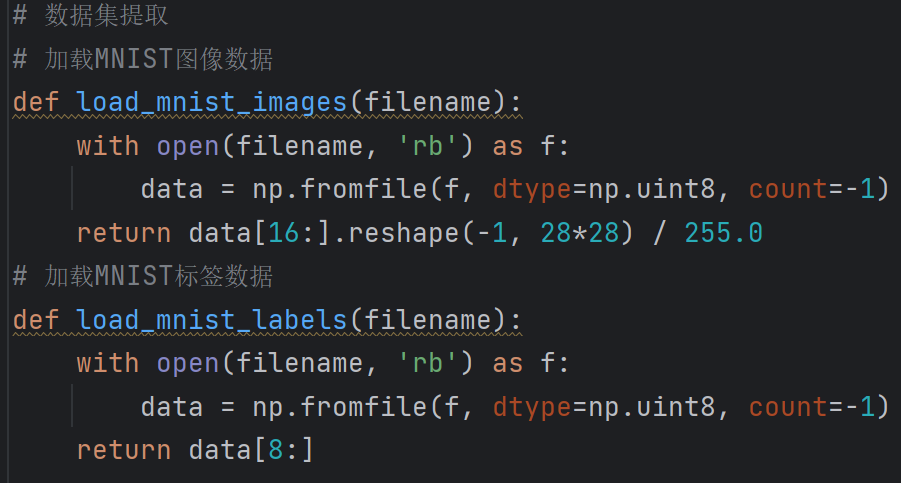
The confusion matrix not only allows us to directly observe the accuracy of the model's predictions and misjudgments but also to calculate a series of evaluation metrics such as accuracy, recall, precision, and the F1 score. The true positive rate (TPR) reflects the proportion of real positive cases correctly identified by the model, calculated as TP / (TP + FN). The proportion at which the model incorrectly identifies actual negative cases as positive is called the false discovery rate (FDR), calculated as FP / (FP + TP).



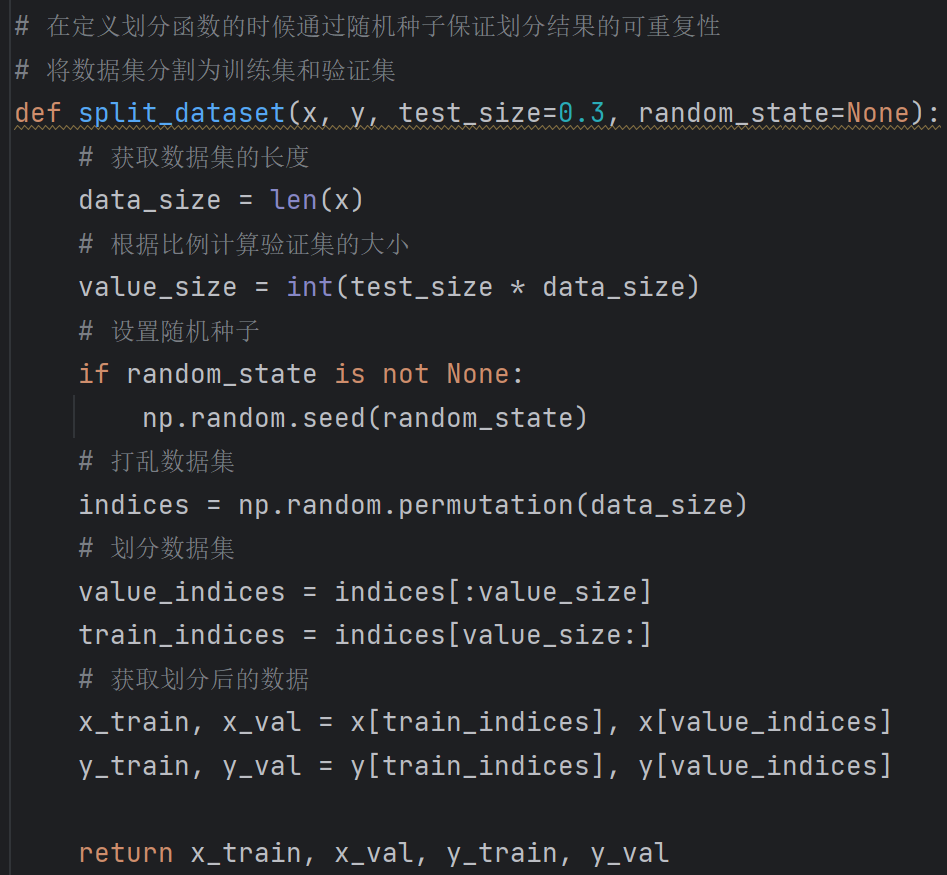
The confusion matrix not only provides an intuitive assessment of model performance but also allows us to delve into the reasons that may cause certain types of misjudgments. This enables us to identify and specifically improve the weaknesses of the model, further enhancing the precision of its predictions.

Many subsequent tasks involve multi-class classification, and I believe that implementing multi-class classification is actually just multiple calls to the SLP function for specific types of classification; the principle is very simple.

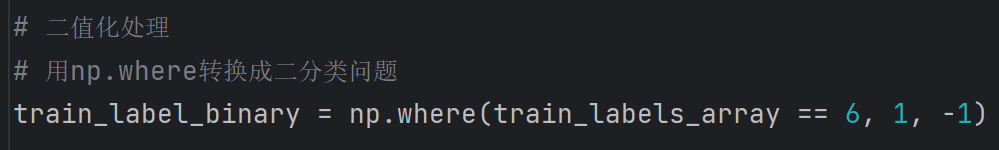
**1.** **Reading the Dataset:** The MNIST dataset has a unique structure where the image file contains a header of 16 bytes with a magic number and the number of images, and the label file has a header of 8 bytes with a magic number, which are not part of the actual data needed. Therefore, it's necessary to skip these bytes to access the relevant image and label data. Here's a snippet of code to extract the useful information:



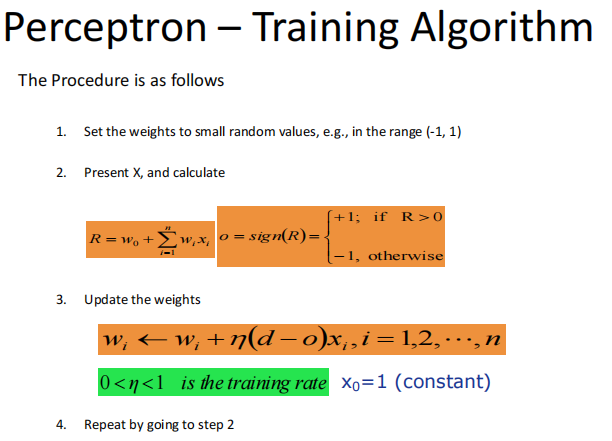
**2.Dividing the Dataset:** The MNIST dataset comes pre-divided with 60,000 training images and 10,000 testing images. However, if you need to further divide the dataset, you can simply shuffle the data randomly and then split it according to the desired ratio. To ensure reproducibility, you can set a random.seed. Below is the code for dataset division:

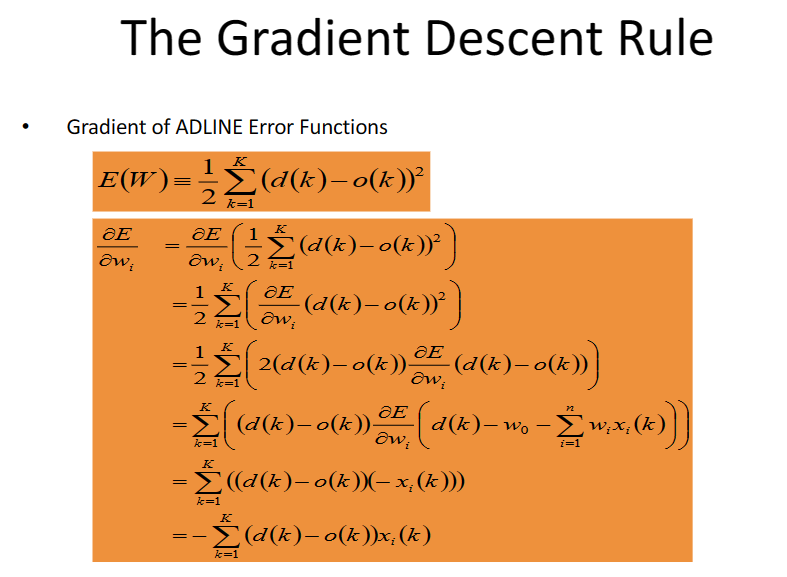
****

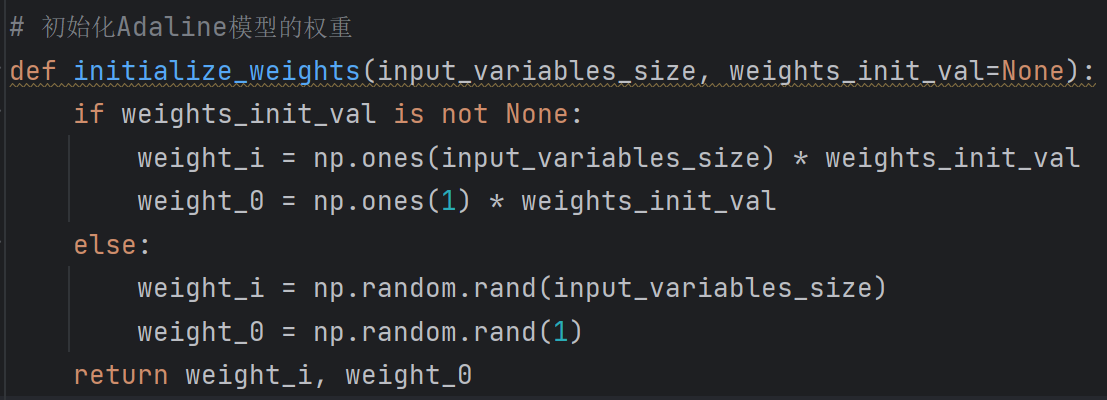
**3. Label Division for the Dataset:** For a binary classification task like identifying the digit '6' from others, label division can be easily done using np.where. For multi-class classification, you would need to perform specific calculations based on the requirements for dividing the digits. The calculations are straightforward, often involving simple arithmetic like multiplying by 2 and subtracting 1 to distinguish between +1 and -1. Below is the approach for handling a binary classification problem:

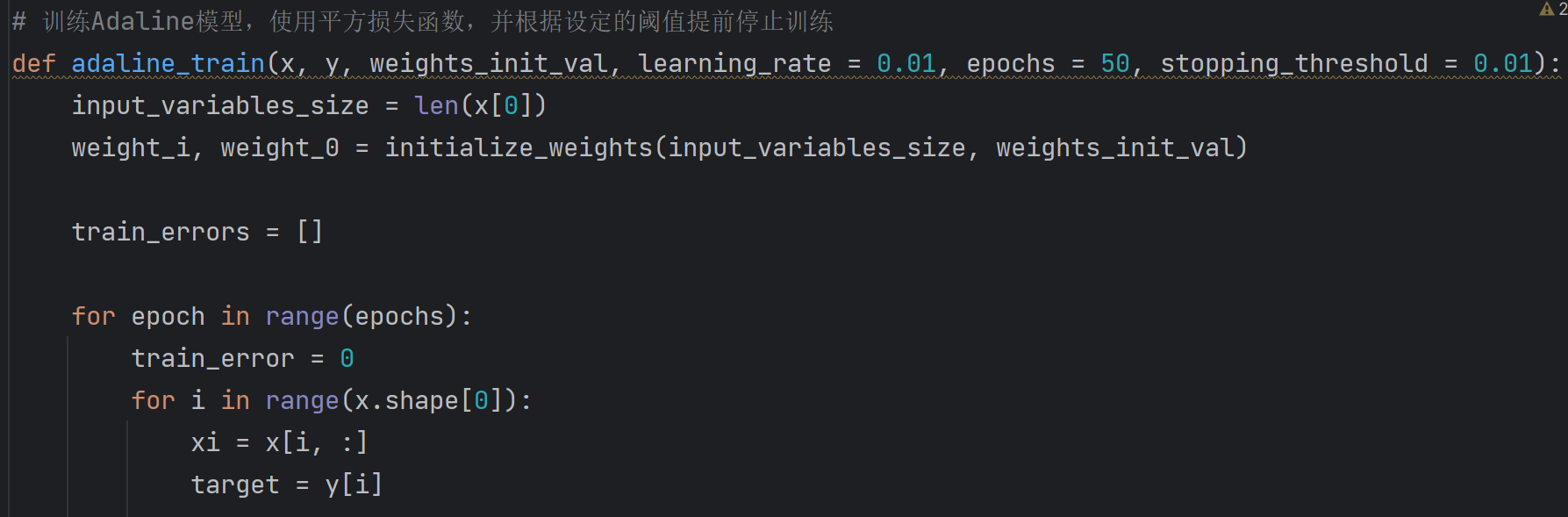
****

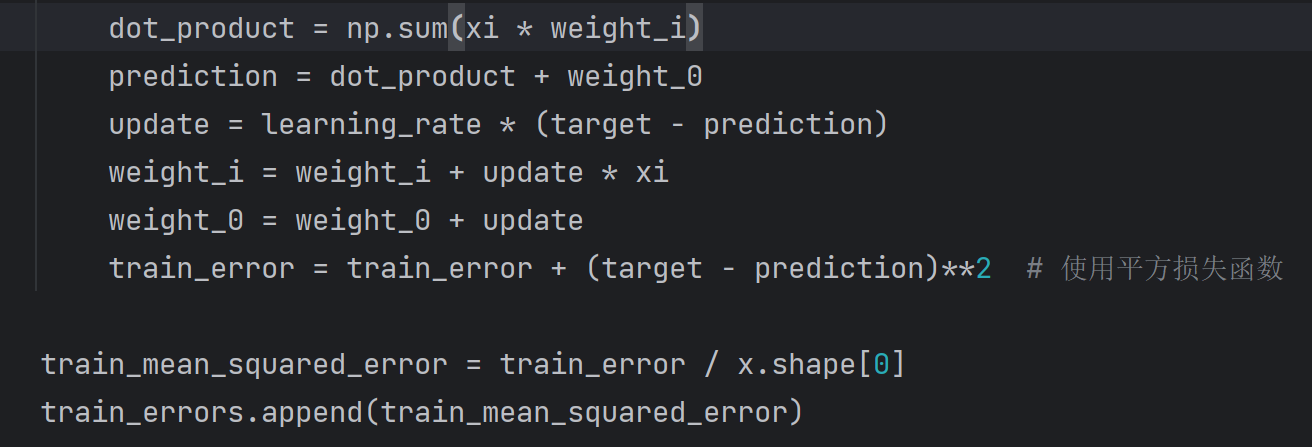
**4.ADLINE Training function**



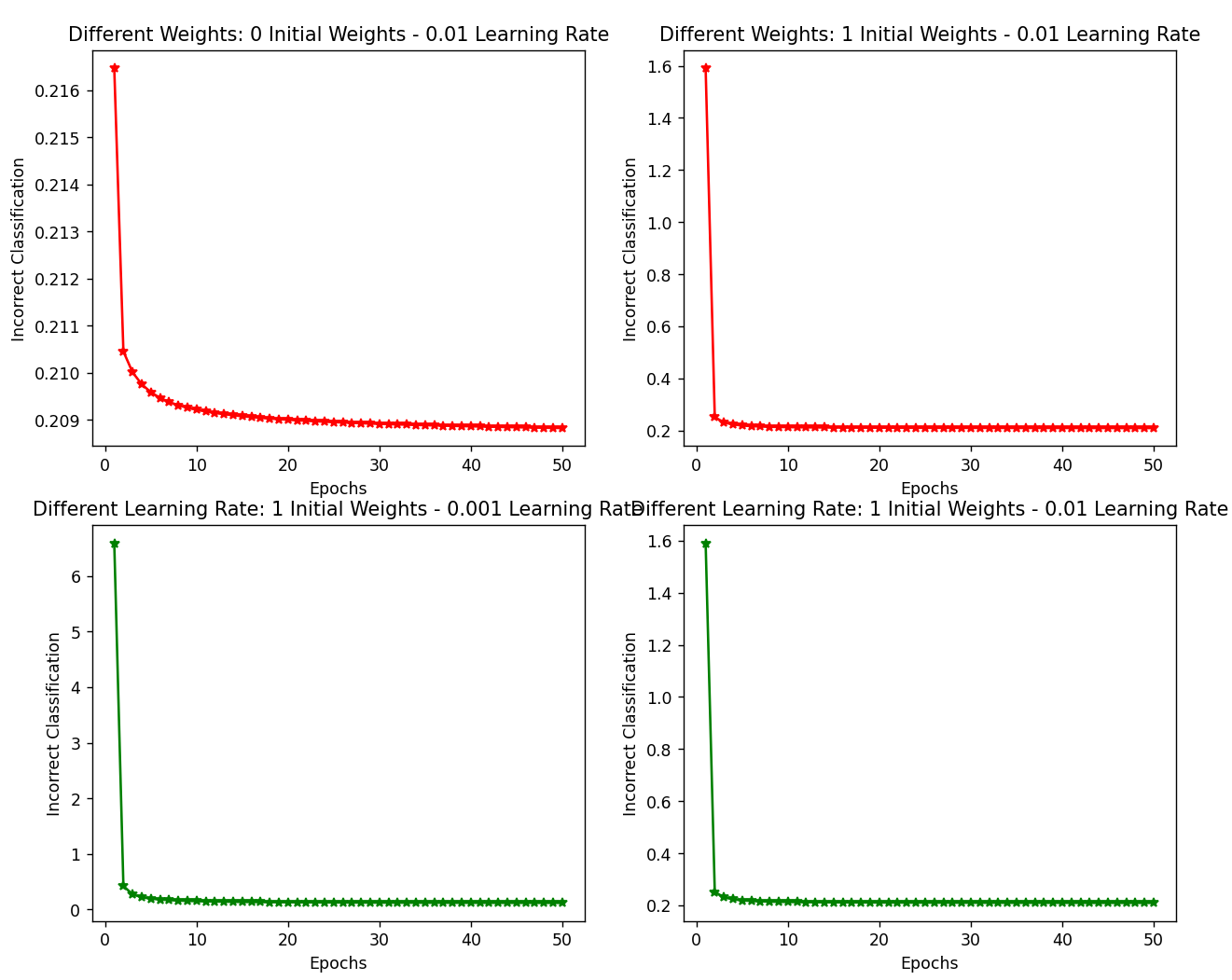
****

The following code can be implemented according to the algorithm of the above two graphs

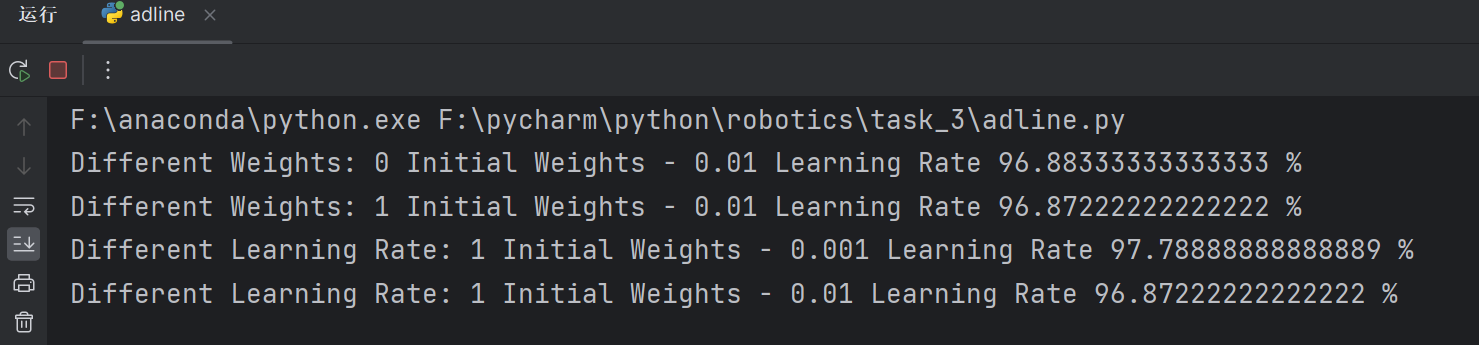


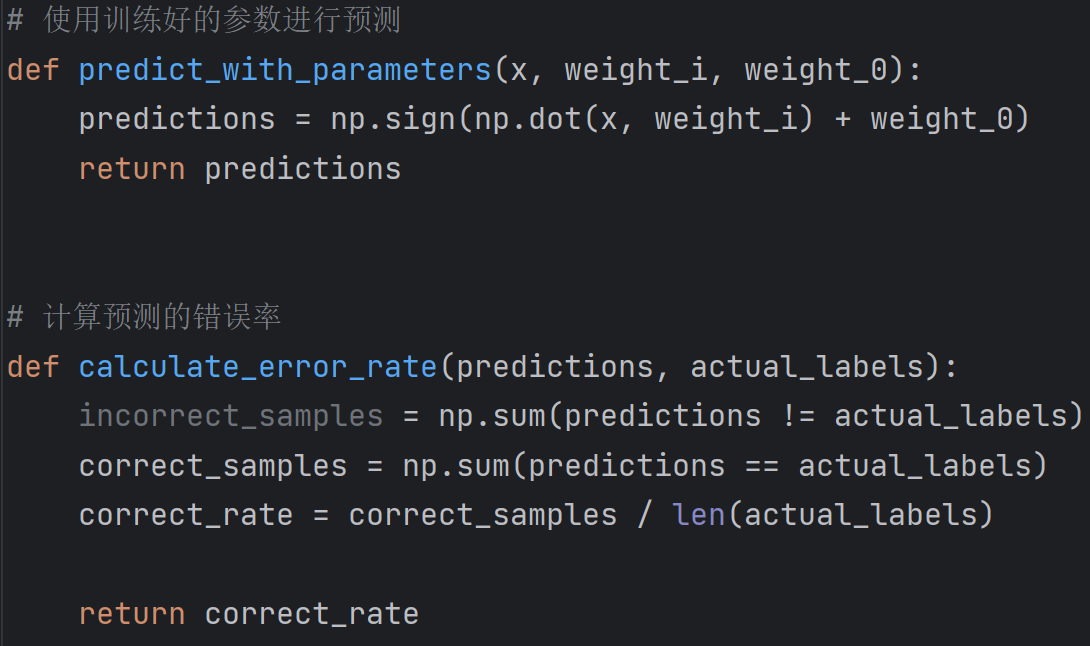


**5.** **Training result**

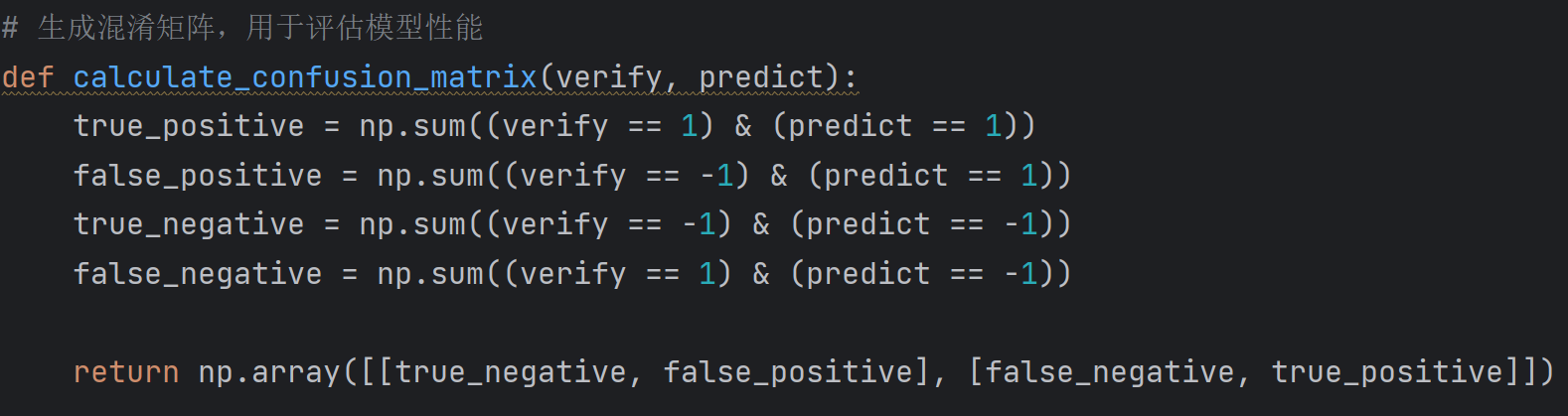
****

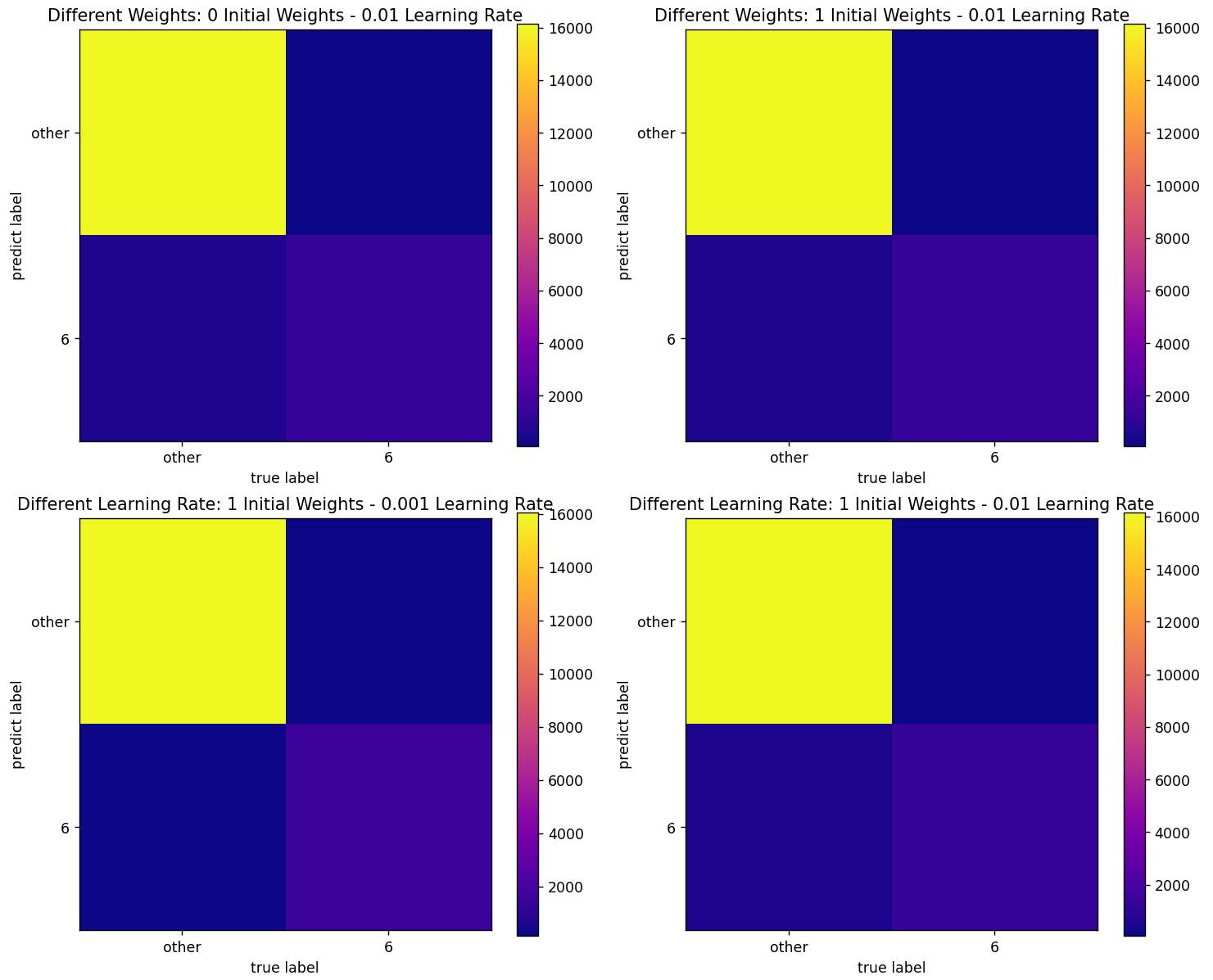
**6.** **Validate set test results**

****

**The data is close to 100%, indicating that the training effect is very good**

**7.** **Confusion matrix performance evaluation**

****

****

* **The top-left corner (True Positive, TP) represents the number of '6' digits that were correctly identified as '6'.**
* **The bottom-right corner (True Negative, TN) represents the number of non-'6' digits (others) that were correctly identified as not being '6'.**

**The top-left and bottom-right corners represent the differentiation between the number 6 and other numbers, respectively. Since the dataset does not contain a large number of the digit '6', the distinction in the bottom-right corner is not very high. However, it is still visually apparent that '6' and other numbers have been correctly and perfectly distinguished.**

# Experiment Summary

1. **Initial Weight Selection:** After a series of experiments, we found that when the number of training iterations is sufficient, the initial values of the weights have little impact on the final outcome. This is because the model will eventually converge to the unique local minimum of the loss function. As long as the training is adequate, the optimal weight configuration can always be found, regardless of how the initial weights are set.
2. **Assessment of Training Effectiveness:** By analyzing data from multiple experiments, we observed that the error rate in all experiments gradually decreases with an increase in the number of training iterations. Particularly when the initial weights are set to zero or one, the rate of decrease in the error rate is rapid and quickly stabilizes, which further proves the existence of an optimal solution for weight configuration. However, in some cases, if the learning rate is too high, the model may oscillate near the minimum error threshold instead of converging, highlighting the importance of setting the appropriate learning rate.
3. **Summary and Conclusion of the Experiment:**

* **Impact of Initial Weights and Learning Rate:** The experiments show that initial weights and learning rate significantly affect the training process and the ultimate performance of the model. A smaller learning rate helps in quickly reaching the minimum error threshold and maintaining stability after reaching it.
* **Multi-layer Perceptrons and Multiclass Classification:** Further experiments suggest that by stacking single-layer perceptrons, we can build multi-layer perceptrons to accomplish more complex multi-class classification tasks.