**Team Project I: Learning of a single neuron and single layer neural networks**

**Part 2: Learning of single layer neural networks**

1. **Introduction**

This report explores the extension of neural network learning mechanisms to a **single-layer neural network** consisting of **multiple neurons**, with support for both **discrete** and **continuous output**. Building on the foundation of single-neuron learning, this stage focuses on handling **multi-output classification tasks** using more complex input patterns. The objective is to evaluate how single-layer networks generalize the principles of Perceptron and Delta learning rules and to analyze their performance in more demanding learning scenarios.

1. **Programming Language and Environment**

All experiments in this project were implemented in **Python**. The initial version of the code was developed in .py format using **PyCharm 2024.3.5**. For visualization purposes and to facilitate interactive development, the final version was converted into a **Jupyter Notebook** (.ipynb) and executed in **Visual Studio Code** with the Jupyter extension.

The following Python libraries were used throughout the project:

* numpy for matrix operations and numerical computations,
* matplotlib.pyplot for plotting,
* mpl\_toolkits.mplot3d.Axes3D for 3D visualizations.

1. **Task Description**

This task extends the program from Part 1 to support a **single-layer neural network** with multiple output neurons. Two different cases were explored:

* **Case 1**: Discrete neurons using the sign function
* **Case 2**: Continuous neurons using the Delta Rule

Input patterns and teacher signals used were:

* **Inputs**: (10,2,-1), (2,-5,-1), (-5,5,-1)
* **Teacher signals**: (1,-1,-1), (-1,1,-1), (-1,-1,1)
* Each input vector includes a fixed bias term (-1), as in Part 1.

1. **Code Design**

The program supports training for both discrete and continuous outputs in a single-layer neural network. Each output neuron is updated independently.

* **Case 1 (Discrete neurons):**
  + Uses the sign() function as the activation function
  + Produces output values strictly limited to -1 or 1.
* **Case 2 (Continuous neurons):**
  + Directly uses the net input as the output, with no activation function
  + Enables smooth convergence based on the gradient descent principle
* In both cases, weights were updated using the general rule:

**w = w + η· (t - o) · x**

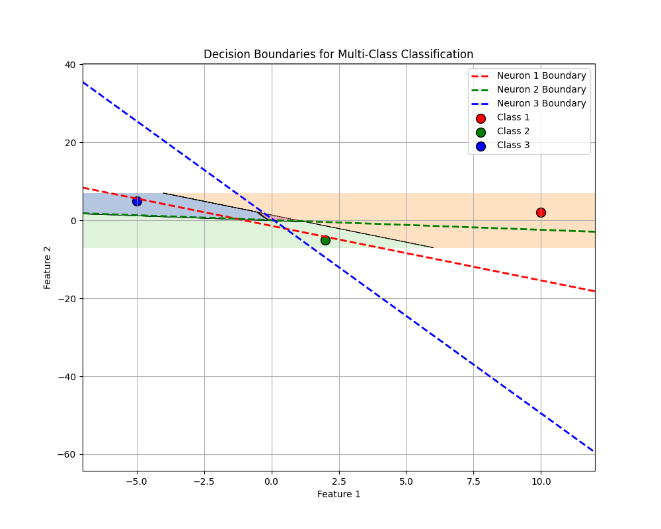
where η is the learning rate, t is the target output, o is the actual output, and x is the input vector.

1. **Visual Analysis**

**5.1 Case 1: Discrete Neurons**

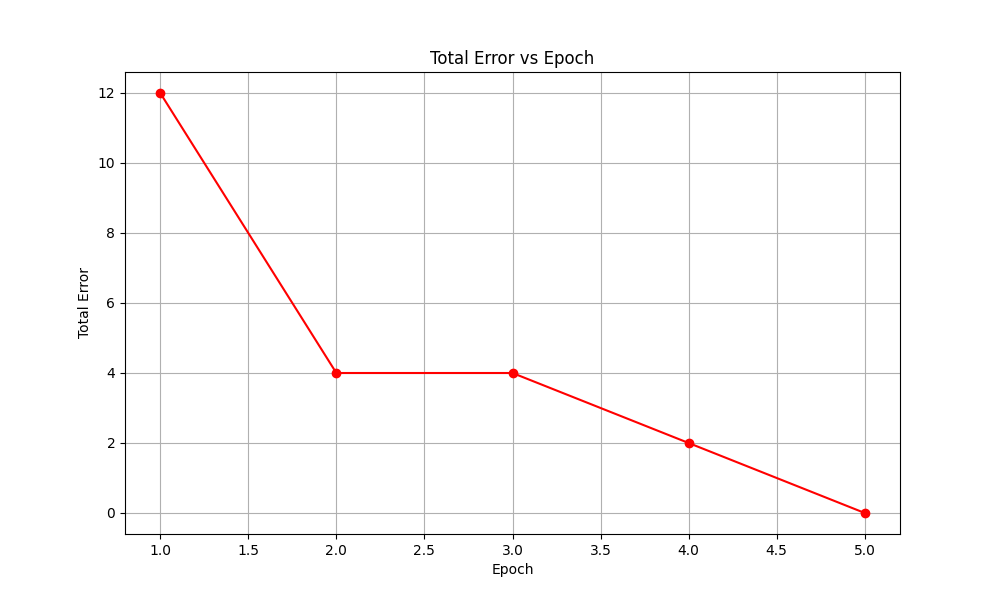
* **Figure D1: Decision Boundaries Plot**

This figure shows the decision boundaries learned by the single-layer neural network with discrete output neurons (using the sign activation function). The input space is clearly partitioned into regions corresponding to each output class, and the plot confirms the model’s ability to correctly classify linearly separable input patterns.



* **Figure D2: Error Plot**

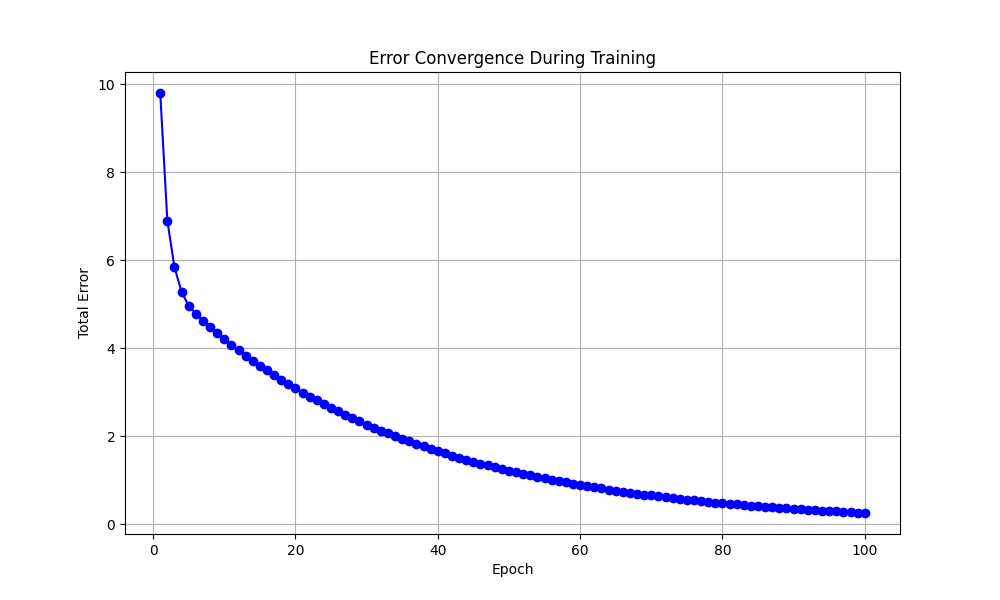
This plot tracks the total classification error across training epochs. It shows a sharp drop in error, with convergence achieved within a few epochs. The discrete nature of the output allowed the network to quickly reach a zero-error solution.



**5.2 Case 2: Continuous Neurons**

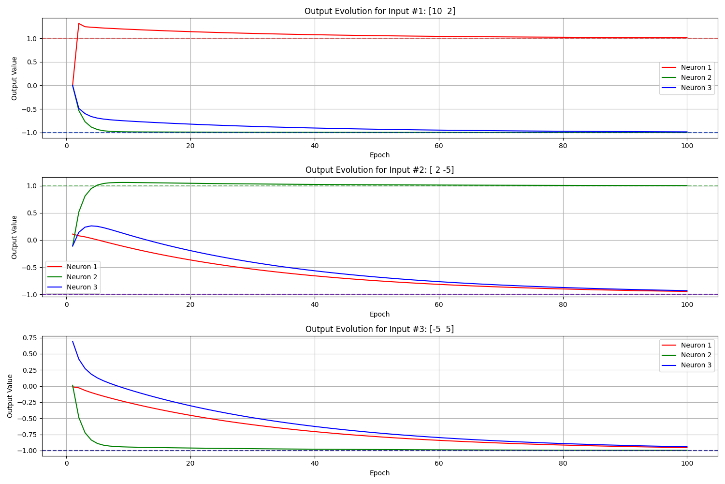
* **Figure C1: Error Convergence Plot**

This figure illustrates the convergence of the error during training with continuous output neurons. The error gradually decreases over time, reflecting the delta rule’s smooth learning trajectory and its reliance on gradient-based updates.



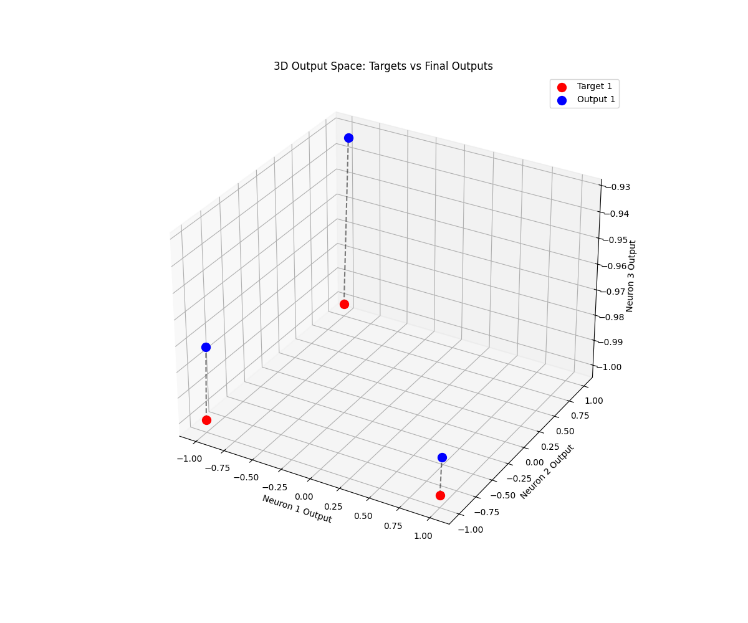
* **Figure C2: Output Evolution**

This graph shows how the outputs for each neuron evolve across the epochs for the three input samples. The gradual progression toward the target values highlights the network’s ability to approximate the desired outputs even in multi-output scenarios.



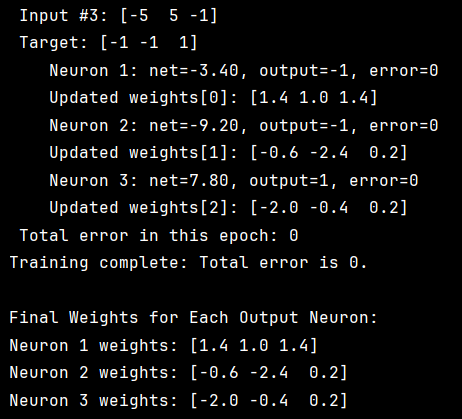
* **Figure C3: 3D Decision Space Visualization**

This 3D plot visualizes the learned decision surfaces in input space. It provides an intuitive representation of how the continuous neurons respond to varying inputs, and how the model learns to form smooth transitions between classes in higher-dimensional space.



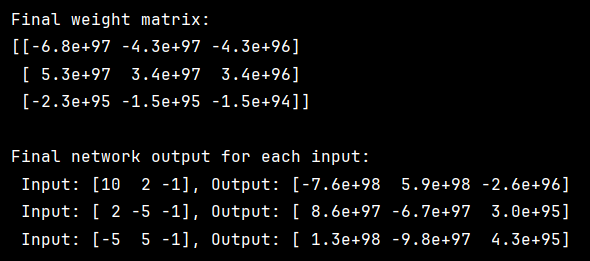
1. **Result and Discussion**

In **Case 1**, the network achieved perfect classification with all outputs matching the expected teacher signals. The training process converged quickly with zero total error, confirming the effectiveness of discrete neurons in classification tasks with clearly separable classes.



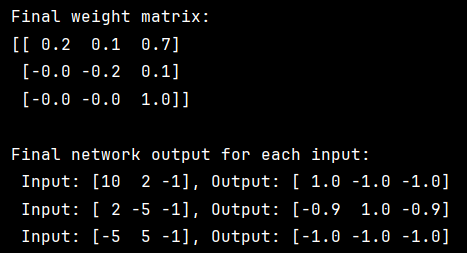
**Figure 1** Output of Case 1

In **Case 2**, the outputs closely approached the target signals, showing a consistent learning trend. Though minor residual errors remained at the final epoch, the results confirmed gradient-based learning.However, an important observation was made regarding the sensitivity of the training process to the learning rate. Initial tests with a large learning rate led to unstable outputs with extremely high values, as shown below:



**Figure 2** outputs instability with learning rate = 0.1

This instability was resolved by reducing the learning rate to 0.01, which stabilized the training process and led to steady convergence toward the target outputs.



**Figure 3** stable learning behavior with learning rate = 0.01

1. **Conclusion**

This part of the project extended the learning setup to a single-layer neural network with multiple output neurons, exploring both discrete and continuous models.

* In Case 1, discrete neurons using the sign activation function achieved perfect classification, demonstrating the effectiveness of binary decision-making for linearly separable data.
* In Case 2, continuous neurons provided smooth approximation of target outputs, though some small errors remained at convergence.

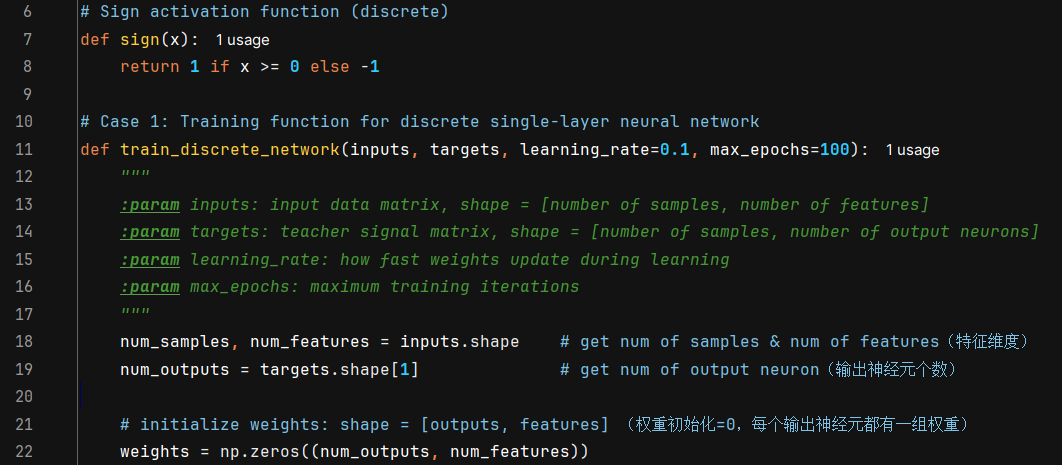
The experiment also revealed that learning rate selection is critical: a value too large caused divergence and instability, while a smaller rate ensured stable learning. This highlighted the importance of hyperparameter tuning in practical neural network training.

Overall, this part reinforced key concepts in multi-output learning, convergence behavior, and the trade-offs between precision and stability in discrete vs. continuous models.

1. **Appendix: Code Screenshots**

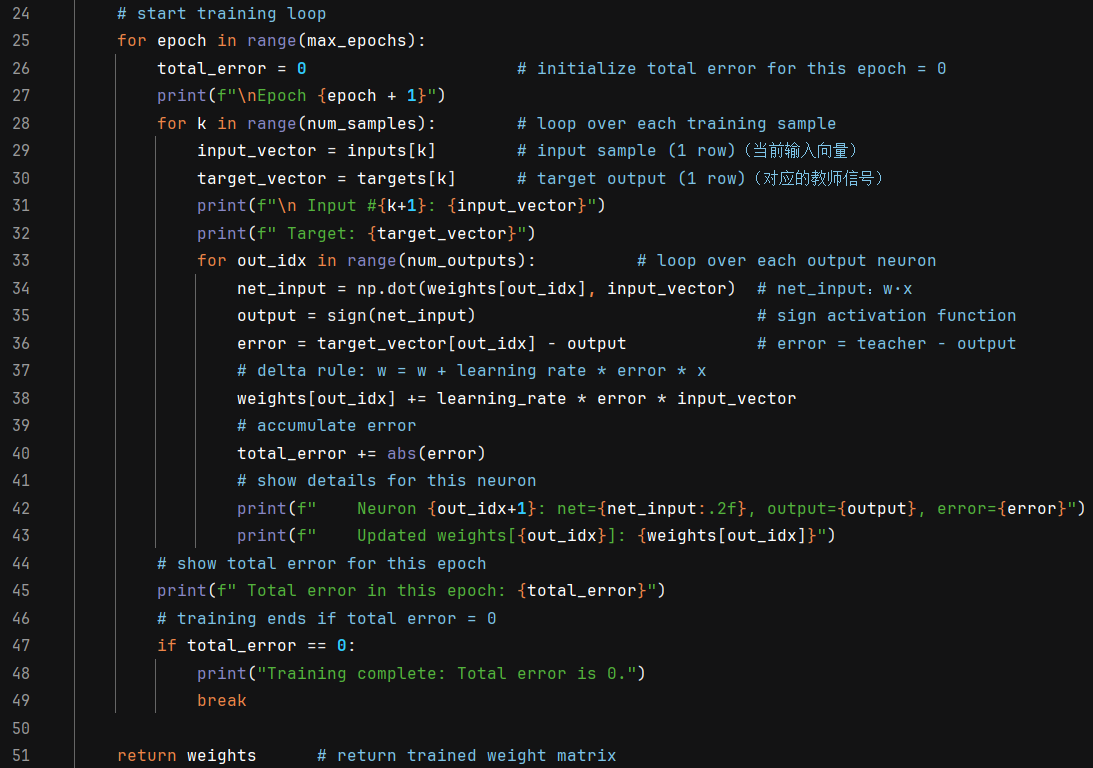
**Figure A1. (Case 1) Activation Function and Training Function Definition**

This part shows the sign activation function and the main training loop for the discrete single-layer neural network.



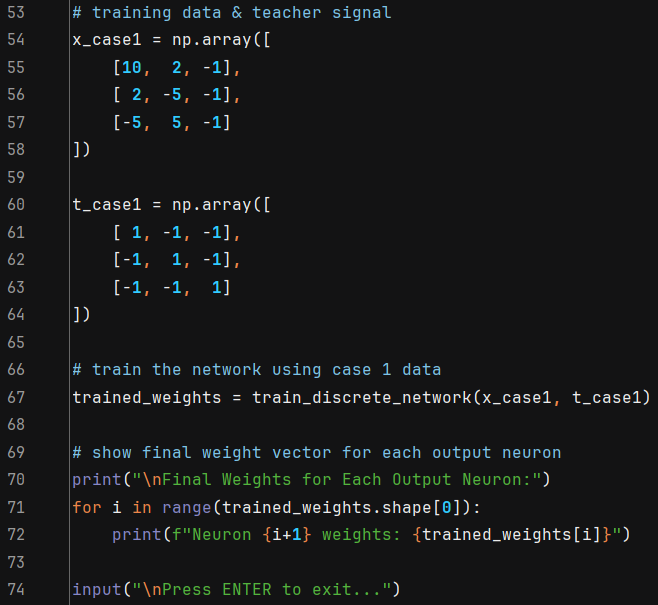
**Figure A2. (Case 1) Epoch Loop and Weight Updates**

This section includes the internal logic for weight updates per output neuron, error accumulation, and termination conditions.

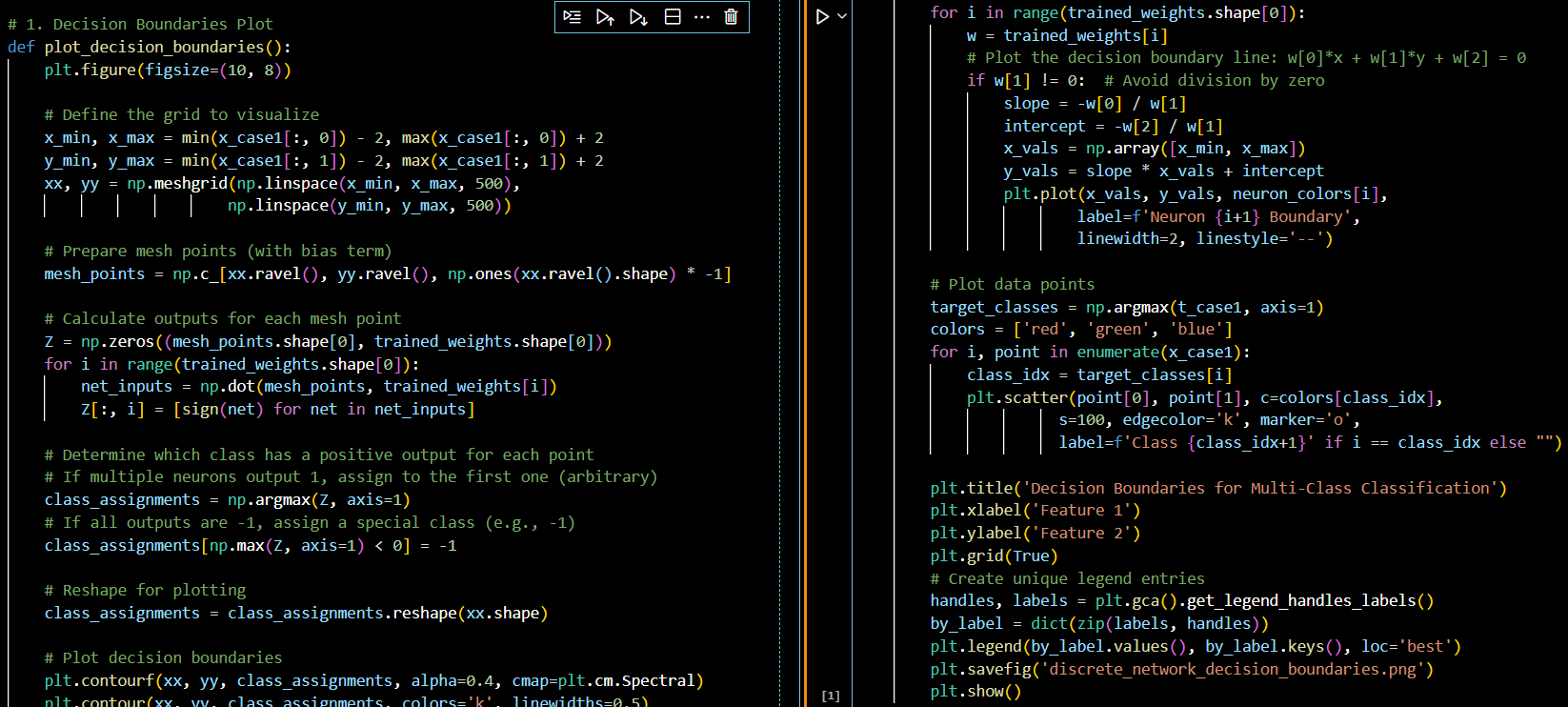


**Figure A3. (Case 1) Training Data and Final Output**

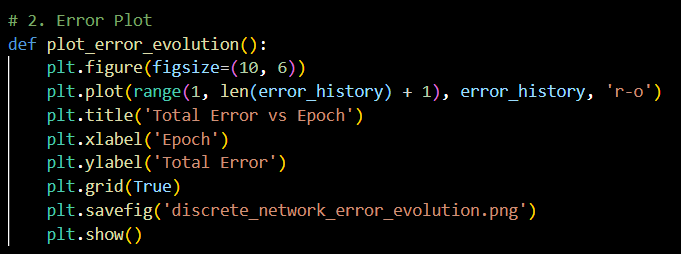
This shows the input/output data, training call, and how the final trained weights are displayed.



**Figure A4. (Case 1) Decision Boundaries Plot**

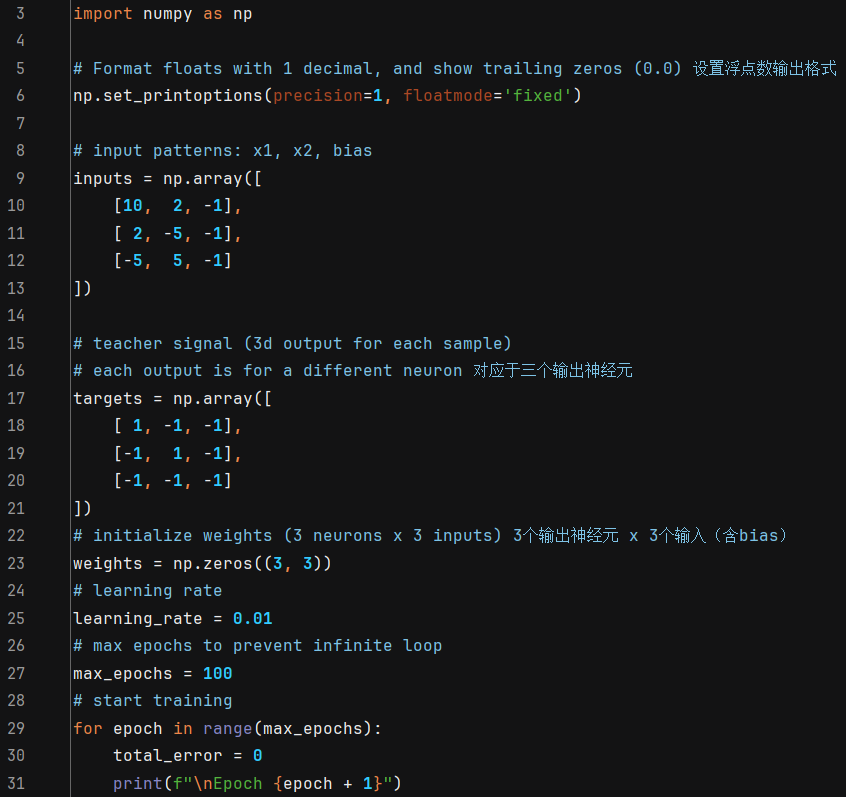


**Figure A5. (Case 1) Error Plot**



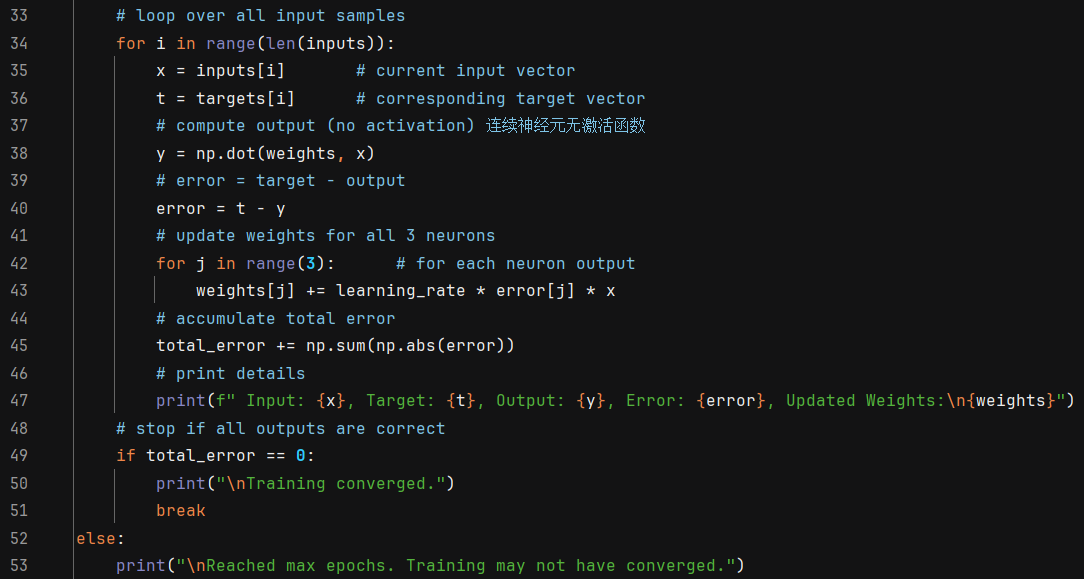
**Figure A6. (Case 2) Initialization and Training Loop**

Shows the setup of input, output, and the main training loop using the Delta Rule without activation.



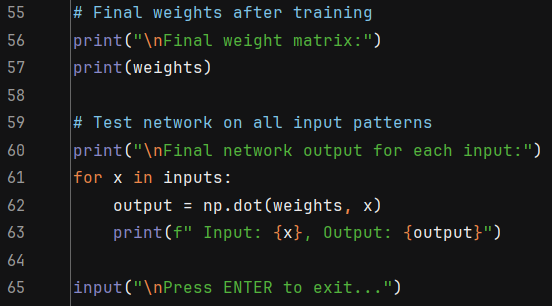
**Figure A7. (Case 2) Output Calculation and Weight Update Logic**

This part illustrates how outputs are calculated via dot product and weights updated per neuron.

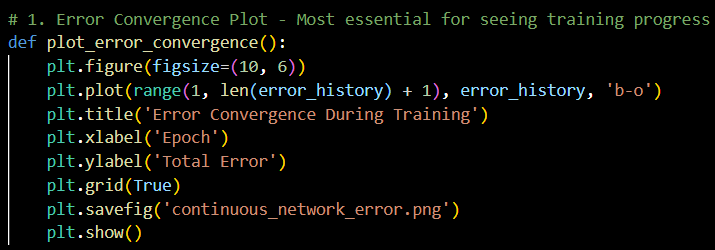


**Figure A8. (Case 2) Final Training Result and Test Outputs**

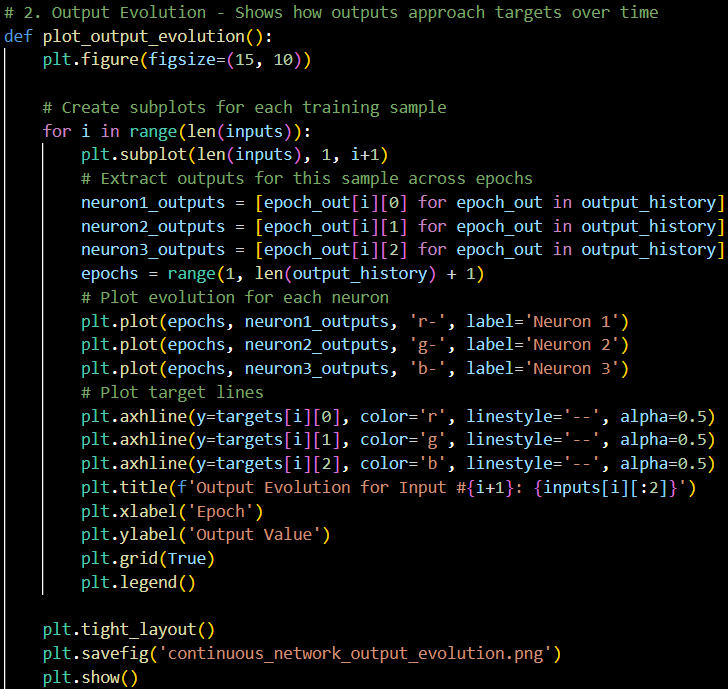
Shows the final weight matrix and the outputs of the trained network.



**Figure A10. (Case 2) Error Convergence Plot**



**Figure A11. (Case 2) Output Evolution**



**Figure A12. (Case 2) 3D Decision Space Visualization**

