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

**Part 1: Learning of a single neuron**

1. **Introduction**

This report investigates the implementation and behavior of neural network learning mechanisms through the training of a **single neuron** using both the **Perceptron Learning Rule** and the **Delta Learning Rule**. The primary objective is to demonstrate that a single neuron can perform a simple classification task: the realization of a logical **AND gate**. The experiment aims to analyze the learning dynamics, convergence behavior, and the effectiveness of each learning rule in guiding the neuron to a correct decision boundary.

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 involves training a single neuron to realize the logical AND gate using both the **Perceptron Learning Rule** and the **Delta Learning Rule**. The neuron receives three inputs: x1, x2, and bias fixed at -1. The target outputs are -1 for all input combinations except for (1,1,-1), where the target output is 1.

The training data is as follows:

* **Inputs**: (0,0,-1), (0,1,-1), (1,0,-1), (1,1,-1)
* **Teacher signals**: -1, -1, -1, 1

1. **Code Structure and Functions**

Key functions and logic in the implementation:

* **sign(n)**: The sign activation function, returning 1 if input >= 0 and -1 otherwise. Used in both Perceptron and Delta rule versions.
* **Perceptron Training Loop**: The perceptron training loop updates the weights using the following rule:

**w = w + η · e · x**

where η is the learning rate, e = t – 0 is the error (target – output), and x is the input vector. This rule applies updates only when the neuron output is incorrect.

* **Delta Rule**: The Delta Rule is used for continuous output neurons. It updates weights using the formula:

**w = w + η· (t - o) · x**

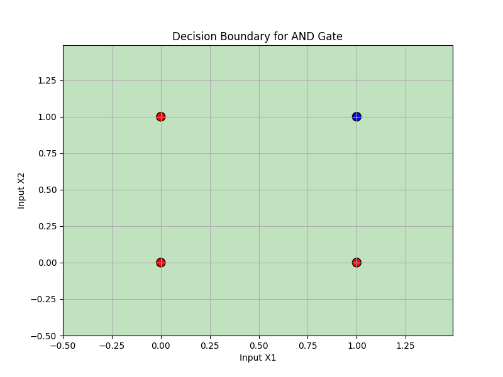
where η is the learning rate, t is the target output, o is the neuron’s actual continuous output (such as the net input), and x is the input vector. This rule allows gradual convergence and works better with differentiable activation functions.

1. **Visual Analysis**

**5.1 Perceptron Learning Rule**

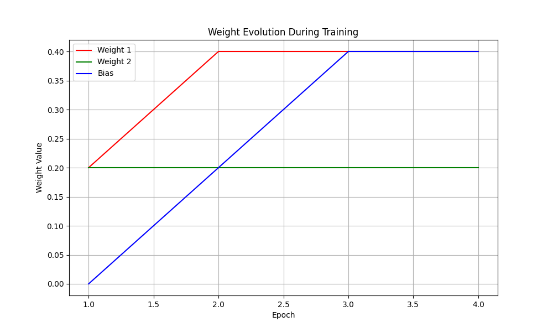
* **Figure P1: Perceptron Decision Boundary Plot**

This plot visualizes the decision boundary learned by the perceptron after training on the AND gate dataset. It clearly shows the linear separator that the perceptron has learned, which divides the input space into two regions, each corresponding to one of the output classes. The plot confirms that the perceptron has successfully learned to separate the two classes of the AND gate.



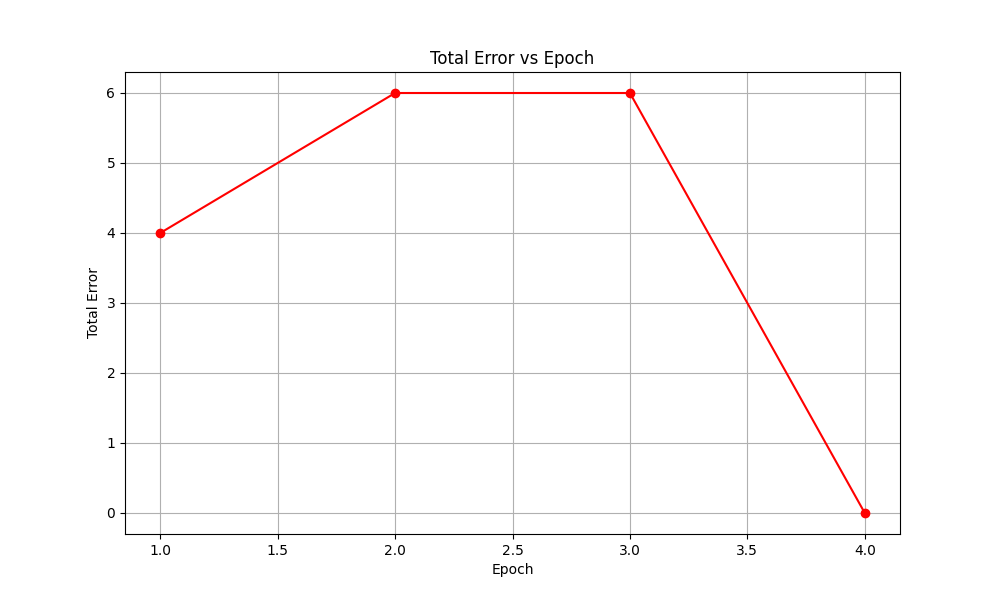
* **Figure P2: Perceptron Weight Evolution Plot**

This graph tracks the evolution of the perceptron’s weight across the training epochs. It demonstrates how the weights gradually adjust to minimize the error and converge toward their final values. The plot reflects the iterative nature of the perceptron learning process.



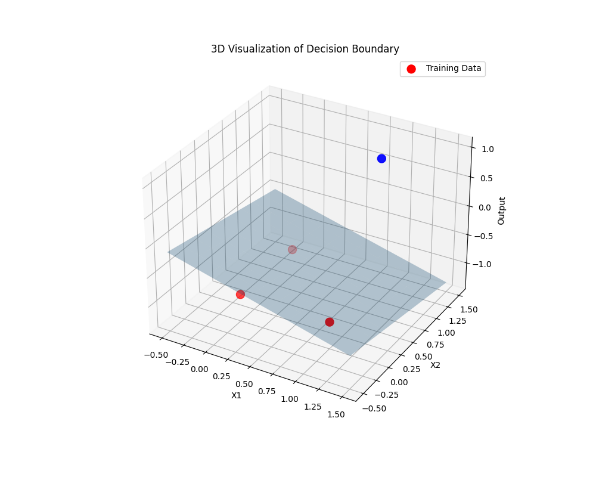
* **Figure P3: Perceptron Error Plot**

This figure depicts the total error during each epoch of training. The error rapidly decreases as the perceptron learns, and eventually reaches zero error, indicating successful convergence after a few epochs.



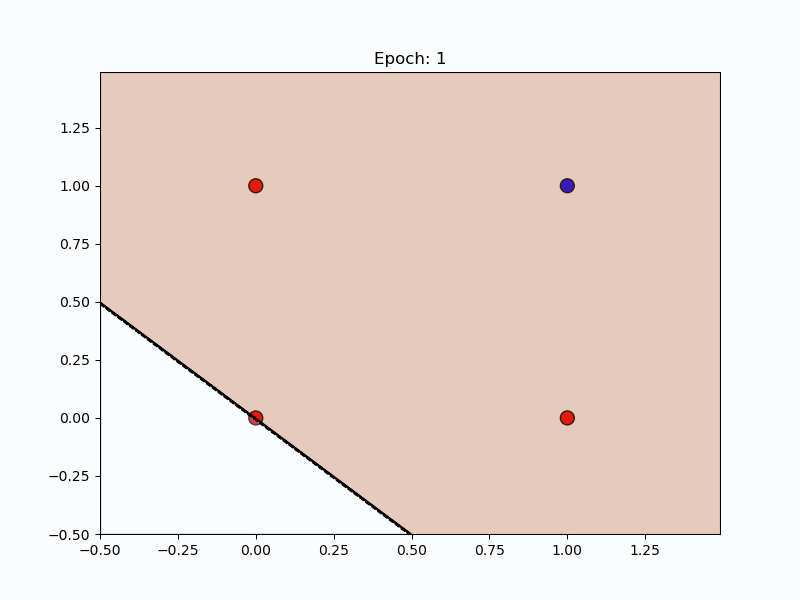
* **Figure P4: Perceptron 3D Visualization of the Separating Plane**

This 3D plot shows the separating plane in the input space as learned by the perceptron. The plot illustrates the spatial relationship between the input space and the decision surface, highlighting how the perceptron divides the space into regions based on its learned weights.



* **Figure P5: Perceptron Animation of Decision Boundary Evolution**

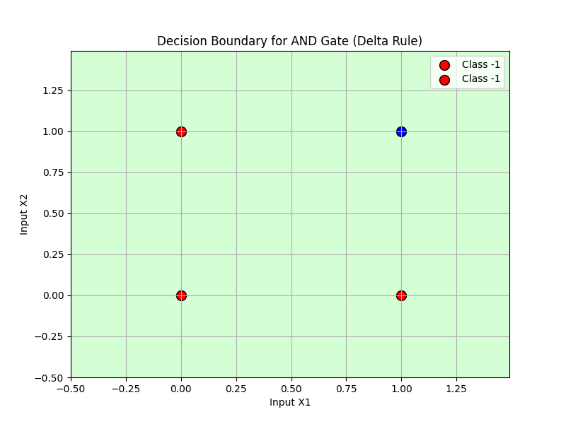
This animation shows the evolution of the decision boundary during training. It illustrates how the boundary adjusts as the perceptron iterates through the training data, gradually improving its ability to classify input points correctly.



**5.2 Delta Learning Rule**

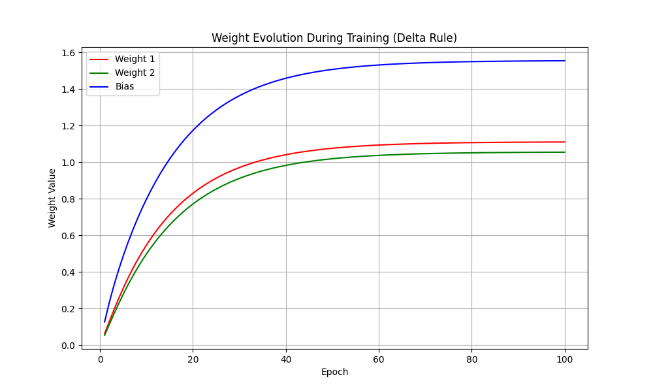
* **Figure D1: Delta Decision Boundary Plot**

This plot shows the decision boundary learned by the delta rule. It illustrates how the decision boundary smoothly separates the different classes in the input space. Unlike the perceptron with discrete outputs, the delta rule produces continuous output values, which results in a smoother and less abrupt decision boundary.



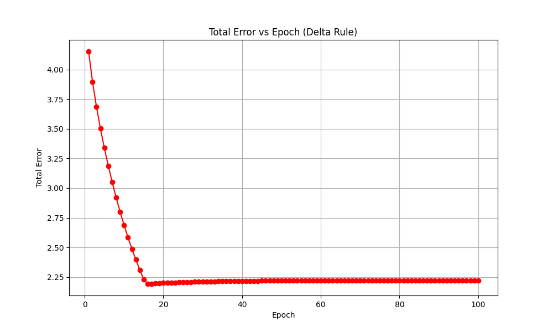
* **Figure D2: Delta Weight Evolution Plot**

This plot tracks how the weights evolve during training with the delta rule. It demonstrates the gradual adjustment of the weights based on continuous output values, highlighting the smoother convergence process compared to the perceptron learning rule.



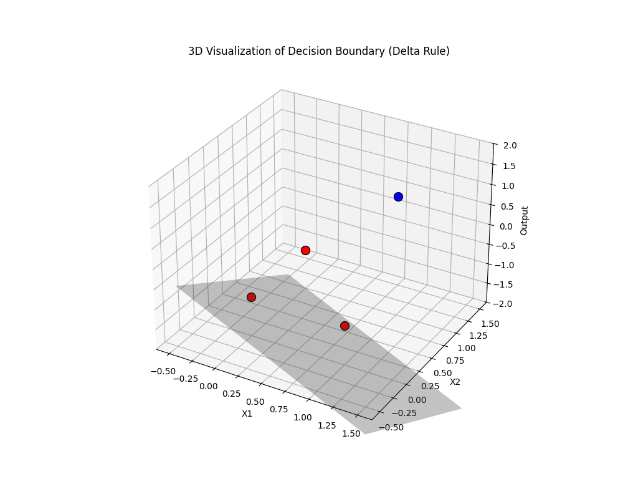
* **Figure D3: Delta Error Plot**

This error plot visualizes how the total error decreases as the network learns with the delta rule. The error decreases more gradually compared to the perceptron, reflecting the continuous nature of the output and the gradual weight updates.



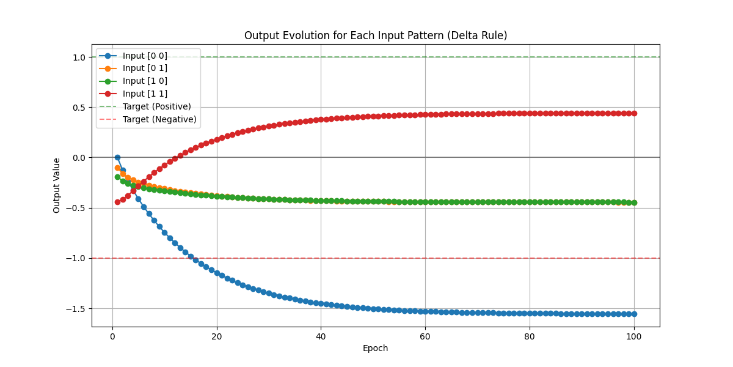
* **Figure D4: Delta 3D Visualization of the Separating Plane**

This 3D visualization illustrates the separating plane in the input space as learned by the delta rule. It demonstrates how the continuous nature of the outputs affects the decision surface, leading to smoother transitions between the output classes.



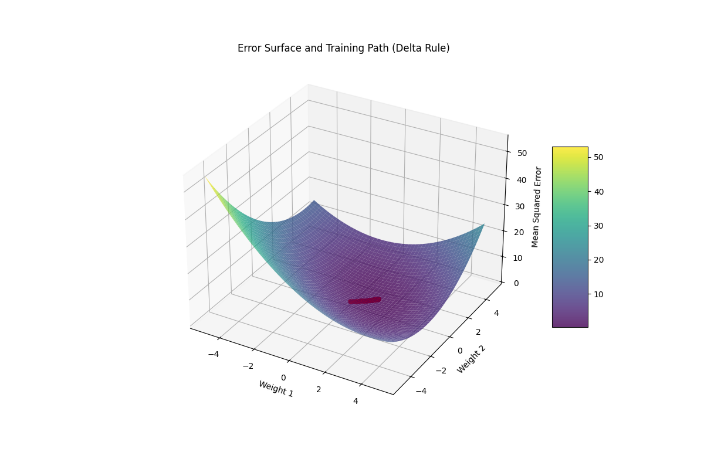
* **Figure D5: Delta Output Evolution for Each Sample**

This figure shows the evolution of the output for each sample throughout training. It highlights how the network’s output for each input sample approaches the target signal over time, providing insights into the learning process and the stability of the training.



* **Figure D6: Delta Learning Surface Plot**

This plot represents the learning surface of the delta rule. It maps the neuron’s output over a range of input combinations, illustrating how the network generalizes to different input patterns and approximates the real-valued target signals.



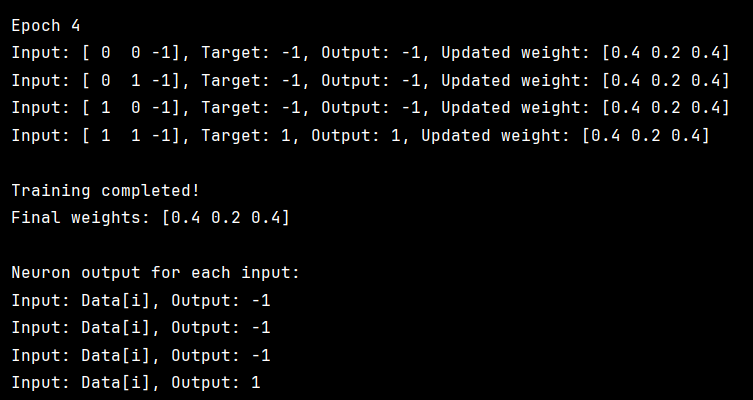
1. **Results and Discussion**

Both learning rules successfully trained the neuron to emulate the AND gate.

The **Perceptron Learning Rule** achieved this by using discrete outputs (either 1 or -1) in each iteration, which made it easier to represent the binary nature of the AND gate. It reached convergence after a few epochs, with the neuron outputting the expected results for the training data.

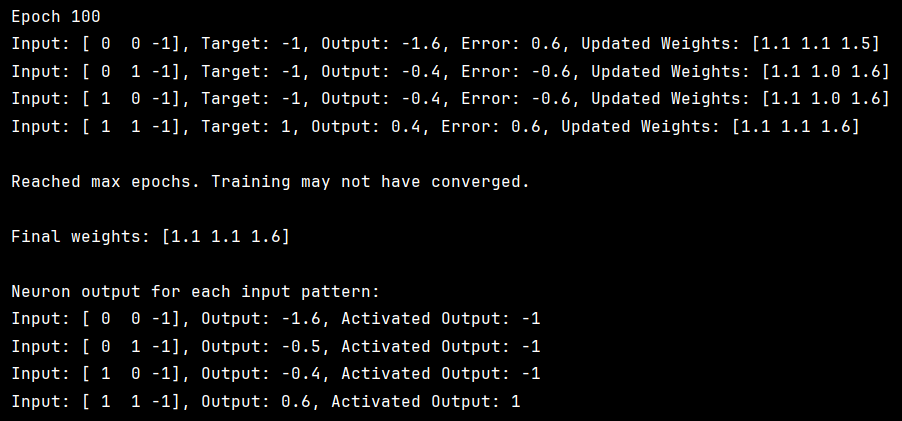
The **Delta Learning Rule** demonstrated the continuous convergence of the weights and exhibited a smoother learning process. The weight adjustments were more gradual because the error was calculated as a continuous value rather than a discrete binary one. This allowed the neural network to converge more precisely, especially when dealing with non-binary outputs, but in this case, it still successfully emulated the AND gate.

* **Perceptron Learning Rule**: Achieved discrete output values matching the AND gate, with training converging after a few epochs.



**Figure 1** Output of Perceptron Rule training

* **Delta Learning Rule**: Demonstrated smoother convergence due to continuous weight adjustments, showing more gradual learning behavior.



**Figure 2** Output of Delta Rule training

1. **Conclusion**

This part of the project successfully demonstrated the effectiveness of both the Perceptron Learning Rule and the Delta Learning Rule in training a single neuron to emulate a logical AND gate. The experiment confirmed that:

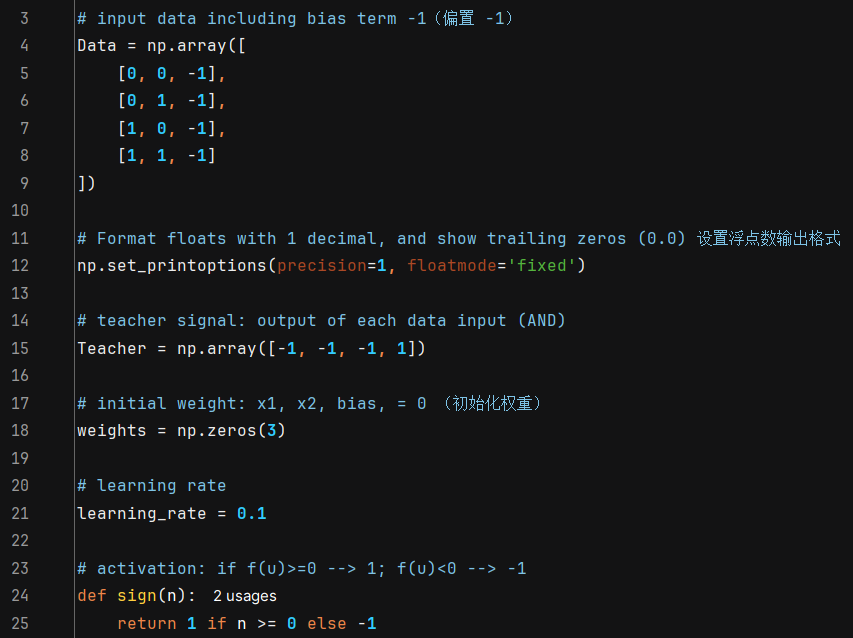
* The Perceptron Rule is highly effective for binary classification tasks, converging quickly with discrete output values.
* The Delta Rule, although more gradual in learning, provides smoother convergence and is well-suited for problems requiring continuous updates.

Both approaches achieved correct classification on the training data, highlighting their foundational roles in neural network learning. This experiment serves as a clear and instructive example of how basic learning algorithms function within a single-neuron model.

1. **Appendix: Code Screenshots**

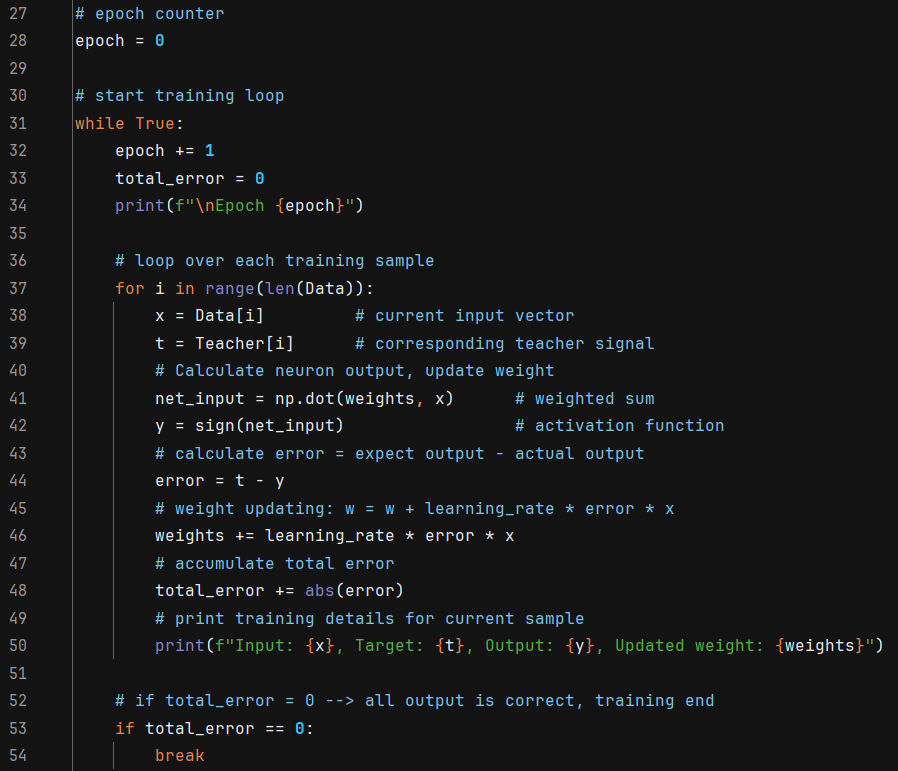
**Figure A1. [Perceptron] Initialization and Data Setup**

This snippet shows the initialization of input patterns, teacher signals, weights, and the definition of the sign activation function.



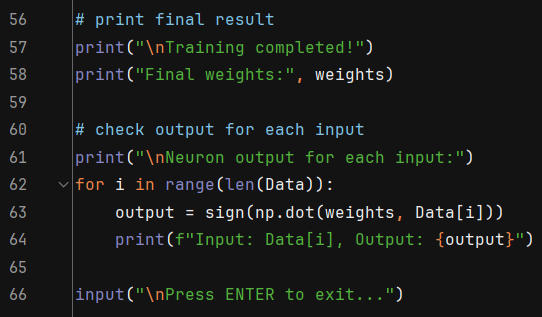
**Figure A2. [Perceptron] Training Loop for Perceptron Learning**

This part implements the core of the Perceptron training algorithm, including weight update rule and training epoch loop.

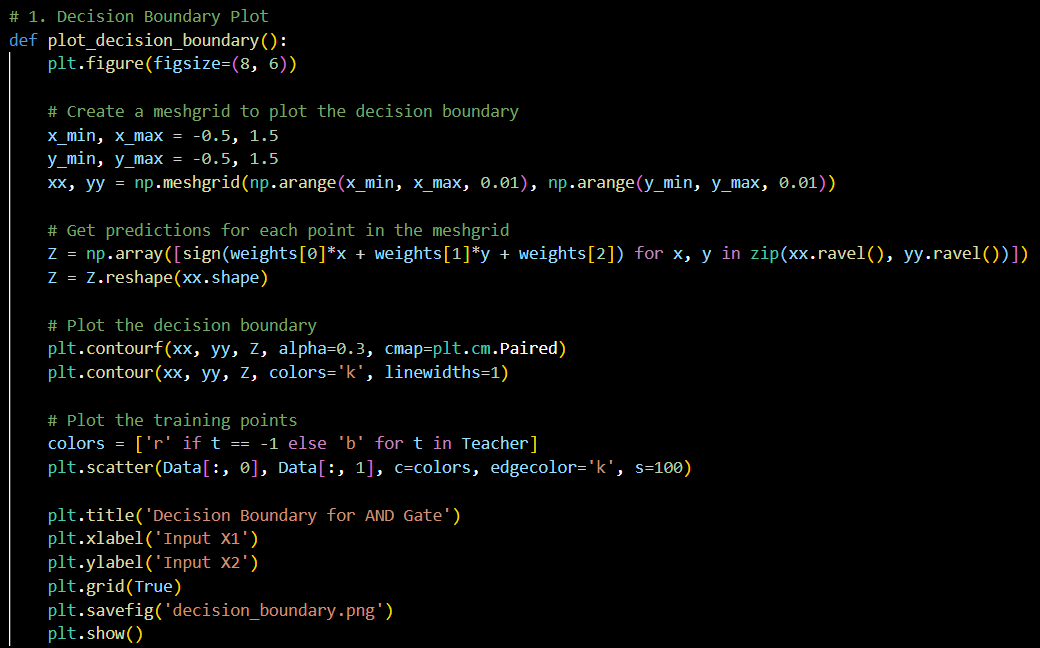


**Figure A3. [Perceptron] Final Output After Training**

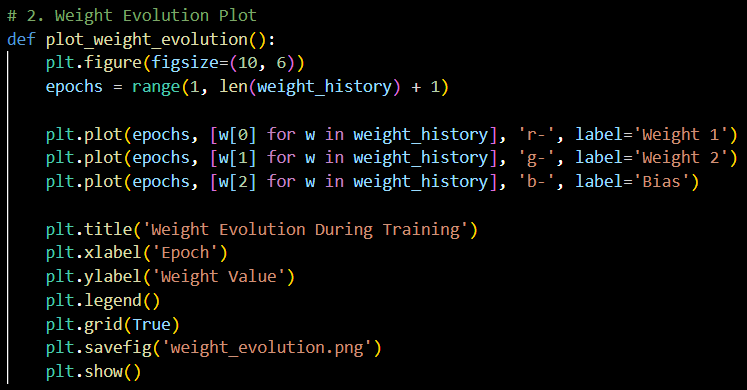
This section prints the final learned weights and tests the neuron outputs for all training data.



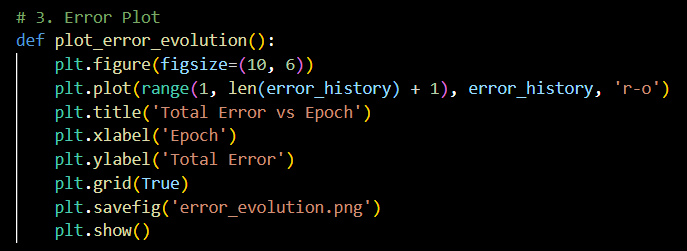
**Figure A4. [Perceptron] Decision Boundary Plot**



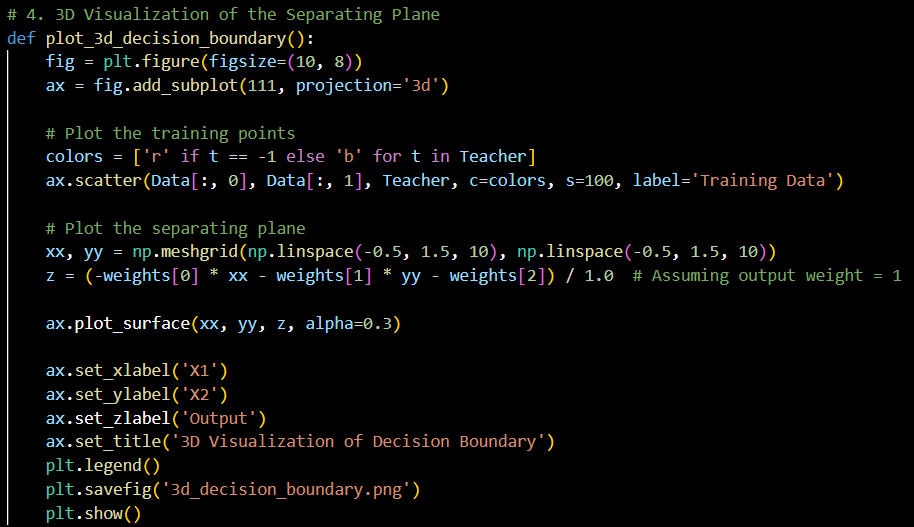
**Figure A5. [Perceptron] Weight Evolution Plot**



**Figure A6. [Perceptron] Error Plot**



**Figure A7. [Perceptron] 3D Visualization of the Separating Plane**

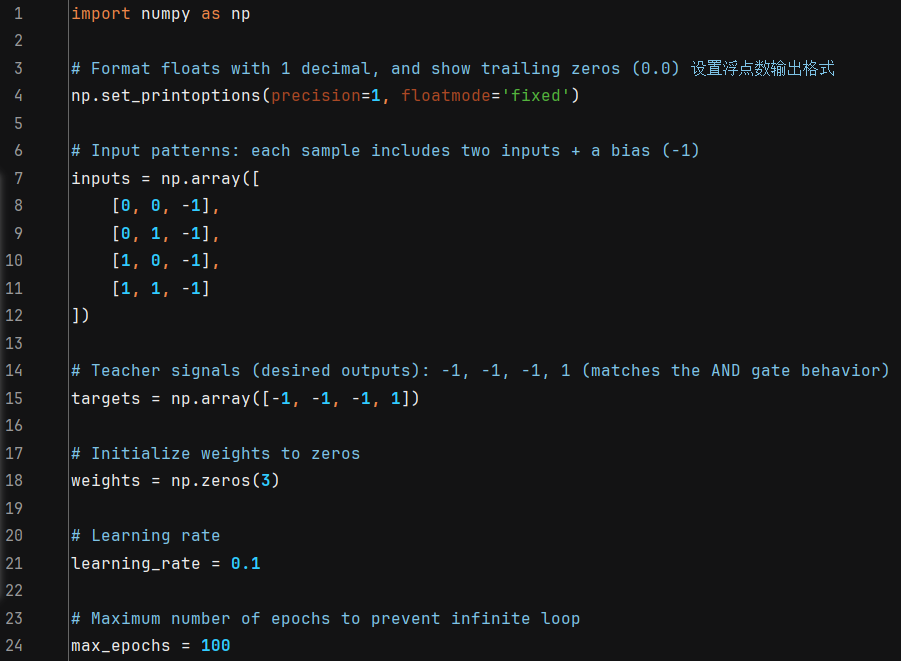


**Figure A8. [Perceptron] Animation of Decision Boundary Evolution**



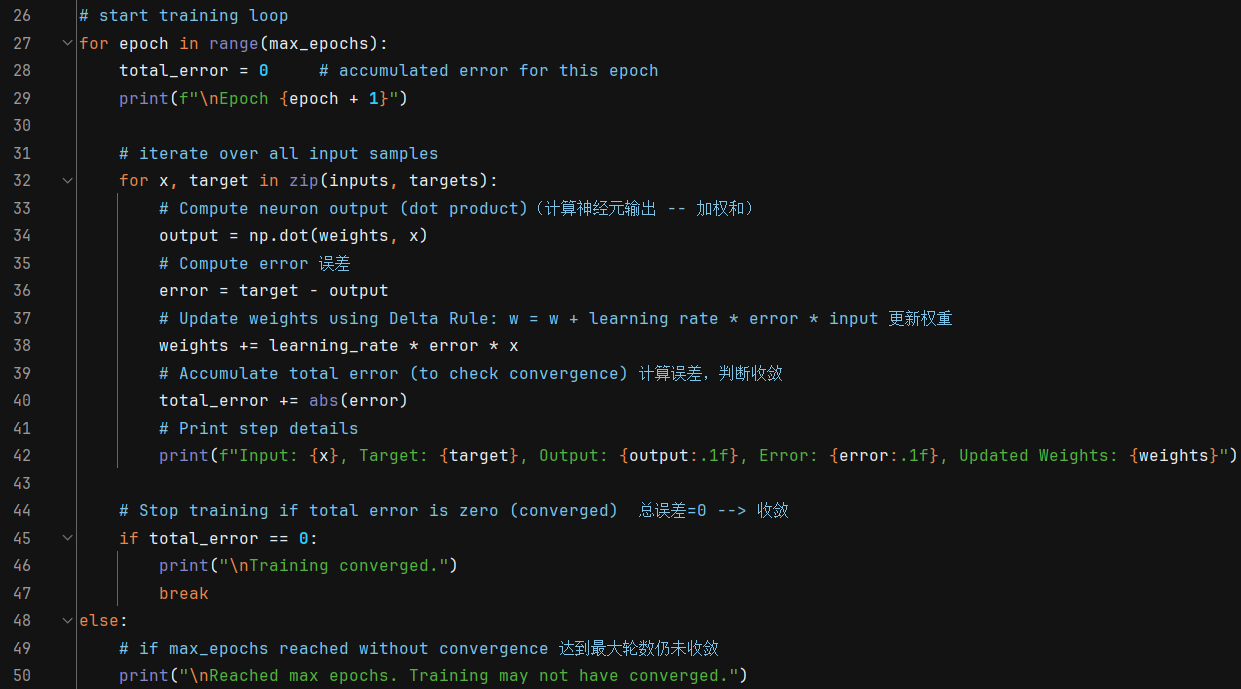
**Figure A9. [Delta] Initialization of Data and Weights**

This snippet shows the setup of input patterns, target signals, weight initialization, and learning rate definition.



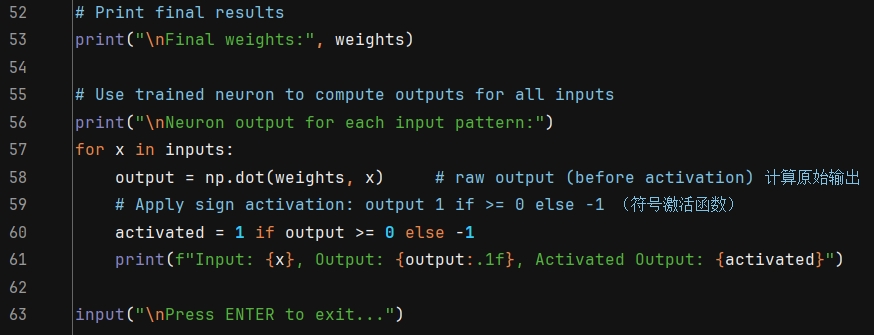
**Figure A10. [Delta] Training Loop and Weight Updates**

This part illustrates the main training loop using the Delta Rule, where weights are updated based on real-valued errors.

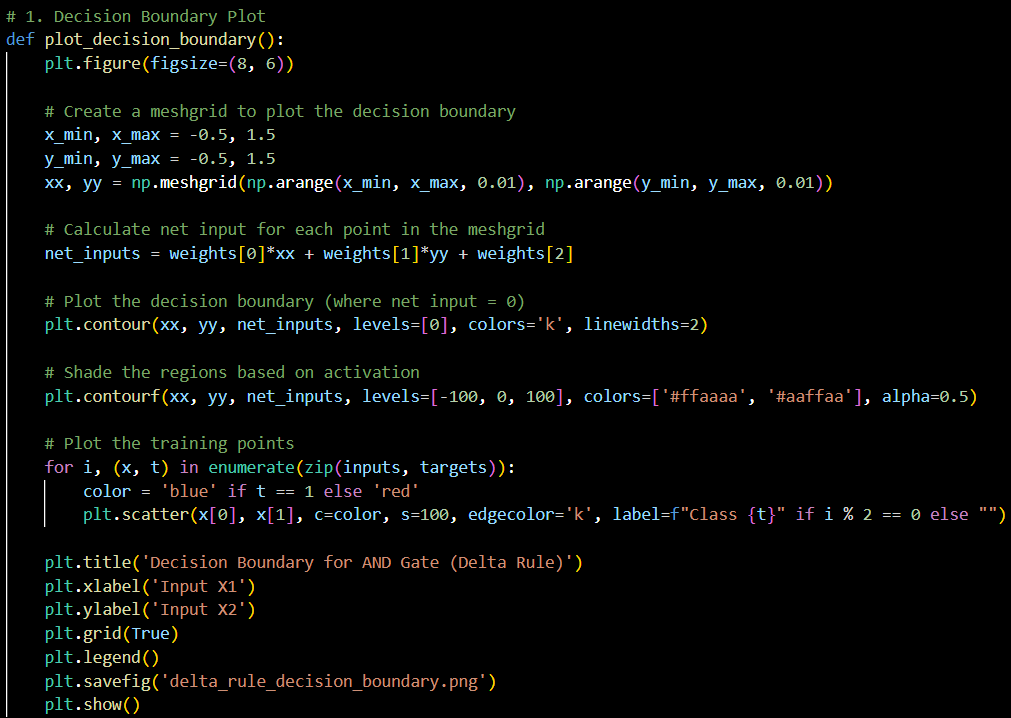


**Figure A11. [Delta] Final Results and Output Evaluation**

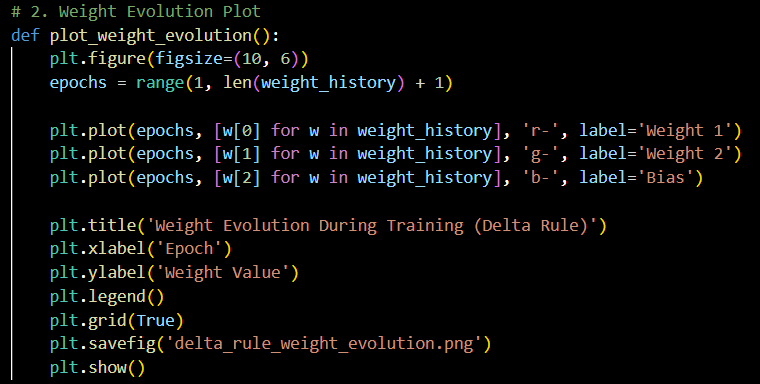
This section prints the final weights and evaluates the neuron’s output after training using both raw and activated outputs.



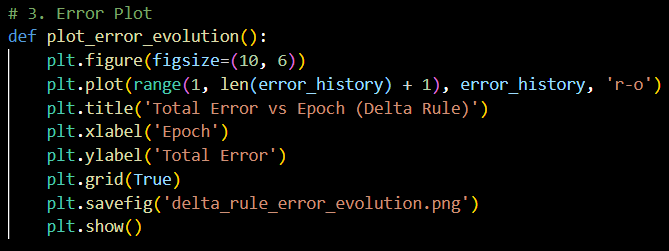
**Figure A12. [Delta] Decision Boundary Plot**



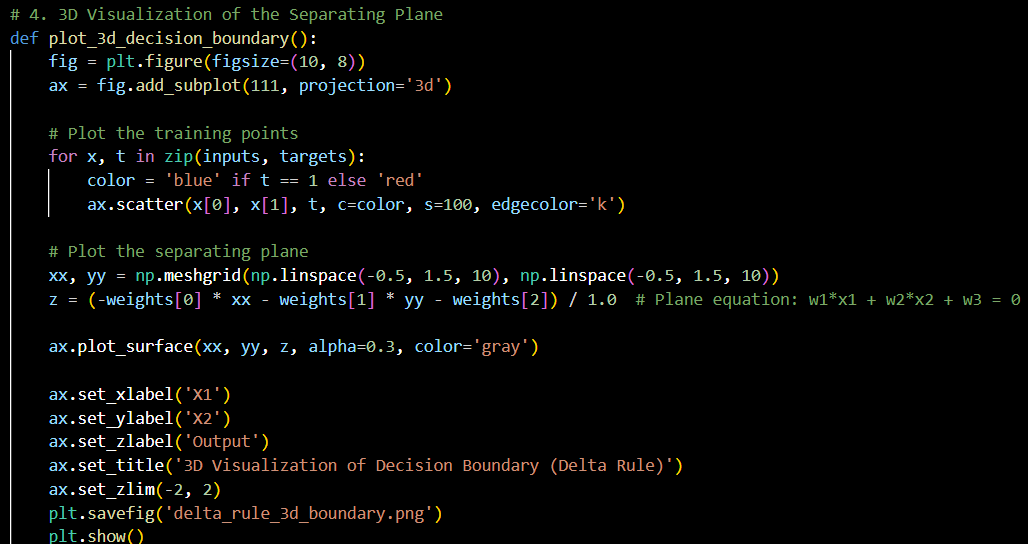
**Figure A13. [Delta] Weight Evolution Plot**



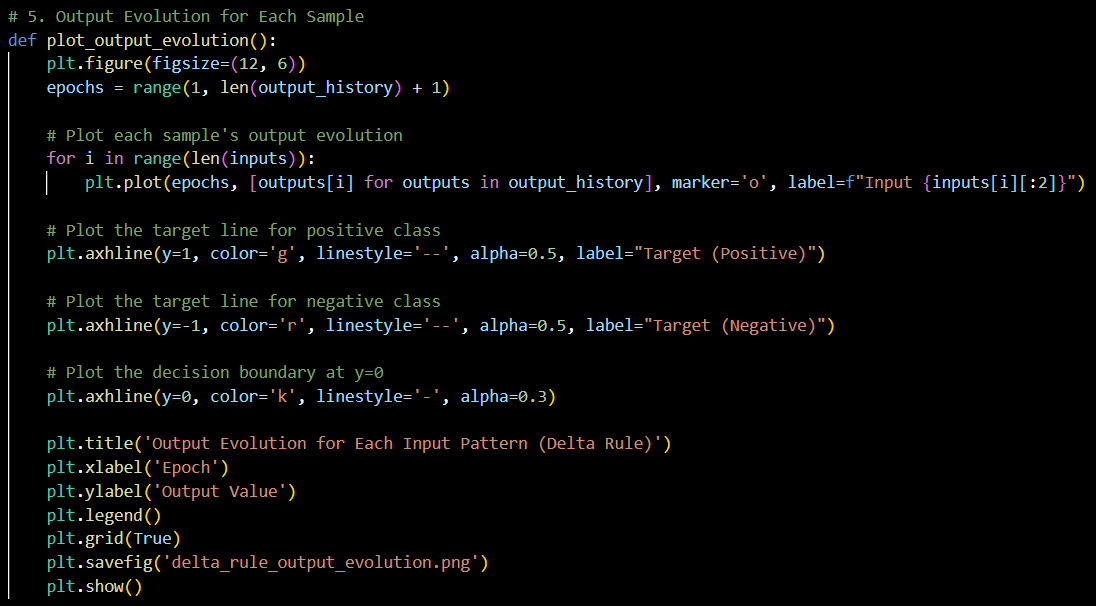
**Figure A14. [Delta] Error Plot**



**Figure A15. [Delta] 3D Visualization of the Separating Plane**



**Figure A16. [Delta] Output Evolution for Each Sample**



**Figure A17. [Delta] Learning Surface Plot (Error Surface)**

