# Data Summary

| **Learning Rate** | **Activation Function** | **Perceptron Accuracy** | **Delta Rule Accuracy** |
| --- | --- | --- | --- |
| 0.001 | step | 0.63 | 0.63 |
| 0.001 | sigmoid | 0.00 | 0.00 |
| 0.001 | tanh | 0.00 | 0.00 |
| 0.001 | relu | 0.33 | 0.33 |
| 0.010 | step | 0.63 | 0.63 |
| 0.010 | sigmoid | 0.30 | 0.30 |
| 0.010 | tanh | 0.30 | 0.30 |
| 0.010 | relu | 0.33 | 0.33 |
| 0.050 | step | 0.43 | 0.43 |
| 0.050 | sigmoid | 0.30 | 0.30 |
| 0.050 | tanh | 0.30 | 0.30 |
| 0.050 | relu | 0.33 | 0.33 |
| 0.100 | step | 0.33 | 0.33 |
| 0.100 | sigmoid | 0.30 | 0.30 |
| 0.100 | tanh | 0.30 | 0.30 |
| 0.100 | relu | 0.33 | 0.33 |
| 0.250 | step | 0.30 | 0.30 |
| 0.250 | sigmoid | 0.30 | 0.30 |
| 0.250 | tanh | 0.30 | 0.30 |
| 0.250 | relu | 0.33 | 0.33 |
| 0.300 | step | 0.30 | 0.30 |
| 0.300 | sigmoid | 0.30 | 0.30 |
| 0.300 | tanh | 0.30 | 0.30 |
| 0.300 | relu | 0.33 | 0.33 |
| 0.500 | step | 0.30 | 0.30 |
| 0.500 | sigmoid | 0.30 | 0.30 |
| 0.500 | tanh | 0.30 | 0.30 |
| 0.500 | relu | 0.33 | 0.33 |
| 1.000 | step | 0.30 | 0.30 |
| 1.000 | sigmoid | 0.30 | 0.30 |
| 1.000 | tanh | 0.30 | 0.30 |
| 1.000 | relu | 0.33 | 0.33 |

**Challenges Encountered and Overcome**

1. **Convergence Issues**: Implementing the Perceptron and Delta Rule algorithms from scratch, I encountered convergence issues, particularly with the Perceptron algorithm. To overcome this, I adjusted the learning rate and implemented a convergence threshold to stop training when the accuracy plateaued.
2. **Activation Function Selection**: Choosing the appropriate activation function for each algorithm was crucial. I experimented with different activation functions (step, sigmoid, tanh, relu) to understand their impact on convergence and accuracy.
3. **Debugging**: Debugging was a significant challenge, especially when implementing the Delta Rule algorithm. I used print statements and visualizations to understand the flow of data and identify errors.

**Reflection on Algorithm Strengths and Limitations**

**Perceptron Algorithm**

* **Strengths**: Simple to implement, effective for linearly separable data, and easy to interpret.
* **Limitations**: Limited to linearly separable problems, may not converge for non-linearly separable data, and sensitive to learning rate.

**Delta Rule (Gradient Descent) Algorithm**

* **Strengths**: Can handle non-linearly separable problems, more flexible due to gradient descent, and can converge with appropriate learning rates.
* **Limitations**: Requires differentiable activation functions, slower convergence compared to Perceptron for some problems, and prone to local minima.

Overall, while both algorithms have their strengths and weaknesses, the Delta Rule (using gradient descent) tends to be more versatile for handling complex, non-linear problems, albeit with careful tuning of learning rates and activation functions.