The primary goal was to implement a simple two-layer neural network using only the NumPy library to solve the classic XOR classification problem. The project aimed to demonstrate that a non-linear model is necessary for this task and to gain a fundamental understanding of the core components of a neural network.

Implementation Steps

The task was completed by following a clear, step-by-step process:

1. **Dataset Setup**: The XOR problem's inputs (X) and outputs (y) were defined as NumPy arrays to create the training data.
2. **Model Architecture**: A two-layer network was designed with a hidden layer using the **tanh** activation function and an output layer using the **sigmoid** activation function. The model's weights and biases were initialized with small random values.
3. **Training Process**: The network was trained over several thousand epochs. Each epoch consisted of a four-part cycle:
    - **Forward Pass**: The input data was fed through the network to generate predictions.
    - **Loss Calculation**: The Mean Squared Error (MSE) was computed to measure the difference between the model's predictions and the actual outputs.
    - **Backpropagation**: The error was propagated backward through the network to calculate the gradients for each weight and bias.
    - **Gradient Descent**: The network's weights and biases were adjusted in the direction that minimizes the loss, using a defined learning rate.
4. **Evaluation and Visualization**: After training, the model's accuracy was tested against the original inputs, and a plot of the decision boundary was generated to visually confirm its ability to solve the non-linear problem.

Understanding of Key Concepts

This task provided a deep understanding of the core components of neural networks:

- **Weights and Biases**: These are the network's learnable parameters. Weights determine the strength of connections between neurons, while biases allow neurons to activate more flexibly. The network's learning is achieved by continuously adjusting these values.

- **Gradients**: A gradient is a crucial calculation during backpropagation that measures the direction and magnitude of the error with respect to each weight and bias. It tells the network how sensitive the error is to a change in a specific parameter.
- **Backpropagation**: This is the algorithm that efficiently calculates the gradients for all parameters. It works by moving backward from the final loss to determine the contribution of each weight and bias to the overall error.
- **Gradient Descent**: This is the optimization process where the network uses the calculated gradients to update its weights and biases. By moving in the opposite direction of the gradient, the model systematically reduces its loss and improves its accuracy over time.

Conclusion

The project was a success, as the implemented neural network correctly classified the XOR problem with **100% accuracy**. The final decision boundary plot also visually confirmed that the network learned a non-linear separation, which proves that a simple neural network with a hidden layer can solve complex problems that are impossible for a linear model.