Simple Neural Network for OR Gate Classification

1. Project Overview

This project implements a very simple **single-layer neural network** (a **perceptron**) using pure Python, without relying on any machine learning libraries like TensorFlow, Keras, or PyTorch.

The goal is to:

- Train the network to model the behavior of a **logical OR gate**.
- Track and save the **loss values during training** into a .csv file.
- Use the saved loss values for further **visualization and analysis** in a Jupyter Notebook.

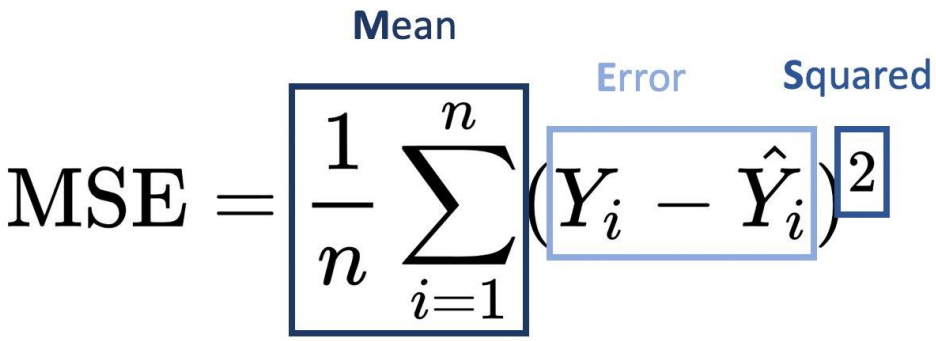
2. Neural Network Details

- **Inputs**: Two binary features (0 or 1) representing logical inputs.
- **Output**: A single output (0 or 1) predicted using a sigmoid activation function.
- **Architecture**:
    - 2 input nodes.
    - 1 output node (no hidden layers).
    - Trainable parameters: weights w1, w2 and bias b.
- **Activation Function**:
    - sigmoid(x) = 1 / (1 + exp(-x))
    - Outputs values between 0 and 1, useful for binary classification.

---

3. Training Process

- **Loss Function**:
  Mean Squared Error (MSE) between predicted and actual outputs:



- **Optimization Algorithm**:
  Manual implementation of gradient descent by **finite difference approximation**:
    - Approximates partial derivatives by perturbing each parameter slightly (eps = 0.01) and calculating change in loss.
- **Hyperparameters**:
    - Learning rate (lr): 0.1
    - Number of epochs: 5000
    - Epsilon for finite difference: 0.01
- **Training Dataset**:
  Models the logical OR function:

x1	x2	y (output)
0	0	0
1	0	1
0	1	1
1	1	1

- **Outputs After Training**:
    - Final values of weights w1, w2, and bias b.
    - Printed predictions for all input combinations.
    - Loss values saved to a CSV file (loss_values_gates.csv).

---

4. Jupyter Notebook Analysis (graphs.ipynb)

The Jupyter notebook:

- Loads the saved loss_values_gates.csv.
- Visualizes the loss values over epochs.
- Helps in observing:
    - How fast the model converges.
    - Stability of the training process.

Main steps inside the notebook:

- Load CSV using pandas.

- Plot loss values using matplotlib.

---

5. Important Code Highlights

sigmoid(x)

Calculates the sigmoid activation.

loss_function(w1, w2, b, train_data)

Computes the mean squared error (MSE) between predicted and true outputs for given weights and bias.

train(train_data)

Main training function:

- Initializes weights and bias randomly.
- Performs manual gradient descent using finite difference approximation.
- Updates parameters and saves loss values for each epoch.

graphs.ipynb

- Reads loss values.
- Plots a line graph of loss versus number of epochs.

---

6. Outputs

At the end of training, I got this console output:

initial loss: [0.13294505]

W1 = [4.53749557], W2 = [4.53528661], bias = [-2.00368412], Loss = [0.00623779]

---

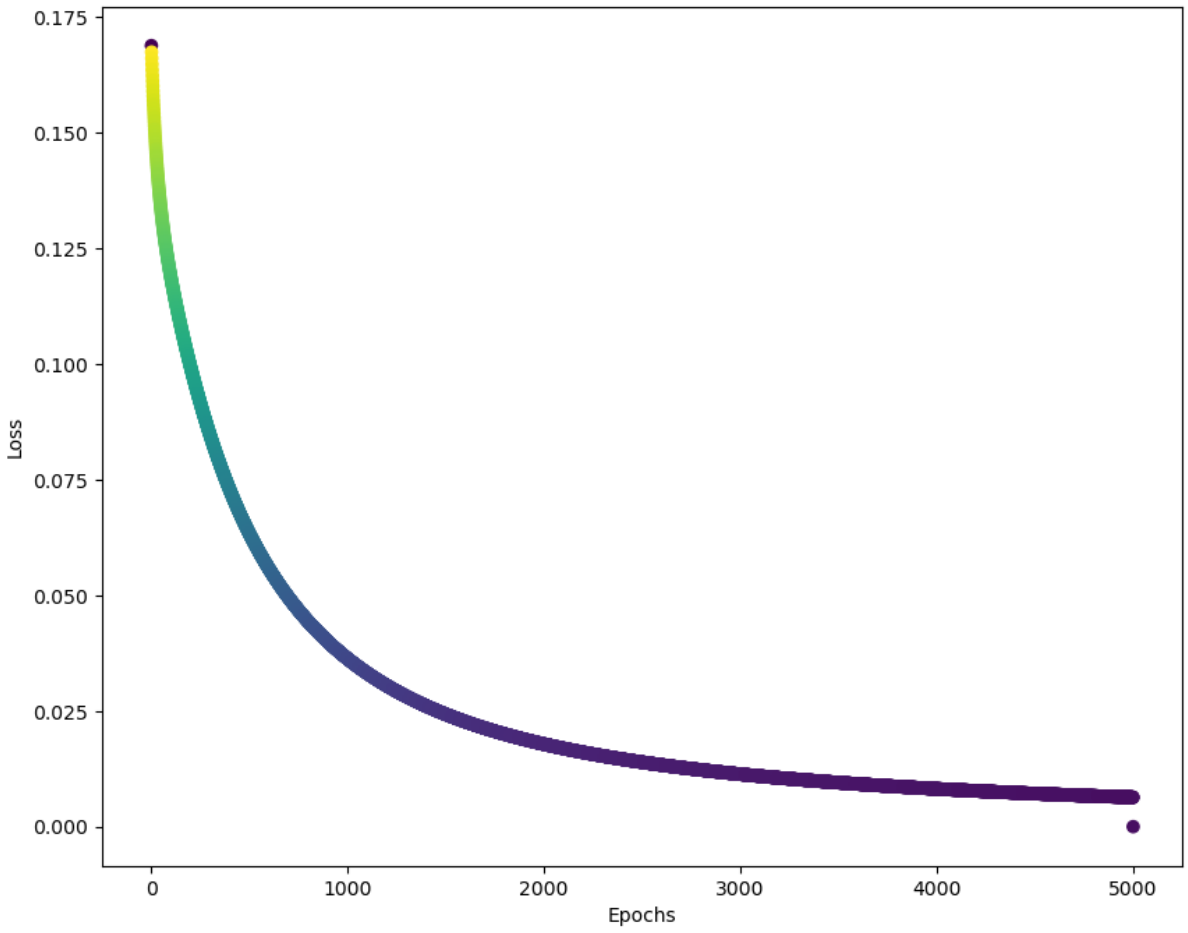
0 | 0 = [0.11881666]

0 | 1 = [0.92632779]

1 | 0 = [0.9264784]

1 | 1 = [0.99914972]

Then, in the Jupyter Notebook, the loss values recorded during training are plotted.



Conclusion from the Graph:

- The loss started relatively **high** at the beginning of training.
- It then **decreased rapidly** during the early epochs.
- After 5000 epochs, the loss **stabilized** at a very low value, close to **zero**, indicating that the model successfully **learned the logical OR gate**.
- The curve shows a **smooth descent**, suggesting that the learning rate was well-chosen and the model converged **efficiently** without oscillations.