Challenge #6 - Perceptron for NAND and XOR - Full Documentation

# 1. Overview

This document covers the full conversation and implementation details for Challenge #6, which involves implementing a simple perceptron with a sigmoid activation function and training it using the Perceptron Learning Rule to perform logical operations such as NAND and XOR.

# 2. User Prompts and Questions

Prompt 1:

Challenge #6  
1. Implement a simple neuron (a.k.a. perceptron) with two inputs and a sigmoid activation function.  
2. Use the perceptron learning rule to train the neuron to realize the following binary logic functions:  
 a. NAND  
 b. XOR  
3. Asked for platform details and explanation.

Prompt 2:

User submitted their implementation code and asked:  
In this code where is the perceptron rule of training of NAND happening?

# 3. Implementation and Explanation

The user correctly implemented a perceptron using Python and NumPy. The training data for the NAND function was correctly formatted. The sigmoid activation function and its derivative were used, and the weights and bias were updated using the perceptron learning rule.

Key lines implementing the learning rule:  
 weights += lr \* error \* x  
 bias += lr \* error  
These lines represent the core perceptron learning update step, where the model adjusts its parameters based on the error between the predicted and actual values.

# 4. Final Working Code for NAND

import numpy as np  
  
# Sigmoid and its derivative  
def sigmoid(x):  
 return 1 / (1 + np.exp(-x))  
  
def sigmoid\_derivative(x):  
 return x \* (1 - x)  
  
# Training data for NAND  
inputs = np.array([[0,0],[0,1],[1,0],[1,1]])  
targets = np.array([[1],[1],[1],[0]]) # NAND truth table  
  
# Initialize weights and bias  
weights = np.random.rand(2)  
bias = np.random.rand(1)  
  
# Learning rate  
lr = 0.1  
  
# Training loop  
for epoch in range(10000):  
 for i in range(len(inputs)):  
 # Forward pass  
 x = inputs[i]  
 y = targets[i]  
 z = np.dot(x, weights) + bias  
 output = sigmoid(z)  
  
 # Error  
 error = y - output  
  
 # Backpropagation (Perceptron Learning Rule)  
 weights += lr \* error \* x  
 bias += lr \* error  
  
# Testing  
for i in range(len(inputs)):  
 x = inputs[i]  
 z = np.dot(x, weights) + bias  
 output = sigmoid(z)  
 print(f"Input: {x}, Predicted: {output}")

# 5. Conclusion

The perceptron successfully learned the NAND function. However, the XOR function cannot be learned using a single-layer perceptron as it is not linearly separable. This limitation introduces the need for multi-layer neural networks (MLPs) which will be explored in future tasks.