**Batch Gradient descent/Vanilla Gradient Descent**

Import numpy as np

# Sample input data for the image

input\_data = np.array([0, 256, 0, 256, 0, 256, 0, 256, 0], dtype=float)

# Manually initialize weights and biases for the first hidden layer

weights = np.array([[-0.02, 0.03, 0.01, 0.04, 0.03, -0.02, 0.05, -0.02],

[-0.01, 0.02, -0.02, -0.03, -0.03, 0.04, -0.03, -0.01],

[0.03, -0.04, -0.02, -0.03, -0.02, 0.01, -0.02, -0.04],

[-0.02, 0.05, 0.03, -0.01, 0.02, -0.03, 0.02, 0.04],

[0.03, -0.02, -0.01, 0.02, -0.03, -0.01, -0.02, 0.03],

[-0.04, -0.03, -0.04, -0.01, 0.02, -0.02, -0.02, -0.03],

[0.04, 0.03, -0.04, -0.01, -0.01, -0.03, 0.03, 0.02],

[-0.03, 0.04, 0.01, -0.01, -0.01, -0.04, 0.01, 0.03],

[0.02, -0.04, 0.03, -0.04, 0.01, -0.01, -0.02, 0.03]], dtype=float)

biases = np.array([0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1], dtype=float)

# Assuming you have target values (labels) y

# For demonstration purposes, let's assume some target values

y\_true = np.array([1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0], dtype=float)

# Define a learning rate and number of iterations

learning\_rate = 0.001

n\_iterations = 1000

# Batch Gradient Descent

for iteration in range(n\_iterations):

# Forward pass

weighted\_sum = np.dot(input\_data, weights) + biases

# print(weighted\_sum[0])

first\_hidden\_activations = np.maximum(0, weighted\_sum) # Applying ReLU activation function

print(first\_hidden\_activations)

# Compute mean squared error

mse = np.mean((first\_hidden\_activations - y\_true) \*\* 2)

#rint(mse[0])

# Backward pass (compute gradients)

gradients = 2 / len(y\_true) \* np.dot((first\_hidden\_activations - y\_true), weights.T)

print(gradients[0])

# Update weights and biases using the mean gradient across the entire batch

For i in range(weights.shape[0]):

weights[i, :] -= learning\_rate \* np.mean(gradients[i] \* input\_data)

biases -= learning\_rate \* np.sum(gradients)

# Print the current iteration's mean squared error

if iteration % 100 == 0:

print(f"Iteration {iteration}, Mean Squared Error: {mse}")

# Print the final weights and biases

print("\nFinal Weights:")

print(weights)

print("\nFinal Biases:")

print(biases)