Neural NET TRAINING

Import numpy as np

Import matplotlib.pyplot as plt

# Given values

A1 = np.array([[0.05], [0.10]])

W2 = np.array([[0.15, 0.20], [0.25, 0.30]])

W3 = np.array([[0.40, 0.45], [0.50, 0.55]])

B2 = np.array([[0.35], [0.35]])

B3 = np.array([[0.60], [0.60]])

# Given R vector

R = np.array([[0.01], [0.99]])

# Learning rate

Alpha = 0.5

# Sigmoid activation function and its derivative

Def sigmoid(x):

Return 1 / (1 + np.exp(-x))

Def sigmoid\_derivative(x):

Return sigmoid(x) \* (1 – sigmoid(x))

# Number of epochs

Epochs = 2

# Initialize an array to store total errors

Errors = np.zeros(epochs)

# Loop through epochs

For epoch in range(epochs):

Print(f”\nEpoch {epoch + 1}:”)

# Forward pass calculations

Z2 = np.dot(W2, A1) + b2

A2 = sigmoid(Z2)

Z3 = np.dot(W3, A2) + b3

A3 = sigmoid(Z3)

# Print forward pass values

Print(f”Z2: {Z2}”)

Print(f”A2: {A2}”)

Print(f”Z3: {Z3}”)

Print(f”A3: {A3}”)

Print(f”W2: {W2}”)

Print(f”W3: {W3}”)

Print(f”b2: {b2}”)

Print(f”b3: {b3}”)

# Calculate Deltas

D3 = (A3 – R) \* sigmoid\_derivative(Z3)

D2 = np.dot(W3.T, D3) \* sigmoid\_derivative(Z2)

# Print deltas

Print(f”D3: {D3}”)

Print(f”D2: {D2}”)

# Calculate gradients

Grad\_W3 = np.dot(D3, A2.T)

Grad\_b3 = D3

Grad\_W2 = np.dot(D2, A1.T)

Grad\_b2 = D2

# Print gradients

Print(f”grad\_W3: {grad\_W3}”)

Print(f”grad\_b3: {grad\_b3}”)

Print(f”grad\_W2: {grad\_W2}”)

Print(f”grad\_b2: {grad\_b2}”)

# Update weights and biases

W2 = W2 – alpha \* grad\_W2

B2 = b2 – alpha \* grad\_b2

W3 = W3 – alpha \* grad\_W3

B3 = b3 – alpha \* grad\_b3

# Calculate total error and store in array

Total\_error = np.sum((A3 – R) \*\* 2) / 2

Errors[epoch] = total\_error

# Plot the graph of total errors

Plt.figure()

Plt.plot(range(1, epochs + 1), errors, ‘-o’)

Plt.xlabel(‘Epochs’)

Plt.ylabel(‘Total Error’)

Plt.title(‘Total Error vs. Epochs’)

Plt.grid(True)

Plt.show()