|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **Hope Foundation’s**  **Finolex Academy of Management and Technology, Ratnagiri** | | | | | |
| **Department of Computer Science and Engineering (AIML)** | | | | | |
| Subject name: Machine Learning | | | | | | Subject Code: CSL604 | |
| Class | | TE CSE | | | Semester –VI (CBCGS) | Academic year: 2024-25 | |
| Name of Student | | GiriPrasath K | | | | **QUIZ Score :6** | |
| Roll No | | 29 | | Experiment No. | | 08 | |
| Title: **To implement Error Backpropagation Perceptron Training Algorithm.** | | | | | | | |
|  | | | | | | | |
| **1. Lab objectives applicable:**  **LOB1:**To introduce platforms such as Anaconda, COLAB suitable to Machine Learning.  **LOB3:** To develop Neural Network based learning models | | | | | | | |
| **2. Lab outcomes applicable:**  **LO1:**Implement various Machine learning models.  **LO3:** Implement Neural Network based models. | | | | | | | |
| **3. Learning Objectives:**   1. To learn backpropagation algorithm used to reduce error in neural network model. | | | | | | | |
| **4. Practical applications of the assignment/experiment:**  To demonstrate error backpropagation algorithm. | | | | | | | |
| **5. Prerequisites**:   1. Python language | | | | | | | |
| **6. Minimum Hardware Requirements**:-  I series processor, RAM 4GB,  **7. Software Requirements:-**  Colab or Visual Studio or Jupyter notebook (Anaconda) | | | | | | | |
| **8. Quiz Questions :** [**https://docs.google.com/forms/d/e/1FAIpQLScZF7jYMVvOdt0pmK0EJlBP1nol1qZ-6Gv\_gDUNM4tAxrqLDw/viewform?usp=dialog**](https://docs.google.com/forms/d/e/1FAIpQLScZF7jYMVvOdt0pmK0EJlBP1nol1qZ-6Gv_gDUNM4tAxrqLDw/viewform?usp=dialog) | | | | | | | |
| **9. Experiment Evaluation:** | | | | | | | |
| **Sr. No.** | **Parameters** | | | | | **Marks obtained** | **Out of** |
| **1** | Technical Understanding (Assessment may be done based on Q & A **or** any other relevant method.) Teacher should mention the other method used - | | | | | 6 | 6 |
| **2** | Lab Performance | | | | |  | 2 |
| **3** | Punctuality | | | | |  | 2 |
| **Date of performance (DOP)** | | |  | | **Total marks obtained** |  | **10** |

**Signature of Faculty**

**10. Theory:**

**Input for XOR Gate: X=**

**Predicted Output , y=**

**Initialize Random Weights and Biases: ( as per program)**

* **Initial Weights (W1**):[
* **Initial Biases (b1):** [-0.62747958 -0.30887855 -0.20646505]
* **Initial Weights (W2):**
* **Initial Biases (b2):** [-0.5910955]

**Learning Rate=0.1**

**Epoch 1:**

**11. Installation Steps / Performance Steps and Results –**

**Source Code:**

import numpy as np

# Sigmoid activation function and its derivative

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

    return x \* (1 - x)

# Training Data for XOR

X = np.array([[0, 0],

              [0, 1],

              [1, 0],

              [1, 1]])

# Expected XOR output

y = np.array([[0], [1], [1], [0]])

# Initialize weights and biases randomly

np.random.seed(1)

input\_size = 2

hidden\_size = 3

output\_size = 1

W1 = np.random.uniform(-1, 1, (input\_size, hidden\_size))  # Weights for input -> hidden

b1 = np.random.uniform(-1, 1, (1, hidden\_size))  # Bias for hidden layer

W2 = np.random.uniform(-1, 1, (hidden\_size, output\_size))  # Weights for hidden -> output

b2 = np.random.uniform(-1, 1, (1, output\_size))  # Bias for output layer

print("Initial Weights (W1):")

print(W1)

print("\nInitial Biases (b1):")

print(b1)

print("\nInitial Weights (W2):")

print(W2)

print("\nInitial Biases (b2):")

print(b2)

# Training parameters

epochs = 10000

learning\_rate = 0.1

# Training loop

for epoch in range(epochs):

    # ---- Forward Pass ----

    hidden\_input = np.dot(X, W1) + b1  # Net input to hidden layer

    hidden\_output = sigmoid(hidden\_input)  # Activation of hidden layer

    final\_input = np.dot(hidden\_output, W2) + b2  # Net input to output layer

    predicted\_output = sigmoid(final\_input)  # Activation of output layer

    # ---- Compute Error ----

    error = (y - predicted\_output)

    #print(final\_output)

    # ---- Backpropagation ----

    # Compute gradients for output layer

    error2 = error \* sigmoid\_derivative(predicted\_output)

    #print (d\_output)

    # Compute gradients for hidden layer

    error1 = np.dot(error2, W2.T) \* sigmoid\_derivative(hidden\_output)

    # ---- Update Weights and Biases ----

    W2 += np.dot(hidden\_output.T, error2) \* learning\_rate

    b2 += np.sum(error2, axis=0, keepdims=True) \* learning\_rate

    W1 += np.dot(X.T, error1) \* learning\_rate

    b1 += np.sum(error1, axis=0, keepdims=True) \* learning\_rate

    # Print loss every 1000 epochs

    if epoch % 1000 == 0:

        loss = np.mean(np.abs(error))

        print(f"Epoch {epoch}: Loss = {loss:.4f}")

        print("Updated Weights (W1):")

        print(W1)

        print("\nUpdated Biases (b1):")

        print(b1)

        print("\nUpdated Weights (W2):")

        print(W2)

        print("\nUpdated Biases (b2):")

        print(b2)

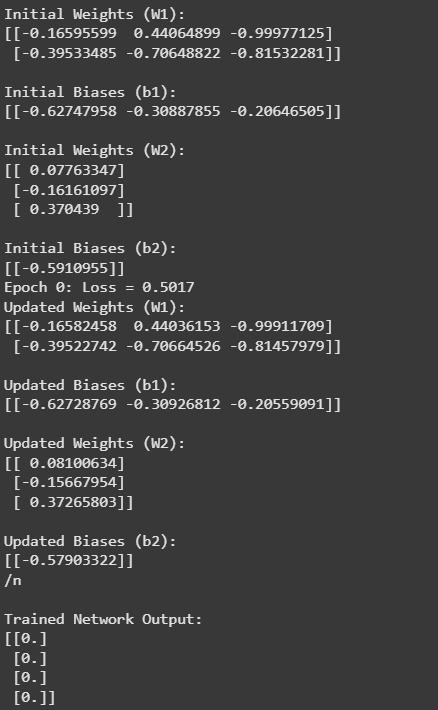
    break

# ---- Testing the trained network ----

print("\nTrained Network Output:")

print(predicted\_output.round())  # Rounded output for XOR operation

**Output:**

****

**12. Learning Outcomes Achieved**

1. Students are able to perform error backpropagation in neural network to reduce overall loss and improve accuracy.

**13. Conclusion:**

**1. Applications of the Studied Technique in Industry**

Backpropagation is widely used in computer vision, speech recognition, and natural language processing to train deep learning models efficiently. Industries leverage it for autonomous vehicles, fraud detection, and medical diagnostics to enhance decision-making.

**2. Engineering Relevance**

This technique is fundamental in AI and automation, enabling engineers to design intelligent systems that learn from data. It is crucial in optimizing neural networks for pattern recognition, predictive analytics, and real-time adaptive systems.

**3. Skills Developed**

This experiment enhances mathematical reasoning through gradient-based optimization and strengthens coding proficiency in implementing neural networks. It also builds problem-solving skills for designing and fine-tuning AI models.

**14. References**:

1. Nathalie Japkowicz & Mohak Shah, ―Evaluating Learning Algorithms: A Classification Perspective‖, Cambridge.
2. Marc Peter Deisenroth, Aldo Faisal, Cheng Soon Ong, ―Mathematics for machine learning‖
3. Samir Roy and Chakraborty, ―Introduction to soft computing‖, Pearson Edition.
4. Ethem Alpaydın, ―Introduction to Machine Learning‖, MIT Press McGraw-Hill Higher

Education

1. Peter Flach, ―Machine Learning‖, Cambridge University Press
2. Tom M. Mitchell, ―Machine Learning‖, McGraw Hill
3. Kevin P. Murphy, ―Machine Learning ― A Probabilistic Perspective‖, MIT Press