|  |  |  |  |
| --- | --- | --- | --- |
|  | **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. | 07 |
| Title: **To implement the Single Layer Perceptron Learning 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 implement AND gate, OR gate, NOR gate logic using Single Layer Perceptron. | | | | | |
| **4. Practical applications of the assignment/experiment:**  To demonstrate the working of single layer perceptron. | | | | | |
| **5. Prerequisites**:  1. Python language | | | | | |
| 1. **Minimum Hardware Requirements**:-   I series processor, RAM 4GB,   1. **Software Requirements:-**   Colab or Visual Studio or Jupyter notebook (Anaconda) | | | | | |
| **8. Quiz Questions :**  [**https://docs.google.com/forms/d/e/1FAIpQLScraln9UQcqqDYGz79Yu8zvJw\_RAiOU7WI3Xzfr7eeLsxtglQ/viewfor m?usp=dialog**](https://docs.google.com/forms/d/e/1FAIpQLScraln9UQcqqDYGz79Yu8zvJw_RAiOU7WI3Xzfr7eeLsxtglQ/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:**

# Theory of Single-Layer Perceptron

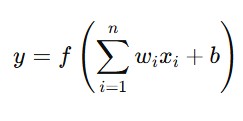
A perceptron is a computational model inspired by the way biological neurons work. It takes multiple inputs, applies weights, sums them, adds a bias, and passes the result through an activation function to produce an output.

# Architecture

* Input Layer: Features or input values (x1,x2,…….,xn)
* Weights (w): Each input has a corresponding weight (w1,w2,……..,wn)
* Bias (b): Adjusts the decision boundary
* Output Layer: Single neuron that provides the final prediction

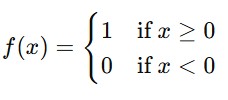
# Mathematical Model

The perceptron computes:



**Activation Function**

The activation function is often a step function for binary classification:



**2. Learning Algorithm (Perceptron Learning Rule)**

The perceptron uses supervised learning and adjusts weights based on the error between predicted and actual values.

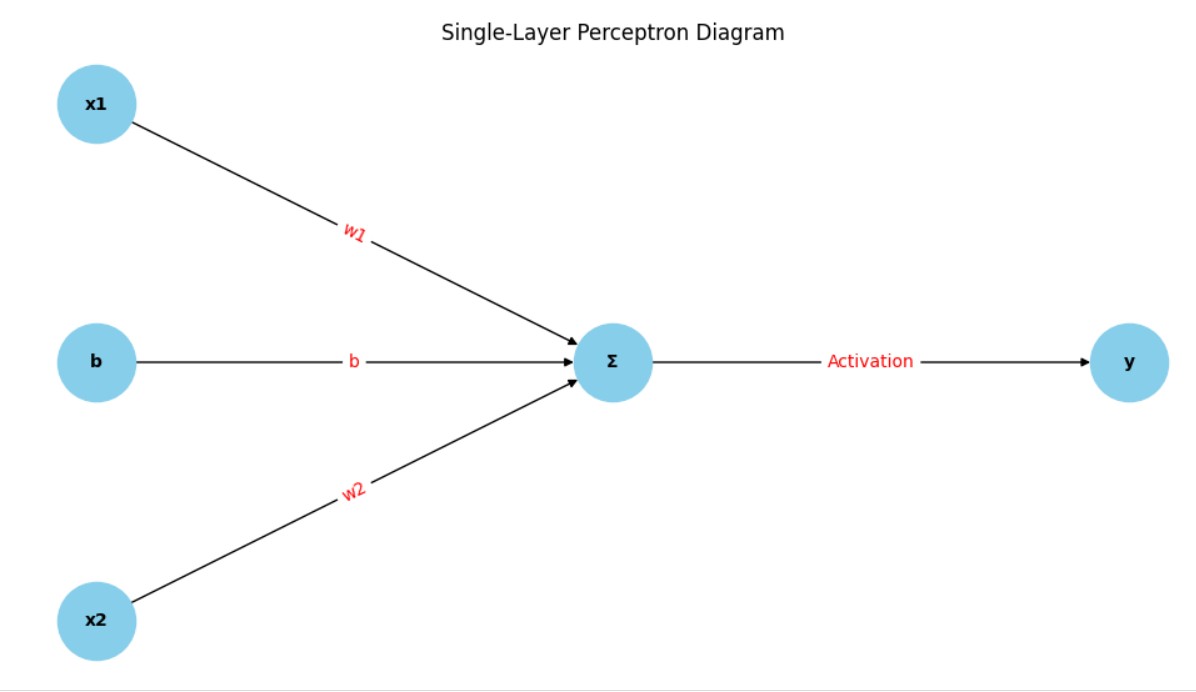
# Algorithm Steps

1. Initialize: Weights wi and bias b to small random values (e.g., 0 or small random numbers).
2. For each training sample (x, y):
   * Calculate the predicted output: y^=f(∑wixi+b)
   * Calculate the error: error=y−y^
   * Update weights and bias if there's an error:

wi=wi+ηerrorxi b=b+ηerror Where η is the learning rate (e.g., 0.1).

1. Repeat until the error becomes zero or for a fixed number of epochs.

**3.Diagram**



1. **Detailed Step-by-Step Perceptron Calculations** Initial Parameters:
   * Weights: w1 = 0.2, w2 = -0.4
   * Bias: b = 0.1
   * Learning Rate: lr = 0.5
   * Input: x1 = 1, x2 = 1

# AND Gate (Target = 1) Epoch 1

1. **Net Input Calculation:** ynet =(0.2×1)+(−0.4×1)+0.1=−0.1
2. **Activation:** y=f(−0.1)=0 3. **Error Calculation:** error=1−0=1 4. **Weight Updates:** w1=0.2+(0.5×1×1)=0.7 w2=−0.4+(0.5×1×1)=0.1

5. **Bias Update:** b=0.1+(0.5×1)=0.6

**Updated Parameters:** w1 = 0.7, w2 = 0.1, b = 0.6

# Epoch 2 to 5

For each of the remaining epochs:

* yne t=(0.7×1)+(0.1×1)+0.6=1.4
* Activation: y=f(1.4)=1
* Error: 1−1=0

**Parameters remain:** w1 = 0.7, w2 = 0.1, b = 0.6

# Final AND Gate Parameters: w1 = 0.7, w2 = 0.1, b = 0.6 OR Gate (Target = 1)

Since the input and target are the same as the AND gate, calculations follow the same pattern. Final OR Gate Parameters:

w1 = 0.7, w2 = 0.1, b = 0.6

# NOR Gate (Target = 1) Epoch 1

1. **Net Input Calculation:**

ynet=(0.2×1)+(−0.4×1)+0.1 2. **Activation:** y=f(−0.1)=0 3. **Error Calculation:** error=1−0=1

1. **Weight Updates:**

w1=0.2+(0.5×1×1)=0.7 w2=−0.4+(0.5×1×1)=0.1

1. **Bias Update:**

b=0.1+(0.5×1)=0.6

**Updated Parameters:** w1 = 0.7, w2 = 0.1, b = 0.6

# Epoch 2 to 5

For the remaining epochs:

 ynet=1.4→ y=1→ error=0

**Parameters remain:** w1 = 0.7, w2 = 0.1, b = 0.6

**Final NOR Gate Parameters:**

w1 = 0.7, w2 = 0.1, b = 0.6

# Final Summary

|  |  |  |
| --- | --- | --- |
| Gate | Weights (w1, w2) | Bias |
| AND | (0.7, 0.1) | 0.6 |
| OR | (0.7, 0.1) | 0.6 |
| NOR | (0.7, 0.1) | 0.6 |

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

import numpy as np

# Define the perceptron class

class Perceptron:

    def \_\_init\_\_(self, input\_size, learning\_rate=0.1):

        self.weights = np.zeros(input\_size)  # Initialize weights to zeros

        self.bias = 0  # Initialize bias to zero

        self.learning\_rate = learning\_rate  # Learning rate

    # Activation function (step function)

    def activation(self, x):

        return 1 if x >= 0 else 0

    # Forward pass: calculate the weighted sum and apply the activation function

    def forward(self, inputs):

        weighted\_sum = np.dot(inputs, self.weights) + self.bias

        return self.activation(weighted\_sum)

    # Train the perceptron using the training data

    def train(self, inputs, targets, epochs=10):

        for epoch in range(epochs):

            for x, target in zip(inputs, targets):

                output = self.forward(x)

                error = target - output  # Calculate the error

                # Update weights and bias based on the error

                self.weights += self.learning\_rate \* error \* x

                self.bias += self.learning\_rate \* error

            print(f"Epoch {epoch+1}/{epochs}, Weights: {self.weights}, Bias: {self.bias}")

# Define input data for AND, OR, and NOR gates

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

# Target outputs for AND, OR, and NOR gates

and\_output = np.array([0, 0, 0, 1])  # AND Gate

or\_output = np.array([0, 1, 1, 1])   # OR Gate

nor\_output = np.array([1, 0, 0, 0])  # NOR Gate

# Create perceptron instances for AND, OR, and NOR gates

and\_perceptron = Perceptron(input\_size=2)

or\_perceptron = Perceptron(input\_size=2)

nor\_perceptron = Perceptron(input\_size=2)

# Train the perceptrons

print("Training for AND Gate:")

and\_perceptron.train(inputs, and\_output)

print("\nTraining for OR Gate:")

or\_perceptron.train(inputs, or\_output)

print("\nTraining for NOR Gate:")

nor\_perceptron.train(inputs, nor\_output)

# Test the perceptrons on all inputs

print("\nTesting AND Gate:")

for x in inputs:

    print(f"Input: {x}, Output: {and\_perceptron.forward(x)}")

print("\nTesting OR Gate:")

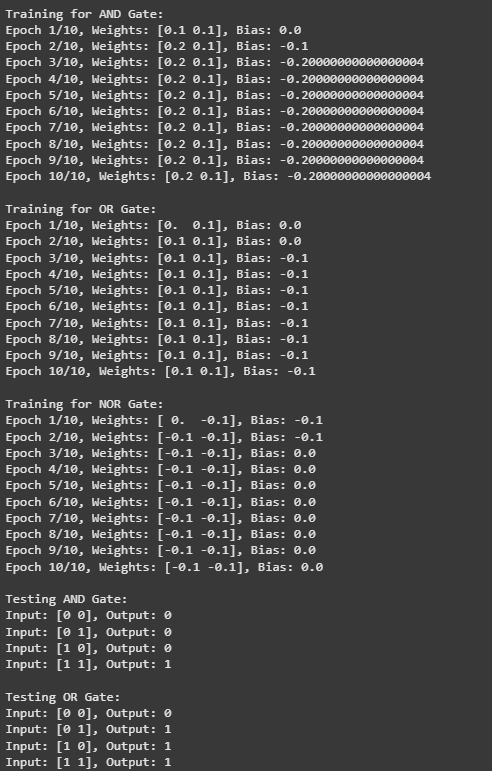
for x in inputs:

    print(f"Input: {x}, Output: {or\_perceptron.forward(x)}")

print("\nTesting NOR Gate:")

for x in inputs:

    print(f"Input: {x}, Output: {nor\_perceptron.forward(x)}")

**Output : **

1. **Learning Outcomes Achieved** 
   1. Students are able to perform implement McCulloch Pitts model.

1. **Conclusion:**

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

From an engineering perspective, the SLP serves as a foundational concept for more complex neural network models. It helps in understanding the working of artificial neurons and how learning is carried out via weight adjustments. Its relevance in various engineering fields like control systems, robotics, and signal processing lies in its ability to model and make predictions based on input data. Although SLPs are limited to linearly separable problems, they offer a simple introduction to neural networks and machine learning principles.

* 1. **Engineering Relevance**

From an engineering perspective, the SLP serves as a foundational concept for more complex neural network models. It helps in understanding the working of artificial neurons and how learning is carried out via weight adjustments. Although SLPs are limited to linearly separable problems, they offer a simple introduction to neural networks and machine learning principles.

* 1. **Skills Developed**

By implementing the SLP model, students and engineers gain programming skills, logical thinking skills, mathematical skills, and understanding of AI basics.

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