Batch: A2 Roll No.: 16010121045

Experiment No. : 03

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Perceptron learning algorithm.

Objective: To write a program to implement perceptron learning rule

Expected Outcome of Experiment:

CO2 : Analyze various training algorithms of neural network and its architectures

Books/ Journals/ Websites referred:

Pre Lab/ Prior Concepts:

Learning in the context of machine learning refers to the process by which a system (algorithm, model) improves its performance on a task based on experience (data). There are different types of learning:

Types of learning

Supervised Learning: In supervised learning, the algorithm is trained on a labeled dataset, where the input data is paired with the correct output or target. The goal is to learn a mapping from inputs to outputs.

Unsupervised Learning: Unsupervised learning involves learning from unlabeled data. The algorithm tries to find patterns, structures, or relationships within the data without explicit target values.

Reinforcement Learning: Reinforcement learning involves an agent that learns to interact with an environment to maximize a reward signal. The agent takes actions and learns from the consequences of those actions.

Perceptron learning rule.

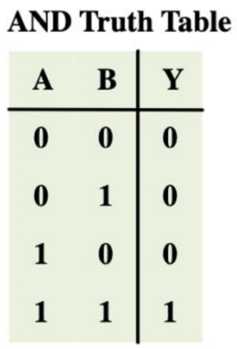
The Perceptron Learning Rule is an algorithm for training a single-layer perceptron, which is a simple neural network used for binary classification. The perceptron learning rule aims to adjust the weights of the perceptron's inputs to correctly classify input data into one of two classes.

Steps of Perceptron learning algorithm/approach for binary classification

1. Initialization: Initialize the weights and bias to small random values.
2. Input Processing: For each input vector, append a bias term (usually set to 1) to the beginning of the input vector.
3. Weighted Sum: Calculate the weighted sum of inputs and weights.
4. Weighted Sum = (Weight 1 * Input 1) + (Weight 2 * Input 2) + ... + Bias * Bias Weight
5. Activation Function: Apply an activation function (often a step function) to the weighted sum. The perceptron output is 1 if the activation condition is met, otherwise, it's 0.
6. Error Calculation: Calculate the error by subtracting the predicted output from the true target value.
7. Weight Update: Update the weights and bias using the following update rule:
8. New Weight = Old Weight + Learning Rate * Error * Input
9. New Bias Weight = Old Bias Weight + Learning Rate * Error * Bias
10. Repeat: Repeat steps 3-6 for a certain number of epochs or until the error becomes sufficiently small.
11. Termination: Stop training when the error is minimized or after a fixed number of epochs.

Single layer perceptron network for each logic functions

AND Logic:



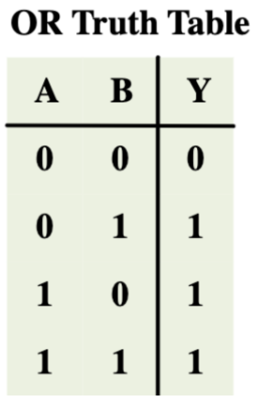
AND Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Activation Function: Step function (1 if weighted sum >= 0, else 0)
Explanation:

- For the AND logic, the perceptron uses a step function as the activation function.
- The weighted sum of inputs and weights is calculated.
- If the weighted sum is greater than or equal to 0, the perceptron outputs 1; otherwise, it outputs 0.

OR Logic:



OR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Activation Function: Step function (1 if weighted sum >= 0, else 0)
Explanation:

- For the OR logic, similar to the AND logic, a step function is used as the activation function.
- The perceptron calculates the weighted sum of inputs and weights.
- If the weighted sum is greater than or equal to 0, the perceptron outputs 1; otherwise, it outputs 0.

NOT Logic:

NOT Truth Tabl

A	B
0	1
1	0

Activation Function: Step function (1 if weighted sum >= 0, else 0)

Explanation:

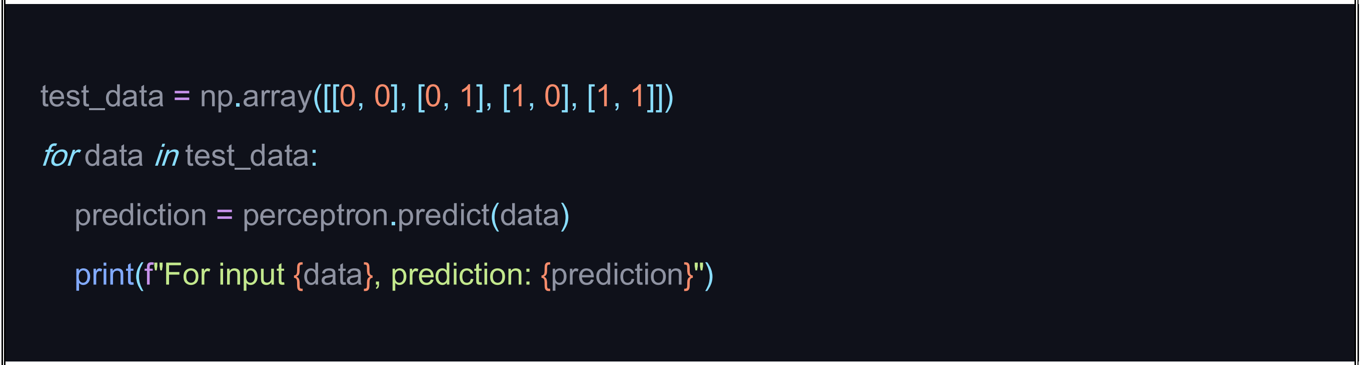
- For the NOT logic, a step function is also used as the activation function.
- The perceptron computes the weighted sum of the input and weight.
- If the weighted sum is greater than or equal to 0, the perceptron outputs 1; otherwise, it outputs 0.

In each case, the activation function determines the output of the perceptron based on the calculated weighted sum. The perceptron learning algorithm adjusts the weights and bias to minimize the error between the predicted output (determined by the activation function) and the target output. This iterative process helps the perceptron learn the correct parameters to classify inputs according to the specified logic function.

Implementation Details:

```
import numpy as np


class Perceptron:
   def __init__(self, input_size, learning_rate=0.1, epochs=100):
      self.weights = np.random.rand(input_size + 1)  # +1 for the bias weight
      self.learning_rate = learning_rate
      self.epochs = epochs

   def activate(self, x):
      # Activation function (step function)
      return 1 if x >= 0 else 0
```

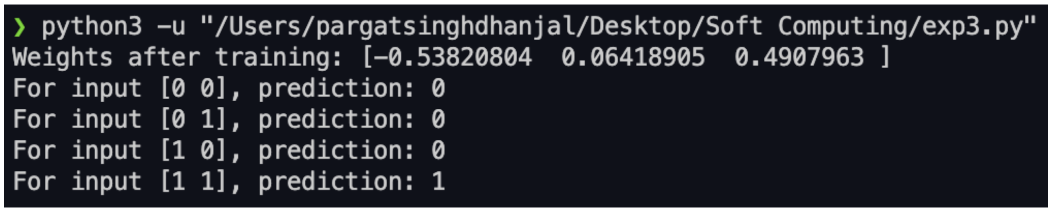
```python
    def train(self, X, y):
        X = np.insert(X, 0, 1, axis=1)  # Adding bias input (always 1)

        for _ in range(self.epochs):
            total_error = 0
            for i in range(X.shape[0]):
                prediction = self.activate(np.dot(self.weights, X[i]))
                error = y[i] - prediction
                total_error += abs(error)
                self.weights += self.learning_rate * error * X[i]
            if total_error == 0:
                break

    def predict(self, X):
        X = np.insert(X, 0, 1)  # Adding bias input (always 1)
        return self.activate(np.dot(self.weights, X))

# Example usage
if __name__ == "__main__":
    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    y = np.array([0, 0, 0, 1])

    perceptron = Perceptron(input_size=2)
    perceptron.train(X, y)

    print("Weights after training:", perceptron.weights)
```



```
test_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
for data in test_data:
    prediction = perceptron.predict(data)
    print(f"For input {data}, prediction: {prediction}")
```

.

Output:



```
❯ python3 -u "/Users/pargatsinghdhanjal/Desktop/Soft Computing/exp3.py"
Weights after training: [-0.53820804  0.06418905  0.4907963 ]
For input [0 0], prediction: 0
For input [0 1], prediction: 0
For input [1 0], prediction: 0
For input [1 1], prediction: 1
```

Conclusion: In this experiment we learnt that Single layer perceptrons employ activation functions like step functions to convert weighted inputs into binary outputs, as seen in logic functions such as AND, OR, and NOT. Through learning, perceptrons adjust weights to effectively categorize input data based on predetermined patterns.

Post Lab Descriptive Questions :

1. **How is linear separability implemented using perceptron networking training**
   Linear separability is the property of data points from different classes being able to be separated by a linear decision boundary. Perceptrons are particularly effective at handling linearly separable problems. The perceptron learning rule adjusts the weights and bias during training to find the correct combination that aligns with this linear separation. The steps involve initializing weights, iterating through the training data, calculating the weighted sum, applying the activation function, calculating errors, and updating weights based on the errors. Through this iterative process, the perceptron "learns" the optimal parameters that lead to linear separation, thereby achieving classification accuracy.

2. **Mention the application of perceptron network.**

Perceptron networks find applications in various domains, particularly in situations where data can be linearly classified:

- Pattern Recognition: Perceptrons can be used to recognize patterns, such as character recognition in OCR systems.
- Binary Classification: They are suitable for binary classification tasks, such as spam detection or medical diagnosis (e.g., classifying tumors as malignant or benign).
- Control Systems: Perceptrons can be used in control systems, such as robot navigation, to make decisions based on sensor inputs.
- Logical Functions: As seen in the examples earlier, perceptrons can implement logical functions like AND, OR, and NOT.
- Neural Network Foundations: Perceptrons are the building blocks of more complex neural networks and have historical significance in neural network research and development.

However, it's important to note that perceptrons have limitations when dealing with problems that are not linearly separable. In practice, more advanced neural network architectures, such as multi-layer perceptrons (MLPs) and convolutional neural networks (CNNs), are used for tasks that involve complex data patterns and non-linear relationships.

.

Date: _____________ **Signature of faculty in-charge**