Experiment 03: Implement Single Layer Perceptron Learning Algorithm..

Learning Objective: Implement Single Layer Perceptron Learning Algorithm.

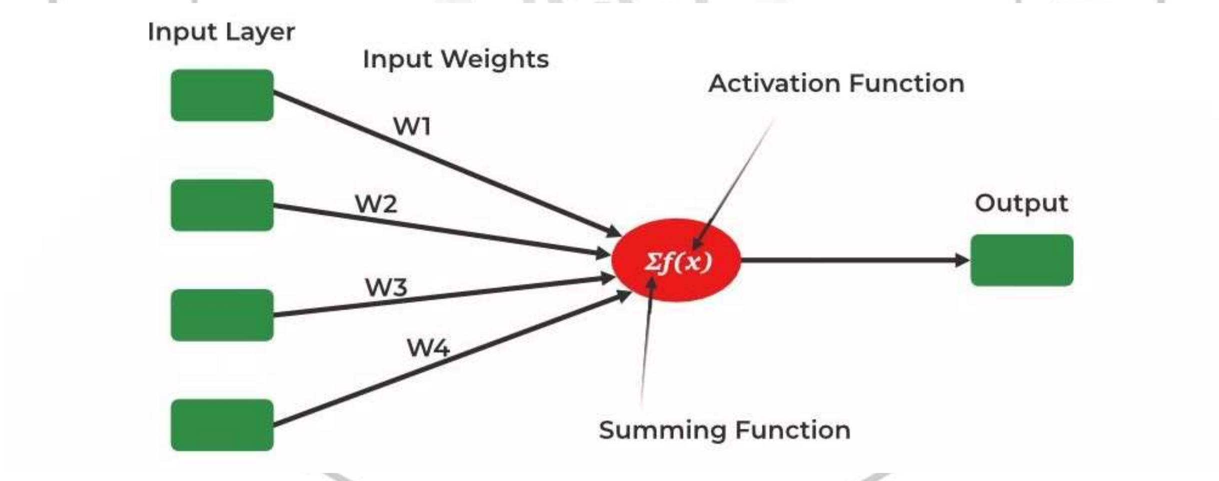
Tools: Python

Theory:

The single layer perceptron is a type of artificial neural network that can be used for supervised learning. It consists of a single layer of artificial neurons, each of which is connected to the inputs of the network.

The goal of the single layer perceptron is to classify input data into two or more classes. It does this by learning a set of weights that are applied to the inputs of the network, which allow the network to make a decision about which class the input data belongs to.

The learning algorithm used for the single layer perceptron is called the perceptron learning rule. It is a supervised learning algorithm that adjusts the weights of the network based on the error between the predicted output of the network and the actual output.

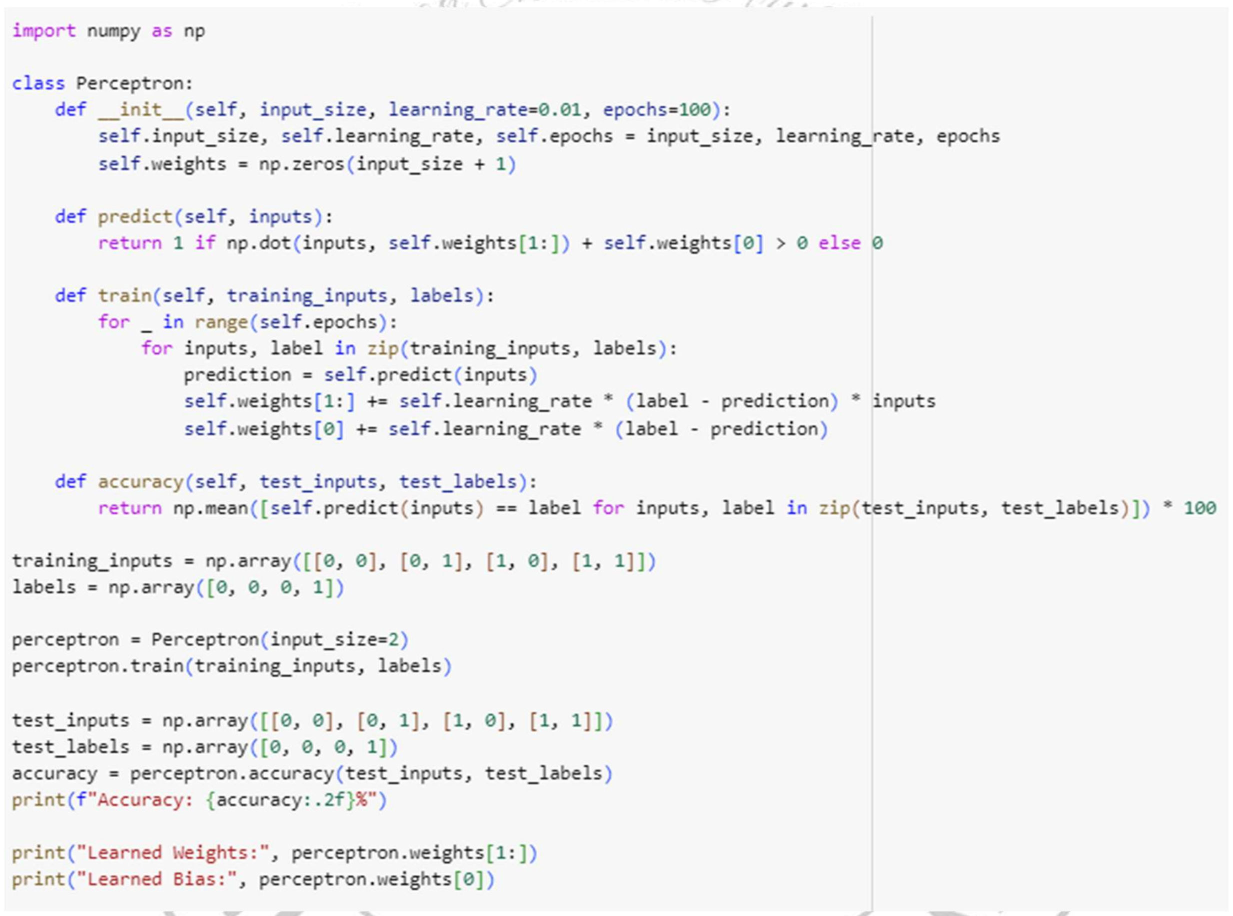


The perceptron learning rule can be summarized as follows:

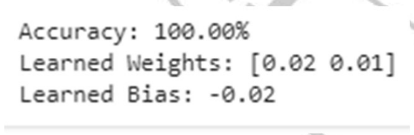
1. Initialize the weights of the network to small random values.
2. For each training example:
   a. Compute the output of the network based on the current weights.
   b. Compare the predicted output to the actual output.

   c. If the predicted output is incorrect, adjust the weights of the network in the direction of the correct output.
3. Repeat step 2 until the network achieves a satisfactory level of performance on the training data.

Implementation Code



```python
import numpy as np

class Perceptron:
    def __init__(self, input_size, learning_rate=0.01, epochs=100):
        self.input_size, self.learning_rate, self.epochs = input_size, learning_rate, epochs
        self.weights = np.zeros(input_size + 1)

    def predict(self, inputs):
        return 1 if np.dot(inputs, self.weights[1:]) + self.weights[0] > 0 else 0

    def train(self, training_inputs, labels):
        for _ in range(self.epochs):
            for inputs, label in zip(training_inputs, labels):
                prediction = self.predict(inputs)
                self.weights[1:] += self.learning_rate * (label - prediction) * inputs
                self.weights[0] += self.learning_rate * (label - prediction)

    def accuracy(self, test_inputs, test_labels):
        return np.mean([self.predict(inputs) == label for inputs, label in zip(test_inputs, test_labels)]) * 100

training_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
labels = np.array([0, 0, 0, 1])

perceptron = Perceptron(input_size=2)
perceptron.train(training_inputs, labels)

test_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
test_labels = np.array([0, 0, 0, 1])
accuracy = perceptron.accuracy(test_inputs, test_labels)
print(f"Accuracy: {accuracy:.2f}%")

print("Learned Weights:", perceptron.weights[1:])
print("Learned Bias:", perceptron.weights[0])
```



```
Accuracy: 100.00%
Learned Weights: [0.02 0.01]
Learned Bias: -0.02
```

Result and discussion:

____
____
____
____

Learning Outcomes: Students should have the ability to

LO 3.1: Ability to understand activation functions.
LO 3.2: Ability to implement Single Layer Perceptron model.

Course Outcomes: Understand and implement Single Layer Perceptron.

Conclusion:

____
____
____
____

Viva Questions:

Q. 1 What are activation functions? What are the types of activation functions?
Q.2 How does an SLP work?
Q.3 What are the limitations of an SLP?

For Faculty Use

Correction Parameters	Formative Assessmen t [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				