Experiment No. 7
Implementation of Single Layer Perceptron Learning Algorithm
Date of Performance:
Date of Submission:
Marks:
Sign:

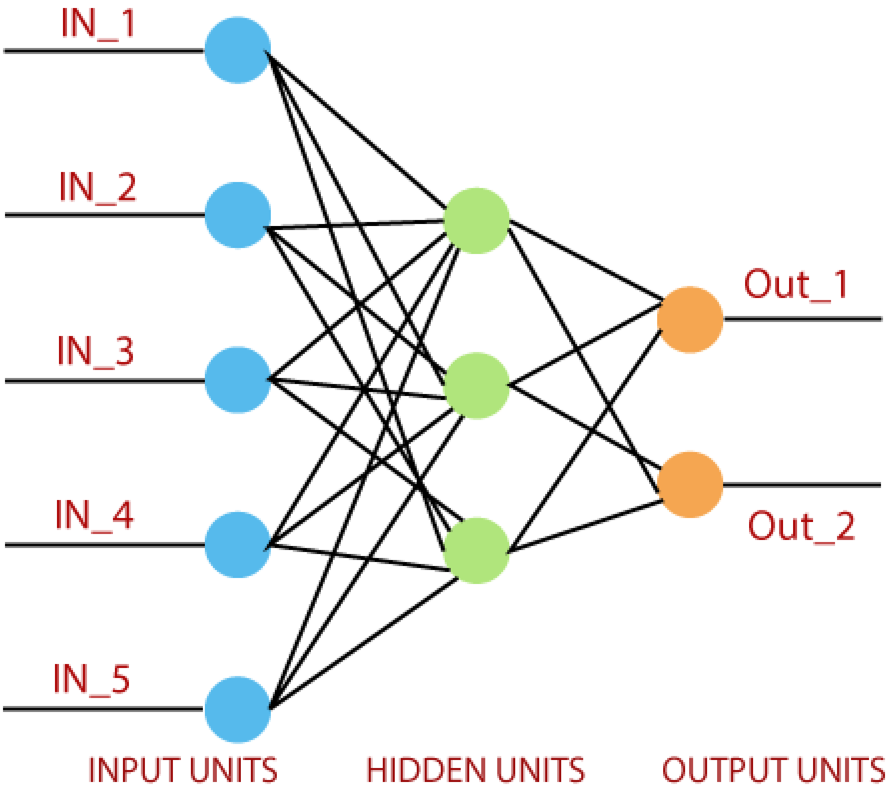
Aim: Implementation of Single Layer Perceptron Learning Algorithm

Objective: Able to implement and understand the aspects of Single Layer Perceptron Learning Algorithm.

Theory:

The perceptron is a single processing unit of any neural network. **Frank Rosenblatt** first proposed in **1958** is a simple neuron which is used to classify its input into one or two categories. Perceptron is a linear classifier, and is used in supervised learning. It helps to organize the given input data.

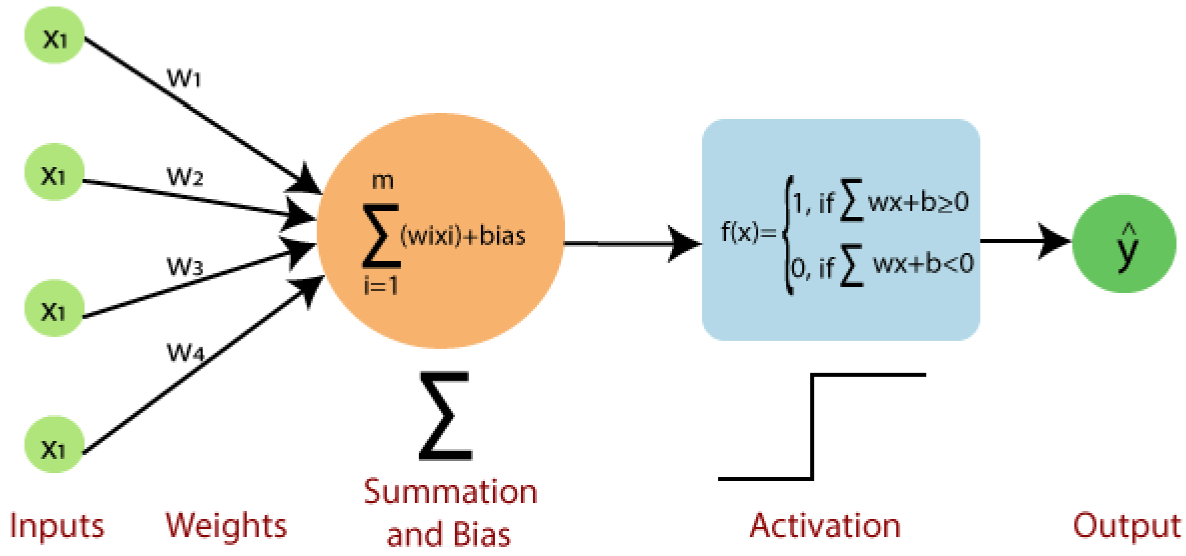
A perceptron is a neural network unit that does a precise computation to detect features in the input data. Perceptron is mainly used to classify the data into two parts. Therefore, it is also known as **Linear Binary Classifier**.



Perceptron uses the step function that returns +1 if the weighted sum of its input 0 and -1.

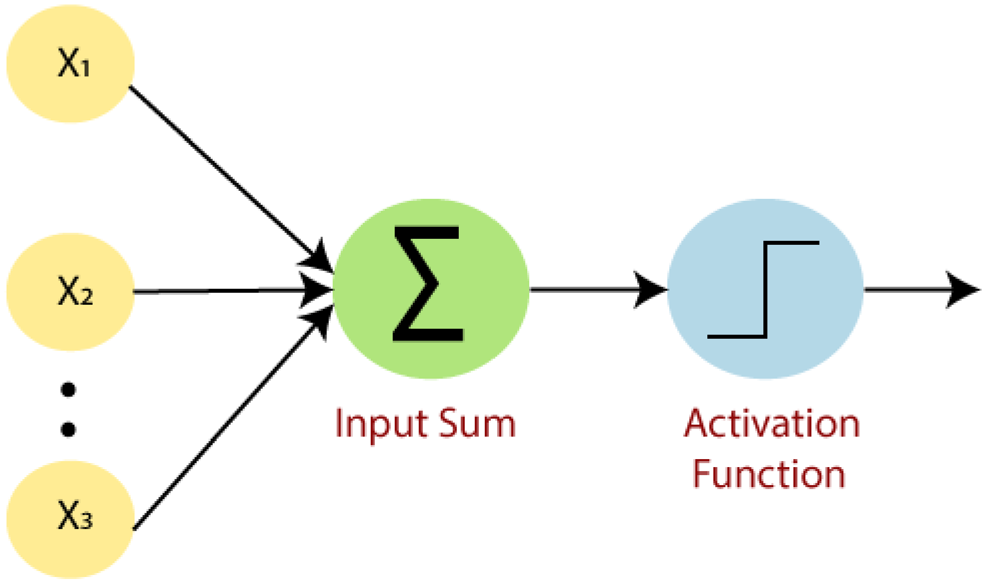
The activation function is used to map the input between the required value like (0, 1) or (-1, 1).

A regular neural network looks like this:

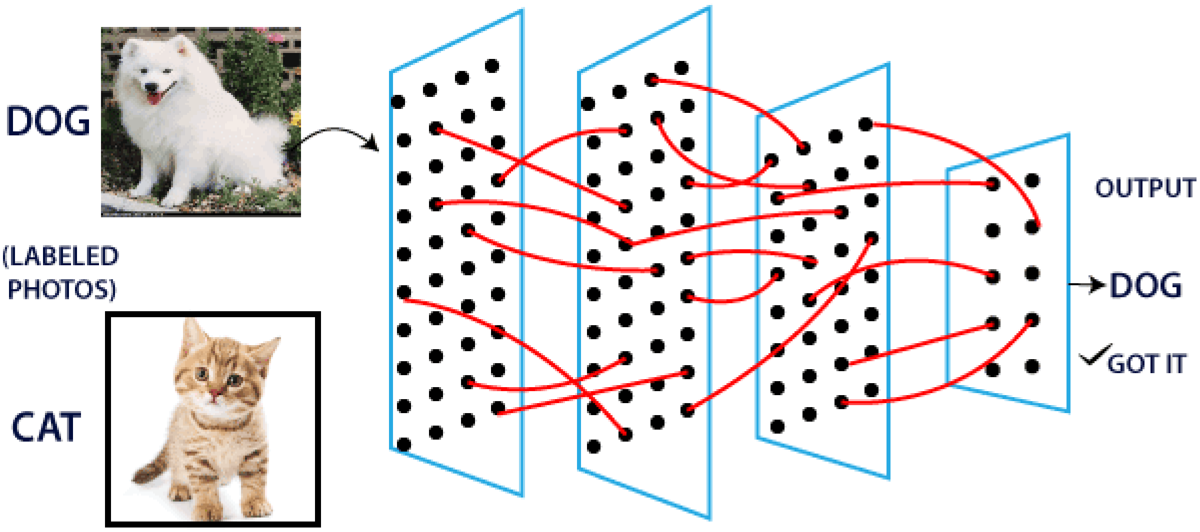


The perceptron consists of 4 parts.

- **Input value or One input layer:** The input layer of the perceptron is made of artificial input neurons and takes the initial data into the system for further processing.
- **Weights and Bias:**
  Weight: It represents the dimension or strength of the connection between units. If the weight to node 1 to node 2 has a higher quantity, then neuron 1 has a more considerable influence on the neuron.
  Bias: It is the same as the intercept added in a linear equation. It is an additional parameter which task is to modify the output along with the weighted sum of the input to the other neuron.
- **Net sum:** It calculates the total sum.
- **Activation Function:** A neuron can be activated or not, is determined by an activation function. The activation function calculates a weighted sum and further adding bias with it to give the result.



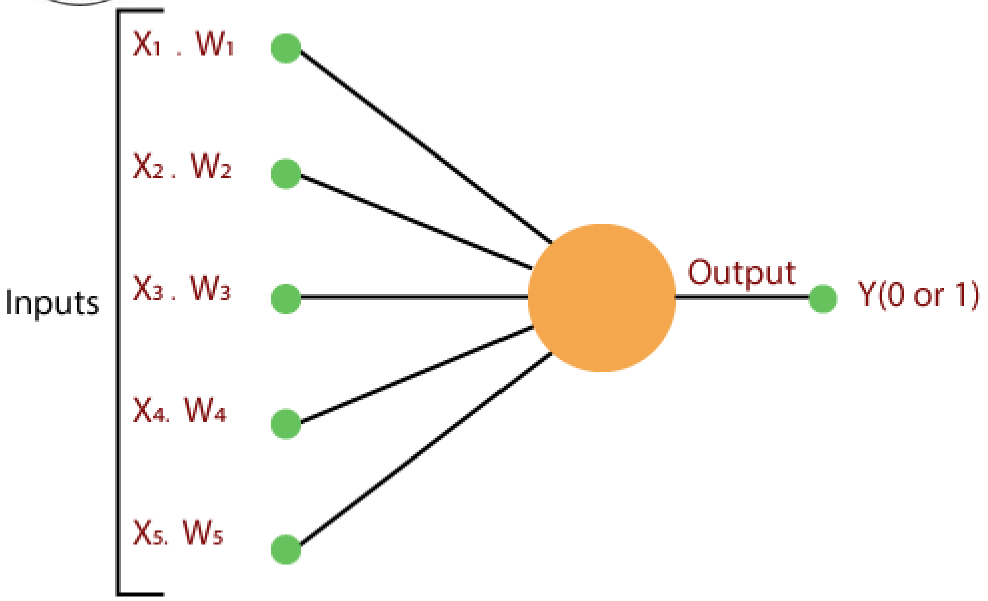
A standard neural network looks like the below diagram.



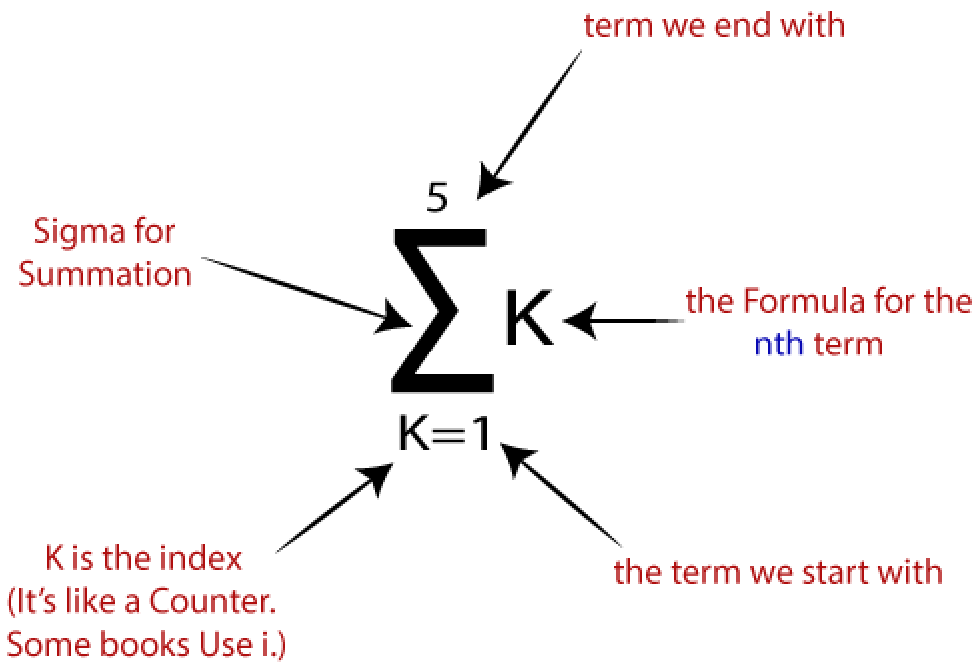
How does it work?

The perceptron works on these simple steps which are given below:

a. In the first step, all the inputs x are multiplied with their weights **w**.



b. In this step, add all the increased values and call them the **Weighted sum**.

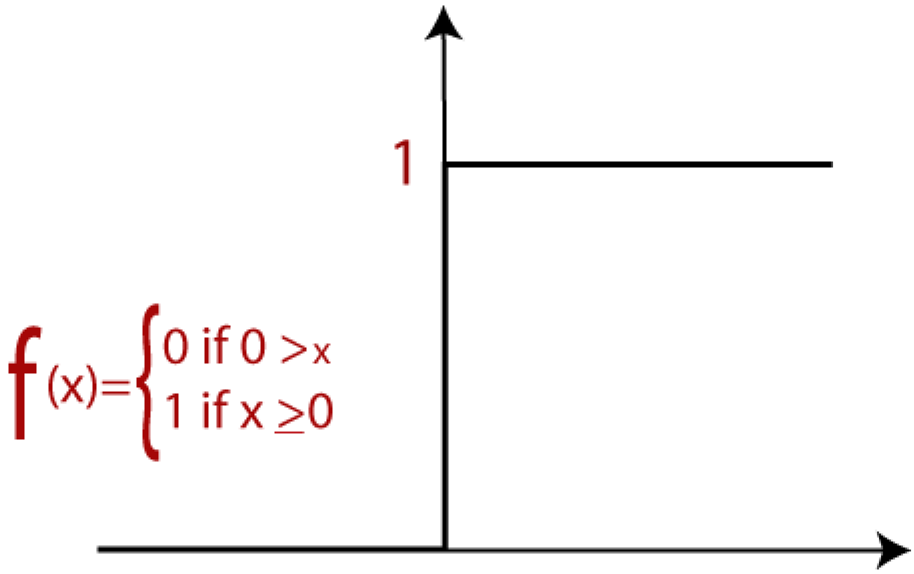


c. In our last step, apply the weighted sum to a correct **Activation Function**.

For Example:

A Unit Step Activation Function

Unit step (threshold)



Implementation:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets

def unit_step_func(x):
    return np.where(x > 0 , 1, 0)

class Perceptron:

    def __init__(self, learning_rate=0.01, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.activation_func = unit_step_func
        self.weights = None
        self.bias = None


    def fit(self, X, y):
```

```
        n_samples, n_features = X.shape

        # init parameters
        self.weights = np.zeros(n_features)
        self.bias = 0

        y_ = np.where(y > 0 , 1, 0)

        # learn weights
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self.activation_func(linear_output)

                # Perceptron update rule
                update = self.lr * (y_[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update


    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        y_predicted = self.activation_func(linear_output)
        return y_predicted



if __name__ == "__main__":

    def accuracy(y_true, y_pred):
        accuracy = np.sum(y_true == y_pred) / len(y_true)
        return accuracy
```

```
X, y = datasets.make_blobs(
    n_samples=150, n_features=2, centers=2, cluster_std=1.05, random_state=2
)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=123
)

p = Perceptron(learning_rate=0.01, n_iters=1000)
p.fit(X_train, y_train)
predictions = p.predict(X_test)

print("Perceptron classification accuracy", accuracy(y_test, predictions))
```

```
Perceptron classification accuracy 1.0
```

Conclusion

In conclusion, the implementation of the Single Layer Perceptron Learning Algorithm demonstrates its ability to create a linear binary classifier, as originally conceptualized by Frank Rosenblatt in 1958. The perceptron, serving as the fundamental unit of neural networks, employs a step activation function to classify input data into two categories. Through iterative learning, the perceptron updates its weights and bias to accurately classify data points, as shown in the provided code. By leveraging the principles of linear classification and activation functions, the perceptron algorithm can effectively learn from training data and make predictions with a satisfactory level of accuracy, showcasing its utility in simple classification tasks.