What is a Perceptron?

A **Perceptron** is the simplest type of **artificial neuron** and the foundation of neural networks. It takes **multiple inputs**, applies **weights**, sums them, and passes the result through an **activation function** to produce an **output (0 or 1)**.

Mathematical Model

$$y = f(w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n)$$

where

- $x_1, x_2, ..., x_n$ → input features
- $w_1, w_2, ..., w_n$ → corresponding weights
- w_0 → bias
- f → activation function (usually step function)

Step Activation Function

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Learning Rule

The perceptron updates its weights to minimize classification errors:

$$w_i = w_i + \eta(y_{true} - y_{pred})x_i$$

where

- η = learning rate (e.g., 0.1)
- y_{true} = actual output
- y_{pred} = predicted output

The algorithm keeps updating weights until predictions are correct or maximum iterations are reached.

🔷 2 Algorithm Steps

1. **Initialize weights** and bias to small random values.
2. **For each training sample:**
   - Compute output using activation function.
   - Update weights if the prediction is incorrect.
3. **Repeat** until convergence (no errors or max epochs reached).

Implementation from Scratch

```
import numpy as np


# Input data (X) and output labels (Y)
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])

Y = np.array([0, 1, 1, 1])  # Output for OR gate

# Initialize weights and bias
weights = np.zeros(X.shape[1])
bias = 0
learning_rate = 0.1
epochs = 10

# Activation function
def step_function(x):
    return 1 if x >= 0 else 0

# Training the perceptron
```

```
for epoch in range(epochs):
    print(f"Epoch {epoch+1}")
    for i in range(len(X)):
        linear_output = np.dot(X[i], weights) + bias
        y_pred = step_function(linear_output)

        # Weight update rule
        error = Y[i] - y_pred
        weights += learning_rate * error * X[i]
        bias += learning_rate * error

        print(f"Input: {X[i]}, Predicted: {y_pred}, Error: {error}")

print("\nFinal weights:", weights)
print("Final bias:", bias)

# Testing the model
for x in X:
    result = step_function(np.dot(x, weights) + bias)
    print(f"Input: {x} => Output: {result}")
```

Output Example

Epoch 1

Input: [0 0], Predicted: 0, Error: 0

Input: [0 1], Predicted: 0, Error: 1

Input: [1 0], Predicted: 0, Error: 1

Input: [1 1], Predicted: 1, Error: 0

Final weights: [0.1 0.1]

Final bias: 0.1

Input: [0 0] => Output: 0

Input: [0 1] => Output: 1

Input: [1 0] => Output: 1

Input: [1 1] => Output: 1

Using Scikit-learn for Comparison

```
from sklearn.linear_model import Perceptron

X = np.array([[0,0],[0,1],[1,0],[1,1]])
Y = np.array([0,1,1,1])

clf = Perceptron(max_iter=1000, eta0=0.1)
clf.fit(X, Y)

print("Weights:", clf.coef_)
print("Bias:", clf.intercept_)
print("Predictions:", clf.predict(X))
```

Real Example:

Titanic Survival

Predict if a passenger survived or not

https://www.kaggle.com/c/titanic/data