**AD4758 NEURO FUZZY LABORATORY**

**Ex 1:3/07/2025 Implementation of Perceptron.**

**AIM:** To implement the Perceptron algorithm for binary classification. The perceptron is a fundamental building block of neural networks and is used to classify data that is linearly separable.

**ALGORITHM**

**Step 1:** Initialize the weights and bias to zero or small random values**.**

**Step 2:** For each epoch:

For each training sample (xᵢ, yᵢ):

* + - Compute the output:  
      y\_pred = activation(w · xᵢ + b)
    - Update weights and bias if there's a misclassification:
      * w = w + α \* (yᵢ - y\_pred) \* xᵢ
      * b = b + α \* (yᵢ - y\_pred)

**Step 3:** Repeat until convergence or for a fixed number of epochs.

**IMMPLEMENTATION:**

import numpy as np

class Perceptron:

def \_\_init\_\_(self, learning\_rate=0.01, epochs=1000):

self.lr = learning\_rate

self.epochs = epochs

self.weights = None

self.bias = None

def activation(self, x):

return np.where(x >= 0, 1, 0)

def fit(self, X, y):

n\_samples, n\_features = X.shape

# Initialize weights and bias

self.weights = np.zeros(n\_features)

self.bias = 0

# Training loop

for \_ in range(self.epochs):

for idx, x\_i in enumerate(X):

linear\_output = np.dot(x\_i, self.weights) + self.bias

y\_pred = self.activation(linear\_output)

# Update rule

update = self.lr \* (y[idx] - y\_pred)

self.weights += update \* x\_i

self.bias += update

def predict(self, X):

linear\_output = np.dot(X, self.weights) + self.bias

return self.activation(linear\_output)

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

# Sample training data (AND logic gate)

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

y = np.array([0, 0, 0, 1]) # Output of AND gate

perceptron = Perceptron(learning\_rate=0.1, epochs=10)

perceptron.fit(X, y)

predictions = perceptron.predict(X)

print("Predictions:", predictions)

**Output:**

**Predictions: [0 0 0 1]**

**Ex2: Implementation of Perceptron Rule**

**AIM:**

To implement the Perceptron Learning Rule for training a single-layer neural network to perform binary classification on linearly separable data.

**ALGORITHM**

**Step 1: Initialize weights w and bias b to 0 or small random values.**

**Step 2: For each epoch (repeat until convergence or max iterations):**

**Step 3: For each training example (xᵢ, yᵢ):**

**Compute weighted sum:  
z = w · xᵢ + b**

**Apply step function:  
ŷ = 1 if z ≥ 0 else 0**

**Update weights and bias:**

* + - **w = w + α \* (yᵢ - ŷ) \* xᵢ**
    - **b = b + α \* (yᵢ - ŷ)**

**Step 4: Repeat until the model converges (no updates needed) or maximum epochs reached.**

**IMPLEMENTATION:**

import numpy as np

class PerceptronRule:

def \_\_init\_\_(self, learning\_rate=0.1, epochs=1000):

self.lr = learning\_rate

self.epochs = epochs

self.weights = None

self.bias = None

def activation(self, x):

return 1 if x >= 0 else 0

def fit(self, X, y):

n\_samples, n\_features = X.shape

self.weights = np.zeros(n\_features)

self.bias = 0

for epoch in range(self.epochs):

for idx, x\_i in enumerate(X):

z = np.dot(x\_i, self.weights) + self.bias

y\_pred = self.activation(z)

# Perceptron Rule Update

error = y[idx] - y\_pred

self.weights += self.lr \* error \* x\_i

self.bias += self.lr \* error

def predict(self, X):

linear\_output = np.dot(X, self.weights) + self.bias

return np.where(linear\_output >= 0, 1, 0)

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

# AND gate dataset

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

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

model = PerceptronRule(learning\_rate=0.1, epochs=10)

model.fit(X, y)

predictions = model.predict(X)

print("Predictions:", predictions)

**OUTPUT**

**Predictions: [0 0 0 1]**