Perceptron Learning Algorithm - Theory and Implementation

# 1. Introduction

The Perceptron is one of the simplest types of artificial neural networks used for binary classification. It was introduced by Frank Rosenblatt in 1958. A perceptron takes several binary inputs, applies a set of weights to them, computes a weighted sum, and passes the result through an activation function (typically a step function) to produce a binary output (0 or 1).

It can be used to solve linearly separable problems like AND, OR, NOT logic gates.

# 2. Perceptron Model

Mathematically, the perceptron can be represented as:

y = f(w · x + b)

Where:

- x: input vector

- w: weight vector

- b: bias (or threshold)

- f: activation function (usually a step function)

In our code, we define the step function manually using a condition:

If the weighted sum > threshold → output = 1, otherwise 0.

# 3. Code Implementation

import numpy as np

features=np.array([

[0,0],

[0,1],

[1,0],

[1,1]

])

labels = np.array([0,0,0,1]) # Represents the AND logic

w=[0.9,0.9] # Initial weights

threshold=0.5

learning\_rate=0.1

epoch = 20 # Number of learning cycles

for j in range(0,epoch):

print("epoch",j)

for i in range(0,features.shape[0]):

instance=features[i]

x0=instance[0]

x1=instance[1]

# Compute weighted sum

sum\_unit=x0\*w[0]+x1\*w[1]

# Activation (step function)

if sum\_unit>threshold:

fire=1

else:

fire=0

# Error (actual - predicted)

delta=labels[i]-fire

print("prediction:", fire,"whereas actual was",labels[i],"(error:",delta,")")

# Weight update rule

w[0]=w[0]+delta\*learning\_rate

w[1]=w[1]+delta\*learning\_rate

print("----------------------------------")

# 4. Code Description

Dataset (features & labels):

We use the AND gate dataset:

- Inputs: [(0,0), (0,1), (1,0), (1,1)]

- Output: [0, 0, 0, 1]

Initial Weights:

- Both weights are set to 0.9

- Threshold = 0.5

- Learning Rate = 0.1

Training Loop:

- The network trains over 20 epochs (iterations).

- For each input:

- Computes the weighted sum

- Applies the activation function

- Calculates the error = actual - prediction

- Updates weights using the Perceptron Learning Rule:

w\_new = w\_old + learning rate × error

# 5. Output Explanation

First Few Epochs:

epoch 0

prediction: 0 whereas actual was 0 (error: 0 )

prediction: 1 whereas actual was 0 (error: -1 )

prediction: 1 whereas actual was 0 (error: -1 )

prediction: 1 whereas actual was 1 (error: 0 )

Here, for inputs [0,1] and [1,0], the output was 1 instead of 0, so the weights are reduced. The model begins learning.

Epoch 3 Onwards:

epoch 3

prediction: 0 whereas actual was 0 (error: 0 )

prediction: 0 whereas actual was 0 (error: 0 )

prediction: 0 whereas actual was 0 (error: 0 )

prediction: 1 whereas actual was 1 (error: 0 )

Now, all predictions are correct, and no weight updates happen. The model has successfully learned the AND gate.

Remaining Epochs:

The output stabilizes and no changes occur, confirming the model has converged.

# 6. Final Notes

- The Perceptron can only solve linearly separable problems. It won't work for XOR.

- This model successfully learns the AND logic gate.

- After convergence, the weights stop changing.

- Number of epochs required can vary based on initial weights, learning rate, and data.

# 7. Conclusion

This implementation demonstrates how a basic Perceptron works using manual weight updates and a step activation. It's a foundational concept in neural networks and machine learning, and understanding it helps in grasping more advanced architectures later.