Harshil Andhariya

[Email address]

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document.   
When you’re ready to add your content, just click here and start typing.]

[Document title]

[Document subtitle]

W = [w0, w1, w2]

X = [x0, x1, x2] where x0 = 1

Pred = 1, actual = 0

w0x0+w1x1+w2x2>=0, w0x0+w1x1+w2x2<0 decrease

w = w – learningrate \* x

Pred = 0, actual = 1, increase

W = w + x

Rosenblatt Model explained by pruthwik sir in lab

W = [w0, w1, w2]

X = [x0, x1, x2] where x0 = 1

Pred = 1, actual = 0

w0x0+w1x1+w2x2>=0, w0x0+w1x1+w2x2<0 decrease

w = w – learningrate \* x

Pred = 0, actual = 1, increase

W = w + x

Q1

Find out the weights through a program for the following logic gates using

both the fixed weights (MP model) and the update rule (Rosenblat’s model)

as mentioned in the slide: x∧y, x∧~y, ~x∧y, ~x∧~y, x∨y, x∨~y, ~x∨y,

~x∨~y where ∧ denotes AND, ∨ denotes OR, and ~ denotes NOT.

import numpy as np

# Define logic gates inputs and outputs

logic\_gates = {

    "x∧y": {

        "inputs": np.array([[0, 0], [0, 1], [1, 0], [1, 1]]),

        "outputs": np.array([0, 0, 0, 1])

    },

    "x∧~y": {

        "inputs": np.array([[0, 0], [0, 1], [1, 0], [1, 1]]),

        "outputs": np.array([0, 1, 0, 0])

    },

    "~x∧y": {

        "inputs": np.array([[0, 0], [0, 1], [1, 0], [1, 1]]),

        "outputs": np.array([0, 0, 1, 0])

    },

    "~x∧~y": {

        "inputs": np.array([[0, 0], [0, 1], [1, 0], [1, 1]]),

        "outputs": np.array([1, 0, 0, 0])

    },

    "x∨y": {

        "inputs": np.array([[0, 0], [0, 1], [1, 0], [1, 1]]),

        "outputs": np.array([0, 1, 1, 1])

    },

    "x∨~y": {

        "inputs": np.array([[0, 0], [0, 1], [1, 0], [1, 1]]),

        "outputs": np.array([1, 1, 1, 0])

    },

    "~x∨y": {

        "inputs": np.array([[0, 0], [0, 1], [1, 0], [1, 1]]),

        "outputs": np.array([1, 0, 1, 1])

    },

    "~x∨~y": {

        "inputs": np.array([[0, 0], [0, 1], [1, 0], [1, 1]]),

        "outputs": np.array([1, 1, 1, 0])

    }

}

# Function to implement the MP model with fixed weights

def mp\_model(inputs, weights, threshold):

    results = []

    for x in inputs:

        activation = np.dot(weights, x)

        results.append(1 if activation >= threshold else 0)

    return results

# Function to train Rosenblatt’s Perceptron Model (Update Rule)

def perceptron\_train(inputs, outputs, learning\_rate=0.1, epochs=20):

    weights = np.zeros(inputs.shape[1] + 1)  # Include bias weight

    for \_ in range(epochs):

        for i in range(len(inputs)):

            x = np.insert(inputs[i], 0, 1)  # Add bias as x0 = 1

            y = np.dot(weights, x) >= 0  # Step function

            error = outputs[i] - y

            weights += learning\_rate \* error \* x  # Update rule

    return weights

# Predefined fixed weights for the MP model

mp\_weights = {

    "x∧y": ([1, 1], 2),

    "x∧~y": ([1, -1], 1),

    "~x∧y": ([-1, 1], 1),

    "~x∧~y": ([-1, -1], -1),

    "x∨y": ([1, 1], 1),

    "x∨~y": ([1, -1], 0),

    "~x∨y": ([-1, 1], 0),

    "~x∨~y": ([-1, -1], 0)

}

# Displaying the results

print("Fixed Weights (MP Model):")

for gate, (weights, threshold) in mp\_weights.items():

    inputs = logic\_gates[gate]["inputs"]

    outputs = mp\_model(inputs, weights, threshold)

    print(f"{gate} => Weights: {weights}, Threshold: {threshold}, Output: {outputs}")

print("\nTrained Weights (Rosenblatt’s Model):")

for gate, data in logic\_gates.items():

    trained\_weights = perceptron\_train(data["inputs"], data["outputs"])

    print(f"Weights for {gate}: {trained\_weights}")

Output:

Fixed Weights (MP Model):

x∧y => Weights: [1, 1], Threshold: 2, Output: [0, 0, 0, 1]

x∧~y => Weights: [1, -1], Threshold: 1, Output: [0, 0, 1, 0]

~x∧y => Weights: [-1, 1], Threshold: 1, Output: [0, 1, 0, 0]

~x∧~y => Weights: [-1, -1], Threshold: -1, Output: [1, 1, 1, 0]

x∨y => Weights: [1, 1], Threshold: 1, Output: [0, 1, 1, 1]

x∨~y => Weights: [1, -1], Threshold: 0, Output: [1, 0, 1, 1]

~x∨y => Weights: [-1, 1], Threshold: 0, Output: [1, 1, 0, 1]

~x∨~y => Weights: [-1, -1], Threshold: 0, Output: [1, 0, 0, 0]

Trained Weights (Rosenblatt’s Model):

Weights for x∧y: [-0.2 0.2 0.1]

Weights for x∧~y: [-0.1 -0.1 0.1]

Weights for ~x∧y: [-0.1 0.1 -0.2]

Weights for ~x∧~y: [ 0. -0.1 -0.1]

Weights for x∨y: [-0.1 0.1 0.1]

Weights for x∨~y: [ 0.2 -0.2 -0.1]

Weights for ~x∨y: [ 0. 0.1 -0.1]

Weights for ~x∨~y: [ 0.2 -0.2 -0.1]

Q2

If you increase the number of inputs, say 3 (x, y, z, ~x, ~y, ~z) or 4 (x, y, z,

w, ~x, ~y, ~z, ~w), what changes do you observe in both the techniques?

Show this in a program.

import numpy as np

# Define logic gates inputs and outputs for 3-variable AND gate: x ∧ y ∧ z

logic\_gates\_extended = {

"x∧y∧z": {

"inputs": np.array([

[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1],

[1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]

]),

"outputs": np.array([0, 0, 0, 0, 0, 0, 0, 1])

},

"x∨y∨z": {

"inputs": np.array([

[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1],

[1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]

]),

"outputs": np.array([0, 1, 1, 1, 1, 1, 1, 1])

}

}

# Function to implement the MP model with fixed weights

def mp\_model(inputs, weights, threshold):

results = []

for x in inputs:

activation = np.dot(weights, x)

results.append(1 if activation >= threshold else 0)

return results

# Function to train Rosenblatt’s Perceptron Model (Update Rule)

def perceptron\_train(inputs, outputs, learning\_rate=0.1, epochs=50):

weights = np.zeros(inputs.shape[1] + 1) # Include bias weight

for \_ in range(epochs):

for i in range(len(inputs)):

x = np.insert(inputs[i], 0, 1) # Add bias as x0 = 1

y = np.dot(weights, x) >= 0 # Step function

error = outputs[i] - y

weights += learning\_rate \* error \* x # Update rule

return weights

# Predefined fixed weights for 3-input logic gates (MP Model)

mp\_weights\_extended = {

"x∧y∧z": ([1, 1, 1], 3), # Weights: [1,1,1] Threshold: 3 for AND

"x∨y∨z": ([1, 1, 1], 1) # Weights: [1,1,1] Threshold: 1 for OR

}

# Displaying the results for 3-input gates

print("Fixed Weights (MP Model) for 3-input gates:")

for gate, (weights, threshold) in mp\_weights\_extended.items():

inputs = logic\_gates\_extended[gate]["inputs"]

outputs = mp\_model(inputs, weights, threshold)

print(f"{gate} => Weights: {weights}, Threshold: {threshold}, Output: {outputs}")

print("\nTrained Weights (Rosenblatt’s Model) for 3-input gates:")

for gate, data in logic\_gates\_extended.items():

trained\_weights = perceptron\_train(data["inputs"], data["outputs"])

print(f"Weights for {gate}: {trained\_weights}")

Output:

Fixed Weights (MP Model) for 3-input gates:

x∧y∧z => Weights: [1, 1, 1], Threshold: 3, Output: [0, 0, 0, 0, 0, 0, 0, 1]

x∨y∨z => Weights: [1, 1, 1], Threshold: 1, Output: [0, 1, 1, 1, 1, 1, 1, 1]

Trained Weights (Rosenblatt’s Model) for 3-input gates:

Weights for x∧y∧z: [-0.3 0.4 0.4 0.4]

Weights for x∨y∨z: [-0.1 0.5 0.5 0.5]

Q3.

