U23AI021

Lab assignment 02
Artificial Intelligence[AI-202]

Harshil Andhariya
u23ai021@coed.svnit.ac.in

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

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])
    }
}

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


def perceptron_train(inputs, outputs, learning_rate=0.1,
epochs=20):
    weights = np.zeros(inputs.shape[1] + 1)
    for _ in range(epochs):
        for i in range(len(inputs)):
            x = np.insert(inputs[i], 0, 1)
            y = np.dot(weights, x) >= 0
            error = outputs[i] - y
            weights += learning_rate * error * x
    return weights


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)
}

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: (in Google colab)

```
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]
```

✓ 0s completed at 8:48 AM

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

logic_gates = {
    "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])
    }
}

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

def perceptron_train(inputs, outputs, learning_rate=0.1,
epochs=20):
    weights = np.zeros(inputs.shape[1] + 1)
    for _ in range(epochs):
        for i in range(len(inputs)):
            x = np.insert(inputs[i], 0, 1)
            y = np.dot(weights, x) >= 0
            error = outputs[i] - y
            weights += learning_rate * error * x
    return weights

mp_weights = {
    "x∧y∧z": ([1, 1, 1], 3),
    "x∨y∨z": ([1, 1, 1], 1)
}

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∧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):
Weights for x∧y∧z: [-0.2  0.1  0.1  0.1]
Weights for x∨y∨z: [-0.1  0.1  0.1  0.1]
```

Q.3

Does the choice of initial weights impact the techniques? How fast does a technique find the relevant weights? Show this through a program.

```
import numpy as np

logic_gates = {
    "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])
    }
}
```

```python
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

def perceptron_train(inputs, outputs, learning_rate=0.1, 
epochs=20, initial_weights=None):
    weights = initial_weights if initial_weights is not None 
else np.zeros(inputs.shape[1] + 1)
    for _ in range(epochs):
        for i in range(len(inputs)):
            x = np.insert(inputs[i], 0, 1)
            y = np.dot(weights, x) >= 0
            error = outputs[i] - y
            weights += learning_rate * error * x
    return weights

mp_weights = {
    "x∧y∧z": ([1, 1, 1], 3),
    "x∨y∨z": ([1, 1, 1], 1)
}

initial_weights_list = [
    np.array([0.5, 0.5, 0.5, 0.5]),
    np.array([-0.5, -0.5, -0.5, -0.5]),
    np.array([0, 0, 0, 0])
]

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 with different
initial weights):")
for gate, data in logic_gates.items():
    for idx, init_weights in
enumerate(initial_weights_list):
        trained_weights = perceptron_train(data["inputs"],
data["outputs"], initial_weights=init_weights)
        print(f"Initial Weights {idx+1} for {gate}:
{init_weights}, Trained Weights: {trained_weights}")
```

Output:

```
Fixed Weights (MP Model):
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 with different initial weights):
Initial Weights 1 for x∧y∧z: [-0.6  0.3  0.2  0.1], Trained Weights: [-0.6  0.3  0.2  0.1]
Initial Weights 2 for x∧y∧z: [-0.3  0.2  0.1  0.1], Trained Weights: [-0.3  0.2  0.1  0.1]
```

Q4.

Can you find the weights for x XOR y? If yes, find the weights through the

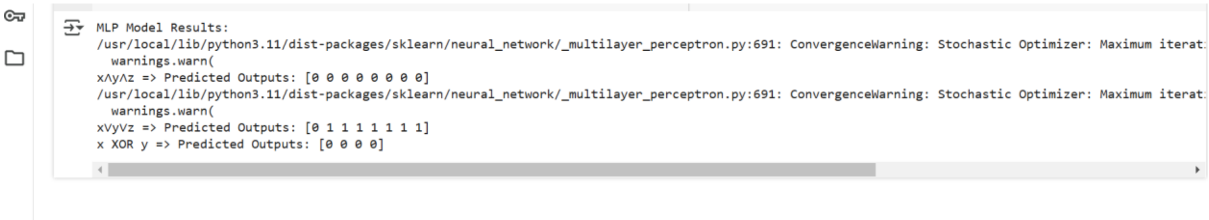
same program proposed in Question 1. If no, can you modify anything in

your program to get to the answer?

```
import numpy as np
from sklearn.neural_network import MLPClassifier

logic_gates = {
    "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])
    },
    "xVyVz": {
        "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])
    },
    "x XOR y": {
        "inputs": np.array([
            [0, 0], [0, 1], [1, 0], [1, 1]
        ]),
        "outputs": np.array([0, 1, 1, 0])
    }
}

def train_mlp(inputs, outputs):
    mlp = MLPClassifier(hidden_layer_sizes=(4,),
activation='relu', max_iter=1000, solver='adam',
random_state=42)
    mlp.fit(inputs, outputs)
    return mlp

print("MLP Model Results:")
for gate, data in logic_gates.items():
    mlp_model = train_mlp(data["inputs"], data["outputs"])
    predictions = mlp_model.predict(data["inputs"])
    print(f"{gate} => Predicted Outputs: {predictions}")
```

Output (In google COLAB) ss



Output In the form of text:

```
x∧y∧z => Predicted Outputs: [0 0 0 0 0 0 0 0]
/usr/local/lib/python3.11/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and the optimization
hasn't converged yet.
  warnings.warn(
x∨y∨z => Predicted Outputs: [0 1 1 1 1 1 1 1]
x XOR y => Predicted Outputs: [0 0 0 0]
```