PRACTICAL 3

A. Aim: Write a program to implement Hebb's rule

```
import numpy as np
X = np.array([[1, 1, 1], [1, -1, -1], [-1, -1, 1]])
w = np.zeros(X.shape[1])
for pattern in X:
    w += pattern
for pattern in X:
    activation = np.dot(w, pattern)
    print(f"Input Pattern: {pattern}, Synaptic Weights: {w}, Activation: {activation:.2f}")
```

Output:

```
Input Pattern: [1 1 1], Synaptic Weights: [ 1. -1.  1.], Activation: 1.00
Input Pattern: [ 1 -1 -1], Synaptic Weights: [ 1. -1.  1.], Activation: 1.00
Input Pattern: [-1 -1  1], Synaptic Weights: [ 1. -1.  1.], Activation: 1.00
```

PRACTICAL 3

B. Aim: Write a program to implement Delta rule.

```
import numpy as np
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
targets = np.array([0, 0, 0, 1])
learning_rate = 0.1
weights = np.random.rand(2)
bias = np.random.rand()
max_epochs = 10000
epoch = 0
while epoch < max_epochs:
    error_count = 0
    for i in range(len(X)):
        weighted_sum = np.dot(X[i], weights) + bias
        prediction = 1 if weighted_sum >= 0 else 0
    error = targets[i] - prediction
    if error != 0:
        error_count += 1
    weights += learning_rate * error * X[i]
    bias += learning_rate * error
    print(f"Epoch {epoch + 1}: {error_count} errors")
    if error_count == 0:
        print("Converged. Weights and bias:")
        print("Weights:", weights)
        print("Bias:", bias)
        break
    epoch += 1
if epoch == max_epochs:
    print("Did not converge. Weights and bias:")
    print("Weights:", weights)
    print("Bias:", bias)
```

Output:

```
Epoch 1: 0 errors
Converged. Weights and bias:
Weights: [0.91643671 0.00337188]
Bias: 0.15193899052526594
```