Batch: C2 **Roll No.: 16010122323**

Experiment No. 05

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of OR function with bipolar inputs and targets using Adaline network. Assume the required parameters for training of the network.

Objective: To learn Adaline network.

Expected Outcome of Experiment:

CO2: To understand the features of neural networks and different learning methods.

Books/ Journals/ Websites referred:

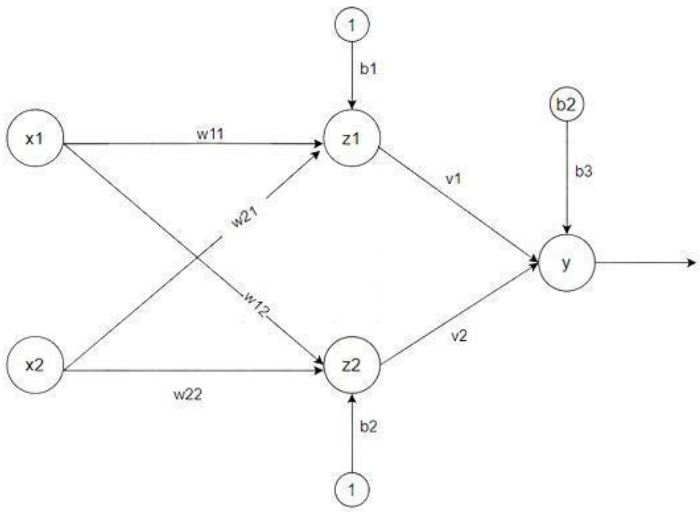
Pre Lab/ Prior Concepts:

Adaptive Linear Neuron (Adaline):
Adaline which stands for Adaptive Linear Neuron, is a network having a single linear unit.
It was developed by Widrow and Hoff in 1960.
Some important points about Adaline are as follows −

- It uses bipolar activation function.
- It tries to minimize the Mean-Squared Error (MSE) between the actual output and the desired/target output.

The weights and the bias are adjustable

Architecture:



Algorithm:

Step 1: Initialize the following to start the training – Weights, Bias, Learning rate α

Step 2: While the stopping condition is False do steps 3 to 7.

Step 3: for each training set perform steps 4 to 6.

Step 4: Set activation of input unit xi = si for (i=1 to n)

Step 5: compute net input to output unit $y_{in} = \sum w_i x_i + b$

Here, b is the bias and n is the total number of neurons.

Step 6: Update the weights and bias for i=1 to n

$$w_i(new) = w_i(old) + \alpha(t - y_{in})x_i$$

$$b(new) = b(old) + (t - y_{in})$$

and calculate $error : (t - y_{in})^2$

Step 7: Test the stopping condition. The stopping condition may be when the weight changes at a low rate or no change.

Implementation Details:

```
import numpy as np

np.random.seed(42)  # For reproducibility
learning_rate = 0.1
epochs = 100
n_inputs = 2

X = np.array([
    [1,  1, 1],  # (1, 1) -> OR -> 1
    [1, -1, 1],  # (1, 0) -> OR -> 1
    [-1, 1, 1],  # (0, 1) -> OR -> 1
    [-1, -1, 1]  # (0, 0) -> OR -> -1
])
# Bipolar target output for the OR function
T = np.array([1, 1, 1, -1])  # Target outputs corresponding to the inputs

# Initialize weights (including bias)
weights = np.random.randn(n_inputs + 1)  # +1 for bias

# Adaline training loop
for epoch in range(epochs):
    total_error = 0  # Sum of squared errors
    for i in range(len(X)):
        # Compute net input
        y_in = np.dot(X[i], weights)

        error = T[i] - y_in

        weights += learning_rate * error * X[i]

        total_error += error**2

    # Check stopping condition (here, we check if the error is very low)
```

```
    if total_error < 0.01:
        print(f"Training stopped after {epoch+1} epochs.")
        break

print("Final weights:", weights)

for i in range(len(X)):
    y_in = np.dot(X[i], weights)
    output = np.sign(y_in)  # Sign function for bipolar output (classification)
    print(f"Input: {X[i][:2]} -> Predicted Output: {output}, Target: {T[i]}")
```

OUTPUT:

```
Run

Final weights: [0.52941176 0.55882353 0.5        ]
Input: [1 1] -> Predicted Output: 1.0, Target: 1
Input: [ 1 -1] -> Predicted Output: 1.0, Target: 1
Input: [-1  1] -> Predicted Output: 1.0, Target: 1
Input: [-1 -1] -> Predicted Output: -1.0, Target: -1
```

Implementation of OR function with bipolar inputs and targets using Adaline network.

x1	x2	t
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

Conclusion: *Learnt and Implemented OR function with bipolar inputs and targets using Adaline network*

Post Lab Descriptive Questions:

Use Adaline network to train AND NOT function with bipolar inputs and targets. Perform 1 epoch of training

```
import numpy as np

def initialize_parameters(n_inputs, seed=42):
    np.random.seed(seed)
    weights = np.random.randn(n_inputs + 1)
    return weights

def net_input(X, weights):
    return np.dot(X, weights)

def train_adaline(X, T, weights, learning_rate=0.1):
    total_error = 0
    for i in range(len(X)):
        y_in = net_input(X[i], weights)
        error = T[i] - y_in
        weights += learning_rate * error * X[i]
        total_error += error**2
    return weights, total_error

def main():
    n_inputs = 2
    X = np.array([
        [1,  1, 1],
        [1, -1, 1],
        [-1, 1, 1],
        [-1, -1, 1]
    ])
    T = np.array([-1, 1, -1, -1])
    weights = initialize_parameters(n_inputs)
    weights, total_error = train_adaline(X, T, weights)
    print("Weights after 1 epoch:", weights)
```

```
    print("Total error after 1 epoch:", total_error)

    for i in range(len(X)):
        y_in = net_input(X[i], weights)
        output = np.sign(y_in)
        print(f"Input: {X[i][:2]} -> Predicted Output: {output}, Target:
{T[i]}")

if __name__ == "__main__":
    main()
```

OUTPUT:

```
Run

Weights after 1 epoch: [ 0.50988686 -0.27274128  0.21687752]
Total error after 1 epoch: 6.663146506881507
Input: [1 1] -> Predicted Output: 1.0, Target: -1
Input: [ 1 -1] -> Predicted Output: 1.0, Target: 1
Input: [-1  1] -> Predicted Output: -1.0, Target: -1
Input: [-1 -1] -> Predicted Output: -1.0, Target: -1
```

Date: ____________ **Signature of faculty in-charge**