Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)
Department of Electronics and Telecommunication

Priciples of Soft Computing

Name: Narayan Gawas **UID: 2021201070**

EXPERIMENT-1

Aim: Implement the single layer perceptron network. Assume all initial weights to be zero. Assume learning rate equal to 1. Use bipolar inputs and outputs. Use binary threshold function such that

y=f(x) =1 if f(x)>0,
=0 if f(x)=0,
=-1 if f(x)<0.

The function: Y=x1.x2 + ~x1.x2.

There should be no weight change (accordingly choose the number of iterations). Show the diagram.

Code:

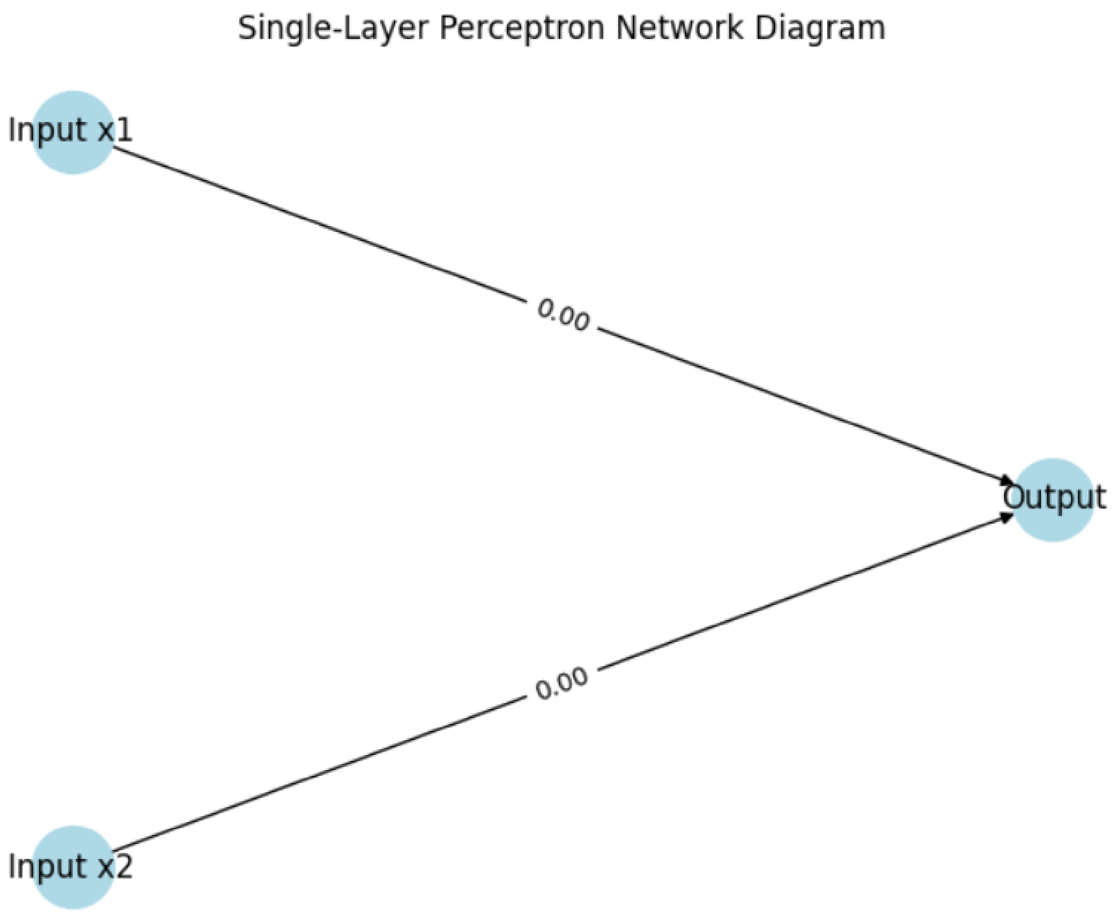
```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
# Define the inputs (bipolar)
x1 = np.array([-1, -1, 1, 1])
x2 = np.array([-1, 1, -1, 1])

# Define the target values for Y
target = np.array([-1, 1, -1, 1])

# Initialize weights to zero
w1 = 0
w2 = 0

# Learning rate
learning_rate = 1

# Maximum number of iterations (chosen to ensure convergence)
max_iterations = 10

# Lists to store weights and errors for plotting
w1_values = [w1]
w2_values = [w2]
errors = []
```

```
# Create a directed graph for the perceptron network
G = nx.DiGraph()
G.add_node("Input x1", pos=(0, 1))
G.add_node("Input x2", pos=(0, -1))
G.add_node("Output", pos=(2, 0))
G.add_edge("Input x1", "Output", weight=w1)
G.add_edge("Input x2", "Output", weight=w2)

# Training the perceptron
# for iteration in range(max_iterations):
while(1):
    iteration = 0
    total_error = 0
    print(f"Iteration {iteration + 1}:")
    print(f"Weight 1 (w1) = {w1}, Weight 2 (w2) = {w2}")

    for i in range(len(x1)):
        # Calculate the weighted sum
        weighted_sum = w1 * x1[i] + w2 * x2[i]

        # Apply the binary threshold function
        if weighted_sum > 0:
            output = 1
        elif weighted_sum == 0:
            output = 0
        else:
            output = -1

        # Calculate the error
        error = target[i] - output
        total_error += abs(error)

        # Calculate weight changes
        delta_w1 = learning_rate * error * x1[i]
        delta_w2 = learning_rate * error * x2[i]

         print(f"  Input: x1 = {x1[i]}, x2 = {x2[i]}, Weighted Sum = {weighted_sum}, Output = 
{output}, Target = {target[i]}, Error = {error}")
        print(f"  Delta w1 = {delta_w1}, Delta w2 = {delta_w2}")

        # Update the weights
        w1 += delta_w1
        w2 += delta_w2

    # Store weights and errors for plotting
    w1_values.append(w1)
    w2_values.append(w2)
    errors.append(total_error)
    print(f"Total Error = {total_error}\n")
    if not total_error:
```

```
        break

# Plot the network diagram
pos = nx.get_node_attributes(G, 'pos')
edge_labels = {(i, j): f"{G[i][j]['weight']:.2f}" for i, j in G.edges()}
nx.draw(G, pos, with_labels=True, node_size=1000, node_color='lightblue')
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
plt.title("Single-Layer Perceptron Network Diagram")
plt.show()

# Final weights and threshold
print("Final weights: w1 =", w1, ", w2 =", w2)
```

Output:

```
Iteration 1:
Weight 1 (w1) = 0, Weight 2 (w2) = 0
  Input: x1 = -1, x2 = -1, Weighted Sum = 0, Output = 0, Target = -1, Error = -1
  Delta w1 = 1, Delta w2 = 1
  Input: x1 = -1, x2 = 1, Weighted Sum = 0, Output = 0, Target = 1, Error = 1
  Delta w1 = -1, Delta w2 = 1
  Input: x1 = 1, x2 = -1, Weighted Sum = -2, Output = -1, Target = -1, Error = 0
  Delta w1 = 0, Delta w2 = 0
  Input: x1 = 1, x2 = 1, Weighted Sum = 2, Output = 1, Target = 1, Error = 0
  Delta w1 = 0, Delta w2 = 0
Total Error = 2

Iteration 1:
Weight 1 (w1) = 0, Weight 2 (w2) = 2
  Input: x1 = -1, x2 = -1, Weighted Sum = -2, Output = -1, Target = -1, Error = 0
  Delta w1 = 0, Delta w2 = 0
  Input: x1 = -1, x2 = 1, Weighted Sum = 2, Output = 1, Target = 1, Error = 0
  Delta w1 = 0, Delta w2 = 0
  Input: x1 = 1, x2 = -1, Weighted Sum = -2, Output = -1, Target = -1, Error = 0
  Delta w1 = 0, Delta w2 = 0
  Input: x1 = 1, x2 = 1, Weighted Sum = 2, Output = 1, Target = 1, Error = 0
  Delta w1 = 0, Delta w2 = 0
Total Error = 0
```




```
Final weights: w1 = 0 , w2 = 2
```

Conclusion:

The single-layer perceptron network is a simple neural network architecture suitable for linearly separable binary classification tasks. However, it has limitations and cannot solve problems that are not linearly separable. For more complex tasks, multi-layer perceptrons (MLPs) or other advanced neural network architectures are typically used. Nevertheless, understanding and implementing the single-layer perceptron is a fundamental step towards grasping the basics of neural networks and their learning mechanisms.