St. Francis Institute of Technology, Mumbai-400 103
Department Of Information Technology

A.Y. 2024-2025
Class: BE-ITA/B, Semester: VII
Subject: Data Science Lab

Experiment – 3

1. **Aim:** To implement perceptron learning rule..

2. **Objectives:** Students should be familiarize with Learning Architectures and Frameworks
3. **Prerequisite:** Python basics

4. **Pre-Experiment Exercise:**
   Theory:
   The term "Artificial Neural Network" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.

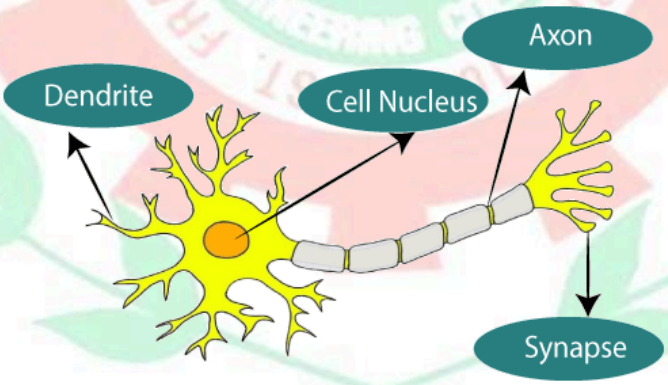

Fig:5.1 Biological Neural Network

Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.

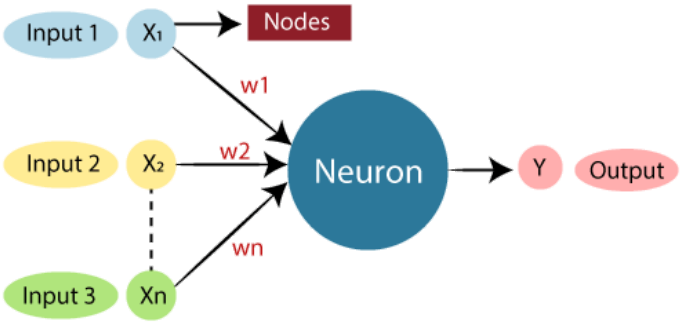

Fig 5.2: Artificial Neural Network

Recurrent neural network:

Recurrent Neural Network (RNN) are a type of Neural Network where the output from previous step are fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is Hidden state, which remembers some information about a sequence. RNN have a "memory" which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

6. **Laboratory Exercise**

A. Procedure

i. Use google colab for programming.
ii. Import neural network packages.
iii. Design a network for AND operation.
iv. Demonstrate the working of Network where input is given as 0 and 1.

Post-Experiments Exercise:

A. Extended Theory:

a. Design an application using neural network for OR operation

B. Conclusion:

1. Write what was performed in the program (s) .
2. What is the significance of program and what Objective is achieved?

C. References:

[1] https://stackabuse.com/image-recognition-in-python-with-tensorflow-and-keras/
[2] . https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

Design an application using neural network for AND operation

```
import math
ch=input("Enter (1) for Bipolar Binary and (2) for Bipolar Continuous: ")
if ch=="1":
    w = [1, -1, 0, 0.5]
    n = input("Enter No. of Inputs: ")
    n=int(n)
    x=[]
    d=[]
    for i in range(0,n):
        m=i+1
        m=str(m)
        p = [float(j) for j in input("Enter 4 Values for X" + m +" [x1, x2, x3, x4]: " ).split()]
        x.append(p)
        p = float(input("Enter Value for D" + m +": "))
        d.append(p)

    c = input("Enter value for Learning Rate (c): ")
    print("\nW1: "+str(w))
    for j in range(0,n):
        deltaw=[]
        tnet=0
        for i in range(0,4):
            tnet=tnet+w[i]*x[j][i]
        print("DeltaW"+str(j+1)+": "+str(deltaw))
        print("W"+str(j+2)+": "+str(tw))
if ch=="2":
    w = [1, -1, 0, 0.5]
    n = input("Enter No. of Inputs: ")
    n=int(n)
    x=[]
    d=[]
    for i in range(0,n):
        m=i+1
        m=str(m)
        p = [float(j) for j in input("Enter 4 Values for X" + m +" [x1, x2, x3, x4]: " ).split()]
        x.append(p)
        p = float(input("Enter Value for D" + m +": "))
        d.append(p)

    c = input("Enter value for Learning Rate (c): ")
    print("\nW1: "+str(w))
    for j in range(0,n):
        deltaw=[]
        tnet=0
        for i in range(0,4):
            tnet=tnet+w[i]*x[j][i]
```

```
        tnet=tnet+w[i]*x[j][i]
    if tnet>=0:
        O=1
    else:
        O=-1

    deltaw=[i * (d[j] - O) * float(c) for i in x[j]]
    tw=[]
    for i in range(0,4):
tw.append(float("{0:.3f}".format(w[i]+deltaw[i])))
    w=tw[:]
    print("\nNet"+str(j+1)+":
"+str("{0:.3f}".format(tnet)))
    print("O"+str(j+1)+":
"+str("{0:.3f}".format(O)))
```

```
    O=(1 - math.exp(-tnet))/(1 + math.exp(-tnet))

    deltaw=[float("{0:.3f}".format(i * (d[j] - O) *
float(c))) for i in x[j]]
    tw=[]
    for i in range(0,4):

tw.append(float("{0:.3f}".format(w[i]+deltaw[i])))
    w=tw[:]
    print("\nNet"+str(j+1)+":
"+str("{0:.3f}".format(tnet)))
    print("O"+str(j+1)+":
"+str("{0:.3f}".format(O)))
    print("DeltaW"+str(j+1)+": "+str(deltaw))
    print("W"+str(j+2)+": "+str(tw))
```

Output:

```
Enter (1) for Bipolar Binary and (2) for Bipolar Continuous: 1
Enter No. of Inputs: 2
Enter 4 Values for X1 [x1, x2, x3, x4]: 1 -1 1 1
Enter Value for D1: 2
Enter 4 Values for X2 [x1, x2, x3, x4]: 1 1 -1 -1
Enter Value for D2: 2
Enter value for Learning Rate (c): 56

W1: [1, -1, 0, 0.5]

Net1: 2.500
O1: 1.000
DeltaW1: [56.0, -56.0, 56.0, 56.0]
W2: [57.0, -57.0, 56.0, 56.5]

Net2: -112.500
O2: -1.000
DeltaW2: [168.0, 168.0, -168.0, -168.0]
W3: [225.0, 111.0, -112.0, -111.5]
```

Design an application using neural network for OR operation

```
import math
# OR Gate Truth Table in Bipolar form:
# 0 -> -1 , 1 -> +1
x = [
   [-1, -1],  # 0 OR 0 -> -1
   [-1,  1],  # 0 OR 1 -> +1
   [ 1, -1],  # 1 OR 0 -> +1
   [ 1,  1]   # 1 OR 1 -> +1]
d = [-1, 1, 1, 1]  # Desired outputs in bipolar form
# Ask user which mode
ch = input("Enter (1) for Bipolar Binary and (2) for
Bipolar Continuous: ")
# Initialize weights and bias
w = [0.0, 0.0]  # for x1, x2
b = 0.0
c = float(input("Enter value for Learning Rate (c):
"))
epochs = int(input("Enter number of training
```

```
        print(f"DeltaW: {deltaw}, DeltaB:
{deltab:.3f}")
        print(f"Updated Weights: {w}, Bias:
{b:.3f}")
elif ch == "2":
   # Bipolar Continuous activation (tanh-like)
   for epoch in range(epochs):
       print(f"\nEpoch {epoch+1}:")
       for j in range(len(x)):
           tnet = w[0] * x[j][0] + w[1] * x[j][1] + b
           O = (1 - math.exp(-tnet)) / (1 +
math.exp(-tnet))  # Bipolar sigmoid
           deltaw = [c * (d[j] - O) * x[j][0],
                     c * (d[j] - O) * x[j][1]]
           deltab = c * (d[j] - O)
           w[0] += deltaw[0]
           w[1] += deltaw[1]
           b += deltab
```

```
epochs: "))
print("\nInitial Weights:", w, "Bias:", b)
if ch == "1":
    # Bipolar Binary activation
    for epoch in range(epochs):
        print(f"\nEpoch {epoch+1}:")
        for j in range(len(x)):
            tnet = w[0] * x[j][0] + w[1] * x[j][1] + b
            O = 1 if tnet >= 0 else -1
            deltaw = [c * (d[j] - O) * x[j][0],
                      c * (d[j] - O) * x[j][1]]
            deltab = c * (d[j] - O)
            w[0] += deltaw[0]
            w[1] += deltaw[1]
            b += deltab
            print(f"Input: {x[j]}, Net: {tnet:.3f}, Output: {O}, Desired: {d[j]}")
            print(f"Input: {x[j]}, Net: {tnet:.3f}, Output: {O:.3f}, Desired: {d[j]}")
            print(f"DeltaW: {deltaw}, DeltaB: {deltab:.3f}")
            print(f"Updated Weights: {w}, Bias: {b:.3f}")
# Testing after training
print("\nTesting OR Gate after training:")
for inp in x:
    tnet = w[0] * inp[0] + w[1] * inp[1] + b
    if ch == "1":
        O = 1 if tnet >= 0 else -1
    else:
        O = (1 - math.exp(-tnet)) / (1 + math.exp(-tnet))
    print(f"Input: {inp} -> Output: {O:.3f}")
```

Output:

```
Enter (1) for Bipolar Binary and (2) for Bipolar Continuous: 1
Enter value for Learning Rate (c): 56
Enter number of training epochs: 2

Initial Weights: [0.0, 0.0] Bias: 0.0

Epoch 1:
Input: [-1, -1], Net: 0.000, Output: 1, Desired: -1
DeltaW: [112.0, 112.0], DeltaB: -112.000
Updated Weights: [112.0, 112.0], Bias: -112.000
Input: [-1, 1], Net: -112.000, Output: -1, Desired: 1
DeltaW: [-112.0, 112.0], DeltaB: 112.000
Updated Weights: [0.0, 224.0], Bias: 0.000
Input: [1, -1], Net: -224.000, Output: -1, Desired: 1
DeltaW: [112.0, -112.0], DeltaB: 112.000
Updated Weights: [112.0, 112.0], Bias: 112.000
Input: [1, 1], Net: 336.000, Output: 1, Desired: 1
DeltaW: [0.0, 0.0], DeltaB: 0.000
Updated Weights: [112.0, 112.0], Bias: 112.000

Epoch 2:
Input: [-1, -1], Net: -112.000, Output: -1, Desired: -1
DeltaW: [-0.0, -0.0], DeltaB: 0.000
Updated Weights: [112.0, 112.0], Bias: 112.000
Input: [-1, 1], Net: 112.000, Output: 1, Desired: 1
DeltaW: [-0.0, 0.0], DeltaB: 0.000
Updated Weights: [112.0, 112.0], Bias: 112.000
Input: [1, -1], Net: 112.000, Output: 1, Desired: 1
DeltaW: [0.0, -0.0], DeltaB: 0.000
Updated Weights: [112.0, 112.0], Bias: 112.000
Input: [1, 1], Net: 336.000, Output: 1, Desired: 1
DeltaW: [0.0, 0.0], DeltaB: 0.000
Updated Weights: [112.0, 112.0], Bias: 112.000

Testing OR Gate after training:
Input: [-1, -1] -> Output: -1.000
Input: [-1, 1] -> Output: 1.000
Input: [1, -1] -> Output: 1.000
Input: [1, 1] -> Output: 1.000
```