Name : Keith Fernandes .   Roll No : 25

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.

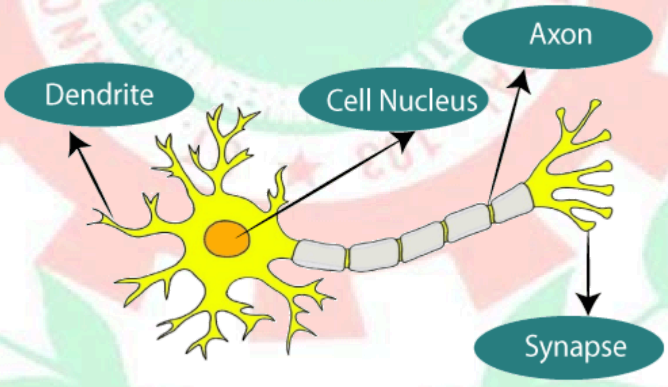


Fig:5.1 Biological Neural Network

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

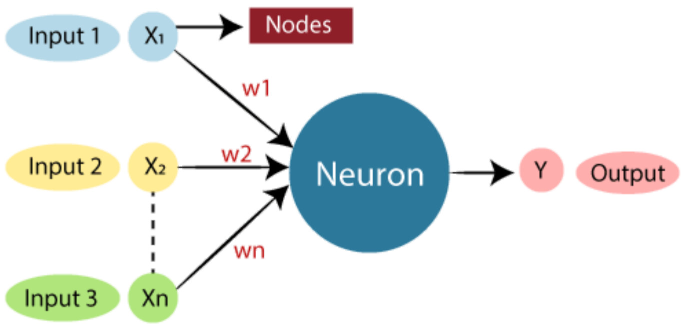


Fig 5.2: Artificial Neural Network

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 a 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?

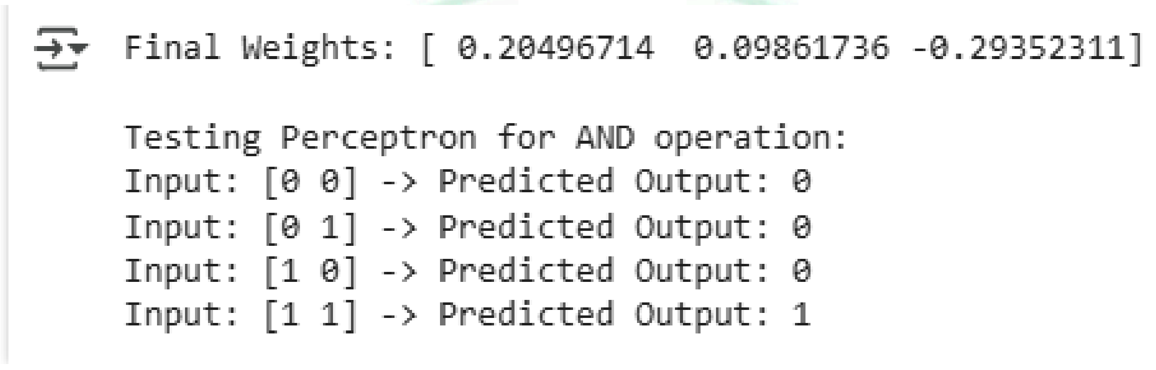
C. References:

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

Name : Keith Fernandes Roll No. : 25

AND Function :

```
import numpy as np

# Step function (activation function)
def step_function(x):
    return np.where(x >= 0, 1, 0)

# Training Data for AND operation
# Input data (X) with bias term included
X = np.array([
    [0, 0, 1],   # (x1, x2, bias)
                [0, 1, 1],
    [1, 0, 1],
    [1, 1, 1]
])

# Output labels for AND operation
y = np.array([0, 0, 0, 1])

# Initialize weights randomly
np.random.seed(42)
weights = np.random.randn(3) * 0.01

# Learning rate
lr = 0.1

# Training the perceptron
for epoch in range(20):
    for i in range(len(X)):
        z = np.dot(X[i], weights)      # weighted sum
        y_pred = step_function(z)      # activation
        error = y[i] - y_pred
        weights += lr * error * X[i]   # update rule

print("Final Weights:", weights)

# Testing the network
print("\nTesting Perceptron for AND operation:")
for i in range(len(X)):
    z = np.dot(X[i], weights)
    y_pred = step_function(z)
    print(f"Input: {X[i][0:2]} -> Predicted Output: {y_pred}"
```

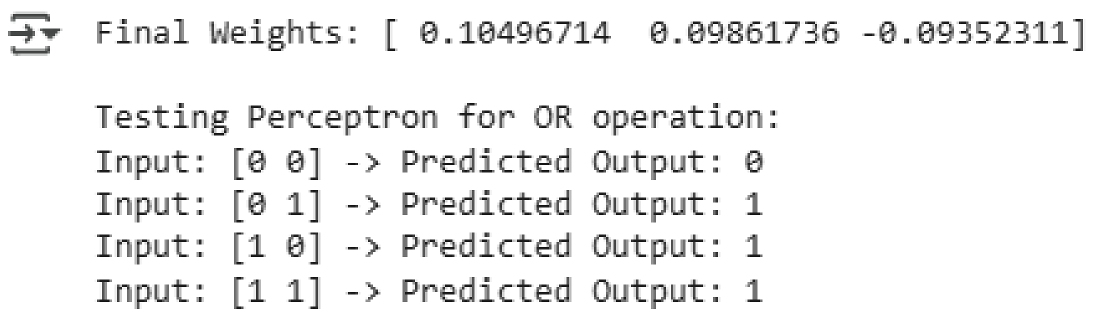


```
Final Weights: [ 0.20496714  0.09861736 -0.29352311]

Testing Perceptron for AND operation:
Input: [0 0] -> Predicted Output: 0
Input: [0 1] -> Predicted Output: 0
Input: [1 0] -> Predicted Output: 0
Input: [1 1] -> Predicted Output: 1
```

Name : Keith Fernandes Roll No. : 25

OR Function :

```
import numpy as np

# Step function (activation function)
def step_function(x):
    return np.where(x >= 0, 1, 0)

# Training Data for OR operation
# Input data (X) with bias term included
X = np.array([
    [0, 0, 1],   # (x1, x2, bias)
    [0, 1, 1],
    [1, 0, 1],
    [1, 1, 1]
])

# Output labels for OR operation
y = np.array([0, 1, 1, 1])

# Initialize weights randomly
np.random.seed(42)
weights = np.random.randn(3) * 0.01

# Learning rate
lr = 0.1

# Training the perceptron
for epoch in range(20):
    for i in range(len(X)):
        z = np.dot(X[i], weights)      # weighted sum
        y_pred = step_function(z)      # activation
        error = y[i] - y_pred
        weights += lr * error * X[i]   # update rule

print("Final Weights:", weights)

# Testing the network
print("\nTesting Perceptron for OR operation:")
for i in range(len(X)):
    z = np.dot(X[i], weights)
    y_pred = step_function(z)
    print(f"Input: {X[i][0:2]} -> Predicted Output: {y_pred}")
```



```
Final Weights: [ 0.10496714  0.09861736 -0.09352311]

Testing Perceptron for OR operation:
Input: [0 0] -> Predicted Output: 0
Input: [0 1] -> Predicted Output: 1
Input: [1 0] -> Predicted Output: 1
Input: [1 1] -> Predicted Output: 1
```