AISC

Class: B.E COMP

Experiment 08: Apply Mc-Culloch Pitts Model to solve a classification problem.

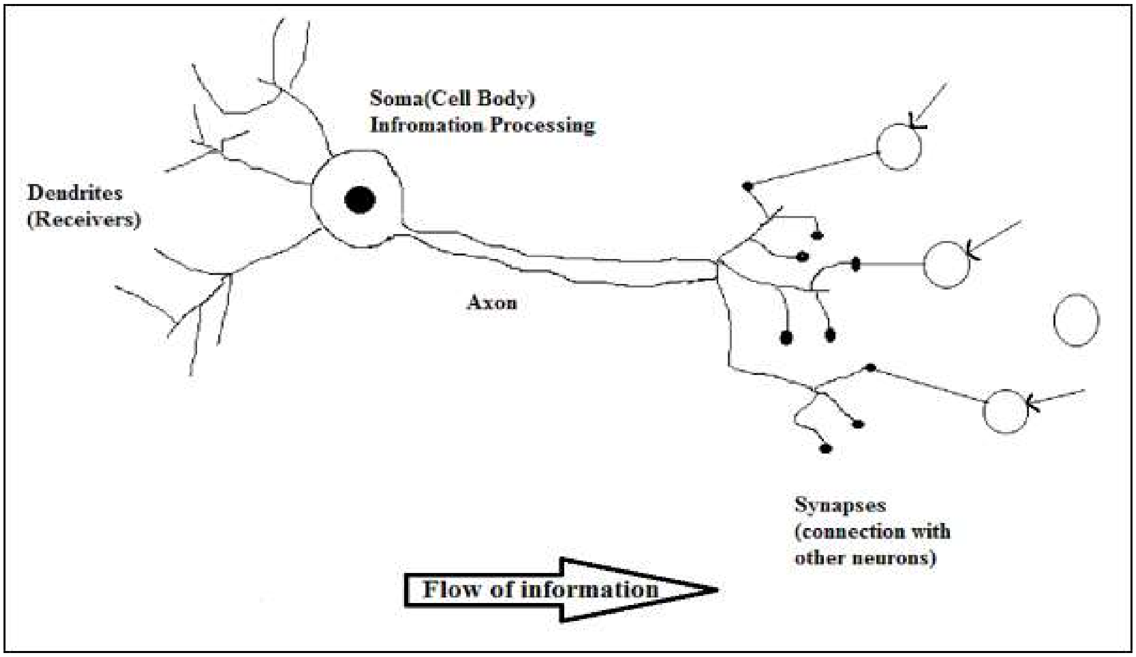
Learning Objective:

Basic Experiments

Apply Mc-Culloch Pitts Model to solve a classification problem.
Tools: Python

Theory:

A nerve cell neuronneuron is a special biological cell that processes information. According to an estimation, there are huge number of neurons, approximately 10^{11} with numerous interconnections, approximately 10^{15}.



Working of a Biological Neuron

As shown in the above diagram, a typical neuron consists of the following four parts with the help of which we can explain its working −

- **Dendrites** − They are tree-like branches, responsible for receiving the information from other neurons it is connected to. In other sense, we can say that they are like the ears of neuron.
- **Soma** − It is the cell body of the neuron and is responsible for processing of information, they have received from dendrites.
- **Axon** − It is just like a cable through which neurons send the information.
- **Synapses** − It is the connection between the axon and other neuron dendrites.

ANN versus BNN

Before taking a look at the differences between Artificial Neural Network ANNANN and Biological Neural Network BNNBNN, let us take a look at the similarities based on the terminology between these two.

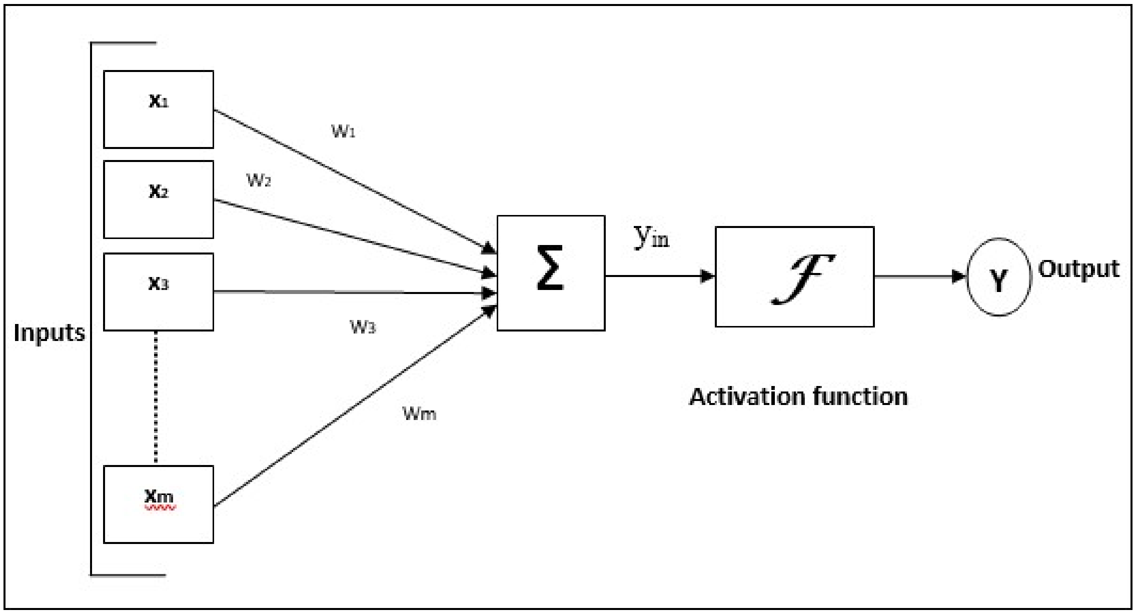
Biological Neural Network BNNBNN	**Artificial Neural Network** ANNANN
Soma	Node
Dendrites	Input
Synapse	Weights or Interconnections
Axon	Output

The following table shows the comparison between ANN and BNN based on some criteria mentioned.

Criteria	**BNN**	**ANN**
Processing	Massively parallel, slow but superior than ANN	Massively parallel, fast but inferior than BNN
Size	10^{11} neurons and 10^{15} interconnections	10^{2} to 10^{4} nodes mainly depends on the type of application and network designer mainly depends on the type of application and network designer
Learning	They can tolerate ambiguity	Very precise, structured and formatted data is required to tolerate ambiguity
Fault tolerance	Performance degrades with even partial damage	It is capable of robust performance, hence has the potential to be fault tolerant
Storage capacity	Stores the information in the synapse	Stores the information in continuous memory locations

Model of Artificial Neural Network

The following diagram represents the general model of ANN followed by its processing.



For the above general model of artificial neural network, the net input can be calculated as follows −

yin=x1.w1+x2.w2+x3.w3…xm.wmyin=x1.w1+x2.w2+x3.w3…xm.wm

i.e., Net input yin=∑mixi.wiyin=∑imxi.wi

The output can be calculated by applying the activation function over the net input.

Y=F(yin)Y=F(yin)

Output = function net input calculated

Processing of ANN depends upon the following three building blocks −

- Network Topology
- Adjustments of Weights or Learning
- Activation Functions

In this chapter, we will discuss in detail about these three building blocks of ANN

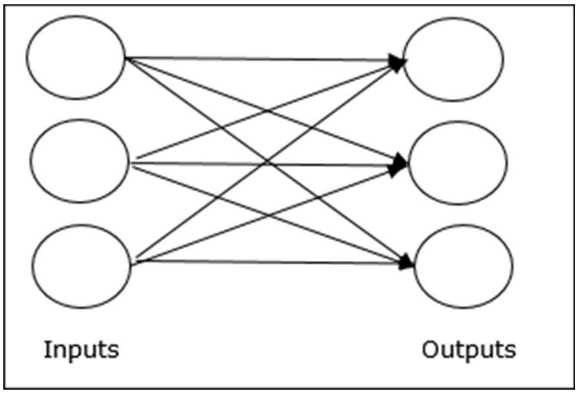
Network Topology

A network topology is the arrangement of a network along with its nodes and connecting lines. According to the topology, ANN can be classified as the following kinds −

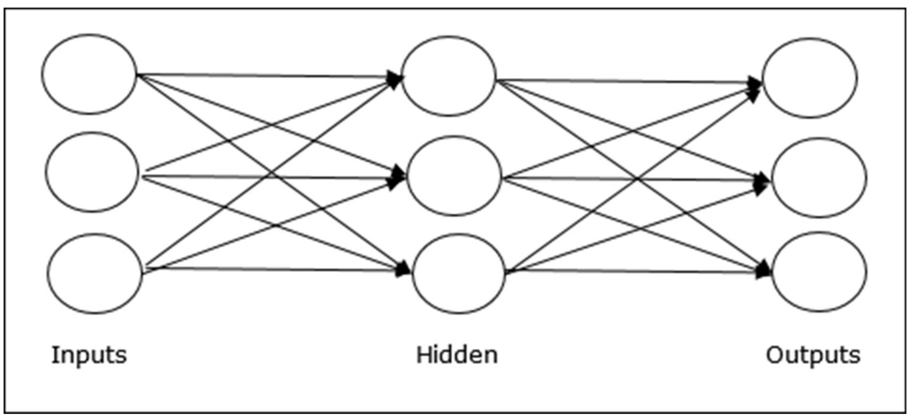
Feedforward Network

It is a non-recurrent network having processing units/nodes in layers and all the nodes in a layer are connected with the nodes of the previous layers. The connection has different weights upon them. There is no feedback loop means the signal can only flow in one direction, from input to output. It may be divided into the following two types −

- **Single layer feedforward network** − The concept is of feedforward ANN having only one weighted layer. In other words, we can say the input layer is fully connected to the output layer.



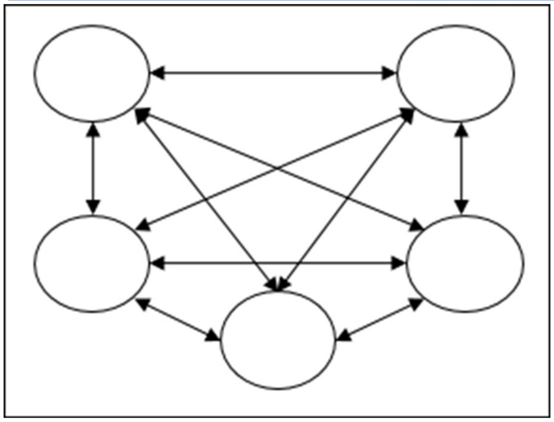
- **Multilayer feedforward network** − The concept is of feedforward ANN having more than one weighted layer. As this network has one or more layers between the input and the output layer, it is called hidden layers.



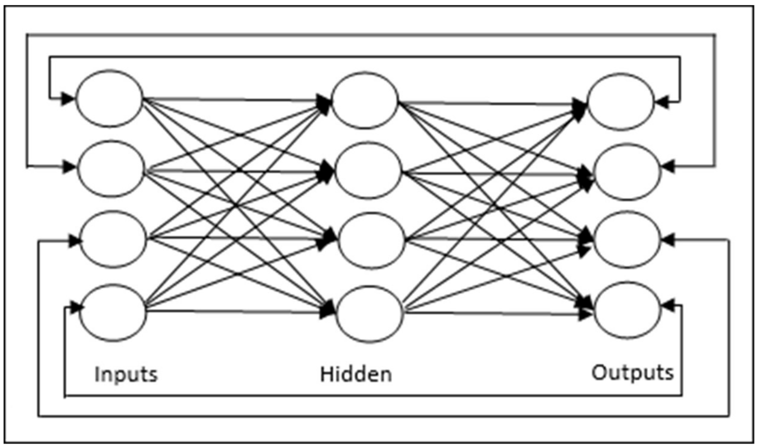
Feedback Network

As the name suggests, a feedback network has feedback paths, which means the signal can flow in both directions using loops. This makes it a non-linear dynamic system, which changes continuously until it reaches a state of equilibrium. It may be divided into the following types −

- **Recurrent networks** − They are feedback networks with closed loops. Following are the two types of recurrent networks.
- **Fully recurrent network** − It is the simplest neural network architecture because all nodes are connected to all other nodes and each node works as both input and output.



- **Jordan network** − It is a closed loop network in which the output will go to the input again as feedback as shown in the following diagram.



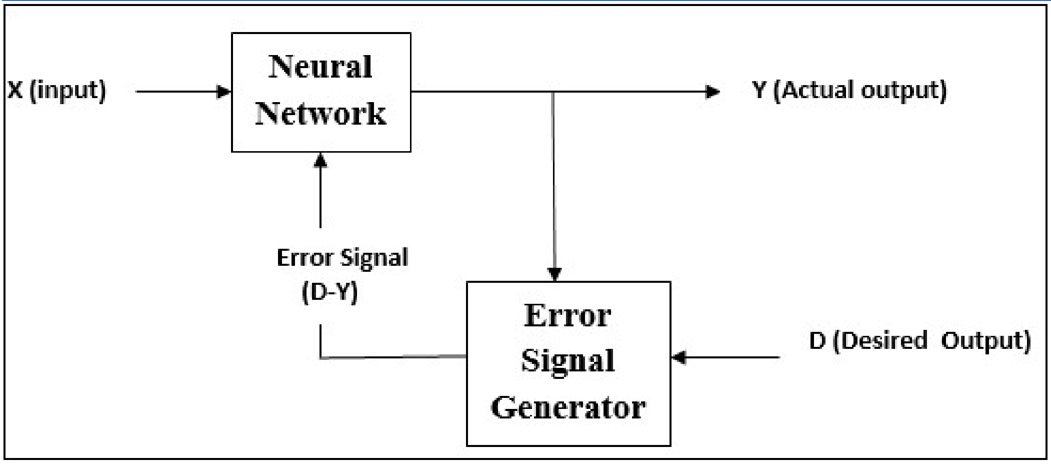
Adjustments of Weights or Learning

Learning, in artificial neural network, is the method of modifying the weights of connections between the neurons of a specified network. Learning in ANN can be classified into three categories namely supervised learning, unsupervised learning, and reinforcement learning.

Supervised Learning

As the name suggests, this type of learning is done under the supervision of a teacher. This learning process is dependent.

During the training of ANN under supervised learning, the input vector is presented to the network, which will give an output vector. This output vector is compared with the desired output vector. An error signal is generated, if there is a difference between the actual output and the desired output vector. On the basis of this error signal, the weights are adjusted until the actual output is matched with the desired output.



Implementation:

SOURCE CODE:

```
import time
from beautifultable import BeautifulTable

class Perceptron:

    def __init__(self, input_file_path, input_length, alpha=1, threshold = 0.2):

        self.input_matrix = []
        self.target_vector = []

        self._load_file(input_file_path, input_length)

        self.weights = [0] * len(self.input_matrix[0])
        self.bias = 0
        self.alpha = alpha
        self.threshold = threshold


    def _load_file(self, input_file_path, input_length):

        f = open(input_file_path, 'r')
        lines = f.readlines()
        num_ip = int(lines[0])

        i=1
        for j in range(num_ip):

            target = int(lines[i])
            input_vector = []
            for k in range(1,input_length+1):

                for x in lines[i+k][:-1].split(' '):
input_vector.append(self._pattern(x))

            i+=(input_length+1)


self.input_matrix.append(input_vector)
            self.target_vector.append(target)
        print("SUCCESS: Parsing of input successful.")
        print(input_vector)
        input_vector[8] = 1

    def _pattern(self, x):

        if x=='*':
            return 1
        elif x=='.':
            return -1


    def _activate(self, y_in):

        if y_in > self.threshold:
            return 1

        elif y_in <= self.threshold and y_in >= -self.threshold:
            return 0
```

```
        elif y_in < -self.threshold:
            return -1


    def _run_input(self, input_vector, target):

        assert len(input_vector) == len(self.weights)

        y_in=self.bias
        for i in range(len(input_vector)):
            y_in += input_vector[i] * self.weights[i]

        y = self._activate(y_in)

        if y != target:
            self._update_weights(input_vector, target)
            return True
        else:
            return False


    def _update_weights(self, input_vector, target):

        self.bias += self.alpha*target

        for i in range(len(self.weights)):
            self.weights[i] += self.alpha*target*input_vector[i]

    def _run_epoch(self, epoch):

        assert len(self.input_matrix) == len(self.target_vector)

        flag = False
        count_updated_inputs = 0

        for i in range(len(self.input_matrix)):
            weights_updated = self._run_input(self.input_matrix[i], self.target_vector[i])
            if weights_updated:
                count_updated_inputs += 1
                flag=True

        print("\n=== Epoch {} Summary ===\n".format(epoch))
        print("Table : Weights after epoch {}".format(epoch))
        self.output_weights()
        print("Number of inputs that updated weights : {}".format(count_updated_inputs))


        if flag:
            return True
        else:
            return False

    def train(self):

        print("========== Training phase ===========")

        start = time.perf_counter()
        stop_train = False

        epoch = 1

        while not stop_train:
            train = self._run_epoch(epoch)
            stop_train = not train
            epoch += 1

        print("\n" + '-'*50)

        print("\t\tFINAL TRAINED WEIGHTS")
        self.output_weights()

        print("\nTraining completed after : {} epochs".format(epoch-1))
        end =  time.perf_counter()
        print("Total time taken for training : {:.2} sec".format(end-start))


    def output_weights(self):

        table = BeautifulTable()
        table.column_headers = ['w'+str(i) for i in range(1, len(self.weights)+1)]
        table.append_row(self.weights)
```

```
        table.append_column("b",
[self.bias])
        print(table)

if __name__ == "__main__":
    p = Perceptron("pattern.txt", 3)
    p.train()
```

OUTPUT:

```
SUCCESS: Parsing of input successful.
[1, 1, 1, 1, 1, 1, 1, -1, None]
========== Training phase ===========

=== Epoch 1 Summary ===

Table : Weights after epoch 1
+----+----+----+----+----+----+----+----+----+---+
| w1 | w2 | w3 | w4 | w5 | w6 | w7 | w8 | w9 | b |
+----+----+----+----+----+----+----+----+----+---+
| 0  | 0  | 0  | -2 | 0  | -2 | 0  | 2  | 0  | 0 |
+----+----+----+----+----+----+----+----+----+---+
Number of inputs that updated weights : 2

=== Epoch 2 Summary ===

Table : Weights after epoch 2
+----+----+----+----+----+----+----+----+----+---+
| w1 | w2 | w3 | w4 | w5 | w6 | w7 | w8 | w9 | b |
+----+----+----+----+----+----+----+----+----+---+
| 0  | 0  | 0  | -2 | 0  | -2 | 0  | 2  | 0  | 0 |
+----+----+----+----+----+----+----+----+----+---+
Number of inputs that updated weights : 0

--------------------------------------------------
                FINAL TRAINED WEIGHTS
+----+----+----+----+----+----+----+----+----+---+
| w1 | w2 | w3 | w4 | w5 | w6 | w7 | w8 | w9 | b |
+----+----+----+----+----+----+----+----+----+---+
| 0  | 0  | 0  | -2 | 0  | -2 | 0  | 2  | 0  | 0 |
+----+----+----+----+----+----+----+----+----+---+

Training completed after : 2 epochs
Total time taken for training : 0.16 sec
```

Learning Outcomes: Students should have the ability to Mc-Culloch Pitts Model

LO1: Understand the problem formulation

Course Outcomes: Upon completion of the course students will be able to understand Mc-Culloch Pitts Model

Conclusion: Thus, the aim to study and implement Mc-Culloch Pitts Model

Viva Questions:

Ques.1 What do you understand by Mc-Culloch Pitts Model?

Ques. 2. Explain the basic steps in Mc-Culloch Pitts Model?

Ques. 3. Write types of problem discussed in detail.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				