# Experiment No. 2

**Title -** Implement an Feed Forward Network

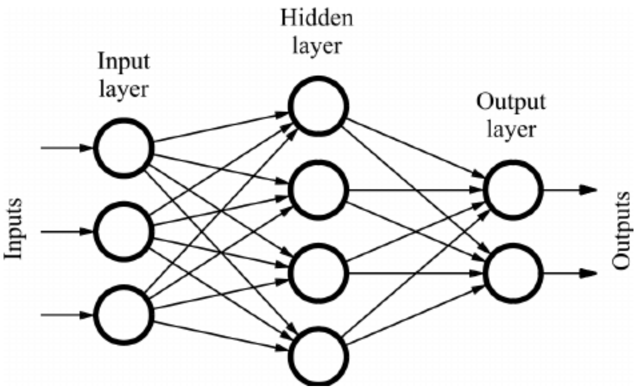
---

**Theory -**

A Feedforward Neural Network (FNN), also known as a multilayer perceptron (MLP), is a foundational architecture in artificial neural networks. It consists of multiple layers of neurons, where information flows in a forward direction from the input layer through hidden layers to the output layer without any feedback loops. Let's delve deeper into the mathematical representation and working of an FNN

**Structure**

- Input Layer: Receives input features.
- Hidden Layers: Intermediate layers between the input and output layers. Each layer contains multiple neurons that perform computations.
- Output Layer: Produces the final output, which can be predictions, classifications, etc.



**Mathematical Representation**

X : input vector or matrix (features); W : weights matrix for the $i^{th}$ layer

$b_i$ : bias vector for the $i^{th}$ layer; A : activation function

## Forward Propagation

The computation in each layer ***i*** can be represented as:

*1. Input to Hidden Layers:*

$$z = \sum_{I=1}^{n} W_i * X_i + b$$

( $W_i$ = weights; $X_i$ = inputs; b = bias )

$$A_z = \sigma(z) = \frac{1}{1 + e^{-z}}$$

*2. Hidden to Output Layer (Final Layer):*

$$Z_{output} = A_{hidden} * W_{output} + b_{output}$$

$$\hat{Y} = \sigma\ ( Z_{output} )$$

( $Z_{output}$ - final weighted sum; $\hat{Y}$ - predicted output )

## Activation Function

Typically, nonlinear functions like the **sigmoid :** $\sigma(z) = \frac{1}{1 + e^{-z}}$

**ReLU** (Rectified Linear Unit) : *relu(z) = max {0, z}* or ***tanh*** are used. Activation functions introduce nonlinearity, enabling the network to model complex relationships in the data.

## Implementation

```
import numpy as np

def activation(z):
  return 1/(1+np.exp(-z))

def loss_function(target,output):
  #Mean Squared Error
  return (1/len(target))*np.square(target-output)

def forwardpass(x,w1,w2,b1,b2):
  #input to hidden layer
  weighted_sum1=np.dot(x,w1)+b1
  print("Weighted Sum from input layer :\n",weighted_sum1)
  h_in=activation(weighted_sum1)
  #hidden to output layer
  weighted_sum2=np.dot(h_in,w2)+b2
  print("Weighted Sum from hidden layer :\n",weighted_sum2)
  h_out=activation(weighted_sum2)
  return h_out

x=np.array([[0.1, 0.3], [0.7, 0.2], [0.2, 0.4]]) #input data
targets=np.array([[0], [1], [1]]) #target data

# Weights and biases for the input layer to hidden layer
w1= np.array([[0.8, 0.1], [0.9, 0.2]])
b1 = np.array([0.1, 0.2])

# Weights and bias for the hidden layer to output layer
w2 = np.array([[0.7], [0.3]])
b2= np.array([0.1])

final_output=forwardpass(x,w1,w2,b1,b2)
loss=loss_function(targets,final_output)
print("Input Data :\n",x)
print("Output :\n",final_output)
print("Loss :\n",loss)
```

## Output

```
Weighted Sum from input layer :
 [[0.45 0.27]
 [0.84 0.31]
 [0.62 0.3 ]]
Weighted Sum from hidden layer :
 [[0.69757534]
 [0.76199123]
 [0.72748574]]
Input Data :
 [[0.1 0.3]
 [0.7 0.2]
 [0.2 0.4]]
Output :
 [[0.66764997]
 [0.6817859 ]
 [0.67425329]]
Loss :
 [[0.1485855 ]
 [0.03375341]
 [0.03537031]]
```

## Conclusion

A Feedforward Neural Network processes data through multiple layers of interconnected neurons, applying weighted sums and activation functions to produce predictions or classifications. The network learns by adjusting parameters during training, aiming to minimise prediction errors through forward and backward propagation. This architecture enables the modelling of complex relationships in data and is widely used in various machine learning tasks.