Name- Shalini Sharma (M.Tech Data Science)

Enrollment number - 21/10/MI/021

Feed Forward Network

In [15]:

```
import numpy as np
import matplotlib.pyplot as plt
i_input=np.array([[0,0],[0,1],[1,0],[1,1]]) #input
labels=np.array([0,0,0,1])
weights=[0.584,0.997]
threshold=0.54         #threshold value

def step_fun(sum):
    if sum>threshold:
        return 1
    else:
        return 0

updated_labels=[]
for i in range (0, i_input.shape[0]):
    actual_value=labels[i]
    instances=i_input[i]
    x0=instances[0]
    x1=instances[1]
    z=x0*weights[0]+x1*weights[1]
    fire= step_fun(z)
    updated_labels.append(fire)
    delta=actual_value-fire
    print("Predicted", fire ," Actual Value", labels[i] ," Error ", delta)
```

```
Predicted 0  Actual Value 0  Error  0
Predicted 1  Actual Value 0  Error  -1
Predicted 1  Actual Value 0  Error  -1
Predicted 1  Actual Value 1  Error  0
```

Training Rule

Wi=Wi+ ΔW

ΔW=η(t-o)Xi

η is positive learning rate.

In [14]:

```python
import numpy as np
import matplotlib.pyplot as plt
i_input=np.array([[0,0],[0,1],[1,0],[1,1]]) #input values of AND gate
y=np.array([0,0,0,1])  #y is target output for each input of i_input set
w=[0.78,0.91]  #associated weights
threshold=0.54  #threshold value
iteration=5
eta=0.1       #eta is learning rate

# Defining step function
def step_fun(sum):
    if sum>threshold:
        return 1
    else:
        return 0
print("Initial Weights ", w)

#iterating through i_input array to  calculate Z
updated_labels=[]
for j in range(0,iteration):
    print("Iteration ",j)
    print("Actual(y)"," ","Predicted(y')"," ","Error")
    for i in range (0, i_input.shape[0]):
        actual_value=y[i]
        instances=i_input[i]
        x0=instances[0]
        x1=instances[1]
        z=x0*w[0]+x1*w[1] # Z is sum of Product of Inputs and their associated
        fire= step_fun(z)
        updated_labels.append(fire)
        delta=actual_value-fire  #delta is Error (When Error is  0 it means pre
        print( y[i],"  "*12,fire,"  "*12,delta)
        w[0]=w[0]+delta*eta #Updating Weights
        w[1]=w[1]+delta*eta
    print("_"*35)
print("Updated Weights",w) #Updated Weights after learning
```

```
Initial Weights  [0.78, 0.91]
Iteration  0
Actual(y)   Predicted(y')   Error
0              0              0
0              1              -1
0              1              -1
1              1              0
___________________________________
Iteration  1
Actual(y)   Predicted(y')   Error
0              0              0
0              1              -1
0              0              0
1              1              0
___________________________________
Iteration  2
Actual(y)   Predicted(y')   Error
0              0              0
0              1              -1
0              0              0
1              1              0
___________________________________
Iteration  3
Actual(y)   Predicted(y')   Error
0              0              0
0              0              0
0              0              0
```

```
1               1               0
_____________________________________
Iteration  4
Actual(y)   Predicted(y')   Error
0               0               0
0               0               0
0               0               0
1               1               0
_____________________________________
Updated Weights [0.3800000000000001, 0.5100000000000001]
```

Summary

Initially a random weight was chosen and the Two predicted outputs were misclassified.

After applying Perceptron Training Rule , Weights were Modified till it classified Examples correctly till some iteration

Initially weights was [0.78,0.91] after Updation [0.38, 0.51] and this updated weights predicted output Correctly after few iterations

Gradient Descent Rule

Activation fun 1(/1+e^-weighted_sum)

weighted_sum=W1*X1 + W2*X2 +....Wi*Xi+Bias

Loss = -(target*log(pred)+(1-target)*log(1-pred))

Wi=Wi+ ΔW

ΔW=η(t-o)Xi

New Bias(b')= Old Bias(b) + η*(target-predicted)

η is Learning rate which ensures gradual weight update

Bias helps to tune our model .

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
def Activation_fun(z): #z is weighted sum of input and associated weights
    return 1/(1+np.e**-z)
def get_prediction(Input,Weights,bias):
    return Activation_fun(np.dot((Input,Weights)+bias))
def Gradient_Descent(Input, Weights, Target, Prediction, eta,bias):
    new_weight=[]
    bias=bias+eta*(Target-Prediction)
    for x,w in zip(Input,Weights):
        new_w=w+eta*(Target-Prediction)*x
        new_weight.append(new_w)
    return new_weight,bias

#DATA
Input=np.array([[0,1,0],[0,1,1],[1,1,0],[1,1,1],[1,0,0]])
Target=np.array([0,1,1,0,1])
Weights=np.array([0.3,0.1,0.5,-0.1,0.45])
bias=0.5
eta=0.01
for i in range(10):
    for x,y in zip(Input, Target):
        pred=get_prediction(x,Weights, bias)
        weights,bias=Gradient_Descent(x,Weights,y,pred,eta,bias)
```