**Aim:** Single Layer Perceptron for Logic Gates

**Program:**

import numpy as np

x=np.array([[1,1],[1,-1],[-1,1],[-1,-1]])

t=np.array([[-1],[1],[-1],[-1]])

w=np.array([[0],[0]])

b=0

theta=float(input("Enter new theta:"))

alpha=float(input("Enter new alpha:"))

yin=np.zeros(shape=(4,1))

y=np.zeros(shape=(4,1))

i=0

found=0

while(found==0):

yin=x[i][0]\*w[0]+x[i][1]\*w[1]

yin = yin+b

if(yin>theta):

y[i] = 1

elif(yin<=theta and yin>=-theta):

y[i]=0

else:

y[i]=-1

if (y[i]==t[i]):

print("NO UPDATION REQUIRED")

print(y[i])

if(i<3):

i=i+1

else:

i=0

else:

print("MODEL IS NOT TRAINED")

print("The value of output is")

print(y)

w[0]=w[0]+alpha\*x[i][0]\*t[i]

w[1]=w[1]+alpha\*x[i][1]\*t[i]

b = b+alpha\*t[i]

if(i<3):

i=i+1

else:

i=0

if(y==t).all():

found=1

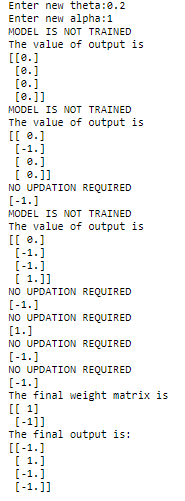
print("The final weight matrix is ")

print(w)

print("The final output is:")

print(y)

**Output:**



**Aim:** Plotting of various activation functions

1. y={1 x<10 0   else

**Program:**

import matplotlib.pyplot as plt

x=[num for num in range(0,20)]

y=[]

for i in x:

if i>=10:

y.append(1)

else:

y.append(0)

plt.plot(x,y)

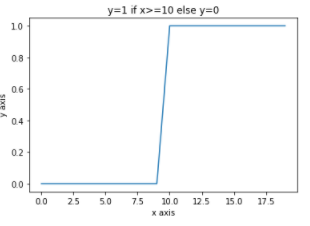
plt.xlabel("x axis")

plt.ylabel("y axis")

plt.title("y=1 if x>=10 else y=0")

plt.show()

**Output:**



**2. y=eax for different values of a.**

**Program:**

import numpy as np

import matplotlib.pyplot as plt

num=[num for num in range(0,10)]

a=2 #constant value

x=np.array(num)\*2

y=np.exp(x)

plt.plot(x,y)

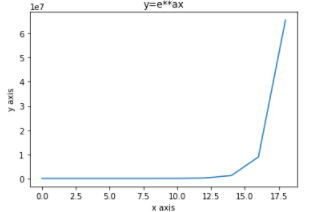
plt.xlabel("x axis")

plt.ylabel("y axis")

plt.title("y=e\*\*ax")

plt.show()

**Output:**



3. y=7x2+3x+10 for 2x5

Program:

import matplotlib.pyplot as plt

x=[x for x in range(2,6)]

y=[]

for i in x:

y.append(7\*pow(i,2)+3\*i+10)

plt.plot(x,y)

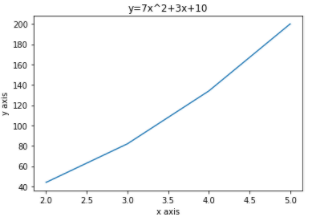
plt.xlabel('x axis')

plt.ylabel('y axis')

plt.title('y=7x^2+3x+10')

plt.show()

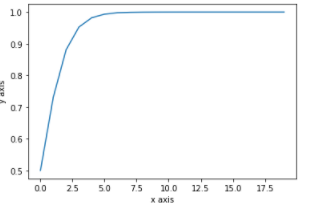
**Output:**



**4. y=11+e-x**

**Program:** import matplotlib.pyplot as plt  
import numpy as np  
x=[x for x in range(0,20)]  
p=np.array(x)  
p=1+np.exp(-p)  
y=1/p  
plt.plot(x,y)  
plt.xlabel('x axis')  
plt.ylabel('y axis')  
plt.show()

**Output:**



**5. y=1-e-ax1+e-axfor different values of a.**

**Program:**

import matplotlib.pyplot as plt

import numpy as np

num=[num for num in range(0,21)]

num=np.array(num)

a=2

p=1-np.exp(-num\*a)

q=1+np.exp(-num\*a)

y=p/q

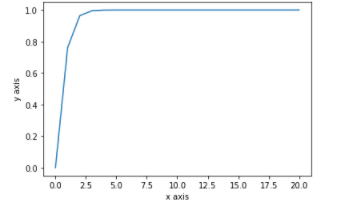
plt.plot(num,y)

plt.xlabel('x axis')

plt.ylabel('y axis')

plt.show()

**Output:**



**6. y=tan hx**

**Program:**

import numpy as np

import matplotlib.pyplot as plt

in\_array = np.linspace(0, np.pi, 12)

h=2

out\_array =h\*np.tan(in\_array)

print("in\_array : ", in\_array)

print("\nout\_array : ",out\_array)

# red for numpy.tan()

plt.plot(in\_array, out\_array, color='red', marker="o")

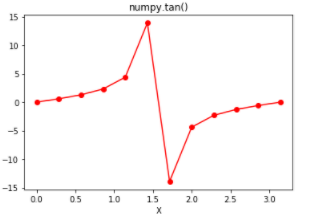
plt.title("numpy.tan()")

plt.xlabel("X")

plt.ylabel("Y")

plt.show()

Output:



**Aim:** Demonstration of Linear seperability

**Program:**

import numpy as np

import matplotlib as plt

x = np.array([0,1])

y = np.array([0,1])

plt.pyplot.scatter(x,y,c='red')

x = np.array([1,0])

y = np.array([0,1])

plt.pyplot.scatter(x,y,c="blue")

plt.pyplot.xlabel('Input 1')

plt.pyplot.ylabel('Input 2')

w=-1

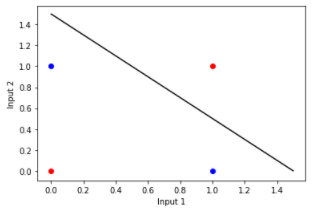
b=1.5

x = np.linspace(0,1.5)

plt.pyplot.plot(x,w\*x+b,c='black')

plt.pyplot.show()

**Output:**



**Aim**:- Adaline demonstration for one logic gate.

**Program:**

importnumpy as np

x1=np.array([[1,1,-1,-1]])

x2=np.array([[1,-1,1,-1]])

t=np.array([[1],[1],[1],[-1]])

w11=0.1

w21=0.1

w01=0.1

alpha=0.1

i=0

bias=1

w1=np.zeros((4,1))

w2=np.zeros((4,1))

w0=np.zeros((4,1))

Yin=np.zeros((4,1))

y=np.zeros((4,1))

error=np.zeros((4,1))

count=0

while(count!=3):

    i=0

if(count!=0):

        w11=w1[3]

        w21=w2[3]

        w01=w0[3]

while(i!=4):

if(i==0):

            Yin[i]= (x1[0][i]\*w11)+(x2[0][i]\*w21)+(bias\*w01)

y[i]=t[i][0]-Yin[i]

w1[i]=w11+(alpha\*y[i]\*x1[0][i])

w2[i]=w21+(alpha\*y[i]\*x2[0][i])

w0[i]=w01+(alpha\*y[i]\*bias)

else:

if(i>0 & i<=4):

                Yin[i]= (x1[0][i]\*w1[i-1])+(x2[0][i]\*w2[i-1])+(bias\*w0[i-1])

y[i]=t[i][0]-Yin[i]

w1[i]=w1[i-1]+(alpha\*y[i]\*x1[0][i])

w2[i]=w2[i-1]+(alpha\*y[i]\*x2[0][i])

w0[i]=w0[i-1]+(alpha\*y[i]\*bias)

error[i]=(y[i])\*\*2

        i=i+1

print('EPOCH',(count+1),':')

print('\n')

print('w1:',w1)

print('\n')

print('w2:',w2)

print('\n')

print('w0:',w0)

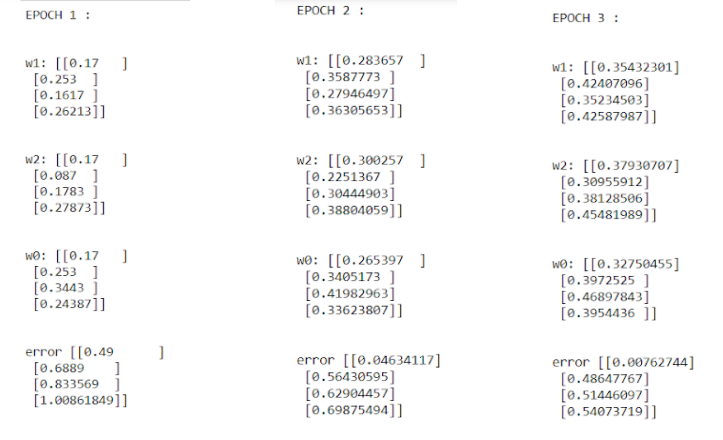
print('\n')

print('error',error)

print('\n\n')

count=count+1

**Output:**



**Aim:** Madaline for any given network.

**Program:**

import numpy as np

x=np.array([[1,1],[1,-1],[-1,1],[-1,-1]])

t=np.array([[1],[1],[1],[-1]])

w=np.array([[0],[0]])

b=0

theta=float(input("enter new theta"))

alpha=float(input("enter new alpha"))

yin=np.zeros(shape=(4,1))

y=np.zeros(shape=(4,1))

i=0

found=0

while(found==0):

yin=x[i][0]\*w[0]+x[i][1]\*w[1]

yin = yin+b

if(yin>theta):

y[i] = 1

elif(yin<=theta and yin>=-theta):

y[i]=0

else:

y[i]=-1

if (y[i]==t[i]):

print("NO UPDATION REQUIRED")

print(y[i])

if(i<3):

i=i+1

else:

i=0

else:

print("MODEL IS NOT TRAINED")

print("The value of output is")

print(y)

w[0]=w[0]+alpha\*x[i][0]\*t[i]

w[1]=w[1]+alpha\*x[i][1]\*t[i]

b = b+alpha\*t[i]

if(i<3):

i=i+1

else:

i=0

if(y==t).all():

found=1

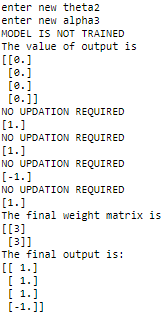
print("The final weight matrix is ")

print(w)

print("The final output is:")

print(y)

**Output:**



**Aim:** Back Propagation algorithm for a given network

**Program:**

import math

import random

import string

class NN:

def \_\_init\_\_(self, NI, NH, NO):

    # number of nodes in layers

    self.ni = NI + 1 # +1 for bias

self.nh = NH

    self.no = NO

    self.ai, self.ah, self.ao = [],[], []

    self.ai = [1.0]\*self.ni

self.ah = [1.0]\*self.nh

    self.ao = [1.0]\*self.no

self.wi = makeMatrix (self.ni, self.nh)

self.wo = makeMatrix (self.nh, self.no)

    # initialize node weights to random vals

randomizeMatrix ( self.wi, -0.2, 0.2 )

randomizeMatrix ( self.wo, -2.0, 2.0 )

    self.ci = makeMatrix (self.ni, self.nh)

    self.co = makeMatrix (self.nh, self.no)

defrunNN (self, inputs):

iflen(inputs) != self.ni-1:

print('incorrect number of inputs')

for i in range(self.ni-1):

      self.ai[i] = inputs[i]

for j in range(self.nh):

sum = 0.0

for i in range(self.ni):

sum +=( self.ai[i] \* self.wi[i][j] )

self.ah[j] = sigmoid (sum)

for k in range(self.no):

sum = 0.0

for j in range(self.nh):

sum +=( self.ah[j] \* self.wo[j][k] )

       self.ao[k] = sigmoid (sum)

return self.ao

defbackPropagate (self, targets, N, M):

output\_deltas = [0.0] \* self.no

for k in range(self.no):

error = targets[k] - self.ao[k]

output\_deltas[k] =  error \* dsigmoid(self.ao[k])

for j in range(self.nh):

for k in range(self.no):

change = output\_deltas[k] \* self.ah[j]

self.wo[j][k] += N\*change + M\*self.co[j][k]

         self.co[j][k] = change

hidden\_deltas = [0.0] \* self.nh

for j in range(self.nh):

error = 0.0

for k in range(self.no):

error += output\_deltas[k] \* self.wo[j][k]

hidden\_deltas[j] = error \* dsigmoid(self.ah[j])

for i in range (self.ni):

for j in range (self.nh):

change= hidden\_deltas[j] \* self.ai[i]

self.wi[i][j] += N\*change + M\*self.ci[i][j]

         self.ci[i][j] = change

error = 0.0

for k in range(len(targets)):

error = 0.5 \* (targets[k]-self.ao[k])\*\*2

return error

def weights(self):

print('Input weights:')

for i in range(self.ni):

print (self.wi[i])

print()

print('Output weights:')

for j in range(self.nh):

print (self.wo[j])

print ('')

def test(self, patterns):

for p in patterns:

inputs = p[0]

print('Inputs:', p[0], '-->', self.runNN(inputs), '\tTarget', p[1])

def train (self, patterns, max\_iterations = 1000, N=0.5, M=0.1):

for i in range(max\_iterations):

for p in patterns:

inputs = p[0]

targets = p[1]

self.runNN(inputs)

error = self.backPropagate(targets, N, M)

if i % 50 == 0:

print('Combined error', error)

self.test(patterns)

def sigmoid (x):

returnmath.tanh(x)

defdsigmoid (y):

return 1 - y\*\*2

defmakeMatrix ( I, J, fill=0.0):

  m = []

for i in range(I):

m.append([fill]\*J)

return m

defrandomizeMatrix ( matrix, a, b):

for i in range ( len (matrix) ):

for j in range ( len (matrix[0]) ):

matrix[i][j] = random.uniform(a,b)

def main ():

pat = [

      [[0,0], [1]],

      [[0,1], [1]],

      [[1,0], [1]],

      [[1,1], [0]]

  ]

myNN = NN ( 2, 2, 1)

myNN.train(pat)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output:**

