INDEX

|  |  |
| --- | --- |
| Practical No | Details |
| 1 | Implement the following |
| a |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Practical 1

1. Design a simple linear neural network

Code:

print("Name: Harsh Save")

X=float(input("Enter the input X"))

weight=float(input("Enter the input weight"))

bias=float(input("Enter the input bias"))

out=-1

Yin=(bias+(X\*weight))

print("Y=X\*weight+bias is {}".format(Yin))

if Yin <0:

out=0

elif Yin>=0 and Yin<1:

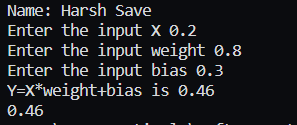
out=Yin

else:

out=1

print(out)

Output:



A neural network for multiple inputs

Code:

print("Name: Harsh Save")

n=int(input("Enter the number of inputs"))

Yin=0

for \_ in range(n):

weight=float(input("Enter the weight"))

X=float(input("Enter the X"))

Yin=Yin+weight\*X

print("Yin= {}".format(Yin))

if Yin <0:

out=0

elif Yin>1:

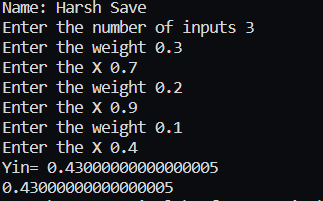
out=1

else:

out=Yin

print(out)

Output:



1. Calculate the output of neural net using both binary and bipolar sigmoidal function

Code:

import math

print("Name: Harsh Save")

n=int(input("Enter number of input"))

Yin=0

for \_ in range(n):

x=float(input("Enter x"))

w=float(input("Enter w"))

b=float(input("Enter b"))

Yin=(Yin+(w\*b)+b)

print("Yin ",Yin)

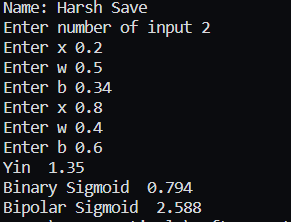
binary\_sigmoid=(1/(1+(math.e\*\*(-Yin))))

bipolar\_sigmoid=(2/(1+(math.e\*\*(-Yin))))+1

print("Binary Sigmoid ",round(binary\_sigmoid,3))

print("Bipolar Sigmoid ",round(bipolar\_sigmoid,3))

Output:



Practical 2

1. Generate AND/NOT function using McCulloch-Pitts neural net.

Code:

print("Name: Harsh Save")

print("XOR function using Mc-Culloch Pitts neuron")

x1inputs=[]

x2inputs=[]

c=input("Enter 1 to input values or press enter to use default values")

if c=='1':

for \_ in range(0,4):

x1=int(input("Enter x1"))

x1inputs.append(x1)

x2=int(input("Enter x2"))

x2inputs.append(x2)

else:

x1inputs=[1,1,0,0]

x2inputs=[1,0,1,0]

print("Considering one weight are excitatory and the other inhibitory")

w1=[1,1,1,1]

w2=[-1,-1,-1,-1]

z1=[]

for i in range(0,4):

z1.append(x1inputs[i]\*w1[i]+x2inputs[i]\*w2[i])

print("X1","X2","Z1")

for i in range(0,4):

print(x1inputs[i]," ",x2inputs[i]," ",z1[i])

print("Considering one weight are excitatory and the other inhibitory")

w2=[1,1,1,1]

w1=[-1,-1,-1,-1]

z2=[]

for i in range(0,4):

z2.append(x1inputs[i]\*w1[i]+x2inputs[i]\*w2[i])

print("X1","X2","Z2")

for i in range(0,4):

print(x1inputs[i]," ",x2inputs[i]," ",z2[i])

print("Applying Threshold")

for i in range(0,4):

if z1[i]>=1:

z1[i]=1

else:

z1[i]=0

if z2[i]>=1:

z2[i]=1

else:

z2[i]=0

print("z1"," ","z2")

for i in range(0,4):

print(z1[i]," ",z2[i])

Y=[]

v1=1

v2=1

for i in range(0,4):

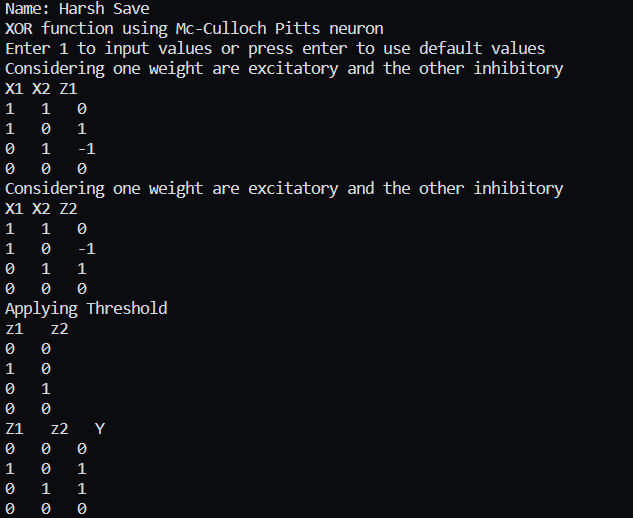
Y.append(z1[i]\*v1+z2[i]\*v2)

print("Z1"," ","z2"," ","Y")

for i in range(0,4):

print(z1[i]," ",z2[i]," ",Y[i])

Output:



1. Generate XOR function using McCulloch-Pitts neural net.

Code:

print("Name: Harsh Save")

print("XOR function using Mc-Culloch Pitts neuron")

c=input("Enter 1 to input values or press enter to use default values")

x1inputs=[]

x2inputs=[]

if c=='1':

for \_ in range(0,4):

x1=int(input("Enter x1"))

x1inputs.append(x1)

x2=int(input("Enter x2"))

x2inputs.append(x2)

else:

x1inputs=[1,1,0,0]

x2inputs=[1,0,1,0]

print("Considering all weights are excitatory")

w1=[1,1,1,1]

w2=[1,1,1,1]

y=[]

for i in range(0,4):

y.append(x1inputs[i]\*w1[i]+x2inputs[i]\*w2[i])

print("X1","X2","Y")

for i in range(0,4):

print(x1inputs[i]," ",x2inputs[i]," ",y[i])

print("Considering one weight are excitatory and the other inhibitory")

w1=[1,1,1,1]

w2=[-1,-1,-1,-1]

y=[]

for i in range(0,4):

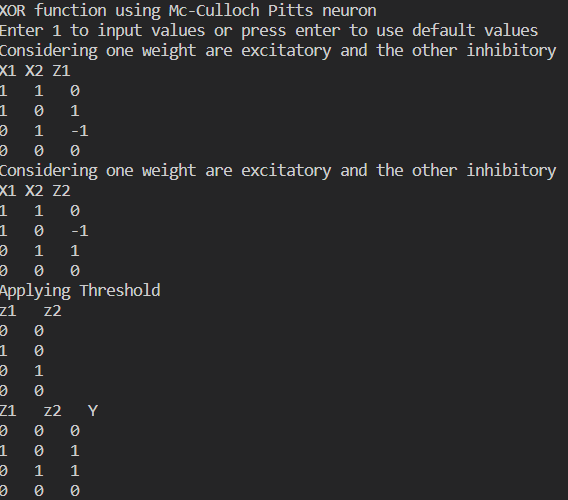
y.append(x1inputs[i]\*w1[i]+x2inputs[i]\*w2[i])

print("X1","X2","Y")

for i in range(0,4):

print(x1inputs[i]," ",x2inputs[i]," ",y[i])

Output:



Practical-3

1. Implement Hebb Rule

Code:

w1=[0,0,0,0]

w2=[0,0,0,0]

for m in range(0,4):

print("Enter 4 binary input values")

s=[]

t=[]

for i in range(0,4):

x=int(input())

s.append(x)

print("Enter 2 binary target values")

for i in range(0,2):

x=int(input())

t.append(x)

print("s= ",s)

print("t= ",t)

w1New=[]

for i in range(0,4):

newWeight=w1[i]+s[i]\*t[0]

w1New.append(newWeight)

for i in range(0,4):

print("W",(i+1),"1= ",w1New[i])

w2New=[]

for i in range(0,4):

newWeight=w2[i]+s[i]\*t[1]

w2New.append(newWeight)

for i in range(0,4):

print("W",(i+1),"2= ",w2New[i])

w1=w1New

w2=w2New

print(w1)

print(w2)

print("The final weight matrix is ")

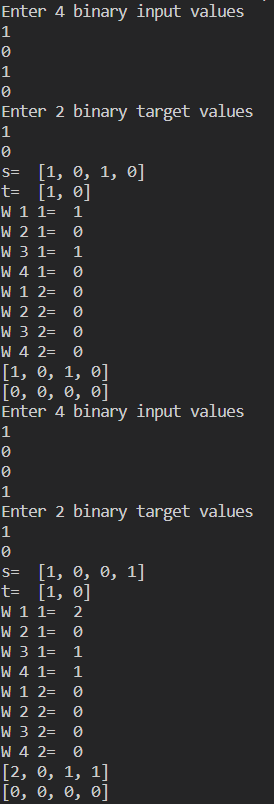
print("W= ")

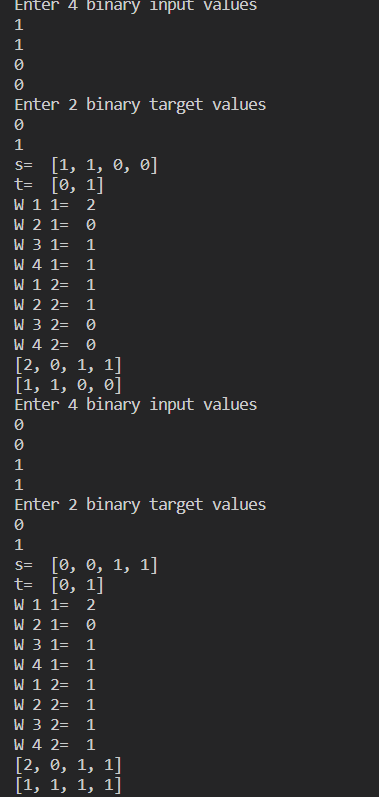
for i in range(0,4):

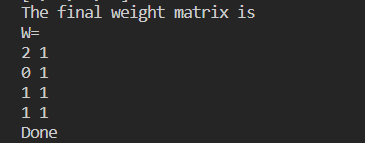
print(w1[i],w2[i])

print("Done")

Output:







1. Implement Delta rule

Code:

import math

x1=[0.3,0.5,0.8]

w1=[0.1,0.1,0.1]

t=1

a=0.1

diff=1

Yin=0

print("Using 3 inputs 3 weights 1 output.")

while(diff>0.4):

for i in range(0,3):

Yin=Yin+(x1[i]\*w1[i])

Yin=Yin+0.25

Yin=round(Yin,3)

print("Yin= ",Yin)

print("Target= ",t)

diff=t-Yin

diff=round(diff,3)

diff=math.fabs(diff)

print("Error= ",diff)

newW1=[]

for i in range(0,3):

W1new=w1[i]+a\*diff\*x1[i]

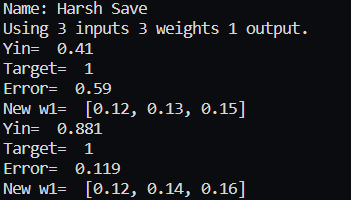
W1new=round(W1new,2)

newW1.append(W1new)

print("New w1= ",newW1)

w1=newW1

Output:



Practical-4

1. Write a program for Back Propagation Algorithm

Code:

import pandas as pd

import numpy as np

X=np.array(([2,9],[1,5],[3,6]),dtype=float)

Y=np.array(([92],[86],[89]),dtype=float)

X=X/np.amax(X,axis=0)

Y=Y/100

class NM:

def \_\_init\_\_(self):

self.inputsize=2

self.outputsize=1

self.hiddensize=3

self.W1=np.random.randn(self.inputsize,self.hiddensize)

self.W2=np.random.randn(self.hiddensize,self.outputsize)

def forward(self,X):

self.z=np.dot(X,self.W1)

self.z2=self.sigmoid(self.z)

self.z3=np.dot(self.z2,self.W2)

op=self.sigmoid(self.z3)

return op

def sigmoid(self,s):

return 1/(1+np.exp(-s))

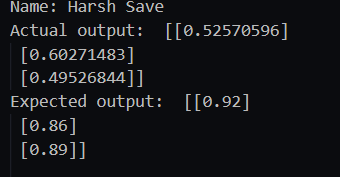
obj=NM()

op=obj.forward(X)

print("Actual output: ",str(op))

print("Expected output: ",str(Y))

Output:



1. Write a program for error Backpropagation algorithm

Code:

import pandas as pd

import numpy as np

X=np.array(([2,9],[1,5],[3,6]),dtype=float)

Y=np.array(([92],[86],[89]),dtype=float)

X=X/np.amax(X,axis=0)

Y=Y/100

class NM:

def \_\_init\_\_(self):

self.inputsize=2

self.outputsize=1

self.hiddensize=3

self.W1=np.random.randn(self.inputsize,self.hiddensize)

self.W2=np.random.randn(self.hiddensize,self.outputsize)

def forward(self,X):

self.z=np.dot(X,self.W1)

self.z2=self.sigmoid(self.z)

self.z3=np.dot(self.z2,self.W2)

op=self.sigmoid(self.z3)

return op

def sigmoid(self,s):

return 1/(1+np.exp(-s))

def sigmoidalprime(self,s):

return s\*(1-s)

def backward(self,X,Y,o):

self.o\_error=Y-o

self.o\_delta=self.o\_error \* self.sigmoidalprime(o)

self.z2\_error=self.o\_delta.dot(self.W2.T)

self.z2\_delta=self.z2\_error \* self.sigmoidalprime(self.z2)

self.W1 = self.W1 + X.T.dot(self.z2\_delta)

self.W2= self.W2+ self.z2.T.dot(self.o\_delta)

def train(self,X,Y):

o=self.forward(X)

self.backward(X,Y,o)

obj=NM()

for i in range(2000):

print("input"+str(X))

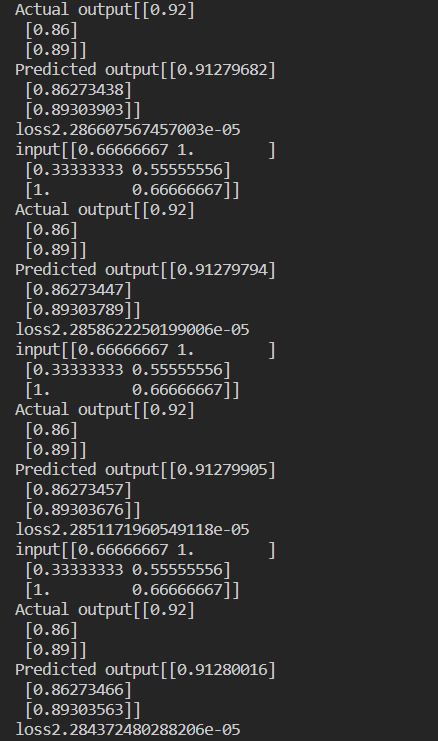
print("Actual output"+str(Y))

print("Predicted output"+str(obj.forward(X)))

print("loss"+str(np.mean(np.square(Y-obj.forward(X)))))

obj.train(X,Y)

Output:



Practical-5

1. Hopfield network

Code: