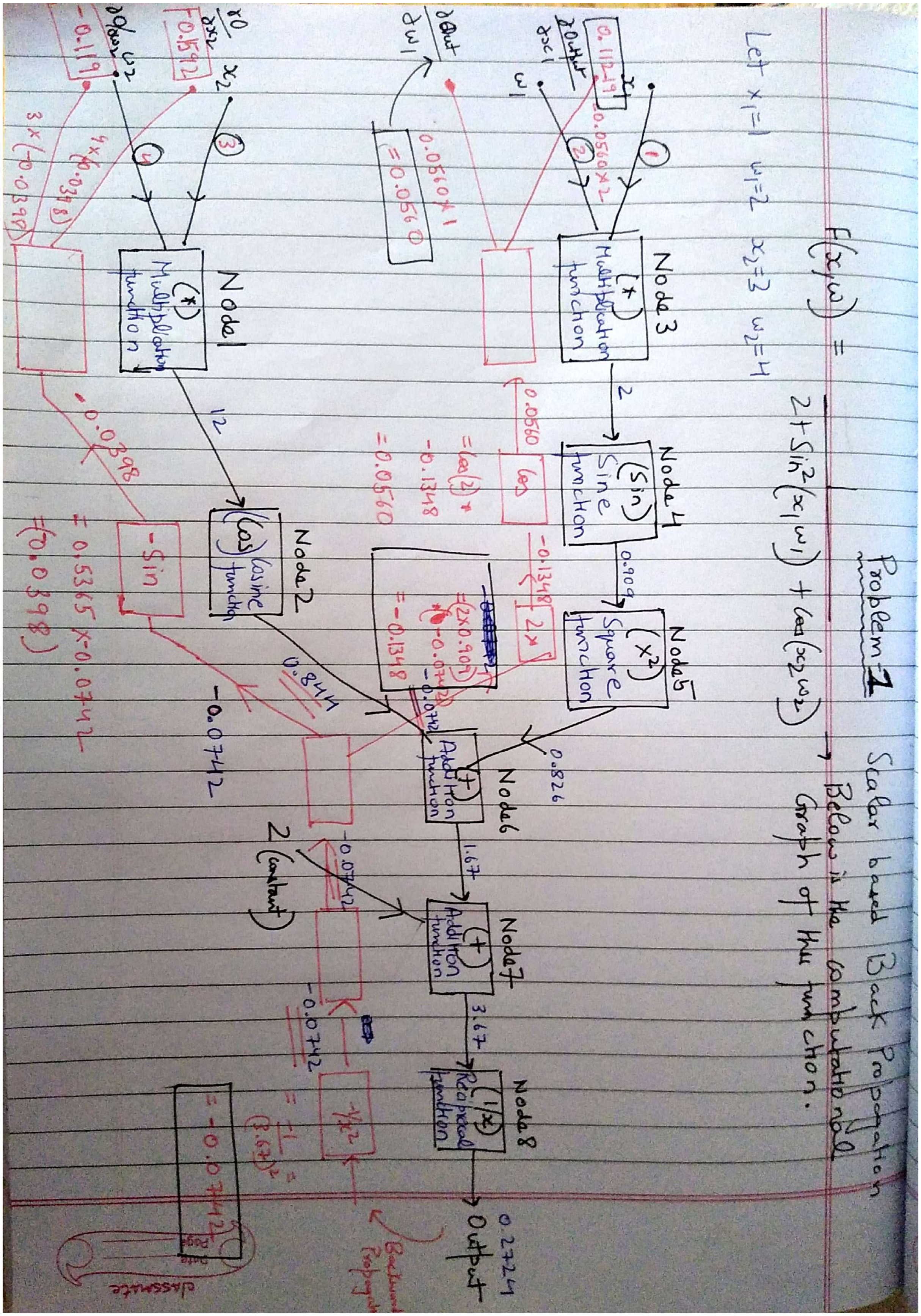
Homework 2

Problem 1 (Practice of scalar based backpropagation).

You are required to calculate the gradients of f**x**,**w** with respect to xi and wi.

(a) Use computational graph for calculation



Solution:-

(b) Based on(a), write a program to implement the computational graph and verify your answer in (a). Note: You are fre e to choose the inputs of your computational graph.

Solution:-

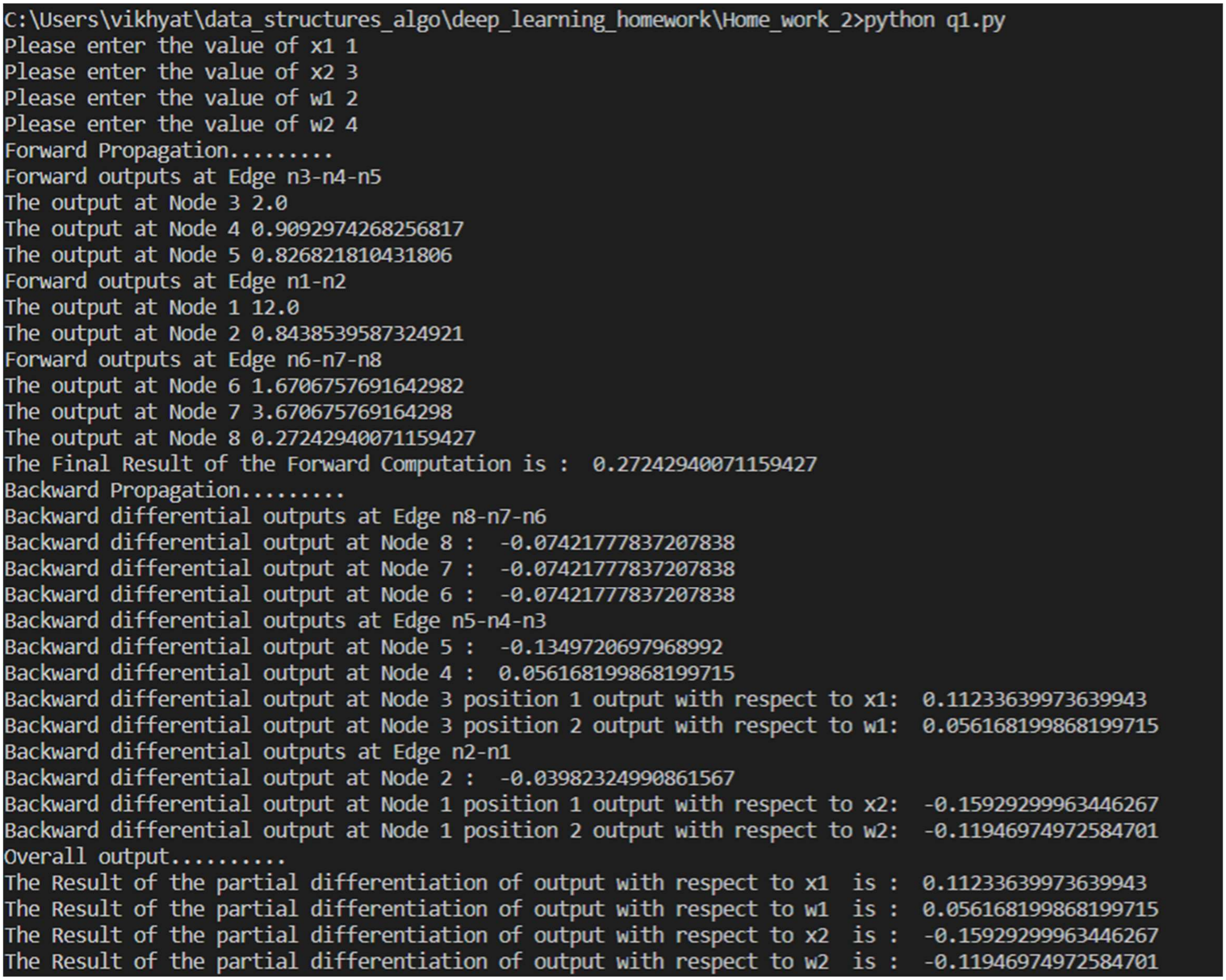
```
import math
import numpy as np
import dask

#Mode is a variable whose value can be 0 or 1 : 1 for normal operation and 0 for differential operation

#inputs to the graph
x1=input("Please enter the value of x1 ")
x2=input("Please enter the value of x2 ")
w1=input("Please enter the value of w1 ")
w2=input("Please enter the value of w2 ")


#Forward Propagation in the graph


# Functions to be used
#Node 1 Multiplication
def node1(a,b,mode,pos,b_input):
    if mode==1:
        return a*b
    elif pos==1:
        return (b*b_input)
    else:
        return (a*b_input)


#Node 2 Cosine
def node2(a,mode,b_input):

    if mode==1:
        return (math.cos(a))
    else:
        return (-(math.sin(a))*b_input)

#Node 3 Multiplication2
def node3(a,b,mode,pos,b_input):
    if mode==1:
        return a*b
    elif pos==1:
        return (b*b_input)
    else:
        return (a*b_input)

#Node 4 sine function
def node4(a,mode,b_input):

    if mode==1:
        return math.sin(a)
    else:
        return ((math.cos(a))*b_input)

#Node 5 square function
def node5(a,mode,b_input):
    if mode==1:
        return a*a
    else:
        return 2*a*b_input
```

```
#Node 6 addition1 function
def node6(a,b,mode,b_input):
    if mode==1:
        return a+b
    else:
        return (1*b_input)

#Node7 addition2 function
def node7(a,b,mode,b_input):
    if mode==1:
        return a+b
    else:
        return (1*b_input)

#Node 8 reciprocal function
def node8(a,mode,b_input):
    if mode==1:
        return (1/a)
    else:
        return(-1/(a*a))


#Forward Computation
input1_n3=float(x1)
input2_n3=float(w1)

input1_n1=float(x2)
input2_n1=float(w2)

print("Forward Propagation.........")

#Edge n3-n4-n5
print("Forward outputs at Edge n3-n4-n5")

output_n3=node3(input1_n3,input2_n3,1,0,0)
print("The output at Node 3",output_n3)

input_n4=output_n3
output_n4=node4(input_n4,1,0)
print("The output at Node 4",output_n4)


input_n5=output_n4
output_n5=node5(input_n5,1,0)
print("The output at Node 5",output_n5)

#Edge n1-n2
print("Forward outputs at Edge n1-n2")
output_n1=node1(input1_n1,input2_n1,1,0,0)
print("The output at Node 1",output_n1)

input_n2=output_n1
output_n2=node2(input_n2,1,0)
print("The output at Node 2",output_n2)

print("Forward outputs at Edge n6-n7-n8")
input1_n6=output_n5
input2_n6=output_n2
output_n6=node6(input1_n6,input2_n6,1,0)
print("The output at Node 6",output_n6)

input_n7=output_n6
output_n7=node7(2,input_n7,1,0)
print("The output at Node 7",output_n7)

input_n8=output_n7
output_n8=node8(input_n8,1,0)
```

```
print("The output at Node 8",output_n8)

print("The Final Result of the Forward Computation is : ",output_n8 )

#Backward Propagation
print("Backward Propagation.........")
print("Backward differential outputs at Edge n8-n7-n6")

b_output_n8=node8(input_n8,0,1)
print("Backward differential output at Node 8 : ",b_output_n8)

b_output_n7=node7(2,input_n7,0,b_output_n8)
print("Backward differential output at Node 7 : ",b_output_n7)

b_output_n6=node6(input1_n6,input2_n6,0,b_output_n7)
print("Backward differential output at Node 6 : ",b_output_n6)

print("Backward differential outputs at Edge n5-n4-n3")
b_output_n5=node5(input_n5,0,b_output_n6)
print("Backward differential output at Node 5 : ",b_output_n5)

b_output_n4=node4(input_n4,0,b_output_n5)
print("Backward differential output at Node 4 : ",b_output_n4)

b_output_n3_x1=node3(input1_n3,input2_n3,0,1,b_output_n4)
b_output_n3_w1=node3(input1_n3,input2_n3,0,2,b_output_n4)
print("Backward differential output at Node 3 position 1 output with respect to x1: ",b_output_n3_x1)
print("Backward differential output at Node 3 position 2 output with respect to w1: ",b_output_n3_w1)


print("Backward differential outputs at Edge n2-n1")
b_output_n2=node2(input_n2,0,b_output_n6)
print("Backward differential output at Node 2 : ",b_output_n2)

b_output_n1_x2=node1(input1_n1,input2_n1,0,1,b_output_n2)
b_output_n1_w2=node1(input1_n1,input2_n1,0,2,b_output_n2)
print("Backward differential output at Node 1 position 1 output with respect to x2: ",b_output_n1_x2)
print("Backward differential output at Node 1 position 2 output with respect to w2: ",b_output_n1_w2)

print("Overall output..........")

print("The Result of the partial differentiation of output with respect to x1  is : ",b_output_n3_x1)

print("The Result of the partial differentiation of output with respect to w1  is : ",b_output_n3_w1)

print("The Result of the partial differentiation of output with respect to x2  is : ",b_output_n1_x2)

print("The Result of the partial differentiation of output with respect to w2  is : ",b_output_n1_w2)
```

Output:

Description of the program is at the end.



```
C:\Users\vikhyat\data_structures_algo\deep_learning_homework\Home_work_2>python q1.py
Please enter the value of x1 1
Please enter the value of x2 3
Please enter the value of w1 2
Please enter the value of w2 4
Forward Propagation.........
Forward outputs at Edge n3-n4-n5
The output at Node 3 2.0
The output at Node 4 0.9092974268256817
The output at Node 5 0.826821810431806
Forward outputs at Edge n1-n2
The output at Node 1 12.0
The output at Node 2 0.8438539587324921
Forward outputs at Edge n6-n7-n8
The output at Node 6 1.6706757691642982
The output at Node 7 3.670675769164298
The output at Node 8 0.27242940071159427
The Final Result of the Forward Computation is :  0.27242940071159427
Backward Propagation.........
Backward differential outputs at Edge n8-n7-n6
Backward differential output at Node 8 :  -0.07421777837207838
Backward differential output at Node 7 :  -0.07421777837207838
Backward differential output at Node 6 :  -0.07421777837207838
Backward differential outputs at Edge n5-n4-n3
Backward differential output at Node 5 :  -0.1349720697968992
Backward differential output at Node 4 :  0.056168199868199715
Backward differential output at Node 3 position 1 output with respect to x1:  0.11233639973639943
Backward differential output at Node 3 position 2 output with respect to w1:  0.056168199868199715
Backward differential outputs at Edge n2-n1
Backward differential output at Node 2 :  -0.03982324990861567
Backward differential output at Node 1 position 1 output with respect to x2:  -0.15929299963446267
Backward differential output at Node 1 position 2 output with respect to w2:  -0.11946974972584701
Overall output..........
The Result of the partial differentiation of output with respect to x1  is :  0.11233639973639943
The Result of the partial differentiation of output with respect to w1  is :  0.056168199868199715
The Result of the partial differentiation of output with respect to x2  is :  -0.15929299963446267
The Result of the partial differentiation of output with respect to w2  is :  -0.11946974972584701
```

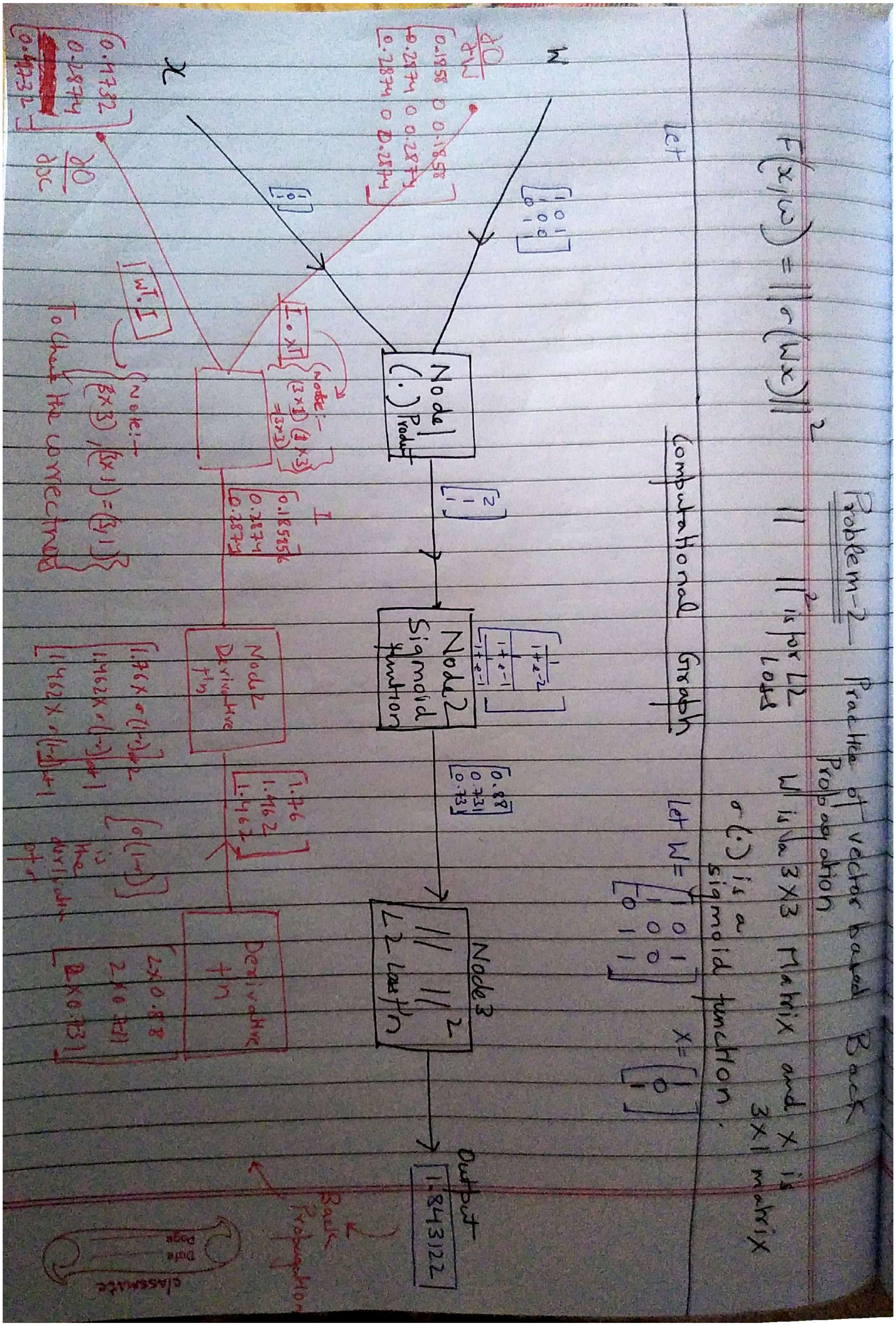
Problem 2 (Practice of vector based backpropagation).

You are required to calculate the gradients of $fx,w\|\sigma\mathbf{Wx}\|$ with respect to xi and Wi,j. Here $\|\cdot\|$ is the calculation of L2 loss, W is 3-by-3 matrix

and x is 3 by 1 vector, and $\sigma\cdot$ is sigmoid function that performs element-wise sigmoid operation.

(a)Use computational graph for calculation

Solution:-



Problem-2 Practice of vector based Back Propagation

$F(x,\omega) = ||\sigma(Wx)||^2$ $|| \quad ||^2$ is for L2 Loss

W is a 3X3 Matrix and x is 3X1 matrix

$\sigma(\cdot)$ is a sigmoid function.

Computational Graph

Let $W = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$ $X = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$

let
W
x
$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$
$\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$
Node1 (.) Product
$\begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$
$\begin{bmatrix} \frac{1}{1+e^{-2}} \\ \frac{1}{1+e^{-1}} \\ \frac{1}{1+e^{-1}} \end{bmatrix}$
Node2 Sigmoid function
$\begin{bmatrix} 0.88 \\ 0.731 \\ 0.73 \end{bmatrix}$
Node3 $|| \quad ||^2$ L2 Loss f'n
Output 1.843122
Back Propagation
Derivative f'n
$\begin{bmatrix} 2 \times 0.88 \\ 2 \times 0.731 \\ 2 \times 0.731 \end{bmatrix}$
$\begin{bmatrix} 1.76 \\ 1.462 \\ 1.462 \end{bmatrix}$
Node2 Derivative f'n
$\begin{bmatrix} 1.76 \times \sigma(1-\sigma) \\ 1.462 \times \sigma(1-\sigma) \\ 1.462 \times \sigma(1-\sigma) \end{bmatrix}$
$[\sigma(1-\sigma)]$ is the derivative of σ
I
$\begin{bmatrix} 0.185856 \\ 0.2874 \\ 0.2874 \end{bmatrix}$
$I \cdot x^T$
Note:- $(3 \times 1)(1 \times 3) = (3 \times 3)$
$W^T \cdot I$
Note:- $(3 \times 3), (3 \times 1) = (3, 1)$
To Check the correctness
$\frac{\partial O}{\partial W}$
$\begin{bmatrix} 0.1858 & 0 & 0.1858 \\ 0.2874 & 0 & 0.2874 \\ 0.2874 & 0 & 0.2874 \end{bmatrix}$
$\frac{\partial O}{\partial x}$
$\begin{bmatrix} 0.4732 \\ 0.2874 \\ 0.4732 \end{bmatrix}$
classmate
Date
Page

(b)Based on(a), write a program to implement the computational graph and verify your answer in (a). You can use vectorized approach to simply your codes. Note: You are free to choose the inputs of your computational graph.

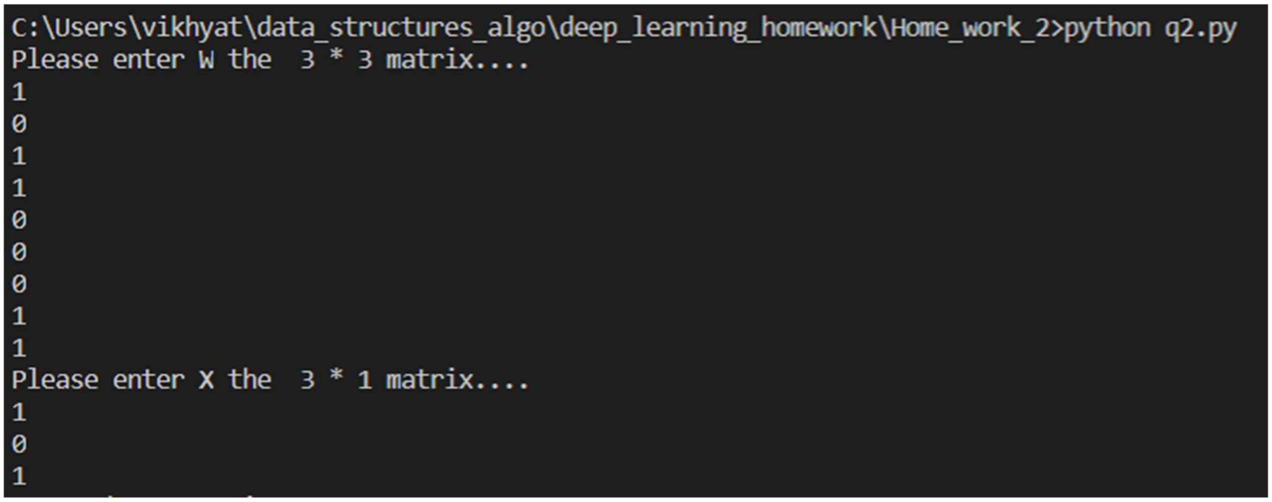
Solution:-

```python
import math
import numpy as np

w=[0]*9
x=[0]*3
print("Please enter W the  3 * 3 matrix....")
for i in range(9):
    w[i]=float(input())


print("Please enter X the  3 * 1 matrix....")
for m in range(3):
    x[m]=float(input())


#Mode is a variable whose value can be 0 or 1 : 1 for normal operation and 0 for differential operation

#Node 1 Matrix Multiplication or Dot Product
def node1(a,b,mode,pos,b_input):
    if mode==1:
       return np.dot(a,b)
    elif pos==1:
       return np.dot(b_input,np.transpose(b))
    else:
        return np.dot(np.transpose(a),b_input)

#Node 2 Sigmoid Function
def node2(a,mode,b_input):
    if mode==1:
       return (1/(1 + np.exp(-a)))
    else:
       return np.multiply((np.exp(-a)/np.square(1 + np.exp(-a))),b_input)

#Node 3 Summation Function
def node3(a,mode,b_input):
    if mode==1:
        return(np.sum(np.square(a)))
    else:
        return(a*2*b_input)

#Forward Propagation
print("Forward propagation........","\n")
input1_n1=np.array(w).reshape([3,3])
input2_n1=np.array(x).reshape([3,1])
#Computation of Node 1
output_n1=node1(input1_n1,input2_n1,1,0,0)
print("The forward path output at Node 1 : ","\n",output_n1)

#Computation of Node 2
input_n2=output_n1
output_n2=node2(input_n2,1,0)
print("The forward path output at Node 2 : ","\n",output_n2)

#Computation of Node 3
input_n3=output_n2
output_n3=node3(input_n3,1,0)
print("The forward path output at Node 3 : ","\n",output_n3)

print("The Result of the Forward Computation is : ","\n",output_n3)

#Backward Propagation
print("Backward Propagation.........")
#Node 3
b_output_n3=node3(input_n3,0,1)
print("The back ward differentiation output at Node 3 is : ","\n",b_output_n3)

#Node 2
b_output_n2=node2(input_n2,0,b_output_n3)
```

```
print("The back ward differentiation output at Node 2 is : ","\n",b_output_n2)
#Node 1

#Partial Differentiation with respect to w
b_output_n1_w=node1(input1_n1,input2_n1,0,1,b_output_n2)
print("The back ward differentiation output at Node 1 at position 1 result partial diff of output w.r.t W is : ","\n",b_output_n1_w)

#Partial Differentiation with respect to x
b_output_n1_x=node1(input1_n1,input2_n1,0,2,b_output_n2)
print("The back ward differentiation output at Node 1 at position 2 result partial diff of output w.r.t x is : ","\n",b_output_n1_x)
```

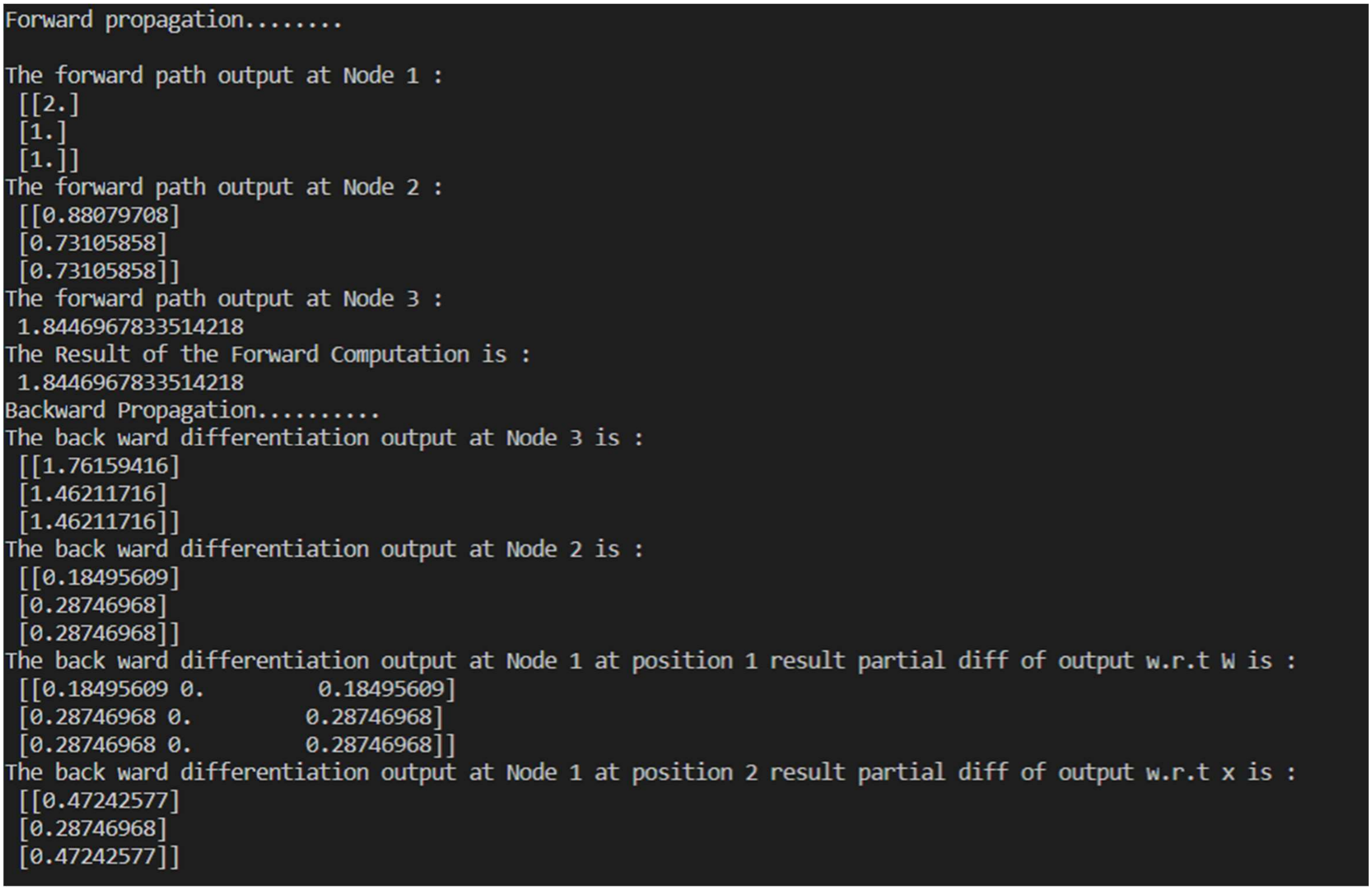
Output:

Description of the program is at the end.



```
C:\Users\vikhyat\data_structures_algo\deep_learning_homework\Home_work_2>python q2.py
Please enter W the  3 * 3 matrix....
1
0
1
1
0
0
0
1
1
Please enter X the  3 * 1 matrix....
1
0
1
```

Note that the row wise inputs of the matrix need to be done.



```
Forward propagation........

The forward path output at Node 1 :
 [[2.]
 [1.]
 [1.]]
The forward path output at Node 2 :
 [[0.88079708]
 [0.73105858]
 [0.73105858]]
The forward path output at Node 3 :
 1.8446967833514218
The Result of the Forward Computation is :
 1.8446967833514218
Backward Propagation..........
The back ward differentiation output at Node 3 is :
 [[1.76159416]
 [1.46211716]
 [1.46211716]]
The back ward differentiation output at Node 2 is :
 [[0.18495609]
 [0.28746968]
 [0.28746968]]
The back ward differentiation output at Node 1 at position 1 result partial diff of output w.r.t W is :
 [[0.18495609 0.         0.18495609]
 [0.28746968 0.         0.28746968]
 [0.28746968 0.         0.28746968]]
The back ward differentiation output at Node 1 at position 2 result partial diff of output w.r.t x is :
 [[0.47242577]
 [0.28746968]
 [0.47242577]]
```

Description

In the Rough Calculation Diagrams , it has been clearly shown , all the nodes with their numbers and their corresponding functions and their inputs and outputs flow in both forward and the backward propagation.

In the program what has been done is that , corresponding to our diagrams , the nodes are created as a programming function (neurons) , which they work in actual also(i.e. as some mathematical function) in case of neural networks also.

In our program , the same node functions are being used both in case of forward computation path and in the backward propagation also and just we need to change the mode value passed to the function . So in our program , mode=1 for forward propagation and mode=0 for backward propagation.

In each node function ,

a. Forward inputs and Backward inputs can be easily seen from the arguments' names of these functions as (a, b --- for forward inputs and b_inputs for backward inputs) and
b. Some other parameters like **pos** is provided in case of multiplication which is to mention that output is requested w.r.t to which position of the input .Like in case of backward propagation , in case of multiplication , there are two inputs so there are two outputs with respect to each position of inputs in case of backward propagation.

[Note that caching of the results by the nodes(like w , b which are cached in neural networks implementation) during forward paths and then being used in the backward path is not being shown explicitly as it is not a proper neural network and this program is just for getting real practice of how backpropagation works , the inputs at every stage of forward path are stored in the variables and are being passed during the backward propagation also , so as to calculate the result based on them and the previous activation or the results from the previous back node]

Note further the results from the hand calculation and the outputs have been verified for various inputs and as asked the case of one particular sample input has been presented here.

Further all the calculation images , py programs , output screenshots have been attached herewith the homework submission.

Py files should be used for execution or just copy paste of the above code in editor should also work but may require slight work.