Back Propagation for n bit data

Generating n bit of data

In [1]:

```
n = int(input('Enter Number of bits : '))
count = 0
i = n
string = 'bit_'
total_number = 2 ** n
```

In [2]:

```
value = list()
dictionary = dict()

while i >= 1 :
    key = string + str(i)
    d = 2 ** count

    while len(value) != total_number:
        for j in range(d):
            value.append(0)
        for j in range(d):
            value.append(1)

    dictionary[key] = value
    value = list()
    count = count + 1
    i = i - 1
```

In [3]:

```
dictionary
```

Out[3]:

```
{'bit_3': [0, 1, 0, 1, 0, 1, 0, 1],
 'bit_2': [0, 0, 1, 1, 0, 0, 1, 1],
 'bit_1': [0, 0, 0, 0, 1, 1, 1, 1]}
```

In [4]:

```
list(dictionary.items())
```

Out[4]:

```
[('bit_3', [0, 1, 0, 1, 0, 1, 0, 1]),
 ('bit_2', [0, 0, 1, 1, 0, 0, 1, 1]),
 ('bit_1', [0, 0, 0, 0, 1, 1, 1, 1])]
```

In [5]:

```
l = list(dictionary.items())
l
```

Out[5]:

```
[('bit_3', [0, 1, 0, 1, 0, 1, 0, 1]),
 ('bit_2', [0, 0, 1, 1, 0, 0, 1, 1]),
 ('bit_1', [0, 0, 0, 0, 1, 1, 1, 1])]
```

In [6]:

```
reversed_dictionary = dict()
```

```
i = n-1
while i >= 0:
    reversed_dictionary[l[i][0]] = l[i][1]
    i = i - 1
reversed_dictionary
```

Out[6]:

```
{'bit_1': [0, 0, 0, 0, 1, 1, 1, 1],
 'bit_2': [0, 0, 1, 1, 0, 0, 1, 1],
 'bit_3': [0, 1, 0, 1, 0, 1, 0, 1]}
```

In [7]:

```
dictionary = reversed_dictionary
dictionary
```

Out[7]:

```
{'bit_1': [0, 0, 0, 0, 1, 1, 1, 1],
 'bit_2': [0, 0, 1, 1, 0, 0, 1, 1],
 'bit_3': [0, 1, 0, 1, 0, 1, 0, 1]}
```

In [8]:

```
output = dictionary['bit_1']
output
```

Out[8]:

```
[0, 0, 0, 0, 1, 1, 1, 1]
```

In [9]:

```
import pandas as pd

df = pd.DataFrame(data=dictionary)
df
```

Out[9]:

	bit_1	bit_2	bit_3
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

In [10]:

```
df['Output'] = output
df
```

Out[10]:

	bit_1	bit_2	bit_3	Output
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1

	bit_1	bit_2	bit_3	Output
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

In [11]:

```
df = df.drop('Output',axis=1)
df
```

Out[11]:

	bit_1	bit_2	bit_3
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Train Test Split

In [12]:

```
train_percentage = 60
test_percentage = 100 - train_percentage

print('Train Percentage :',train_percentage)
print('Test Percentage :',test_percentage)
```

```
Train Percentage : 60
Test Percentage : 40
```

In [13]:

```
import math

no_of_train_data = math.ceil(( total_number * train_percentage ) / 100)
no_of_test_data = total_number - no_of_train_data

print('No of Train Data :',no_of_train_data)
print('No of Test Data :',no_of_test_data)
```

```
No of Train Data : 5
No of Test Data : 3
```

Inititalizing Wij , bj , Wjk and bk with random values

In [14]:

```
# n is the number of nodes in input layer and hidden layer
# m is the number of nodes in output layer

unique = dict()

for i in output:
    if i not in unique:
        unique[i] = 1
    else:
```

```
        unique[i] = unique[i] + 1

print('Total Class in Output :',len(unique))
```

```
Total Class in Output : 2
```

In [15]:

```
import numpy as np
import math

# n will be as it is
m = math.ceil(np.log2(len(unique)))

print('Number of nodes in input layer :',n)
print('Number of nodes in hidden layer :',n)
print('Number of nodes in output layer :',m)
```

```
Number of nodes in input layer : 3
Number of nodes in hidden layer : 3
Number of nodes in output layer : 1
```

In [16]:

```
np.random.seed(113)
Wij = np.random.rand(n,n)
bj = np.random.rand(n)
Wjk = np.random.rand(n,m)
bk = np.random.rand(m)
```

In [17]:

```
print('Weights from i to j :\n',Wij)
print('\nBias to j :\n',bj)
print('\nWeights from j to k :\n',Wjk)
print('\nBias to k :\n',bk)
```

```
Weights from i to j :
 [[0.85198549 0.0739036  0.89493176]
 [0.43649355 0.12767773 0.57585787]
 [0.84047092 0.43512055 0.69591056]]

Bias to j :
 [0.6846381  0.70064837 0.77969426]

Weights from j to k :
 [[0.64274937]
 [0.96102617]
 [0.10846489]]

Bias to k :
 [0.79610634]
```

In [18]:

```
Wij[0,0]
```

Out[18]:

```
0.8519854927300882
```

Initializing Oi , netj , Oj and netk with 0

In [19]:

```
Oi = np.zeros(n)

netj = np.zeros(n)
activj = np.zeros(n)
Oj = np.zeros(n)
```

```
netk = np.zeros(m)
activk = np.zeros(m)
Ok = np.zeros(m)
```

In [20]:

```
# learning rate
learning_rate = 0.5
```

initializing delta_Wjk , delta_bk , delta_Wij , delta_bj with 0

In [21]:

```
delta_Wjk = np.zeros((n,m))
delta_bk = np.zeros(m)
delta_Wij = np.zeros((n,n))
delta_bj = np.zeros(n)
```

Forward Propagation and Backward Propagation

In [22]:

```
df
```

Out[22]:

	bit_1	bit_2	bit_3
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

In [23]:

```
output
```

Out[23]:

```
[0, 0, 0, 0, 1, 1, 1, 1]
```

In [24]:

```
df[df.columns[0]][4]
```

Out[24]:

```
1
```

In [25]:

```
np.exp(1)
```

Out[25]:

```
2.718281828459045
```

In [26]:

```
n
```

Out[26]:

3

In [27]:

```
l = list()
count = 0
while count != no_of_train_data :
    for row in range(no_of_train_data):
        # forward propagation starts

        for column in range(len(df.columns)):
            Oi[column] = df.iloc[row,column]

        for i in range(n):
            for j in range(n):
                netj[j] = netj[j] + Oi[i] * Wij[i,j]


        for j in range(n):
            activj[j] = netj[j] + bj[j]

        for j in range(n):
            Oj[j] = 1 / (1 + np.exp(-1*activj[j]))

        for j in range(n):
            for k in range(m):
                netk[k] = netk[k] + Oj[j] * Wjk[j,k]

        for k in range(m):
            activk[k] = netk[k] + bk[k]

        for k in range(m):
            Ok[k] = 1 / (1 + np.exp(-1*activk[k]))

        delta = output[row] - Ok[0] # not generalized
        error = 0.5 * ( delta ** 2)
        if error <= 0.01:
            l.append(Ok[0])
            count = count + 1
            netj = np.zeros(n)
            netk = np.zeros(m)
            continue


        # Backpropagation starts

        for j in range(n):
            for k in range(m):
                delta_Wjk[j,k] = learning_rate * delta * Oj[j] * Ok[k] * (1 - Ok[k])

        for j in range(n):
            for k in range(m):
                Wjk[j,k] = Wjk[j,k] + delta_Wjk[j,k]

        for k in range(m):
            delta_bk[k] = learning_rate * delta * Ok[k] * (1 - Ok[k])

        for k in range(m):
            bk[k] = bk[k] + delta_bk[k]

        summation = 0
        for j in range(n):
            summation = summation + Wjk[j,0] * delta

        for i in range(n):
            for j in range(n):
                delta_Wij[i,j] = learning_rate * Oi[i] * Oj[j] * (1-Oj[j]) * summation
```

```
        for i in range(n):
            for j in range(n):
                Wij[i,j] = Wij[i,j] + delta_Wij[i,j]

        for j in range(n):
            delta_bj[j] = learning_rate * Oj[j] * (1 - Oj[j]) * summation

        for j in range(n):
            bj[j] = bj[j] + delta_bj[j]

        netj = np.zeros(n)
        netk = np.zeros(m)
        count = 0
        l = list()
        break
```

In [28]:

```
count
```

Out[28]:

```
10
```

In [29]:

```
l
```

Out[29]:

```
[0.13860830144318903,
 0.11418434918960044,
 0.10888551745357877,
 0.09765934984803684,
 0.11732680057811581,
 0.10206618514048248,
 0.09910573770542183,
 0.09194846080211491,
 0.8683331736813442,
 0.8601205469920238]
```

In [30]:

```
right = 0
wrong = 0
l = list()
for row in range(no_of_train_data,total_number):
    # forward propagation starts

    for column in range(len(df.columns)):
        Oi[column] = df[df.columns[column]][row]

    for i in range(n):
        for j in range(n):
            netj[j] = netj[j] + Oi[i] * Wij[i,j]


    for j in range(n):
        activj[j] = netj[j] + bj[j]

    for j in range(n):
        Oj[j] = 1 / (1 + np.exp(-1*activj[j]))

    for j in range(n):
        for k in range(m):
            netk[k] = netk[k] + Oj[j] * Wjk[j,k]

    for k in range(m):
        activk[k] = netk[k] + bk[k]

    for k in range(m):
        Ok[k] = 1 / (1 + np.exp(-1*activk[k]))
```

```python
    delta = output[row] - Ok[0] # not generalized
    error = 0.5 * ( delta ** 2)
    l.append(error)

    if error <= 0.01:
        right = right + 1
    else:
        wrong = wrong + 1

accuracy = ( right * 100 ) / no_of_test_data

print(l)
print("No of Test Data :",no_of_test_data)
print("Right :",right)
print("Wrong :",wrong)
print("Accuracy :",accuracy)
```

```
[0.010175588270903613, 0.010216402747925865, 0.010216622044655751, 0.01021662425910416, 0
.010216624284780242, 0.010216624285336566]
No of Test Data : 6
Right : 0
Wrong : 6
Accuracy : 0.0
```