

```
In [48]: 1 # Importing Libraries
        2 import numpy as np
        3 import pandas as pd
```

```
In [49]: 1 #reading .csv file and storing it in "dataset" dataframe
        2 dataset=pd.read_csv("/home/ankit/Desktop/mnist_train.csv")
```

```
In [50]: 1 print(dataset.head()) # printing the whole dataset
```

	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	...	28x19	28x20	\
0	5	0	0	0	0	0	0	0	0	0	...	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	
2	4	0	0	0	0	0	0	0	0	0	...	0	0	
3	1	0	0	0	0	0	0	0	0	0	...	0	0	
4	9	0	0	0	0	0	0	0	0	0	...	0	0	

	28x21	28x22	28x23	28x24	28x25	28x26	28x27	28x28
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0

[5 rows x 785 columns]

```
In [51]: 1 print(dataset.shape) #28*28 columns for 28*28 pixel images and one column
        (60000, 785)
```

```
In [52]: 1 print(dataset['label'].unique()) # printing the unique classes of dataset
        [5 0 4 1 9 2 3 6 7 8]
```

```
In [53]: 1 data1 = dataset.to_numpy() # storing dataset as numpy array
```

```
In [54]: 1 list0=[]
        2 list1=[]
        3 list2=[]
        4 list3=[]
        5 list4=[]
        6 list5=[]
        7 list6=[]
        8 list7=[]
        9 list8=[]
       10 list9=[]
       11 count0=0
       12 count1=0
       13 count2=0
       14 count3=0
       15 count4=0
       16 count5=0
       17 count6=0
       18 count7=0
       19 count8=0
       20 count9=0
       21
       22 for i in range(60000):
       23     if data1[i][0] == 0:
       24         if count0 < 10:
       25             list0.append(i)
       26             count0=count0+1
       27
       28     if data1[i][0] == 1:
       29         if count1 < 10:
       30             list1.append(i)
```

```
31         count1=count1+1
32
33     if data1[i][0] == 2:
34         if count2 < 10:
35             list2.append(i)
36             count2=count2+1
37
38     if data1[i][0] == 3:
39         if count3 < 10:
40             list3.append(i)
41             count3=count3+1
42
43     if data1[i][0] == 4:
44         if count4 < 10:
45             list4.append(i)
46             count4=count4+1
47
48     if data1[i][0] == 5:
49         if count5 < 10:
50             list5.append(i)
51             count5=count5+1
52
53     if data1[i][0] == 6:
54         if count6 < 10:
55             list6.append(i)
56             count6=count6+1
57
58     if data1[i][0] == 7:
59         if count7 < 10:
60             list7.append(i)
61             count7=count7+1
62
63     if data1[i][0] == 8:
64         if count8 < 10:
65             list8.append(i)
66             count8=count8+1
67
68     if data1[i][0] == 9:
69         if count9 < 10:
70             list9.append(i)
71             count9=count9+1
72     if count0 == 11 and count1 == 11 and count2 == 11 and count3 == 11
73         break
74
```

```
In [55]: 1 print(len(list0))
2 print(len(list1))
3 print(len(list2))
4 print(len(list3))
5 print(len(list4))
6 print(len(list5))
7 print(len(list6))
8 print(len(list7))
9 print(len(list8))
10 print(len(list9))
11
```

```
10
10
10
10
10
10
10
10
10
10
```

```
In [56]: 1 #for dk(p), if class label belongs to class 1, we make 1st value as 1 and 0
2 #others
3
4 labels=[]
5 for i in range(100):
6     if data1[i][1]==0:
7         labels.append([1,0,0,0,0,0,0,0,0,0])
8     if data1[i][1]==1:
9         labels.append([0,1,0,0,0,0,0,0,0,0])
10    if data1[i][1]==2:
11        labels.append([0,0,1,0,0,0,0,0,0,0])
12    if data1[i][1]==3:
13        labels.append([0,0,0,1,0,0,0,0,0,0])
14    if data1[i][1]==4:
15        labels.append([0,0,0,0,1,0,0,0,0,0])
16    if data1[i][1]==5:
17        labels.append([0,0,0,0,0,1,0,0,0,0])
18    if data1[i][1]==6:
19        labels.append([0,0,0,0,0,0,1,0,0,0])
20    if data1[i][1]==7:
21        labels.append([0,0,0,0,0,0,0,1,0,0])
22    if data1[i][1]==8:
23        labels.append([0,0,0,0,0,0,0,0,1,0])
24    if data1[i][1]==9:
25        labels.append([0,0,0,0,0,0,0,0,0,1])
```

```
In [57]: 1 print(labels)
```

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

```
In [58]: 1 data=[]
2         for i in list0:
3             temp=[]
4             temp.append(1) #appending bias
5             for j in range(1,785):
6                 temp.append(data1[i][j])
7             data.append(temp)
8
9         for i in list1:
10            temp=[]
11            temp.append(1) #appending bias
12            for j in range(1,785):
13                temp.append(data1[i][j])
14            data.append(temp)
15        for i in list2:
16            temp=[]
17            temp.append(1) #appending bias
18            for j in range(1,785):
19                temp.append(data1[i][j])
20            data.append(temp)
21        for i in list3:
22            temp=[]
23            temp.append(1) #appending bias
24            for j in range(1,785):
25                temp.append(data1[i][j])
26            data.append(temp)
27        for i in list4:
28            temp=[]
29            temp.append(1) #appending bias
30            for j in range(1,785):
31                temp.append(data1[i][j])
32            data.append(temp)
33        for i in list5:
34            temp=[]
35            temp.append(1) #appending bias
36            for j in range(1,785):
37                temp.append(data1[i][j])
38            data.append(temp)
39        for i in list6:
40            temp=[]
41            temp.append(1) #appending bias
42            for j in range(1,785):
43                temp.append(data1[i][j])
44            data.append(temp)
45        for i in list7:
46            temp=[]
47            temp.append(1) #appending bias
48            for j in range(1,785):
49                temp.append(data1[i][j])
50            data.append(temp)
51        for i in list8:
52            temp=[]
53            temp.append(1) #appending bias
54            for j in range(1,785):
55                temp.append(data1[i][j])
56            data.append(temp)
57        for i in list9:
58            temp=[]
59            temp.append(1) #appending bias
60            for j in range(1,785):
61                temp.append(data1[i][j])
62            data.append(temp)
```

```
In [59]: 1 print(data)
```

## Implementing Forward Propagation MLP for Mnist dataset

5 of 11 07/10/20, 2:49 pm

input weight vector for output layer:

```
[[-0.7957419044077394, 0.6380130458601733, 1.1004641725461934, 1.22124649004
88498, 1.4204735980127532, 1.9283977127593492, 0.3827047412014703, -0.23996653
407298857, -0.18289352038486906, -0.6145250454605087, 0.8096783558931424, -1.0
161110058835847, -0.07537107860191218, -0.9674708109159177, 0.516781288744741
2 -0.1400471181550136 1 2663602306027887 -0.5623253408760216 -0.5010625355
```

In [61]: `1 len(data[0])`

Out[61]: 785

```
In [63]: 1 def forward_prop(p):
2         forward_prop_output=[]
3
4         # Input on input layer -1 (hidden layer ) by using McCullouch Pit mode
5
6         u1=[]
7         for j in range(784):
8             sum=0
9             for i in range(785):
10                sum += weight2[j][i]*data[p][i]
11            u1.append(sum)
12            #print("Input on layer 1 (hidden layer) : ",u)
13
14            # For output on layer 1, by using sigmoid function,
15
16            v1=[]
17            v1.append(1) # appending bias=1
18
19            for i in range(784):
20                sig=1/(1+ np.exp(-u1[i]))
21                v1.append(sig)
22            #print("output on layer 1 : ",v)
23
24
25            # For input on layer 2 by using McCullouch Pit model,
26
27            u2=[]
28            for j in range(10):
29                sum=0
30                for i in range(785):
31                    sum += weight1[j][i]*v1[i]
32                u2.append(sum)
33                #print("input of layer 2",u)
34
35
36            # For output on layer 2(output layer), by using sigmoid function,
37
38            v2=[]
39
40            for i in range(10):
41                sig=1/(1+ np.exp(-u2[i]))
42                v2.append(sig)
43            #print("output on layer 2 (output layer) : ",v)
44            forward_prop_output.append(v2)
45            yk=forward_prop_output
46
47            return vk
```

## Implementing Backpropagation MLP for Mnist dataset

```
In [86]: 1 def back_prop(p):
2         # computing delta w_ji(0) (learning rule) #weight2
3
4         learning_rate = 0.5 #assuming
5
```

```

6     forward_prop_output=[]
7
8     # Input on input layer -1 (hidden layer ) by using McCullouch Pit mode
9
10    u1=[]
11    for j in range(784):
12        sum=0
13        for i in range(785):
14            sum += weight2[j][i]*data[p][i]
15        u1.append(sum)
16        #print("Input on layer 1 (hidden layer) : ",u)
17
18    # For output on layer 1, by using sigmoid function,
19
20    v1=[]
21    v1.append(1) # appending bias=1
22
23    for i in range(784):
24        sig=1/(1+ np.exp(-u1[i]))
25        v1.append(sig)
26    #print("output on layer 1 : ",v)
27
28
29    # For input on layer 2 by using McCullouch Pit model,
30
31    u2=[]
32    for j in range(10):
33        sum=0
34        for i in range(785):
35            sum += weight1[j][i]*v1[i]
36        u2.append(sum)
37        #print("input of layer 2",u)
38
39
40    # For output on layer 2(output layer), by using sigmoid function,
41
42    v2=[]
43
44    for i in range(10):
45        sig=1/(1+ np.exp(-u2[i]))
46        v2.append(sig)
47        #print("output on layer 2 (output layer) : ",v)
48    forward_prop_output.append(v2)
49    yk=forward_prop_output
50
51
52    for k in range(10):
53        for j in range(785):
54            delta_w_kj_1 = (labels[p][k] - yk[0][k])*(yk[0][k])*(1-yk[0][k])
55
56    delta_w_kj_1=learning_rate*delta_w_kj_1
57    #print(delta_w_kj_1)
58
59
60    # computing delta w_ji(0) (learning rule) #weight2
61
62    learning_rate = 0.5 #assuming
63
64
65    for k in range(10):
66        for j in range(785):
67            for l in range(785):
68                delta_w_ji_0 = (labels[p][k] - yk[0][k])*yk[0][k]*(1-yk[0][k])
69
70    delta_w_ji_0=learning_rate*delta_w_ji_0
71    #print(delta_w_ji_0)
72
73    # modifying weight1 and weight2 according to delta_w

```

```

74     #print("previous weight1 matrix : ", weight1)
75     temp1=0
76     for k in range(10):
77         for j in range(785):
78             temp1 =weight1[k][j]
79             weight1[k][j] =temp1+delta_w_kj_1
80
81
82     #print("modified input weight vector for output layer:\n ",weight1)
83
84     #print("Previous weight2 matrix : ", weight2)
85
86     temp2=0
87     for k in range(784):
88         for j in range(785):
89             temp2=weight2[k][j]
90             weight2[k][j] = temp2+ delta_w_ji_0
91
92
93     #print("modified input weight vector for hidden layer:\n ",weight2)
94
95     return
96

```

```

In [69]: 1 error1=0
          2 for p in range(100):
          3     output1=forward_prop(p)
          4     #print(output)
          5     for j in range(10):
          6         error1 += pow((labels[p][j] - output1[0][j]),2)
          7     E1 = error1/2
          8     print(E1)      #error after forward propagation initially

```

/home/ankit/anaconda3/lib/python3.7/site-packages/ipykernel\_launcher.py:20: RuntimeWarning: overflow encountered in exp

112.99455005748383

```

In [73]: 1 for n in range(10):# number of iterations
          2     error=0
          3     for i in range(100):
          4         output=forward_prop(i)
          5         #print(output)
          6         #error = 0
          7         for j in range(10):
          8             error += pow((labels[i][j] - output[0][j]),2)
          9         E = error/2
          10        #print(E)
          11        #if E<0.1:
          12            # break
          13        back_prop(i)
          14    print("Minimized error after 10 iteration: ", E) # Our objective was to Minimize error
          15
          16    # if error difference is not much then incearease the number of iteration to 100

```

/home/ankit/anaconda3/lib/python3.7/site-packages/ipykernel\_launcher.py:20: RuntimeWarning: overflow encountered in exp

/home/ankit/anaconda3/lib/python3.7/site-packages/ipykernel\_launcher.py:24: RuntimeWarning: overflow encountered in exp

Minimized error after 10 iteration: 73.83016436734928

```

In [78]: 1 print("Final input weight matrix for hidden layer: \n",weight2)
          2 print("\nFinal input weight matrix for output layer: \n" weight1)

```



IOPub data rate exceeded.  
 The notebook server will temporarily stop sending output  
 to the client in order to avoid crashing it.  
 To change this limit, set the config variable  
 `NotebookApp.iopub\_data\_rate\_limit`

## Incorporating momentum factor

In [79]:

```
1 gamma=0.9
2 mom1=0
3 mom2=0
```

In [87]:

```
1 def back_prop(p,mom1,mom2):
2     # computing delta w_ji(θ) (learning rule) #weight2
3     learning_rate = 0.5 #assuming
4
5     forward_prop_output=[]
6
7     # Input on input layer -1 (hidden layer ) by using McCullouch Pit mod
8
9     u1=[]
10    for j in range(784):
11        sum=0
12        for i in range(785):
13            sum += weight2[j][i]*data[p][i]
14        u1.append(sum)
15        #print("Input on layer 1 (hidden layer) : ",u)
16
17    # For output on layer 1, by using sigmoid function,
18
19    v1=[]
20    v1.append(1) # appending bias=1
21
22    for i in range(784):
23        sig=1/(1+ np.exp(-u1[i]))
24        v1.append(sig)
25        #print("output on layer 1 : ",v)
26
27
28    # For input on layer 2 by using McCullouch Pit model,
29
30    u2=[]
31    for j in range(10):
32        sum=0
33        for i in range(785):
34            sum += weight1[j][i]*v1[i]
35        u2.append(sum)
36        #print("input of layer 2",u)
37
38
39    # For output on layer 2(output layer), by using sigmoid function,
40
41    v2=[]
42
43    for i in range(10):
44        sig=1/(1+ np.exp(-u2[i]))
45        v2.append(sig)
46        #print("output on layer 2 (output layer) : ",v)
47    forward_prop_output.append(v2)
48    yk=forward_prop_output
49
50
51    for k in range(10):
52        for j in range(785):
53            delta_w_kj_1 = (labels[p][k] - yk[0][k])*(yk[0][k])*(1-yk[0][k]
54
55    delta_w_kj_1=learning_rate*delta_w_kj_1
```

```
56     #print(delta_w_kj_1)
57
58
59     # computing delta w_ji(0) (learning rule) #weight2
60
61     learning_rate = 0.5 #assuming
62
63
64     for k in range(10):
65         for j in range(785):
66             for l in range(785):
67                 delta_w_ji_0 = (labels[p][k] - yk[0][k])*yk[0][k]*(1-yk[0]
68
69     delta_w_ji_0=learning_rate*delta_w_ji_0
70     #print(delta_w_ji_0)
71
72     # modifying weight1 and weight2 according to delta_w
73     #print("previous weight1 matrix : ", weight1)
74
75     # Here, I am using the concept of momentum factor
76     mom1 += pow(gamma,i)*(delta_w_kj_1)
77     mom2 += pow(gamma,i)*(delta_w_ji_0)
78
79
80     temp1=0
81     for k in range(10):
82         for j in range(785):
83             temp1 =weight1[k][j]
84             weight1[k][j] =temp1+(mom1+delta_w_kj_1)
85
86
87     #print("modified input weight vector for output layer:\n ",weight1)
88
89     #print("Previous weight2 matrix : ", weight2)
90
91     temp2=0
92     for k in range(784):
93         for j in range(785):
94             temp2=weight2[k][j]
95             weight2[k][j] = temp2+ (mom2+delta_w_ji_0)
96
97
98     #print("modified input weight vector for hidden layer:\n ",weight2)
99
100     return mom1,mom2
101
```

```
In [92]: 1 for n in range(10):# number of iterations
2         error=0
3         for i in range(100):
4             output=forward_prop(i)
5             #print(output)
6             #error = 0
7             for j in range(10):
8                 error += pow((labels[i][j] - output[0][j]),2)
9             E = error/2
10            #print(E)
11            #if E<0.1:
12                # break
13            back_prop(i,mom1,mom2)
14 print("Minimized error after 10 iteration: ", E) # Our objective was to Min
15
16 # if error difference is not much then incearease the number of iteration b
```

/home/ankit/anaconda3/lib/python3.7/site-packages/ipykernel\_launcher.py:20: RuntimeWarning: overflow encountered in exp  
/home/ankit/anaconda3/lib/python3.7/site-packages/ipykernel\_launcher.py:24: RuntimeWarning: overflow encountered in exp

Minimized error after 10 iteration: 15.723754933879016

In [ ]: 1