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

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

In [50]:
```
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]:
```
print(dataset.shape) #28*28 coulumns for 28*28 pixel images and one column
```

```
(60000, 785)
```

In [52]:
```
print(dataset['label'].unique()) # printing the unique classes of dataset
```

```
[5 0 4 1 9 2 3 6 7 8]
```

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

In [54]:
```
list0=[]
list1=[]
list2=[]
list3=[]
list4=[]
list5=[]
list6=[]
list7=[]
list8=[]
list9=[]
count0=0
count1=0
count2=0
count3=0
count4=0
count5=0
count6=0
count7=0
count8=0
count9=0

for i in range(60000):
        if data1[i][0] == 0:
            if count0 < 10:
                list0.append(i)
                count0=count0+1

        if data1[i][0] == 1:
            if count1 < 10:
                list1.append(i)
```

```
                count1=count1+1

        if data1[i][0] == 2:
            if count2 < 10:
                list2.append(i)
                count2=count2+1

        if data1[i][0] == 3:
            if count3 < 10:
                list3.append(i)
                count3=count3+1

        if data1[i][0] == 4:
            if count4 < 10:
                list4.append(i)
                count4=count4+1

        if data1[i][0] == 5:
            if count5 < 10:
                list5.append(i)
                count5=count5+1

        if data1[i][0] == 6:
            if count6 < 10:
                list6.append(i)
                count6=count6+1

        if data1[i][0] == 7:
            if count7 < 10:
                list7.append(i)
                count7=count7+1

        if data1[i][0] == 8:
            if count8 < 10:
                list8.append(i)
                count8=count8+1

        if data1[i][0] == 9:
            if count9 < 10:
                list9.append(i)
                count9=count9+1
        if count0 == 11 and count1 == 11 and count2 == 11 and count3 == 11
            break

```

In [55]:

```
print(len(list0))
print(len(list1))
print(len(list2))
print(len(list3))
print(len(list4))
print(len(list5))
print(len(list6))
print(len(list7))
print(len(list8))
print(len(list9))

```

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

In [56]:
```
#for dk(p), if class label belongs to class 1, we make 1st value as 1 and c
#others

labels=[]
for i in range(100):
    if data1[i][1]==0:
        labels.append([1,0,0,0,0,0,0,0,0,0])
    if data1[i][1]==1:
        labels.append([0,1,0,0,0,0,0,0,0,0])
    if data1[i][1]==2:
        labels.append([0,0,1,0,0,0,0,0,0,0])
    if data1[i][1]==3:
        labels.append([0,0,0,1,0,0,0,0,0,0])
    if data1[i][1]==4:
        labels.append([0,0,0,0,1,0,0,0,0,0])
    if data1[i][1]==5:
        labels.append([0,0,0,0,0,1,0,0,0,0])
    if data1[i][1]==6:
        labels.append([0,0,0,0,0,0,1,0,0,0])
    if data1[i][1]==7:
        labels.append([0,0,0,0,0,0,0,1,0,0])
    if data1[i][1]==8:
        labels.append([0,0,0,0,0,0,0,0,1,0])
    if data1[i][1]==9:
        labels.append([0,0,0,0,0,0,0,0,0,1])
```

In [57]:
```
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]:

```
data=[]
for i in list0:
    temp=[]
    temp.append(1) #appending bias
    for j in range(1,785):
        temp.append(data1[i][j])
    data.append(temp)

for i in list1:
    temp=[]
    temp.append(1) #appending bias
    for j in range(1,785):
        temp.append(data1[i][j])
    data.append(temp)
for i in list2:
    temp=[]
    temp.append(1) #appending bias
    for j in range(1,785):
        temp.append(data1[i][j])
    data.append(temp)
for i in list3:
    temp=[]
    temp.append(1) #appending bias
    for j in range(1,785):
        temp.append(data1[i][j])
    data.append(temp)
for i in list4:
    temp=[]
    temp.append(1) #appending bias
    for j in range(1,785):
        temp.append(data1[i][j])
    data.append(temp)
for i in list5:
    temp=[]
    temp.append(1) #appending bias
    for j in range(1,785):
        temp.append(data1[i][j])
    data.append(temp)
for i in list6:
    temp=[]
    temp.append(1) #appending bias
    for j in range(1,785):
        temp.append(data1[i][j])
    data.append(temp)
for i in list7:
    temp=[]
    temp.append(1) #appending bias
    for j in range(1,785):
        temp.append(data1[i][j])
    data.append(temp)
for i in list8:
    temp=[]
    temp.append(1) #appending bias
    for j in range(1,785):
        temp.append(data1[i][j])
    data.append(temp)
for i in list9:
    temp=[]
    temp.append(1) #appending bias
    for j in range(1,785):
        temp.append(data1[i][j])
    data.append(temp)
```

In [59]:

```
print(data)
```

```
[[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 51,
159, 253, 159, 50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 48, 238, 252, 252, 252, 237, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 54, 227, 253, 252, 239, 233, 252, 57, 6, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 60, 224, 252, 253, 252, 202, 84, 252,
253, 122, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 163, 252, 252, 25
2, 253, 252, 252, 96, 189, 253, 167, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 51, 238, 253, 253, 190, 114, 253, 228, 47, 79, 255, 168, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 48, 238, 252, 252, 179, 12, 75, 121, 21, 0, 0, 2
53, 243, 50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 38, 165, 253, 233, 208, 8
4, 0, 0, 0, 0, 0, 0, 253, 252, 165, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 17
8, 252, 240, 71, 19, 28, 0, 0, 0, 0, 0, 0, 253, 252, 195, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 57, 252, 252, 63, 0, 0, 0, 0, 0, 0, 0, 0, 0, 253, 252, 195, 0,
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 198. 253. 190. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

## Implementing Forward Propagation MLP for Mnist dataset

In [60]:

```
input_no_of_nodes = 784 # as features are 784-dimensional for 28*28 pixel i
no_of_hidden_layers = 1 # just for simplicity , I am taking one hidden laye
no_of_classes = 10 # 0,1,2,3,4,5,6,7,8,9

# initializing input weights for output layer

weight1 = [] # initializing weights


for k in range(10):
    w=[]
    for j in range(785):
        w.append(np.random.randn())
    weight1.append(w)

print("input weight vector for output layer:\n ",weight1)

# initializing input weights for hidden layer

weight2 = [] # initializing weights

bias= 1



for k in range(784):
    w=[]
    for j in range(785):
        w.append(np.random.randn())
    weight2.append(w)

print("\ninput weight vector for hidden layer:\n " weight2)
```

```
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  -0.1490471484559136  1.2663692306027887  -0.5623253498769216  -0.5040625355
```

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

Out[61]: 785

In [63]:
```
def forward_prop(p):
    forward_prop_output=[]

 # Input on input layer -1 (hidden layer )  by using McCullouch Pit mode

    u1=[]
    for j in range(784):
        sum=0
        for i in range(785):
            sum += weight2[j][i]*data[p][i]
        u1.append(sum)
     #print("Input on layer 1 (hidden layer) : ",u)

     # For output on layer 1, by using sigmoid function,

    v1=[]
    v1.append(1) # appending bias=1

    for i in range(784):
        sig=1/(1+ np.exp(-u1[i]))
        v1.append(sig)
    #print("output on layer 1 : ",v)


    # For input on layer 2 by using McCullouch Pit model,

    u2=[]
    for j in range(10):
        sum=0
        for i in range(785):
            sum += weight1[j][i]*v1[i]
        u2.append(sum)
       #print("input of layer 2",u)


    # For output on layer 2(output layer), by using sigmoid function,

    v2=[]

    for i in range(10):
        sig=1/(1+ np.exp(-u2[i]))
        v2.append(sig)
       #print("output on layer 2 (output layer) : ",v)
    forward_prop_output.append(v2)
    yk=forward_prop_output

    return yk
```

## Implementing Backpropagation MLP for Mnist dataset

In [86]:
```
def back_prop(p):
    # computing delta w_ji(0) (learning rule) #weight2

    learning_rate = 0.5 #assuming

```

```
    forward_prop_output=[]

    # Input on input layer -1 (hidden layer )  by using McCullouch Pit mode

    u1=[]
    for j in range(784):
        sum=0
        for i in range(785):
            sum += weight2[j][i]*data[p][i]
        u1.append(sum)
         #print("Input on layer 1 (hidden layer) : ",u)

    # For output on layer 1, by using sigmoid function,

    v1=[]
    v1.append(1) # appending bias=1

    for i in range(784):
        sig=1/(1+ np.exp(-u1[i]))
        v1.append(sig)
    #print("output on layer 1 : ",v)


    # For input on layer 2 by using McCullouch Pit model,

    u2=[]
    for j in range(10):
        sum=0
        for i in range(785):
            sum += weight1[j][i]*v1[i]
        u2.append(sum)
        #print("input of layer 2",u)


    # For output on layer 2(output layer), by using sigmoid function,

    v2=[]

    for i in range(10):
        sig=1/(1+ np.exp(-u2[i]))
        v2.append(sig)
        #print("output on layer 2 (output layer) : ",v)
    forward_prop_output.append(v2)
    yk=forward_prop_output


    for k in range(10):
        for j in range(785):
            delta_w_kj_1 = (labels[p][k] - yk[0][k])*(yk[0][k])*(1-yk[0][k]

    delta_w_kj_1=learning_rate*delta_w_kj_1
    #print(delta_w_kj_1)


    # computing delta w_ji(0) (learning rule) #weight2

    learning_rate = 0.5 #assuming


    for k in range(10):
        for j in range(785):
            for l in range(785):
                delta_w_ji_0 = (labels[p][k] - yk[0][k])*yk[0][k]*(1-yk[0][

    delta_w_ji_0=learning_rate*delta_w_ji_0
    #print(delta_w_ji_0)

    # modifying weight1 and weight2 according to delta_w
```

```
    #print("previous weight1 matrix : ", weight1)
    temp1=0
    for k in range(10):
        for j in range(785):
            temp1 =weight1[k][j]
            weight1[k][j] =temp1+delta_w_kj_1


    #print("modified input weight vector for output layer:\n ",weight1)

    #print("Previous weight2 matrix : ", weight2)

    temp2=0
    for k in range(784):
        for j in range(785):
            temp2=weight2[k][j]
            weight2[k][j] = temp2+ delta_w_ji_0


    #print("modified input weight vector for hidden layer:\n ",weight2)

    return
```

In [69]:

```
error1=0
for p in range(100):
    output1=forward_prop(p)
    #print(output)
    for j in range(10):
        error1 += pow((labels[p][j] - output1[0][j]),2)
    E1 = error1/2
print(E1)    #error after forward propagation initially
```

```
/home/ankit/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: Ru
ntimeWarning: overflow encountered in exp

112.99455005748383
```

In [73]:

```
for n in range(10):# number of iterations
    error=0
    for i in range(100):
        output=forward_prop(i)
        #print(output)
        #error = 0
        for j in range(10):
            error += pow((labels[i][j] - output[0][j]),2)
        E = error/2
        #print(E)
        #if E<0.1:
         #   break
        back_prop(i)
print("Minimized error after 10 iteration: ", E) # Our objective was to Mir

# 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: Ru
ntimeWarning: overflow encountered in exp
/home/ankit/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:24: Ru
ntimeWarning: overflow encountered in exp

Minimized error after 10 iteration:  73.83016436734928
```

In [78]:

```
print("Final input weight matrix for hidden layer: \n",weight2)
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]:
```
gamma=0.9
mom1=0
mom2=0
```

In [87]:
```
def back_prop(p,mom1,mom2):
    # computing delta w_ji(0) (learning rule) #weight2
    learning_rate = 0.5 #assuming

    forward_prop_output=[]

    # Input on input layer -1 (hidden layer )  by using McCullouch Pit mod

    u1=[]
    for j in range(784):
        sum=0
        for i in range(785):
            sum += weight2[j][i]*data[p][i]
        u1.append(sum)
         #print("Input on layer 1 (hidden layer) : ",u)

    # For output on layer 1, by using sigmoid function,

    v1=[]
    v1.append(1) # appending bias=1

    for i in range(784):
        sig=1/(1+ np.exp(-u1[i]))
        v1.append(sig)
    #print("output on layer 1 : ",v)


    # For input on layer 2 by using McCullouch Pit model,

    u2=[]
    for j in range(10):
        sum=0
        for i in range(785):
            sum += weight1[j][i]*v1[i]
        u2.append(sum)
        #print("input of layer 2",u)


    # For output on layer 2(output layer), by using sigmoid function,

    v2=[]

    for i in range(10):
        sig=1/(1+ np.exp(-u2[i]))
        v2.append(sig)
        #print("output on layer 2 (output layer) : ",v)
    forward_prop_output.append(v2)
    yk=forward_prop_output


    for k in range(10):
        for j in range(785):
            delta_w_kj_1 = (labels[p][k] - yk[0][k])*(yk[0][k])*(1-yk[0][k

    delta_w_kj_1=learning_rate*delta_w_kj_1
```

```
    #print(delta_w_kj_1)


    # computing delta w_ji(0) (learning rule) #weight2

    learning_rate = 0.5 #assuming


    for k in range(10):
        for j in range(785):
            for l in range(785):
                delta_w_ji_0 = (labels[p][k] - yk[0][k])*yk[0][k]*(1-yk[0][k]

    delta_w_ji_0=learning_rate*delta_w_ji_0
    #print(delta_w_ji_0)

    # modifying weight1 and weight2 according to delta_w
    #print("previous weight1 matrix : ", weight1)

    # Here, I am using the concept of momentum factor
    mom1 += pow(gamma,i)*(delta_w_kj_1)
    mom2 += pow(gamma,i)*(delta_w_ji_0)


    temp1=0
    for k in range(10):
        for j in range(785):
            temp1 =weight1[k][j]
            weight1[k][j] =temp1+(mom1+delta_w_kj_1)


    #print("modified input weight vector for output layer:\n ",weight1)

    #print("Previous weight2 matrix : ", weight2)

    temp2=0
    for k in range(784):
        for j in range(785):
            temp2=weight2[k][j]
            weight2[k][j] = temp2+ (mom2+delta_w_ji_0)


    #print("modified input weight vector for hidden layer:\n ",weight2)

    return mom1,mom2

```

In [92]:

```
for n in range(10):# number of iterations
    error=0
    for i in range(100):
        output=forward_prop(i)
        #print(output)
        #error = 0
        for j in range(10):
            error += pow((labels[i][j] - output[0][j]),2)
        E = error/2
        #print(E)
        #if E<0.1:
         #   break
        back_prop(i,mom1,mom2)
print("Minimized error after 10 iteration: ", E) # Our objective was to Mir

# 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: Ru
ntimeWarning: overflow encountered in exp
/home/ankit/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:24: Ru
ntimeWarning: overflow encountered in exp

Minimized error after 10 iteration:  15.723754933879016
```

In [ ]:

```
1
```