foundation of machine learning

November 21, 2018

1 lab1

2 Q1

In [ ]:
```
# hebbian learning with signum function.............
import numpy as np
x=np.matrix([[1,-2,1.5,0],[1,-.5,-2,-1.5],[0,1,-1,1.5]])
w0=np.array([1,-1,0,.5])
n=.05
#err =np.matlib.zeros([1,3])
eps=.001
while(1):
    for i in range(0,2):
        net=np.dot(w0,x[i].T)
        y=np.sign(net)
        del_w=n*y*x[1]
        w_nxt=w0+del_w
        err=np.linalg.norm(w_nxt-w0)
        if(err<eps):
            break
        w0=w_nxt
print(w0)
```

3 output - the code is not converging

4 Q2

In [ ]:
```
# hebbian learning with bipolar sigmoidal function
import numpy as np

x=np.matrix([[1,-2,1.5,0],[1,-.5,-2,-1.5],[0,1,-1,1.5]])
w0=np.array([1,-1,0,.5])
n=.05
eps=.0001
while(1):
    for i in range(0,2):
```

```
            net=np.dot(w0,x[i].T)
            y = (2/(1+exp(-net)))-1
            del_w=n*y*x[1]
            w_nxt=w0+del_w
            err=np.linalg.norm(w_nxt-w0)
            if(err<eps):
                break
            w0=w_nxt
    print(w0)
```

5 output - the code is not converging

6 Q3

In [11]:
```
# by perceptron learning with noise data.........
import numpy as np
x=np.array([[1,-2,0,-1],[0,-1.5,-0.5,-1.0],[-1,1,0.5,-1]])
d=np.array([[-1],[-1],[1]])
w0=np.array([1,-1,0,0.5])
no_data_pts,tmp=np.shape(x)
iteration=0
while(1):
    iteration=iteration+1
    cont=0
    for i in range(0,3):
        net=np.dot(w0,x[i].T)
        y=np.sign(net)
        r=d[i]-y

        w_nxt=w0+r*x[i]
        tmp=w_nxt-w0
        err=np.linalg.norm(tmp)
        if(err<.001):
            cont=cont+1
        w0=w_nxt
    if(cont==no_data_pts):
        break
print(w0)
print('iteration is {}'.format(iteration))
noisy_data=np.array([1,-2,0,-1.5])
y=np.sign(np.dot(w0.T,noisy_data))
noisy_data=np.array([-1,1.2,0.5,-1.2])
y1=np.sign(np.dot(w0.T,noisy_data))
```

```
[-1.   3.   0.   2.5]
iteration is 2
```

7 output - it converge in 2 iteration

8 Q4

```
In [ ]: #train a neuron in a xy plane by habbian rule.....
        import numpy as np

        x=np.matrix([[5,10],[1,3],[2,4],[5,1],[1,-3],[2,-4]])
        y=np.array([[1],[1],[1],[-1],[-1],[-1]])
        w0=np.matrix([1,-1])
        n=.05
        #err =np.matlib.zeros([1,3])
        eps=.001
        while(1):
            cnt=0
            for i in range(0,2):
                net=np.dot(w0,x[i].T)
                y=np.sign(net)
                del_w=n*y*x[1]
                w_nxt=w0+del_w
                err=np.linalg.norm(w_nxt-w0)
                if(err<eps):
                    cnt=cnt+1
                w0=w_nxt
            if(cnt==2):
                break

        print(w0)
```

9 the code is not converging

```
In [ ]: # train the neuron by perceptron rule .........
        import numpy as np
        x=np.matrix([[5,10],[1,3],[2,4],[5,1],[1,-3],[2,-4]])
        d=np.array([[1],[1],[1],[-1],[-1],[-1]])
        w0=np.matrix([1,-1])
        no_data_pts,tmp=np.shape(x)
        iteration=0
        n=1
        while(1):
            iteration=iteration+1
            cont=0
```

```
        for i in range(0,no_data_pts):
            net=np.dot(w0,x[i].T)
            y=np.sign(net)
            r=d[i]-y

            w_nxt=w0+n*r*x[i]
            tmp=w_nxt-w0
            err=np.linalg.norm(tmp)
            if(err<.001):
                cont=cont+1
            w0=w_nxt
        if(cont==no_data_pts):
            break
    print(w0)
    print('number of iteration is {}'.format(iteration))
```

10 OUTPUT= W0=[[-9 15]

11 number of iteration is 3

12 lab 2

13 Q1

```
In [ ]: #single discrete perceptron training algorithm keeping eta=1
        import numpy as np

        x=np.matrix([[0.8,0.5,0],[0.9,0.7,0.3],[1,0.8,0.5],[0,0.2,0.3],[0.2,0.1,1.3],[0.2,0.7,
        d=np.array([[1],[1],[1],[-1],[-1],[-1]])
        w0=np.array([0,0,0])
        no_data_pts,tmp=np.shape(x)
        iteration=0
        n=1# ete=1
        while(1):
            iteration=iteration+1
            cont=0
            for i in range(0,no_data_pts):
                net=np.dot(w0,x[i].T)
                y=np.sign(net)
                r=d[i]-y

                w_nxt=w0+n*r*x[i]
                tmp=w_nxt-w0
                err=np.linalg.norm(tmp)
                if(err<.001):
                    cont=cont+1
```

```
            w0=w_nxt
        if(cont==no_data_pts):
            break
    print(w0)
    print(iteration)
```

14 output

15 W0=[[0.8 0.1 -0.6]]

16 number of iteration=12

17 Q2

In [ ]:
```
#delta learning rule for a single contineous perceptron.....
import numpy as np
x=np.matrix([[0.8,0.5,0],[0.9,0.7,0.3],[1,0.8,0.5],[0,0.2,0.3],[0.2,0.1,1.3],[0.2,0.7,
d=np.array([[1],[1],[1],[-1],[-1],[-1]])
w0=np.array([0,0,0])
no_data_pts,tmp=np.shape(x)
iteration=0
n=.001#eta=.001
while(1):
    iteration=iteration+1
    cont=0
    for i in range(0,no_data_pts):
        net=np.dot(w0,x[i].T)
        y=1/(1+np.exp(-net))
        dy=0.5*(1-y**2)
        r=d[i]-y

        w_nxt=w0+n*r*dy*x[i]
        tmp=w_nxt-w0
        err=np.linalg.norm(tmp)
        if(err<.001):
            cont=cont+1
        w0=w_nxt/np.linalg.norm(w_nxt)
    if(cont==no_data_pts):
        break
print(w0)
print(iteration)
```

18 output is = w0=[[0.84816134 0.52973658 -0.00122033]]

19 number of iteration is 6

20 Q3

```
In [ ]: #WIDROW-HOFF LEARNING ALGORITHM..........
        import numpy as np
        x=np.matrix([[1,2],[1,-1],[-1,2],[-2,-1]])
        d=np.array([[-1],[1],[1],[-1]])
        w0=np.array([0.1,-0.3])
        no_data_pts,tmp=np.shape(x)
        n=0.001
        iteration=0
        while(1):

            cnt=0
            for i in range(0,no_data_pts):
                iteration=iteration+1
                net=np.dot(w0,x[i].T)
                r=d[i]-net
                w0_nxt=w0+n*r*x[i]
                tmp=w0_nxt-w0
                err=np.linalg.norm(tmp)
                print(err)
                if err<10**-2:

                    cnt=cnt+1
                w0=w0_nxt/np.linalg.norm(w0_nxt)
            if(cnt==no_data_pts):
                break
        print(w0)
        print(iteration)
```

21 output is w0=[[0.70574373 -0.70846721]]

22 number of iteration is 4

23 LAB 3

24 q1- AND operation

```
In [1]: #and operation bipolar input by perceptron learning
        import numpy as np
        x=np.matrix([[1,1],[1,-1],[-1,1],[-1,-1]])
        d=np.array([[1],[-1],[-1],[-1]])
```

```
        w0=np.array([0.5,-0.5])
        n=.002
        iteration=0
        col,tmp=np.shape(x)
        while(1):
            cnt=0
            for i in range(0,col):
                iteration=iteration+1
                net=np.dot(w0,x[i].T)
                y=1/(1+np.exp(-net))
                r=d[i]-y
                w_nxt=w0+n*r*x[i]
                tmp=w_nxt-w0
                err=np.linalg.norm(tmp)
                print(err)
                if err<10**-2:

                    cnt=cnt+1
                w0=w_nxt/np.linalg.norm(w_nxt)
            if(cnt==col):
                break
        print(w0)
        print(iteration)
```

```
0.0014142135623730963
0.00510369660012198
0.0033815847870624803
0.0042406372076714705
[[ 0.71150818 -0.70267782]]
4
```

In [2]:
```
        #AND operation by signum function using widrow-hoff learning rule
        import numpy as np
        x=np.matrix([[1,1],[1,-1],[-1,1],[-1,-1]])
        d=np.array([[1],[-1],[-1],[-1]])
        w0=np.array([0.5,-0.5])
        n=.002
        iteration=0
        col,tmp=np.shape(x)
        while(1):
            cnt=0
            for i in range(0,col):
                iteration=iteration+1
                net=np.dot(w0,x[i].T)
                r=d[i]-net
                w_nxt=w0+n*r*x[i]
                tmp=w_nxt-w0
```

```
                err=np.linalg.norm(tmp)
                print(err)
                if err<10**-2:

                    cnt=cnt+1
                w0=w_nxt/np.linalg.norm(w_nxt)
            if(cnt==col):
                break
        print(w0)
        print(iteration)
```

```
0.0028284271247461927
0.006828395125130148
0.0011715404341103351
0.0028122983536799876
[[ 0.71192993 -0.70225051]]
4
```

```
In [4]: #input AND operation using signum function done by perceptron learning
        import numpy as np
        x=np.matrix([[1,1],[1,-1],[-1,1],[-1,-1]])
        d=np.array([[1],[-1],[-1],[-1]])
        w0=np.array([0.5,-0.5])
        n=.002
        iteration=0
        col,tmp=np.shape(x)
        while(1):
            cnt=0
            for i in range(0,col):
                iteration=iteration+1
                net=np.dot(w0,x[i].T)
                y=np.sign(net)
                r=d[i]-y
                w_nxt=w0+n*r*x[i]
                tmp=w_nxt-w0
                err=np.linalg.norm(tmp)
                print(err)
                if err<10**-2:

                    cnt=cnt+1
                w0=w_nxt/np.linalg.norm(w_nxt)
            if(cnt==col):
                break
        print(w0)
        print(iteration)
```

```
0.0028284271247461927
```

```
0.005656854249492385
0.0
0.0
[[ 0.70994556 -0.70425656]]
4
```

25 OR operation

```
In [5]: #OR operation bipolar input by perceptron learning
        import numpy as np
        x=np.matrix([[1,1],[1,-1],[-1,1],[-1,-1]])
        d=np.array([[1],[1],[1],[-1]])
        w0=np.array([0.5,-0.5])
        n=.004
        iteration=0
        col,tmp=np.shape(x)
        while(1):
            cnt=0
            for i in range(0,col):
                iteration=iteration+1
                net=np.dot(w0,x[i].T)
                y=1/(1+np.exp(-net))
                r=d[i]-y
                w_nxt=w0+n*r*x[i]
                tmp=w_nxt-w0
                err=np.linalg.norm(tmp)
                print(err)
                if err<10**-2:

                    cnt=cnt+1
                w0=w_nxt/np.linalg.norm(w_nxt)
            if(cnt==col):
                break
        print(w0)
        print(iteration)
```

```
0.0028284271247461927
0.0011063228501433527
0.004550531421590335
0.008477253770786631
[[ 0.71588352 -0.69821972]]
4
```

```
In [6]: #or operation bipolar input by widrow-hoff learning
        import numpy as np
        x=np.matrix([[1,1],[1,-1],[-1,1],[-1,-1]])
```

```
        d=np.array([[1],[1],[1],[-1]])
        w0=np.array([0.5,-0.5])
        n=.002
        iteration=0
        col,tmp=np.shape(x)
        while(1):
            cnt=0
            for i in range(0,col):
                iteration=iteration+1
                net=np.dot(w0,x[i].T)

                r=d[i]-net
                w_nxt=w0+n*r*x[i]
                tmp=w_nxt-w0
                err=np.linalg.norm(tmp)
                print(err)
                if err<10**-2:

                    cnt=cnt+1
                w0=w_nxt/np.linalg.norm(w_nxt)
            if(cnt==col):
                break
        print(w0)
        print(iteration)
```

```
0.0028284271247461927
0.0011715408756377626
0.0068283950500208496
0.002812298353674021
[[ 0.71192993 -0.70225051]]
4
```

26 xor operation

```
In [7]: #xor operation bipolar input by perceptron learning
        import numpy as np
        x=np.matrix([[1,1],[1,-1],[-1,1],[-1,-1]])
        d=np.array([[-1],[1],[1],[-1]])
        w0=np.array([0.5,-0.5])
        n=.004
        iteration=0
        col,tmp=np.shape(x)
        while(1):
            cnt=0
            for i in range(0,col):
                iteration=iteration+1
```

```
                net=np.dot(w0,x[i].T)
                y=1/(1+np.exp(-net))
                r=d[i]-y
                w_nxt=w0+n*r*x[i]
                tmp=w_nxt-w0
                err=np.linalg.norm(tmp)
                print(err)
                if err<10**-2:

                    cnt=cnt+1
                w0=w_nxt/np.linalg.norm(w_nxt)
            if(cnt==col):
                break
        print(w0)
        print(iteration)
```

```
0.008485281374238578
0.0011064033924073154
0.004550451057256001
0.008509362117028373
[[ 0.7046052  -0.70959954]]
4
```

```
In [9]: #xor operation bipolar input by widrow-hoff learning rule
        import numpy as np
        x=np.matrix([[1,1],[1,-1],[-1,1],[-1,-1]])
        d=np.array([[-1],[1],[1],[-1]])
        w0=np.array([0.5,-0.5])
        n=.002
        iteration=0
        col,tmp=np.shape(x)
        while(1):
            cnt=0
            for i in range(0,col):
                iteration=iteration+1
                net=np.dot(w0,x[i].T)

                r=d[i]-net
                w_nxt=w0+n*r*x[i]
                tmp=w_nxt-w0
                err=np.linalg.norm(tmp)
                print(err)
                if err<10**-2:

                    cnt=cnt+1
                w0=w_nxt/np.linalg.norm(w_nxt)
            if(cnt==col):
```

```
            break
        print(w0)
        print(iteration)
```

```
0.0028284271247461927
0.0011715408756377626
0.0068283950500208496
0.0028445558958183643
[[ 0.70626649 -0.70794607]]
4
```

27 lab- 6

```
In [10]: #lab=6.............................
         import numpy as np
         my_data=np.array(('heli.csv','plane.csv','tank.csv'))
         w=np.zeros((144,144))
         image=[]
         for i in range(0,3):
             data=np.genfromtxt(my_data[i],delimiter=",")
             tmp=data.flatten()
             pattern=np.matrix(tmp)
             image.append(pattern)
             tmp=np.outer(pattern,pattern)
             w=w+tmp

         #Recall equation
         data=np.genfromtxt('testdata.csv',delimiter=",")
         input_pattern=data.flatten()
         no_clm=np.shape(input_pattern)[0]
         tmp1=np.matmul(input_pattern,w)
         y=np.sign(tmp1)
         for i in range(0,no_clm):
             if (y[i]==0):
                 y[i]=input[i]
         dist_list=[]
         for i in range(0,3):
             sub=abs(y-image[i])
             dist=np.sum(sub)
             dist_list.append(dist)
         indx=dist_list.index(min(dist_list))
         print(indx)
```

```
1
```

28 output is helicopter

29 lab 5

In [11]:
```
# lab=5.............................
import numpy as np
#filename=np.array(('zero.csv','one.csv','two.csv','three.csv','four.csv','five.csv',
my_data=np.array(('four.csv','five.csv','six.csv','seven.csv','eight.csv','nine.csv'))
w=np.zeros((35,35))
input_pattern=np.zeros((10,35))
n=6
img_arr=[]
for i in range(0,n):
    data=np.genfromtxt(my_data[i],delimiter=",")

    tmp=data.flatten()
    input_pattern[i]=tmp
    pattern=np.matrix(tmp)
    img_arr.append(pattern)
    tmp=np.outer(pattern,pattern)
    w=w+tmp
    #Recall equation
for k in range (0,n):
    no_clm=np.shape(input_pattern[k])[0]
    tmp1=np.matmul(input_pattern[k],w)
    y1=np.sign(tmp1)
    for i in range(0,no_clm):
        if (y1[i]==0):
            y1[i]=input[i]

    dist_list=[]
    for i in range(0,n):
        sub=abs(y1-img_arr[i])
        dist=np.sum(sub)
        dist_list.append(dist)
    indx=dist_list.index(min(dist_list))
    print(indx)
```

```
0
1
2
3
4
5
```

In [13]:
```
import numpy as np
w=np.zeros((70,70))
```

```
img=[]
input_pattern=np.zeros((5,70))
my_data=np.array(('A.csv','B.csv','C.csv','D.csv','E.csv'))
for i in range(0,5):
    data=np.genfromtxt(my_data[i],delimiter=",")

    tmp=data.flatten()
    input_pattern[i]=tmp
    pattern=np.matrix(tmp)
    img.append(pattern)
    tmp=np.outer(pattern,pattern)
    w=w+tmp
 #Recall
for k in range (0,5):
    x=np.shape(input_pattern[k])[0]
    mem=np.matmul(input_pattern[k],w)
    y=np.sign(mem)
    for i in range(0,x):
        if (y[i]==0):
            y[i]=input_pattern[k][i]

    dist_list=[]
    for i in range(0,5):
        diff=np.abs(y[i]-img[i])
        dist=np.sum(diff)
        dist_list.append(dist)
    indx=dist_list.index(min(dist_list))
    print(indx)
```

```
3
3
3
3
3
```

30 OUTPUT =index is 3

31 lab 8

32 Q1

```
In [12]: import numpy as np
         w=np.zeros((144,120))

         A_vector=[]
         B_vector=[]
```

```
a=np.array(('heli.csv','plane.csv','tank.csv'))
b=np.array(('three.csv','one.csv','two.csv'))

for i in range(0,3):
    data_a=np.genfromtxt(a[i],delimiter=",")
    data_b=np.genfromtxt(b[i],delimiter=",")
    tmp=data_a.flatten()
    pattern_a=np.array(tmp)
    A_vector.append(pattern_a)
    tmp2=data_b.flatten()
    pattern_b=np.array(tmp2)
    B_vector.append(pattern_b)
    tmp=np.outer(pattern_a,pattern_b)
    w=w+tmp

#Recall equation
data=np.genfromtxt('testdata.csv',delimiter=",")
x=data.flatten()
tmp1=np.matmul(x,w)
new_B=np.sign(tmp1)
new_B[new_B==0] = 1
first_time=0
iteration=0
limit_pts=[]
while(1):
    iteration=iteration +1
    for i in range (0,3):
        if(new_B==B_vector[i]).all():
            print("Value corresponding to pattern index - :",i)
            break
    if(first_time !=0):
        if any((new_B == x).all() for x in limit_pts):
            #print('Limit pt...')
            break
    else:
        first_time=1
    limit_pts.append(new_B)
    tmp2=np.matmul(new_B,w.T)
    update_A=np.sign(tmp2)
    update_A[update_A==0] = x[update_A==0]
    new_A=update_A
    tmp1=np.matmul(new_A,w)
    B=new_B
    new_B=np.sign(tmp1)
    new_B[new_B==0] = B[new_B==0]
print(iteration)
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-12-34ed4fb628e0> in <module>()
     17     B_vector.append(pattern_b)
     18     tmp=np.outer(pattern_a,pattern_b)
---> 19     w=w+tmp
     20
     21 #Recall equation


ValueError: operands could not be broadcast together with shapes (144,120) (144,35)
```

Value corresponding to pattern index - : 1 Value corresponding to pattern index - : 1

33 Q2

```
In [7]: import numpy as np


        weight_vector=np.zeros((5,4))
        A_vector=[]
        B_vector=[]

        def compute_hamming_dist(u,v):
            tmp=abs(u-v)
            dist=np.sum(tmp)
            return dist

        def transfer_func(weight_vector,input):
            global A_vector,B_vector
            limit_pts=[]
            new_A=input

            tmp1=np.matmul(input,weight_vector)
            new_B=np.sign(tmp1)
            new_B[new_B==0] = -1
            first_time=0
            iteration=0
            while(1):
                for i in range (0,2):
                    if(new_B==B_vector[i]).all():
                        print("Value corresponding to pattern - :",i)
                        print("iterations=",iteration)
                        print("hammind dist from 0-",compute_hamming_dist(new_B,B_vector[0]))
```

```
                print("hammind dist from 1-",compute_hamming_dist(new_B,B_vector[1]))
                return
        if(first_time !=0):
            if any((new_B == x).all() for x in limit_pts):
                print("iterations=",iteration)
                print('Limit pt...')
                print("hammind dist from 0-",compute_hamming_dist(new_B,B_vector[0]))
                print("hammind dist from 1-",compute_hamming_dist(new_B,B_vector[1]))
                return
        else:
            first_time=1
        limit_pts.append(new_B)
        tmp2=np.matmul(new_B,weight_vector.T)
        update_A=np.sign(tmp2)
        update_A[update_A==0] = new_A[update_A==0]
        new_A=update_A
        tmp1=np.matmul(new_A,weight_vector)
        B=new_B
        new_B=np.sign(tmp1)
        new_B[new_B==0] = B[new_B==0]
        iteration=iteration +1


a1 = np.matrix([0, 1, 0, 1, 0])
a2 = np.matrix([1, 1, 0, 0, 0])
b1 = np.matrix([1, 0, 0, 1])
b2 = np.matrix([0, 1, 0, 1])
a1[a1 == 0] = -1
a2[a2 == 0] = -1
b1[b1 == 0] = -1
b2[b2 == 0] = -1
A_vector.append(a1)
B_vector.append(b1)
tmp=np.outer(a1,b1)
weight_vector=weight_vector+tmp
A_vector.append(a2)
B_vector.append(b2)
tmp=np.outer(a2,b2)
weight_vector=weight_vector+tmp


input_pattern=np.array([-1,1,1,1,-1])

transfer_func(weight_vector,input_pattern)

input_pattern=np.array([-1,-1,1,1,-1])

transfer_func(weight_vector,input_pattern)
```

```
Value corresponding to pattern - : 0
iterations= 0
hammind dist from 0- 0.0
hammind dist from 1- 4.0
iterations= 1
Limit pt...
hammind dist from 0- 4.0
hammind dist from 1- 8.0
```

34 LAB 9

```
In [8]: import numpy as np
```

```
        weight_vector=np.zeros((15,10))

        A_vector=[]
        B_vector=[]


        def transfer_func(weight_vector,input):
            global A_vector,B_vector
            limit_pts=[]
            new_A=input

            tmp1=np.matmul(input,weight_vector)
            new_B=np.sign(tmp1)
            new_B[new_B==0] = 1
            first_time=0
            iteration=1
            while(1):
                for i in range (0,4):
                    if(new_B==B_vector[i]).all():
                        print("Value corresponding to pattern index - :",i)
                        print("Number of iterations",iteration)
                        return
                if(first_time !=0):
                    if any((new_B == x).all() for x in limit_pts):
                        print('Limit pt...')
                        print("Number of iterations",iteration)
                        return
                else:
                    first_time=1
                limit_pts.append(new_B)
                tmp2=np.matmul(new_B,weight_vector.T)
                update_A=np.sign(tmp2)
```

```
                update_A[update_A==0] = new_A[update_A==0]
                new_A=update_A
                tmp1=np.matmul(new_A,weight_vector)
                B=new_B
                new_B=np.sign(tmp1)
                new_B[new_B==0] = B[new_B==0]
                iteration=iteration +1

        data_a=np.genfromtxt('input_data.csv',delimiter=",")
        data_b=np.genfromtxt('output_data.csv',delimiter=",")
        for i in range (0,4):
            A_vector.append(data_a[i])
            B_vector.append(data_b[i])
            tmp=np.outer(data_a[i],data_b[i])
            weight_vector=weight_vector+tmp


        input_pattern=np.array([-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1])
        transfer_func(weight_vector,input_pattern)

Limit pt...
Number of iterations 3
```