

# Machine Learning (ENME808E): Homework 5

Koyal Bhartia(116350990)

May 1, 2019

## 1 Homework 6 Problems from LFD

### Overfitting and Deterministic Noise

#### 1.1 Question 1

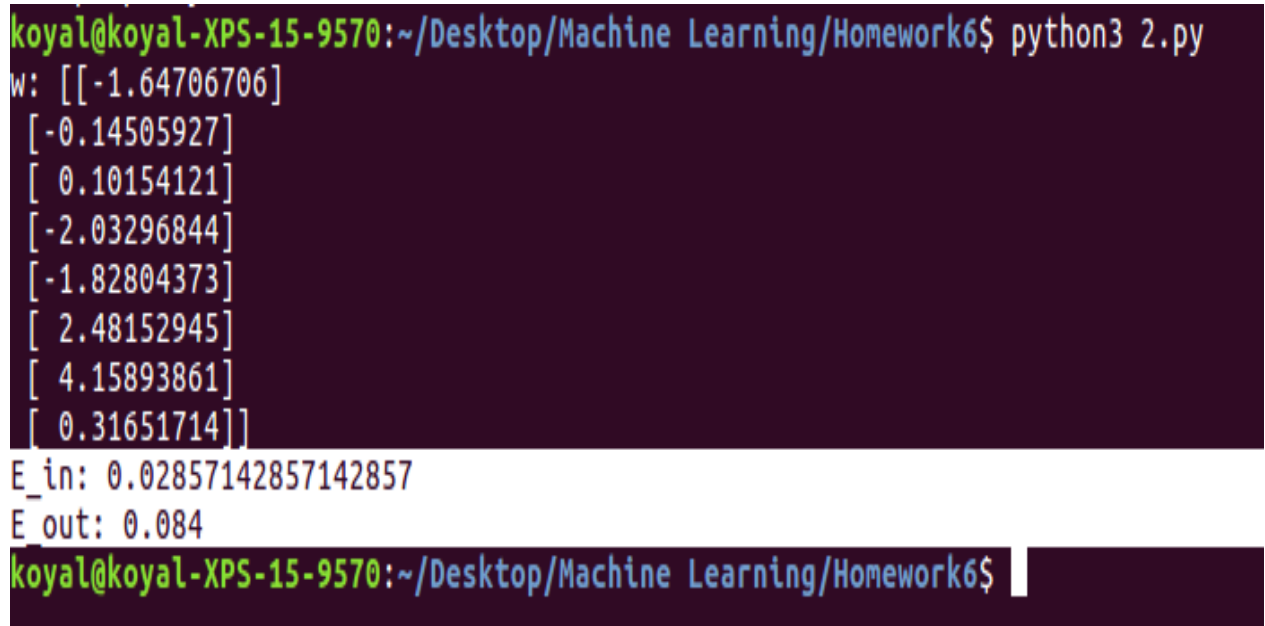
ANSWER:[b]

Deterministic noise increases by using less than the available number of hypotheses Using  $H'$  will increase the deterministic noise as we will have less hypotheses available to deal with a higher-order deterministic function. Hence the answer is option b.

### Regularization with Weight Decay

#### 1.2 Question 2

ANSWER:[a]



```
koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework6$ python3 2.py
w: [[-1.64706706]
 [-0.14505927]
 [ 0.10154121]
 [-2.03296844]
 [-1.82804373]
 [ 2.48152945]
 [ 4.15893861]
 [ 0.31651714]]
E_in: 0.02857142857142857
E_out: 0.084
koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework6$
```

Figure 1: Term. Screenshot of Ques 2

Linear Regression has been implemented on the data provided, using the non-linear transformation as given in the question. The weight is got by taking pseudo inverse of the data set as shown below:

$$W = (X^T X)^{-1} X^T Y \quad (1)$$

where  $X = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2, |x_1 - x_2|, |x_1 + x_2|)$ . Here  $x$  and  $y$  is the training data given.

The above hypothesis is used to calculate the no of misclassified points. This is done using:

$$\text{sign}(w^{*T} x_n) = y_n \quad (2)$$

If the above sign doesnt match, then the number of misclassified points is incremented. Thus  $E_{in}$  is the no of misclassified points divided by the total length of training data set.

Similarly,  $E_{out}$  is calculated using the hypothesis  $w$  found using the training dataset.

As seen from the screenshot, the value of  $E_{in}$  we get is 0.028571 and  $E_{out}$  as 0.084.

This is closest to the values of 0.03 and 0.08 for  $E_{in}$  and  $E_{out}$  respectively. Thus the answer is option a.

### 1.3 Question 3

ANSWER:[d]

Now we add the term while calculating  $w$  using the training data. This is done as follows: Here,

$$X = (A^T A + \lambda I)^{-1} A^T B \quad (3)$$

where  $\lambda$  is  $10^k$ . Here shift in the value of  $\lambda$ , shifts the hypothesis. This is responsible to change bias and variance of the line. If we decrease bias it is good to approximate and if variance is less you have less error in sample.

So there is a trade off between Bias and variance which is shifted by  $\lambda$ .

Below is the terminal screenshot of the values of  $w$ ,  $E_{in}$  and  $E_{out}$  after adding the weight decay term to linear regression.

```

koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework6$ python3 3.py
Input value k for the lambda term:-3
Regularization with k= -3
w: [[-1.6432827 ]
     [-0.14333537]
     [ 0.10144329]
     [-2.02456533]
     [-1.81721505]
     [ 2.45550685]
     [ 4.14009201]
     [ 0.31960135]]
E_in: 0.02857142857142857
E_out: 0.08
koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework6$ python3 3.py

```

Figure 2: Term. Screenshot of Ques 3

As seen the values obtained for  $E_{in}$  and  $E_{out}$  are 0.028 and 0.08 respectively. This is closest to 0.03 and 0.08 from among the given options. Hence the answer is option d.

#### 1.4 Question 4

ANSWER:[e]

Now, in the above case, we just change the value of  $k$  to 3. This basically changes the weight decay term which brings a change in the hypothesis, hence the bias and variance and hence the error or the no of misclassified points changes. Below is the terminal screenshot of the same.

```

koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework6$ python3 3.py
Input value k for the lambda term:3
Regularization with k= 3
w: [[ 0.00435688]
     [-0.00134416]
     [ 0.0024939 ]
     [ 0.00328695]
     [ 0.00484127]
     [-0.00862023]
     [ 0.01786706]
     [-0.00490192]]
E_in: 0.37142857142857144
E_out: 0.436
koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework6$ python3 3.py

```

Figure 3: Term. Screenshot of Ques 4

As seen we get the values of  $E_{in}$  and  $E_{out}$  as 0.371 and 0.436 respectively. This is closest to the repesitive values of 0.4 and 0.4 from among the given options. Hence the answer is option e.

### 1.5 Question 5

ANSWER:[d]

Now we test the same for different values of k and we get to notice how a change in the value of the weight decay terms changes the error values.

Below you can see a terminal screenshot for values of k:2,2,0,-1 and -2.

```

koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework6$ python3 3.py
Input value k for the lambda term:2
Regularization with k= 2
w: [[ 0.01252047]
     [-0.01060738]
     [ 0.01930888]
     [ 0.01871963]
     [ 0.03378485]
     [-0.07194768]
     [ 0.13723846]
     [-0.05494131]]
E_in: 0.2
E_out: 0.228
koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework6$ python3 3.py
Input value k for the lambda term:1
Regularization with k= 1
w: [[-0.1463526 ]
     [ 0.01217748]
     [ 0.08556331]
     [-0.00265865]
     [ 0.12887457]
     [-0.27755605]
     [ 0.55370637]
     [-0.26705289]]
E_in: 0.05714285714285714
E_out: 0.124
koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework6$ python3 3.py
Input value k for the lambda term:0
Regularization with k= 0
w: [[-0.69158657]
     [ 0.09465205]
     [ 0.11629499]
     [-0.41331575]
     [ 0.0912442 ]
     [-0.15351665]
     [ 1.34250371]
     [-0.12458368]]
E_in: 0.0
E_out: 0.092
koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework6$ python3 3.py
Input value k for the lambda term:-1
Regularization with k= -1
w: [[-1.35650275]
     [-0.03513301]
     [ 0.09927707]
     [-1.44734213]
     [-1.06651993]
     [ 1.09497493]
     [ 2.99472736]
     [ 0.33998272]]
E_in: 0.02857142857142857
E_out: 0.056
koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework6$ python3 3.py
Input value k for the lambda term:-2
Regularization with k= -2
w: [[-1.61015542]
     [-0.12868699]
     [ 0.10069025]
     [-1.95218948]
     [-1.72367067]
     [ 2.24053348]
     [ 3.98096905]
     [ 0.34204144]]
E_in: 0.02857142857142857
E_out: 0.084
koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework6$ █

```

Figure 4: Code of Ques 5.6

Thus from the above figures can be summarized as follows which shows the variation of errors with k.

Value of k	$E_{in}$	$E_{out}$
-3	0.028571	0.08
3	0.371428	0.436
2	0.2	0.228
1	0.057142	0.124
0	0.0	0.092
-1	0.028571	0.056
-2	0.028571	0.084

We see that the smallest value of  $E_{out}$  obtained is 0.056 which we get using k as -1. Hence the answer is option d.

## 1.6 Question 6

ANSWER:[b]

Referring back to the table formulated above, it is clear that the smallest value of  $E_{out}$  obtained is 0.056 which is closest to 0.06 from the given options. Hence the answer is option b.

## Regularization for Polynomials

## 1.7 Question 7

ANSWER:[c]

Here the given hypothesis set is as follows:

$$H_Q = \left[ h | h(x) = w^T z = \sum_{q=0}^Q w_q L_q(x) \right] \quad (4)$$

where z is the vector of Legendre polynomials i.e  $z = (1, L_1(x), L_2(x), \dots, L_Q(x))$  and  $L_0(x) = 1$ .

And, the hypothesis set defined by the constraint is given as:

$$H(Q, C, Q_0) = h | h(x) = w^T z \in H_Q; w_q = C \text{ for } q \geq Q_0 \quad (5)$$

Now consider  $H(10,0,3)$ :

Here if we take  $C=0$ , and  $Q_0 = 3$ , then from eqn 4 and 5 we see that  $w_q$  would be 0 for all values of q greater than or equal to 3. Thus the largest degree polynomial here would be of degree 2.

Now considering  $H(10,0,4)$ :

Similar to the above case, here  $C=0$ , and  $Q_0 = 4$ , thus from eqn 4 and 5 we see that  $w_q$  would be 0 for all values of q greater than or equal to 4. Thus the largest degree polynomial here would be of degree 3.

Thus the union between  $H(10,0,3)$  and  $H(10,0,4)$ , will result in a polynomial of degree 3, which is  $H_3$ . Hence this rules out option a.

And intersection between  $H(10,0,3)$  and  $H(10,0,4)$ , will result in a polynomial of degree 2, which is  $H_2$ . This is option c.

Moreover, when  $c=1$ , the  $H$  will be of degree  $Q$ ; i.e both the intersection and union will give  $H_Q$ .

Hence as already shown above, the option c is correct.

## Neural Networks

### 1.8 Question 8

ANSWER:[d]

To calculate the total no of operations following are the equations that will be needed:

$$s_j^{(l)} = \sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)} \quad (6)$$

$$\delta_i^{(l-1)} = (1 - (x_i^{l-1})^2) \sum_{j=1}^{d^{(l)}} w_{ij}^{(l)} \delta_j^{(l)} \quad (7)$$

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta x_i^{(l-1)} \delta_j^{(l)} \quad (8)$$

Using the algorithm of back propagation, the first step is to compute all the  $x_j^{(l)}$  i.e calculate the no of operations required for forward propagation. It is given that the no of nodes in the input layer is 5. The no of nodes in the 2nd layer is 3, and 1 in the final layer. Thus the total no of operations required here is: This is in accordance to eqn 6 From 1st to 2nd layer:  $5*3=15$  From 2nd to 3rd layer:  $3*1=3$  Thus  $15+3=18$

Now the 2nd step is computing the operations of back propagation: This is calculated using eqn 7; here this intuitively requires the same no of operations as the operations required is 18.

The 3rd step uses eqn 8. Here the weight will be updated for each of the nodes. The total no of nodes here is  $5+3+1$  i.e 9.

Hence the total no of operations required is the summation of operations of the 3 steps i.e  **$18+18+9=45$** .

Hence the answer is option d.

### 1.9 Question 9

ANSWER:[a] The number of weights of a neural network depends on the number of nodes per layer. This is also clear from eqn 6 and eqn 7. This can be intuitively solved. If we imagine the simplest case of 2 nodes per layer, we get 18 layers. Thus now considering the 10 input units, the no of weights that will be required will be  $10*1 + (\text{nodes per layer}) * (\text{no of layers})$  i.e  $10+2*18=46$  Another possible way is if we

consider 10 input units and 36 weights, 1 for each hidden layer i.e  $10+36=46$

### 1.10 Question 10

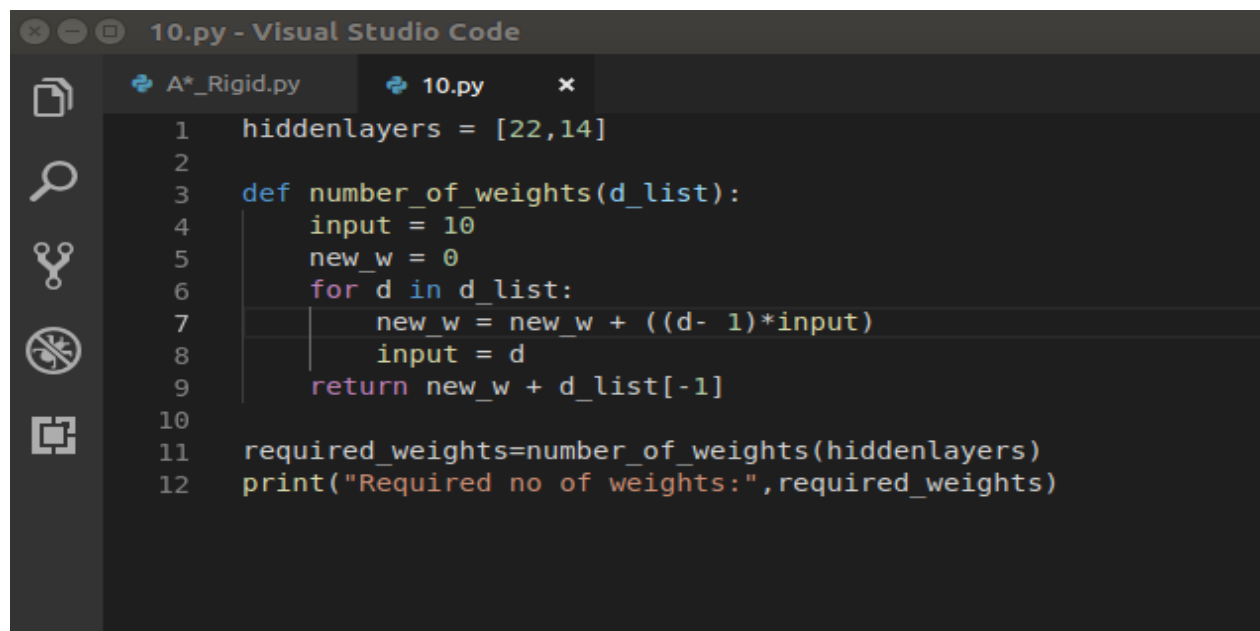
ANSWER:[e] Maximum no of weights will occur when we have a large no of nodes in a single layer. We know that when we transitioning from one layer to the next requires the following no of weights which needs to be maximized.

$$\rho = d^{l-1}x(d^{(l)} - 1) \quad (9)$$

To maximize this we can differentiate this wrt d, and we see that the maximum occurs when  $d(l-1) = d(l)-1$  as this will give us a square term. Intuitively changing the no of hidden layers, with a little trial and error we get the following no of layers as shown in the code below.

```
koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework6$ python3 10.py
Required no of weights: 510
koyal@koyal-XPS-15-9570:~/Desktop/Machine Learning/Homework6$
```

Figure 5: Code of Ques 10



```
10.py - Visual Studio Code
A*_Rigid.py 10.py x
1 hiddenlayers = [22,14]
2
3 def number_of_weights(d_list):
4     input = 10
5     new_w = 0
6     for d in d_list:
7         new_w = new_w + ((d- 1)*input)
8         input = d
9     return new_w + d_list[-1]
10
11 required_weights=number_of_weights(hiddenlayers)
12 print("Required no of weights:",required_weights)
```

Figure 6: Term. Screenshot of Ques 10



Note: Codes for each of the questions is attached below for reference.

## 2 Codes 1(Question 2 from Homework 6)

```
#
# Copyright 2019 Koyal Bhartia
# @file Regularization with Weight Decay.py
# @author Koyal Bhartia
# @date 03/06/2019
# @version 1.0
#
# @brief This is the code for Question 2 of Homework 6 from "Learning from Data"
#
# @Description This code applies Linear Regression for Classification on a
# Nonlinear function. The function is simulated with noise; and calculation of
# Ein and Eout is made using the weights of a Nonlinear feature vector
#
#Import statments
import argparse
import numpy as np
import os, sys
from numpy import linalg as LA
import math
import pickle
import matplotlib.pyplot as plt
import random

#Fetch data from given files
def fetch_data(path):
    #in_path = "in.dta.txt"
    data = np.loadtxt(path)
    N=len(data)
    X_data=np.column_stack((np.ones(N),np.ones(N)))
    fx=np.zeros((N))
    X_data[:,0] = data[:,0]
    X_data[:,1] = data[:,1]
    fx[:]=data[:,2]
    return X_data, fx

def signCheck(num):
    if (num>0):
        return 1
    else:
```

```
return -1
```

*#Calculation of Ein using Linear Regression*

```
def LinRegression_in():
    misclassify=0
    X_data,fx=fetch_data("in.dta.txt")
    N=len(X_data)
    X_new=np.column_stack((np.ones(N),X_data[:,0],X_data[:,1],np.ones(N),np.ones(N),np.ones(N)))
    for i in range(N):
        X_new[i,3]=X_data[i,0]*X_data[i,0]
        X_new[i,4]=X_data[i,1]*X_data[i,1]
        X_new[i,5]=X_data[i,0]*X_data[i,1]
        X_new[i,6]=np.abs(X_data[i,0]-X_data[i,1])
        X_new[i,7]=np.abs(X_data[i,0]+X_data[i,1])

    w0=np.matmul(np.transpose(X_new),X_new)
    w1=LA.inv(w0)
    w2=np.matmul(w1,np.transpose(X_new))
    w3=np.matmul(w2,fx)
    w4=np.array([w3])
    w=np.transpose(w4)
    print("w:",w)
    for i in range(N):
        mul1=np.matmul(X_new,w)
        y_test=signCheck(float(mul1[i,0]))
        if(y_test!=fx[i]):
            misclassify+=1
    return misclassify,N,w
```

*#calcualtion of Eout:*

```
def LinRegression_out(w):
    misclassify=0
    X_data,fx=fetch_data("out.dta.txt")
    N=len(X_data)
    X_new=np.column_stack((np.ones(N),X_data[:,0],X_data[:,1],np.ones(N),np.ones(N),np.ones(N)))
    for i in range(N):
        X_new[i,3]=X_data[i,0]*X_data[i,0]
        X_new[i,4]=X_data[i,1]*X_data[i,1]
        X_new[i,5]=X_data[i,0]*X_data[i,1]
        X_new[i,6]=np.abs(X_data[i,0]-X_data[i,1])
        X_new[i,7]=np.abs(X_data[i,0]+X_data[i,1])
```

```

    for i in range(N):
        mul1=np.matmul(X_new,w)
        y_test=signCheck(float(mul1[i,0]))
        if(y_test!=fx[i]):
            misclassify+=1
    return misclassify,N

misclassify_in ,N_train,w=LinRegression_in()
misclassify_out ,N_test=LinRegression_out(w)
E_in=misclassify_in/N_train
E_out=misclassify_out/N_test
print("E_in:",E_in)
print("E_out:",E_out)

```

### 3 Codes 1(Question 3,4,5,6 from Homework 6)

```

#
# Copyright 2019 Koyal Bhartia
# @file Regularization with Weight Decay.py
# @author Koyal Bhartia
# @date 03/06/2019
# @version 1.0
#
# @brief This is the code for Question 2 of Homework 6 from "Learning from Data"
#
# @Description This code applies Linear Regression for Classification on a
# Nonlinear function. The function is simulated with noise; and calcuation of
# Ein and Eout is made using the weights of a Nonlinear feature vector
#
#Import statments
import argparse
import numpy as np
import os, sys
from numpy import linalg as LA
import math
import pickle
import matplotlib.pyplot as plt
import random

#Fetch data from given files
def fetch_data(path):
    #in_path = "in.dta.txt"

```

```

data = np.loadtxt(path)
N=len(data)
X_data=np.column_stack((np.ones(N),np.ones(N)))
fx=np.zeros((N))
X_data[:,0] = data[:,0]
X_data[:,1] = data[:,1]
fx[:]=data[:,2]
return X_data, fx

def signCheck(num):
    if (num>0):
        return 1
    else:
        return -1

def LinRegression_in():
    misclassify=0
    X_data,fx=fetch_data("in.dta.txt")
    N=len(X_data)
    lambdaa=math.pow(10,k)*np.identity(8)
    X_new=np.column_stack((np.ones(N),X_data[:,0],X_data[:,1],np.ones(N),np.ones(N),np.ones(N),np.ones(N),np.ones(N)))
    for i in range(N):
        X_new[i,3]=X_data[i,0]*X_data[i,0]
        X_new[i,4]=X_data[i,1]*X_data[i,1]
        X_new[i,5]=X_data[i,0]*X_data[i,1]
        X_new[i,6]=np.abs(X_data[i,0]-X_data[i,1])
        X_new[i,7]=np.abs(X_data[i,0]+X_data[i,1])

    w0=np.matmul(np.transpose(X_new),X_new)
    w1=LA.inv(w0+lambdaa)
    w2=np.matmul(w1,np.transpose(X_new))
    w3=np.matmul(w2,fx)
    w4=np.array([w3])
    w=np.transpose(w4)
    print("w:",w)

    for i in range(N):
        mul1=np.matmul(X_new,w)
        y_test=signCheck(float(mul1[i,0]))
        if (y_test!=fx[i]):
            misclassify+=1
    return misclassify,N,w

```

*#calcualtion of Eout using the weights using non linear feature vector*

```
def LinRegression_out(w):
    misclassify=0
    X_data,fx=fetch_data("out.dta.txt")
    N=len(X_data)
    X_new=np.column_stack((np.ones(N),X_data[:,0],X_data[:,1],np.ones(N),np.ones(N),np.ones(N)))
    for i in range(N):
        X_new[i,3]=X_data[i,0]*X_data[i,0]
        X_new[i,4]=X_data[i,1]*X_data[i,1]
        X_new[i,5]=X_data[i,0]*X_data[i,1]
        X_new[i,6]=np.abs(X_data[i,0]-X_data[i,1])
        X_new[i,7]=np.abs(X_data[i,0]+X_data[i,1])

    for i in range(N):
        mul1=np.matmul(X_new,w)
        y_test=signCheck(float(mul1[i,0]))
        if(y_test!=fx[i]):
            misclassify+=1
    return misclassify,N
```

```
k=int(input("Input value k for the lambda term:"))
print("Regularization with k=",k)
misclassify_in,N_train,w=LinRegression_in()
misclassify_out,N_test=LinRegression_out(w)
E_in=misclassify_in/N_train
E_out=misclassify_out/N_test
print("E_in:",E_in)
print("E_out:",E_out)
```