# CS 156a - Problem Set 6

Samuel Patrone, 2140749

November 7, 2022

The following notebook is publicly available at the following link.

## Contents

## Problem 1

**Answer: [b] In general, deterministic noise will increase.**

**Derivation:**

Given that we are using a less complex hypothesis in $\mathcal{H}' \subset \mathcal{H}$, we expect the best fit $g' \in \mathcal{H}'$ in this less complex hypothesis space to have an higher deterministic noise since there will be more of the target function that cannot be captured by it. In more mathematical terms, the deterministic noise is represented by the bias, which naturally increases for less complex hypothesis. The bias can be seen as the asymptotic value at which both $E_{in}$ and $E_{out}$ converge for $N \to \infty$ (in the case of zero stochastic noise) which is lower for more complex hypothesis.

## Problem 2

**Answer: [a]** 0.03, 0.08

**Code:**

[72]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#import data

training_set=pd.read_csv('in.dta',header=None,delim_whitespace=True)
testing_set=pd.read_csv('out.dta',header=None,delim_whitespace=True)

train_pts=training_set[[0, 1]].to_numpy()
train_y=training_set[2].to_numpy()

test_pts=testing_set[[0, 1]].to_numpy()
test_y=testing_set[2].to_numpy()


def color_pts(y):
    #green is +1, red is -1
    col=[]
    for i in range(len(y)):
        if(y[i]>0): col.append('green')
        else: col.append('red')
    return col

def plot_pts(pts,y):
    col=color_pts(y)
    plt.scatter(pts[:,0],pts[:,1],color=col)
    #plt.xlim([-1, 1])
    #plt.ylim([-1, 1])
```

```
    plt.legend()
    plt.show()

#non-linear transformation

def transform(pts):
    res=[]
    for i in range(len(pts)):
        x1=pts[i][0]
        x2=pts[i][1]
        res.append([1,x1,x2,x1**2,x2**2,x1*x2,np.abs(x1-x2),np.abs(x1+x2)])
    return np.array(res)

def lin_reg_w(X,y):
    return np.dot(np.linalg.pinv(X),y)

def h(pts,w):
    return np.sign(np.dot(w,pts.T))

def lin_reg(train_pts,train_y,test_pts,test_y,res=True):
    N_train=len(train_pts)
    N_test=len(test_pts)

    w=lin_reg_w(train_pts,train_y)

    #Ein computation
    gin=h(train_pts,w)
    testgin=(gin==train_y)
    Ein=len(np.where(testgin==False)[0])/N_train

    #Eout computation
    gout=h(test_pts,w)
    testgout=(gout==test_y)
    Eout=len(np.where(testgout==False)[0])/N_test

    #print results
    if(res==True):
        print(f'Linear Regression results:\nEin={Ein:.2f}\nEout={Eout:.2f}')

    return w,Ein,Eout
```

[151]:
```
train_pts_transf=transform(train_pts)
test_pts_transf=transform(test_pts)

ex2=lin_reg(train_pts_transf,train_y,test_pts_transf,test_y)
```

```
Linear Regression results:
Ein=0.03
```

```
Eout=0.08
```

## Problems 3-6

**Answers: [d]** $[0.03, 0.08]$**, [e]** $[0.4, 0.4]$**, [d]** $-1$**, [b]** $0.06$

**Code:**

[121]:
```
def lin_reg_w_lam(X,y,lam):
    pinv_decay=np.dot(np.linalg.inv(np.dot(X.T,X)+lam*np.identity(len(X.T))),X.T)
    return np.dot(pinv_decay,y)

def lin_reg_wdecay(train_pts,train_y,test_pts,test_y,lam,res=True):
    N_train=len(train_pts)
    N_test=len(test_pts)

    w=lin_reg_w_lam(train_pts,train_y,lam)

    #Ein computation
    gin=h(train_pts,w)
    testgin=(gin==train_y)
    Ein=len(np.where(testgin==False)[0])/N_train

    #Eout computation
    gout=h(test_pts,w)
    testgout=(gout==test_y)
    Eout=len(np.where(testgout==False)[0])/N_test

    #print results
    if(res==True):
        print(f'Linear Regression results with k={np.log10(lam):.0f}:\nEin={Ein:.
↪2f}\nEout={Eout:.2f}')

    return w,Ein,Eout
```

[165]:
```
k=[4,3,2,1,0,-1,-2,-3,-4]
Ein=[]
Eout=[]
wnorm=[]

for i in k:
    ␣
↪ex5,Ein5,Eout5=lin_reg_wdecay(train_pts_transf,train_y,test_pts_transf,test_y,10**(i),res=Fal
    Ein.append(Ein5)
    Eout.append(Eout5)
    wnorm.append(np.dot(ex5,ex5))
```

```
pd.options.display.float_format = '{:,.2f}'.format
pd.DataFrame(list(zip(k, Ein,Eout,wnorm)), columns =['k', 'Ein','Eout','w.w'])
```

[165]:

| | k | Ein | Eout | w.w |
|---|---|---|---|---|
| 0 | 4 | 0.43 | 0.45 | 0.00 |
| 1 | 3 | 0.37 | 0.44 | 0.00 |
| 2 | 2 | 0.20 | 0.23 | 0.03 |
| 3 | 1 | 0.06 | 0.12 | 0.50 |
| 4 | 0 | 0.00 | 0.09 | 2.52 |
| 5 | -1 | 0.03 | 0.06 | 15.37 |
| 6 | -2 | 0.03 | 0.08 | 30.39 |
| 7 | -3 | 0.03 | 0.08 | 33.40 |
| 8 | -4 | 0.03 | 0.08 | 33.74 |

## Problem 7

**Answer: [c]** $\mathcal{H}(10,0,3) \cap \mathcal{H}(10,0,4) = \mathcal{H}_2$

**Derivation:**

We first observe that the following identity holds:

$$\mathcal{H}(Q, C = 0, Q_0) = \mathcal{H}_{Q_0 - 1}\,. \tag{1}$$

where $Q \geq Q_0$.

Furthermore, by definition we have that:

$$\mathcal{H}_{N-1} \subset \mathcal{H}_N\,. \tag{2}$$

Therefore,

$$\mathcal{H}(10,0,3) \cap \mathcal{H}(10,0,4) = \mathcal{H}_2 \cap \mathcal{H}_3 = \mathcal{H}_2\,. \tag{3}$$

## Problem 8

**Answer: [d]** 45

**Derivation:**

One single iteration of the backpropagation algorithm using one data point starts with the forward propagation, i.e. the computation of the neural network hypothesis which is obtained by computing the output vector at each layer

$$\mathbf{x}^{(l)} = \begin{pmatrix} 1 \\ \theta(\mathbf{s}^{(l)}) \end{pmatrix} \tag{4}$$

where $\mathbf{s}^{(l)}$ is a vector whose component are

$$s_i^{(l)} = \sum_{j=0}^{d^{(l-1)}} w_{ij}^{(l)} x_j^{(l-1)} . \tag{5}$$

For each layer $l$, there are $d^{(l)}(d^{(l-1)}+1)$ multiplications to do, which is exactly the number of weights between the layer $l-1$ and the layer $l$. Hence, for a complete forward propagation, we have $N_w$ total multiplications, where

$$N_w = \sum_{l=1}^{L} d^{(l)}(d^{(l-1)}+1) . \tag{6}$$

In the stochastic gradient descent algorithm, the weights are updated by taking a step in the negative gradient direction. To compute the gradient, we have to take the derivatives of the error function with respect to each single weight $w_{ij}^{(l)}$, which is given by

$$\frac{\partial e}{\partial w_{ij}^{(l)}} = x_i^{(l-1)} \delta_j^{(l)} . \tag{7}$$

There are as many derivatives as weights, so we have other $N_w$ operations. Finally, the sensitivity $\delta_j^{(l)}$ can be computed using the backpropagation algorithm by recursion with the following formula

$$\delta_j^{(l)} = \theta'(s_j^{(l)}) \sum_{k=1}^{d^{(l+1)}} w_{jk}^{(l+1)} \delta_k^{(l+1)} . \tag{8}$$

where $\delta^{(L)} = 2(x^{(L)} - y)\theta'(s^{(L)})$.

There are $L-1$ sensitivities to compute, for a total number of additional operations equal to

$$N_b = \sum_{l=1}^{L-1} d^{(l)} d^{(l+1)} . \tag{9}$$

Hence, the total number of multiplications needed to carry out a single iteration of backpropagation is $N_{tot} = 2N_w + N_b$, being $N_w$ the number of weights of the network and $N_b$ the number of operations needed to compute the sensitivities.

In the example given, $d^{(0)} = 5$, $d^{(1)} = 3$, $d^{(2)} = 1$. Hence, $N_w = 22$, $N_b = 3$ and $N_{tot} = 47$.

## Problem 9

**Answer: [a]** 46

**Derivation:**

To minimize the number of weights, we want to reduce the number of connections. In order to achieve that, we consider the case in which the 36 units are equally distributed in 18 hidden layers

of 2 units each.

The following code allows as to compute the number of weights in this case, which turns out to be 46.

[195]:
```
def nweights(d):
    w=0
    for i in range(len(d)-1):
        w+=(d[i]+1)*d[i+1]
    return w

d=[9]
for i in range(19):
    d.append(1)

print(nweights(d))
```

```
46
```

## Problem 10

**Answer: [e]** 510

**Derivation:**

To solve this problem, we will analytically compute the number of weights for all the possible configurations of one, two or three hidden layers.

For one inner layer of 36 units, $N^{(1)}_{max} = 10 \times 35 + 36 \times 1 = 386$.

For two inner layers, let $d^{(1)} = x$ be the dimension of the first layer (i.e. the number of units minus one). The number of weights is then an analytic function of $x$, specifically:

$$N^{(2)}(x) = 10x + (x+1)(36-x-2) + (36-x-1) = 69 + 42x - x^2\,. \tag{10}$$

This function is maximized for $x_{max} = 21$, where $N^{(2)}(x_{max}) = N^{(2)}_{max} = 510$, which correspond to a first layer of 22 units followed by a second layer of 14 units.

For three layers, let $d^{(1)} = x$ and $d^{(2)} = y$ be the dimension of the first and second layer respectively (i.e. the number of units minus one). The number of weights now is the following two-dimensional surface:

$$N^{(3)}(x) = 10x + (x+1)y + (y+1)(36-x-y-3) + (36-x-y-2) = 67 + 8x + 32y - y^2\,. \tag{11}$$

Using the script below, we obtain that the function is maximized for $x_{max} = 20$ and $y_{max} = 12$, giving $N^{(3)}_{max} = 467$.

Hence, the maximum number of weights are $N^{(2)}_{max} = 510$.

[194]:
```
Nmax=0
dmax=[]

for i in range(31):
    for j in range(33-i):
        k=36-(i+j+2)-1
        d=[9,i,j,k,1]
        if(nweights(d)>Nmax):
            Nmax=nweights(d)
            dmax=d

print(f'N3max={Nmax} found at d={dmax}!')
```

```
N3max=467 found at d=[9, 20, 12, 1, 1]!
```

[ ]: