**ML Techniques HW6**

**1. D**

公式是：  
δ(l)j=d(l+1)∑k=1δ(l+1)k⋅w(l+1)jk⋅tanh′(s(l)j)δj(l)=∑k=1d(l+1)δk(l+1)⋅wjk(l+1)⋅tanh′⁡(sj(l))  
注意那些編號0的節點不用算δδ，也不需要在∑∑裡面。因為∑∑裡面只出現一次δ⋅wδ⋅w，我們就計算總共∑∑需要加幾次即可。所以第ll層需要計算加d(l)⋅d(l+1)d(l)⋅d(l+1)次。  
總共需要加4⋅5+5⋅6+6⋅1=564⋅5+5⋅6+6⋅1=56次。

**2. D**

事實上，我是硬爆處理的：

#include <bits/stdc++.h>

using namespace std;

int update(vector<int> &d\_vec){

vector<int>::reverse\_iterator riter = d\_vec.rbegin();

while(riter != d\_vec.rend() && \*riter == 0) ++riter;

if(riter == d\_vec.rend() || riter+1 == d\_vec.rend()) return 1;

int tmp = \*riter;

(\*(riter+1))++;

\*riter = 0;

\*d\_vec.rbegin() = tmp - 1;

return 0;

}

int main(){

vector<int> d\_vec = {48};

int c = 0;

int max\_n = 0;

while(d\_vec.back() >= 0){

while(1){

int cur = (d\_vec.front() + 1) \* 19 + (d\_vec.back() + 1) \* 3

+ 53 - d\_vec.size();

for(int i = 0; i != d\_vec.size() - 1; ++i){

cur += (d\_vec[i] + 1) \* (d\_vec[i+1] + 1);

}

max\_n = max(max\_n, cur);

if(update(d\_vec)) break;

}

d\_vec.push\_back(d\_vec.front() - 2);

d\_vec.front() = 0;

cout << "cur max: " << max\_n << ", next last number: "

<< d\_vec.back() << endl;

}

return 0;

}

$ ./main

cur max: 1130, next last number: 46

cur max: 1219, next last number: 44

cur max: 1219, next last number: 42

cur max: 1219, next last number: 40

--(omit middle outputs)--

cur max: 1219, next last number: 4

cur max: 1219, next last number: 2

cur max: 1219, next last number: 0

cur max: 1219, next last number: -2

這個程式會對所有LL窮舉所有dd的可能性。這裡假設d≥1d≥1。  
注意如果真的要跑的話，時間大約是均攤O(p)O(p)，pp是所有可能的組合數，大約有p≈∑25i=1Hi50−2i=7,778,742,049p≈∑i=125H50−2ii=7,778,742,049種組合，會跑很久。  
他窮舉的原理是update這個函數可以對一個從vec = {0,0,...,K}開始一直窮舉所有sum(vec)=K的所有非負整數解。可以對一個像是vec = {0,0,0,5}的std::vector<int>print出每次update後的d\_vec看，直到update回傳1告知termination。  
話說為了保證d≥1d≥1，在d\_vec裡的數字要加1才會是真正的d。

#define all(container) (container).begin(), (container).end()

ostream &operator<<(ostream &os, const vector<int> &vec){

os << "{ ";

for\_each(all(vec), [&](int i){os << i << " ";});

return os << "}";

}

**3. D**

首先，為了打字方便，我們這邊所有sksk都代表s(L)ksk(L)。  
我們先做∂lnqi∂sk∂ln⁡qi∂sk：

* i=ki=k：  
  ∂lnqk∂sk=∂lnqk∂qk⋅∂qk∂sk=1qk⋅∂exp(sk)exp(sk)+α∂sk(α:=∑j∈[1,K],j≠kexp(sj))=∑Kj=1exp(sj)exp(sk)⋅exp(sk)⋅α(∑Kj=1exp(sj))2=1−qk∂ln⁡qk∂sk=∂ln⁡qk∂qk⋅∂qk∂sk=1qk⋅∂exp⁡(sk)exp⁡(sk)+α∂sk(α:=∑j∈[1,K],j≠kexp⁡(sj))=∑j=1Kexp⁡(sj)exp⁡(sk)⋅exp⁡(sk)⋅α(∑j=1Kexp⁡(sj))2=1−qk
* i≠ki≠k：  
  ∂lnqi∂sk=∂lnqi∂qi⋅∂qi∂sk=1qi⋅∂exp(si)exp(sk)+α∂sk(α:=∑j∈[1,K],j≠kexp(sj))=∑Kj=1exp(sj)exp(si)⋅⎛⎜
* ⎜⎝−exp(sk)exp(si)(∑Kj=1exp(sj))2⎞⎟
* ⎟⎠=−qk∂ln⁡qi∂sk=∂ln⁡qi∂qi⋅∂qi∂sk=1qi⋅∂exp⁡(si)exp⁡(sk)+α∂sk(α:=∑j∈[1,K],j≠kexp⁡(sj))=∑j=1Kexp⁡(sj)exp⁡(si)⋅(−exp⁡(sk)exp⁡(si)(∑j=1Kexp⁡(sj))2)=−qk

所以我們有：  
∂err∂sk=−K∑i=1vj∂lnqi∂sk=qkL∑j=1vj−vk=qk−vk∂err∂sk=−∑i=1Kvj∂ln⁡qi∂sk=qk∑j=1Lvj−vk=qk−vk  
注意∑vj=1∑vj=1。所以選D。

**4. A**

我們就真的來做第一遍：  
W(1)=⎡⎢

⎢

⎢

⎢

⎢

⎢⎣0000000000000000000000000⎤⎥

⎥

⎥

⎥

⎥

⎥⎦,W(2)=⎡⎢

⎢

⎢

⎢

⎢

⎢⎣00000⎤⎥

⎥

⎥

⎥

⎥

⎥⎦,x(0)=⎡⎢

⎢

⎢

⎢

⎢

⎢⎣11000⎤⎥

⎥

⎥

⎥

⎥

⎥⎦⇒s(1)=(W(1))Tx(0)=⎡⎢

⎢

⎢

⎢

⎢

⎢⎣00000⎤⎥

⎥

⎥

⎥

⎥

⎥⎦,x(1)=tanh(s(1))=⎡⎢

⎢

⎢

⎢

⎢

⎢⎣00000⎤⎥

⎥

⎥

⎥

⎥

⎥⎦s(2)=(W(2))Tx(1)=[0],x(2)=tanh(s(2))=[0]⇒δ(2)=−2(1−[0])=[−2],δ(1)=δ(2)W(2)tanh′(s(2))=⎡⎢

⎢

⎢

⎢

⎢

⎢⎣00000⎤⎥

⎥

⎥

⎥

⎥

⎥⎦⇒W(1)←W(1)+x(0)(δ(1))T=W(1)+⎡⎢

⎢

⎢

⎢

⎢

⎢⎣11000⎤⎥

⎥

⎥

⎥

⎥

⎥⎦[00000]=W(1)W(2)←W(2)+x(1)(δ(2))T=W(2)+[0][−2]=W(2)W(1)=[0000000000000000000000000],W(2)=[00000],x(0)=[11000]⇒s(1)=(W(1))Tx(0)=[00000],x(1)=tanh⁡(s(1))=[00000]s(2)=(W(2))Tx(1)=[0],x(2)=tanh⁡(s(2))=[0]⇒δ(2)=−2(1−[0])=[−2],δ(1)=δ(2)W(2)tanh′⁡(s(2))=[00000]⇒W(1)←W(1)+x(0)(δ(1))T=W(1)+[11000][00000]=W(1)W(2)←W(2)+x(1)(δ(2))T=W(2)+[0][−2]=W(2)  
中間過程有一點省略，但是基本上就是照做212\_handout.pdf的第16頁的結果。可發現所有parameter WW根本沒變，我們又拿同一筆資料，所以可知不管做幾次，參數全部都是0。

**5. E**

V=⎡⎢⎣||...|v1v2...vN||...|⎤⎥⎦=[22...2],rm=⎡⎢

⎢

⎢

⎢⎣r1mr2m⋮rNm⎤⎥

⎥

⎥

⎥⎦Let X=VT⇒w∗m=(XTX)−1XTy=(VVT)−1Vrm=∑Ni=1rim2NV=[||...|v1v2...vN||...|]=[22...2],rm=[r1mr2m⋮rNm]Let X=VT⇒wm∗=(XTX)−1XTy=(VVT)−1Vrm=∑i=1Nrim2N  
所以E。

**6. B**

我們需要得到∇amerr∇amerr：  
∇amerr=∂((rnm−wTmvn−am−bn)2)∂am=−2(rnm−wTmvn−am−bn)∇amerr=∂((rnm−wmTvn−am−bn)2)∂am=−2(rnm−wmTvn−am−bn)  
所以更新函數應該是am←am−η2∇amerr=am+η(rnm−wTmvn−am−bn)=(1−η)am+η(rnm−wTmvn−bn)am←am−η2∇amerr=am+η(rnm−wmTvn−am−bn)=(1−η)am+η(rnm−wmTvn−bn)

**7. D**

GG分錯代表小gg們至少兩個要分錯。假設我們把全體資料對一個gg來分成兩區(即 w.r.t. gg)：一區是他分錯的，一區是他分對的。所以這三個gg分錯的區域，至少有兩個區域交集的區域加起來至少要0.20。只有D做得到，如果要說出一種可能的話就是：

* g3g3有0.24分錯。
* g2g2分錯的區域跟g3g3分錯區域交集。
* g1g1分錯的區域有0.04跟g3g3分錯區域交集，但是不碰到g2g2分錯的區域。

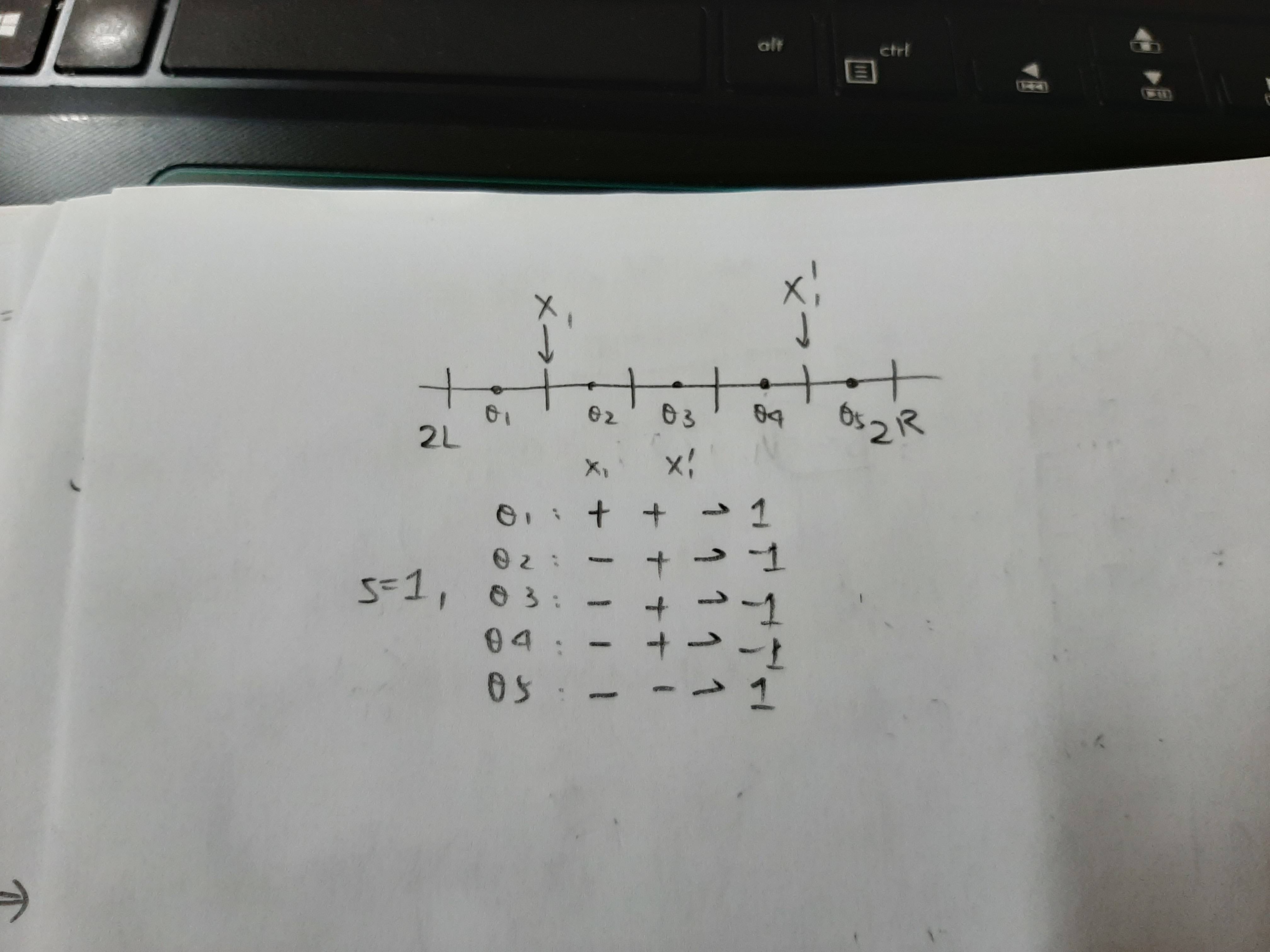
**8. C**

所以每個gg分錯的期望值是0.4，為了讓GG分錯，必須有3,4或5個gg分錯。  
所以：  
Eout(G)=(C35⋅(0.4)3(0.6)2)+(C45⋅(0.4)4(0.6)1)+(C55⋅(0.4)5(0.6)0)=0.31744Eout(G)=(C53⋅(0.4)3(0.6)2)+(C54⋅(0.4)4(0.6)1)+(C55⋅(0.4)5(0.6)0)=0.31744

**9. B**

(1−1N)0.5N=((1−1N)N)0.5=√1e=0.6065(1−1N)0.5N=((1−1N)N)0.5=1e=0.6065

**10. E**

給定其中一項gs,i,θgs,i,θ，有：  
gs,i,θ(x)⋅gs,i,θ(x′)=s2⋅sign(x)⋅sign(x′)=sign(x)⋅sign(x′)gs,i,θ(x)⋅gs,i,θ(x′)=s2⋅sign(x)⋅sign(x′)=sign(x)⋅sign(x′)  
所以s=+1s=+1和s=−1s=−1算出來的和應該不會有差，等等討論∑∀s∑∀s的時候就乘以2即可。  
我們舉個例子：  


可以發現，∑∀θg(x)⋅g(x′)∑∀θg(x)⋅g(x′)的值為「所有奇數數量 - (兩者中的奇數數量 x 2)」。所以：  
∑∀θgs,i,θ(x)⋅gs,i,θ(x′)=2R−2L2−|xi−x′i|2⋅2=(R−L)−|xi−x′i|∑∀θgs,i,θ(x)⋅gs,i,θ(x′)=2R−2L2−|xi−xi′|2⋅2=(R−L)−|xi−xi′|  
對所有s,ds,d做和：  
∑∀s∑∀d∑∀θgs,i,θ(x)⋅gs,i,θ(x′)=2∑∀d(R−L)−|xi−x′i|=2d(R−L)−2∥xi−x′i∥1∑∀s∑∀d∑∀θgs,i,θ(x)⋅gs,i,θ(x′)=2∑∀d(R−L)−|xi−xi′|=2d(R−L)−2‖xi−xi′‖1  
沒有答案，選E。

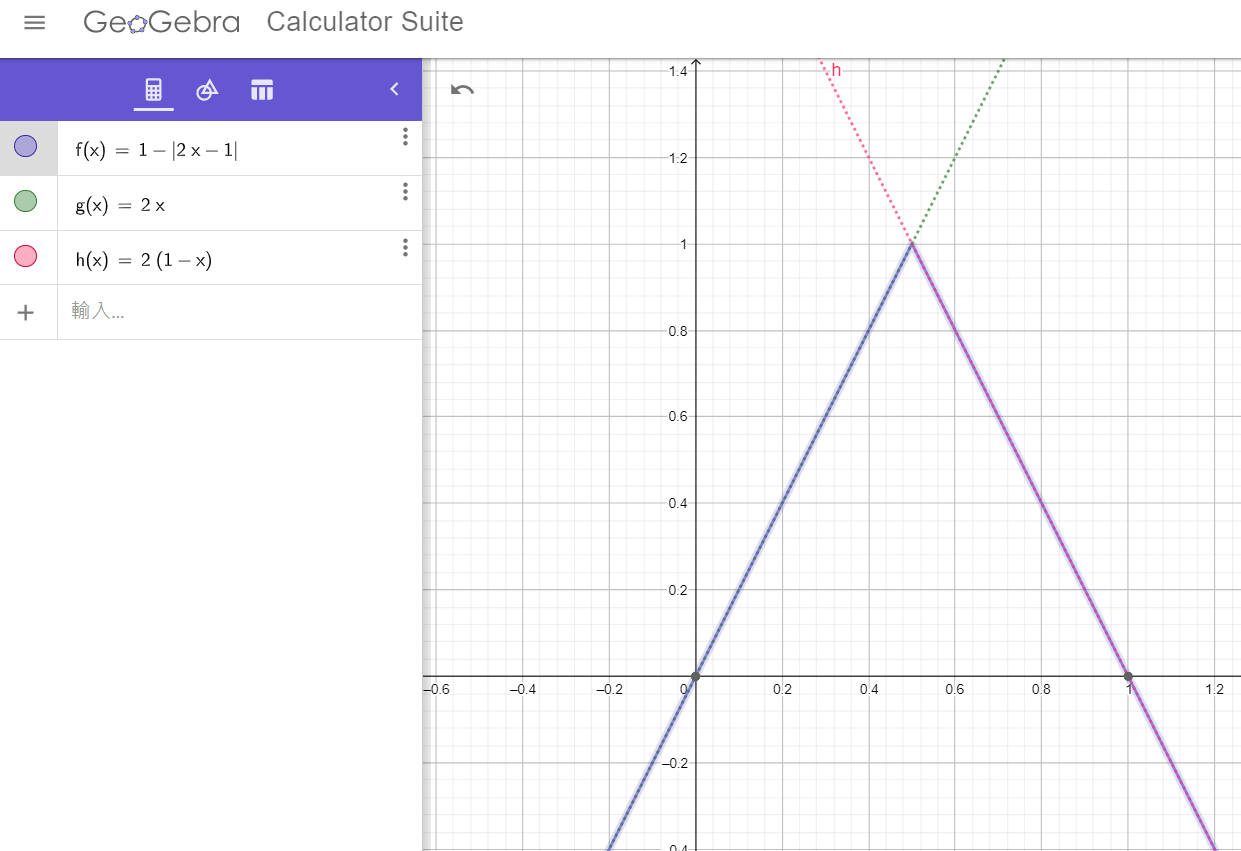
**11. A**

u(2)+←u(1)+⋅⧫tu(2)−←u(1)−/⧫tu(2)+u(2)−=⧫2t=19u+(2)←u+(1)⋅⧫tu−(2)←u−(1)/⧫tu+(2)u−(2)=⧫t2=19

**12. D**

我們對Ut−1Ut−1簡化：  
Ut+1=1NN∑n=1exp(−ynt∑τ=1ατgτ(xn))=1NN∑n=1exp(−ynt−1∑τ=1ατgτ(xn)−ynαtgt(xn))=1NN∑n=1exp(−ynt−1∑τ=1ατgτ(xn))⋅exp(−ynαtgt(xn))(1)=N∑n=1u(t−1)n⋅exp(−ynαtgt(xn))=N∑n=1u(t−1)n⋅{exp(−αt) if yn=gt(xn)exp(αt) if yn≠gt(xn)(2)=(N∑n=1u(t−1)n)⋅((1−ϵt)(exp(−αt))+(ϵt)(exp(−αt)))Ut+1=1N∑n=1Nexp⁡(−yn∑τ=1tατgτ(xn))=1N∑n=1Nexp⁡(−yn∑τ=1t−1ατgτ(xn)−ynαtgt(xn))=1N∑n=1Nexp⁡(−yn∑τ=1t−1ατgτ(xn))⋅exp⁡(−ynαtgt(xn))=(1)∑n=1Nun(t−1)⋅exp⁡(−ynαtgt(xn))=∑n=1Nun(t−1)⋅{exp⁡(−αt) if yn=gt(xn)exp⁡(αt) if yn≠gt(xn)=(2)(∑n=1Nun(t−1))⋅((1−ϵt)(exp⁡(−αt))+(ϵt)(exp⁡(−αt)))  
其中(1)(1)用了式子u(T)n=1Nexp(−yn∑Tt=1αtgt(xn))un(T)=1Nexp⁡(−yn∑t=1Tαtgt(xn))，來自211\_handout.pdf page.7。(2)(2)用了同一份pdf中page.12下半部的推導。  
接下來我們推導Ut+1UtUt+1Ut：  
Ut+1Ut=(∑Nn=1u(t−1)n)⋅((1−ϵt)(exp(−αt))+(ϵt)(exp(−αt)))(∑Nn=1u(t−1)n)=(1−ϵt)(exp(−αt))+(ϵt)(exp(−αt))(3)=(1−ϵt)√ϵt1−ϵt+(ϵt)√1−ϵtϵt=2⋅√ϵt(1−ϵt)Ut+1Ut=(∑n=1Nun(t−1))⋅((1−ϵt)(exp⁡(−αt))+(ϵt)(exp⁡(−αt)))(∑n=1Nun(t−1))=(1−ϵt)(exp⁡(−αt))+(ϵt)(exp⁡(−αt))=(3)(1−ϵt)ϵt1−ϵt+(ϵt)1−ϵtϵt=2⋅ϵt(1−ϵt)  
其中(3)(3)使用了αt:=ln(⧫t):=ln(√1−ϵtϵt)αt:=ln⁡(⧫t):=ln⁡(1−ϵtϵt)。  
接下來，我們發現：  
√ϵt(1−ϵt)≤√ϵ(1−ϵ)≤12exp(−2(12−ϵ)2)ϵt(1−ϵt)≤ϵ(1−ϵ)≤12exp⁡(−2(12−ϵ)2)  
因為f(x):=√x(1−x)f(x):=x(1−x)在x∈(0,0.5)x∈(0,0.5)是遞增的，代表0<ϵt≤ϵ<0.5⇒√ϵt(1−ϵt)≤√ϵ(1−ϵ)0<ϵt≤ϵ<0.5⇒ϵt(1−ϵt)≤ϵ(1−ϵ)。後面的式子從提示來。  
所以我們終於能推導出UT+1UT+1的上限：  
UT+1=UT+1UTUTUT−1⋯U2U1⋅U1(4)=UT+1UTUTUT−1⋯U2U1=T∏t=1Ut+1Ut=T∏t=12⋅√ϵt(1−ϵt)≤T∏t=12⋅12exp(−2(12−ϵ)2)=T∏t=1exp(−2(12−ϵ)2)=exp(−2T(12−ϵ)2)UT+1=UT+1UTUTUT−1⋯U2U1⋅U1=(4)UT+1UTUTUT−1⋯U2U1=∏t=1TUt+1Ut=∏t=1T2⋅ϵt(1−ϵt)≤∏t=1T2⋅12exp⁡(−2(12−ϵ)2)=∏t=1Texp⁡(−2(12−ϵ)2)=exp⁡(−2T(12−ϵ)2)  
在(4)(4)，因為∀n:u(1)n:=1N∀n:un(1):=1N，所以U1=∑Nn=1u(1)n=1U1=∑n=1Nun(1)=1，可以直接忽略最後一項。  
所以：  
Ein(GT)≤UT+1≤exp(−2T(12−ϵ)2)Ein(GT)≤UT+1≤exp⁡(−2T(12−ϵ)2)  
答案為D。

**13. D**

我們直接看( D )選項。根據絕對值，可知此函數在μ+=0.5μ+=0.5時有最大值1，所以他normalized後還是一樣。  
可以畫圖知1−|μ+−μ−|=2min(μ+,μ−)1−|μ+−μ−|=2min(μ+,μ−)，得證。  
  
紫色的線即紅色和綠色線的逐點最小值。

**14. C**

首先，因為我們寫出了一個class CnRT，不過為了理解起見，我們還是先寫出spec，再看code：  
class CnRT有幾個函式：

* CnRT.fit(data): 給予N×(d+1)N×(d+1)的np.ndarray，其中前dd行是x，最後一行才是y。使這個class去做C&RT，得到一顆樹。
* CnRT.transform(x): 給予N×dN×d的np.ndarray，基本上是data去掉最後的y列。得出一個長度NN的一維np.ndarray，表示預測結果。

裡面常常用到這兩個函數，以下舉例：

* x[np.where(x[:, 3] == 1)]：取出所有第三column為1的rows。<https://stackoverflow.com/questions/23359886/selecting-rows-in-numpy-ndarray-based-on-the-value-of-two-columns>  
  隨版本或是array不同(如arr.shape = (10,)和arr.shape = (10,1)等等的差別)，回傳值可能會有所不同。請注意。
* np.bincount(y == 1)：np.bincount會回傳一個array，可以數一array的每種非負整數數量，放到他對應的index中。所以這整個就是array([#(-1), #(+1)])，其中#(x)表示y中x的數目。如果是np.bincount(y + 1)的話會是array([#(-1), 0, #(+1)])

所以我們的class CnRT長這樣，之後為了不占版面所以會直接省略這一塊：

class CnRT:

def \_\_init\_\_(self):

self.g\_i = None # self.g 's parameter, can see if it is constant

self.g\_s = None # hypothesis by seeing whether self.g\_const

self.g\_theta = None # isn't None.

self.g\_const = None

self.g = None # lambda x: g(x), where x is np.ndarray

self.subtrees = None # dict{ {1, -1} -> G }, ref. CnRT.transform()

def get\_constant\_g(self, const):

""" get a constant function, works on type(x) = np.ndarray """

return lambda x: np.ones(x.shape[0]) \* const

def get\_stump\_g(self, i, s, theta):

""" get a stump function, works on type(x) = np.ndarray """

return lambda x: np.sign(s \* (x[:, i] - theta))

def gini\_index(self, y):

""" assume all y in {-1, 1} """

if y.size == 0: return 0

return 1 - np.sum((np.bincount(y == 1) / y.size) \*\* 2)

def fit(self, data):

# assume data N x (d+1), where last column is y

x = data[:, :-1]; y = data[:, -1]

# see if base case: all y = 1 or -1, or all x same

if ( np.all(y == 1) or np.all(y == -1)

or np.all([np.all(x[i]==x[i+1]) for i in range(x.shape[0]-1)]) ):

majority = np.argmax(np.bincount(y == 1)) \* 2 - 1

self.g\_const = majority

self.g = self.get\_constant\_g(majority)

return self

# get b

# actually not good, O(d\*N). need DP to make it better: maybe O(d+N)

min\_err = np.inf

for i in range(x.shape[1]):

sorted\_x = np.unique(np.sort(x[:, i]))

theta\_ls = [(sorted\_x[j]+sorted\_x[j+1]) / 2

for j in range(len(sorted\_x) - 1)]

for theta in theta\_ls:

for s in [1, -1]:

# got i, s, theta

cur\_g = self.get\_stump\_g(i, s, theta)

cur\_y = cur\_g(x)

y\_count = np.bincount(cur\_y == 1) # array[#(-1), #(+1)]

cur\_err = (y\_count[0] \* self.gini\_index(y[np.where(cur\_y == -1)]) +

y\_count[1] \* self.gini\_index(y[np.where(cur\_y == 1)]))

if cur\_err < min\_err:

min\_err = cur\_err

self.g\_i = i

self.g\_s = s

self.g\_theta = theta

self.g = cur\_g

for const in [1, -1]:

cur\_g = self.get\_constant\_g(const)

cur\_y = cur\_g(x)

y\_count = np.bincount(cur\_y == 1, minlength=3) # array[#(-1), #(+1)]

cur\_err = (y\_count[0] \* self.gini\_index(y[np.where(cur\_y == -1)]) +

y\_count[1] \* self.gini\_index(y[np.where(cur\_y == 1)]))

if cur\_err < min\_err:

min\_err = cur\_err

self.const = const

self.g = cur\_g

# make subtrees

cur\_y = self.g(x)

self.subtrees = {

1: CnRT().fit(data[np.where(cur\_y == 1)]),

-1: CnRT().fit(data[np.where(cur\_y == -1)])

}

return self

def transform(self, x):

y = self.g(x)

ret = np.copy(y)

if self.subtrees != None:

ret[y == 1] = self.subtrees[1].transform(x[np.where(y == 1)])

ret[y == -1] = self.subtrees[-1].transform(x[np.where(y == -1)])

return ret

所以我們這題的code是：

import numpy as np

train\_file = 'hw6\_train.dat'

test\_file = 'hw6\_test.dat'

class CnRT:

pass # refer to prob.14 for full code

def E\_01(y1, y2):

return np.mean(y1 != y2)

data\_train = np.loadtxt(train\_file)

data\_test = np.loadtxt(test\_file)

print(

E\_01(

data\_test[:, -1],

CnRT().fit(data\_train).transform(data\_test[:, :-1])

)

)

答案是0.166，選C。

**15. D**

因為我們上面的code在train每一顆節點時需時約O(d⋅|D|)O(d⋅|D|)，且DD為此顆節點拿到的資料。雖然可以DP，不過我們還是這樣train了。因為跑得有點慢，為了不要重train一次，我們把所有樹全部分批pickle：  
我們每100棵樹就把這些樹的root和他們取的資料的index num\_ls全部丟進一個list，pickle起來。之後要用的時候把20個檔案的list全部合起來即可。注意每次的seed都不一樣，根據numpy.random.default\_rng的文件，每次做出物件時會拿系統變數來做seed。  
因為object裡面有local variable和lambda，所以必須使用import dill as pickle才能把CnRT pickle。

import numpy as np

import dill as pickle

train\_file = 'hw6\_train.dat'

test\_file = 'hw6\_test.dat'

class CnRT:

pass # refer to prob.14 for full code

data\_train = np.loadtxt(train\_file)

data\_test = np.loadtxt(test\_file)

# train 2000 trees

TREE\_N = 2000

data\_n = data\_train.shape[0]

root\_ls = [] # list[ CnRT() ]

root\_num\_ls = [] # list[ list of index of data used to train this CnRT ]

# train trees, make files

for tree\_i in range(TREE\_N):

print("[\*] current tree\_i =", tree\_i)

rng = np.random.default\_rng()

num\_ls = rng.choice(data\_n, data\_n // 2)

root\_num\_ls.append(num\_ls)

root\_ls.append(CnRT().fit(data\_train[num\_ls, :]))

if (tree\_i + 1) % 100 == 0:

# will dump every 100 trees, containing their data

pickle.dump([root\_ls, root\_num\_ls],

open('pickle\_folder/rt\_nm\_ls\_{}.pickle'.format(tree\_i+1), 'wb'))

root\_ls = []

root\_num\_ls = []

這樣就有所有tree的pickle檔了，這是會重複使用的。  
所以我們載入並解這一題：

import numpy as np

import dill as pickle

import os

train\_file = 'hw6\_train.dat'

test\_file = 'hw6\_test.dat'

class CnRT:

pass # refer to prob.14 for full code

def E\_01(y1, y2):

return np.mean(y1 != y2)

data\_train = np.loadtxt(train\_file)

data\_test = np.loadtxt(test\_file)

root\_ls = []

root\_num\_ls = []

# load in random forest

for fname in os.listdir('./pickle\_folder/'):

a = pickle.load(open('./pickle\_folder/' + fname, 'rb'))

root\_ls.extend(a[0])

root\_num\_ls.extend(a[1])

print(

np.mean(

[E\_01(root.transform(data\_test[:, :-1]), data\_test[:, -1])

for root in root\_ls]

)

)

跑出來是0.229459，所以D。

**16. A**

請把第15題最後的code中，最後print(...)的部分改成：

# ...

def G(x, root\_ls):

sum\_y = np.zeros(x.shape[0])

for root in root\_ls:

sum\_y += root.transform(x)

return np.sign(sum\_y)

print(

E\_01(

G(data\_train[:, :-1], root\_ls),

data\_train[:, -1]

)

)

跑出來是0.015，所以A。

**17. D**

請把第15題最後的code中，最後print(...)的部分改成：

# ...

def G(x, root\_ls):

sum\_y = np.zeros(x.shape[0])

for root in root\_ls:

sum\_y += root.transform(x)

return np.sign(sum\_y)

print(

E\_01(

G(data\_test[:, :-1], root\_ls),

data\_test[:, -1]

)

)

跑出來是0.152。

**18. B**

請把第15題最後的code中，最後print(...)的部分改成：

# ...

y = np.zeros(data\_train.shape[0])

# used to determine which data was taken by all g

sum\_trained\_bin = np.copy(y)

for root, num\_ls in zip(root\_ls, root\_num\_ls):

untrained\_x\_pos = np.where(np.bincount(num\_ls, minlength=y.shape[0]) == 0)[0]

sum\_trained\_bin += (np.bincount(num\_ls, minlength=y.shape[0]) == 0)

y[untrained\_x\_pos] += root.transform(data\_train[untrained\_x\_pos, :-1])

y = np.sign(y)

# for those data that was taken by all root, make them become -1

y[np.where(sum\_trained\_bin == 0)] = -1

print(E\_01(y, data\_train[:, -1]))

跑出來是0.071，所以B。

**19. D**

比較是覺得推導很有趣，回去自己整個重來一遍的時候總感覺他們的觀察和過程真的很厲害。不過即使重導一遍，感覺還是很快就又忘了，不論是過程或甚至是本身的演算法在幹嘛都是。猜想是實戰經驗不足或是還是沒有對他們太熟悉吧，也有可能是我導到後來全部都只被理解成純符號了，導致中間沒有多少深刻的見解。  
事實上我覺得SVM那一段也很厲害，不過昨天剛導完的gradient boost隔天連他的策略是甚麼都忘掉的我，能想起來的東西更少了…。

**20. E**

簡而言之，感覺給的篇幅有點少，而且被塞在最後，一大部分也沒有被功課cover到，導致最後幾堂課的內容都沒有什麼真的學習到。

對我來說，我覺得把這兩堂課擺在一個學期的實驗做的最成功的部分大概是證明了：以大量題目當作作業當作複習，的確是讓學生能自主回去複習的好方法。因為這個學期課比較多，所以基本上複習都是在寫功課的時間的前半部重看投影片，之後再馬上寫完題目。寫題目的同時也能注意到剛才讀的時候哪裡沒有注意到，甚至能再多根據解題時的發現來做更多筆記到檔案上。

我想其實這可以被認為是因為功課的壓力而作的複習，但是根據我的直覺，這學期其他課的功課量實在有點多，多到我基本上不會另外找時間重溫上課內容。上完這兩堂也可以發現，後面的課程reference前面的內容的部分是相當多，假設初期沒有打好基礎(不論是培養直覺或是理論的基礎)，後面基本上是很難趕得上了。有功課這樣的推力讓我回去複習我覺得對我來說是必要的，且後來的結果我覺得是成功的。事實上，~~相對於上個學期系上開的科學計算~~，這堂課有足夠強度的功課讓同學趕的上進度和系統性的規劃教學地圖，我是有感受到極大的差距的。

不過大概是要把兩堂課排到一學期的進度考量，後面的deep neural network的部分有很大篇幅沒有被排到作業內，而內容也有一定的複雜度，也是因為期末沒有時間的關係，導致我大概得暑假再重上一次技法後半部的課了。如果要提出之後能夠改進的點的話，我認為這樣的速度配上功課我是能勉強趕得上基石和技法前面的內容的，所以為了讓同學能更加深內容印象~~而不是自己導完算式的隔天就忘記~~，兩個學期的時間加上更多實作題應該能讓教學速度再慢一些讓同學有更多時間跟上，並且透過實作真的讓同學理解每一個模型。不過原本想說每個模型都讓同學實作一遍應該會很有效果，但是光想到作業篇幅可能會太長以及考慮到不是每個同學都是資工領域的，再加上也不能無視掉同學其他科科目的份量，我覺得今年這個題目量雖然不盡理想但也許是迫於現實吧。

總而言之，我覺得這個模式算是在現實的constraint做出的近似optimal的解，也有在這樣的功課/教學模式中學到東西。只是一個學期的話速度其實是勉強能跟的上的等級，但是分兩個學期的話應該要增加內容及功課的量，有好有壞就是了。

發表於 [**HackMD**](https://hackmd.io/)

 28

[讚賞](https://hackmd.io/@Kaiserouo/HkYViNNTP) [收藏](https://hackmd.io/@Kaiserouo/HkYViNNTP) [訂閱](https://hackmd.io/@Kaiserouo/HkYViNNTP)