Machine Learning HW6

B06502152, 許書銓

Neural Networks

1. The answer should be (b).

   Computing the number of $\delta_k^{(l+1)} w_{jk}^{(l+1)}$, each

   $$\delta_j^{(l)} = \Sigma_k \delta_k^{(l+1)} w_{jk}^{(l+1)} \tanh(s_j^{(l)}) \tag{1.0}$$

   Hence,

   $$\text{for } \delta^{(2)}, \text{we need to compute } 6 \times 1 = 6$$

   $$\text{for } \delta^{(1)}, \text{we need to compute } 5 \times 6 = 30$$

   $$\text{for } \delta^{(0)}, \text{we need to compute } 4 \times 5 = 20$$

   For computing $l = \{1, 2\}$, we need to compute 6 + 30 = 36 times.

2. The answer should be (d).

   - Consider that there is only one layer for hidden layer, which lead a 19-49-3 neural network. Then, the total number of weights would be

     $$\text{For 1 hidden layer }, \quad \#\text{ of weights } = 20 \times 49 + 50 \times 3 = 1130. \tag{2.0}$$

   - Consider that there are two layers for hidden layer, which lead a 19--x--(48-x)--3 neural network. Then, the

total number of weights would be

$$\text{For 2 hidden layers , } \quad \#\text{ of weights } = 20 \times x + (x+1) \times (48-x) + (49-x) \times 3 \tag{2.1}$$

To find the maximum number of layers in the case of 2 hidden layers, we would like to take the derivatives of equation (2.1).

$$\frac{\partial \text{ equ}(2.1)}{\partial x} = 19 - 2x + 48 - 1 - 3 = 0 \tag{2.3}$$

$$x = 32 \tag{2.4}$$

Then,

$$\text{For 2 hidden layers , } \quad \#\text{ of weights } = 20 \times 32 + (32+1) \times (48-32) + (49-32) \times 3 = 1219 \tag{2.5}$$

- Consider Consider that there are three layers for hidden layer, which lead a 19--x--y--(47-x-y) --3 neural network. The constraint is 47 -x - y ≥ 1 . Then, the total number of weights would be

$$\begin{aligned}\text{For 3 hidden layers , } \quad \#\text{ of weights } &= 20 \times x + (x+1) \times y + (y+1) \times (47-x-y) \\ &\quad + (48-x-y) \times 3\end{aligned} \tag{2.6}$$

To find the optimal number of weights, we would like to take the derivative of equation (2.6)

$$\frac{\partial \,\text{equ}(2.6)}{\partial x} = 20 + y - y - 1 - 3 = 16$$

$$\frac{\partial \,\text{equ}(2.6)}{\partial y} = x + 1 + 47 - x - 1 - 2y - 3 = 0 \tag{2.7}$$

Then,

$$y = 22 \tag{2.8}$$

Moreover, x should be as large as possible, the max possible x is from the constraint 47 -x - y ≥ 1 . The last hidden layer = 1 and, x = 24.

$$\begin{aligned} \text{For 3 hidden layers , } \quad \#\text{ of weights } &= 20 \times 24 + (24+1) \times 22 + (22+1) \times (47-24-22) \\ &\quad + (48-24-22) \times 3 \\ &= 1059 \end{aligned} \tag{2.9}$$

- Apply more layers (layers ≥ 3)

  From the upper calculation, we realize that if we apply more layers, the maximum number of weight in the case of more layers would be less than aply two layers. Thus, the optimal solution is 1219.

3. The answer should be (d).

$$\begin{aligned} \frac{\partial \text{err}(\mathbf{x}, y)}{\partial s_k} &= -\frac{\partial}{\partial s_k} \ln q_k \\ &= -\frac{\partial}{\partial s_k} \ln \frac{\exp(s_k^{(L)})}{\Sigma_{k=1}^{K} \exp(s_k^{(L)})} \\ &= \frac{\partial}{\partial s_k} \ln(\Sigma_{k=1}^{K} \exp(s_k^{(L)})) - \frac{\partial}{\partial s_k} \ln(\exp(s_k^{(L)})) \end{aligned} \tag{3.0}$$

By taking the derivative of equation (3.0), and noting that when $y! = k$,the second term in equation becomes 0. Thus,

$$\begin{aligned} \frac{\partial \text{err}(\mathbf{x}, y)}{\partial s_k^{(L)}} &= \frac{\exp(s_k^{(L)})}{\Sigma_{k=1}^{K} \exp(s_k^{(L)})} - [|y = k|] \frac{\exp(s_k^{(L)})}{\exp(s_k^{(L)})} \\ &= q_k - v_k \end{aligned} \tag{3.1}$$

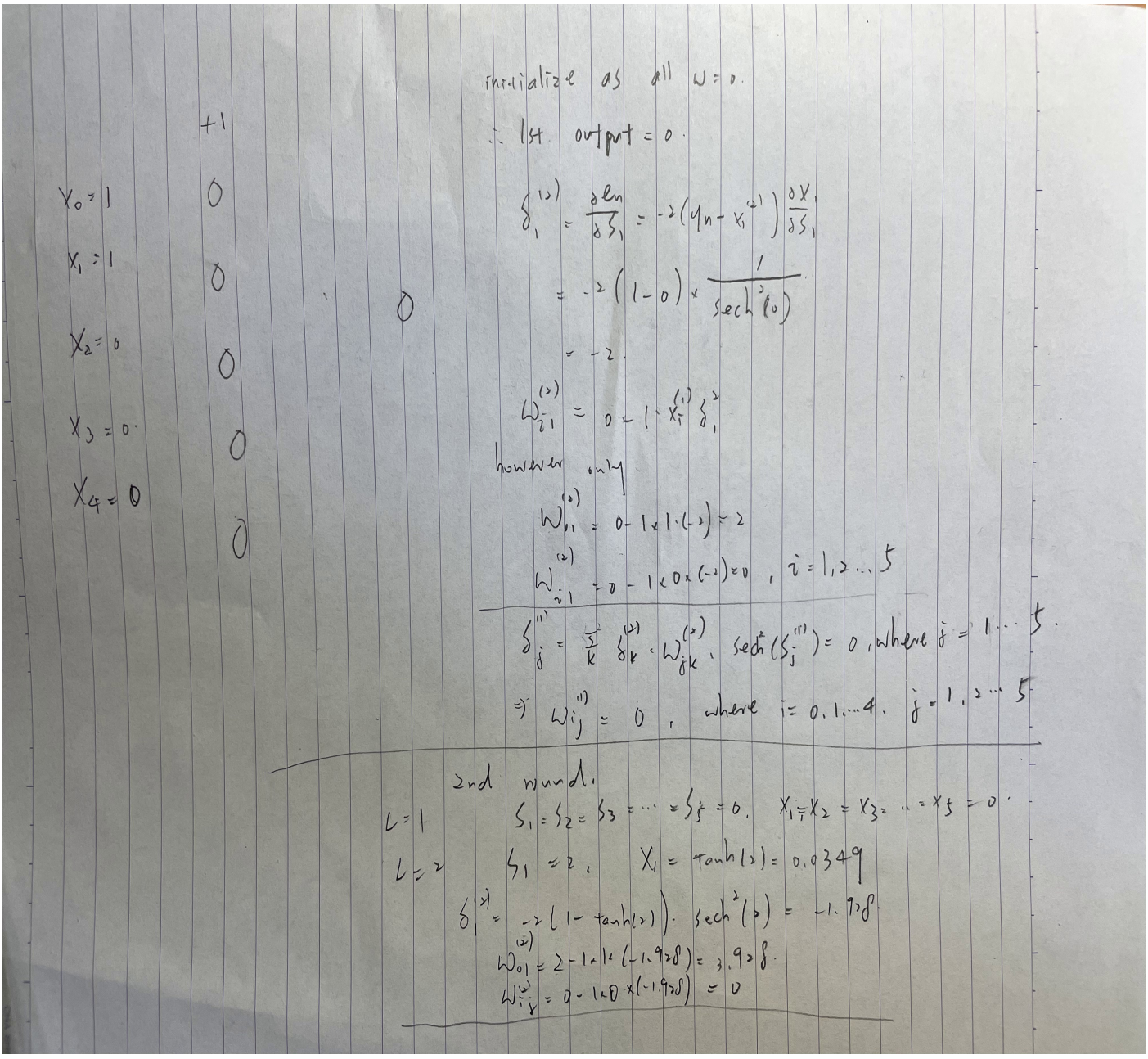
4. The answer should (a).

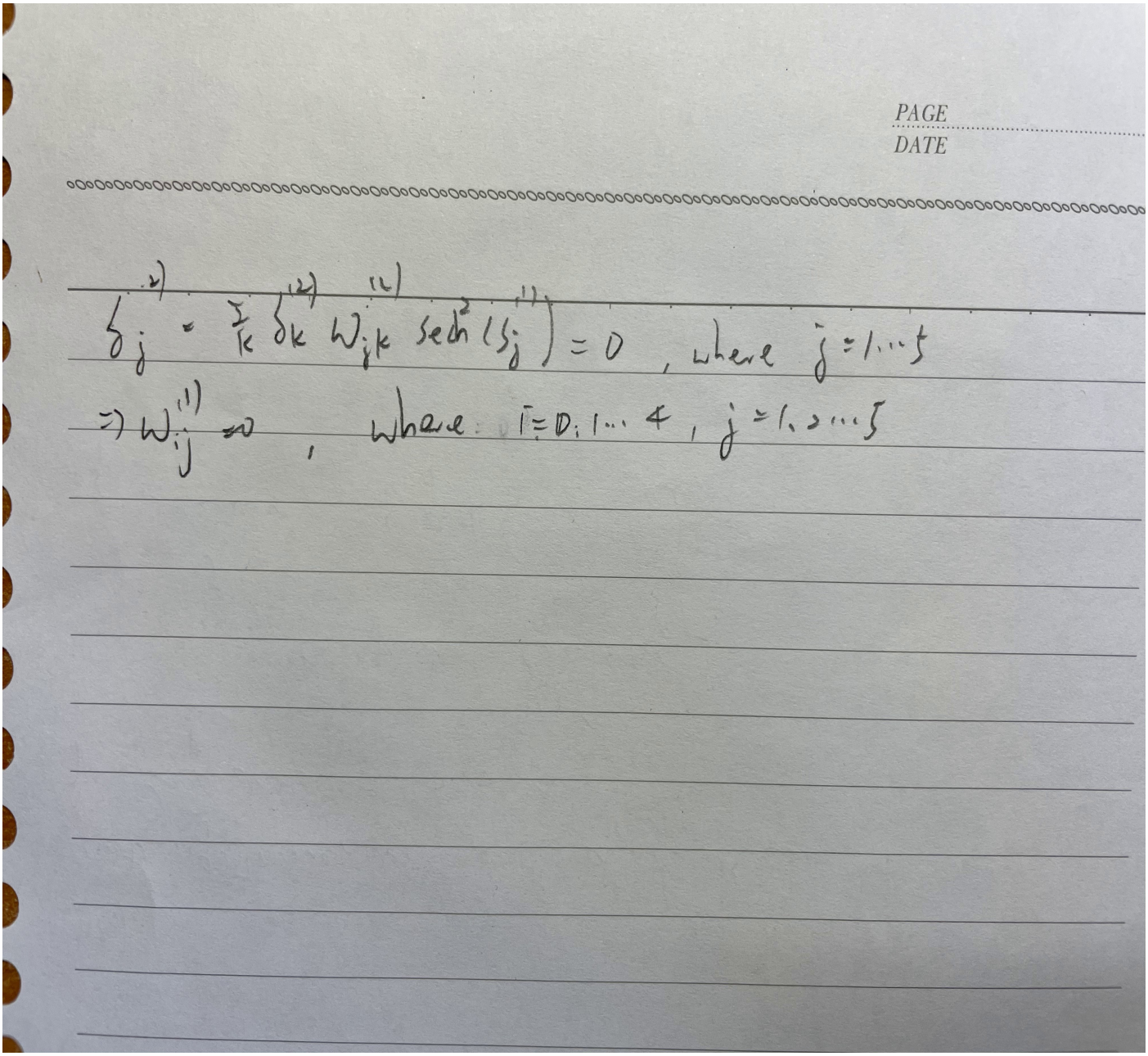
From the definition of δ of the last layer, and condering the output layer also need to do the hyperbolic tangent transform ,

$$\delta^{(L)} = \frac{\partial e_n}{\partial s^{(L)}} = -2(y_n - \tanh(s_1^{(L)}))\text{sech}^2(s_1^{(L)}) \tag{4.0}$$

Thus, in the first iteration with all w initializ as 0,

$$\delta_1^{(L)} = \frac{\partial e_n}{\partial s^{(L)}} = -2(y_n - \tanh(0))\text{sech}^2(0) = 0 \tag{4.1}$$





From the upper induction, we can observe that after each iteration, only $w_{01}^{(2)}$ will be updated while the other w will remian 0. Thus, $w_{01}^{(1)}$ will always be 0.

Matrix Factorization

5. The answer should be (e).

   From the pseudo code in p.10 of lecture (215), we realize that step 2.1 aims to optimize $\mathbf{w}$. Furthermore, we know that E_{in} here is,

$$\min_{\mathbf{w},\ \mathbf{v}} E_{in}(\{\mathbf{w}\}\{\mathbf{v}\}) \propto \Sigma_{\text{user n related movie m}} \left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n\right)^2$$

$$= \Sigma_{m\ =\ 1}^{M} \Sigma_{x_n, r_m \in \mathcal{D}_m} \left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n\right)^2 \tag{5.0}$$

Now, with fixed v_n,

$$\nabla_{w_m} \quad \Sigma_{x_n, r_m \in \mathcal{D}_m} \left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n\right)^2 = \Sigma_{x_n, r_m \in \mathcal{D}_m} - 2(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)\mathbf{v}_n \tag{5.1}$$

To find the optimal w_m, equation (5.1) must be 0. Thus,

$$\Sigma_{x_n, r_m \in \mathcal{D}_m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)\mathbf{v}_n = 0$$

$$\rightarrow \quad \Sigma_{x_n, r_m \in \mathcal{D}_m} r_{nm}\mathbf{v}_n - \mathbf{w}_m^T \mathbf{v}_n \mathbf{v}_n = 0 \tag{5.2}$$

From the problem itself, since $\tilde{d} = 1$, v_n is a 1×1 constant 2.

$$\Sigma_{x_n, r_m \in \mathcal{D}_m} (r_{nm} \times 2) - \Sigma_{x_n, r_m \in \mathcal{D}_m} (\mathbf{w}_m^T \times 4) = 0 \tag{5.3}$$

$$\Sigma_{x_n, r_m \in \mathcal{D}_m} (r_{nm}) = \Sigma_{x_n, r_m \in \mathcal{D}_m} (\mathbf{w}_m^T \times 2)$$

$$\rightarrow \quad \mathbf{w}_m = \frac{1}{2} \frac{\Sigma_{x_n, r_m \in \mathcal{D}_m} r_{nm}}{\Sigma_{x_n, r_m \in \mathcal{D}_m} 1} \tag{5.4}$$

The physical meaning of equation (5.4), is that w is equal to half the average rating of the m-th movie.

6. The answer should be (b).

Consider introducing the bias term into the equation in the lecture , the err would be changed to

$$\text{err}(\text{user n}, \text{rating m}, \text{rating } \text{r}_{\text{nm}}) = (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n - a_m - b_n)^2 \tag{6.0}$$

Hence,

$$\nabla_{a_m} err = -2(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n - a_m - b_n) \tag{6.1}$$

From equation (6.1), we realize that a_m would be updated as

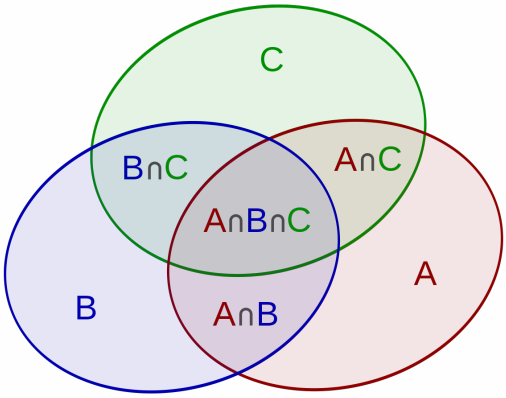
$$a_m \leftarrow a_m + \frac{\eta}{2}(2(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n - a_m - b_n))$$

$$= (1-\eta)a_m \ + \ \eta \cdot (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n - b_n) \tag{6.2}$$

Aggregation

7. The answer should be (d).

   Consider the following figure,



   the $G(x) = \text{sign}(\Sigma_{t=1}^{T} g_t(x))$. Hence, the data, which predict wrong must be the "and (^) " part of the three $g_t(x)$. That is, the error data must also be error data in at least two $g_t(x)$. Consider that the error data for $g_1(x) = A$, the error data for $g_2(x) = B$ and the error data for $g_3(x) = C$.

   Looking at option (d), if C∩ A = 0.16, C ∩ B = 0.04, A∩ B = 0, will satisfy the condition. However, in the other options, the max "and" of at least two set is less than 0.2, which cannot be the answer.

8. The answer should be (c).

   Using the $G(x)$, which is similiar to the G(x) in p10 of lecture 207 (also in the problem 7), the possible error predicting must be selected by at least 3 $g_t(x)$. Hence, the E_{out} may be,

$$E_{out}(x) = 1 - C_0^5 \ (0.6)^5 - C_1^5 \ (0.4)^1 \ (0.6)^4 - C_2^5 \ (0.4)^2 \ (0.6)^3$$

$$= 0.31744 \tag{8.0}$$

   The closest answer is 0.32.

9. The answer should be (b).

$$
\begin{aligned}
(1-\frac{1}{N})^{\frac{N}{2}} &= \frac{1}{(\frac{N}{N-1})^{\frac{N}{2}}} \\
&= \frac{1}{(1+\frac{1}{N-1})^{\frac{N}{2}}} \\
&\approx \frac{1}{\sqrt{e}} = 0.605
\end{aligned} \tag{9.0}
$$

10. The answer should be (e).

Let's first looking at $\phi(x)$, there are $(R-L)$ different θ between $2L$ and $2R$, each θ can operate differently bt multiplying s and selecting different d to work at. Hence, the number of $\phi(x)$ is $2d(R-L)$.

$$
|\phi| = 2d(R-L) \tag{10.0}
$$

Now, examing

$$
\begin{aligned}
K_{ds}(\mathbf{x},\mathbf{x}') &= (\phi_{ds}(\mathbf{x}))^T(\phi_{ds}(\mathbf{x}')) \\
&= \Sigma_{t=1}^{|\phi|} s_t(\text{sign}(x_t-\theta_t))s_t(\text{sign}(x'_t-\theta_t)) \\
&= \Sigma_{t=1}^{|\phi|} (\text{sign}(x_t-\theta_t))(\text{sign}(x'_t-\theta_t))
\end{aligned} \tag{10.1}
$$

Considering that,

$$
\begin{aligned}
(\text{sign}(x_i-\theta_i))(\text{sign}(x'_i-\theta_i)) &= +1, \text{when } \theta_{\text{i}} \in [\min(\text{x}_{\text{i}},\text{x}'_{\text{i}}), \max(\text{x}_{\text{i}},\text{x}'_{\text{i}})) \\
&= -1, \text{when } \theta_{\text{i}} \notin [\min(\text{x}_{\text{i}},\text{x}'_{\text{i}}), \max(\text{x}_{\text{i}},\text{x}'_{\text{i}}))
\end{aligned} \tag{10.2}
$$

Thus, we have to know how many -1 there are. That is, how many θ can be $\in [\min(x_i,x'_i), \max(x_i,x'_i))$. The number of θ are $2 \times \Sigma_{i=1}^{d} \frac{|x'_i-x_i|}{2}$. The first 2 stands for s = {+1, -1}, and the second 2 stands for that x_i only contains even integers.

$$
2\Sigma_{i=1}^{d} \frac{|x'_i-x_i|}{2} = ||x'_i-x_i||_1 \tag{10.3}
$$

On the other hand, the number of +1 are $|\phi| - ||x'_i-x_i||_1$. Hence,

$$
\begin{aligned}
K_{ds}(\mathbf{x}, \mathbf{x}') &= (\phi_{ds}(\mathbf{x}))^T(\phi_{ds}(\mathbf{x}')) \\
&= \Sigma_{t=1}^{|\phi|}(\text{sign}(x_t - \theta_t))(\text{sign}(x'_t - \theta_t)) \\
&= 2d(R-L) - ||x'_i - x_i||_1 - ||x'_i - x_i||_1 \\
&= 2d(R-L) - 2||x'_i - x_i||_1
\end{aligned} \tag{10.4}
$$

Adaptive Boosting

11. The answer should be (a).

    From the lecture note, we would like to mutiple the incorrect data with $\sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$, where ϵ is the rate of error, and mutiple the correct data with $\frac{1}{\sqrt{\frac{1-\epsilon_t}{\epsilon_t}}}$. Hence,

$$
\begin{aligned}
\text{incorrect data}: \quad & u_+^{(2)} \leftarrow u_+^{(1)} \times \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \\
\text{correct data}: \quad & u_-^{(2)} \leftarrow u_-^{(1)} \times \frac{1}{\sqrt{\frac{1-\epsilon_t}{\epsilon_t}}}
\end{aligned} \tag{11.0}
$$

From the update equation in (11.0), and in the first round $u_+ = u_-$

$$
\begin{aligned}
\frac{u_+^{(2)}}{u_-^{(2)}} &= \frac{1-\epsilon_t}{\epsilon_t} \\
&= \frac{0.95}{0.05} = 19
\end{aligned} \tag{11.1}
$$

12. The answer should be (d).

    From the definition of U_t

$$
U_{t+1} = \Sigma_{n=1}^{N} u_t^{(t+1)} \tag{12.0}
$$

From the lecture note p.12 of (211), we can further induct into,

$$U_{t+1} = \Sigma_{n=1}^{N} u_t^{(t+1)} \tag{12.0}$$

$$= \Sigma_{n=1}^{N} u_n^{(t)} [(1-\epsilon_t)e^{-\alpha t} + \epsilon_t e^{-\alpha t}] \tag{12.1}$$

$$= \Sigma_{n=1}^{N} u_n^{(1)} \ \Pi_{t=1}^{T} [(1-\epsilon_t)e^{-\alpha t} + \epsilon_t e^{-\alpha t}] \tag{12.2}$$

Since $\Sigma_{n=1}^{N} u_n^{(1)} = N \times \frac{1}{N} = 1$,

$$\Sigma_{n=1}^{T} u_n^{(1)} \ \Pi_{t=1}^{N} [(1-\epsilon_t)e^{-\alpha t} + \epsilon_t e^{-\alpha t}] = \Pi_{t=1}^{N} [(1-\epsilon_t)e^{-\alpha t} + \epsilon_t e^{-\alpha t}] \tag{12.3}$$

$$= \Pi_{t=1}^{T} [(1-\epsilon_t)\sqrt{\frac{\epsilon_t}{1-\epsilon_t}} + \epsilon_t \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}] \tag{12.4}$$

$$= \Pi_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} \tag{12.5}$$

From the problem hint, which is $\sqrt{\epsilon(1-\epsilon)} \leq \frac{1}{2}\exp(-2(\frac{1}{2}-\epsilon)^2)$, the equation (12.5) can be rewrited as

$$\Pi_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} \leq \Pi_{t=1}^{T} \exp(-2(\frac{1}{2}-\epsilon_t)^2) \tag{12.6}$$

$$= \exp(-2T(\frac{1}{2}-\epsilon_t)^2) \tag{12.7}$$

Hence,

$$E_{in}(GT) \leq \exp(-2T(\frac{1}{2}-\epsilon_t)^2) \tag{12.8}$$

Decision Tree

13. The answer should be (d).

(a.) The Gini index, 1-$\mu_+^2 - \mu_-^2$.

$$\begin{aligned} 1-\mu_+^2-\mu_-^2 &= 1-\mu_+^2-(1-\mu_+)^2 \\ &= -\mu_+^2+2\mu_+-\mu_+^2 \\ &= \frac{1}{2}-2(\mu_+-\frac{1}{2})^2 \end{aligned} \tag{13.0}$$

From equation (13.0), the max Gini index between $\mu_+ \in [0,1]$ is $\frac{1}{2}$.

Hence, the normalized Gini index is

$$\text{Nomornized Gini index} = \frac{2\mu_+-2\mu_+^2}{\frac{1}{2}} = 4\mu_+-4\mu_+^2 \tag{13.1}$$

(b.) The squared error

$$\begin{aligned} \mu_+(1-(\mu_+-\mu_-))^2+\mu_-(-1-(\mu_+-\mu_-))^2 &= \mu_+(1-(\mu_+-\mu_-))^2+\mu_-(1+(\mu_+-\mu_-))^2 \\ &= 1-2(\mu_+-\mu_-)(\mu_+-\mu_-)+(\mu_++\mu_-)(\mu_+-\mu_-)^2 \\ &= 1-2(\mu_+-\mu_-)^2+(\mu_+-\mu_-)^2 \\ &= 1-(\mu_+-\mu_-)^2 \end{aligned} \tag{13.2}$$

From equation (13.2), the max squared error between $\mu_+ \in [0,1]$ is 1.

Hence, the normalized Gini index is

$$\text{Nomornized squared error} = \mu_+(1-(\mu_+-\mu_-))^2+\mu_-(-1-(\mu_+-\mu_-))^2 = 1-(\mu_+-\mu_-)^2 \quad (13.3)$$

(c) The entropy

(d) The closeness

$$\begin{aligned}\text{closeness} &= 1-|\mu_+-\mu_-| \\ &= 2\min(\mu_+,\mu_-) \end{aligned} \quad (13.4)$$

From equation (13.4), the max squared error between $\mu_+ \in [0,1]$ is 1.

Hence, the

$$\begin{aligned}\text{Nomornized closeness} &= 1-|\mu_+-\mu_-| \\ &= 2\min(\mu_+,\mu_-) \end{aligned} \quad (13.5)$$

Experiments with Decision Tree and Random Forest

14. The answer should be (c).
15. The answer should be (d).
16. The answer should be (a).
17. The answer should be (d).
18. The answer should be (b).

```
##Library
import numpy as np
import sys
import random
from time import sleep
from tqdm import tqdm, trange

##Traing and Testing Data
!wget http://www.csie.ntu.edu.tw/~htlin/course/ml20fall/hw6/hw6_train.dat
!wget http://www.csie.ntu.edu.tw/~htlin/course/ml20fall/hw6/hw6_test.dat

##Class and Funtions
class DTree:
    def __init__(self, theta, feature, value = None):
        self.theta = theta
        self.feature = feature
        self.value = value
        self.left = None
```

```
        self.right = None

def gini(y):
    if len(y) == 0:
        return 1

    mu = np.sum(y == 1)
    mu = mu/len(y)
    g = 1 - mu**2 - (1-mu)**2
    return g

def decision_stump(X, y):
    _feature = 0
    _theta = 0
    _min_err = float('inf')

    for i in range(X.shape[1]):
        ## i column of data x
        x = X[:, i]

        ##generator theta
        x_sort = np.sort(x)

        theta = []
        for j in range(len(x_sort) - 1):
            theta.append((x_sort[j+1] + x_sort[j])/2 )

        theta.insert(0, (x_sort[0]-1))
        theta.append((x_sort[-1]+1))

        ##calculate err
        for j in range(len(theta)):
            ##divide y into y1 and y2, y1 for x < theta, y2 for x >= theta
            y1 = y[x < theta[j]]
            y2 = y[x >= theta[j]]

            ##calculate err based on Gini idx
            gi1 = gini(y1)
            gi2 = gini(y2)
            err = len(y1)*gi1 + len(y2)*gi2

            if _min_err > err:
                _min_err = err
                _theta = theta[j]
                _feature = i

    return _feature, _theta, _min_err

def stop(X, y):
    ## if all y the same
    n1 = np.sum(y != y[0])

```

```
    ## if all x the same
    n2 = np.sum(X != X[0, :])

    return n1 == 0 or n2 == 0

def DecisionTree(X,y):
    if stop(X, y):
        #print("stop! ")
        g = DTree(None, None, y[0])
        return g

    else:
        ##learning criteria based on decision stump
        feature, theta, score = decision_stump(X, y)
        g = DTree(theta, feature, None)
        #print(feature, theta, score)
        ##plit data in to two parts
        x1 = X[X[:, feature] < theta]
        y1 = y[X[:, feature] < theta]
        #print(x1.shape)
        x2 = X[X[:, feature] >= theta]
        y2 = y[X[:, feature] >= theta]
        #print(x2.shape)

        ##build sub_tree
        g1 = DecisionTree(x1, y1)
        g2 = DecisionTree(x2, y2)

        g.left = g1
        g.right = g2

        ##return tree data
        return g

def predict_helper(tree, X):
    if tree.value != None:
        return tree.value

    if X[tree.feature] < tree.theta:
        return predict_helper(tree.left, X)
    else:
        return predict_helper(tree.right, X)

def predict(tree, X):
    y = [predict_helper(tree, x) for x in X ]
    return y

def score(tree, pred_y, y):
    E_out = 0
    for i in range(y_out.shape[0]):
        if(pred_y[i] != y[i]):
            E_out += 1
```

```
    E_out /= y.shape[0]
    return E_out

def bagging(data):
    selected_idx = []
    idx = random.randint(0, data.shape[0]-1)
    selected_idx.append(idx)

    new_data = data[idx, :]
    new_data = new_data.reshape((1, -1))
    for i in range(500-1):
        idx = random.randint(0, data.shape[0]-1)
        selected_idx.append(idx)
        row = data[idx, :]
        row = row.reshape((1,-1))
        new_data = np.concatenate((new_data,row), axis=0)

    split_train_x = new_data[:, :-1]
    split_train_y = new_data[:, -1]

    return split_train_x, split_train_y, selected_idx

def random_forest(data_in, train_X, train_y, test_X, test_y, num):
    tmp_data = data_in
    #train_X = data_in[:, :-1]
    #train_y = data_in[:, -1]

    avg_Eout = 0
    G_in= np.zeros((train_y.shape[0],1))
    G_in = G_in.squeeze()
    G_out= np.zeros((test_y.shape[0],1))
    G_out = G_out.squeeze()
    G_oob= np.zeros((test_y.shape[0],1))
    G_oob = G_oob.squeeze()
    temp = 0
    total = num

    for i in range(num):
        split_train_X, split_train_y, selected_idx= bagging(data_in)
        G = DecisionTree(split_train_X, split_train_y)

        #print(train_y)

        ##for E_in
        pred_train_y = predict(G, train_X)
        ##for E_out
        pred_test_y = predict(G, test_X)
        ##for average E_out
        E_out = score(G, pred_test_y, test_y)
        ##for E_oob
        selected_idx.sort()
        #print(selected_idx)
```

```
        #print(0 in selected_idx)
        for i in range(data_in.shape[0]):
            if i not in selected_idx:
                G_oob[i] += predict_helper(G, train_X[i, :])

        avg_Eout += E_out
        G_in += pred_train_y
        G_out += pred_test_y

        temp += 1
        print('\r' + '[Progress]:[%s%s]%.2f%%;' % (
        '█' * int(temp*20/total), ' ' * (20-int(temp*20/total)),
        float(temp/total*100)), end='')
        sleep(0.01)

    avg_Eout /= num

    g_in = np.sign(G_in)
    for i in range(g_in.shape[0]):
        if(g_in[i] == 0):
            g_in[i] = -1

    g_out = np.sign(G_out)
    for i in range(g_out.shape[0]):
        if(g_out[i] == 0):
            g_out[i] = -1

    g_oob = np.sign(G_oob)
    for i in range(g_oob.shape[0]):
        if(g_oob[i] == 0):
            g_oob[i] = -1

    E_in = 0
    for i in range(train_y.shape[0]):
        if(g_in[i] != train_y[i]):
            E_in += 1
    E_in /= train_y.shape[0]

    E_out = 0
    for i in range(test_y.shape[0]):
        if(g_out[i] != test_y[i]):
            E_out += 1
    E_out /= test_y.shape[0]

    E_oob = 0
    for i in range(train_y.shape[0]):
        if(g_oob[i] != train_y[i]):
            E_oob += 1
    E_oob /= train_y.shape[0]
    return avg_Eout, E_in, E_out, E_oob

##Main
```

```
data_in = np.genfromtxt("hw6_train.dat")
data_out = np.genfromtxt("hw6_test.dat")
X = data_in[:, :-1]
y = data_in[:, -1]

X_out = data_out[:, :-1]
y_out = data_out[:, -1]

##learn tree
G = DecisionTree(X, y)

##predict output data
pred_y = predict(G, X_out)

##calculate E_out
E_out_DTree = score(G, pred_y, y_out)
#print("P14 :" , E_out_DTree)

avg_Eout, E_in, E_out, E_oob = random_forest(data_in, X, y, X_out, y_out,
2000)

print("\n")
print("P14 :" , E_out_DTree)
print("P15 :", avg_Eout)
print("P16 :", E_in)
print("P17 :", E_out)
print("P18 :", E_oob)
```

Learning Comes from Feedback

19. The answer should be (d).

Aggregation provide me another way to lower my E_{out}, which broaden my horizons about how machine learning can work. Furthermore, it is quite human-like for that "two heads are better than one" huh.

20. The answer should be (e).

For me, the mathematics in Neural Networks is really too much, which hinder the way for me to learn this chapter. QQ.