Homework #5

5.1 Powerfulness of Neural Networks

(1) We implement $f_A(\mathrm{x}) = g_A(\mathrm{x}|\theta, \mathrm{w}_1, \mathrm{w}_2, \dots, \mathrm{w}_\mathrm{d})$, with $\mathrm{W} = (\theta, \mathrm{w}_1, \mathrm{w}_2, \dots, \mathrm{w}_\mathrm{d})$.

$$\mathrm{W}_{i1}^{(1)} = \begin{cases} \mathrm{d}-1, & \text{if i} = 0 \\ 1, & \text{otherwise} \end{cases}$$

To proof it can work, we show the function $g_\mathrm{A}(\mathrm{x}) = 1$ if and only if all inputs x are 1. The statement is satisfied because the truth that we only have d's inputs argument, each of them either 1 or -1. All of inputs should be 1 and sum up to greater than d - 1. Otherwise, the function $g_\mathrm{A}(\mathrm{x}) = -1$.

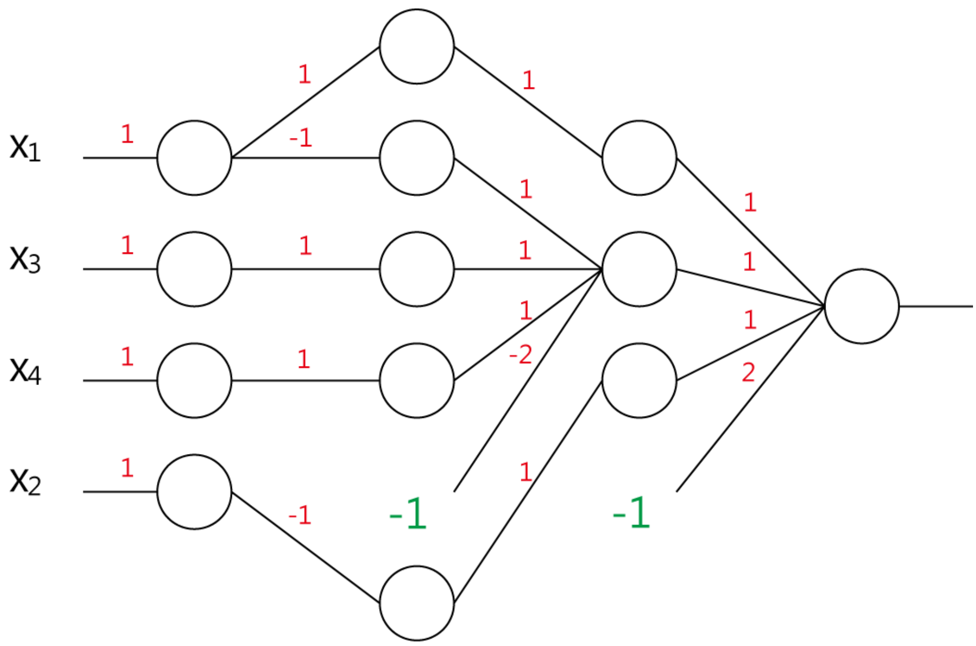
(2) We implement $f_O(\mathrm{x}) = g_\mathrm{O}(\mathrm{x}|\theta, \mathrm{w}_1, \mathrm{w}_2, \dots, \mathrm{w}_\mathrm{d})$, where

$$\mathrm{W}_{i1}^{(1)} = \begin{cases} 1-\mathrm{d}, & \text{if i} = 0 \\ 1, & \text{otherwise} \end{cases}$$

To proof it works, we show the function $g_O(\mathrm{x}) = -1$ if and only if all inputs x are -1. The statement is satisfied with the similar reason. All of inputs should be -1 and sum up to smaller than 1 – d. Otherwise, the function $g_O(\mathrm{x}) = 1$.

(3) $f_N(\mathrm{x}) = g_N(\mathrm{x}|\theta, \mathrm{w}_1)$, where $(\theta, \mathrm{w}_1) = (0, -1)$. The result is trivial.

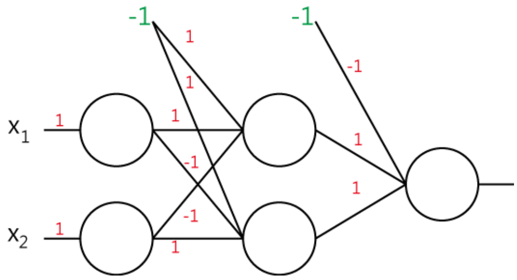
(4) Illustrated the result by the combination of f_A, f_O and f_N perceptron we define above as to a Neural Networks (NN) (5.1.1).



(5.1.1)

(5) By (1), (2) and (3). We know that the 3 element gates can be generated as a NN form. For any Boolean system, we can change it to conjunctive normal form. Thus, we can systematically implement any Boolean function by back propagate each gates to a NN by Boolean operation.

(6) Try the simpler case $XOR((\mathrm{x})_1,(\mathrm{x})_2) = OR\left(AND(NOT((\mathrm{x})_1),(\mathrm{x})_2), AND\left((\mathrm{x})_1, NOT((\mathrm{x})_2)\right)\right)$ and get the figure (5.1.2)



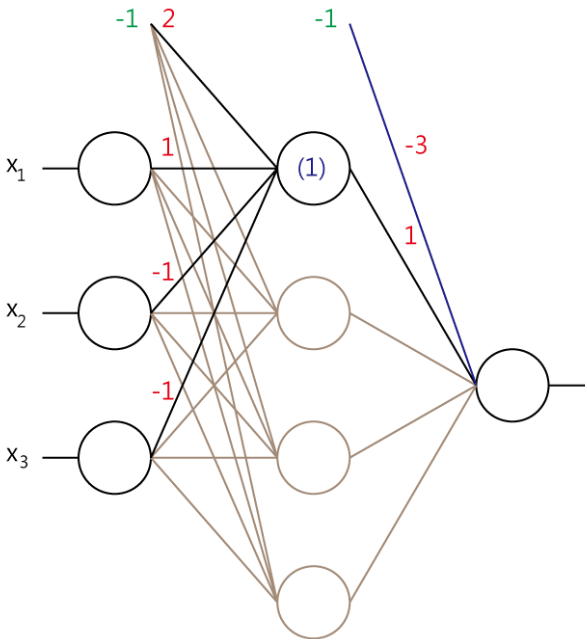
(5.1.2)

Use this form we can derive the Boolean equation (5.1.3) that implement a 3 inputs *XOR* with a hidden layer for 4 perceptrons.

$$XOR((\mathrm{x})_1,(\mathrm{x})_2,(\mathrm{x})_3) =$$

$$OR\left(\underbrace{AND\left((\mathrm{x})_1,\overline{(\mathrm{x})_2},\overline{(\mathrm{x})_3}\right)}_{(1)}, AND\left(\overline{(\mathrm{x})_1},(\mathrm{x})_2,\overline{(\mathrm{x})_3}\right), AND\left(\overline{(\mathrm{x})_1},\overline{(\mathrm{x})_2},(\mathrm{x})_3\right), AND((\mathrm{x})_1,(\mathrm{x})_2,(\mathrm{x})_3)\right) \tag{5.1.3}$$

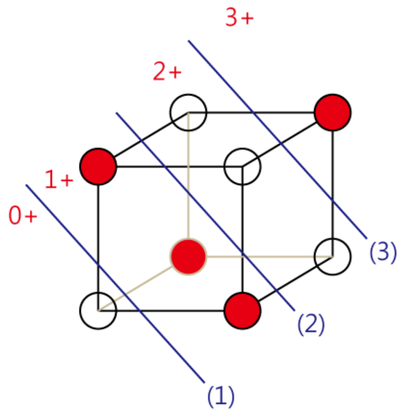
We can get the figure (5.1.4) by the equation (5.1.3)



(5.1.4)

To simplify figure, we use black edges to represent the *AND* gate structures, and the gray edges are similar to black edges but notice the position of *NOT* operation.

(7) Figure (5.1.5) show that we only need 3 perceptrons to separate the hypercube in three dimensions.



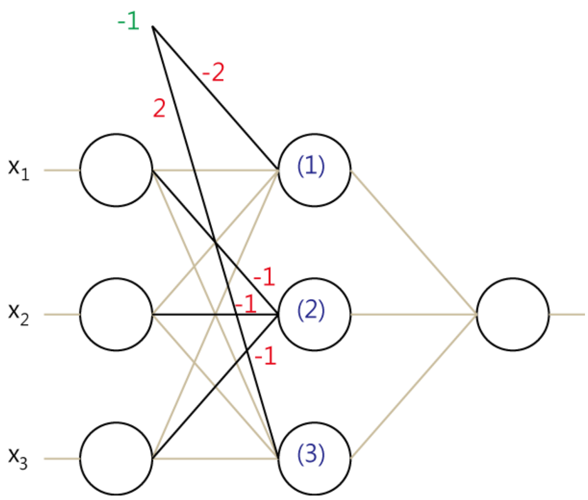
(5.1.5)

The formal way to represent our finding is using the table (5.1.6)

<table>
<tr><th>Patterns</th><th>Perceptron (1)</th><th>Perceptron (2)</th><th>Perceptron (3)</th><th>SUM</th></tr>
<tr><td>$+++$</td><td rowspan="7">$+1$</td><td rowspan="4">-1</td><td>$+1$</td><td>$+1$</td></tr>
<tr><td>$++-$</td><td rowspan="7">-1</td><td rowspan="3">-1</td></tr>
<tr><td>$+-+$</td></tr>
<tr><td>$-++$</td></tr>
<tr><td>$+--$</td><td rowspan="4">$+1$</td><td rowspan="3">$+1$</td></tr>
<tr><td>$-+-$</td></tr>
<tr><td>$--+$</td></tr>
<tr><td>$---$</td><td>-1</td><td>-1</td></tr>
</table>

Table (5.1.6)

Our implement as picture (5.1.7), note that all edge without notification weighted 1



(5.1.7)

(8) To extend the result we found at table (5.1.6). We try the case in four dimensions

<table>
<tr><th>Patterns</th><th>$1+\uparrow$</th><th>$1+\downarrow$</th><th>$3+\uparrow$</th><th>$3+\downarrow$</th><th>$\mathrm{SUM}-1$</th></tr>
<tr><td>$++++$</td><td rowspan="15">$+1$</td><td rowspan="11">-1</td><td rowspan="5">$+1$</td><td>-1</td><td>-1</td></tr>
<tr><td>$+++-$</td><td rowspan="15">$+1$</td><td rowspan="4">$+1$</td></tr>
<tr><td>$++-+$</td></tr>
<tr><td>$+-++$</td></tr>
<tr><td>$-+++$</td></tr>
<tr><td>$++--$</td><td rowspan="11">-1</td><td rowspan="6">-1</td></tr>
<tr><td>$+--+$</td></tr>
<tr><td>$--++$</td></tr>
<tr><td>$-++-$</td></tr>
<tr><td>$-+-+$</td></tr>
<tr><td>$+-+-$</td></tr>
<tr><td>$---+$</td><td rowspan="5">$+1$</td><td rowspan="4">$+1$</td></tr>
<tr><td>$--+-$</td></tr>
<tr><td>$-+--$</td></tr>
<tr><td>$+---$</td></tr>
<tr><td>$----$</td><td>-1</td><td>-1</td></tr>
</table>

Table (5.1.8)

We can add -1 on the SUM column, and can get the desire output we want. To formalize the table, we combine the finding at table (5.1.6) and (5.1.8):

(a) If the d is odd number:

The table is constructed by some rules:

-1 The perceptrons given with that more than odd number +'s fire a positive one, and less than even number +'s fire a negative one. Where odd numbers are belong to $\{1,3 \ldots \mathrm{d}\}$.

-2 Sum up the table values as output.

(b) If the d is even number:

The table is constructed by some rules:

-1 The perceptrons given with that more than odd number +'s fire a positive one, and less than even number +'s fire a negative one. Where even numbers are belong to $\{2,4 \ldots \mathrm{d}\}$.

-2 Sum up the table values with a -1 as output.

The way we transform the table to neural networks is by a simple observation on linear combination of inputs value. With proper weight, the outputs value will be monotonic function. So the neural network is existed.

Consider about k's perceptron in d dimensions. If k is an odd number, than we fire the input more than and equal to k's $+$ in +1. Give the weight $(\mathrm{w}_1, \mathrm{w}_2, \dots, \mathrm{w}_\mathrm{d}) = (1,1,\dots,1)$. We have the inequality (5.1.9)

$$\mathrm{k} - (\mathrm{d} - \mathrm{k}) - \mathrm{C}_{odd} \geq +1$$

$$\mathrm{C}_{odd} \leq 2\mathrm{k} - \mathrm{d} - 1 \qquad (5.1.9)$$

If k is an even number, than we fire the input less than k's $+$ in +1. Give the weight $(\mathrm{w}_1, \mathrm{w}_2, \dots, \mathrm{w}_\mathrm{d}) = (-1-,1,\dots,-1)$. We have the inequality (5.1.10)

$$-(\mathrm{k} - 1) + \left(\mathrm{d} - (\mathrm{k} - 1)\right) - \mathrm{C}_{even} \geq +1$$

$$\mathrm{C}_{\mathrm{even}} \leq \mathrm{d} - 2\mathrm{k} + 1 \qquad (5.1.10)$$

Therefore we have perceptrons:

- Hidden Layer:
  - -1 $\mathrm{Perceptron}(\mathrm{k} = 0 \bmod 2) = g_{even}(\mathrm{x}|\mathrm{d} - 2\mathrm{k} + 1, -1, -1, \dots, -1)$
  - -2 $\mathrm{Perceptron}(\mathrm{k} = 1 \bmod 2) = g_{odd}(\mathrm{x}|2\mathrm{k} - \mathrm{d} - 1,1,1,\dots,1)$

- Outputs Layer:
  - -1 If $\mathrm{d} = 0 \bmod 2$:
    $\mathrm{Perceptron}(1) = g(\mathrm{x}|1,1,1,\dots,1)$
  - -2 Else $\mathrm{d} = 1 \bmod 2$:
    $\mathrm{Perceptron}(1) = g(\mathrm{x}|0,1,1,\dots,1)$

For a conclusion, we get the d-d-1 neural networks.

$$\mathrm{W}_{\mathrm{ij}}^{(1)} = \begin{cases} (2\mathrm{j} - \mathrm{d} - 1)(-1)^{\mathrm{j}+1}, & if\ i = 0 \\ (-1)^{\mathrm{j}+1}, & \text{otherwise} \end{cases}$$

$$\mathrm{W}_{\mathrm{ij}}^{(2)} = \begin{cases} \mathrm{d} + 1 \bmod 2, & \text{if } \mathrm{i} = 0 \\ 1, & otherwise \end{cases}$$

5.2 Limitations of Neural Networks

(1) Express the formula at left hand side inequality

$$(\mathrm{N^D}+1)^3 = \mathrm{N^{3D}} + \mathrm{N^{2D}} + \mathrm{N^D} + 1 \leq 3\mathrm{N^{3D}} + 1 \leq \mathrm{N^{3D+1}} + 1 \qquad (5.2.1)$$

The inequality is satisfy for $\mathrm{D} \geq 1$ and $\mathrm{N} \geq 3$.

(2) Proof $2^{\mathrm{N}} > \mathrm{N^k} + 1$, if $\mathrm{N} \geq 3\mathrm{k}\log_2 \mathrm{k}$, assume $\mathrm{k} = \Delta \geq 1$ and $\mathrm{N} > 1$.

(a) If $\mathrm{k} = 1$, $2^{\mathrm{N}} > N + 1, \forall N \in \mathbb{N}$

(b) If $\mathrm{k} \geq 2$, Consider a function f such that

$$\mathrm{f(n)} = \mathrm{n} - \mathrm{k}\log_2 \mathrm{n} - \tfrac{1}{2} \qquad (5.2.2)$$

-1 If $\mathrm{n} > k\log_2 \mathrm{e}$,

$$\mathrm{f'(n)} = 1 - \tfrac{\mathrm{k}}{\mathrm{n}}\log_2 \mathrm{e} > 0 \qquad (5.2.3)$$

-2 If $\mathrm{n} = 3\mathrm{k}\log_2 \mathrm{k}$,

$$\begin{aligned}\mathrm{f(n)} &= 3\mathrm{k}\log_2 \mathrm{k} - \mathrm{k}(\log_2 3 + \log_2 \mathrm{k} + \log_2\log_2 \mathrm{k}) - \frac{1}{2}\\ &= \mathrm{k}(2\log_2 \mathrm{k} - \log_2 3 - \log_2\log_2 \mathrm{k}) - \frac{1}{2}\\ &= \mathrm{k}\left(\log_2 \tfrac{\mathrm{k}^2}{3\log_2 \mathrm{k}}\right) - \tfrac{1}{2} > 0 \qquad (5.2.4)\end{aligned}$$

By inequality (5.2.3) and (5.2.4), and $3\mathrm{k}\log_2 \mathrm{k} \geq \mathrm{k}\log_2 \mathrm{e}$. We get the inequality

$$\mathrm{f(n)} > 0, \text{ for } \mathrm{n} \geq 3\mathrm{k}\log_2 \mathrm{k}, \ \mathrm{k} \geq 2 \qquad (5.2.5)$$

(c) Use the inequality (5.2.5), we have

$$\begin{aligned}&\mathrm{f(n)} > 0\\ &\rightarrow \mathrm{n} > k\log_2 \mathrm{n} + \frac{1}{2}\\ &\rightarrow 2^{\mathrm{n}} > \sqrt{2}\mathrm{n^k} > \mathrm{n^k} + 1 \qquad (5.2.6)\end{aligned}$$

Proof is done.

(3) Suppose we have N nodes, assume

$$\mathrm{N} \geq 3(3(\mathrm{d}+1)+1)\log_2(3(\mathrm{d}+1)+1) \geq 3 \tag{5.2.7}$$

By proof of (2), with $\Delta = 3(\mathrm{d}+1)+1 > 1$, we get

$$\mathrm{N}^{\Delta} + 1 < 2^{\mathrm{N}} \tag{5.2.8}$$

By proof of (1), with $\mathrm{D} = \mathrm{d}+1 > 1$ and $\mathrm{N} \geq 3$, we get

$$(\mathrm{N}^{\mathrm{D}} + 1)^3 \leq \mathrm{N}^{\Delta} + 1 < 2^{\mathrm{N}} \tag{5.2.9}$$

We know that several edges of our d-3-1 neural networks has fixed. We can only change three perceptrons of hidden layer. Each of them has D inputs.

By problem 3.2, we know that the

$$\text{VC dimension of linear model with `sign gate' with d's input is D.} \tag{5.2.10}$$

Suppose each perceptron can generate $L_{GA}(\mathrm{N})$ patterns. We have the inequality

$$L_{GA}(\mathrm{N}) \leq \mathrm{B}(\mathrm{N}, \mathrm{D}+1) \leq \mathrm{N}^{\mathrm{D}} + 1 \tag{5.2.11}$$

Therefore, three independent perceptrons can generate $L_{3GA}(\mathrm{N})$ patterns but less than 2^{N}

$$L_{G3A}(\mathrm{N}) \leq (\mathrm{N}^{\mathrm{D}} + 1)^3 \leq 2^{\mathrm{N}} \tag{5.2.12}$$

It means we cannot generate all patterns. So we have

$$VCD(G_{3A}) < 3(3(\mathrm{d}+1)+1)\log_2(3(\mathrm{d}+1)+1) \geq 3 \tag{5.2.13}$$

(4) Neural networks can implement any Boolean function and $VCD(G_M)$ is bounded, this statement is true. For a given dimensions d, our Boolean functions only can produce 2^{d}'s points at most, which mean our maximum $VCD(G_M)$ Is bounded by d.

(5) Let $\mathbb{S}$ be some set of N input vectors for the neural networks NN. By (5.2.10), we know that a perceptron G can only compute at most $|\mathbb{X}|^{\mathrm{D}} + 1$ different functions from any finite set $\mathbb{X} \subset \mathbb{R}^{\mathrm{d}} \to \{0,1\}$. Denote the total number of weight $\mathrm{w} = \mathrm{M} * \mathrm{d}$.

Consider the relation as a mapping, so our NN generate at most have

$$\prod_G(\mathrm{N}^{\mathrm{D}} + 1) \le \mathrm{N}^{2\mathrm{w}} \tag{5.2.14}$$

different functions from $\mathbb{S}$ into $\{0,1\}$.

Suppose we yield all possible patterns, which means 2^{N} functions. By (5.2.14) we get

$$\prod_G(\mathrm{N}^{\mathrm{D}} + 1) = 2^{\mathrm{N}} \le \mathrm{N}^{2\mathrm{w}} \tag{5.2.15}$$

It imply

$$\mathrm{N} \le 2\mathrm{w} * \log_2 \mathrm{N} \tag{5.2.16}$$

Consider about big-O notation, we claim that

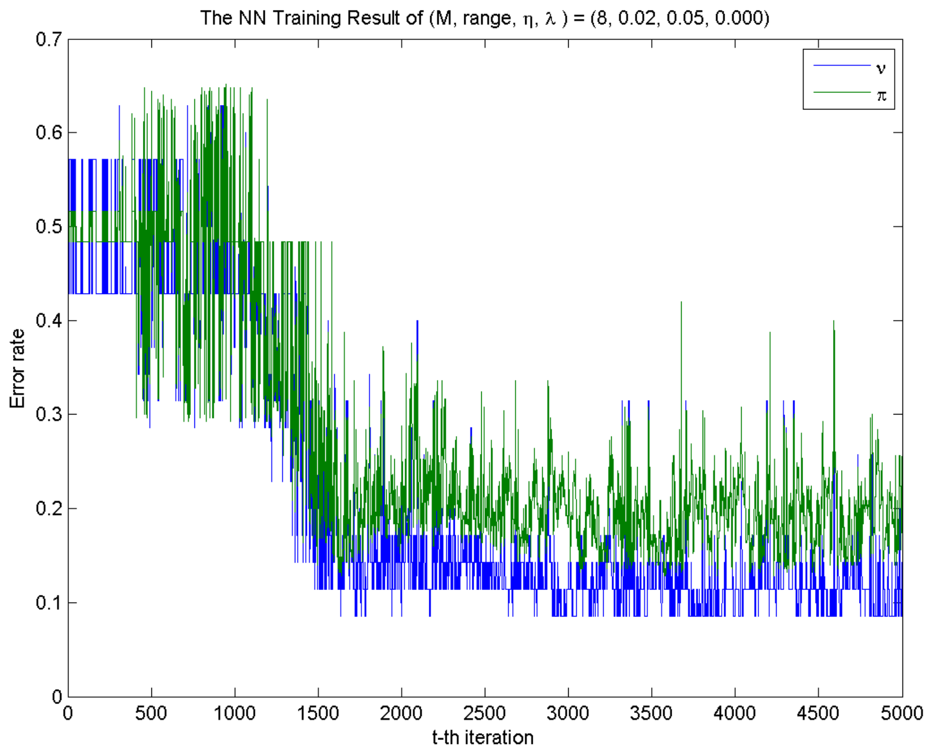
$$\log_2 \mathrm{N} = \mathrm{O}(\log \mathrm{w}) \tag{5.2.17}$$

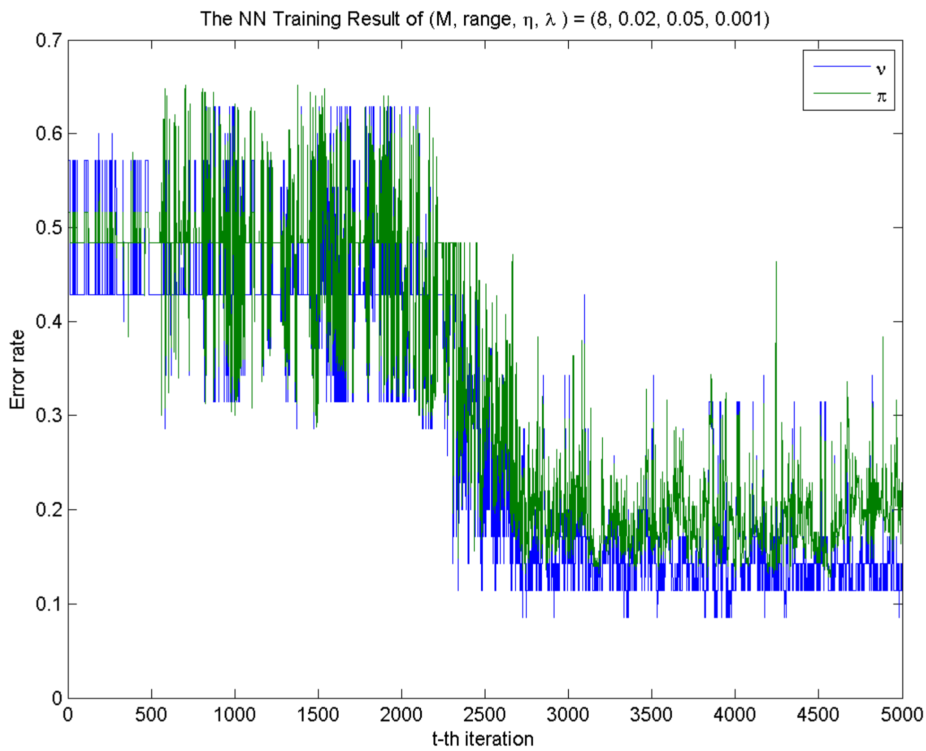
Combine (5.2.16) and (5.2.17), we have the result

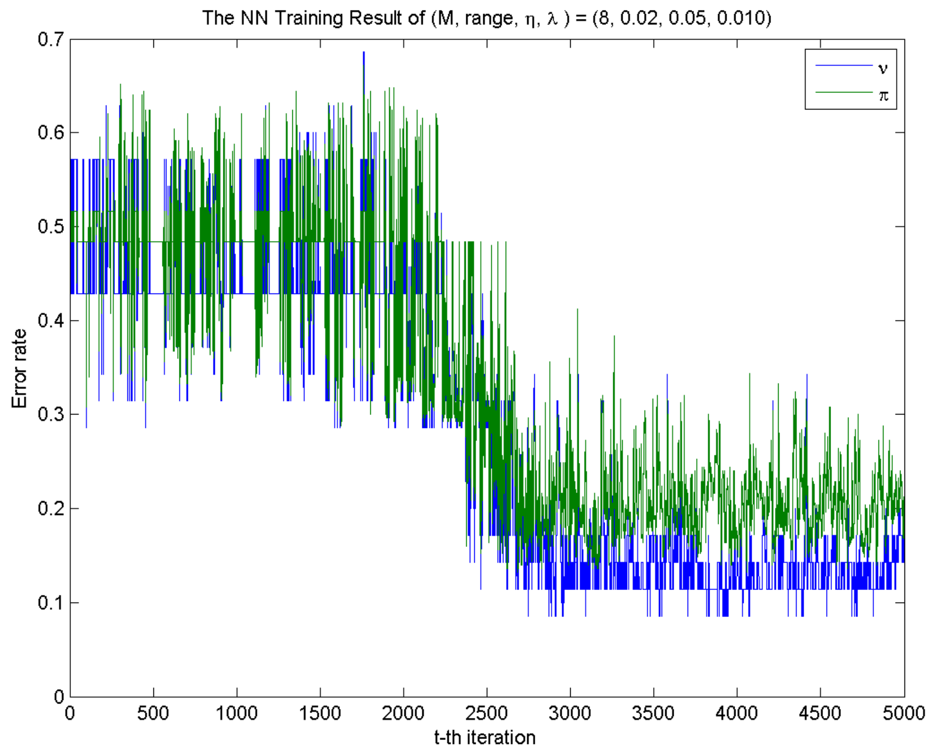
$$VCD(G_M) = \mathrm{N} = \mathrm{O}(\mathrm{w} * \log \mathrm{w}) = \mathrm{O}(\mathrm{M} * \mathrm{d} * \log(\mathrm{M} * \mathrm{d})) \tag{5.2.18}$$

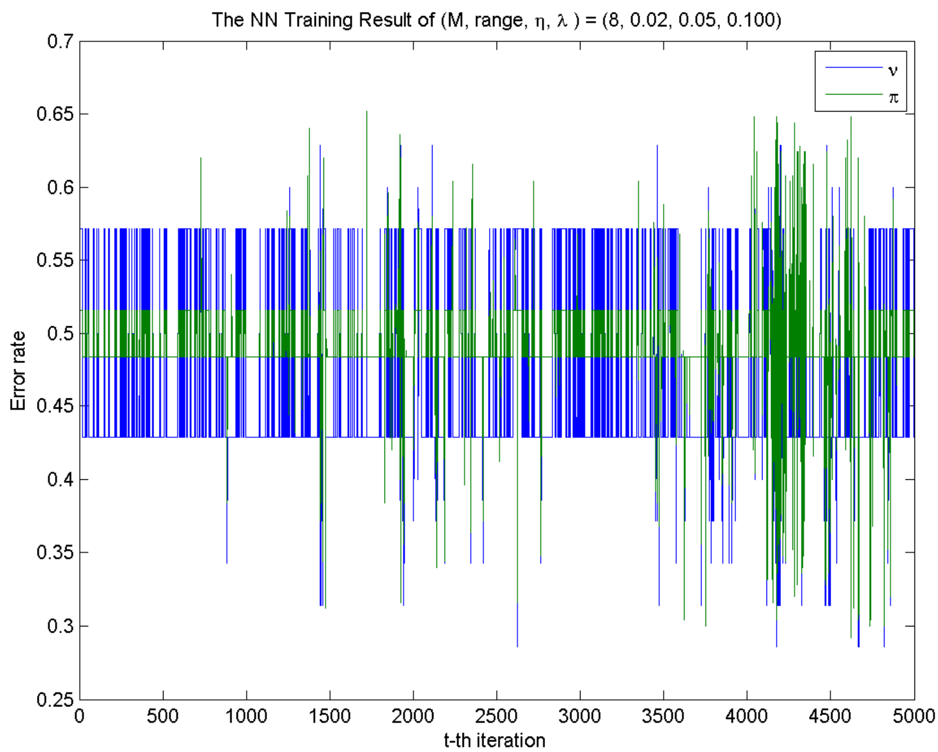
5.3 Experiment with Weight Decay in Neural Network

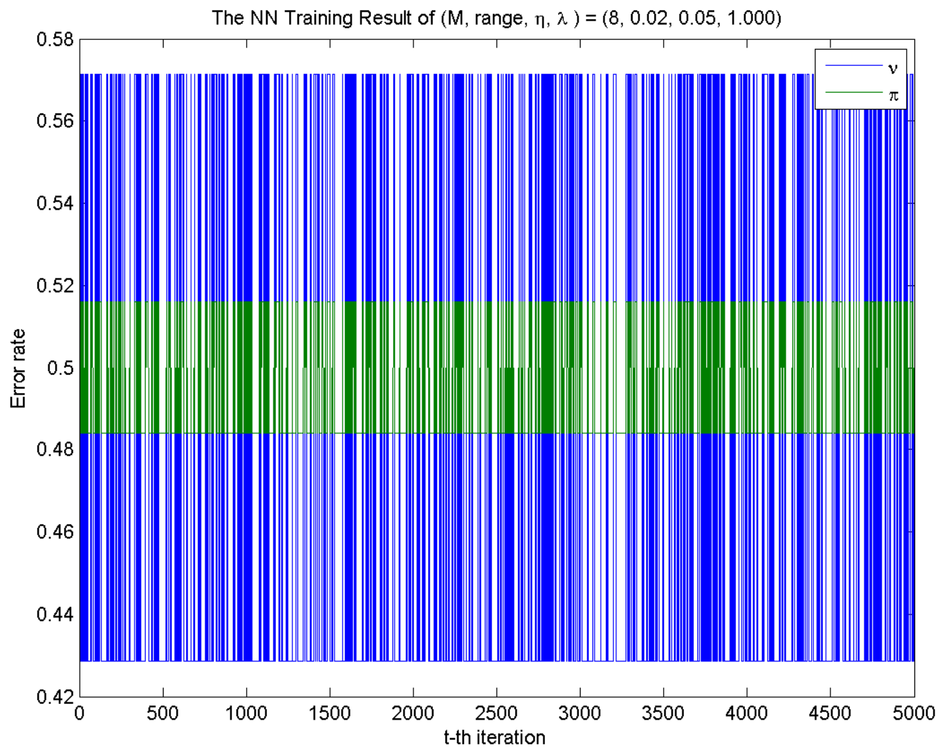
(1) The implement of the algorithm isn't very complicated, but should be careful about one thing, that we have to change the δ^2 term of back propagation. Plot the figure as fellow,






The NN Training Result of (M, range, η, λ) = (8, 0.02, 0.05, 0.010)
ν
π
Error rate
t-th iteration
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000


The NN Training Result of (M, range, η, λ) = (8, 0.02, 0.05, 0.100)
ν
π
Error rate
t-th iteration
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000



Conclusion:

-1 Error rate doesn’t change much at $\mathrm{K} = \{0, 0.001, 0.01\}$. But we can see the curve kind of border when K got smaller.

-2 When $\mathrm{K} = 0.1$ or higher, the curve cannot converge at T = 5000.

5.4 Experiment with Naïve Bayes

(1) The results show as Table (5.4.1), (5.4.2) and (5.4.3)

	$K = 2$	$K = 5$	$K = 10$	$K = 20$	$K = 50$
$\nu(g_K)$	0.22	0.17	0.12	0.11	0.02
$\hat{\pi}(g_K)$	0.211	0.204	0.211	0.234	0.301

Table (5.4.1)

	$K = 2$	$K = 5$	$K = 10$	$K = 20$	$K = 50$
$\nu(g_K)$	0.22	0.17	0.12	0.11	0.02
$\hat{\pi}(g_K)$	0.211	0.204	0.215	0.267	0.384

Table (5.4.2)

	$K = 2$	$K = 5$	$K = 10$	$K = 20$	$K = 50$
$\nu(g_K)$	0.22	0.17	0.12	0.11	0.02
$\hat{\pi}(g_K)$	0.211	0.204	0.211	0.238	0.299

Conclusion:

-1 It may not good for a big K, looks like an over fitting result.

-2 For a testing data in which never showed up our training bin, it may raise a problem. We don't know how to classify such case, so we sign it as a guess.

1. Guess it is $+1$, we get Table (5.4.1).
2. Guess it is -1, we get Table (5.4.2).
3. Guess it by random guess with function, $\mathrm{sign}\big(\mathrm{random} \in (-0.1, 0.9)\big)$, we get Table (5.4.3).