**Homework #5**

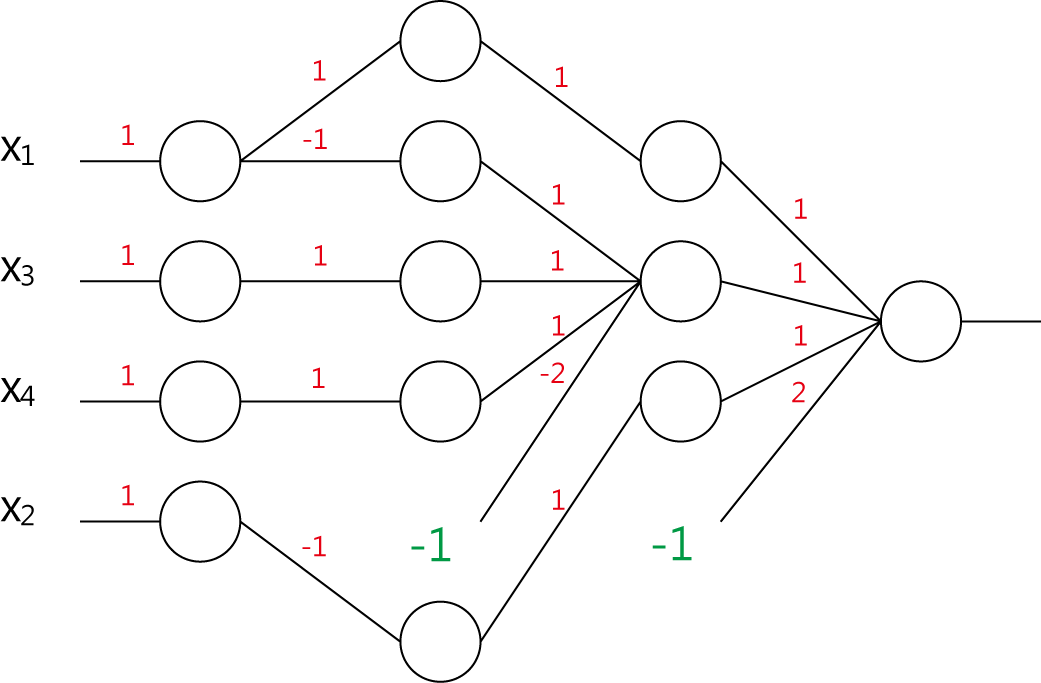
1. **Powerfulness of Neural Networks**
   1. We implement , with .

To proof it can work, we show the function 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 .

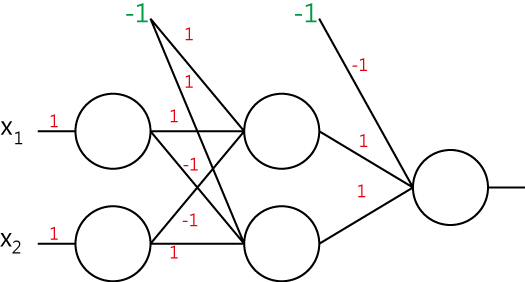
* 1. We implement , where

To proof it works, we show the function 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 .

* 1. , where . The result is trivial.
  2. Illustrated the result by the combination of , and perceptron we define above as to a Neural Networks (NN) (5.1.1).



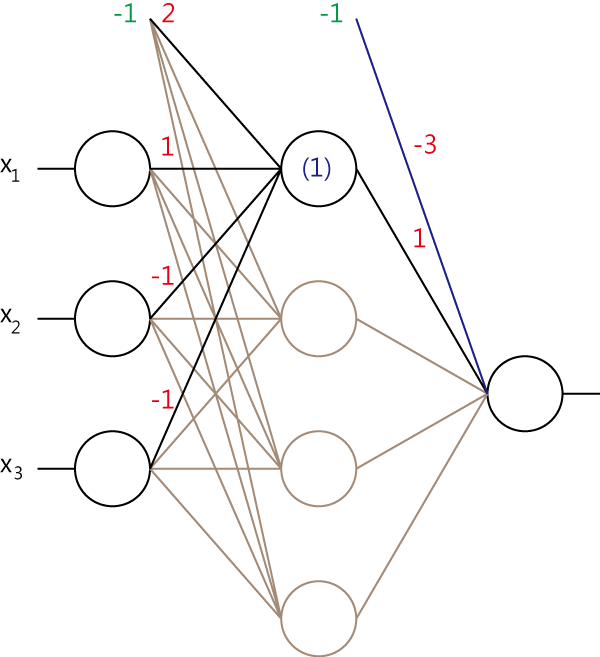
* 1. By (1), (2) and (3). We know that the 3 element gates can be generated as a NN form. For any Boolean system, we can change it to conjunctive normal form. Thus, we can systematically implement any Boolean function by back propagate each gates to a NN by Boolean operation.
  2. Try the simpler case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

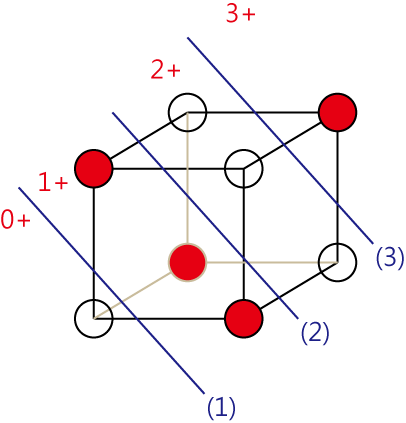
(1)

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



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.

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

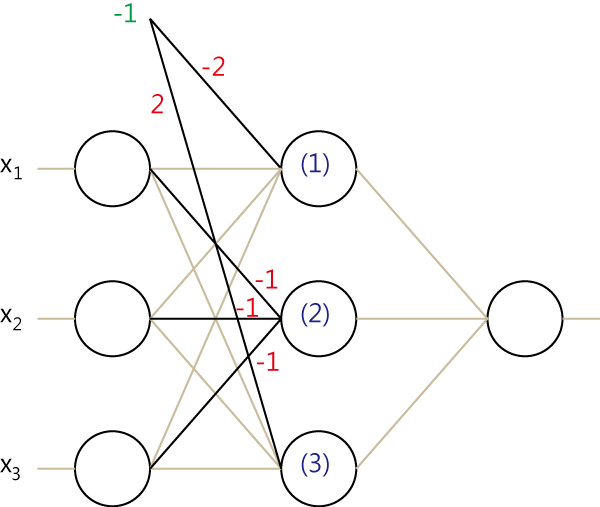


The formal way to represent our finding is using the table (5.1.6)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Patterns | Perceptron (1) | Perceptron (2) | Perceptron (3) |  |
|  |  |  |  |  |
|  |  |  |
|  |
|  |
|  |  |  |
|  |
|  |
|  |  |  |

Table (5.1.6)

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



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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Patterns |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |
|  |
|  |
|  |
|  |  |  |
|  |
|  |
|  |
|  |
|  |
|  |  |  |
|  |
|  |
|  |
|  |  |  |

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):

* + 1. 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 .
      2. Sum up the table values as output.
    1. 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. Sum up the table values with a 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 . We have the inequality (5.1.9)

If k is an even number, than we fire the input less than k’s in +1. Give the weight . We have the inequality (5.1.10)

Therefore we have perceptrons:

* Hidden Layer:
* Outputs Layer:
  + - 1. If :
      2. Else :

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

1. **Limitations of Neural Networks**
   1. Express the formula at left hand side inequality

The inequality is satisfy for .

* 1. Proof , if , assume and .
     1. If ,
     2. If Consider a function f such that

* + - 1. If ,

* + - 1. If ,

By inequality (5.2.3) and (5.2.4), and . We get the inequality

* + 1. Use the inequality (5.2.5), we have

Proof is done.

* 1. Suppose we have N nodes, assume

By proof of (2), with , we get

By proof of (1), with and , we get

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

VC dimension of linear model with `sign gate’ with d’s input is .

Suppose each perceptron can generate patterns. We have the inequality

Therefore, three independent perceptrons can generate patterns but less than

It means we cannot generate all patterns. So we have

* 1. Neural networks can implement any Boolean function and is bounded, this statement is true. For a given dimensions d, our Boolean functions only can produce ’s points at most, which mean our maximum Is bounded by d.
  2. Let 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 different functions from any finite set . Denote the total number of weight .

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

different functions from into .

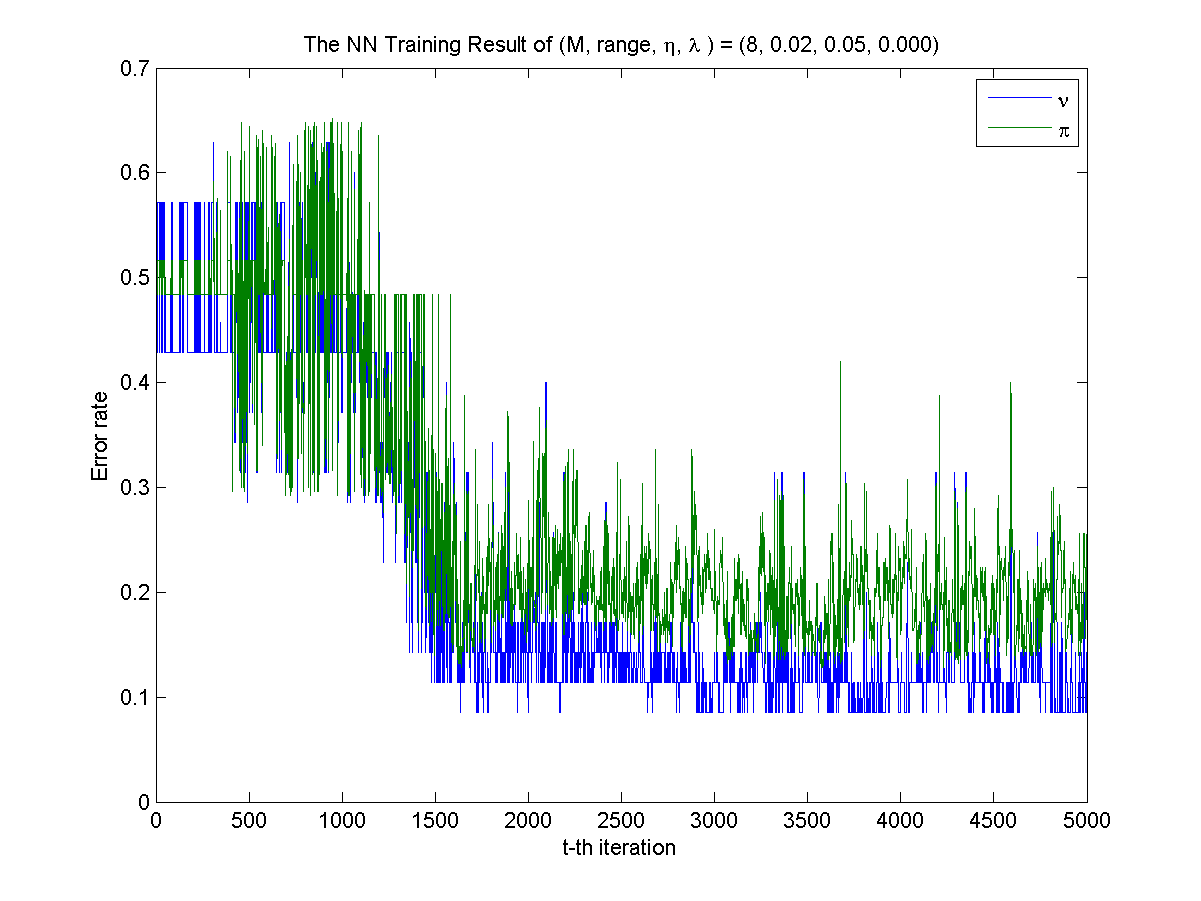
Suppose we yield all possible patterns, which means functions. By (5.2.14) we get

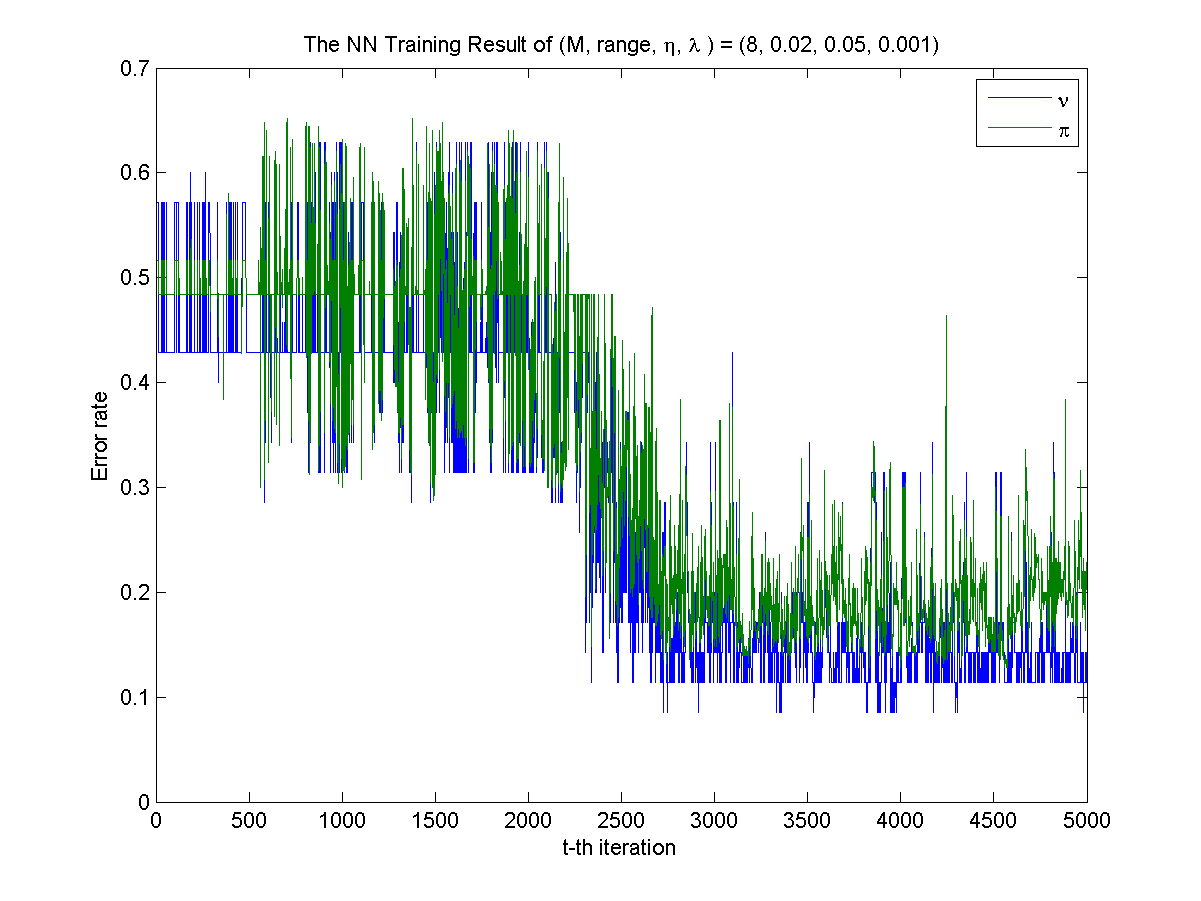
It imply

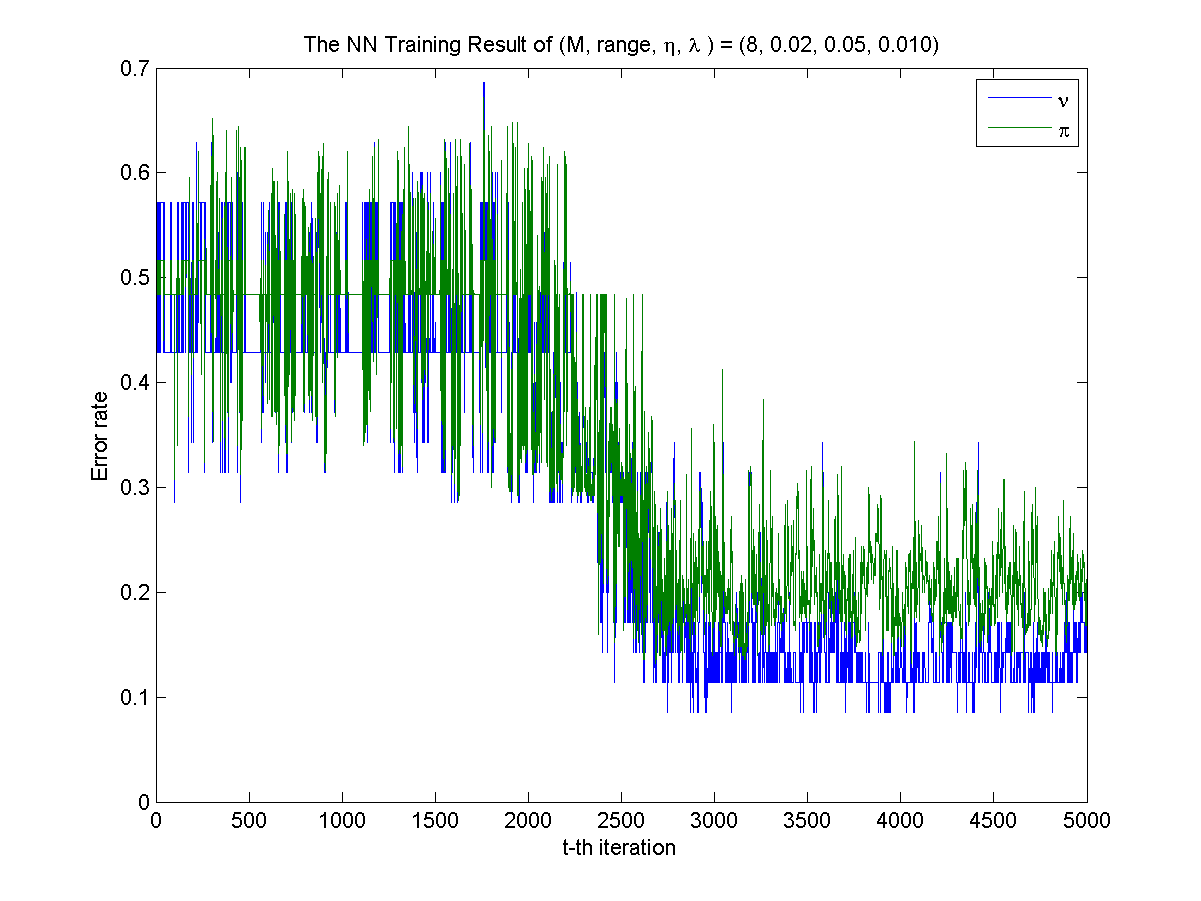
Consider about big-O notation, we claim that

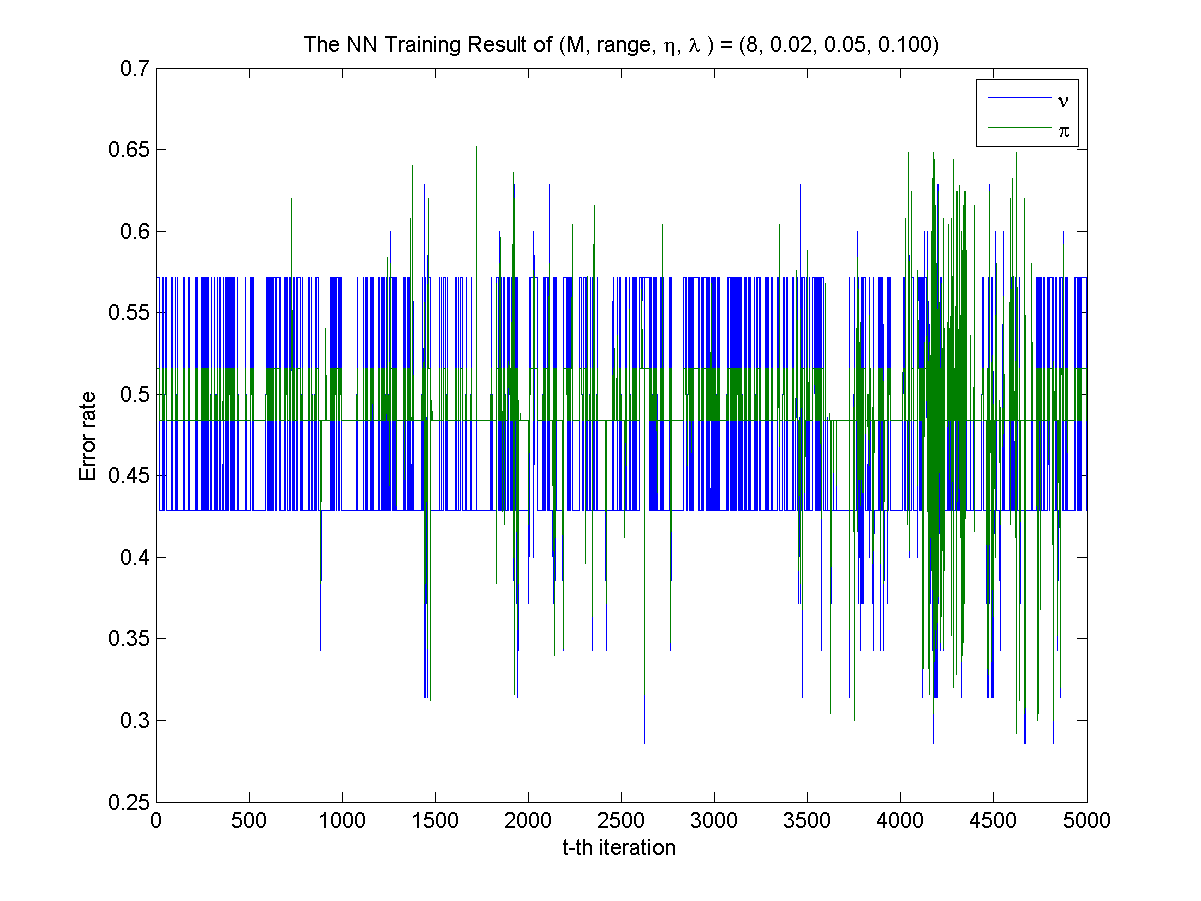
Combine (5.2.16) and (5.2.17), we have the result

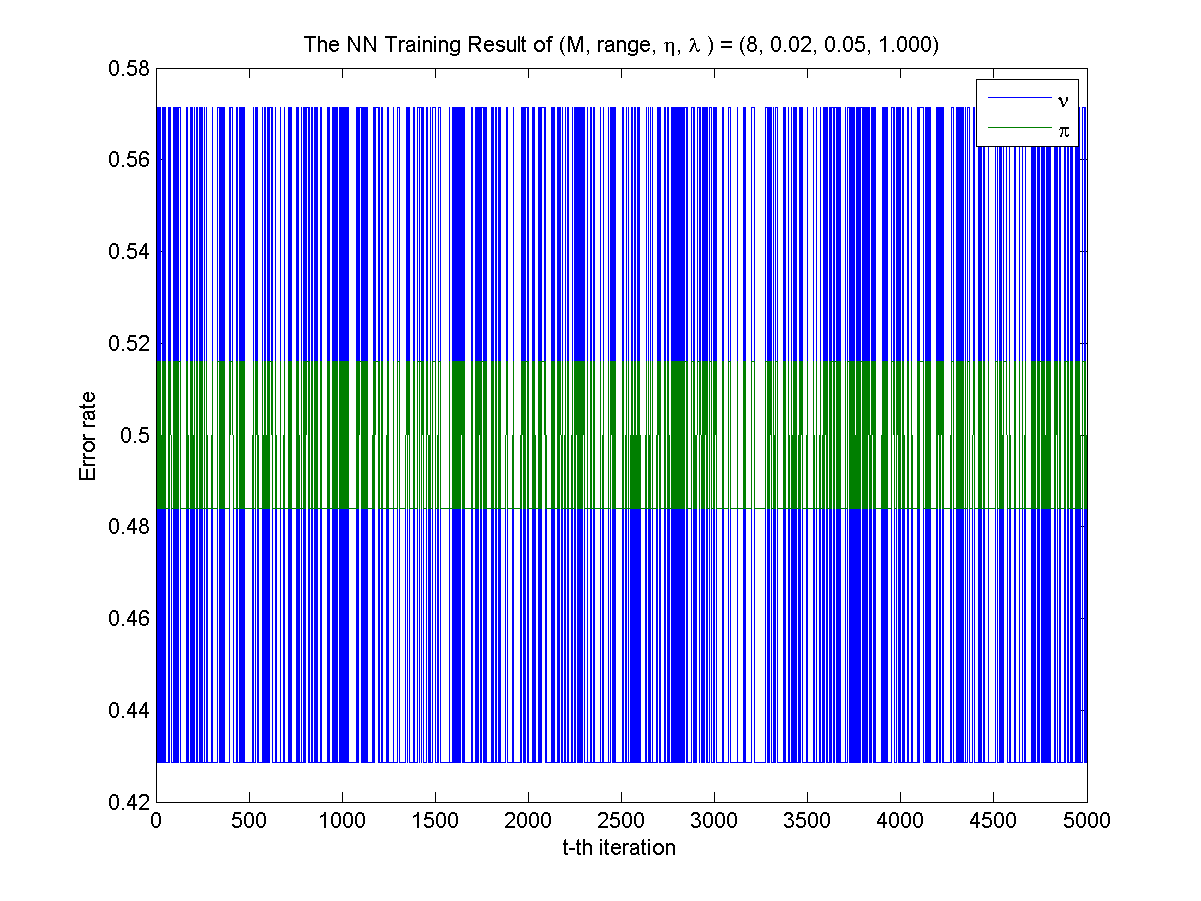
1. **Experiment with Weight Decay in Neural Network**
   1. The implement of the algorithm isn’t very complicated, but should be careful about one thing, that we have to change the term of back propagation. Plot the figure as fellow,











Conclusion:

* + - 1. Error rate doesn’t change much at . But we can see the curve kind of border when K got smaller.
      2. When or higher, the curve cannot converge at T = 5000.

1. **Experiment with Naïve Bayes**
   1. The results show as Table (5.4.1), (5.4.2) and (5.4.3)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Table (5.4.1)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  | Table (5.4.2) |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

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 , we get Table (5.4.1).
         2. Guess it is , we get Table (5.4.2).
         3. Guess it by random guess with function, , we get Table (5.4.3).