A Technical Report On

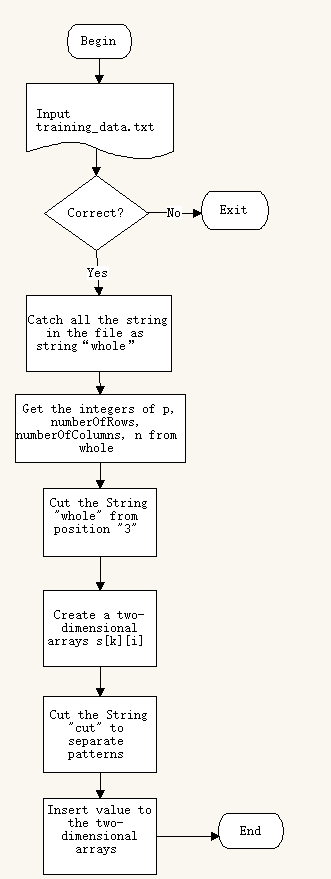
Hebb neural network

Contents

1. Introduction………………………………………………3
2. Code description………………………………………….4
   1. Training data part…………………………………….4
   2. Target data part……………………………………….7
   3. Weight part…………………………………………..10
   4. Test data part…………………………………………13
3. Flowchart…………………………………………………17
4. Program Test………………………………………………18
   1. Test Example…………………………………………18
   2. How to complete the program…………………….......21
5. Conclusion………………………………………………...22
6. References………………………………………………...23
7. Introduction

This report:

1. Discusses a JAVA program designed to make the machine learn patterns to recognize all the 52 letters include uppercase and lowercase letters. The program will be work depends on three different files “training data”, “target data” and “test data”. The “training data” file let the program learn patterns made by users. The “target data” make the program compare to the “test data” and “training data”. The “test data” also provided by users that used to test the program worked or not.
2. Uses Hebb neural network. Neural networks represent a type of computing that is based on the way that the brain performs computations. Neural networks are good at fitting non-linear functions and recognizing patterns. (Sanger, 1989b) Consequently, they are used in the aerospace, automotive, banking, defense, electronics, entertainment, financial, insurance, manufacturing, oil and gas, robotics, telecommunications, and transportation industries (Oja, E. 1992).
3. Illustrates how I developed the program to learn and recognize patterns. And how the program can be completed better.
4. Code description
   1. Training data part

This is the flowchart of this part. In this part we will get a two-dimensional arrays s[k][i], which depends on the training data.

**Part 1:**

Scanner input = **new** Scanner(Paths.*get*("training\_data.txt"))

pattern = input.nextLine();

RC = input.nextLine();

**while** (input.hasNext()) {

take = input.next();

whole = whole + take;

}

I used the class Scanner and the method Paths.*get* to read the "training\_data.txt". Then use the method hasNext() to put the strings in string “whole”.

**Part 2:**

String[] rowAndColumn = RC.split(" ");

**int** p = Integer.*parseInt*(String.*valueOf*(pattern));

**int** numberOfRows = Integer.*parseInt*(String.*valueOf*(rowAndColumn[0]));

**int** numberOfColumns = Integer.*parseInt*(String.*valueOf*(rowAndColumn[1]));

**int** n = numberOfRows \* numberOfColumns;

I split the string RC to get row and column, also get the p and n. Because the “whole” is a string, I transit the string to integer by using the method Integer.*parseInt*()

**Part 3:**

String[] patterns = **new** String[p];

**for** (**int** a = 0; a < p; a++) {

String cut = whole.substring((n + 1) \* a,

(n + 1) \* (a + 1));

patterns[a] = cut;

}

I Create a String array patterns to store the patterns from test\_data.txt. I cut the string “whole” depends on the position of the beginning and end of every pattern.

**Part 4:**

**int**[][] s = **new** **int**[p+1][n+1];

**for** (**int** k = 1; k < p+1; k++) {

**for** (**int** i = 1; i < n+1; i++) {

**char**[] temp = patterns[k-1].toCharArray();

s[k][i] = Integer.*parseInt*(String.*valueOf*(temp[i-1])) \* 2 - 1;}}

I create a two-dimensional array s[k][i] to store the 0 and 1 from patterns. Because it is a two-dimensional array, I use two for loops to insert the value.

**Part 5:**

The training data will look like this:

52

9 7

0 0 1 1 0 0 0

0 0 0 1 0 0 0

0 0 0 1 0 0 0

0 0 1 0 1 0 0

0 0 1 0 1 0 0

0 1 1 1 1 1 0

0 1 0 0 0 1 0

0 1 0 0 0 1 0

1 1 1 0 1 1 1

A

1 1 1 1 1 1 0

0 1 0 0 0 0 1

0 1 0 0 0 0 1

0 1 0 0 1 0 1

0 1 1 1 1 1 0

0 1 0 0 0 0 1

0 1 0 0 0 0 1

0 1 0 0 0 0 1

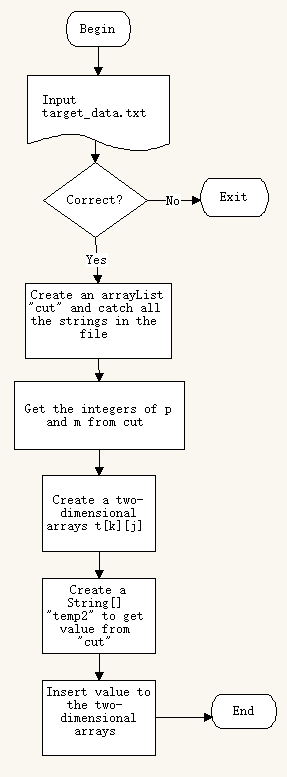
1 1 1 0 1 1 0

B

……

52 means there are 52 patterns. The 9 and 7 means the numberOfRows and numberOfColumns. You can see the matrix combine with 0 and 1. Also, the appearance of all the 1 looks like letter A and B.

* 1. Target data part



This is the flowchart of this part. In this part we will get a two-dimensional arrays t[k][j], which depends on the target data.

**Part 1:**

BufferedReader hunt = null;

ArrayList<String> cut = **new** ArrayList<String>();

String path = "target\_data.txt";

String temp;

hunt = **new** BufferedReader(**new** FileReader(path));

**while** ((temp = hunt.readLine()) != **null**) {

cut.add(temp);

}

I used the class BufferedReader and the method FileReaderto read the “target\_data.txt ". Then use the method readLine() to put the strings in string “cut”.

**Part 2:**

**int** m = Integer.*parseInt*(String.*valueOf*((String) cut.get(0)));

**int** p = cut.size() - 1;

I get the m from the arraylist cut’s first position. Then get the p by the method size() of BufferedReader.

**Part 3:**

**int**[][] t = **new** **int**[p+1][m+1];

**for** (**int** k = 1; k < p+1; k++) {

**for** (**int** j = 1; j < m+1; j++) {

String[] temp2 = cut.get(k).split(" ");

**int** count = Integer.*parseInt*(String.*valueOf*(temp2[j-1]));

t[k][j] = count \* 2 - 1;

}

}

I create a two-dimensional array t[k][j] to store the 0 and 1 from “temp2”. Because it is a two-dimensional array, I use two for loops to insert the value.

**Part 4:**

The target data will look like this:

11

1 0 0 0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0 0 0 0

0 0 1 0 0 0 0 0 0 0 0

0 0 0 1 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0 0 0 0

0 0 0 0 0 1 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 0

0 0 0 0 0 0 0 1 0 0 0

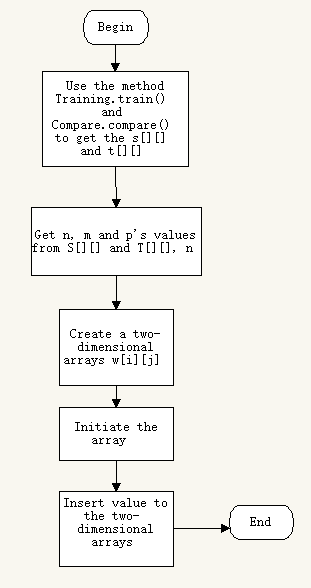
0 0 0 0 0 0 0 0 1 0 0

0 0 0 0 0 0 0 0 0 1 0

……

For example, the “1 0 0 0 0 0 0 0 0 0 0” can be corresponded letter “A”. The “11” means every line has 11 numbers. If you have many different patterns, you could change it to a bigger number.

* 1. Weight part



This is the flowchart of this part. In this part we will get a two-dimensional arrays w[i][j], which depends on the S[][] and T[][].

**Part 1:**

**int**[][] S = Training.*train*();

**int**[][] T = Compare.*compare*();

I Use the methods Training.train() and Compare.compare() to get the s[][] and t[][]. When you use these methods, the arrays will return.

**Part 2:**

**int** n = 0, m = 0, p = 0;

n = S[n].length - 1;

m = T[m].length - 1;

p = S.length - 1;

I create three integers: n, m and p. Then get then from the s[][] and t[][].

**Part 3:**

**int**[][] w = **new** **int**[n + 1][m + 1];

**for** (**int** i = 0; i < n + 1; i++) {

**for** (**int** j = 0; j < m + 1; j++) {

w[i][j] = 0;

}

}

I Create a two-dimensional arrays w[i][j], and Initiate the array.

**Part 4:**

**int** xi, yj, dw;

**for** (**int** k = 1; k < p + 1; k++) {

**for** (**int** i = 1; i < n + 1; i++) {

**for** (**int** j = 1; j < m + 1; j++) {

xi = S[k][i];

yj = T[k][j];

dw = xi \* yj;

w[i][j] = w[i][j] + dw;

}

}

}

Insert value to the two-dimensional arrays by using 3 for loops. Because the weight depends on two arrays, we need use 3 for loops.

**Part 5:**

The value in w[][] will look like this:

1 1 -20

1 2 -22

1 3 -26

1 4 -8

1 5 0

1 6 14

1 7 6

1 8 12

1 9 4

1 10 8

1 11 12

2 1 -30

2 2 -32

2 3 -20

2 4 -6

2 5 10

2 6 20

2 7 24

2 8 18

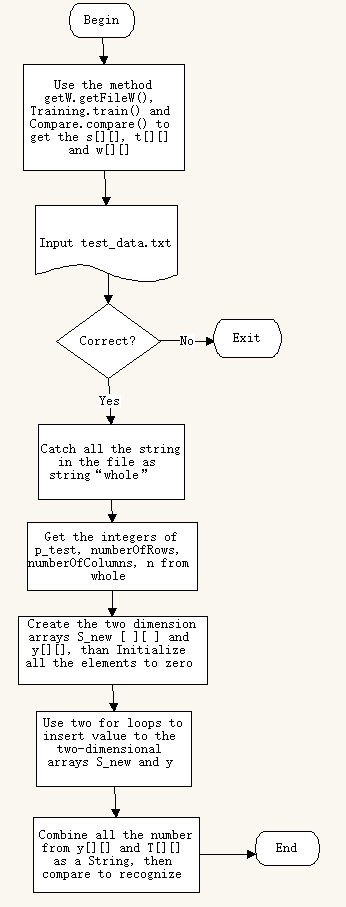
2 9 18

2 10 18

……

The first line “1 1 -20” means w[1][1] = -20.

* 1. Test data part



This is the flowchart of this part. In this part we will get two-dimensional arrays S\_new[k][i] and y[k][j]. Then the program will try to recognize each patterns.

**Part 1:**

**int**[][] w = getW.*getFileW*();

**int**[][] T = Compare.*compare*();

**int**[][] S = Training.*train*();

I will Use the method getW.getFileW(), Training.train() and Compare.compare() to get the s[][], t[][] and w[][].

**Part 2:**

Scanner input = **new** Scanner(Paths.*get*("test\_data.txt"))

pattern = input.nextLine();

RC = input.nextLine();

**while** (input.hasNext()) {

take = input.next();

whole = whole + take;

}

I used the class Scanner and the method Paths.*get* to read the "test\_data.txt". Then use the method nextLine() to put the strings in string “whole”.

**Part 3:**

String[] rowAndColumn = RC.split(" ");

**int** p\_test = Integer.*parseInt*(String.*valueOf*(pattern));

**int** numberOfRows = Integer.*parseInt*(String.*valueOf*(rowAndColumn[0]));

**int** numberOfColumns = Integer.*parseInt*(String.*valueOf*(rowAndColumn[1]));

**int** n = numberOfRows \* numberOfColumns;

I split the string RC to get row and column, also get the p\_test and n. Because the “whole” is a string, I transit the string to integer by using the method Integer.*parseInt*()

**Part 4:**

String[] patterns = **new** String[p\_test];

**for** (**int** a = 0; a < 9; a++) {

String cut = whole.substring((numberOfRows \* numberOfColumns + 1) \* a,

(numberOfRows \* numberOfColumns + 1) \* (a + 1));

patterns[a] = cut;

}

**if** (p\_test > 10) {

**for** (**int** b = 9; b < p\_test; b++) {

String cut2 = whole.substring(

(numberOfRows \* numberOfColumns + 2) \* (b - 9) + (numberOfRows \* numberOfColumns + 1) \* 9,

(numberOfRows \* numberOfColumns + 2) \* (b - 8) + (numberOfRows \* numberOfColumns + 1) \* 9);

patterns[b] = cut2;

}

}

I want to cut the string “whole” as patterns. However, there are 2 situations: if the patterns are less than 10, the beginning position is (numberOfRows \* numberOfColumns + 1) \* a and the end position is (numberOfRows \* numberOfColumns + 1) \* (a + 1)); if the patterns are more than 10, the beginning position will change as (numberOfRows \* numberOfColumns + 2) \* (b - 9) + (numberOfRows \* numberOfColumns + 1) \* 9, also the end position is (numberOfRows \* numberOfColumns + 2) \* (b - 8) + (numberOfRows \* numberOfColumns + 1) \* 9).

**Part 5:**

**int**[][] S\_new = **new** **int**[p\_test + 1][n + 1];

**for** (**int** k = 1; k < p\_test + 1; k++) {

**for** (**int** i = 1; i < n + 1; i++) {

**char**[] temp = patterns[k - 1].toCharArray();

S\_new[k][i] = Character.*digit*(temp[i - 1], 10) \* 2 - 1;

}

}

I create a two-dimensional array S\_new[k][i] to store the 0 and 1 from patterns. Because it is a two-dimensional array, I use two for loops to insert the value.

**Part 6:**

**int** m = 0;

m = T[m].length - 1;

**int** y\_in;

**int** y[][] = **new** **int**[p\_test + 1][m + 1];

**for** (**int** k = 0; k < p\_test + 1; k++) {

**for** (**int** j = 0; j < m + 1; j++) {

y[k][j] = 0;

}

}

I Create a two-dimensional arrays y[k][j], and Initiate the array.

**Part 7:**

**for** (**int** k = 1; k < p\_test + 1; k++) {

**for** (**int** j = 1; j < m + 1; j++) {

y\_in = 0;

**for** (**int** i = 1; i < n + 1; i++) {

**int** xi = S\_new[k][i];

y\_in = y\_in + xi \* w[i][j];

}

y[k][j] = *activationFunction*(y\_in);

}

}

Insert value to the two-dimensional arrays by using 3 for loops. Because the Y[][] depends on two arrays, we need use 3 for loops.

The *activationFunction* method looks like this:

**static** **int** activationFunction(**int** f) {

**int** fy = -1;

**if** (f >= 0)

fy = 1;

**return** fy;

}

**Part 8:**

String[] combine = **new** String[p\_test + 1];

**for** (**int** j = 0; j < p\_test + 1; j++) {

combine[j] = "";

}

**for** (**int** k = 1; k < p\_test + 1; k++) {

**for** (**int** j = 1; j < m + 1; j++) {

combine[k] = combine[k] + y[k][j] + " ";

}

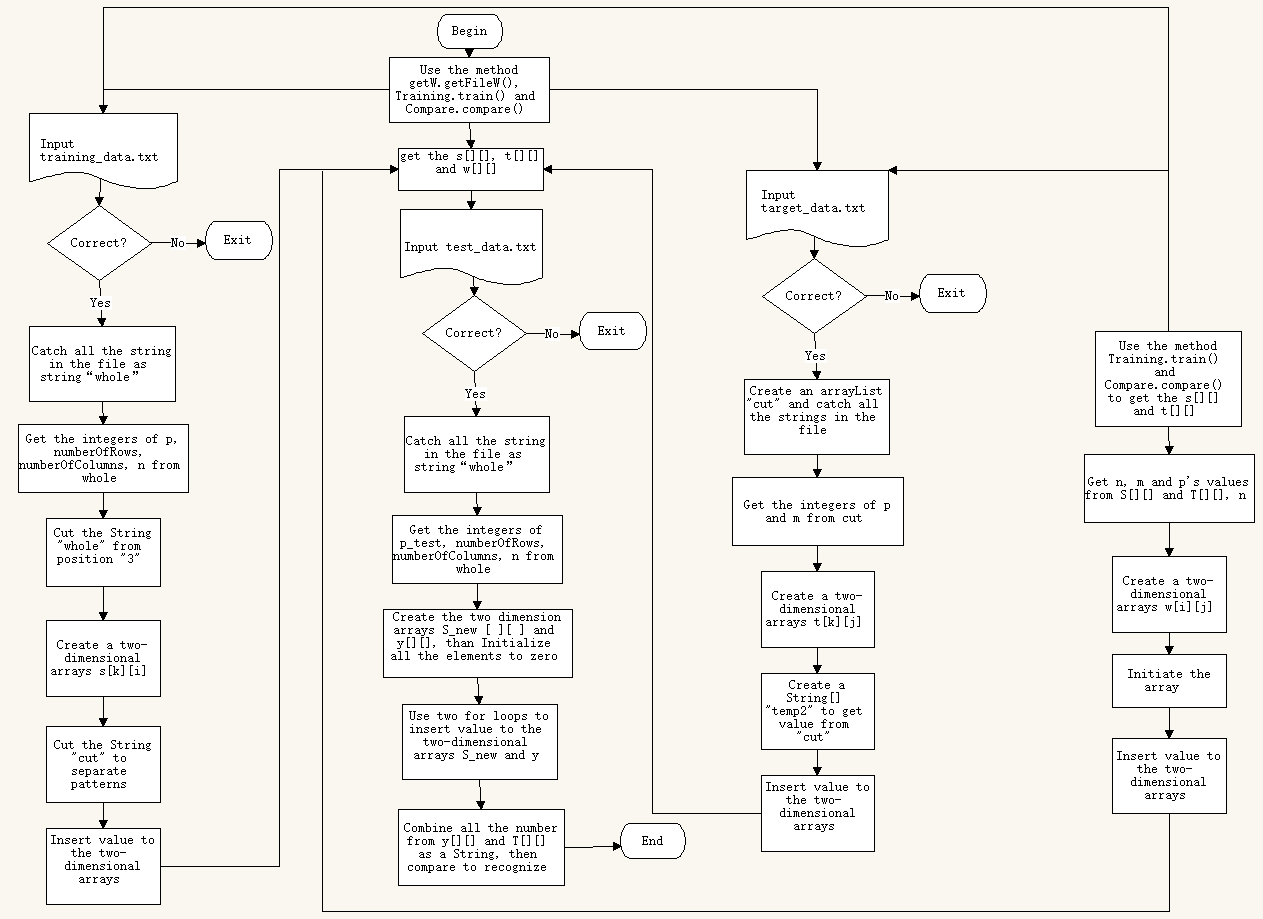
}

**int** p = S.length - 1;

String[] combine2 = **new** String[p + 1];

Combine all the number from y[][] and T[][] as a String, then compare to recognize.

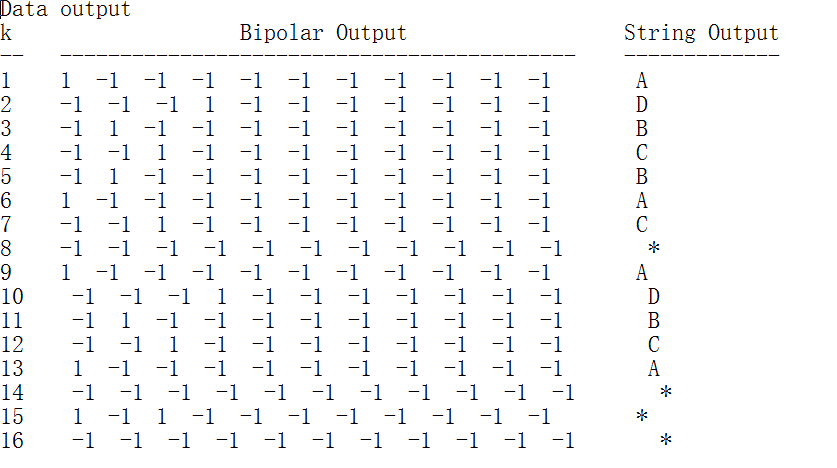
1. Flowchart



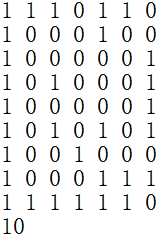
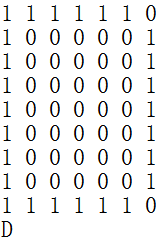
1. Program Test
   1. Test Example

In this part, I will split the test in 3 different situation: 1.Pattrens=4 in the train data. 2. Pattrens=5 in the train data. 3. Pattrens=52 in the train data.

1. When patterns=4 in the train data:

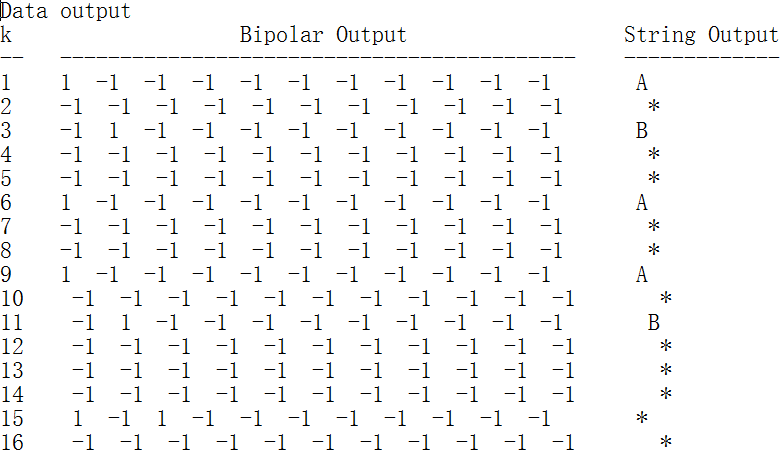


The program runs so good! For example, I choose the No. 10 of the test data. The program told me it’s a D.

On the left pattern is catch from test data, and the right pattern is catch from training data.

1. When patterns=5 in the train data:



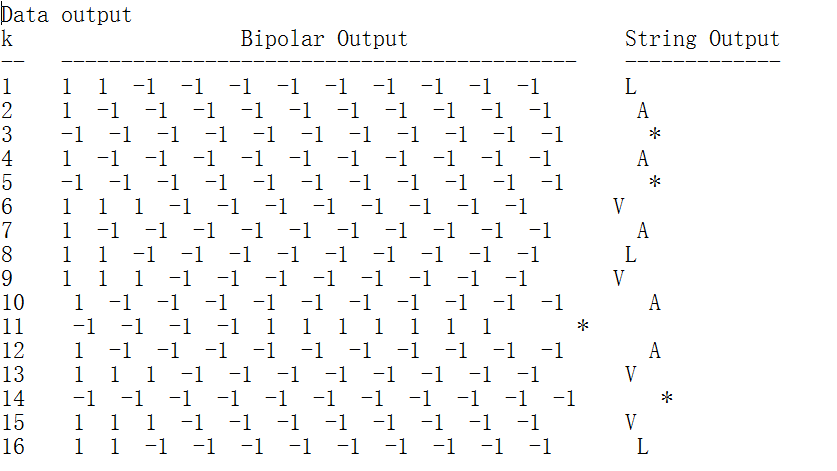
There is something wrong. The letters “C” and “D” can’t be recognized. However, all the letters left are right. For example, I choose the No. 11 letter. The program told that is a B.

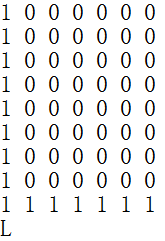
On the left pattern is catch from test data, and the right pattern is catch from training data.

That’s right.

1. When patterns=52 in the train data



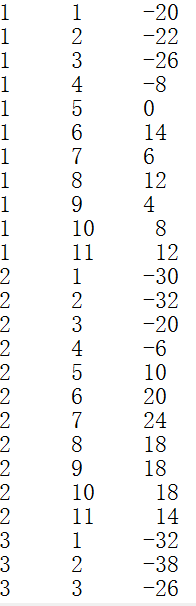
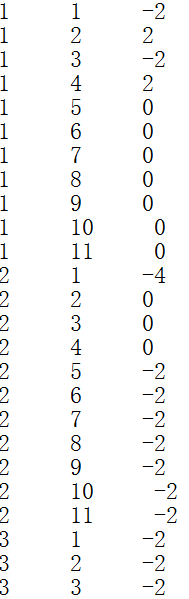
It fail. No doubt the first one must be A, but here, the program told it’s a L.

It’s totally different.

* 1. How to complete the program

After 4.1 test example, we know that the program has its limitation. You will see different weight between the situation patterns=4 and the situation patterns=52:

Patterns=52 Patterns=4

The number when the patterns=52 is too big. However the size of every pattern is too small: it is just 9\*7. If I want to complete the program and make it learn patterns more accurate, I need to increase the size of the pattern such as 1000\*1000. The bigger the size of pattern has, the more accurate the learning will be.

1. Conclusion

After all the work for this program, I earn much more experience than other individual code writing. The new concept of the neural network is lovely and interesting.

I try to complete all the work and make the program recognize all the letters. However, the size of the patterns influence the accuracy of program’s learning. Small size will limit the neural networks. Maybe in the future, I could develop a tool that can transform the picture or magazine to the matrix of 0 and 1. It will be easy to create the training data.

The solutions are always more than problems. That is what I learned from this project.

1. **Reference**

[1] Oja, E. (1992). Principal components, minor components, and linear neural networks. Neural networks, 5(6), 927-935.

[2] Sanger, T. D. (1989b). An optimality principle for unsupervised learning. In D. Touretzky (Ed.), Advances in neural information processing systern, ~. San Marco, CA: Morgan Kaufmann.