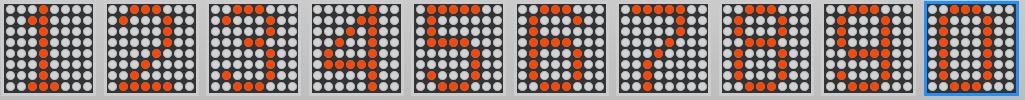
**Alphanumeric Recognition using Perceptron (Single Layer Neural Network)**

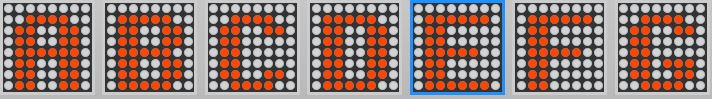
**Name : Ivan (이반)**

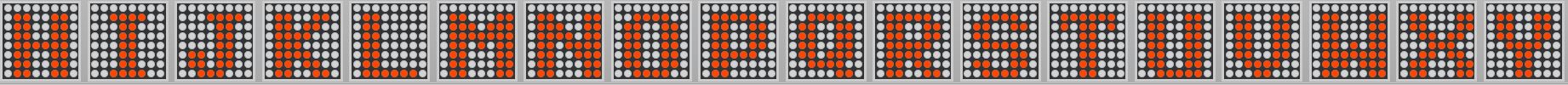
**학번 : 2014758139**

**AlphaNumeric Input Pattern**

AlphaNumeric consists 10 numbers (0-9) and 26 alphabets (A-Z). We are using 8x8 matrix to create the alphanumeric in binary mode. So, we will have 64 inputs (8x8) in total + 1 Bias (the value of bias is 1). We will have 36 results (0-9, A-Z) which means we will train the neural network to do 36 patterns repetition until we find the result we want.









I use these patterns as a guide to create the binary file. Red dot represents ‘1’ value and the white dots represents ‘-1’ value. Text below is the binary number for each pattern.

-1 -1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 -1 1 1 1 -1 -1 -1 1

-1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 1 1 -1 -1 -1 1

-1 -1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 -1 -1 1

-1 -1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 -1 1 1 1 -1 -1 -1 1

-1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 -1 1 -1 -1 -1 -1 1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 1

-1 1 1 1 1 1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 -1 1 1 1 -1 -1 -1 1

-1 -1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 -1 1 1 1 -1 -1 -1 1

-1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 1

-1 -1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 -1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 -1 1 1 1 -1 -1 -1 1

-1 -1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 -1 -1 -1 1 1 1 -1 -1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 1 1 1 -1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 -1 1 1 1 1 -1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 1 1 1 -1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 -1 1 1 1 1 -1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 1 1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 -1 1 1 1 -1 -1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 1 1 1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 1 -1 -1 1 1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 1 1 -1 1 1 1 -1 1 1 1 -1 1 1 1 1 1 1 1 -1 1 1 -1 1 -1 1 1 -1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 1 1 -1 1 1 1 -1 -1 1 1 -1 1 1 1 1 -1 1 1 -1 1 1 -1 1 1 1 1 -1 1 1 -1 -1 1 1 1 -1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 -1 1 1 1 1 -1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 1 1 1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 1 -1 -1 1 1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 -1 1 1 1 1 -1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 1 -1 1 1 -1 1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 -1 1 1 1 1 1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 1 -1 1 1 -1 1 -1 1 1 -1 1 1 1 1 1 1 1 -1 1 1 1 -1 1 1 1 -1 1 1 -1 -1 -1 1 1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 1 -1 -1 1 1 -1 1 1 -1 -1 -1 -1 1 1 1 -1 -1 -1 -1 1 1 -1 1 1 -1 -1 1 1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 1

-1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 1 -1 1

999

The first 10 pattern is numeric (0-9) and the rest of them is alphabet (A-Z). The last digit of each pattern is set to 1 (destination result value). At the final row 999 means the end of the file. I save this file as alphanumeric.txt which will be called by function in C program later.

**C program**

This C project includes 3 files, main.c, function.h (header file for all function declaration), and function.c. All explanation is written on the source code as a comment.

**Main.c**

#include <stdio.h>

#include <stdlib.h>

/\* for rand() \*/

#include <time.h>

#include <math.h>

#include <unistd.h>

#include <stdbool.h>

/\* for type bool \*/

#include "functions.h"

/\* all function prototypes are here \*/

#define THRESHOLD 0

#define DATA\_FILE "alphanumeric.txt"

#define LEARNING\_RATE 1

int main(void)

{

int i;

/\* open alphanumeric.txt for reading \*/

FILE \* file\_pointer = fopen(DATA\_FILE, "r");

/\* make sure the file can be opened \*/

if (file\_pointer == NULL)

{

fprintf(stderr, "Cannot open training data file.\n");

fprintf(stderr, "Check permissions for data file.\n");

exit(1);

}

float input[65];

/\* the inputs for the artifical neural network \*/

input[64]=1;

/\*bias value=1\*/

float weight[65];

/\* the weights for the artifical neural network, 64 for input + 1 for bias \*/

float threshold;

/\* used in Activation Function \*/

/\* if summation of weighted inputs > threshold,

\* Activation Function returns true. Otherwise,

\* Activation Function returns false. \*/

/\* get learning rate from macro \*/

float learning\_rate = LEARNING\_RATE;

float dot\_product[36];

/\* dot product = (a1 \* b1) + (a2 \* b2) + ... + (an \* bn) \*/

/\* This will be the summation of all the weighted inputs.

\* This value will be given to the activation function. \*/

/\* What actual\_output is the object being classified in? (Y) \*/

int actual\_output = 0;

/\* error = expected output - actual ouput (Y-d) \*/

/\* error is used in the update weight formula \*/

float error = 0;

/\* keep track of if there are any incorrect

\* classifications left by using a boolean value.

\* true means there are still incorrect classifications

\* false means all classifications are correct. \*/

bool incorrectClassifications = true;

printf("PERCEPTRON TRAINING ALGORITHM IMPLEMENTATION for ALPHANUMERIC\n");

/\* We need to seed the random number generator.

\* Otherwise, it will produce the same number every time

\* the program is run. \*/

srand(time(NULL));

/\* I am using the current time to seed rand(). \*/

/\* get threshold from macro \*/

threshold = THRESHOLD;

/\* The weights will start off as random numbers in

\* the range [-1, 1]. \*/

for(i=0;i<65;i++){

weight[i] = ((float)rand())/RAND\_MAX\*2-1;

}

while (incorrectClassifications == true)

{

incorrectClassifications = false;

/\* Let's loop through all the data sets. \*/

int i = 0; //numeric counter

char alphabet='A'; //alphabet counter

int x; //input counter

int p=0; //pattern counter

/\* loop will break if input = 999 \*/

while (1)

{

/\* get input from the data set file \*/

for(x=0;x<64;x++){

input[x] = getInput(file\_pointer);

}

if (input[0] == 999){

break;

}

/\* sum the weighted inputs \*/

dot\_product[p] = sumWeightedInputs(input, weight);

/\* apply activation function to sum of weighted inputs \*/

actual\_output = activationFunction(dot\_product, threshold);

/\* print which data set we are on \*/

if(i<=9){

printf("Data Set %d\n", i); //print data Numeric (0-9)

}

if(i>9){

printf("Data Set %c\n", alphabet); //print data alphabet (A-Z)

alphabet++;

}

int z;

/\* print the inputs including bias \*/

for(z=0;z<65;z++){

printf("\n"); // new line

printf("Input [%d] = %.2f ", z+1,input[z]);

/\* print the weights \*/

printf("\n"); // new line

printf("Weight [%d] = %.2f ", z+1,weight[z]);

}

/\* print the summation of weighted inputs \*/

printf("\n"); // new line

printf("Summation of pattern number %d = %.2f\n",p+1, dot\_product[p]);

/\* print the actual\_output \*/

printf("Object classified to class %d.\n", actual\_output);

/\* check the output \*/

error = checkOutput(file\_pointer, actual\_output);

/\* print the result \*/

if (error == 0) printf("Ouput correct.\n");

else

{

/\* set incorrectClassifications to true

\* to loop through the data set once more \*/

incorrectClassifications = true;

printf("Output incorrect.\n");

printf("Error = %.0f\n", error);

/\* we need to update the weights if \*

\* there is an error \*/

int c;

for(c=0;c<65;c++){

weight[c] = updateWeights(weight[c], learning\_rate, input[c], error);

}

/\* print the new weights \*/

printf("\n"); // new line;

printf("NEW WEIGHTS: \n");

for(c=0;c<65;c++){

printf("\*\*\* New weight [%d]: %.2f ",c, weight[c]);

}

} // ends else

printf("\n");

printf("-----------------------------------------------\n");

printf("-----------------------------------------------\n\n");

i++; //increment i

p++; //increment pattern counter

} // ends while (1)

/\* set the file pointer back to beginning of file \*/

rewind(file\_pointer);

} // ends while(incorrectClassifications == true)

/\* Print the final weights \*/

printf("\n");

printf("Final Weights: \n");

int m;

for(m=0;m<65;m++){

printf("Weight [%d]: %.2f\n",m+1, weight[m]);

}

float newinput[65]; //input any pattern for pattern recognition

newinput[64]=1; //bias automatically set to 1

float dot\_product\_input; //new summation of inputted pattern

printf("Please input 64 bits of alphanumeric (8x8 matrix 1 & -1) : ");

for(m=0;m<64;m++){

scanf("%f",&newinput[m]); //input bit of matrix

}

dot\_product\_input=sumWeightedInputs(newinput, weight); //sum the weighted inputs (input\*final weight)

printf("Summation of Weighted new Input = %.2f\n",dot\_product\_input); //print the summation

int p,location;

float diff[36]; //summation value difference

for(p=0;p<36;p++){

diff[p]=fabs(dot\_product[p]-dot\_product\_input); //difference between 36 pattern's dot product - new inputted dot product

}

float minimum=diff[0]; //set the minimum value, which means the closest value, to find the closest pattern

for(p=0;p<36;p++){

if(diff[p]<minimum){

minimum=diff[p];

location=p; //location of the closest value

}

}

printf("The MINIMUM DIFFERENCE of inputted pattern and alphanumeric pattern is= %.2f\n",minimum);

printf("The LOCATION of minimum difference of inputted pattern and alphanumeric pattern is at pattern number %d\n",location+1);

char alphabet='A'; //alphabet counter

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

if(location<=9){

printf("PATTERN RECOGNITION RESULT = %d\n",location); //print the final result which is numeric

}

if(location>9){

alphabet=alphabet+(location-9);

printf("PATTERN RECOGNITION RESULT = %c\n",alphabet); //print the final result which is alphabet

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

/\* close the input file \*/

fclose(file\_pointer);

getchar();

return 0;

} // ends main()

**Functions.h**

#include <stdio.h>

/\* for FILE type \*/

/\* get inputs for the perceptron from a data file \*/

float getInput(FILE \* ftp);

/\* A function for the dot product (summation) of weighted

\* inputs (i.e., (input1 \* weight1) + (input2 \* weight2) ... \*/

float sumWeightedInputs(float input[], float weight[]);

/\* Activation Function -- sees if the weights is greater

\* than a threshold, and returns 1, returns 0 if the same as threshold

\* and -1 otherwise \*/

int activationFunction(float dot\_product[], float threshold);

/\* Function checks the actual output of the perceptron's ouput

\* against the training data set. \*/

float checkOutput(FILE \* ftp, float actual\_output);

/\* a function to update weights \*/

float updateWeights(float weight, float learning\_rate, float input, float error);

**Functions.c**

#include "functions.h"

#include <stdio.h>

#include <stdlib.h>

/\* gets a input for the perceptron from a data file \*/

float getInput(FILE \* ftp)

{

float input;

/\* scan the input from the training data \*/

fscanf(ftp, "%f ", &input);

/\* Return the input \*/

return input;

}

/\* A function for the dot product (summation) of weighted

\* inputs (i.e., (input1 \* weight1) + (input2 \* weight2) ... \*/

float sumWeightedInputs(float input[], float weight[])

{

/\* sum means dot product here \*/

float sum = 0;

int n;

/\* figure out the dot product here \*/

for(n=0;n<64;n++){

sum =sum+(input[n] \* weight[n]);

}

/\* return sum \*/

return sum;

}

/\* This is the function that updates the weights

\* if the neuron misclassified input. \*/

/\* I am using this formual to update weights:

\* new weight = old weight + (learning rate \* current input \* (error)

\* where error = expected output - actual output \*/

float updateWeights(float weight, float learning\_rate, float input, float error)

{

float new\_weight = 0;

/\* use the formula Wn = Wn-1 + alpha(d-y)X \*/

new\_weight = weight + (learning\_rate \* error \* input);

return new\_weight;

}

/\* Activation Function -- sees if the weights is greater

\* than a threshold, and returns 1, returns 0 if same as threshold,

\* and -1 otherwise. \*/

int activationFunction(float dot\_product, float threshold)

{

if (dot\_product > threshold) return 1;

/\* object actual\_outputified to class 1 \*/

else if(dot\_product < threshold) return -1;

/\* object actual\_outputified to class -1 \*/

else if(dot\_product = threshold) return 0;

/\* object actual\_outputified to class 0 \*/

}

/\* Function checks the actual output of the perceptron's ouput against the expected output. \*/

float checkOutput(FILE \* ftp, float actual\_output)

{

float expected\_output = 0;

/\* error = expected\_ouput - actual\_output \*/

float error = 0;

/\* the value of error is needed in the update

\* weight formula \*/

/\* get expected output from data file \*/

fscanf(ftp, "%f ", &expected\_output);

printf("\n"); // new line

printf("Expected Output: %.2f\n", expected\_output);

printf("Actual Output: %.2f\n", actual\_output);

/\* calculate error \*/

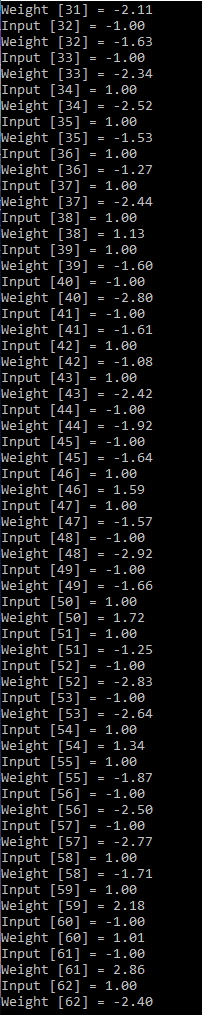
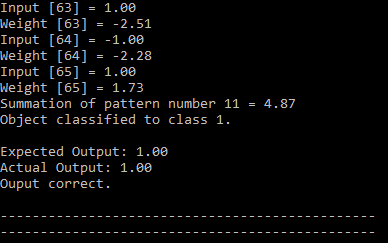
error = expected\_output - actual\_output;

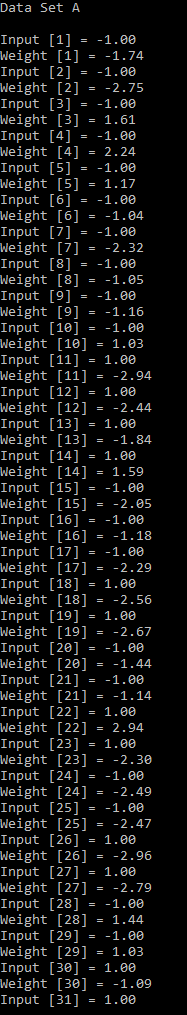
return error;

}

**Result**

The result will print every set Pattern with 64 input values + bias, and 65 RANDOM weight values + weight of bias. Input[65] and weight[65] are for bias.

**If the result of the pattern is TRUE**



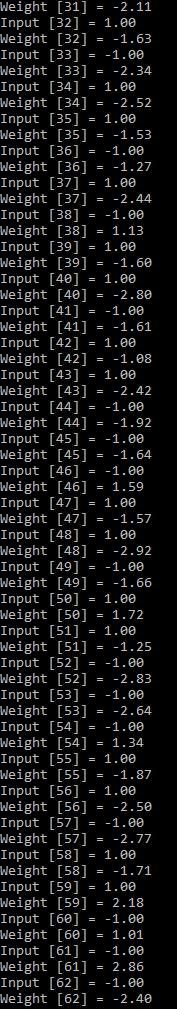
For data set A, we apply summation of weighted input function (input \* weight), then we threshold the result of dot product,

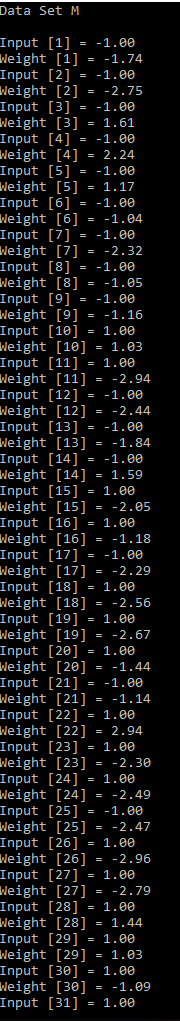
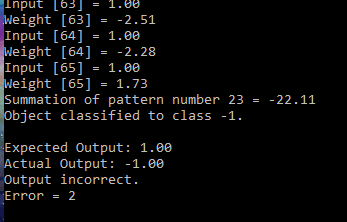
if dot\_product>0, actual output = 1

if = 0, actual output = 0

if <0, actual output = -1

If there is no error, as we can see the expected output (destination output) is the same as actual output (Y), Output is Correct!

**If the result of the pattern is FALSE**



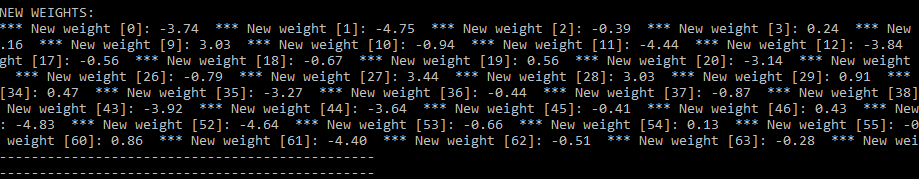
For data set alphabet M, as we can see the expected output (destination output) is not the same as actual output (Y). With the error of 2 (D-Y), the output is incorrect!

After that the system will perform an WEIGHT UPDATE operation, resulting a new 65 weight values.

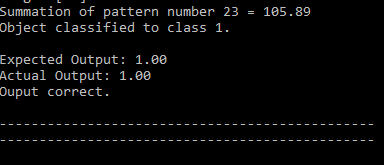
**Perceptron-Training (Update the Weight)**

For updating the weight value, we apply the updateWeights function which take the formula : New weight = weight+(learning rate\*error\*input) ; the error value is expected output – actual output (D – Y).

This is just a small piece of update weight update operation. I cannot put them all here because it’s too long. You can see the result clearly on the exe file.

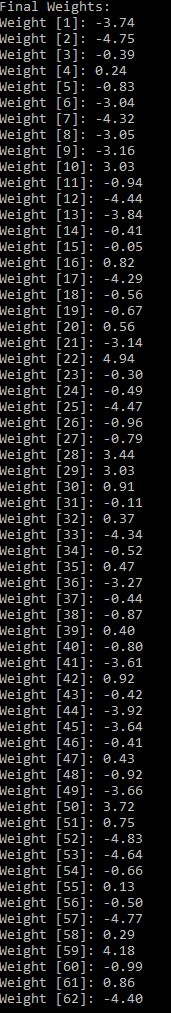


After updating the weight, the training keeps looping. In the next loop or repetition the system will print the training set of data M again. Now the output becomes correct.



**Final Weight**

After all error weight values have been updated and there is no more error, finally the training is finished and we get the final weight values of each input.



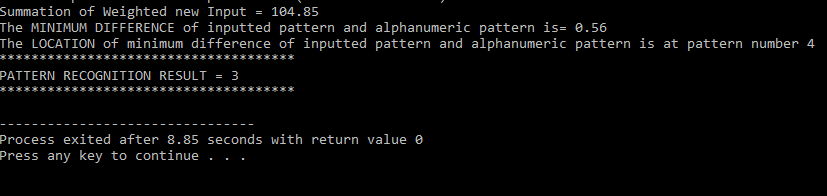
**Input New Pattern to be Recognized**

After the training of the perceptron (neural network) is finished, now we need to input 64 bits of 8x8 matrix. We can input anything we want as long as it consists 1 and -1. For example, I take the number 4 pattern from alphanumeric.txt.





But at the final bit of input I change from -1 to 1 (as you can see at picture above). Originally, the pattern number 4 (4th row) in alphanumeric.txt results a “3” numeric pattern. After I change 1 bit at the end, we want to see if the neural network could guess the CLOSEST pattern possible to the inputted pattern. In this case we hope the program will result a “3” numeric pattern as well even though we changed the last bit.



First, we sum the weighted input (input \* final weights) then we look for the minimum difference of inputted pattern and alphanumeric pattern value. Then after we got the minimum difference value, we know the position of the minimum value which is pattern number 4. Pattern number 4 in our training data is “3” numeric pattern. So, the neural network is successfully trained and could recognize the new inputted random pattern.

Another example of final result of pattern T recognition without changing any bit, so the difference value is 0.

