Neural Network I: Fundamental Theory and Applications (CSA01)

Team Project III

Image restoration based on associate memory

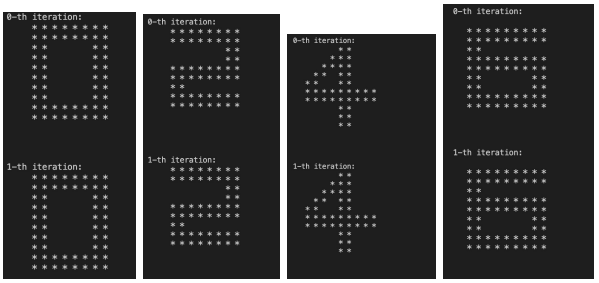
m5261113 Akihiro Hattori
m5261159 Taisei Shibakura
m5261162 Taisei Shiraishi
m5261176 Yuta Uchida

a)
We checked how the results changed when the noise was changed to 0%, 10%, and 15%.

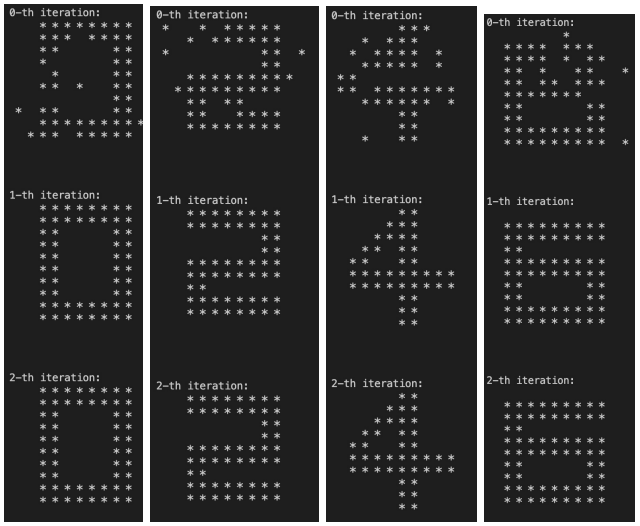
b)
We changed noise rates.

```
#define noise_rate 0.1
```

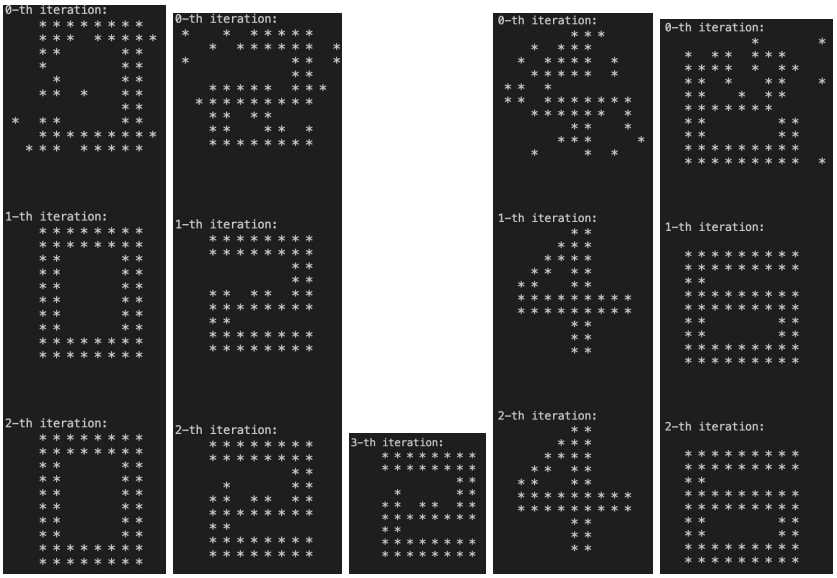
0%



10%

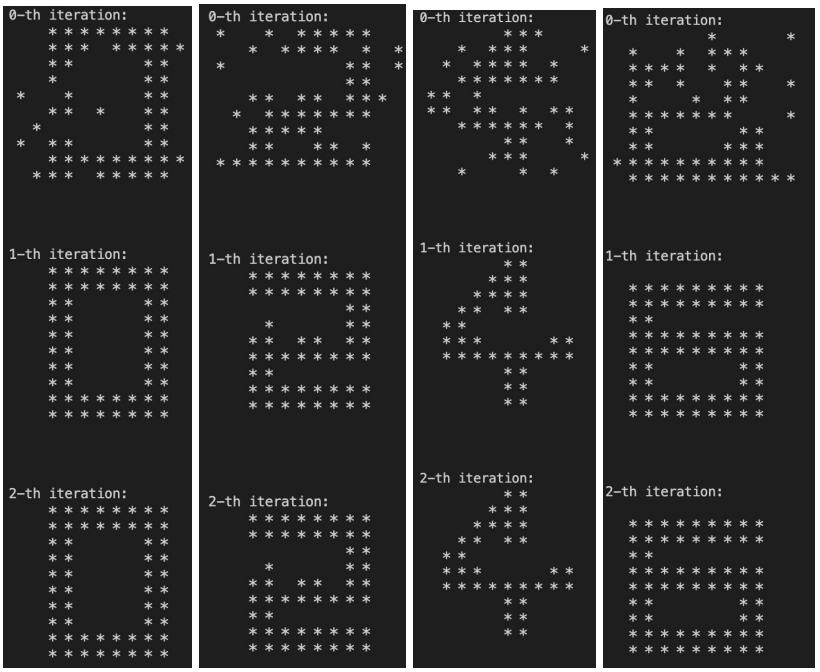


15%

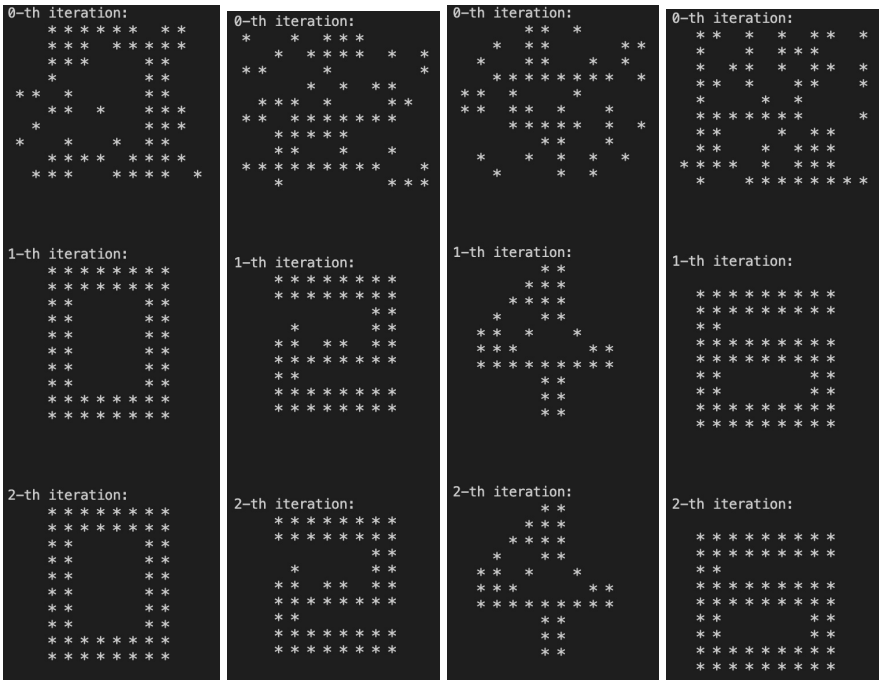


c) It was found that 15% took more time to repair than 10%.
At 15%, it could not be repaired correctly. (2 is not correct)

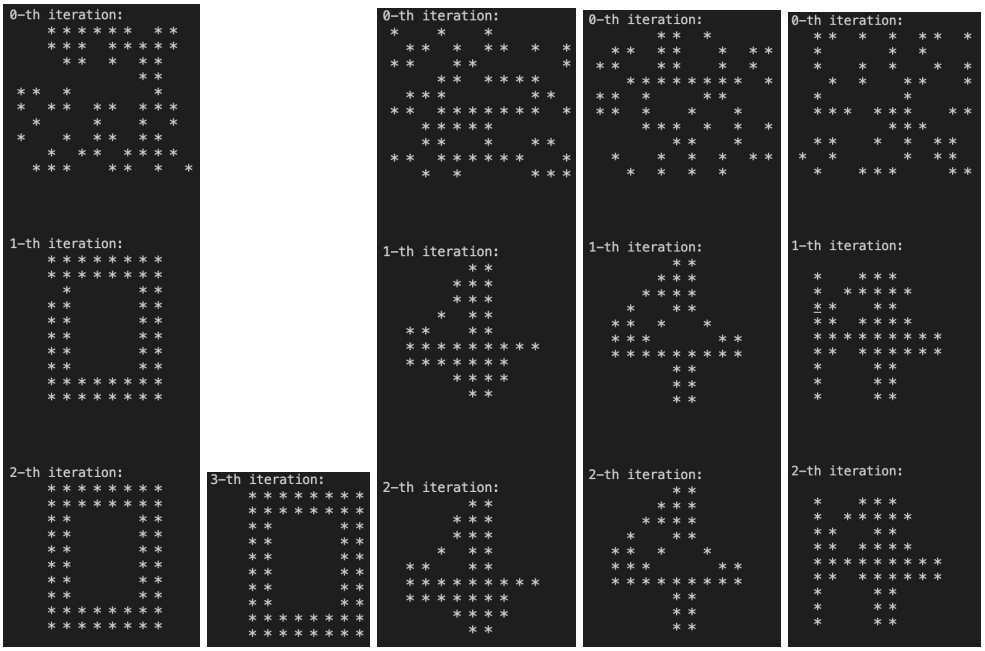
d) At what % will it become irreparable?
e) We increased noise. (noise_rate changed 20, 30, 40)
f) Result outputs
20%



30%



40%



From the results, if we changed 40%, HNN may not change the correct result.

Source code----------

```
/*********************************************************************/
/* c-Program for constructing an autoassociative memory              */
/* The model is the Hopfield neural network with asynchorous update  */
/* If you want to use it for applications other than that described  */
/* in this project, please provide the patterns to be stored         */
/*                                                                   */
/* This program is produced by Qiangfu Zhao.                         */
/* You are free to use it for educational purpose                    */
/*********************************************************************/
#include <stdlib.h>
#include <stdio.h>

#define n_neuron 120
#define n_pattern 4
#define n_row 10
#define n_column 12
#define noise_rate 0.15

int pattern[n_pattern][n_neuron] = {
   {1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1,
    1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1,
    1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1,
    1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1,
    1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1,
    1, 1, -1, -1, 1, 1, 1, 1, -1, -1, 1, 1,
    1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1},
   {1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, -1, -1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, -1, -1, 1, 1,
    1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, 1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
   {1, 1, 1, 1, 1, -1, -1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, -1, -1, -1, 1, 1, 1, 1, 1,
    1, 1, 1, -1, -1, -1, -1, 1, 1, 1, 1, 1,
```

```
    1, 1, -1, -1, 1, -1, -1, 1, 1, 1, 1, 1,
    1, -1, -1, 1, 1, -1, -1, 1, 1, 1, 1, 1,
    1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, 1, 1, 1, 1, -1, -1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, -1, -1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, -1, -1, 1, 1, 1, 1, 1},
   {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, -1, -1, 1, 1, 1, 1, 1, -1, -1, 1, 1,
    1, -1, -1, 1, 1, 1, 1, 1, -1, -1, 1, 1,
    1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1,
    1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1}};

int w[n_neuron][n_neuron];
int v[n_neuron];
/*************************************************************************/
/* Output the patterns, to confirm they are the desired ones             */
/*************************************************************************/
Output_Pattern(int k)
{
 int i, j;

 printf("Pattern[%d]:\n", k);
 for (i = 0; i < n_row; i++)
 {
   for (j = 0; j < n_column; j++)
     printf("%2c", (pattern[k][i * n_column + j] == -1) ? '*' : ' ');
   printf("\n");
 }
 printf("\n\n\n");
 getchar();
}

/*************************************************************************/
/* Output the state of the network in the form of a picture              */
/*************************************************************************/
Output_State(int k)
{
```

```
 int i, j;

 printf("%d-th iteration:\n", k);
 for (i = 0; i < n_row; i++)
 {
   for (j = 0; j < n_column; j++)
     printf("%2c", (v[i * n_column + j] == -1) ? '*' : ' ');
   printf("\n");
 }
 printf("\n\n\n");
 getchar();
}

/*********************************************************************/
/* Store the patterns into the network                               */
/*********************************************************************/
Store_Pattern()
{
 int i, j, k;

 for (i = 0; i < n_neuron; i++)
 {
   for (j = 0; j < n_neuron; j++)
   {
     w[i][j] = 0;
     for (k = 0; k < n_pattern; k++)
       w[i][j] += pattern[k][i] * pattern[k][j];
     w[i][j] /= (double)n_pattern;
   }
   w[i][i] = 0;
 }
}

/*********************************************************************/
/* Recall the m-th pattern from the network                          */
/* The pattern is corrupted by some noises                           */
/*********************************************************************/
Recall_Pattern(int m)
{
 int i, j, k;
 int n_update;
 int net, vnew;
 double r;
```

```
 for (i = 0; i < n_neuron; i++)
 {
   r = (double)(rand() % 10001) / 10000.0;
   if (r < noise_rate)
     v[i] = (pattern[m][i] == 1) ? -1 : 1;
   else
     v[i] = pattern[m][i];
 }
 Output_State(0); /* show the noisy input pattern */

 k = 1;
 do
 {
   n_update = 0;
   for (i = 0; i < n_neuron; i++)
   {
     net = 0;
     for (j = 0; j < n_neuron; j++)
       net += w[i][j] * v[j];
     if (net >= 0)
       vnew = 1;
     if (net < 0)
       vnew = -1;
     if (vnew != v[i])
     {
       n_update++;
       v[i] = vnew;
     }
   }
   Output_State(k); /* show the current result */
   k++;
 } while (n_update != 0);
}

/*************************************************************************/
/* Initialize the weights                                                */
/*************************************************************************/
Initialization()
{
 int i, j;

 for (i = 0; i < n_neuron; i++)
```

```
   for (j = 0; j < n_neuron; j++)
     w[i][j] = 0;
}

/*************************************************************************/
/* The main program                                                      */
/*************************************************************************/
main()
{
 int k;

 for (k = 0; k < n_pattern; k++)
   Output_Pattern(k);

 Initialization();
 Store_Pattern();
 for (k = 0; k < n_pattern; k++)
   Recall_Pattern(k);
}
```