Project3

1.Team member

- **m5268101 Liu Jiahe**
- **m5251140 Ken Sato**
- **m5271051 Keisuke Utsumi**

2.Team Project III

- Team Project III (cont.)
  - Write a computer program to implement the algorithm.
  - Try to recall the patterns with the noise level being 0%, 10% or 15%.
  - We say a pixel is a noise if its value is changed from 1 (or 0) to 0 (or 1).

3.数学原理

Hopfield神经网络的使用主要包括两个方面：生成权重矩阵和对指定模式进行Recall

1.生成权重矩阵

$$w_{ij} = \frac{1}{P}\sum_{k=1}^{P} x_i^{(k)} x_j^{(k)} \quad \text{for} \quad i \neq j, x_{ii} = 0 \tag{1}$$

其中，w_{ij} 是权重矩阵的第 i 行第 j 列的元素，N 是神经元的数量(即模式中元素的数量)，P 是模式的数量，$x_i^{(k)}$ 是第 k 个模式的第 i 个神经元的值。

写成矩阵的形式：

$$W_{N\times N} = \frac{1}{P}\sum_{k=1}^{P} X^{(k)T} X^k \quad \text{for} \quad W_{i,i} = 0 \tag{2}$$

其中X^k表示第k个模式，它是一个含有N个神经元的行向量

```
for key in original_patterns:
    pattern = original_patterns[key]
    w += np.outer(pattern, pattern)
w /= n_pattern
np.fill_diagonal(w, 0)
```

2. Recall

Hopfield 网络的 recall 过程可以表示为：

$$x_i(t+1) = \text{sgn}\left(\sum_{j=1}^{N} w_{ij} x_j(t)\right) \tag{3}$$

其中，$x_i(t)$ 是第 i 个神经元在时间步 t 的值，sgn 是符号函数（$sgn(x) = 1 \;\; x \geq 0; sgn(x) = -1 \;\; x < 0$），$w_{ij}$ 是权重矩阵的第 i 行第 j 列的元素。

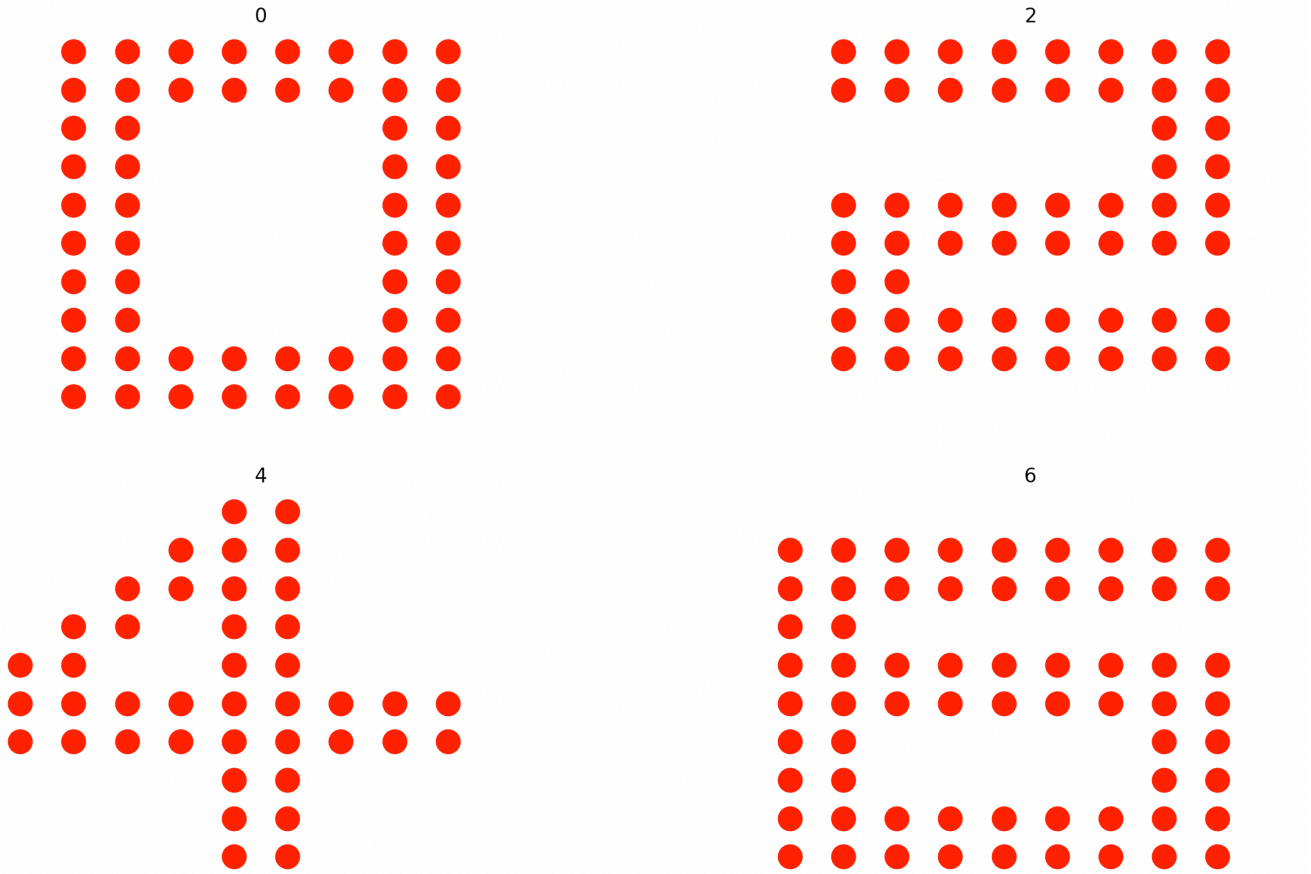
有异步更新和同步更新两种方式，异步更新一次更新一个神经元，同步更新一次更新所有神经元，全部神经元更新完一次为一轮。当下一轮神经元相较于上一轮神经元没有变化时，迭代停止。

```
while True:

        if recall_method == synchronous_update:
            new_pattern = np.where(w @ noisy_pattern >= 0, 1, -1)

            if np.array_equal(new_pattern, noisy_pattern):
                break

            noisy_pattern = new_pattern

        if recall_method == asynchronous_update:

            prev_pattern = noisy_pattern.copy()

            for i in range(n_neuron):
                net_input = w[i] @ noisy_pattern

                noisy_pattern[i] = 1 if net_input >= 0 else -1

            if np.array_equal(noisy_pattern, prev_pattern):
                break
```

4.代码实现

已有一个原始的模式pattern，通过该模式生成权重矩阵，然后对该模式进行加噪处理，而后利用权重矩阵对噪声pattern进行recall，尝试还原为原始pattern

1.原始pattern图像



2.代码：

C

project3.c

```c
/*********************************************************************/
/*We refer to the code for class. */
/*********************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#define n_neuron   120 //12*10
#define n_pattern  4 //number of pattern (0,2,4,6)
#define n_row      10
#define n_column   12
#define noise_rate 0.15 //{0% 10% 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];//120*120
int v[n_neuron]; //vector 120
/*****************************************************************/
/* Output the patterns, to confirm they are the desired ones     */
/*****************************************************************/
void Output_Pattern(int k) {
  FILE *outputfile;
  int i,j;

  outputfile = fopen("./outpat3.txt", "a");
  fprintf(outputfile,"Pattern[%d]:\n",k);
```

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

/*********************************************************************/
/* Output the state of the network in the form of a picture          */
/*********************************************************************/
void Output_State(int k) {
  int i,j;
  FILE *outputfile;

  outputfile = fopen("./outsta3.txt", "a");

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

/*********************************************************************/
/* Store the patterns into the network                               */
/*********************************************************************/
void Store_Pattern() {
  //Phase1: Storage
  int i,j,k;
  for(i=0; i<n_neuron; i++){
    for(j=0; j<n_neuron; j++){
      w[i][j] = 0; //Initialization W =0

      for(k=0;k<n_pattern;k++)
        w[i][j] += pattern[k][i]*pattern[k][j];//W=W+s(m)*s(m)^T
     w[i][j] /= (double)n_pattern; //nomalazation
   }
      w[i][i]=0;//w_ii=0
  }
}

/*********************************************************************/
/* Recall the m-th pattern from the network                          */
/* The pattern is corrupted by some noises                           */
/*********************************************************************/
```

```
void Recall_Pattern(int m){
  //Phase 2: Recall
  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;//r:random
    if(r<noise_rate)
      v[i]=(pattern[m][i]==1)?-1:1;//corrupted by noises
    else
      v[i]=pattern[m][i]; //v<-pattern
  }
  Output_State(0);         /* show the noisy input pattern */

  k=1;
  do {//recall
    n_update=0;//
    for(i=0; i<n_neuron; i++){
      net=0;
      for(j=0; j<n_neuron; j++)//Find the new output
        net+=w[i][j]*v[j];//Σ(w*v)
      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); //Stop if there is no change after many iterations
}

/*************************************************************************/
/* Initialize the weights                                                */
/*************************************************************************/
void 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                                                      */
/*************************************************************************/
int 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);
}
```

Python

project3.py:

```python
import json
import numpy as np
import matplotlib.pyplot as plt
np.set_printoptions(threshold=np.inf)

# Load patterns and noisy patterns

noisy_level = 15 # 0, 10, 15, 20, 25, 40, 50

noisy_file = "./pattern_"+str(noisy_level)+".json"

with open("./pattern.json", "r") as f:
    original_patterns = json.load(f)

with open(noisy_file, "r") as f:
    noisy_patterns = json.load(f)

n_neuron = 120
n_pattern = 4

# Convert patterns to numpy arrays
for key in original_patterns:
    original_patterns[key] = np.array(original_patterns[key])

for key in noisy_patterns:
    noisy_patterns[key] = np.array(noisy_patterns[key])

# Initialize weight matrix
w = np.zeros((n_neuron, n_neuron))


# Store patterns in the weight matrix
for key in original_patterns:
    pattern = original_patterns[key]
    w += np.outer(pattern, pattern)
w /= n_pattern
np.fill_diagonal(w, 0)
# print(w)
```

```
39
40
41  # for key in original_patterns:
42  #     pattern = original_patterns[key]
43  #     print(key,pattern)
44  #     for i in range(n_neuron):
45  #         for j in range(n_neuron):
46  #             w[i, j] += pattern[i] * pattern[j]
47  #         # w[i, i] = 0
48
49  # w /= n_pattern
50  # np.fill_diagonal(w, 0)
51  # # print(w)
52
53
54  # Recall patterns and store in pattern_store
55  pattern_store = {}
56
57  synchronous_update = 1
58  asynchronous_update = 2
59  recall_method = 2
60
61  for key in noisy_patterns:
62      noisy_pattern = noisy_patterns[key]
63      original_pattern = original_patterns[key]
64      # print(original_pattern)
65      # print(noisy_pattern)
66      recalled_patterns = [original_pattern.tolist(), noisy_pattern.tolist()]
67
68      while True:
69
70          if recall_method == synchronous_update:
71              new_pattern = np.where(w @ noisy_pattern >= 0, 1, -1)
72
73              if np.array_equal(new_pattern, noisy_pattern):
74                  break
75
76              noisy_pattern = new_pattern
77
78          if recall_method == asynchronous_update:
79
80              prev_pattern = noisy_pattern.copy()
81
82              for i in range(n_neuron):
83                  net_input = w[i] @ noisy_pattern
84
85                  noisy_pattern[i] = 1 if net_input >= 0 else -1
86
87              if np.array_equal(noisy_pattern, prev_pattern):
88                  break
89
90          recalled_patterns.append(noisy_pattern.tolist())
```

```
    pattern_store[key] = recalled_patterns


def num_to_color(num):
    if num == 1:
        return "white"
    elif num == -1:
        return "red"


max_length = 0
for key, values in pattern_store.items():
    max_length = max(max_length, len(values))

fig, axs = plt.subplots(4, max_length, figsize=(20, 10),gridspec_kw=
{'wspace': 0.5, 'hspace': 0.5})
spacing = 0.2
row = 0
for key, values in pattern_store.items():

    count = 0
    for p in values:
        for i in range(10):
            for j in range(12):
                color = num_to_color(p[i * 12 + j])
                axs[row, count].scatter(j * spacing, -i * spacing, c=color,
s=50)
        axs[row, count].axis("off")
        axs[row, count].set_xlim(-0.5 * spacing, 11.5 * spacing)
        axs[row, count].set_ylim(-9.5 * spacing, 0.5 * spacing)
        if count == 0:
            axs[row, count].set_title(f"{key}-original")
        if count == 1:
            axs[row, count].set_title(f"noisy_level:{noisy_level}%")
        if count > 1:
            axs[row, count].set_title(f"{count-1}th-iter")
        count += 1
    row += 1

for i in range(row):
    for j in range(max_length):
         axs[i, j].axis("off")

plt.show()
```

5.结果讨论

C

[原始图像]():

```
Pattern[0]:
     * * * * * * * * *
     * * * * * * * * *
     * *           * *
     * *           * *
     * *           * *
     * *           * *
     * *           * *
     * * * * * * * * *
     * * * * * * * * *


Pattern[1]:
     * * * * * * * * *
     * * * * * * * * *
                 * *
                 * *
     * * * * * * * * *
     * * * * * * * * *
     * *
     * * * * * * * * *
     * * * * * * * * *


Pattern[2]:
           * *
         * * *
       * * * *
     * *   * *
   * *     * *
   * * * * * * * * *
   * * * * * * * * *
           * *
           * *
           * *


Pattern[3]:

   * * * * * * * * *
```

```
  * * * * * * * * * *
  * *
  * * * * * * * * * *
  * * * * * * * * * *
  * *             * *
  * *             * *
  * * * * * * * * * *
  * * * * * * * * * *
```

用c语言完成的代码比较好将加噪后的模式复原，接下来是复原15%噪声等级的效果

```
0-th iteration:
     * * * * * * * * *
     * * *   * * * * *
     * *           * *
     *             * *
       *           * *
     * *   *       * *
                   * *
 *   * *           * *
     * * * * * * * * * *
   * * *   * * * * *


1-th iteration:
     * * * * * * * * *
     * * * * * * * * *
     * *           * *
     * *           * *
     * *           * *
     * *           * *
     * *           * *
     * *           * *
     * * * * * * * * *
     * * * * * * * * *


2-th iteration:
     * * * * * * * * *
     * * * * * * * * *
     * *           * *
     * *           * *
     * *           * *
     * *           * *
     * *           * *
     * *           * *
     * * * * * * * * *
```

```
     * * * * * * * * *



0-th iteration:
 *     *   * * * * *
     *   * * * * * * *   *
 *               * *   *
                 * *
     * * * * *   * * *
   * * * * * * * * * *
     * *   * *
     * *     * *   *
     * * * * * * * * *



1-th iteration:
     * * * * * * * * *
     * * * * * * * * *
                 * *
                 * *
     * *   * *   * *
     * * * * * * * * *
     * *
     * * * * * * * * *
     * * * * * * * * *



2-th iteration:
     * * * * * * * * *
     * * * * * * * * *
                 * *
       *         * *
     * *   * *   * *
     * * * * * * * * *
     * *
     * * * * * * * * *
     * * * * * * * * *



3-th iteration:
     * * * * * * * * *
     * * * * * * * * *
                 * *
       *         * *
     * *   * *   * *
```

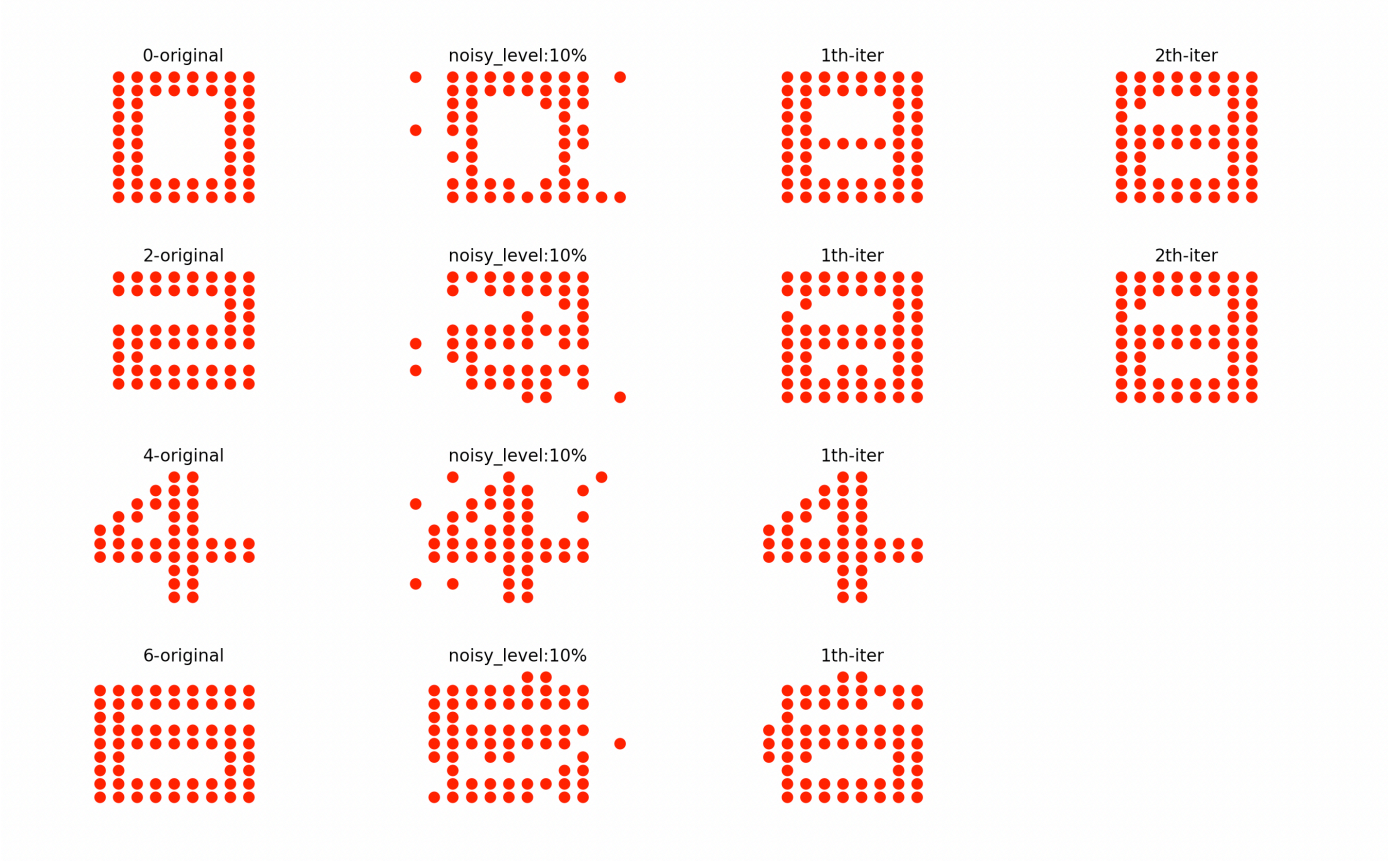
```
     * * * * * * * * *
     * *
     * * * * * * * * *
     * * * * * * * * *



0-th iteration:
           * * *
     *   * * *
   *   * * * * *   *
     * * * * * *   *
 * *   *
 * *   * * * * * * * *
     * * * * * * *   *
           * *     *
         * * *       *
     *       *   *



1-th iteration:
           * *
         * * *
       * * * *
     * *   * *
   * *     * *
   * * * * * * * * *
   * * * * * * * * *
           * *
           * *
           * *



2-th iteration:
           * *
         * * *
       * * * *
     * *   * *
   * *     * *
   * * * * * * * * *
   * * * * * * * * *
           * *
           * *
           * *



0-th iteration:
             *         *
```

```
   *   * *   * * *
   * * * *   *   * *
   * *   *     * *     *
   * *     *   * *
   * * * * * * *
   * *           * *
   * *           * *
   * * * * * * * * * *
   * * * * * * * * * *   *


1-th iteration:

   * * * * * * * * * *
   * * * * * * * * * *
   * *
   * * * * * * * * * *
   * * * * * * * * * *
   * *           * *
   * *           * *
   * * * * * * * * * *
   * * * * * * * * * *


2-th iteration:

   * * * * * * * * * *
   * * * * * * * * * *
   * *
   * * * * * * * * * *
   * * * * * * * * * *
   * *           * *
   * *           * *
   * * * * * * * * * *
   * * * * * * * * * *
```

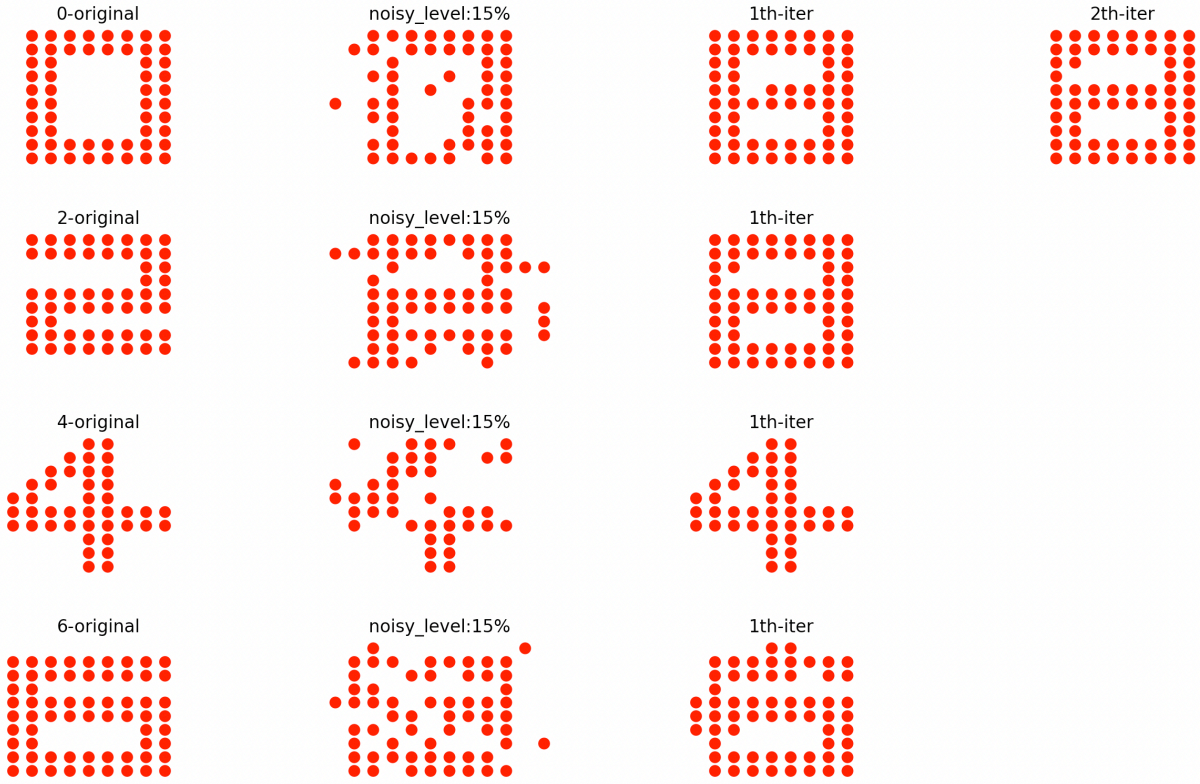
Python

接下来在噪声等级分别为10%，15%，20%，25%下测试Hopfield网络的记忆能力：

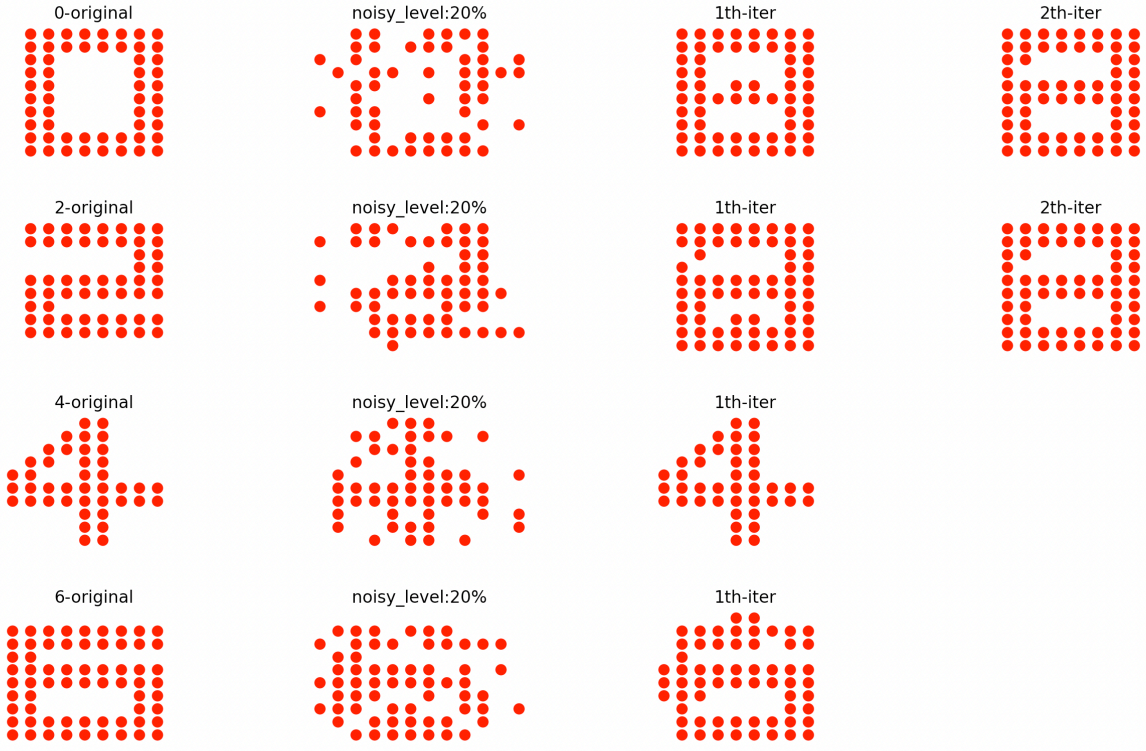
1.10%-noisy



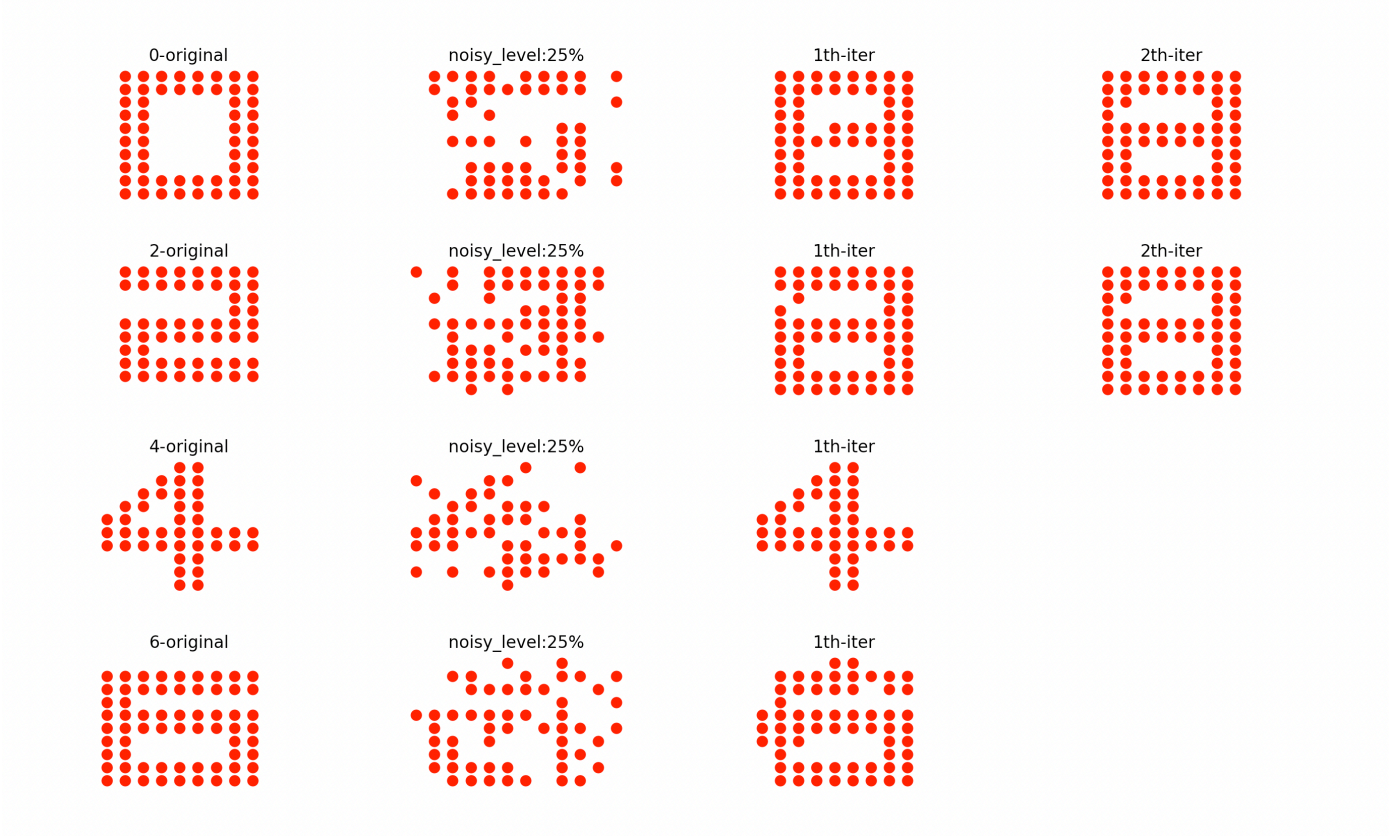
2.15%-noisy


0-original
noisy_level:15%
1th-iter
2th-iter
2-original
noisy_level:15%
1th-iter
4-original
noisy_level:15%
1th-iter
6-original
noisy_level:15%
1th-iter

3.20%-noisy


0-original
noisy_level:20%
1th-iter
2th-iter
2-original
noisy_level:20%
1th-iter
2th-iter
4-original
noisy_level:20%
1th-iter
6-original
noisy_level:20%
1th-iter

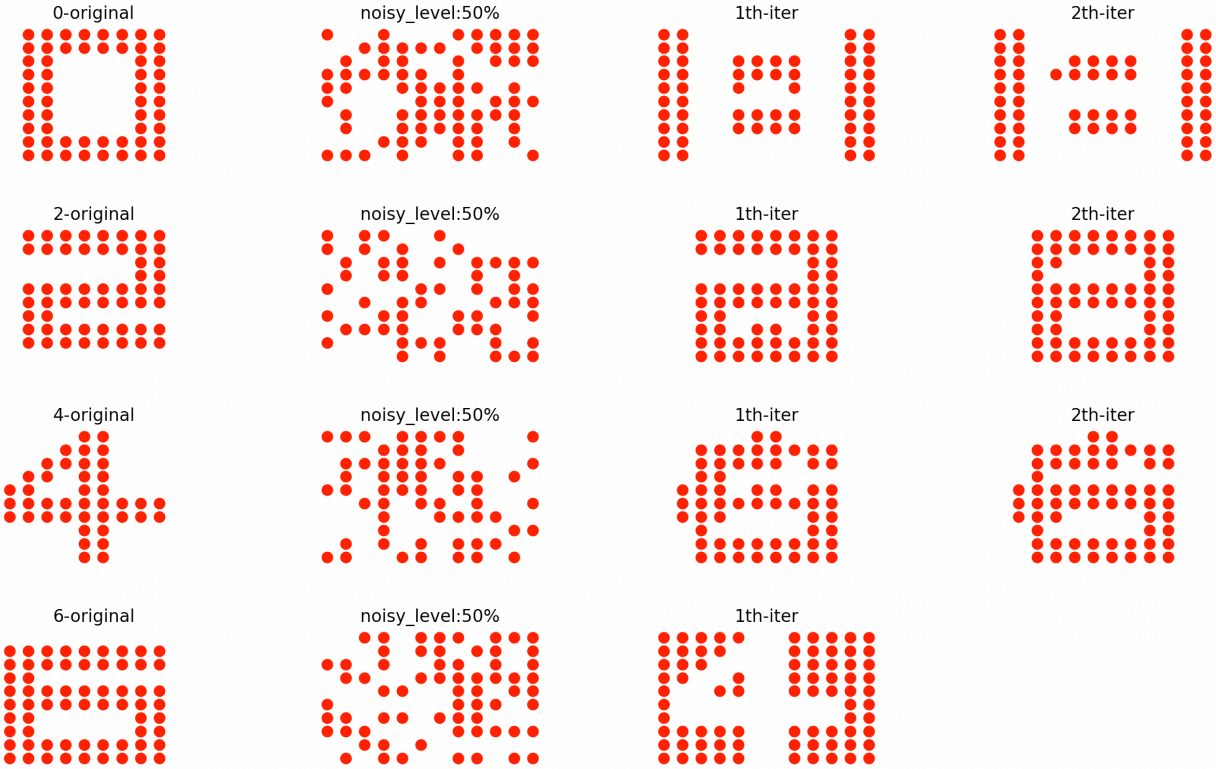
4.25%-noisy



可以看出Hopfield网络仅仅需要迭代很少的次数就可以还原模式，具有很好的记忆能力。

但是Hopfield网络在0，2这两个模式下没有成功Recall到原模式，可能的原因是网络陷入了局部稳定状态而无法跳出

尝试将噪声加大到50%，可以发现Hopfield网络的记忆能力也有限度


0-original
noisy_level:50%
1th-iter
2th-iter
2-original
noisy_level:50%
1th-iter
2th-iter
4-original
noisy_level:50%
1th-iter
2th-iter
6-original
noisy_level:50%
1th-iter