

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

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

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：

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

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

```
1 | for key in original_patterns:  
2 |     pattern = original_patterns[key]  
3 |     w += np.outer(pattern, pattern)  
4 | w /= n_pattern  
5 | np.fill_diagonal(w, 0)
```

2. Recall

Hopfield 网络的 recall 过程可以表示为：

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

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 ($\text{sgn}(x) = 1$ $x \geq 0$; $\text{sgn}(x) = -1$ $x < 0$)， w_{ij} 是权重矩阵的第 i 行第 j 列的元素。

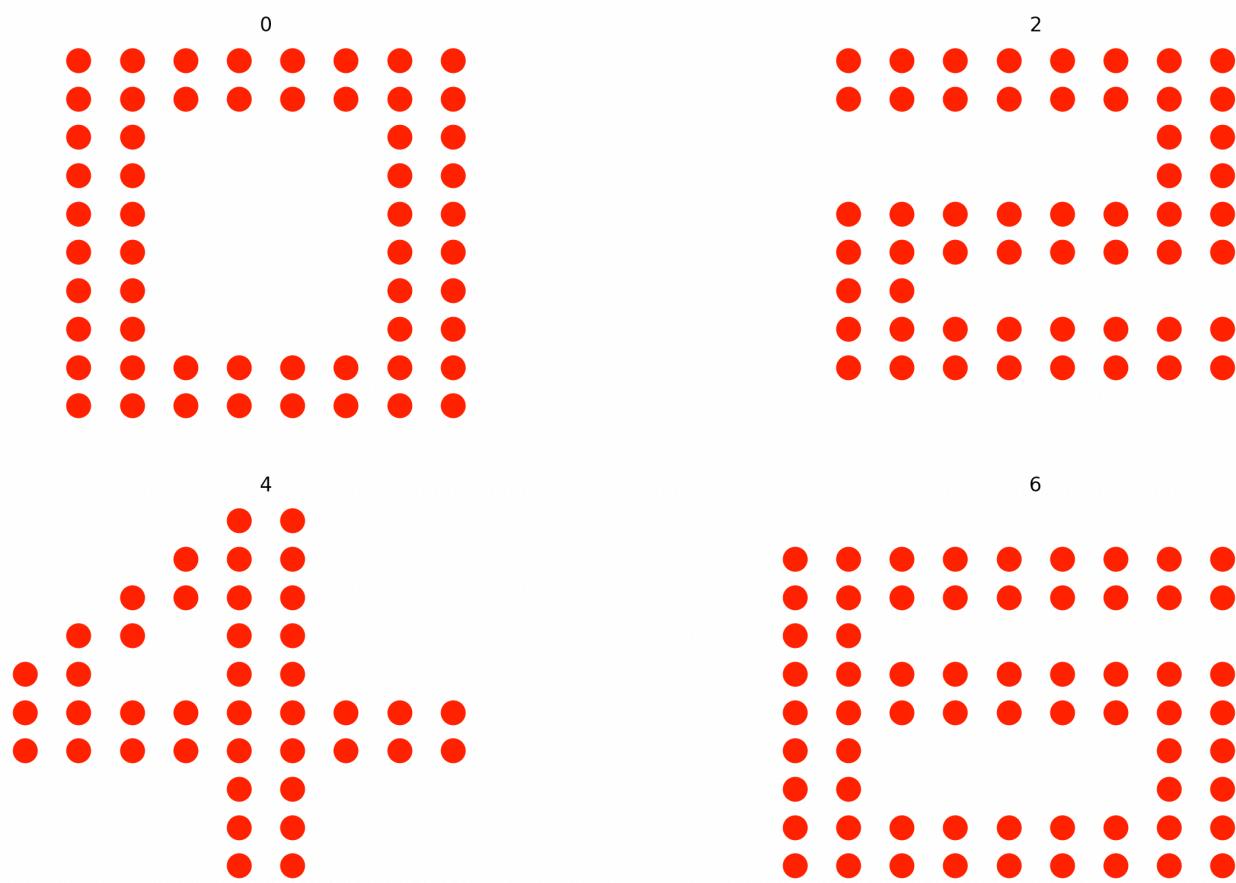
有异步更新和同步更新两种方式，异步更新一次更新一个神经元，同步更新一次更新所有神经元，全部神经元更新完一次为一轮。当新一轮神经元相较于上一轮神经元没有变化时，迭代停止。

```
1 while True:
2
3     if recall_method == synchronous_update:
4         new_pattern = np.where(w @ noisy_pattern >= 0, 1, -1)
5
6         if np.array_equal(new_pattern, noisy_pattern):
7             break
8
9         noisy_pattern = new_pattern
10
11    if recall_method == asynchronous_update:
12
13        prev_pattern = noisy_pattern.copy()
14
15        for i in range(n_neuron):
16            net_input = w[i] @ noisy_pattern
17
18            noisy_pattern[i] = 1 if net_input >= 0 else -1
19
20        if np.array_equal(noisy_pattern, prev_pattern):
21            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](#)

```
1 //*****
2 /*We refer to the code for class. */
3 //*****
4 #include <stdlib.h>
5 #include <stdio.h>
6 #include <time.h>
7
8 #define n_neuron 120 //12*10
9 #define n_pattern 4 //number of pattern (0,2,4,6)
10 #define n_row 10
11 #define n_column 12
12 #define noise_rate 0.15 //{0% 10% 15%}
13
14 int pattern[n_pattern][n_neuron]={
15     {1, 1,-1,-1,-1,-1,-1,-1,-1, 1, 1,
```

```

16     1, 1,-1,-1,-1,-1,-1,-1,-1, 1, 1,
17     1, 1,-1,-1, 1, 1, 1, 1,-1,-1, 1, 1,
18     1, 1,-1,-1, 1, 1, 1, 1,-1,-1, 1, 1,
19     1, 1,-1,-1, 1, 1, 1, 1,-1,-1, 1, 1,
20     1, 1,-1,-1, 1, 1, 1, 1,-1,-1, 1, 1,
21     1, 1,-1,-1, 1, 1, 1, 1,-1,-1, 1, 1,
22     1, 1,-1,-1, 1, 1, 1, 1,-1,-1, 1, 1,
23     1, 1,-1,-1,-1,-1,-1,-1,-1, 1, 1,
24     1, 1,-1,-1,-1,-1,-1,-1,-1, 1, 1},
25 {1, 1,-1,-1,-1,-1,-1,-1,-1, 1, 1,
26     1, 1,-1,-1,-1,-1,-1,-1,-1, 1, 1,
27     1, 1, 1, 1, 1, 1, 1, 1,-1,-1, 1, 1,
28     1, 1, 1, 1, 1, 1, 1, 1,-1,-1, 1, 1,
29     1, 1,-1,-1,-1,-1,-1,-1,-1, 1, 1,
30     1, 1,-1,-1,-1,-1,-1,-1,-1, 1, 1,
31     1, 1,-1,-1, 1, 1, 1, 1, 1, 1, 1, 1,
32     1, 1,-1,-1,-1,-1,-1,-1,-1, 1, 1,
33     1, 1,-1,-1,-1,-1,-1,-1,-1, 1, 1,
34     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
35 {1, 1, 1, 1, 1,-1,-1, 1, 1, 1, 1, 1, 1,
36     1, 1, 1, 1,-1,-1,-1, 1, 1, 1, 1, 1, 1,
37     1, 1, 1,-1,-1,-1,-1, 1, 1, 1, 1, 1, 1,
38     1, 1,-1,-1, 1,-1,-1, 1, 1, 1, 1, 1, 1,
39     1,-1,-1, 1, 1,-1,-1, 1, 1, 1, 1, 1, 1,
40     1,-1,-1,-1,-1,-1,-1,-1,-1, 1, 1, 1, 1,
41     1,-1,-1,-1,-1,-1,-1,-1,-1, 1, 1, 1, 1,
42     1, 1, 1, 1, 1,-1,-1, 1, 1, 1, 1, 1, 1,
43     1, 1, 1, 1, 1,-1,-1, 1, 1, 1, 1, 1, 1,
44     1, 1, 1, 1, 1,-1,-1, 1, 1, 1, 1, 1, 1},
45 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
46     1,-1,-1,-1,-1,-1,-1,-1,-1, 1, 1, 1, 1,
47     1,-1,-1,-1,-
48     1,-1,-1,-1,-1,-1, 1, 1,
49     1,-1,-1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
50     1,-1,-1,-1,-1,-1,-1,-1,-1, 1, 1, 1,
51     1,-1,-1,-1,-1,-1,-1,-1,-1, 1, 1, 1,
52     1,-1,-1, 1, 1, 1, 1, 1,-1,-1, 1, 1, 1,
53     1,-1,-1, 1, 1, 1, 1, 1,-1,-1, 1, 1, 1,
54     1,-1,-1,-1,-1,-1,-1,-1,-1, 1, 1, 1, 1,
55     1,-1,-1,-1,-1,-1,-1,-1,-1, 1, 1, 1}}};

56

57 int w[n_neuron][n_neuron];//120*120
58 int v[n_neuron]; //vector 120
59 /*************************************************************************/
60 /* Output the patterns, to confirm they are the desired ones          */
61 /* /*************************************************************************/
62 void Output_Pattern(int k) {
63     FILE *outputfile;
64     int i,j;
65
66     outputfile = fopen("./outpat3.txt", "a");
67     fprintf(outputfile,"Pattern[%d]:\n",k);

```

```

68 for(i=0;i<n_row;i++)
69 { for(j=0;j<n_column;j++)
70 fprintf(outputfile,"%2c",(pattern[k][i*n_column+j]==-1)?'*':' ');
71     fprintf(outputfile,"\n");
72 }
73 fprintf(outputfile,"\n\n\n");
74 fclose(outputfile);
75 getchar();
76 }

77 ****
78 /* Output the state of the network in the form of a picture */
79 ****
80 ****
81 void Output_State(int k) {
82     int i,j;
83     FILE *outputfile;
84
85     outputfile = fopen("./outsta3.txt", "a");
86
87     fprintf(outputfile, "%d-th iteration:\n",k);
88     for(i=0;i<n_row;i++)
89     { for(j=0;j<n_column;j++)
90     fprintf(outputfile,"%2c",(v[i*n_column+j]==-1)?'*':' ');
91         fprintf(outputfile,"\n");
92     }
93     fprintf(outputfile,"\n\n\n");
94     fclose(outputfile);
95     getchar();
96 }

97 ****
98 /* Store the patterns into the network */
99 ****
100 ****
101 void Store_Pattern() {
102     //Phase1: Storage
103     int i,j,k;
104     for(i=0; i<n_neuron; i++){
105         for(j=0; j<n_neuron; j++){
106             w[i][j] = 0; //Initialization W =0
107
108             for(k=0;k<n_pattern;k++)
109                 w[i][j] += pattern[k][i]*pattern[k][j]; //W=W+s(m)*s(m)^T
110             w[i][j] /= (double)n_pattern; //nomalazation
111         }
112         w[i][i]=0;//w_ii=0
113     }
114 }

115 ****
116 /* Recall the m-th pattern from the network */
117 /* The pattern is corrupted by some noises */
118 ****
119 ****

```

```

120 void Recall_Pattern(int m){
121     //Phase 2: Recall
122     int i,j,k;
123     int n_update;
124     int net, vnew;
125     double r;
126
127     for(i=0;i<n_neuron;i++){
128         r=(double)(rand()%10001)/10000.0; //r:random
129         if(r<noise_rate)
130             v[i]=(pattern[m][i]==1)?-1:1;//corrupted by noises
131         else
132             v[i]=pattern[m][i]; //v<-pattern
133     }
134     Output_State(0); /* show the noisy input pattern */
135
136     k=1;
137     do { //recall
138         n_update=0; //
139         for(i=0; i<n_neuron; i++){
140             net=0;
141             for(j=0; j<n_neuron; j++)//Find the new output
142                 net+=w[i][j]*v[j]; //Σ(w*v)
143                 if(net >= 0) vnew = 1;
144                 if(net < 0) vnew = -1;
145                 if(vnew != v[i]) {
146                     n_update++;
147                     v[i]=vnew;
148                 }
149             }
150             Output_State(k); /* show the current result */
151             k++;
152         } while(n_update!=0); //Stop if there is no change after many iterations
153     }
154
155     ****
156     /* Initialize the weights */ ****
157     ****
158     void Initialization()
159     {int i,j;
160
161         for(i=0; i<n_neuron; i++)
162             for(j=0; j<n_neuron; j++)
163                 w[i][j] = 0;
164     }
165
166     ****
167     /* The main program */ ****
168     ****
169     int main(){
170         int k;
171

```

```

172 |     for(k=0;k<n_pattern;k++) Output_Pattern(k);
173 |
174 |     Initialization();
175 |     Store_Pattern();
176 |     for(k=0;k<n_pattern;k++) Recall_Pattern(k);
177 }
178

```

Python

[project3.py](#):

```

1 import json
2 import numpy as np
3 import matplotlib.pyplot as plt
4 np.set_printoptions(threshold=np.inf)
5
6 # Load patterns and noisy patterns
7
8 noisy_level = 15 # 0, 10, 15, 20, 25, 40, 50
9
10 noisy_file = "./pattern_"+str(noisy_level)+".json"
11
12 with open("./pattern.json", "r") as f:
13     original_patterns = json.load(f)
14
15 with open(noisy_file, "r") as f:
16     noisy_patterns = json.load(f)
17
18 n_neuron = 120
19 n_pattern = 4
20
21 # Convert patterns to numpy arrays
22 for key in original_patterns:
23     original_patterns[key] = np.array(original_patterns[key])
24
25 for key in noisy_patterns:
26     noisy_patterns[key] = np.array(noisy_patterns[key])
27
28 # Initialize weight matrix
29 w = np.zeros((n_neuron, n_neuron))
30
31
32 # Store patterns in the weight matrix
33 for key in original_patterns:
34     pattern = original_patterns[key]
35     w += np.outer(pattern, pattern)
36 w /= n_pattern
37 np.fill_diagonal(w, 0)
38 # 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())

```

```

91     pattern_store[key] = recalled_patterns
92
93
94
95 def num_to_color(num):
96     if num == 1:
97         return "white"
98     elif num == -1:
99         return "red"
100
101
102 max_length = 0
103 for key, values in pattern_store.items():
104     max_length = max(max_length, len(values))
105
106 fig, axs = plt.subplots(4, max_length, figsize=(20, 10), gridspec_kw=
107 { 'wspace': 0.5, 'hspace': 0.5})
108 spacing = 0.2
109 row = 0
110 for key, values in pattern_store.items():
111     count = 0
112     for p in values:
113         for i in range(10):
114             for j in range(12):
115                 color = num_to_color(p[i * 12 + j])
116                 axs[row, count].scatter(j * spacing, -i * spacing, c=color,
s=50)
117                 axs[row, count].axis("off")
118                 axs[row, count].set_xlim(-0.5 * spacing, 11.5 * spacing)
119                 axs[row, count].set_ylim(-9.5 * spacing, 0.5 * spacing)
120                 if count == 0:
121                     axs[row, count].set_title(f"{key}-original")
122                 if count == 1:
123                     axs[row, count].set_title(f"noisy_level:{noisy_level}%")
124                 if count > 1:
125                     axs[row, count].set_title(f"{count-1}th-iter")
126                 count += 1
127             row += 1
128
129 for i in range(row):
130     for j in range(max_length):
131         axs[i, j].axis("off")
132
133 plt.show()
134
135

```

5.结果讨论

C

原始图像：

```
1 Pattern[0]:  
2     * * * * * * * * *  
3     * * * * * * * * *  
4     * *           * *  
5     * *           * *  
6     * *           * *  
7     * *           * *  
8     * *           * *  
9     * *           * *  
10    * * * * * * * * *  
11    * * * * * * * * *  
12  
13  
14  
15 Pattern[1]:  
16    * * * * * * * * *  
17    * * * * * * * * *  
18          * *  
19          * *  
20    * * * * * * * * *  
21    * * * * * * * * *  
22    * *  
23    * * * * * * * * *  
24    * * * * * * * * *  
25  
26  
27  
28  
29 Pattern[2]:  
30      * *  
31      * * *  
32      * * * *  
33      * *   * *  
34      * *   * *  
35    * * * * * * * * *  
36    * * * * * * * * *  
37      * *  
38      * *  
39      * *  
40  
41  
42  
43 Pattern[3]:  
44  
45    * * * * * * * * *
```

```
46 * * * * * * * * *  
47 * *  
48 * * * * * * * * *  
49 * * * * * * * * *  
50 * * * * * * * *  
51 * * * * * * * *  
52 * * * * * * * * *  
53 * * * * * * * * *  
54  
55
```

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

```
1 0-th iteration:  
2 * * * * * * * * *  
3 * * * * * * * * *  
4 * * * * * * * *  
5 * * * * * * * *  
6 * * * * * * * *  
7 * * * * * * * *  
8 * * * * * * * *  
9 * * * * * * * *  
10 * * * * * * * * *  
11 * * * * * * * * *  
12  
13  
14  
15 1-th iteration:  
16 * * * * * * * * *  
17 * * * * * * * * *  
18 * * * * * * * *  
19 * * * * * * * *  
20 * * * * * * * *  
21 * * * * * * * *  
22 * * * * * * * *  
23 * * * * * * * *  
24 * * * * * * * * *  
25 * * * * * * * * *  
26  
27  
28  
29 2-th iteration:  
30 * * * * * * * * *  
31 * * * * * * * * *  
32 * * * * * * * *  
33 * * * * * * * *  
34 * * * * * * * *  
35 * * * * * * * *  
36 * * * * * * * *  
37 * * * * * * * *  
38 * * * * * * * * *
```

```
39      * * * * * * *
40
41
42
43 0-th iteration:
44      *      *      * * * * *
45          *      * * * * *   *
46      *                  * *   *
47                      * *
48          * * * * *       * * *
49      * * * * * * * * *
50      * *      * *
51      * *          * *   *
52      * * * * * * * * *
53
54
55
56
57 1-th iteration:
58      * * * * * * * *
59          * * * * * * *
60                      * *
61                      * *
62      * *      * *   * *
63      * * * * * * * * *
64      * *
65      * * * * * * * *
66      * * * * * * * *
67
68
69
70
71 2-th iteration:
72      * * * * * * * *
73          * * * * * * *
74                      * *
75          *           * *
76      * *      * *   * *
77      * * * * * * * * *
78      * *
79      * * * * * * * *
80      * * * * * * * *
81
82
83
84
85 3-th iteration:
86      * * * * * * * *
87          * * * * * * *
88                      * *
89          *           * *
90      * *      * *   * *
```

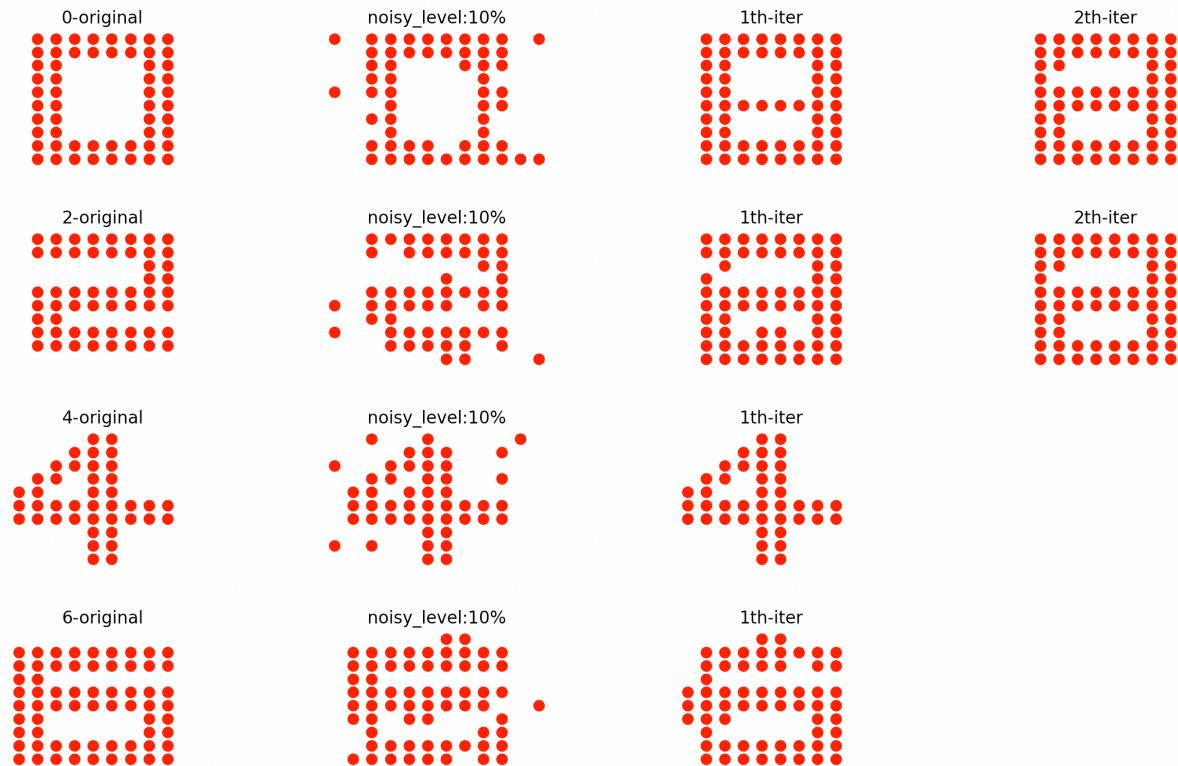
```
91      * * * * * * *
92      *
93      * * * * * * *
94      * * * * * * *
95
96
97
98
99 0-th iteration:
100     * *
101     * * *
102     * * * * *
103     * * * * *
104     * *
105     * * * * * * *
106     * * * * * *
107     * *
108     * * *
109     *
110
111
112
113 1-th iteration:
114     *
115     * *
116     * * *
117     * *
118     * *
119     * * * * * * *
120     * * * * * * *
121     *
122     *
123     *
124
125
126
127 2-th iteration:
128     *
129     * *
130     * * *
131     * *
132     * *
133     * * * * * * *
134     * * * * * * *
135     *
136     *
137     *
138
139
140
141 0-th iteration:
142     *
```

```
143 * * * * *
144 * * * * * *
145 * * * * * * *
146 * * * * *
147 * * * * * * *
148 * * * *
149 * *
150 * * * * * * * * *
151 * * * * * * * * * *
152
153
154
155 1-th iteration:
156
157 * * * * * * * * *
158 * * * * * * * * *
159 * *
160 * * * * * * * * *
161 * * * * * * * * *
162 * *
163 * *
164 * * * * * * * * *
165 * * * * * * * * *
166
167
168
169 2-th iteration:
170
171 * * * * * * * * *
172 * * * * * * * * *
173 * *
174 * * * * * * * * *
175 * * * * * * * * *
176 * *
177 * *
178 * * * * * * * * *
179 * * * * * * * * *
180
181
182
183
```

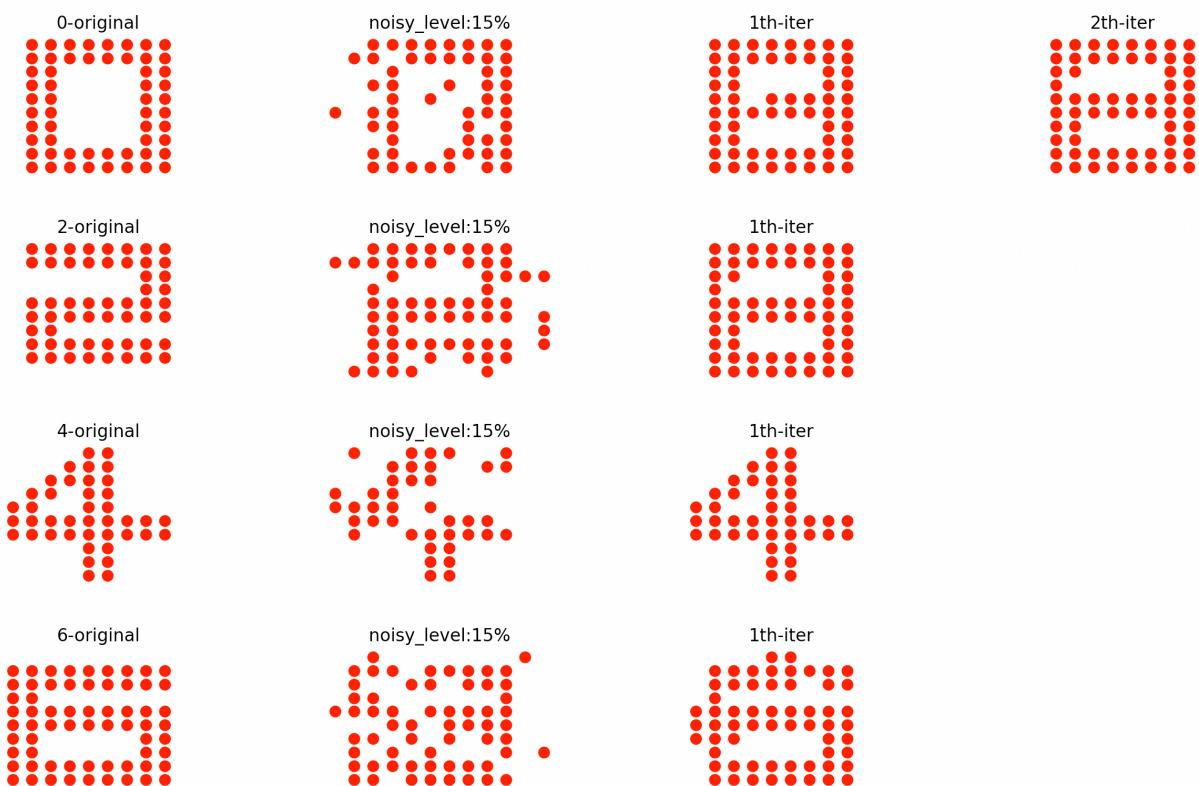
Python

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：

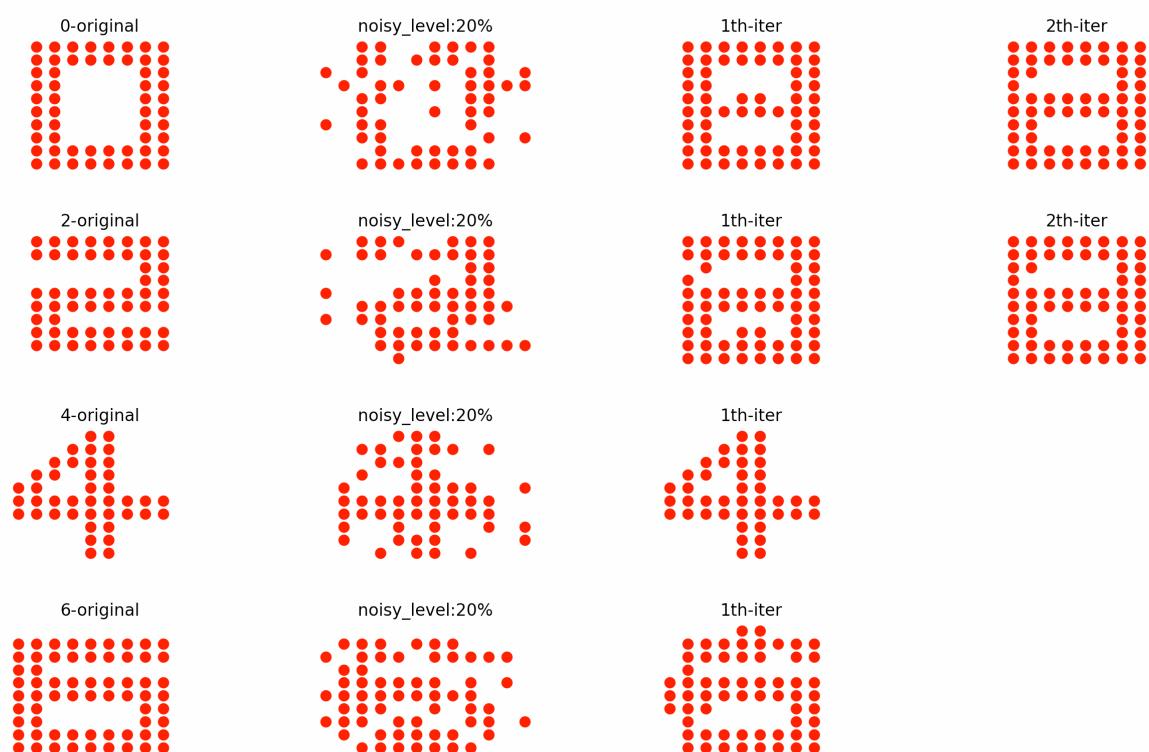
1.10%-noisy



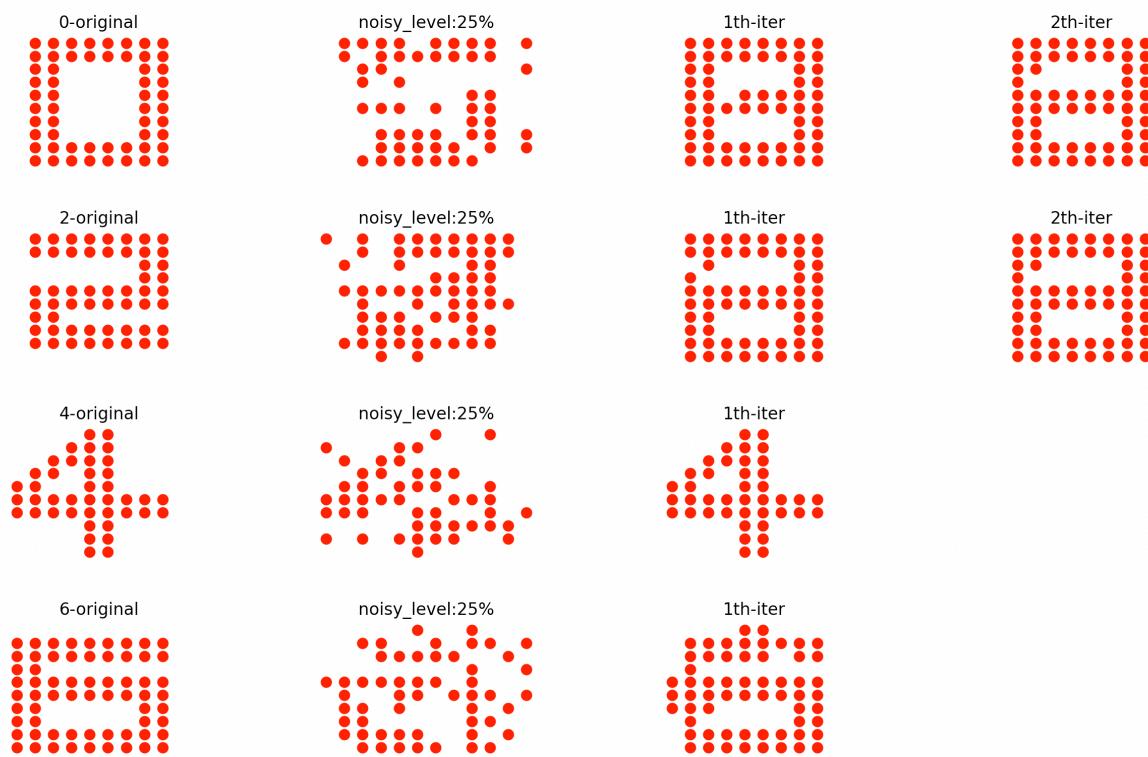
2.15%-noisy



3.20%-noisy



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

