卒業論文

# スパース符号化された連想記憶モデルの想起特性

76050580　**山本 浩史**

**指導教員　伊達 章 准教授**

2009 **年** 2 **月**

**宮崎大学 工学部 情報システム工学科**

# 目次

# 第 1 章

# はじめに

## 1.1 背景

人は誰かの顔を見たとき，知り合いかどうか，すぐに分かる．人は「知らない」ことを一点の曇もなく即座に答えることができる．よく考えてみると，これは不思議だ．どのような仕組みで動いているのだろうか．この機能をコンピュータで再現したい．

## 1.2 本研究のねらい

脳の記憶をモデル化したものに連想記憶モデルがある．連想記憶とは，「りんご」という言葉から，「赤い」や「丸い」を思い出したり，知人の「声」から「顔」を思い出すように，一部の情報から実際の記憶に近いものを思い出すことである．連想記憶モデルの研究は古くから行われてきたが，想起の成否を容易に判断できるモデルはなかった．Hopfield は 2008 年に，この機能を説明する新しい連想記憶モデル（2008 モデル）を発表した [3]．これまでのモデル [1, 2] とは記憶パターンの生成方法，結合係数の設定方法，活動のダイナミクスが異なる．本研究では，そのうち何が本質的に効いているのか明らかにする．さらに，同じ機能をより単純化したモデルを用い実現できたので，それを報告する．

## 1.3 本論文の構成

本論文の以下の構成は，次のようになっている．第 2 章では，2008 モデルの紹介，第 3 章では，想起の成否が容易に判別できることは，本質的に何が効いているのか調べるため実験を行った．最後に，第 4 章で本論文のまとめを述べる．なお，付録として各実験で使用したモデルの想起精度，ソースコードを添付した．

# 第 2 章

# Hopfield2008 モデル

## 2.1 数理モデル

まず Hopfield によって提案された，連想記憶モデルについて簡単に紹介しよう．このモデルと通常の連想記憶モデルとは，記憶パターンの生成方法，結合係数の設定方法，活動のダイナミクスの 3 点が異なる．

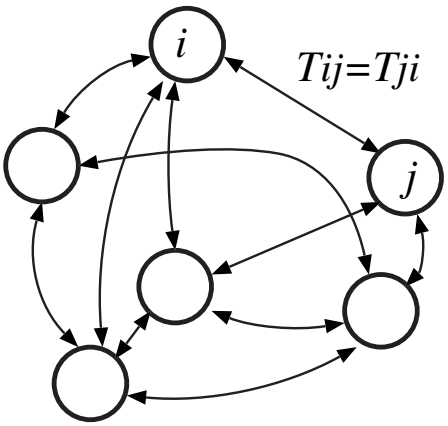


**図** 2.1. 相互結合型の神経回路モデル

モデルは $N$ 個のニューロンとそれらをつなぐ結合から成り立っている (図 2.1)．$i$ 番目のニューロンの内部状態を $u_i$，出力を $V_i$ で表す．各々の結合に対して，その強さを表す実数値 $T_{ij}$ が定められている．ここで，$T_{ij}$ は $j$ 番目から $i$ 番目へのニューロンの結合の重みであり，$T_{ii} = 0$ とする．各素子は，その総入力に応じて，自分の状態を更新する．具体的には，以下の式に従う．

$$\frac{d}{dt}u_i(t) = -\frac{u_i(t)}{\tau} + \sum_{j=1}^{1000} T_{ij}V_j(t) - h(t) \tag{2.1}$$

$$V_i(t+1) = u_i(t+1),\ \text{if}\ u_i(t+1) < 0\ \Rightarrow\ 0 \tag{2.2}$$

$h$ はしきい値に対応する．$h$ は以下の式に従い，時間的に変化する．

$$h = -30.0 + 3.5(\sum_{k=1}^{1000} V_k(t)) \tag{2.3}$$

$$\text{if } h < 0 \Rightarrow h = 0 \tag{2.4}$$

## 2.2 記憶パターンの学習

$N = 1000$ の回路を考えよう．回路に記憶させる記憶パターンの一例を図 2.2 に示す．$50 \times 20 = 1000$ 個のニューロンの活動が示されている．このパターンは 50 行から構成されているが，1 つの行には 1 つのニューロンしか興奮していない．さらに，左側に表示されているニューロンほど興奮しやすくなっている．Zipf の法則に従い $k$ 列目のニューロンが選ばれる確率が，

$$p(k) = \frac{1/k}{\sum_{n=1}^{20} 1/n} \tag{2.5}$$

に従う．このようなスパース符号化された記憶パターンを複数生成する． 通常のモデルは，

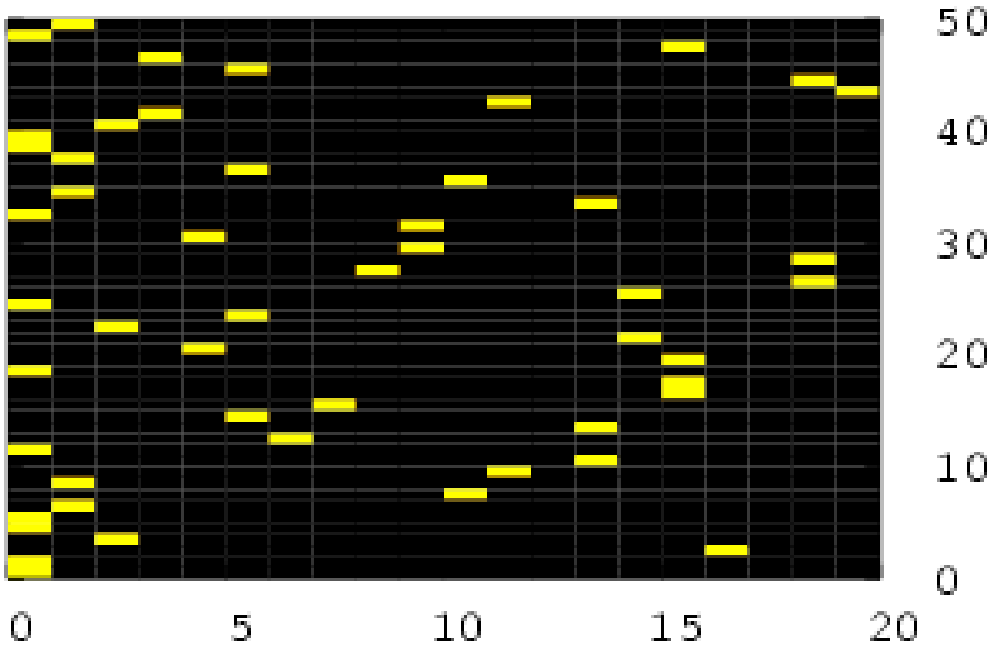


**図** 2.2. Zipf の法則で生成された記憶パターンの例

結合係数を Hebb 学習で変更するが，2008 モデルでは $i$ 番目と $j$ 番目のニューロンが 1 度でも同時に興奮すれば，$T_{ij} = T_{ji} = 1$ と設定される．

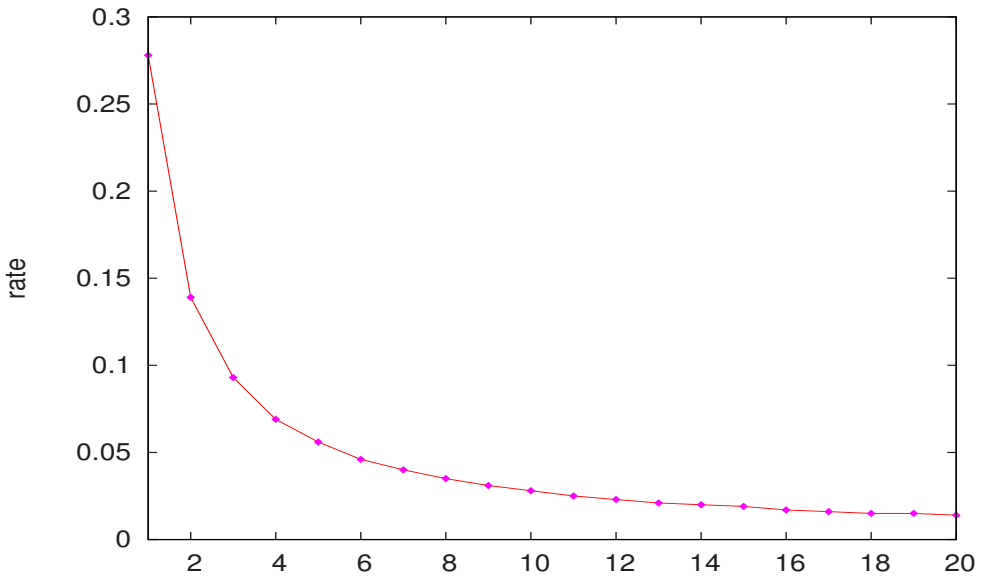


**図** 2.3. 各列の平均発火頻度．横軸は図 2.2 の列に対応する．

## 2.3　シミュレーション

2008 モデルにおける想起過程の様子を紹介しよう．記憶させるパターン数 $m = 225$ とした．回路の初期状態を $V_i(0),\ i = 1, \ldots, 1000$ として，1 番目のパターンの 50 行のうち，1 から $\alpha$ 番目の行に正しい 0，1 の値を入れる．例えば，$\alpha = 10$ のときは，1000 個のうち最初の 200 個に正しい 0，1 のパターンを入れて，残りの 800 個は 0 にしておく．どの程度正しい記憶が想起できているか調べるため，式 (2.6) で類似度を求めた．

$$\text{類似度} = \frac{\text{正しく想起できた数} - \text{間違って想起した数}}{50} \tag{2.6}$$

ただし，(類似度) $< 0$ となるときは，(類似度) $= 0$ とした．想起の過程を図 2.4[*1]に示す．明るい色はニューロンの活動度が高く，暗い色ほど活動度が低いことを表し，黒い部分は興奮していないことを表す．活動度の値は図右側のカラースケールに従う．初期値 $\alpha = 10$ の場合で想起させると図 2.4 のようになった．すぐに類似度は 1.0 となり，想起に成功したことが分かる．次に，想起に失敗する様子を調べるため，図 2.5 のように初期値 $\alpha = 5$ で想起させた．更新後すぐに類似度は 0.0 となり，想起に失敗したことが分かる．600 回更新した時点で，活動度の高いニューロンが左側に集中している．また，2400 回更新しても全体の活動度は一定になっていない．このように 2008 モデルでは，想起に失敗することが，活動度の違いと，平均活動度の高いニューロンが左側に集中することで容易に判断できる．

[*1] 図の左下の文字が重なっているのは 3 次元の表現を 1 方向からプロットしているため．

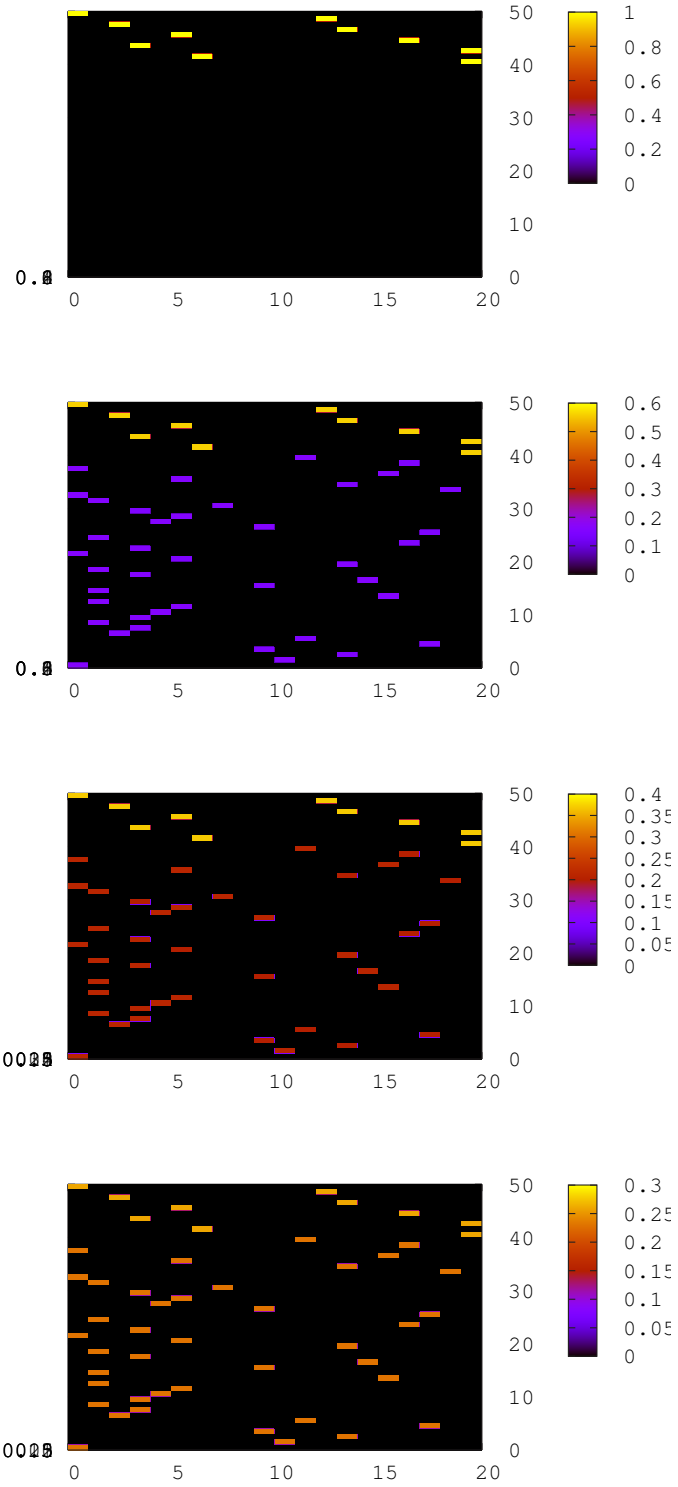


**図** 2.4. $\alpha = 10$. 左上から 0 回，600 回，1200 回，2400 回更新した様子

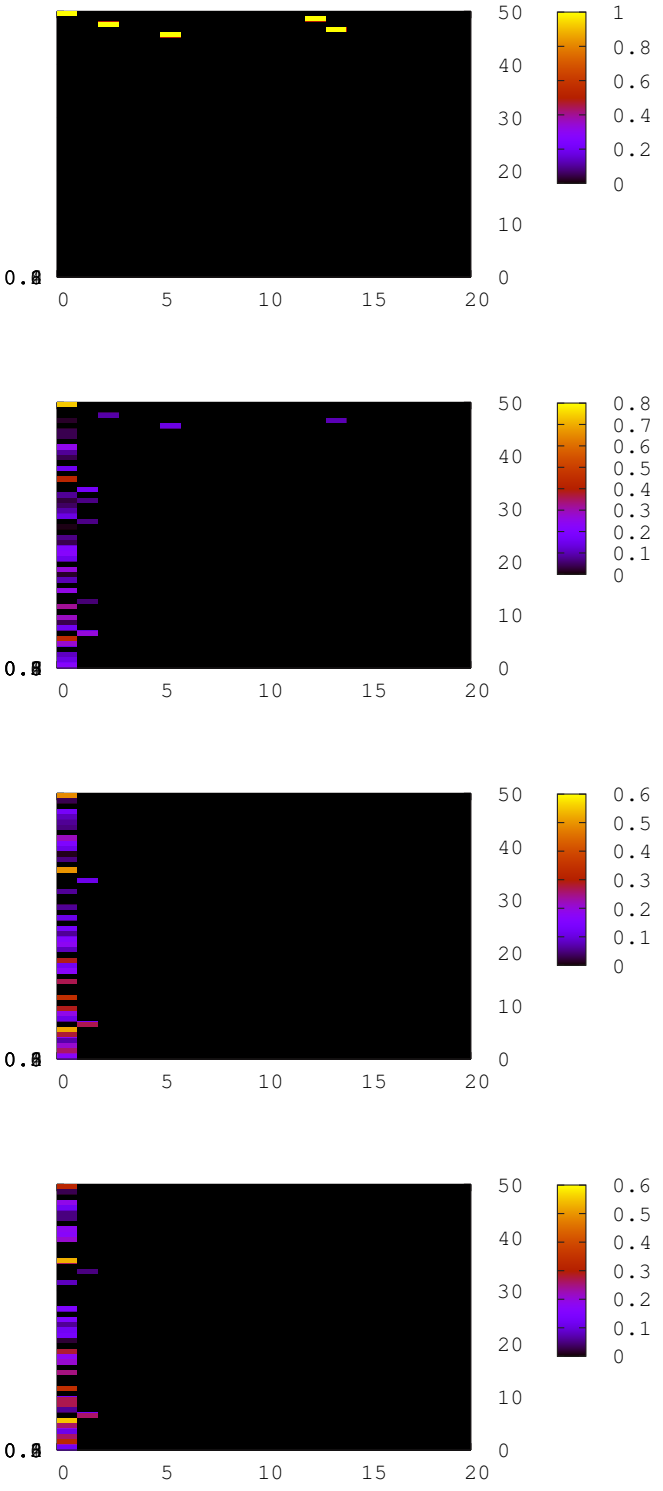


**図** 2.5. $\alpha = 5$. 左上から 0 回，600 回，1200 回，2400 回更新した様子

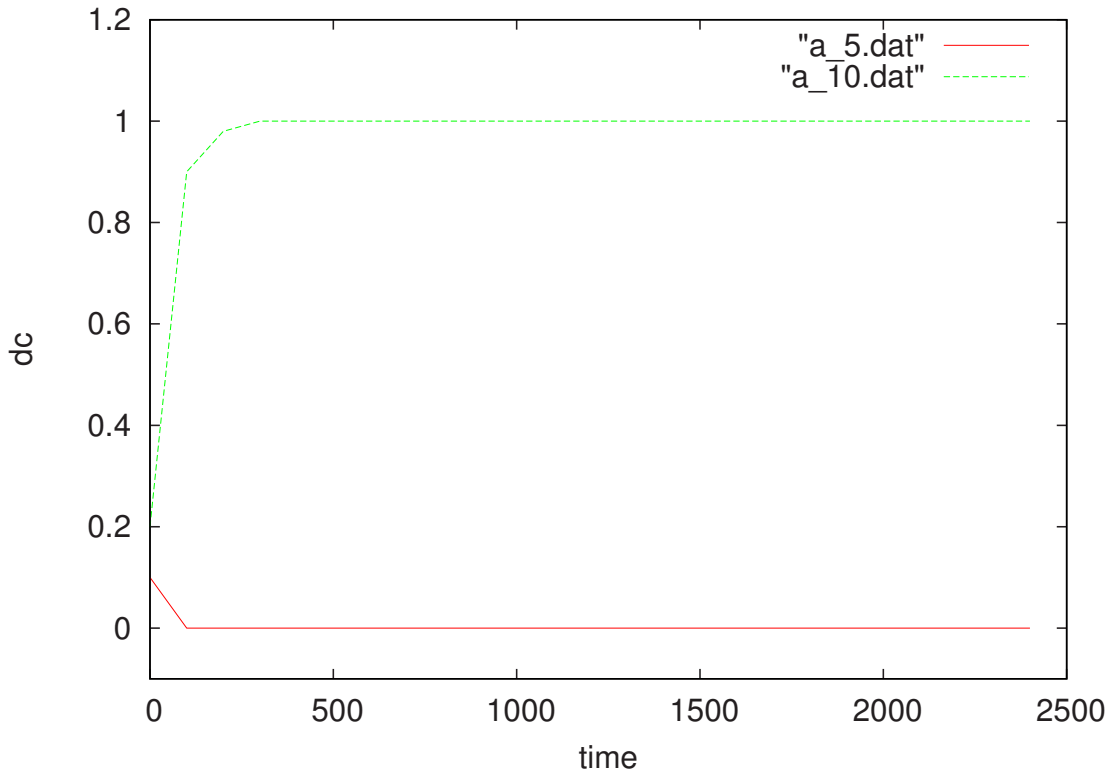


**図** 2.6. （2008 モデル）$\alpha = 10$，$\alpha = 5$ で想起した類似度

# 第 3 章

# 実験

## 3.1 記憶パターンを一様分布で生成する

記憶パターンを Zipf の法則ではなく，一様分布で生成し想起させた．つまり，各行 20 個のニューロンが，それぞれ 5% で興奮する．各行で，1 つのニューロンしか興奮しない条件は変えていない．記憶パターン数 $m = 225$，初期値 $\alpha = 10$ で想起させると図 3.1 のようになった．更新後すぐに類似度は 1.0 となり，想起に成功している．次に，初期値 $\alpha = 5$ で想起した様子を図 3.2 に示す．興奮するニューロンはすぐに決まるが，類似度は 0.0 となり想起に失敗している．失敗するときは，記憶パターンを Zipf の法則で生成した時とは違い，活動度の高いニューロンの位置は分散している．活動度をみると，100 回程度の更新では失敗を判断するのは難しい．700 回程度更新した時点で，失敗と判断できた．ただし，記憶パターンによっては活動度の微妙なものがあり，1000 回程度更新しないと判断できない場合があった．

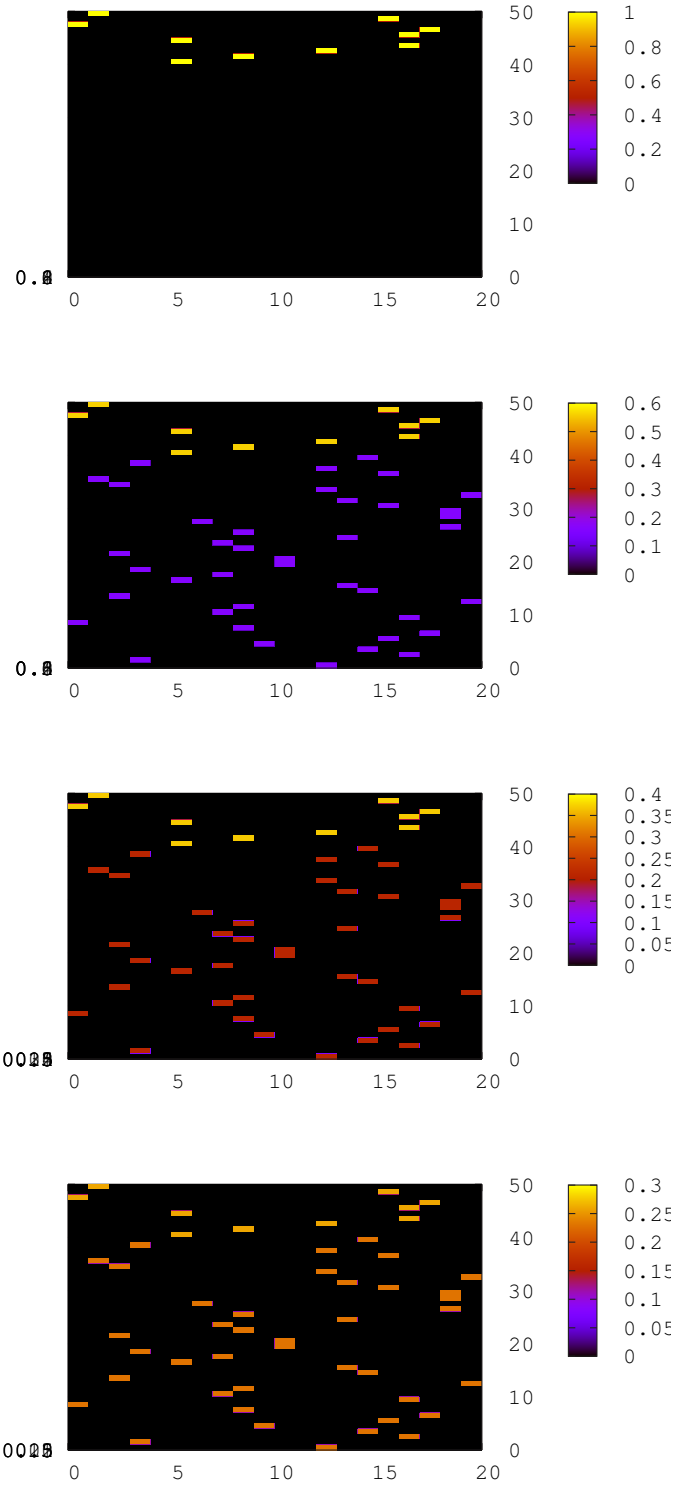


**図 3.1.** $\alpha = 10$. 左上から 0 回，600 回，1200 回，2400 回更新した様子

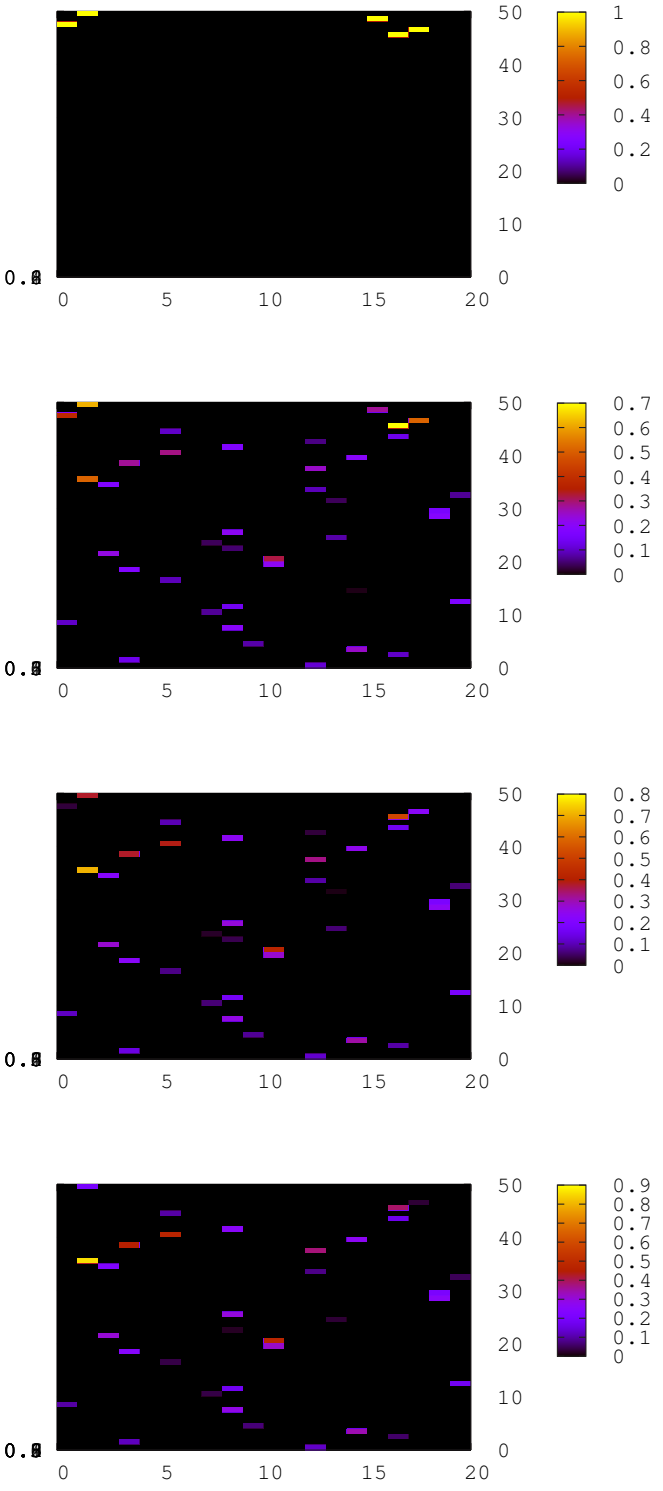


**図** 3.2. $\alpha = 5$. 左上から 0 回，600 回，1200 回，2400 回更新した様子

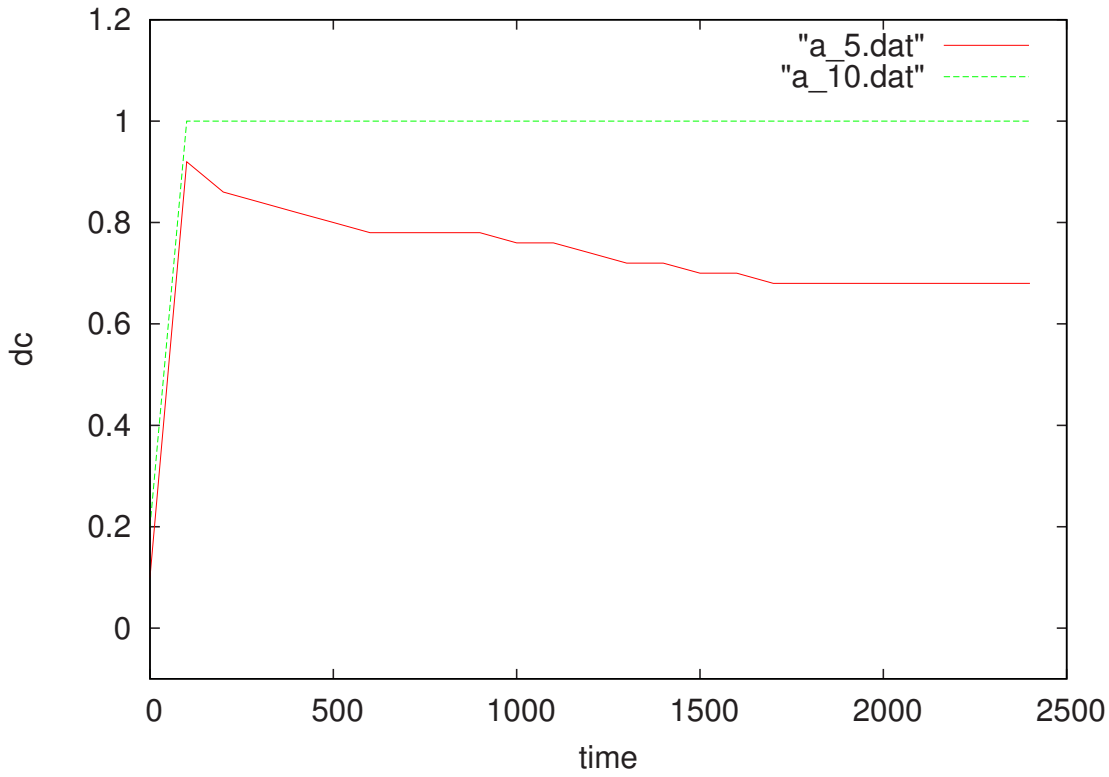


**図** 3.3. （記憶パターン一様分布）$\alpha = 10$, $\alpha = 5$ で想起した類似度

## 3.2　活動のダイナミクスを従来型にする

想起失敗時に活動度の高いニューロンが左に寄ることは，Zipf の法則による記憶パターンの生成が効いているようだ．次に，従来のモデルで用いられている 0，1 の 2 値をとるダイナミクスを利用して想起させた．活動のダイナミクスは，式 (3.1)[1] に従い，記憶パターンの生成は Zipf の法則に従い生成する．なお，このままでは活動が抑えられなくなるので，内部状態を計算し大きいものから 50 個を取り出して更新する．

$$V_i(t) = \begin{cases} 1\ ,\ \text{if}\ \sum_{j=1}^{n} T_{ij} V_j(t-1) > 0 \\ 0\ ,\ \text{otherwise} \end{cases} \tag{3.1}$$

図 3.4 のように，記憶パターン数 $m = 225$，初期値 $\alpha = 10$ で想起させた場合，すぐに類似度 1.0 となり正しく想起できている．図 3.5 に初期値 $\alpha = 5$ で想起したときの様子を示す．類似度はすぐに 0.0 になり，想起に失敗したことが分かる．ダイナミクスが 0，1 の 2 値のため，活動度の違いでは想起の失敗を判断できない．しかし，活動度の高いニューロンが左側に集中するため，すぐに失敗を判断できる．

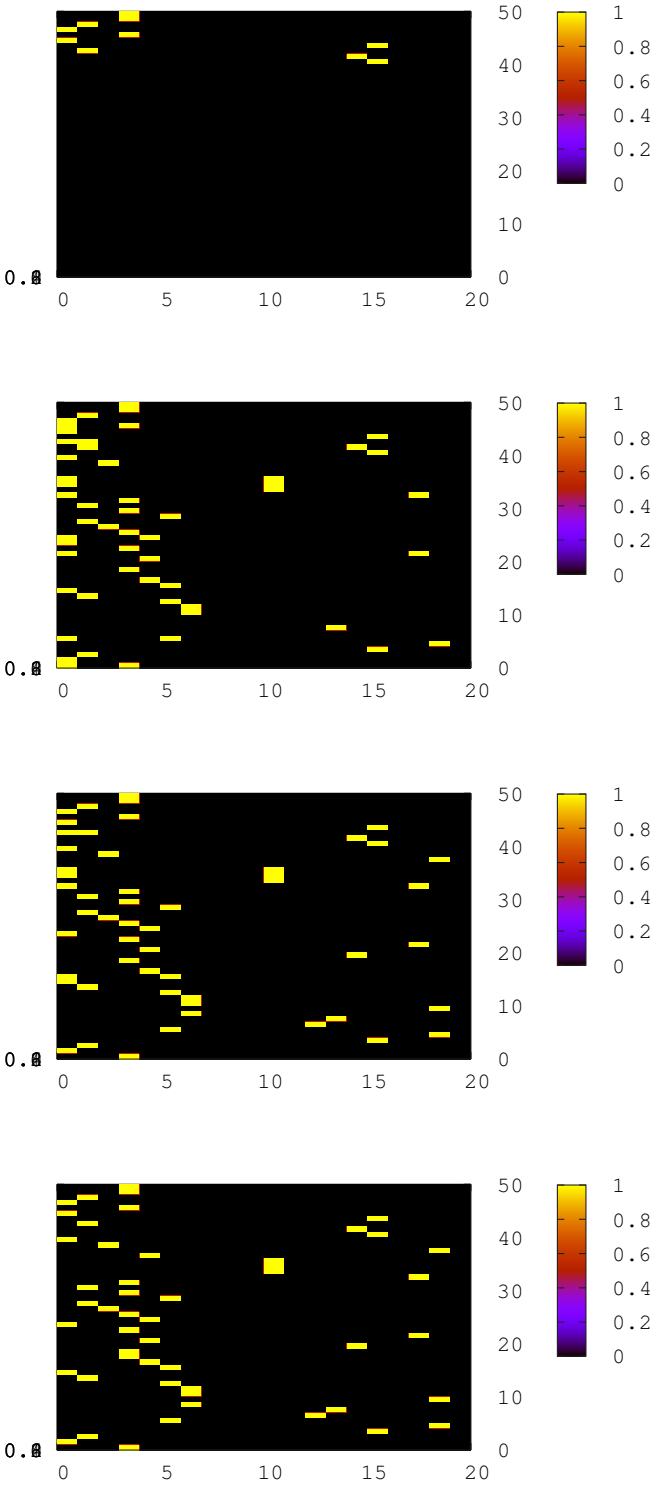


**図 3.4.** $\alpha = 10$. 左上から 0 回，2 回，4 回，30 回更新した様子

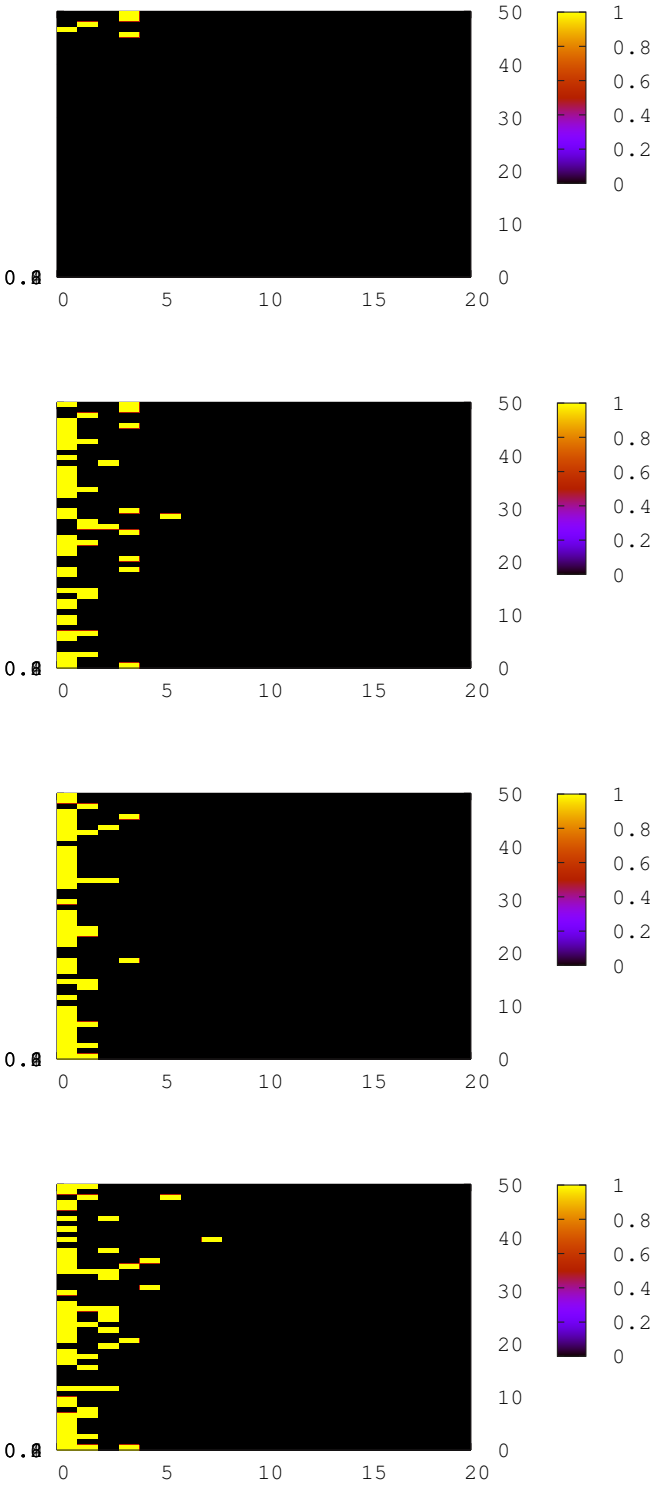


**図 3.5.** $\alpha = 5$. 左上から 0 回，2 回，4 回，30 回更新した様子

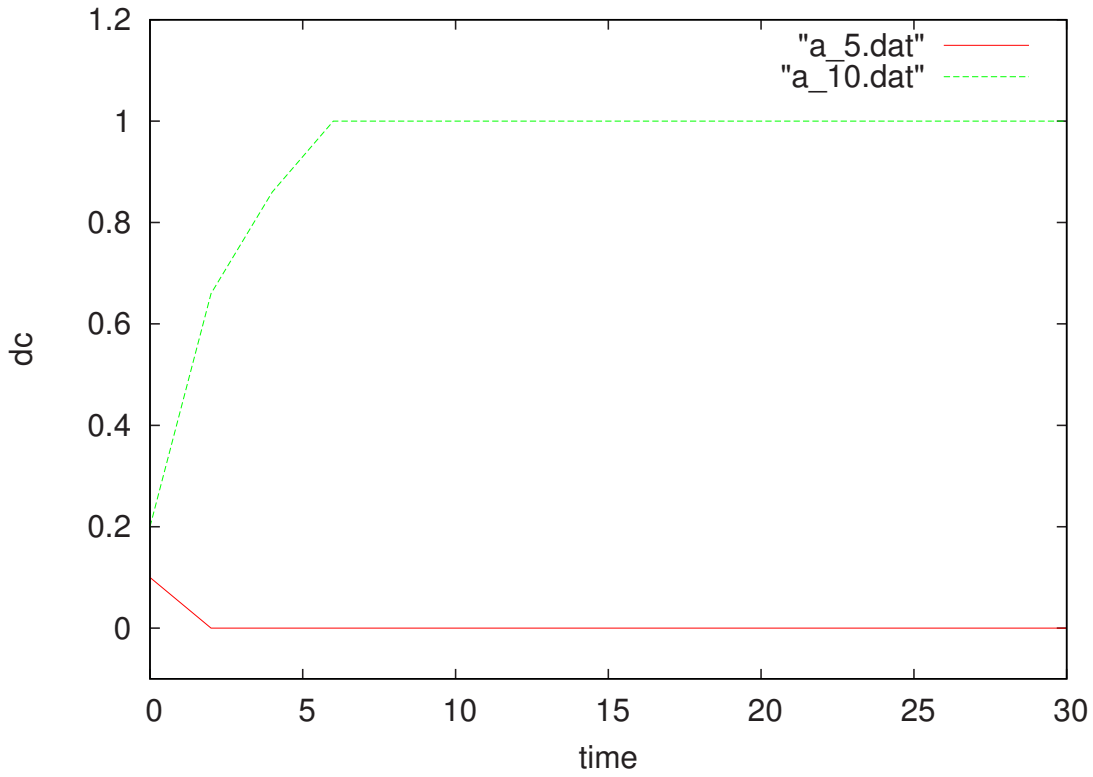


**図** 3.6. （従来型ダイナミクス）$\alpha = 10$，$\alpha = 5$ で想起した類似度

## 3.3　結合係数をコバリアンス型にする

従来型のダイナミクスを用いても，想起の成否は容易に判断できた．最後に，結合係数をコバリアンス型にして想起させた．結合係数は式 (3.2)，(3.3) に従って設定した [2].

$$T_{ij} = \frac{1}{n}\sum_{p=1}^{1000}(u_i^p - \overline{u}_i)(u_j^p - \overline{u}_j) \tag{3.2}$$

$$\overline{u}_i = \frac{1}{p}\sum_{p=1}^{1000} u_i^p \tag{3.3}$$

初期値 $\alpha = 10, 20$ で想起させた様子を図 (3.7)，(3.8) に示す．どちらも正しく想起できなかった．想起に失敗するときは，2008 モデルと同様に活動度の高いニューロンが左側に集中し，活動度が一定にならなかった.

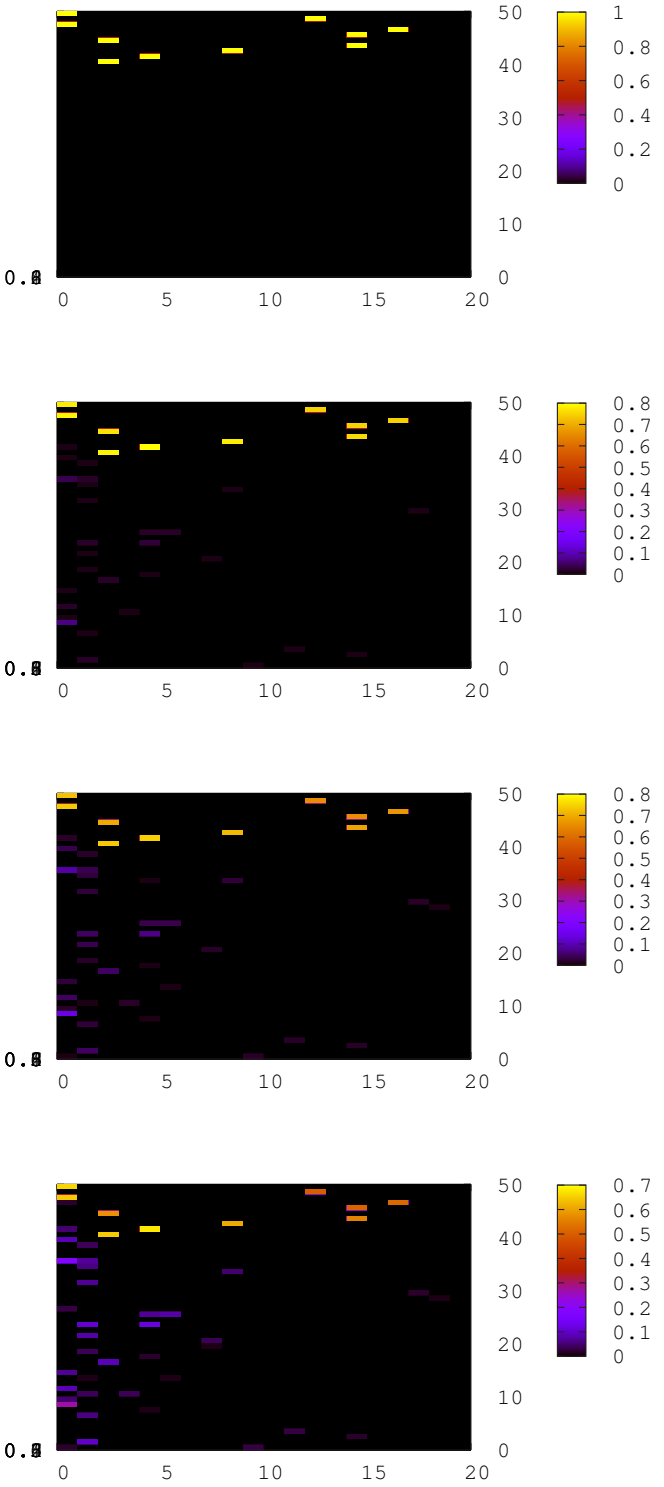


**図 3.7.** $\alpha = 10$. 左上から 0 回，600 回，1200 回，2400 回更新した様子

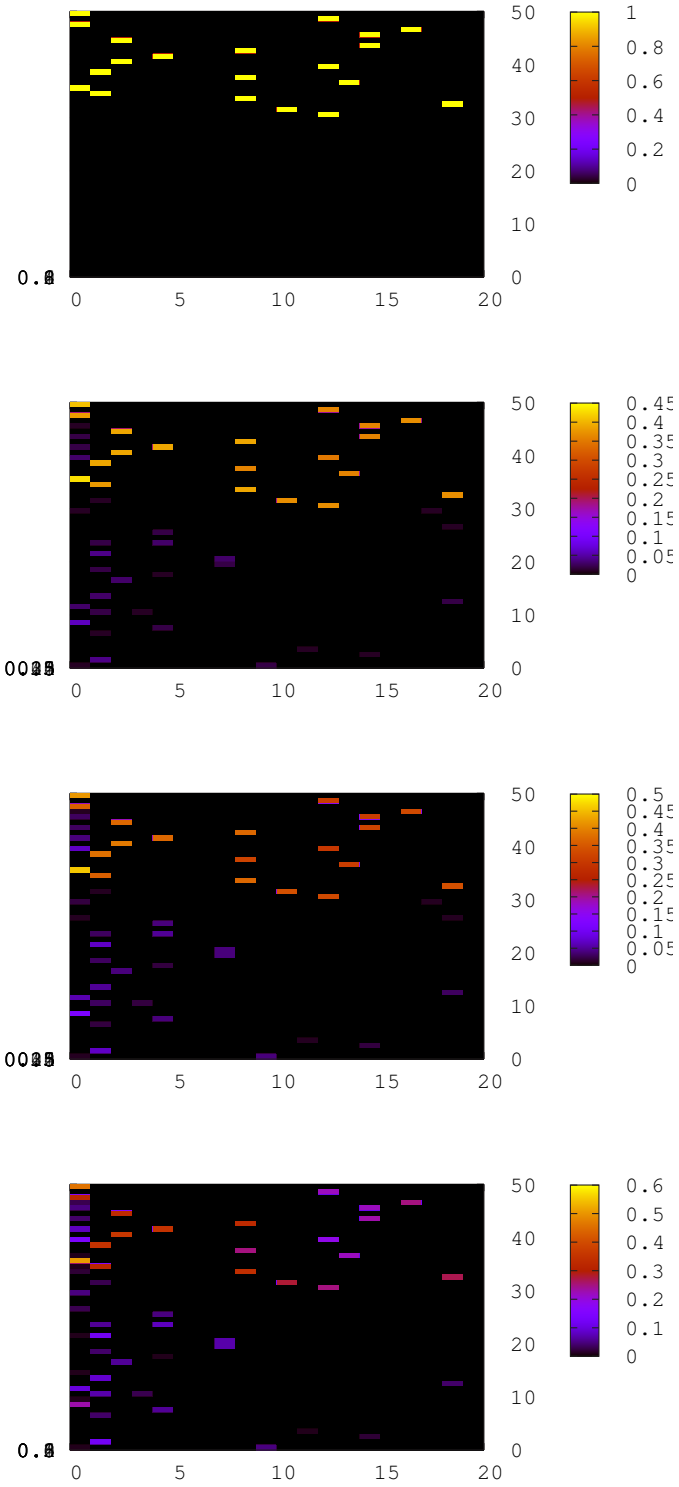


**図** 3.8. $\alpha = 20$. 左上から 0 回，600 回，1200 回，2400 回更新した様子

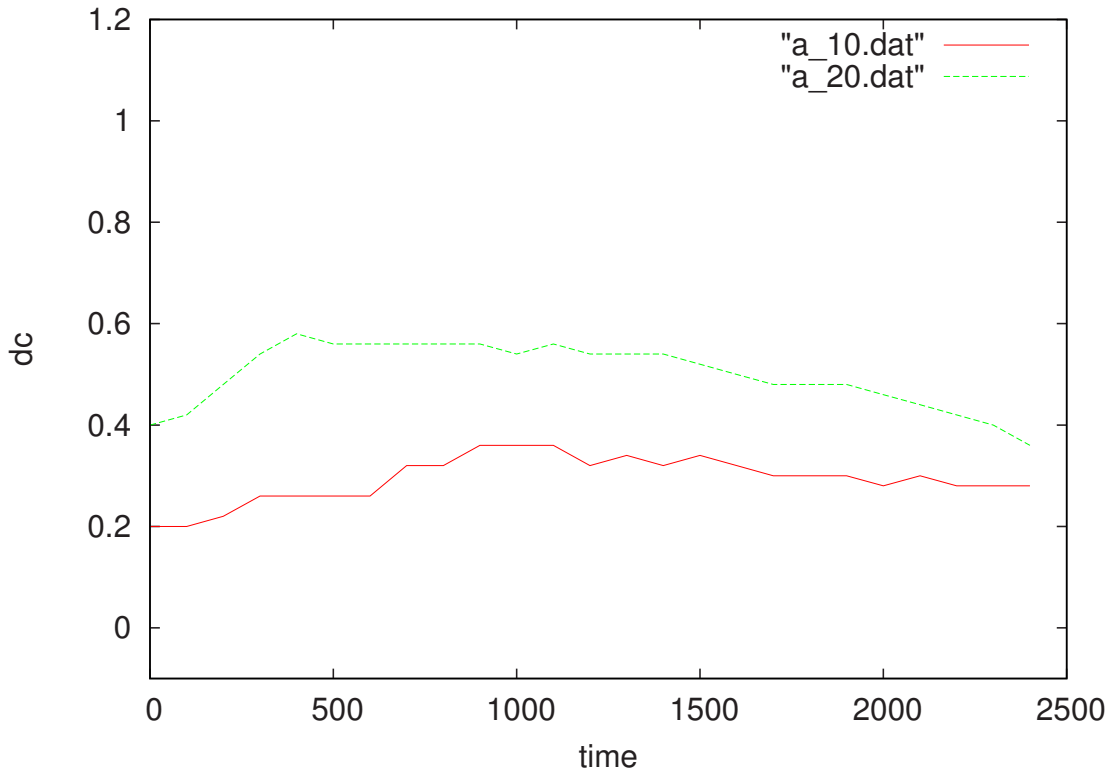


**図** 3.9. （コバリアンス型結合係数）$\alpha = 10$，$\alpha = 20$ で想起した類似度

# 第 4 章

# まとめ

今回の研究で，2 つのことを確認した．まず，活動度の違いで失敗と判断できることは，2008 モデルのダイナミクスが効いている．そして，想起に失敗するとき，活動度の高いニューロンが左側に集中することは，Zipf の法則による記憶パターン生成が効いているということである．さらに，今回，単純化した 2 値のダイナミクスで，2008 モデルと同様に，想起の成否を容易に判断できるモデルを実現できた．

# 謝辞

卒業研究に際して，情報システム工学科，伊達章准教授には多大な御指導を賜わりました．また，同研究室の皆様とも有意義な時間を過ごすことができました．

ここで，心から感謝申し上げたいと思います．ありがとうございました．

# 参考文献

[1] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences, U.S.A., vol.79, pp.2554-2558, 1982.

[2] 甘利 俊一，神経回路網モデルとコネクショニズム，東京大学出版会，1989.

[3] J. J. Hopfield. Searching for memories, Sudoku, implicit check-bits, and the iterative use of not-always-correct rapid neural computation. Neural Computation, vol. 20, pp. 1119-1164, 2008

# 付録 A

# 各モデルの想起精度

図 A.1 に，第 3 章の各実験で使用したモデルの想起精度を示す．初期値は $\alpha = 10$ で固定し，記憶させるパターン数を 0 から 600 まで増やす．各想起パターンにおいて，2008 モデルとコバリアンス型のモデルは 2400 回，従来型ダイナミクスのモデルは 30 回更新させた．更新後，類似度が 1.0 となったパターンのみをカウントした．縦軸に想起に成功した数，横軸に記憶させたパターン数を表す．シミュレーションに時間が掛かるため，記憶させるパターン数は 600 までとした．

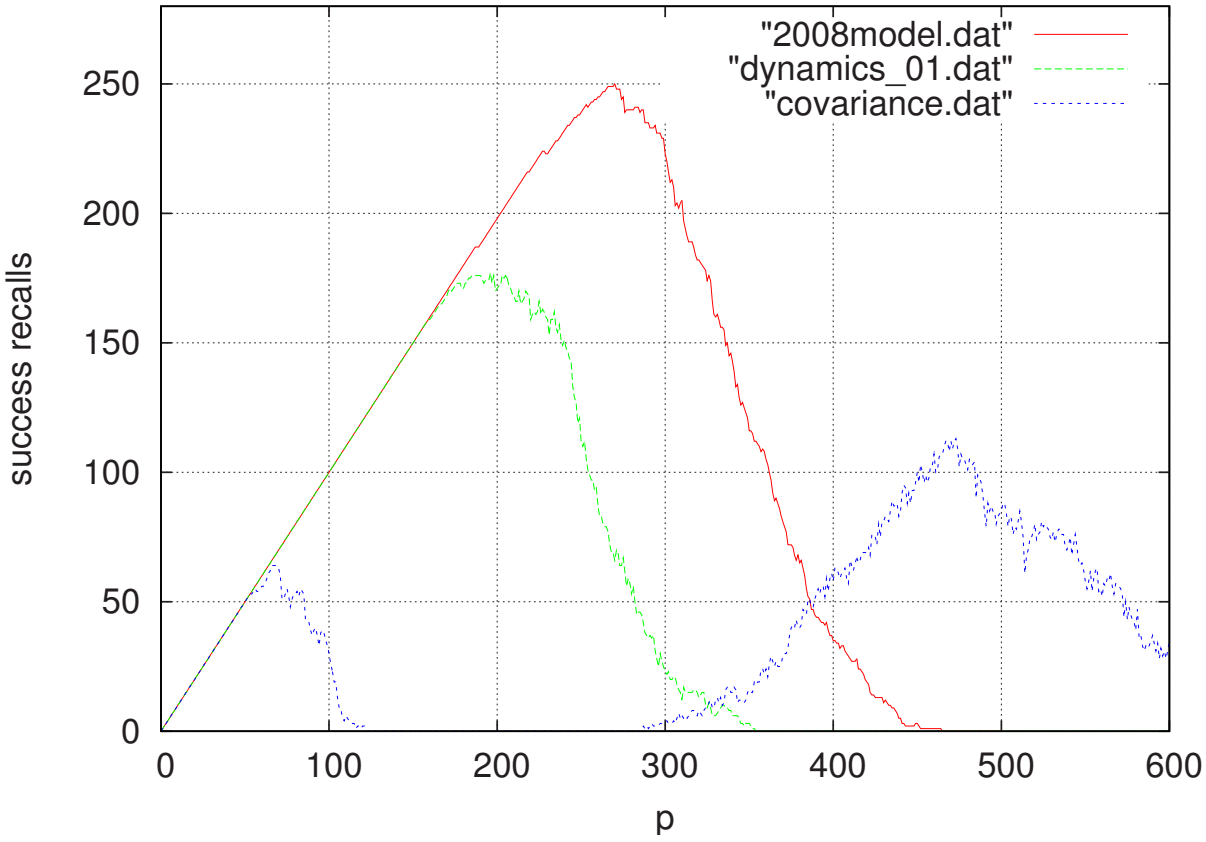


**図** A.1. $\alpha = 10$，記憶パターン数 0-600 での各モデルの想起の精度．赤：2008 モデル，緑：従来型ダイナミクス，青：コバリアンス型結合係数

# 付録 B

# ソースコード

```
1  #include<stdio.h>
2  #include<math.h>
3  #include<stdlib.h>
4  #include<time.h>
5
6  #define N_CATEGORY 50
7  #define N_PROPERTY 20
8  #define N_PATTERNS 800
9  #define N_NEURONS 1000 //ニューロンの数
10 #define DELTA_T 0.0015
11 int N_PAT;
12
13 void connect_reset(int m);
14 int compare( const void *c1, const void *c2 );
15 void sort_PV();
16 int count_ans();
17 int check_property1();
18 void check_ans();
19 int check_property(double t,double Z[]);
20 void make_memories();
21 void random_make_memories();
22 void copy_state(int p);
23 void connect_matrix();
24 void connect_matrix_analog();
25 void init_stat(int r);
26 void random_init_stat(double r);
27 void add_noise_init_stat(int c, double r);
28 void activity_dynamics();
29 void activity_dynamics2();
30 int check_all();
31 void update_UV();
32 void update_UV2();
33 double inner_product(int k);
34 int inner_product2(int k);
35 void write_data();
36 void write_data2();
37
38 FILE *gp,*gp1,*gp2;
39 FILE *fp1;
```

```
40
41 double T[N_NEURONS][N_NEURONS];
42 double V[N_NEURONS];
43 double PV[N_NEURONS];
44 double U[N_NEURONS];
45 double PU[N_NEURONS];
46 double PAT[N_PATTERNS][N_NEURONS];
47 double W[N_NEURONS];
48 double PW[N_NEURONS];
49 double Z[N_PROPERTY];
50
51
52 int main(int argc, char *argv[] ){
53
54   int i,j=0,a=0,cp,k;
55   long seed=12345678;
56
57   double r = 0.0; //ノイズの割合 random_init
58   int c = 20; //init_stat 正しい初期値の数
59   int n=0,p=0;
60   double h,t,dc,dc_p;
61   int dc2,chk;
62
63   srand48(seed);
64   gp=popen("gnuplot␣-geometry␣500x360","w");
65   //fprintf(gp, "set terminal postscript eps color \"times\" 20\n");//file
66   gp1=popen("gnuplot␣-geometry␣500x360+505+0","w");
67   fp1=fopen("a_.dat","w");
68
69   fprintf(gp,"set␣pm3d\n");
70   fprintf(gp,"set␣view␣0,0\n");
71   fprintf(gp,"unset␣key\n");
72
73   fprintf(gp1,"set␣pm3d\n");
74   fprintf(gp1,"set␣view␣0,0\n");
75   fprintf(gp1,"unset␣key\n");
76
77   /* 記憶パターンの生成 make_memories*/
78   make_memories();//Zipf
79   //random_make_memories();//Random
80
81   /* パターン数の変更 */
82   for(N_PAT=225; N_PAT<701; N_PAT+=10){
83     printf("Patterns:%d␣\n",N_PAT);
84
85     /* 2 結合行列の設定 T[][]*/
86     //connect_matrix();//結合係数hopfield2008
87     connect_matrix_analog();//結合係数 Covariance 型
88
89     for(k = 0; k < N_PAT; k++){
90       //for(k = 1; k < 5; k++){
91       //p = (int)(drand48() * N_PAT);//想起パターンランダム
92       p = k; //想起パターン固定
93       copy_state(p);
```

```
94
95      /* 3 回路に初期値を与える。init_stat(int r) r:正しい情報*/
96      add_noise_init_stat(c * 20 , r);
97
98      /* 4 回路の状態更新 */
99      for(i=0; i<2401; i++){
100
101       //gnuplot
102       if(i%600 == 0){
103         //fprintf(gp,"set out 'covari_%d-up_%d.eps'\n",N_PAT,i);//file
104         //fprintf(gp, "splot '-' with pm3d\n");//file
105         //monitor
106         fprintf(gp,"set␣title␣\"Pattern:%d update=%d dc1=%.2lf n=%d \"\n",p,i,dc,
                a);//
107         fprintf(gp, "splot␣'-'␣with␣pm3d\n");//
108
109         dc = inner_product(p);
110         dc2 = inner_product2(p);
111         dc_p = (double) dc2 / N_CATEGORY;
112         if(dc==1.0){j++;}
113         a = check_all();
114         fprintf(gp1,"set␣title␣\"Check_Ans correct=%d \"\n",dc2);
115         fprintf(gp1, "splot␣'-'␣with␣pm3d\n");
116         write_data();
117         check_ans();
118         write_data2();
119         check_ans();
120         printf("Init:%d␣\t␣Update%d:dc=%.2lf␣ActiveN:%d␣Cor:%d\n",c,i,dc,a,dc2);
121       }
122       fprintf(gp,"e\n");fprintf(gp1,"e\n");//sim
123       fprintf(fp1,"%d␣%.2lf␣\n",i,dc);fflush(fp1);
124       fflush(gp);fflush(gp1);
125       if(j == 30){goto end;chk++;}
126       //if(dc < 0.9){goto end;}
127       activity_dynamics();update_UV();//Hopfield2008ダイナミクス
128       //activity_dynamics2();sort_PV();update_UV2(); //従来型ダイナミクス
129     }
130   end: j=0;
131     if(dc > 0.9){chk++;}
132     printf("\n");//simulation
133   }
134   printf("PAT:%d--␣%d␣\n",N_PAT,chk);
135   chk = 0;
136   printf("\n");
137 }
138 fprintf(fp1,"\n");
139 fprintf(stderr, "check-END\n");
140 fclose(fp1);
141 fclose(gp);fclose(gp1);
142 return 0;
143 }
144
145
146 void connect_reset(int m){
```

```
147   int i,j,k;
148
149   i = N_PAT;
150   for(k=0; k<m; k++){
151     for(j=0; j<N_NEURONS; j++){
152       T[i][j] = 0.0;
153       T[j][i] = 0.0;
154     }
155     i++;
156   }
157 }
158
159
160 void sort_PV(){
161   int i,a;
162   double tmp;
163
164   qsort( V, N_NEURONS, sizeof(double), compare );
165   a = 0;
166   for(i=0; i<N_NEURONS; i++){
167     if(U[i] < V[N_CATEGORY - 1 ]){
168       U[i] = 0.0;
169     }
170   }
171 }
172
173
174 int compare( const void *c1, const void *c2 )
175 {
176   double tmp1 = *(double *)c1;
177   double tmp2 = *(double *)c2;
178
179   if( tmp1 == tmp2 ) return 0;
180   if( tmp1 > tmp2 ) return -1;
181   if( tmp1 < tmp2 ) return 1;
182 }
183
184
185 int check_property1(){
186   int i,k;
187
188   k = 0;
189   for(i=0;i<N_NEURONS;i+=20){
190     if(PU[i] > 0.0){k++;}
191   }
192   return k;
193 }
194
195
196 void check_ans(){
197   int i;
198
199   for(i=0;i<N_NEURONS;i++){
200     PW[i] = 0.0;
```

```
  }

  for(i=0;i<N_NEURONS;i++){
    PW[i] = W[i] + PU[i];
    if(PU[i]==1.0 && W[i]==0.0){PW[i]=0.5;}
    if(PW[i] > 1.01)
      {
        PW[i] = 2.0;
      }
    else if(PW[i] > 0.01 && PW[i] < 1.0)
      {
        PW[i] = 0.5;
      }
  }
}


void activity_dynamics(){

  int i,j;
  double h,sum=0.0;
  double ans=0.0;

  for(i=0; i<N_NEURONS; i++){
    ans = PV[i] + ans;
  }

  h = -30.0 + (3.5 * ans);
  if(h < 0){
    h = 0.0;
  }

  for(i=0; i<N_NEURONS; i++){
    sum = 0.0;
    for(j=0; j<N_NEURONS; j++){
      sum = T[i][j] * PV[j] + sum;
    }
    V[i] = PV[i] + DELTA_T * ((-PV[i] / 100.0) + sum - h);
    if(V[i] > 0.0){
      U[i] = V[i];
    }
    else{U[i] = 0.0;
    }
  }
}


//従来型ダイナミクス
void activity_dynamics2(){

  int i,j;
  double h,sum=0.0;

  for(i=0; i<N_NEURONS; i++){
```

```
255     sum = 0.0;
256     for(j=0; j<N_NEURONS; j++){
257       sum = T[i][j] * PV[j] + sum;
258     }
259     if(sum > 0.0){
260       V[i] = sum;
261     }
262     else{
263       V[i] = 0.0;
264     }
265     U[i] = V[i];
266   }
267 }
268
269
270 void connect_matrix(){
271
272   int i,j,k;
273
274   for(i=0; i<N_NEURONS; i++){
275     for(j=0; j<N_NEURONS; j++){
276       T[i][j] = 0.0;
277     }
278   }
279
280   for(i=0; i<N_NEURONS; i++){
281     T[i][i] = 0.0;
282     for(j=i+1; j<N_NEURONS; j++){
283       for(k=0; k<N_PAT; k++){
284         if(PAT[k][i] == 1.0 && PAT[k][j] == 1.0){
285           T[i][j] = 1.0;
286           T[j][i] = 1.0;
287         }
288       }
289     }
290   }
291 }
292
293
294 //結合係数 Covariance
295 void connect_matrix_analog(){
296
297   int i,j,k;
298   double sum=0.0,sum2=0.0;
299   double PAT_d[N_NEURONS];
300
301
302   for(i=0; i<N_NEURONS; i++){
303     for(k=0; k<N_PAT; k++){
304       sum2 = PAT[k][i] + sum2;
305     }
306     PAT_d[i] = sum2 / N_PATTERNS;
307     sum2 = 0.0;
308   }
```

```
309
310   for(i=0; i<N_NEURONS; i++){
311     T[i][i] = 0.0;
312     for(j=i+1; j<N_NEURONS; j++){
313       for(k=0; k<N_PAT; k++){
314         sum = (PAT[k][i] - (PAT_d[i])) * (PAT[k][j] - (PAT_d[j])) + sum;
315       }
316       T[i][j] = sum / N_NEURONS;
317       T[j][i] = sum / N_NEURONS;
318        sum = 0.0;
319     }
320   }
321 }
322
323
324 void init_stat(int r){
325
326   int i,j=0,t;
327
328   for(i=r; i<N_NEURONS; i++){
329     PV[i] = 0.0;
330     PU[i] = 0.0;
331   }
332 }
333
334
335 void add_noise_init_stat(int c, double r){
336
337   int i;
338   double d;
339
340   for(i=c; i<N_NEURONS; i++){
341     PV[i] = 0.0;
342     PU[i] = 0.0;
343   }
344
345   for(i=0; i<N_NEURONS; i++){
346     d = drand48();
347     if(d < r){
348       PV[i] = 1.0;
349       PU[i] = 1.0;
350     }
351   }
352 }
353
354
355 void random_init_stat(double r){
356
357   int i,j,k,a;
358   double d;
359
360   a=0;
361   k=0;
362   for(j=0; j<N_CATEGORY; j++){
```

```
    d = drand48();
    if(d < r){a++;
    if(a < 49){
      for(i=k; i < k+20; i++){
        PV[i] = 0.0;
        PU[i] = 0.0;
      }
    }

    } k = k+20;
  }
}


void update_UV(){
  int i;

  for(i=0; i<N_NEURONS; i++){
    PU[i] = U[i];
    PV[i] = U[i];
  }
}


void update_UV2(){
  int i;

  for(i=0; i<N_NEURONS; i++){
    if(U[i] > 0.0){
      U[i] = 1.0;
    }
    PU[i] = U[i];
    PV[i] = U[i];
  }
}


int check_all(){

  int i,a=0;

  for(i=0; i<N_NEURONS; i++){
    if(PU[i] > 0.01){a++;}
  }
  return a;
}


void random_make_memories(){
  int i,j,k,t,n;
  double d;
  double sum;

  Z[0] = 0.05;
```

```
  for(i=1; i<N_PROPERTY; i++){
    Z[i] = Z[i-1] + 0.05;
  }

  for(n=0; n<N_PATTERNS; n++){
    t=0;
    for(j=0; j<N_NEURONS; j++){
      PAT[n][j] = 0.0;
    }

    for(j=0; j<N_CATEGORY; j++){
      k = check_property( drand48(), Z );
      PAT[n][k + t] = 1.0;
      t = t + N_PROPERTY;
    }
  }

}


void make_memories(){

  int i,j,k,t,n;
  double d;
  double sum;

    sum = 0.0;
    for(i=0; i<N_PROPERTY; i++){
      sum += 1.0 / sqrt( (double)(i+1) );
    }

    Z[0]=1.0 / sum;
    for(i=1; i<N_PROPERTY; i++){
      Z[i] = 1.0 / ( sum*sqrt((double)(i+1) )) + Z[i-1];
    }

  for(n=0; n<N_PATTERNS; n++){
    t=0;
    for(j=0; j<N_NEURONS; j++){
      PAT[n][j] = 0.0;
    }

    for(j=0; j<N_CATEGORY; j++){
      k = check_property( drand48(), Z );
      PAT[n][k + t] = 1.0;
      t = t + N_PROPERTY;
    }
  }
}


int check_property(double t,double Z[]){

  int i,a=0;
```

```
471
472   if(t <= Z[0]){
473     a=0;
474   }
475   for(i=0; i<N_PROPERTY-1; i++){
476     if(Z[i] < t && t <= Z[i+1]){
477       a=i+1;
478     }
479   }
480   return (int) a;
481 }
482
483
484 void copy_state(int p){
485
486   int i,j;
487   double k;
488
489   for(j=0; j<N_NEURONS; j++){
490     if(PAT[p][j] == 1.0){
491     PV[j] = 1.0;
492     PU[j] = 1.0;
493     W[j] = 1.0;
494     }else{
495       PV[j] = 0.0;
496       PU[j] = 0.0;
497       W[j] = 0.0;
498     }
499   }
500 }
501
502
503 double inner_product(int k){
504
505   int i,j;
506   int c=0;
507   for(i=0; i<N_NEURONS; i++){
508     if((PU[i] > 0.01 && PAT[k][i] == 1.0)){
509       c++;
510     }
511     else if((PU[i] > 0.01 && PAT[k][i]==0.0 )){
512       c--;
513     }
514   }
515   if(c < 0){c=0;}
516   return ((double)c/(double)N_CATEGORY);
517 }
518
519
520 int inner_product2(int k){
521   int i;
522   int c=0;
523
524   for(i=0; i<N_NEURONS; i++){
```

```
525     if((PU[i] > 0.01 && PAT[k][i] == 1.0)){
526      c++;
527     }
528   }
529   return c;
530 }
531
532
533 void write_data(){
534
535   int i,j;
536
537   for(j=0; j<N_PROPERTY; j++){
538     for(i=0; i<N_CATEGORY; i++){
539       fprintf(gp,"%d␣␣%d␣%.2lf\n",j,i,PU[j+(N_CATEGORY-i-1)*N_PROPERTY]);
540       fprintf(gp,"%d␣␣%d␣%.2lf\n",j,i+1,PU[j+(N_CATEGORY-i-1)*N_PROPERTY]);
541     }
542     fprintf(gp,"\n");
543
544     for(i=0; i<N_CATEGORY; i++){
545       fprintf(gp,"%d␣␣%d␣%.2lf\n",j+1,i,PU[j+(N_CATEGORY-i-1)*N_PROPERTY]);
546       fprintf(gp,"%d␣␣%d␣%.2lf\n",j+1,i+1,PU[j+(N_CATEGORY-i-1)*N_PROPERTY]);
547     }
548     fprintf(gp,"\n");
549   }
550 }
551
552
553 void write_data2(){
554
555   int i,j;
556
557   for(j=0; j<N_PROPERTY; j++){
558     for(i=0; i<N_CATEGORY; i++){
559       fprintf(gp1,"%d␣␣%d␣%.2lf\n",j,i,PW[j+(N_CATEGORY-i-1)*N_PROPERTY]);
560       fprintf(gp1,"%d␣␣%d␣%.2lf\n",j,i+1,PW[j+(N_CATEGORY-i-1)*N_PROPERTY]);
561     }
562     fprintf(gp1,"\n");
563
564     for(i=0; i<N_CATEGORY; i++){
565       fprintf(gp1,"%d␣␣%d␣%.2lf\n",j+1,i,PW[j+(N_CATEGORY-i-1)*N_PROPERTY]);
566       fprintf(gp1,"%d␣␣%d␣%.2lf\n",j+1,i+1,PW[j+(N_CATEGORY-i-1)*N_PROPERTY]);
567     }
568     fprintf(gp1,"\n");
569   }
570 }
```