卒業論文

要素和エントロピー最小化原理により学習する神経回路モデルの研究

76050460　馬場 一暢

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

第 2 章　要素和エントロピー最小化原理 2
2.1　神経回路モデルの構造 2
2.2　入出力信号の情報量とエントロピー 3
2.3　要素和エントロピー最小化 3
2.4　学習アルゴリズム 4

第 3 章　計算機実験 6
3.1　2 次元のデータを入力とした場合 6
3.2　64 次元の画像データを入力とした場合 9

第 4 章　まとめ 12

謝辞 13

参考文献 14

付録 A　プログラムリスト 15

第 1 章

はじめに

ものを見ているときの情報は，脳内ではどの様に表現しているのだろうか．目から入った情報は、網膜に投影され，そこから視覚野と呼ばれる大脳の部位に伝達される．視覚野は，網膜に近い方から第一視覚野，第二視覚野と多重階層的に情報伝達場があり，網膜に映ったものの一部から情報を読み取り，特徴を抽出し処理して，次の視覚野に情報を伝えていき，最終的にどの様なものを見たのかを判断する．このような仕組みを情報表現の可能性を示す一つの方法として，神経回路モデルを使った研究がある．本研究では，自己組織化の回路を考える．自己組織化とは，外界からの入力情報に応じた情報表現を自動的に獲得する能力を持つもので，自己組織化の情報表現の仕方には二つの特徴があり，局所表現と分散表現がある．局所表現とは，一つの概念を脳内に表現するとき，一つの細胞を使う．つまり，例えば犬や猫といった概念に対し，それに対応した犬細胞，猫細胞が作られ，犬を認識したとき，犬細胞が興奮する．このような場合の認識結果は，特定細胞の興奮，つまり局所化した興奮パターンが表示される．これに対し，分散表現とは，一つの概念は脳内のある領野の神経興奮パターンとして表現する．犬を認識したとしたら犬興奮パターンが，猫を認識したら猫興奮パターンが生じる．なので先ほどと違い，一つのニューロンには，いくつもの概念に関与するので，一つのニューロンの興奮を見ただけでは，なにを認識したのかが分からない．脳内では認識細胞を作ることで情報を局所化，単純化すると同時に，こうした表現をいろいろな視点で行い，その結果を組み合わせて分散表現することを繰り返しているかもしれない．記憶や思考には分散表現をすることの利点が多い．それは，ダイナミックスを通して演算や連想ができるからである．一方一つのシステムからもう一つのシステムへ情報を転送するときは，ある程度局所化した方が都合がいい．これをうまく組み合わせたものとしてスパース表現というものもある．

本研究では，特に，要素和エントロピー最小化原理により学習を行う自己組織化の回路についての実験を行い，神経回路モデルの性能の評価をした．

第 2 章

要素和エントロピー最小化原理

2.1 神経回路モデルの構造

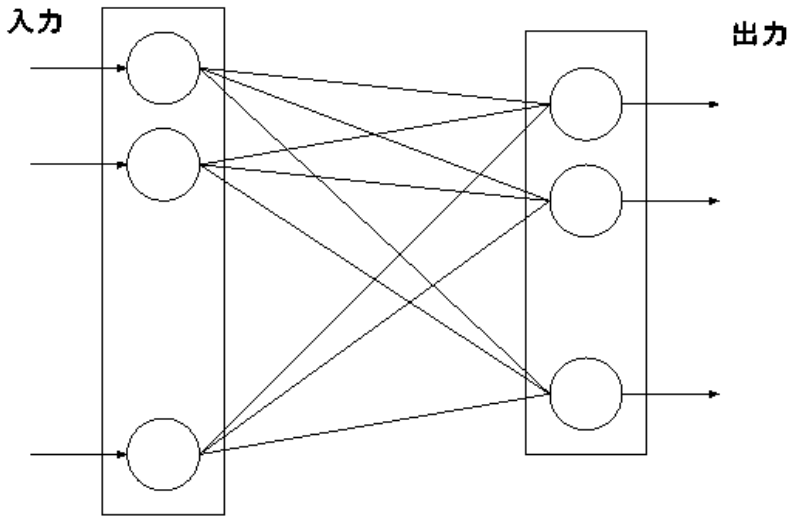

図 2.1. 神経回路モデル

外界からの入力としてある $\vec{x}$ が次々に観測され，これを層状の神経回路モデルで情報処理をすることを考える．m 個の信号 $\vec{x} = (x_1, x_2, ..., x_m)$ を受け取り，n 個の信号 $\vec{a} = (a_1, a_2, ..., a_n)$ を出力する回路を考える．典型的な例では，入力 $\vec{x}$ の次元 m は画像のように大きく例えば 1000 次元，出力 $\vec{y}$ の次元 n は 10 次元のように外界の次元が高い場合，言い換えれば $m >> n$ の状況を考える．通常，外界の情報は高次元であるが，実際に詰め込まれている情報は少ない．目的は入力 $\vec{x}$ の規則性を反映した表現 $\vec{a}$ を作り，かつ，高次元の入力を情報の損失なく低次元で表現することである．

2.2 入出力信号の情報量とエントロピー

今，神経回路が入力 $\vec{x}$ を受け取り，$\vec{a}$ を出力したとする．入力信号 $\vec{x}$ のエントロピーを $H(\vec{x})$，出力信号 $\vec{a}$ のエントロピーを $H(\vec{a})$ と書こう．このとき，入力信号 $\vec{x}$ のエントロピー $H(\vec{x})$ と出力信号 $\vec{a}$ のエントロピーを $H(\vec{a})$ から $\vec{x}$ と $\vec{a}$ の相互情報量を

$$I(\vec{a};\vec{x}) = H(\vec{a}) - H(\vec{a}|\vec{x}) \tag{2.1}$$

と書く．入力信号の情報が出力に損失なく伝わっている状況を考えよう．その場合，出力 $\vec{a}$ について不確実な点は一切なくなる．このため条件付きエントロピーは $H(\vec{a}|\vec{x}) = 0$ であり

$$I(\vec{a};\vec{x}) = H(\vec{a}) \tag{2.2}$$

となる．つまり，出力のエントロピーを最大化すれば，変換の効率を最大化でき，出力が入力からできるだけ多くの情報を受け取ることができる事を意味している．

出力のエントロピー $H(\vec{a})$ は各出力素子の値 a_i のエントロピー $H(a_i)$ の総和からすべての $i, j, i \neq j$ について a_i, a_j 間の相互情報量を指し引いたものであり

$$H(\vec{a}) = \sum_{i=1}^{n} H(a_i) - \sum_{i=1}^{n} I(a_i; a_{i-1}, \ldots, a_1) \tag{2.3}$$

と書ける．もし入力情報が損失なく出力に伝わっているなら冗長性はゼロにでき，

$$H(a) = \sum_{i=1}^{n} H(a_i) \tag{2.4}$$

となり，これより $H(\vec{a})$ を小さくすることはできない．これに従い，$\vec{a}$ の確率分布を $p(\vec{a})$ と書くと，同時分布は

$$p(\vec{a}) = \prod_{i=1}^{n} p(a_i) \tag{2.5}$$

と各要素の確率分布の掛け算で書け，各 a_i のとる値が独立であることを意味する．

2.3 要素和エントロピー最小化

要素和エントロピー最小化とは，教師なし学習の自己組織化モデルで，出力素子のとる値のエントロピーを測り，そのエントロピーを最小にすることである．以上の事をまとめて図で表したものが，図 2.2 である．重要なのは，すべての出力のエントロピーの $H(a)$ の処理であり，個々のエントロピーの総和（$\sum_{i=1}^{n} H(a_i)$）がそれぞれに最小化するとき，出力間の冗長性も最小化された．以上より，2 つの一般的な符号化に関する目標をに従うことにした．

1. 出力と入力間の相互情報量の最大化する．
2. 1 の条件を達成したら，個々の出力のエントロピーの総和を最小化する．

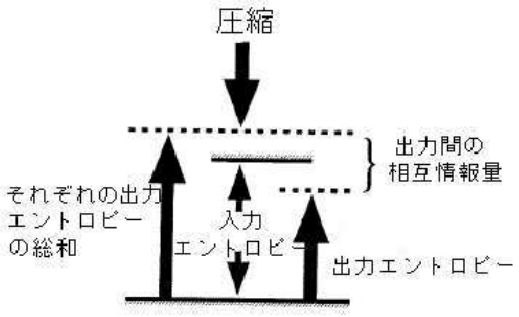

図 2.2. 情報処理過程におけるエントロピーのそれぞれの値

2.4　学習アルゴリズム

ここで用いるネットワークは，図 2.3 のようなモデルである．m 個の $\vec{x} = (x_1, x_2, ..., x_m)$

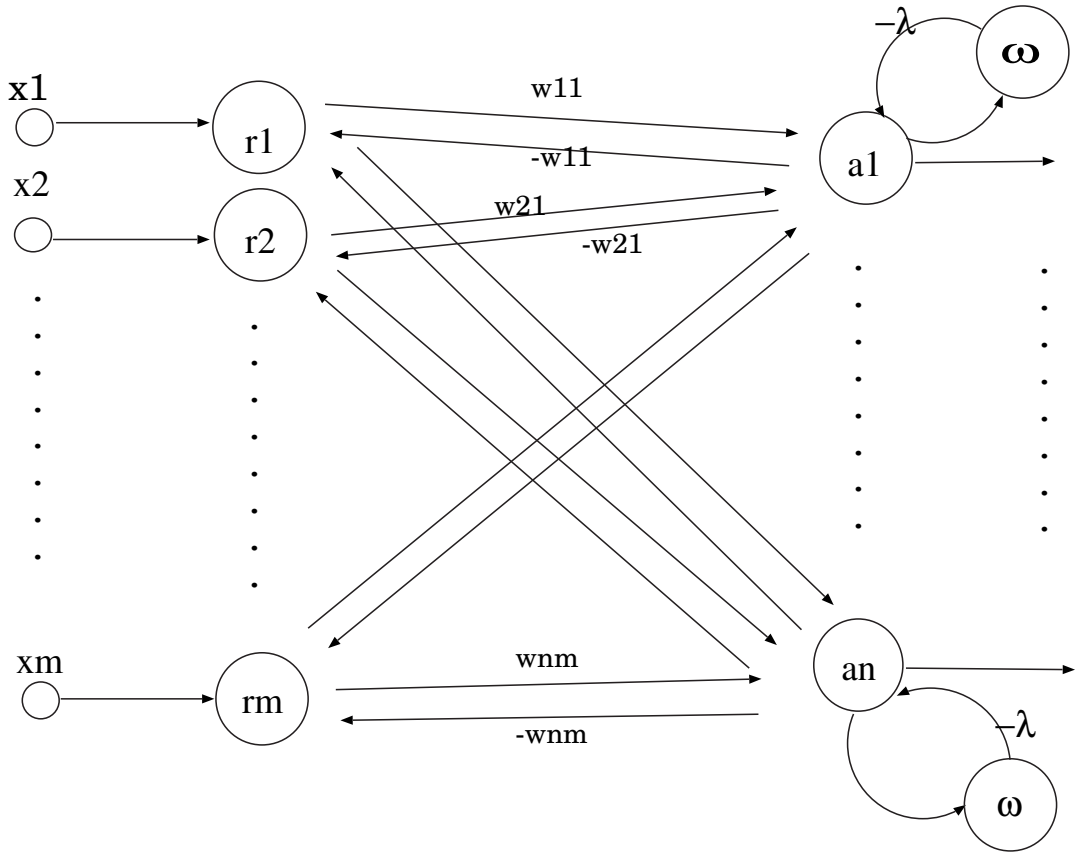

図 2.3. 神経回路モデル

が，外界から入力されたとする．まず，入力 $\vec{x}$ は，それぞれ $\vec{r} = (r_1, r_2, ..., r_m)^{\mathrm{T}}$ という素子に入力される．そのあと，すべての $\vec{r}$ と $\vec{w}$ という重み付けの総和によって，出力素子 $\vec{a}$ を求

められる．それと同時に，出力 $\vec{a}$ は，$-\vec{w}$ という重み付けで $\vec{r}$ にフィードバックする．この事を式で書くと (2.6) 式のように書ける．

$$\vec{r} = \vec{x} - \sum_{k=1}^{h} a_k \vec{w}_k \tag{2.6}$$

以上のことを繰り返し，$\vec{w}$ を学習させていく．学習回数を重ねていくと $\vec{r}$ と $\vec{a}$ が，ある一定の値に決定する．この時に少しだけ $\vec{w}$ 学習させる．$\vec{w}$ の学習を表す式は，参照論文 [2] より以下の様に書く．

$$\triangle \vec{w}_i = \eta a_i \vec{r} \tag{2.7}$$

学習をすることで，適切な $\vec{w}$ をだし，入力が反映されたような出力を導いてゆく．以上のことを、前節のエントロピーの話を考えた上で式に書くと

$$E = ||\vec{r}||^2 + \lambda \sum_{i=1}^{n} \omega(a_i) \tag{2.8}$$

と書くことができる．最初の項は，$\vec{r}$ と $\vec{a}$ の誤差について計算している．次項の，入力 $\vec{x}$ と出力 $\vec{a}$ の間の誤差を調節し，疎性を得るための制御関数である $\omega(a)$ は，制御関数としてよく使われる式から選ぶこととした．制御関数には a^2 や $log(1+a^2)$，$\sqrt{|a|}$ などがあるが，本研究では，計算する際も扱いやすい $\omega(a) = |a|$ を制御関数として使用した．この制御関数によって，

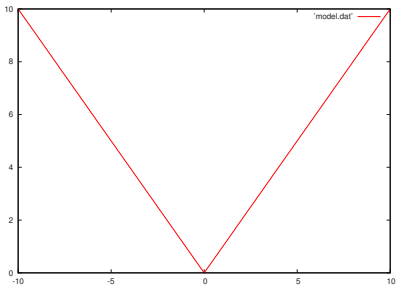

図 2.4. 制御関数：$\omega(a) = |a|$

いかに少ない出力で入力情報を情報の損失なく作ることができるかを調節することができる．そして上記計算によって E を最小化することで，要素和エントロピーの最小化を実現する．

第 3 章

計算機実験

3.1　2 次元のデータを入力とした場合

先ほどのアルゴリズムを評価するために，問題を考える．入力 $\vec{x} = (x_1, x_2)$，出力 $\vec{y} = (y_1, y_2)$ の 2 次元のデータとする．y_1, y_2 はそれぞれ独立の値である．今，この y_1, y_2 の 2 つの値が混ざり合い，それぞれの値が分からないとき，値の分かっている入力 $\vec{x} = (x_1, x_2)$ からそれぞれのもとの y_1, y_2 がどの様な値だったかを再現することができるかどうかを考える．基底関数 $\vec{w}$ と y_1, y_2 の傾きを見ることで，y_1, y_2 が再現できているかを確認する．本研究では，ワイブル分布で乱数を作り，そこから $\vec{y}$ を生成した．

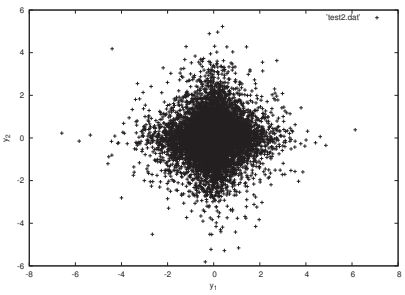

図 3.1. 混じりあった y_1, y_2 の値

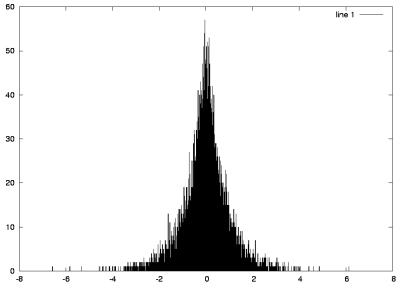

図 3.2. ワイブル分布

ワイブル関数は，

$$p(y) = \alpha^{-\beta}\beta|y|^{(\beta-1)}exp\left[-\left(\frac{|y|}{\alpha}\right)^{\beta}\right] \tag{3.1}$$

(3.1) 式で表される関数で，参照論文 [2] より，それぞれ $\alpha = 1$，$\beta = 2/3$ と設定した．入力 $\vec{x}$ は，

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \sigma_1 \cos\theta_1 & \sigma_2 \cos\theta_2 \\ \sigma_1 \sin\theta_1 & \sigma_2 \sin\theta_2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \tag{3.2}$$

で表す関数を使い，ここで使用する係数は以下に設定した．

σ_1	σ_2	θ_1	θ_2
1.0	2.0	16°	60°

学習に必要な係数は $\lambda = 0.1$，$\mu = 0.00001$，$\eta = 0.005$ とし，500 回毎に 1 回学習する方法で 80 回学習を繰り返した．このときの結果が図 3.3 である．

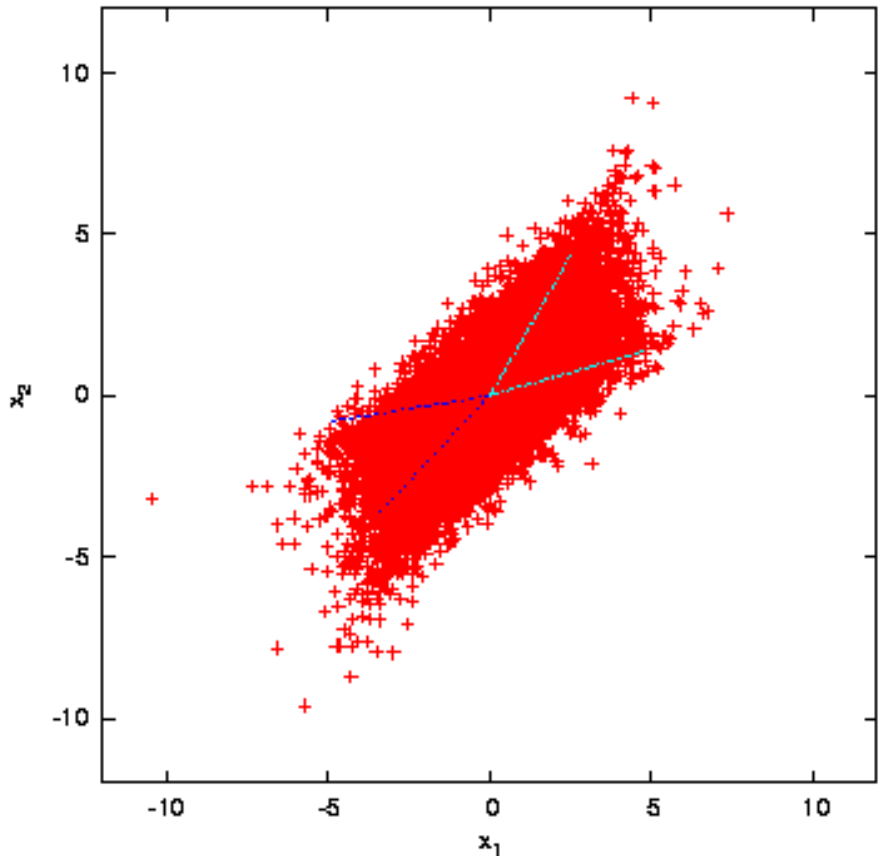

図 3.3. 入力 $\vec{x}$：赤い点，基底：青線，$\vec{y}$ の傾き（回転角）：水色の線

図 3.3 は，入力と学習後の基底関数を表している．図の青線は基底関数，水色の線は y_1, y_2 の傾きを表しており，赤い点の集合は入力を表している．入力の点の広がりの尖っている方向と，$\vec{x}$ を作成したときの y_1, y_2 の回転角を表す水色の線の向きが，基底関数の青い点でできた線分の逆方向とほぼ同じであることを図 3.3 より観測することができた．この事より，学習アルゴリズムから導かれた基底関数から出力が再構成されていることが分かった．

次に追実験として，$\vec{y} = (y_1, y_2, y_3)$ の 3 つの独立した値が 2 次元上で混ざりあい，それぞれの値が分からない場合に，値の分かっている値 x_1, x_2 から y_1, y_2, y_3 を表現することができるか調べた．学習係数と学習回数は先ほどと同じとし，入力 $\vec{x}$ を

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \sigma_1 \cos\theta_1 & \sigma_2 \cos\theta_2 & \sigma_3 \cos\theta_3 \\ \sigma_1 \sin\theta_1 & \sigma_2 \sin\theta_2 & \sigma_3 \sin\theta_3 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \tag{3.3}$$

のようにして，入力で使う係数を以下に設定しなおした．

σ_1	σ_2	σ_3	θ_1	θ_2	θ_3
0.7	2.0	0.2	18°	60°	125°

この時の結果を図 3.4 に表す．図の見方は，先ほどと同じである．図 3.4 を見ると，真ん中の

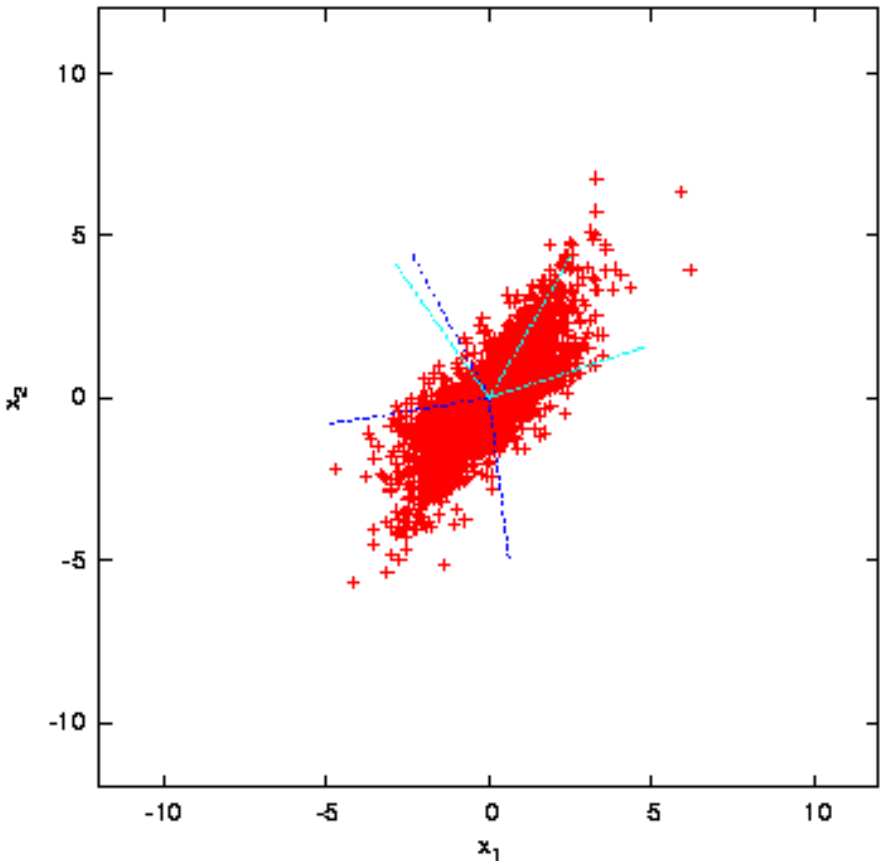

図 3.4. 入力 $\vec{x}$：赤い点，基底：青線，$\vec{y}$ の傾き（回転角）：水色の線

青線（第二基底）だけ出力を表現できているが，そのほかの 2 本の線は出力を表現できているとは言いがたい結果となった．

3.2　64 次元の画像データを入力とした場合

モデルを調べるための応用実験として，64 次元のデータを 16 次元のデータで表すことができるかを調べた．実験では，入力素子数 64 個，出力素子数 16 個とし，図 3.5 のような 8×8 の 2 値画像を入力とした．しかし，図 3.5 のような適当な画像を入力としたときには，入力の情報の損失なく低次元の出力で反映させようとしても，情報の損失のある出力しかできない．もし，反映させたものを作りたいのなら，入力と同等の次元でしか再現できないだろう．なので，実験では 2^{64} 通りの画像の中でも，縦線や横線の組み合わせでできている 2^{16} 通りの図 3.6 の様な画像を使用した．この様な画像であれば，最低でも 16 個の出力，つまり，各出力ごとに縦線あるいは横線が 1 本だけ表示されているような画像の組み合わせで入力が再現可能である．縦線や横線の組合わせは，縦 1/8，横 1/8 の確率で線が表示されるように作成した．学習に必要な係数は以下に設定し，学習は 1000 回に 1 回行うことを 10000 回繰り返した．

λ	μ	η
0.4	0.00001	0.2

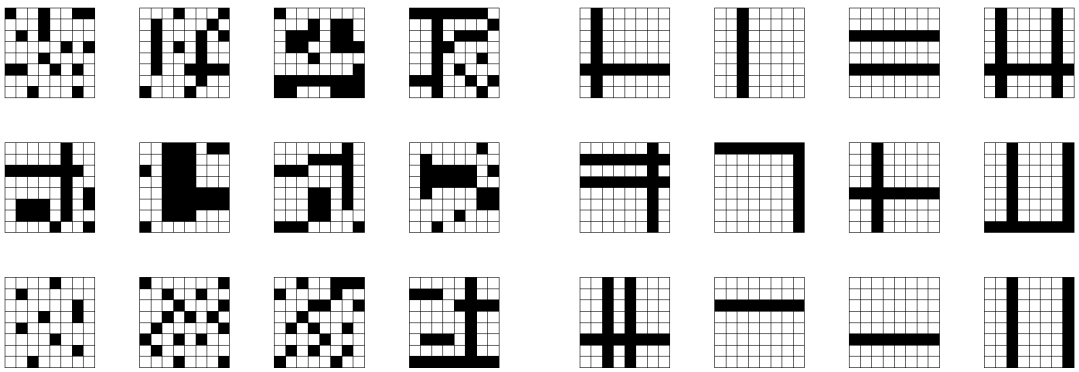

図 3.5. 2^{64} 通りの画像

図 3.6. 入力画像の例（2^{16} 通りの画像）

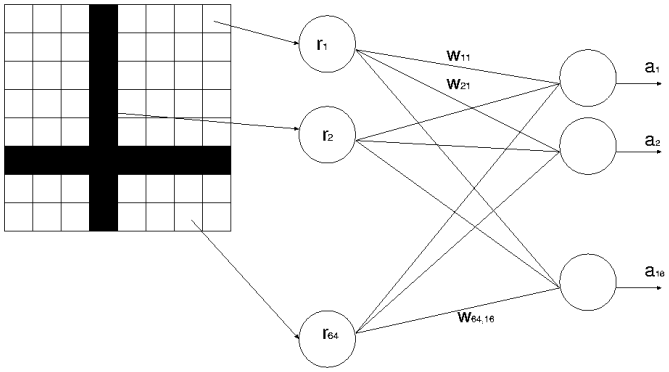

図 3.7. アルゴリズム

図 3.8，3.9 は，図 3.6 のような画像を入力としたときの，出力 a_1 に対する基底関数の画像である．左側は，第 1 基底、右側は第 2 基底で，上から初期状態，2500 回目，5000 回目，8500 回目，9650 回目，10000 回目の基底画像である．基底画像の 1 ピクセルは，そのとき入ってきた入力画像 1 ピクセルに対して，学習アルゴリズムによって算出された値が入っている．色が明るくなるにしたがって，入力画像の線部分をより良く再構成していることが分かり，暗くなるにしたがって，線以外の部分を再構成していることが分かるようになっている．

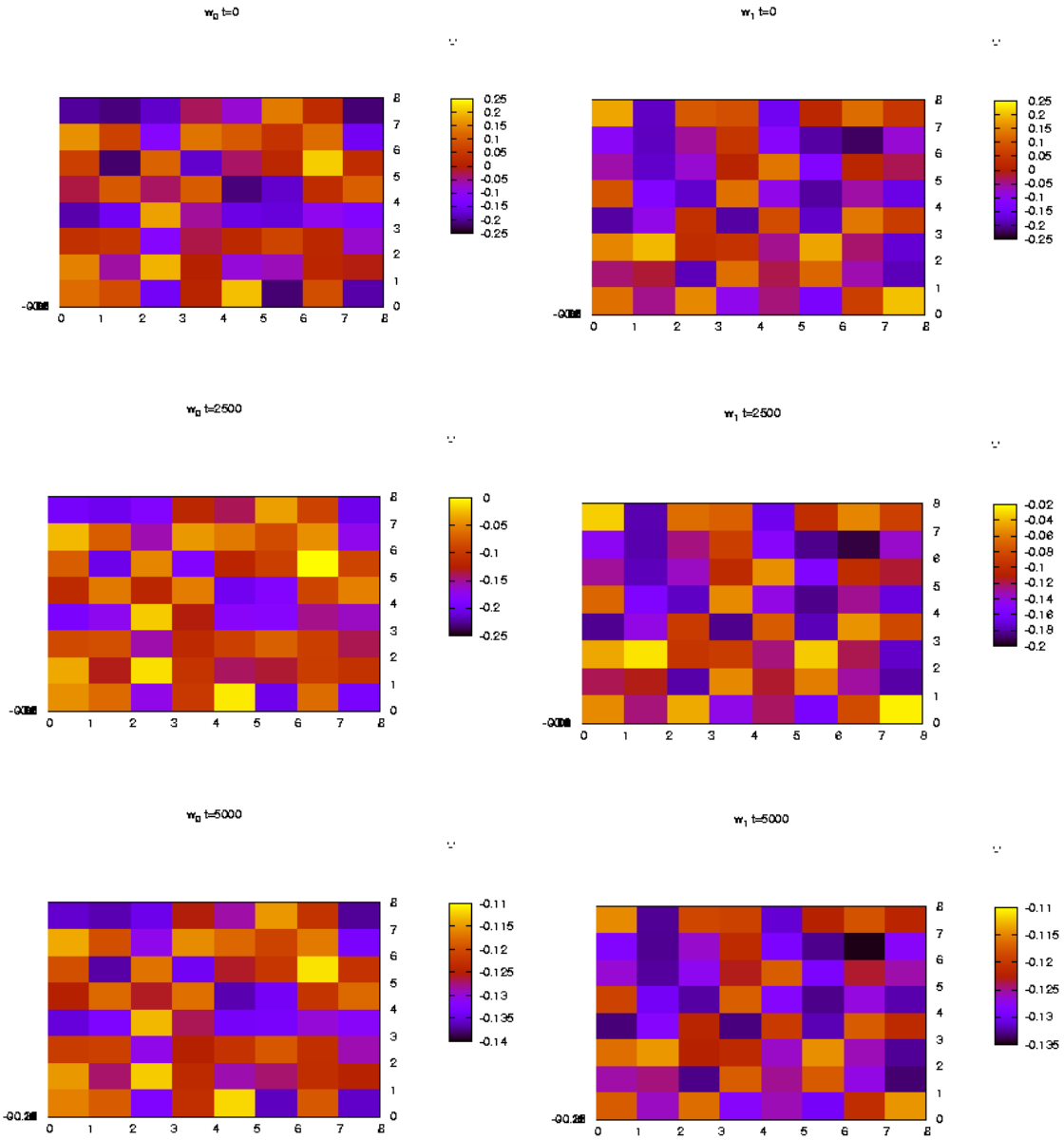

図 3.8. 基底画像：左：第 1 基底，右；第 2 基底：上から初期状態，2500 回目，5000 回目

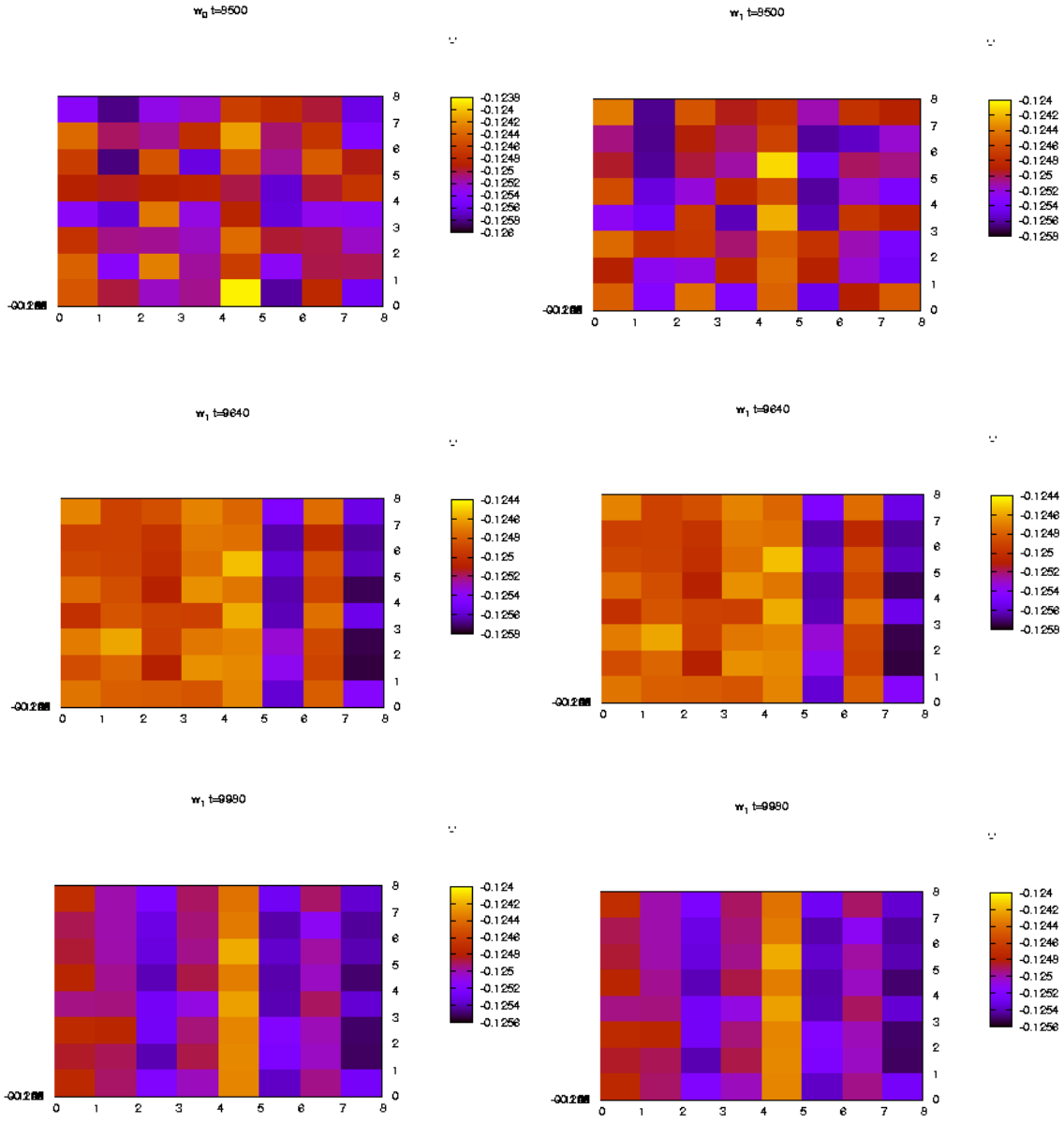

図 3.9. 基底画像：左：第 1 基底，右；第 2 基底：上から 8500 回目，9650 回目，10000 回目

図 3.9 からも分かるように，大体 9000 回目の画像から黄色い縦線を観測する事ができた．しかし，学習回数が増える毎に色々な線を表示してしまい，基底が確定することができなかった．第 1，第 2 基底を見ると分かるが，学習回数を重ねる毎に同じような画像を表示している．ここでは，2 つの基底のみを載せているが，16 個すべての基底画像が同じような画像，もしくは，正（黄色い部分）と負（青い部分）が反転した画像を表示していた．参照論文 [2] のように各基底ごとに縦線もしくは横線が確定された画像が望ましいが，それを結果として得ることはできなかった．

第 4 章

まとめ

要素和エントロピー最小化原理により学習する神経回路モデルについて性能の評価をした．2 次元のデータを入力したときの基底関数は，ほぼ入力を再現できる事が分かったが，64 次元からなる縦棒や横棒の画像データを 16 次元に収縮する場合の基底画像の獲得はできなかった．学習係数を変化させることで学習の頻度が変化するので，実験の過程で色々な値に変化させたが，3 章のように縦線横線は観測されるが，画像内で描かれる線が確定せず，各基底の画像が同じような画像になっていった．考えられる原因としては，プログラム内の数式のミスや書き方が考えられる．

謝辞

この研究を卒業論文として形にすることが出来たのは，指導教官の伊達 章准教授の熱心なご指導のおかげです．また，日常の議論を通じて多くの知識や示唆を頂いた伊達研究室の皆様に感謝します．本当にありがとうございました．

参考文献

[1] B. A. Olshausen, D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. Nature, vol. 381, pp.607-609, 1996.

[2] G Harpur, R Prager, 2000, Experiments with low-entropy neural networks, in R Baddeley, P Hancock, P F*öldiák* (eds.) Information Theory and the Brain, Cambridge University Press, pp 84-100.

[3] 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（2008）.

[4] 村田 昇：”入門 独立成分分析”，東京電気大学出版局（2004）

付録 A

プログラムリスト

実験 2 で使用したプログラム．実験 1 のプログラムは，実験 2 のプログラムのアルゴリズム部分を使用し，入力の部分をワイブル分布と x の式を書き加えればよいので割愛する．

```
1  #include <stdio.h>
2  #include <math.h>
3  #include <stdlib.h> /* for drand48 */
4
5  #define N_INPUTS 64
6  #define N_OUTPUTS 16
7  #define APPERED_RATE 0.125
8  #define N_LEARNING 10000
9  #define N_DATA 500
10 #define SLEEP 100000
11
12 int N_SET_LEARNING = 10000;
13 int EVERY_N_LEARNINGS = 1000;
14
15 double X[N_INPUTS],X2[N_INPUTS];
16 double R[N_INPUTS],R2[N_INPUTS];
17 double A[N_OUTPUTS];
18 double PA[N_OUTPUTS];
19 double W[N_OUTPUTS][N_INPUTS];
20
21 double ER[N_INPUTS]; /* average value */
22 double EA[N_OUTPUTS]; /* average value */
23
24 /* set this value according to your data */
25 int N_XUNITS = 8;
26 int N_YUNITS = 8;
27
28 double LAMBDA = 0.4;
29 double MU = 0.00001;
30 double ETA = 0.5;
31 int RAND_SEED= 12345678;
32 double CONVERGE_CONDITION = 0.000121;
33
34 char A3[N_OUTPUTS];
35 char R3[N_INPUTS];
36
```

```
37 void general_input();
38 void generate_input_random ();
39 void init_parameters(); // set wij random */
40 void calculate_activities();
41 void learning ();
42 void generate_data();
43 void write_data ();
44
45 void reset_average_activity();
46 void update_average_activity();
47
48
49 FILE *gp;
50
51 int main(int argc, char *argv[])
52 {
53   int i,j,k;
54   long seed = RAND_SEED;
55
56   double img[N_INPUTS];
57   int nx,ny;
58
59   nx = N_XUNITS;
60   ny = N_YUNITS;
61
62   srand48(seed);
63   init_parameters();
64
65   gp = popen("gnuplot␣-geometry␣480x480","w");
66   fprintf(gp, "set␣pm3d\n");
67   fprintf(gp, "set␣view␣0,0\n");
68   fprintf(gp, "unset␣key\n");
69
70   for(k = 0;k <N_SET_LEARNING;k++){
71     reset_average_activity();
72     for(j = 0; j < EVERY_N_LEARNINGS; j++){
73       general_input();
74       calculate_activities();
75       update_average_activity();
76     }
77     learning();
78     if( k % 10 == 0){
79       for(i=0;i<N_INPUTS;i++){
80         //fprintf(gp,"set terminal postscript eps enhanced color \n");
81         img[i] = W[0][i];
82         //fprintf(gp,"set output 'img/file0-0%d.eps'\n",k);
83       }
84       fprintf(gp,"set␣title␣\"W_0 t = %d\"\n",k);
85       fprintf(gp, "splot␣'-'␣with␣pm3d\n");
86
87       write_data(img,nx,ny);
88
89       fprintf(gp,"e\n");
90       fflush(gp);
```

```
91      usleep(SLEEP);
92      // printf("t = %d \n",k);
93    }
94  }
95  fclose(gp);
96  return 0;
97 }
98
99
100 void init_parameters() //パラメータの初期化
101 {
102   int i,j;
103   double sum;
104
105
106   for(i=0; i < N_OUTPUTS; i++){
107     sum = 0.0;
108     for(j=0; j < N_INPUTS; j++){
109       W[i][j] = drand48 () - 0.5;
110       sum = sum + W[i][j]*W[i][j];
111     }
112
113     for (j = 0; j < N_INPUTS; j++) {
114       W[i][j] = W[i][j] / sqrt(sum);
115     }
116
117   }
118 }
119
120 void reset_average_activity()
121 {
122         int i;
123
124         for(i=0; i < N_OUTPUTS; i++){
125                 EA[i] = 0.0;
126         }
127         for(i=0; i < N_INPUTS; i++){
128                 ER[i] = 0.0;
129         }
130 }
131
132 void update_average_activity(){
133
134         int i;
135
136         for(i=0; i < N_OUTPUTS; i++){
137                 EA[i] = EA[i] + A[i];
138         }
139         for(i=0; i < N_INPUTS; i++){
140                 ER[i] = ER[i] + R[i];
141         }
142 }
143
144
```

```
145
146 void calculate_activities() // A,R,OMEGA の計算
147 {
148   int i,j,k;
149   double sum;
150   double lambda;
151   double diff = 100.0;
152
153
154   /* init_activities */
155
156   for(i=0; i < N_INPUTS;i ++){
157     R[i] = X[i];
158   }
159
160
161   /* no need ? */
162   for(i = 0;i < N_OUTPUTS;i++){
163     sum = 0.0;
164     for(j = 0;j < N_INPUTS;j++){
165       sum = sum + R[j]*W[i][j];
166     }
167     A[i] = sum;
168   }
169
170
171   k=0;
172   while ( diff > CONVERGE_CONDITION ){
173
174     for(i = 0; i < N_OUTPUTS;i++){
175      PA[i] = A[i];
176     }
177
178
179     for(i=0; i<N_INPUTS; i++){
180       sum = 0.0;
181       for(j = 0;j < N_OUTPUTS;j++){
182         sum = sum + PA[j]*W[j][i];
183       }
184       R[i] = X[i] - sum;
185     }
186
187     for(i = 0;i < N_OUTPUTS;i++){
188
189       sum = 0.0;
190       for(j = 0;j < N_INPUTS;j++){
191         sum = sum + R[j]*W[i][j];
192       }
193
194       if ( PA[i] >= 0.0 ){
195         lambda = LAMBDA;
196       }
197       else{
198         lambda = -LAMBDA;
```

```
199       }
200
201       A[i] = MU*(sum - lambda);
202     }
203
204
205     diff = 0.0;
206     for(i = 0;i < N_OUTPUTS;i++){
207       diff = diff + fabs(PA[i] - A[i]);
208     }
209     // printf("%d: diff = %f\n",k, diff);
210     k++;
211   }
212
213 }
214
215
216 void learning () // Hebb 学習
217 {
218
219   int i, j;
220   double sum;
221
222   //printf("A[5] = %.5lf \n",A[5]);
223   //printf("R[5] = %.5lf \n",R[5]);
224
225   for (i = 0; i < N_OUTPUTS; i++) {
226     sum = 0.0;
227     for (j = 0; j < N_INPUTS; j++) {
228
229       W[i][j] = W[i][j] + ETA * EA[i]*ER[j]/(double)EVERY_N_LEARNINGS;
230       sum = sum + W[i][j]*W[i][j];
231     }
232     for (j = 0; j < N_INPUTS; j++) {
233       W[i][j] = W[i][j] / sqrt(sum);
234     }
235   }
236
237   //printf("W[5][5] = %.20lf \n",W[5][5]);
238 }
239
240
241 void general_input() // 8掛ける 8の枠の中に縦線,横線をランダムで作成
242 {
243
244   int i, j, k;
245
246   for (i = 0; i < N_INPUTS; i++) {
247     X[i] = 0.0;
248   }
249
250   for (j = 0; j < N_XUNITS ; j++) {
251     if (drand48 () < APPERED_RATE ) {
252       for (k = 0; k < N_YUNITS ; k++) {
```

```
253         X[j + k*N_XUNITS] = 1.0;
254       }
255     }
256   }
257
258   for (j = 0; j < N_YUNITS; j++) {
259     if (drand48 () < APPERED_RATE ) {
260       for (k = 0; k < N_XUNITS; k++) {
261         X[N_XUNITS*j + k] = 1.0;
262       }
263     }
264   }
265 }
266
267
268 void write_data( double *img, int nx, int ny){
269
270         int i,x,y;
271
272         for (x=0; x < nx; x++){
273                 for (y=0; y < ny; y++){
274                         i = x + (ny -y -1)*nx;
275                         fprintf(gp, "%d␣␣%d␣%lf\n", x, y, img[i]);
276                         fprintf(gp, "%d␣␣%d␣%lf\n", x, y+1, img[i]);
277                 }
278                 fprintf(gp, "\n");
279
280                 for (y=0; y < ny; y++){
281                         i = x + (ny -y -1)*nx;
282                         fprintf(gp, "%d␣␣%d␣%lf\n", x+1, y, img[i]);
283                         fprintf(gp, "%d␣␣%d␣%lf\n", x+1, y+1, img[i]);
284                 }
285                 fprintf(gp, "\n");
286         }
287
288 }
```