計算機アーキテクチャ　113－3
（１９９５．　８．　２３）

擬似Kohonen Networkの LVQ学習

福本 光　豊崎 剛　阿江 忠
hikaru@aial.hiroshima-u.ac.jp
広島大学 工学部 第二類　知能情報工学研究室
〒739 東広島市鏡山 1-4-1

ニューラルネットでのリアルタイム処理を考えると、そのハード化は不可欠である。本報告では、専用ニューラルネットとして、自己組織化機能を有するKohonen Netをとりあげ、そのハード化について考察する。ニューラルネットの動作モードにおいて学習のしめる割合が多いことより、LVQ学習アルゴリズムの並列処理と、そのチップ化について考察した。その際、トランジスタ数の制約上、Kohonen Net自体を修正し、擬似Kohonen Netを提案した。動作のシミュレーションを行った結果、擬似Kohonen Netの有用性が確かめられた。

LVQ Learning using Simplified Kohonen Network

Hikaru Fukumoto, Tuyoshi Toyosaki and Tadashi Ae
Electrical Engineering,Faculty of Engineering,Hiroshima University
1-4-1,Kagamiyama,Higashihiroshima-shi,Hiroshima,739,Japan

Considering a real-time processing on Neural Net, the hardware neuron is indispensable. In this paper, we adopt the Kohonen Net as a specific Neural Net which realizes feature maps through a self-organizing process, and consider the hardware implementation of it. Since learning dominates the total behavior, we introduce the parallel processing of LVQ (Learning Vector Quantization) algorithm and it's hardware into consideration. Under the restriction of the number of transistors, we propose a simplified Kohonen Net which is modified from the Kohonen Net. We simulated it, and verified its applicability.

1 はじめに

ニューラルネットの効用はよく知られているが、BP(誤差逆伝搬学習アルゴリズム)を用いる多層型ニューラルネットはもっともポピュラーなものであろう [1]。

しかし、BPにも適用限界があり、最近では、自己組織化機能を持つタイプが注目を集めている [2]。

一方、リアルタイム処理を考えると、高速処理の必要性から、ニューラルネットのハード化は不可欠であり、人工ニューラルネットとしての研究も数多くなされている [3]。ハードウェアニューラルネットはマルチプロセッサ的な大規模なものから、ニューロチップ的なものまで数多く分布している。前者が各種のニューラルネットモデルに対応できる汎用性を有するのに対し、後者は簡単になればなるほど特定のニューラルネットモデルに専用のものになる傾向がある。

本報告では、専用ニューロチップを対象とする。これは、ニューロン数の増大を考えると、何らかの犠牲を考えないと、リーズナブルなサイズのハードウェアニューラルネットの実現は困難であるためである [4]。専用ニューラルネットとしては、自己組織化タイプの代表例であるKohonen Netをとりあげる。Kohonen Netはパターン認識のみならず、データベースや記号処理のような応用も可能である [5]。このようにニューラルネットを専用化した場合、ニューラルネットそのもののハード化という方法の他に、学習アルゴリズムのハード化という方法がクローズアップされてくる。ニューラルネットの動作モードのうち、学習のしめる割合が大きいからである。とりわけ、リアルタイム応用では、学習が動作途中に何度も繰り返されることも多い。ディジタル化したKohonen Netの場合、学習アルゴリズムのハード化は、ニューラルネットそのもののハード化に比べて、非常に効率がよい。

このような観点から、我々の研究室では、Kohonen Netのハードウェア化を試み、概念設計、主要チップ設計を行なった [6] [7]。

今回は、チップ設計をゲートアレイで行なったため、トランジスタ数に制約が加わった [8]。そのため、Kohonen Net自体を修正した。修正された擬似Kohonen Netの動作をソフトウェアでシミュレーションし、応用によっては十分適用可能であることを確認することが出来た。

2 LVQアルゴリズムとその手続き

Kohonen Netの場合、ニューラルネット自体のハード化の代わりに、学習アルゴリズムのハード化の試みは、すでに、アナログニューロンで行なわれている [9]。しかし、アナログニューロンでは、ニューロン数に限界があり、我々はKohonenの学習アルゴリズムのディジタルハードウェア実現の研究を始めた [6]。Kohonenの学習アルゴリズムとしては、LVQ(学習ベクトル量子化)アルゴリズムをとりあげる。

LVQ学習アルゴリズムは、各カテゴリの境界をより精密に区別するために参照ベクトルを判定面から引き離していく。なお、以下の文章内で用いる、入力ベクトル(input vector) x は学習を行なうためのサンプルのベクトルで名前、データ、及びカテゴリ名を持っている。参照ベクトル(reference vector または codebook vector) m_i は、ニューロンに対応するベクトルで、各カテゴリに付き1個から数個存在する。

m_cをxにもっとも近い参照ベクトルとする。この時$m_i = m_i(t)$ の更新は次のようになる。

$$
\begin{aligned}
m_c(t+1) &= m_c(t) + \alpha(t)[x(t) - m_c(t)] \\
&\quad \text{if } x \text{ is classified correctly,} \\
m_c(t+1) &= m_c(t) - \alpha(t)[x(t) - m_c(t)] \\
&\quad \text{if the classification of } x \text{ is incorrect,} \\
m_i(t+1) &= m_i(t) \qquad \text{for } i \neq c.
\end{aligned}
$$

ここで$\alpha(t)$は、時間により単調に減少するスカラー値である $(0 < \alpha(t) < 1)$。この値により更新一回あたりの参照ベクトルの変化の大きさを決め、かなり小さな値、例えば$\alpha = 0.01$以下の値から数百から数万ステップでε以下まで減少させる。なお、各カテゴリに幾つかの参照ベクトルが分配されている時、入力xのカテゴリは、xの最近傍m_iのラベルにより定義される。

上記のアルゴリズムを実行するprocedureは次のように表される。

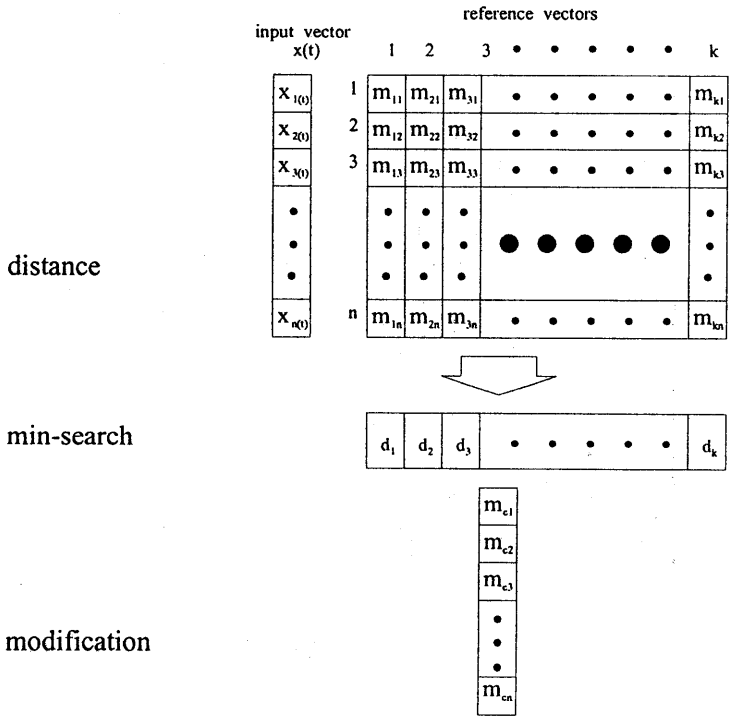

図 1: LVQ 学習アルゴリズムの本体

```
procedure lvq (ε : real)
    while α(t) ≥ ε
    do
        distance;
        min-search;
        modification;
    od;
end lvq
```

実行の本体である distance、min-search、modification は図 1 に示す通りである。distance では入力ベクトルと各参照ベクトルとの距離を計算する。その距離の最小値を min-search で検索し、更新する参照ベクトルを決めている。modification でその参照ベクトルを更新している。これらが必要な回数だけ繰り返される。

3 LVQ アルゴリズムの並列処理

図 2の実行本体のそれぞれの並列化は次のようになる。

(1) 距離計算 distance

ニューロン数に依存した処理：O(k)
各ニューロン内でのシナプス数に依存した処理：O(n)
すなわち、ニューロン単位での並列化は k までであるが、シナプス計算も並列化すると、最大 nk まで並列化可能である (ハードウェア量に制約がある場合、ニューロンの個数を

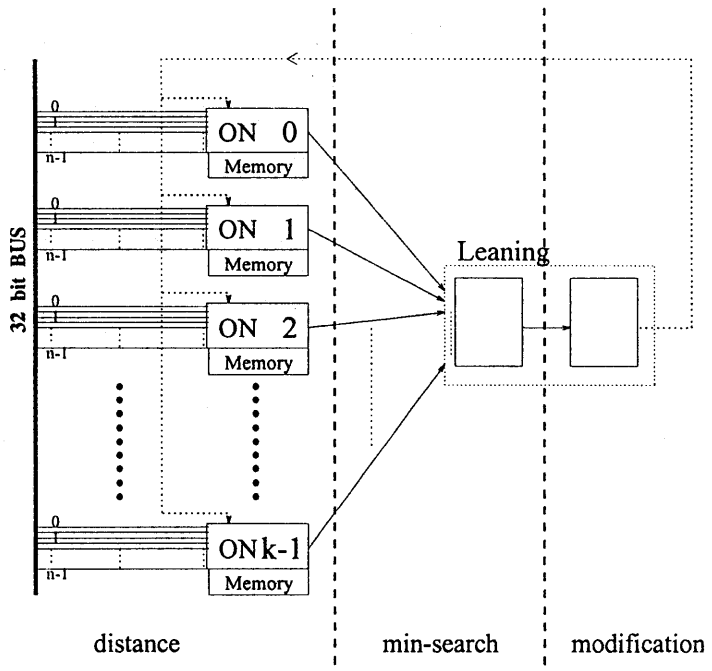

図 2: 対象とするハードウェア LVQ 学習アルゴリズム

限定して、シナプス並列に重点をおいたインプリメンテーションもある [4])。

(2) 最小値検出 min-search

k 個のニューロンの出力の比較であり、次の 3 通りがある。

逐次的：O(k)

比較木：O(log n)

バス：O(1)

いずれも計算の手数をオーダで示している。(バスは光バスのような低損失なものを仮定している [10])。min-search にどれを採用すべきかは、トータルの手続きに占めるウエイト次第であり、逐次的なもので十分なこともある。

(3) 更新 modification

シナプスに依存した処理：O(n)

Kohonen Net の特長は、この更新の手数が 1 ニューロンのみであることである。

LVQ 学習アルゴリズムの実行本体は極めて自然に並列化できる。以下、本報告では、図 2のようにハードウェア化した LVQ 学習アルゴリズムを対象とする。

4 対象とするハードウェアニューラルネット

図 2のような LVQ 学習アルゴリズムのハードウェア化を、現状のチップ集積度から考えると次のようになる。

ケース 1：100 万トランジスタ程度

64 ニューロン相当プロセッサ/チップ

この場合、シナプス計算は逐次処理となる [10]。

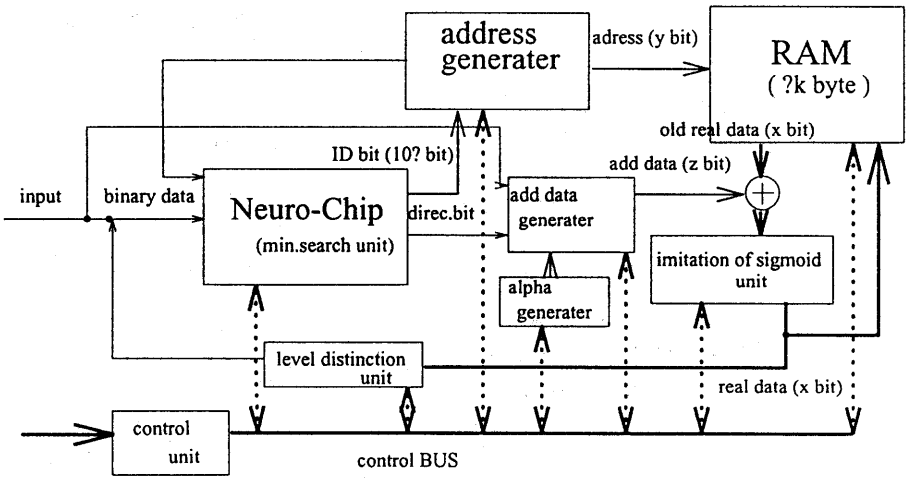

図 3: 具体的な構成

ケース 1：100 万トランジスタ程度
　64 シナプス相当プロセッサ/チップ
この場合、ニューロン単位では逐次処理となるがパイプライン化は容易である。[10]。
(以上の概算は LVQ 学習アルゴリズムに特定したものではない。一般的な議論がそのまま適用できるので引用している。)

一般のニューラルネットの場合、ニューロン、シナプスともに並列化しようとすると、現状の集積化技術では限界がある。むろん、チップを増やすことで規模を増大させることは可能であるが、今度はチップ間の相互結合が問題となる。ニューラルネットは相互結合が特徴であるが、インプリメンテーションではそれが障害となる。

その点、Kohonen Net をネットワークそのものではなく [11]、学習アルゴリズムのハードウェア化に置き換えると、チップ間結合は、図 2からわかるように、単純であるため、比較的容易となる。

ただし、本報告は次のケースにあたる。

ケース 3：10 万トランジスタ程度

これは、2 万ゲート程度のチップを HDL により、ゲートアレイ実現させたためである [8] [12]。

その結果、

(1) **距離計算** O(nk) の並列化を実現するため、距離をハミング距離とした。

(2) **最小値検出** OR 回路によって実現しており、バスを用いる考え方と同じである。

(3) **更新** 付加プロセッサによる逐次処理：図 3 参照

という簡略化を行ない、n = 32, k = 12 を 1 チップ内に実現した [8]。

この Kohonen Net を以下擬似 Kohonen Net と呼ぶ。n = 32 は固定であるが、k は必要なだけチップを増やせば大きくすることができる。

5 擬似 Kohonen Net の場合

5.1 概要

距離計算では、ハミング距離を用いる。すなわち、2 値の要素を持つ 2 つのベクトルを

$$\boldsymbol{o_i} = (o_{i1}, o_{i2}, \ldots, o_{in})$$
$$\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$$

とすると、ベクトル間の距離 Dは

$$D = \sum_{k=1}^{n} |x_k - o_{ik}| \qquad (1)$$

で表される。しかし参照データの変化がなければ、学習は不可能であるので、付加的に実数値 (Reference value) $m_{ik}(m_{i1}, m_{i2}, m_{i3} \cdots, m_{in})$ を格納す

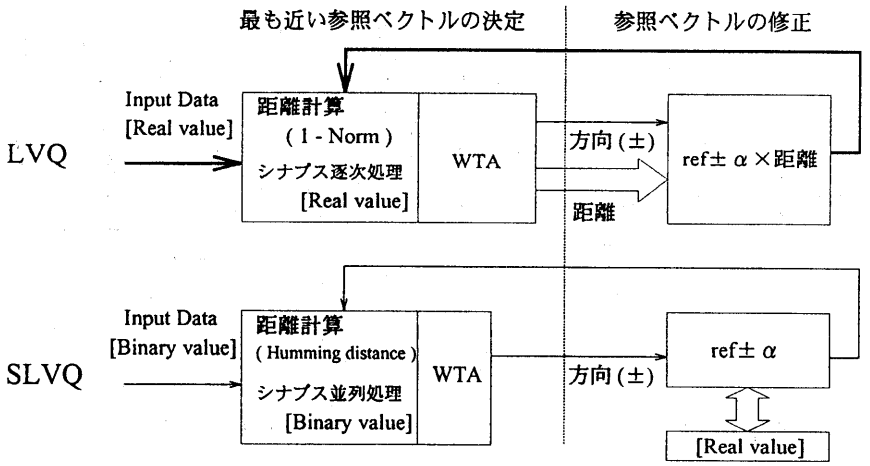

図 4: LVQ と SLVQ

る。この結果 o_iは Reference value のスレシホールドをとったものとなる。

ここで式 (1) は、次のように修正される。

$$D = \sum_{k=1}^{n} |x_k - h(m_{ik})| \qquad (2)$$

関数 h はスレシホールドをとるもので、内部の数値をバイナリに変換する。

文献 [6] の Kohonen Net [LVQ] と本稿の擬似 Kohonen Net[SLVQ] との違いを図 4に示す。

5.2 Real value 発散の問題

擬似 Kohonen Net の学習において、Reference value (内部状態) が発散するおそれがある。

この問題の解決策として次の 2 つを考えた。

(1) シグモイド関数を使う。

(2) 速度関数を使う。

次にこれらを説明する。

(1) シグモイド関数 関数の使い方を図 5に示す。

逐次、Reference value をシグモイド関数に通して非線形圧縮が行い、発散を防ぐ。ここで注意することは、通常のシグモイド関数にこだわらなくても非線形圧縮が行なわれるものならどんな関数を用いても良いこと

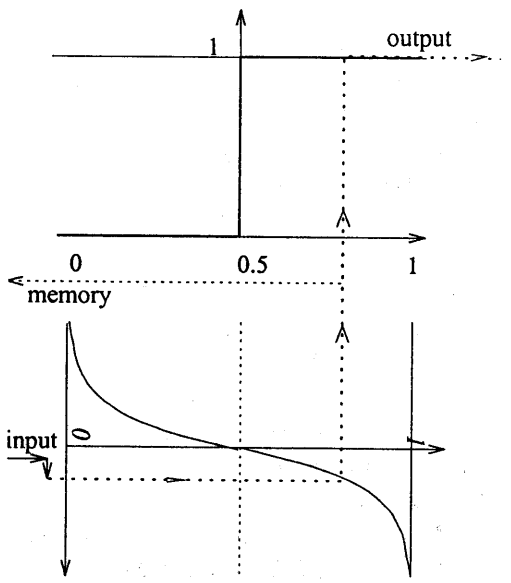

図 5: シグモイド関数の使い方

である。ただし、できれば将来のハードウェア化を考えたものが望ましい。そこで、擬似シグモイド関数なるもの、具体的には階段状のもの、折れ線状のもので代用したものをシミュレートした。

(2) 速度関数 この方法は先のシグモイド関数を使うものとは異なり、1 または 0 をつき抜けないよう、変化する量を決める関数である。式として速度関数は次のように簡単に表される。

$$f(x) = \begin{cases} x & (x < \frac{1}{2}) \\ -x+1 & (x \geq \frac{1}{2}) \end{cases} \qquad (3)$$

6 シミュレーション

シミュレーションとしては 32(4 × 8) ビットパターンデータを用いた。結果の一例を図 6に示す。図 6は擬似 Kohonen Net が十分適応できることを示している。

(1) シグモイド関数、(2) 速度関数 については、どちらを用いても結果はほとんど同じであった。

7 おわりに

チップサイズの制約上、1カテゴリにつき32ビットでシミュレーションを行なった。応用上は、ビット数をもっと大きくする必要があるが、擬似 Kohonen Net という簡略化の有用性は32 ビットのシミュレーション結果から、類推することができる。

ハードウェアニューロチップのデモシステムは別途製作中である。

文献

[1] 久間 和生, 中山 高 "ニューロコンピュータ工学", 工業調査会, 1991

[2] 阿江忠,"神経回路網と機能メモリ", 情報処理, Vol.32, No.12, pp.1301-1309, 1991

[3] 阿江 忠,"VLSI ニューロコンピュータ -21 世紀のアーキテクチャをめざして-", 共立出版, 1991

[4] T. Ae, and R. Aibara,"Non von Neumann Chip Architecture", IEICE Trans.,Vol. E76-C, No.7., pp.1034-1044, 1993

[5] T. Kohonen "The Self Organizing Map", Proceedings of the IEEE, 78, 9, pp.1464-1480, 1990

[6] Ae,T., Aibara,R. and Kioi, K.,"Design of Neural Self-Organization Chip for Semantic Applications", 3rd Inten, Workshop on VLSI for Neural Networks and Artificial Intelligence, Oxford, 1992

[7] T.Ando, K.Kioi, R.Aibara, and T.Ae, "A Chip Design of Generalized Winner-Take-All Circuit", IJCNN, pp.1971-1974, 1993

[8] T.Toyosaki and T.Ae, "A Neuro-Chip For Kohonen's LVQ Algorithm", ISIC-95, (to appear on Sep. 1995)

[9] J.Mann, R.Lippmann, B.Berger, and J. Raffel, "A Self-Organizing Neural Net Chip", Proc. CICC, pp.10.3.1-10.3.5, 1988

[10] 安藤 健, 豊崎 剛, 酒居 敬一, 阿江 忠, "直交光バスを想定した並列演算器" IEICE technical report, CPSY94-51 pp.81-87, 1994

[11] 小野寺 秀俊, 竹下 潔, 田丸 啓吉, "Kohonen ネットワークのハードウェアアーキテクチャ", 電子情報通信学会技術報告, ICD 89-146,Nov. 1989

[12] T.Ae, T.Toyosaki, H.Fukumoto and K.Sakai, "ONBAM : An Objective-Neuron-Based Active Memory", Proc. ICAPP 95, pp.231-234, 1995

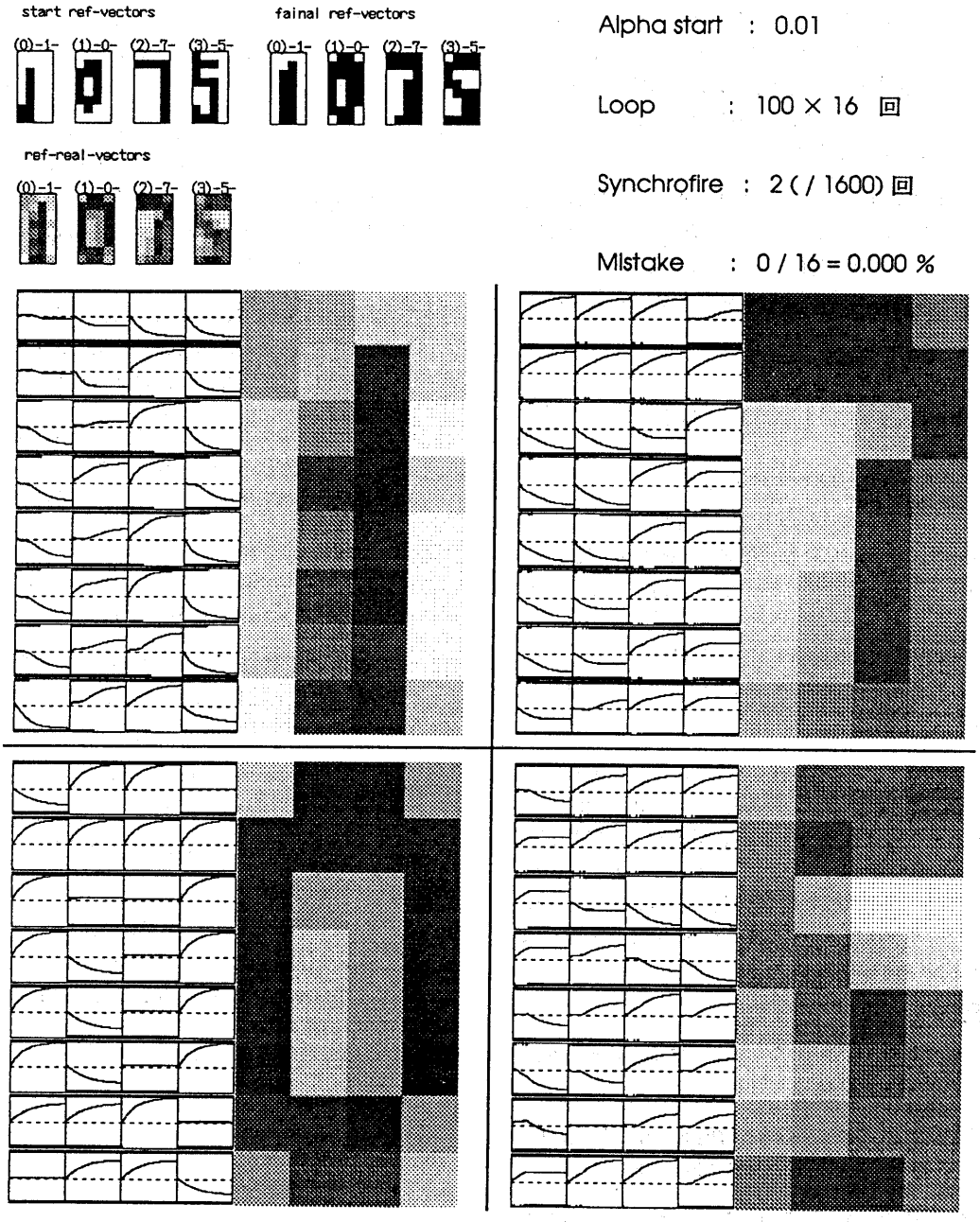

図 6: シミュレーション結果の一例