関西大学理工学会誌
理工学と技術
Vol.15（2008）

トピックス

自己組織化マップと画像圧縮への応用

肥川宏臣*

Self-Organizing Map and Its Application to Image Compression

Hiroomi HIKAWA

1．まえがき

ニューラルネットワークの1つに自己組織化マップ（Self-Organizing Map：SOM）がある。自己組織化マップは人間や動物の脳の視覚に関する自己組織化に関する研究を起源とする。視覚細胞は外界からの刺激に反応するが、視覚第1次野と呼ばれる部位には、さまざまな方位に反応する細胞がたくさん並んでいる。そして、縦方位の明暗の境界線に反応する細胞の隣には少しだけ斜めに傾いた境界線に反応する細胞があり、細胞の反応する方位が皮質上で連続的に変化することが知られている。このように、似たような外界からの刺激に対して隣接する細胞が反応することをトポロジカルマッピングと呼ぶ。Malsburgは、脳の内部で行われているトポロジカルマッピングが競合学習モデルによって、形成されることを明らかにしている。さらに、Kohonenは、これらのモデルを基に、任意次元の特徴空間から2次元ニューロン配列へのトポロジカルマッピングが可能であることを示し、モデルの一般化を行っている[1]。これは同時に、自己組織化マップが入力された高次元のベクトルデータを2次元のデータに変換する機能を持つことを示している。

自己組織化マップの機能は、高次元のベクトルデータを2次元に配置されたニューロンにマッピングすることでもあるが、トポロジカルマッピングが行われるため、差が小さい異なる入力データは隣接するニューロンに割り当てられる。また、差が非常に小さい入力データは、同じニューロンへ割り当てられる。自己組織化マップは、その名前が示すように、入力データにより自己組織化を行い、マッピングを自分自身で学習する。学習終了後、入力ベクトルはいずれかのニューロンへ割り当てられることになるが、これは、ある程度似た入力ベクトルをニューロンの個数に等しいグループに分けるクラスタリングを行うことに他ならない。このグループごとのベクトルの特性を調べることで様々なデータ解析を行うことができる。例として、自己組織化マップを用いた情報分析に基づく経済分析や事業戦略管理、建設分野における地盤構造の推定や斜面崩壊予測システム、効率的なロボット制御方法の学習などへの応用研究が多数行われている[2]。

また、入力ベクトルを割り当てられたニューロンが持つベクトルと置き換えることで、ほぼ無限の値を取りうる入力ベクトルを有限の個数（ニューロンの個数）にまとめることができる。これをベクトル量子化と呼ぶ。このグループ化とベクトル量子化の性質を使うことでデータの分類やパターン認識、画像圧縮といった処理を行うことができる。本文の後半では、自己組織化マップの画像圧縮への応用について説明を行う。

2．自己組織化マップの動作

SOMへの入力は、K個の変数を持つK次元のベクトルデータ$\vec{x}$である。

$$\vec{x} = \{\xi_1, \xi_2, \cdots, \xi_K\} \in \Re^K \tag{1}$$

自己組織化マップは、図1に示すように入力層（Input layer）と2次元の格子状に配置されたニューロンの競合層（Competitive layer）により構成される。ニューロンの配列は本質的ではなく、1次元でも3次元の配列なども可能である。各ニューロンは入力

原稿受付　平成20年9月17日
*システム理工学部 電気電子情報工学科 教授

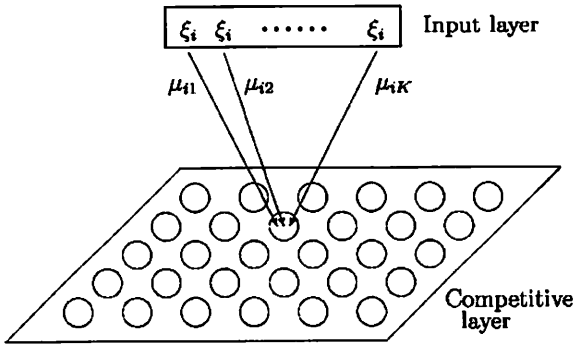


図1　自己組織化マップの基本構成

ベクトルと同じ K 次元の重みベクトル $\vec{m_i}$ を持つ。

$$\vec{m_i} = \{\mu_{i1}, \mu_{i2}, \cdots, \mu_{iK}\} \in \Re^K \tag{2}$$

そして、自己組織化マップの動作は、学習と想起の2つに分けることができる。学習時には、入力ベクトルを随時自己組織化マップに与えることで重みベクトルの更新を行い、マッピングを学習する。学習後は、重みベクトルの更新は行わず、マッピングは想起に用いられる。

学習は、重みベクトルの初期化を行った後、入力ベクトル、$\vec{x}$ が繰り返しSOMに与えられることにより行われる。SOMの学習は次の動作の繰り返しである。

1．入力ベクトルに対する勝ちニューロンを検索する。
2．勝ちニューロンとその近傍にあるニューロンの重みベクトルを入力ベクトルに近づくように更新する。

勝ちニューロンとは、入力ベクトルとの距離がもっとも小さい重みベクトルを持つニューロンのことで、勝ちニューロンの検索では、すべてのニューロンの重みベクトルと入力ベクトルとの距離を求め、それらの中で最も距離が小さな重みベクトルを持つニューロンを勝ちニューロンとする。ベクトル距離としては次に示すユークリッド距離が広く用いられている。

$$\|\vec{x}-\vec{m}\| = \sum_{j=1}^{K} \sqrt{(\xi_j-\mu_{ij})^2} \tag{3}$$

計算コストを削減したい場合、たとえば自己組織化マップを直接ハードウェアに実装するような場合には、次に示すマンハッタン距離が使われることも多い。

$$d_i = \sum_{j=1}^{K} |\xi_j-\mu_{ij}| \tag{4}$$

勝ちニューロンが決定されると勝ちニューロンの重みベクトルは、さらに入力ベクトルとの距離が小さくなるように更新される。また、勝ちニューロンの近傍にあるニューロンの重みベクトルも入力ベクトルとの差が小さくなるように更新されるが、近傍ニューロンについては、重みベクトルの更新量は勝ちニューロンからの距離が大きいほど小さくなり、近傍外のニューロンの重み更新は行われない。以上の更新量の調整は、近傍関数により実現される。近傍関数は勝ちニューロンとの距離を引数として、距離が大きいほど小さな値を返す関数で、距離がある値より大きく近傍外となった場合には0となる。そして、この関数値を更新係数として勝ちニューロンと、その近傍にあるニューロンの重みベクトルの更新を行う。

$$\vec{m_i}(t+1) = \vec{m_i}(t) + h_{ci}\{\vec{x}(t) - \vec{m_i}(t)\} \tag{5}$$

近傍関数の値 h_{ci} は勝ちニューロンに対して最大となる。また、この関数の近傍は時間とともに小さくなる性質を持つ。近傍関数は次式で表される。

$$h_{ci} = \alpha(t)\exp\left(-\frac{\|\vec{r_c}-\vec{r_i}\|}{2\sigma^2(t)}\right) \tag{6}$$

ここで、$\vec{r_c}$ と $\vec{r_i}$ は2次元ベクトルで、それぞれ、勝者ニューロン c とニューロン i の位置ベクトルで、$\|\vec{r_c}-\vec{r_i}\|$ は、それらの位置ベクトルの距離を表す。

学習終了後、想起モードとなり、入力されたベクトルは勝ちニューロンにマッピングされることになる。近傍関数の性質により、類似したベクトルは隣接したニューロンに割り当てられるトポロジカルマッピングが実現される。そして、同じニューロンにマッピングされるベクトルは1つのクラスタを構成し、そのニューロンの重みベクトルがそのクラスタを代表するプロトタイプベクトルとなる。

3．自己組織化マップの画像圧縮への応用

ここでは、自己組織化マップのベクトル量子化を利用した画像圧縮について述べる。

3．1　ベクトル量子化による画像圧縮の原理

ベクトル量子化とは、任意のベクトル集合を少数のベクトル集合で近似することで、多次元量の空間を少数の部分空間（クラスタ）に分割し、そのクラスタに属するベクトルを、そのクラスタの代表値に置き換えることである。ここで述べる画像圧縮では、$P\times Q$ ピクセルの画像を $M\times M$ ピクセルのブロックに分割し、ブロック単位で量子化を行う。画像に含まれるブロックの総数は $N_{blk} = (P\times Q)/(M\times M)$ 個となる。また、ニューロンが $N\times N$ の格子状に配置された構成の自己組織化マップを用いることにする。これにより、

N_{blk}個のベクトル集合を$N\times N$個のベクトル集合で近似することになる。$N_{blk} > N\times N$とすることで、画像を表現するためのデータサイズを圧縮することができる。

ブロック内の$M\times M$個のピクセル輝度値（Rビット）をベクトル要素とする$M\times M$次元のベクトルとして、自己組織化マップへ学習ベクトルとして与える。当然、ニューロンの重みベクトルも$M\times M$次元である。$M=4$の場合、ベクトルの次元数は16次元となる。

学習により各入力ベクトルは、$N\times N$個あるニューロンのいずれかに割り当てられる。割り当てられるニューロンとは、そのベクトルが入力された際に勝ちニューロンとなるニューロンのことである。比較的類似したベクトル（ブロック）は同じニューロンに割り当てられることになるが、これは、それらのベクトルが$M\times M$次元空間の同じクラスタに属することを意味する。そして、そのニューロンの重みベクトルをそのクラスタのプロトタイプベクトル、つまり、そのクラスタを代表するベクトルとして扱う。通常、プロトタイプベクトルはそのクラスタの中心に位置し、そのクラスタに属するベクトルの平均値とみなすことができる。画像圧縮では同じクラスタに属するベクトル（ブロック）をこのプロトタイプベクトルと置き換えることでベクトルの量子化を行い、画像を表現するためのデータ量の削減を図る。つまり、自己組織化マップは画像に含まれるブロックの中で似ているものを検索し、ベクトル空間の分割を行い、それらの平均値を自動的に求める。そして、同じニューロンに割り当てられるブロック（同一クラスタに属するブロック）に、そのニューロンの重みベクトルを$M\times M$に展開したパターンを書き込む。これにより、原画像に含まれていた$(P\times Q)/(M\times M)$個のブロックパターンを$N\times N$個のパターンに減らすことになり、画像表現に必要なデータのサイズを圧縮することができる。ただ、同じパターンを複数のブロックで使いまわすので、画質は悪化してしまうが、うまく学習できれば、画像の悪化は人間の目には許容できる範囲にとどめることができる。

以上のベクトル量子化により、画像を再現するために必要なパターン（ベクトル）の種類を減らすことができるが、各ブロックがどのパターンに置き換えられるかを記録する必要がある。$N\times N$個のパターンを2進数の番号で識別すると、パターン番号には$ceiling(log_2(N\times N))$ビットを必要とする。$ceiling(x)$は天井関数で、実数$x$に対して$x$以上の最小の整数を返す関数である。各ブロックにどのパターンが使われるかを記録するためには$N_{blk}\times ceiling(log_2(N\times N))$ビットのインデックスが必要となる。

ピクセル当たりのビット数がRなので、原画像と圧縮後の画像を表現するための総ビット数、No、Ncは、それぞれ次のようになる。

$$No = R\times P\times Q \tag{7}$$

$$Nc = R\times M\times M\times N\times N + ceiling(log_2(N\times N))\times\frac{P\times Q}{M\times M} \tag{8}$$

式(8)の右辺第1項は表示に必要な$M\times M$画素のブロックを$N\times N$個表すための総ビット数、第2項はブロックに書き込まれるパターンの番号を記録するためのインデックスの総ビット数である。そして、画像の圧縮率はNc/Noとなる。また、圧縮された画像の質を評価する指標に量子化ひずみEがある。これは、原画像と圧縮画像とのピクセルごとの輝度差の総合計で表わされる。

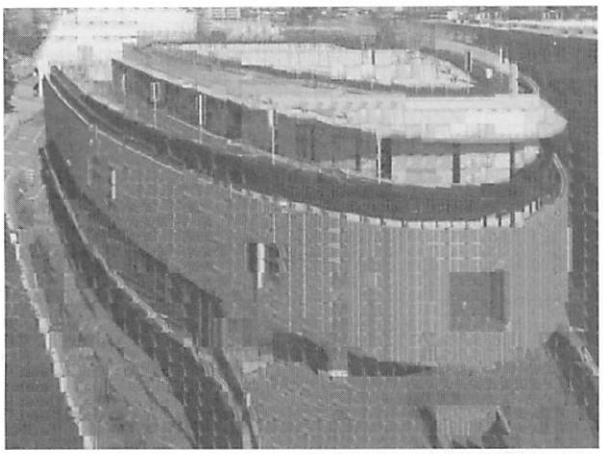
3．2　画像圧縮の例

図2にベクトル量子化の例を示す。図2(A)は、640×480ピクセル、8ビットのグレースケールの原画像で、これを16×16のブロックに分割し、16×16のニューロン構成の自己組織化マップにより量子化して圧縮したものが図2(B)である。そして、図2(C)には同じクラスタに属するブロックを示している。つまり、これらのブロックはすべて同じパターンに置き換えられていることになる。主に壁の一部と階段の一部のパターンが類似しているため、同一クラスタとしてまとめられている。

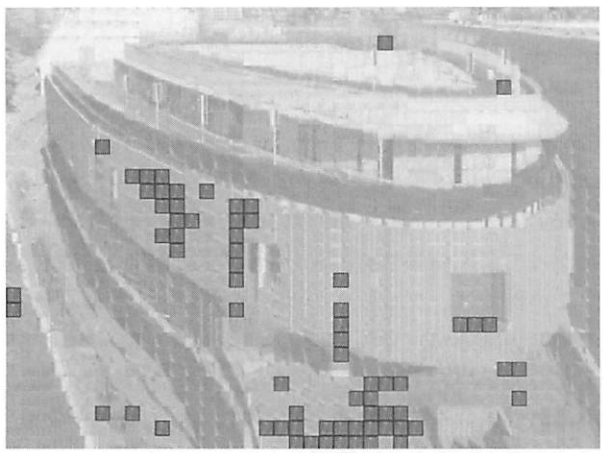
圧縮率については、この例の場合、$No = 640\times 480\times 8 = 2{,}457{,}600$。パターン番号のビット数が$ceiling(log_2(16\times 16)) = 8$なので、$Nc = 8\times 16\times 16\times 16\times 16 + 8\times(640\times 480)/(16\times 16) = 533{,}888$。したがって、圧縮率は$Nc/No = 0.217$となる。

この例では、ブロックのサイズが大きいためブロックの境界部分が目立つことになり四角いブロックが見えてしまい（ブロックノイズという）、圧縮率の割に画像のクオリティが良くない。そこで、ブロックのサイズを変えた場合の圧縮画像を図3に示す。ここでは、画像に含まれるブロック数とSOMのニューロン数の比を75として、ブロック数についての圧縮率が同じになるようにしている。ブロックサイズが小さい方がクオリティの高い圧縮が行われていることがわかる。しかし、これらの圧縮率（Nc/No）は図3(A)が0.326、図3(B)が0.076、図3(C)が0.025と大きく異なる。これ

(A)

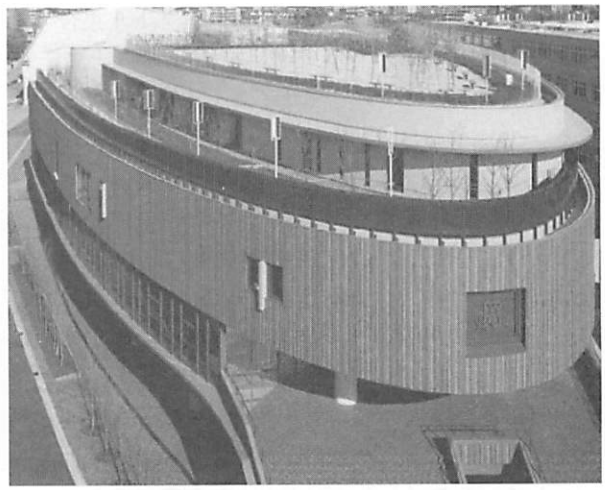


(B)

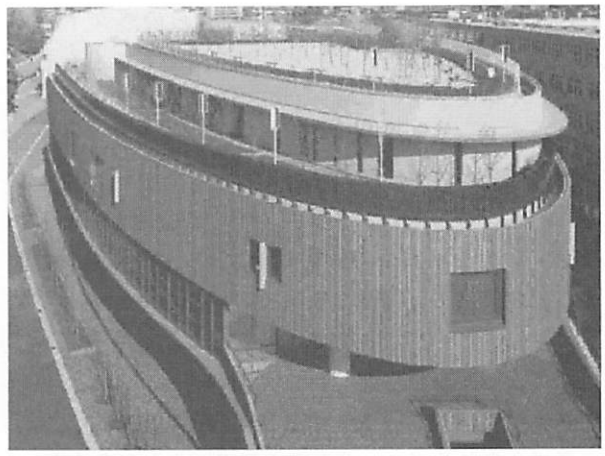


(C)

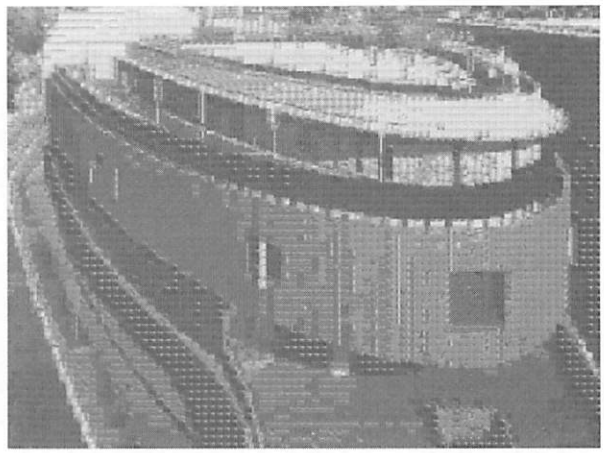
図2　ベクトル量子化の例(A)原画像、(B)16×16 SOMによるベクトル量子化（ブロックサイズ16×16)、(C)同一クラスに属するブロック



(A)

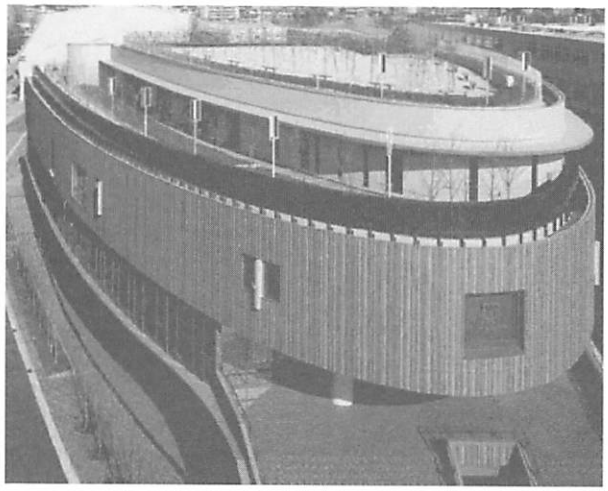


(B)



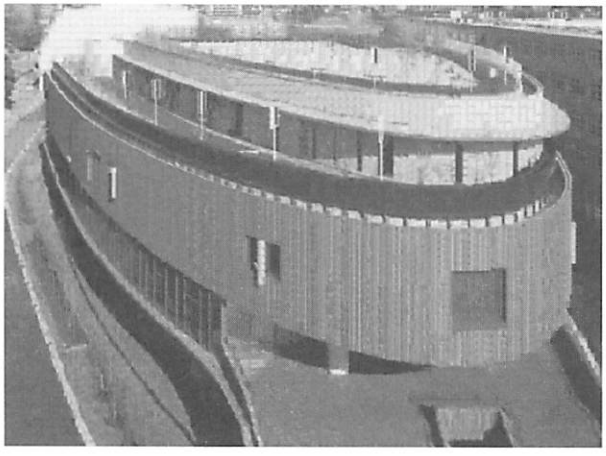
(C)

図3　ブロックサイズを変えた場合の圧縮例(その1)
(A) $M=2, N=32$（圧縮率：0.326、量子化ひずみ：927,534)、(B) $M=4, N=16$（圧縮率：0.076、量子化ひずみ：3,067,592)、(C) $M=8, N=8$（圧縮率：0.025、量子化ひずみ：6,315,800)



(A)

(B)



(C)

図4　ブロックサイズを変えた場合の圧縮例（その2）
(A) $M=2, N=8$（圧縮率：0.188、量子化ひずみ：3,075,443）、(B) $M=4, N=32$（圧縮率：0.131、量子化ひずみ：2,123,787）、(C) $N=8, N=32$（圧縮率：0.233、量子化ひずみ：2,848,798）

は、ブロックのサイズが小さくなると、画像に含まれるブロック数が増え、各ブロックに置かれるパターン番号の個数も増加してしまうためである。その結果、ブロックサイズが小さいと圧縮後の総ビット数も増えることになり、圧縮率が悪化してしまう。また、ここではブロック数とニューロン数（パターンの数）の比を一定にしているので、ブロック数の増加に伴いニューロン数も増えるので、パターン番号を表すビット数が増加するのも圧縮率が悪くなる原因である。

次に圧縮率の差が比較的同じぐらいになるようにして、ブロックサイズの違いによる画質を比較した実験結果を図4に示す。圧縮率は、図4(A)が0.188、図4(B)が0.131、図4(C)が0.233である。ブロックサイズが大きい図4(C)では、量子化ひずみ E は、それほど大きいわけではないが、ブロックノイズが目立ってしまい、他の2枚より見劣りする結果となった。これは、ブロックサイズが大きくなり、ブロック境界部分の誤差が目立ってしまうためである。ブロックサイズを 2×2 とした図4(A)とブロックサイズ 4×4 の図4(B)では、建物のカーブも自然に表現され、それほど違和感は無い。特に、図4(A)は量子化ひずみが(C)より大きいが原画との差は小さく見える。圧縮率、量子化ひずみから判断すると図4(B)のブロックサイズとしては、4×4 がもっとも効率が良い値といえる。

4. ま と め

本文では、自己組織化マップについて説明を行った。そして、自己組織化マップが持つベクトル量子化の能力の画像圧縮への応用例を示した。

自己組織化マップは非常に広範囲な応用が可能な技術で非常に広範囲な分野での応用研究がなされている。また、自己組織化マップの画像圧縮への応用は、そういった研究段階にある応用の1つである。ベクトル量子化の利用は、画像だけでなく音声などの圧縮にも使用可能な技術である[3]。実際の画像圧縮で使われるベクトル量子化には他のアルゴリズムが使用されている。

自己組織化マップを画像圧縮に用いる場合、問題となるのは計算処理である。特にニューロンの数とブロック数に比例して処理時間が長くなる。今回の実験でも結果を得るためまでに2GHz Core2 Duo プロセッサを使用したPCで10分以上要したものもあった。処理速度の問題には、専用ハードウェアの開発が解決法の1つで、筆者の研究室では自己組織化マップのハードウェア実装が研究テーマの1つとなっている。

[1] T. コホネン、(徳高平蔵、岸田 悟、藤村喜久朗 訳)“自己組織化マップ”、シュプリンガー・フェアラーク、東京 (1996)
[2] 徳高平蔵、藤村喜久朗、山川 烈 監修、“自己組織化マップ応用事例集”、海文堂出版 (2002)
[3] Allen Gersho、Robert M. Gray (古井貞熙、小寺 博、田崎三郎、渡辺 裕 訳)“ベクトル量子化と情報圧縮”、コロナ社 (1998)