自己組織化マップ入門
Introduction of Self-Organizing Map

古川 徹生[*1]

Ver. 1.00.00 (2017 年 6 月 3 日)

[*1] 九州工業大学大学院生命体工学研究科　E-mail: furukawa@brain.kyutech.ac.jp

目次

第 1 章

自己組織化マップの概要

自己組織化マップ *(Self-Organizing Map: SOM)* はフィンランドの研究者，T. Kohonen の発明したニューラルネットの一種である．SOM は教師なし学習を行う**位相保存写像** *(topology preserving mapping)* の一種である [1, 2, 3]．高次元の観測データセットに対し，SOM はデータ分布の位相的構造を保存しつつ低次元空間へ写像する．特に 2 次元空間へ写像する場合はデータ分布が地図 (topographic map) のように可視化される．この地図をデータマイニングに用いるのが典型的な SOM の利用法である．

歴史的に見れば，SOM は大脳視覚野における機能地図の自己組織化モデルに由来する．大脳皮質ではさまざまな機能を持つ神経細胞が規則性を持って配置されており，発生・成長の過程で適切な配線が自己組織的に行われる．**自己組織化** *(self-organization)* の名前はここに由来する．これをデータマイニングなど工学的に利用したのが Kohonen の SOM の出発点である．したがって神経系における自己組織的機能分化というサイエンス的側面と，データマイニングなどの工学的側面の 2 つの顔を SOM は持つ．

SOM のアルゴリズムはさまざまな解釈が可能である．ニューラルネットワークの視点では (1) Winner-Take-All による競合，(2) 近傍関数による協調的学習分配，(3) Hebb 則に基づくシナプス荷重の更新という 3 つのプロセスの繰り返し計算と解釈できる [4]．一方，機械学習的視点では広義の EM アルゴリズムと見ることができ，E ステップと M ステップのループを繰り返す [5].

SOM はさまざまなニューラルネットや機械学習の手法と関係している．次元削減による可視化という点では**主成分分析** *(Principal Component Analysis: PCA)* や**多次元尺度法** *(Multidimensional Scaling: MDS)* などが関連する．多様体による高次元データ分布の近似という点では**多様体学習** *(minifold learning)* の一種でもある．一方で SOM は**ベクトル量子化** *(Vector Quantization: VQ)* に位相の概念を組み込んだもの，あるいはデータ分布をグラフ構造でモデル化したものとも解釈できる．また SOM は**潜在変数モデル** *(latent variable model)* でもある．中でも**生成位相写像** *(Generative Topographic Mapping: GTM)*[6] および**ガウス過程潜在変数モデル** *(Gaussian Process Latent Variable Model: GPLVM)*[7] はベイズ的機械学習の観点で再構築した SOM と見ることができる．

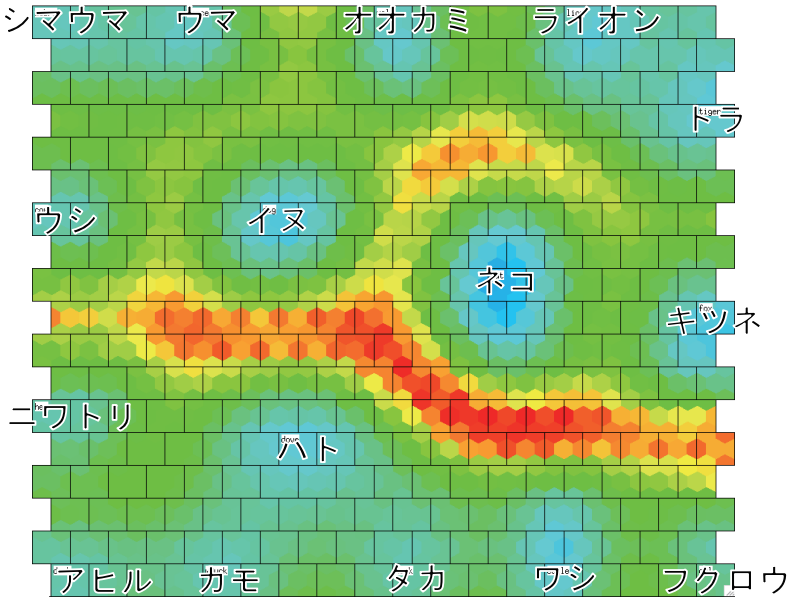

図 1.1　表 1.1 のデータから SOM が生成した動物マップ．

表 1.1　動物データ．

	ハト	ニワトリ	アヒル	カモ	フクロウ	タカ	ワシ	キツネ	イヌ	オオカミ	ネコ	トラ	ライオン	ウマ	シマウマ	ウシ
小さい	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
中ぐらい	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
大きい	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
2 本脚	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
4 本脚	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
体毛	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
ひづめ	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
たてがみ	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
羽	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
狩猟	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
走る	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
飛ぶ	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
泳ぐ	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
家畜/ペット	0	1	1	0	0	0	0	0	1	0	1	0	0	1	0	1
夜行性	0	0	0	0	1	0	0	1	0	0	1	1	1	0	0	0

1.1　マップの具体例

表 1.1 は 16 種の動物の性質を 15 次元ベクトルで表したものである[*1]．このデータを SOM で 2 次元の地形図（topographic map, 以下「マップ」と呼ぶ）にしたものが図 1.1 である．各々の動物は 15 次元空間の 1 点から 2 次元空間の 1 点へ写像された．すなわち 15 次元から 2 次元へと削減されたわけである．

マップ上で隣りあう動物は観測空間でも互いに近いデータベクトルを持つ．すなわち 15 次元空間におけるデータ同士の近隣関係は，おおむね（厳密ではないものの）2 次元

[*1] 文献 [3] p.170 のデータを修正して使用．コホネンのデータでは，ウマとシマウマ，アヒルとガチョウが同じデータベクトルとなり，マップ上で区別できない．そこで「家畜/ペット」「夜行性」を追加した．

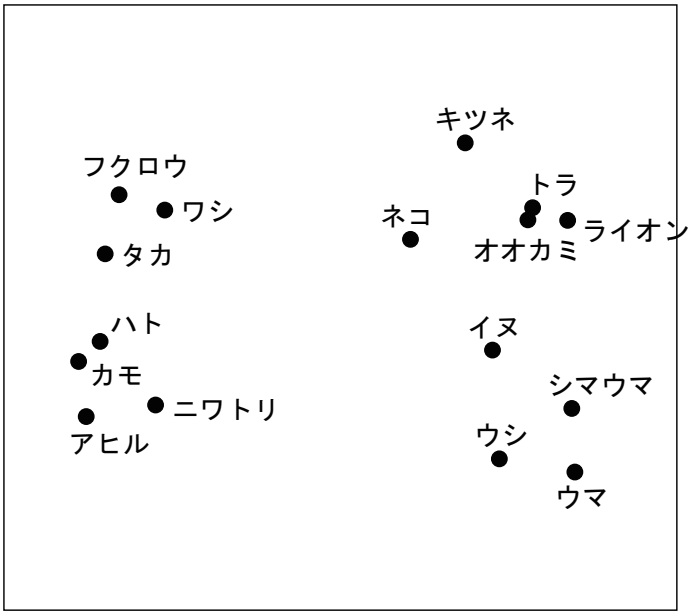

図 1.2　表 1.1 のデータから MDS が生成した動物マップ．SOM と異なり，哺乳類と鳥類のクラスタ境界はデータ点の存在しない空白領域として示される．

マップでも反映されている．たとえばウマとシマウマ（マップ左上），アヒルとカモ（マップ左下）は互いに似た性質の動物であることを示唆する．ライオンとトラ，ワシとタカなども同様である．

SOM の場合，高次元空間から低次元空間への縮尺は一定ではない．一般にデータ密度が高い領域はマップ上で拡大され，密度の低い部分は縮小される[*2]．いわば人口の密集する都市部は高縮尺の精密な地図を作り，人口の少ない山間部は縮尺を下げて粗い地図を作るようなものである．マップで青〜緑で色づけされた箇所は拡大率の大きい領域であり，元の空間におけるデータ密度は一般に高い．一方，赤い箇所は拡大率の小さい領域であり，一般にデータ密度は低い．地形図的に言えば，人口密度の高い領域が平野部，人口密度の低い領域が山間部になる．たとえば図 1.1 ではキツネからトラとフクロウがほぼ等距離にある．しかし緑で示されたキツネ－トラ間の実際のデータ間距離は短く，赤い領域をはさむキツネ－フクロウ間は距離が長い．すなわちキツネから見て同じ平野に住むトラの方が，山向こうに住むフクロウよりも似た動物であると言える．また哺乳類と鳥類の間に山脈が横たわることから，両者は異なるクラスタを形成することも読み取れる．このようなマップの色づけを U-matrix と呼ぶ．U-matrix を使うことでマップを地図のように読めるため，専門家でなくても容易にデータ分析ができる．なお U-matix を 3 次元的に表示するといっそう地形図らしくなる（図**??**）．

SOM のマップはデータ間距離を保存しないため，データ解析する際は注意が必要である．これは距離関係も保存しようとする多次元尺度法 (MDS) と対照的である．図 1.2 同

[*2] マップ上でデータ点がほぼ均等に分布する傾向があるが，しかし完全に均等というわけでもない．これを magnification factor と呼ぶ．データ密度が均等となるように（すなわち事前分布に等しくなるように）した場合，必ずしも写像が滑らかになるとは限らない．一方で，距離が保存されるようにすると写像は滑らかさになるが，潜在変数の事前分布からは離れてしまう．潜在変数の事前分布と写像の事前分布（すなわち滑らかさ）の双方に関して折り合いをつけた点が解となる．

じ動物データを MDS で 2 次元マップにしたものである．SOM の場合と異なり，MDS ではクラスタ境界がデータ点の存在しない空白領域として現される．

1.2　歴史

SOM は 1980 年初頭にコホネンによって提案された [1]．SOM はもともと大脳視覚野における機能地図の自己組織化現象を説明するための数理モデルに由来する．

網膜から伸びた視神経は外側膝状体を中継して大脳の一次視覚野 (V1) に投射される．この神経投射には位相保存性があることが知られていた．すなわち網膜上で隣り合う 2 点に対応する視神経は，V1 でも隣り合うように投射されるのである．これを**レチノトピー** *(retinotopy)* という．同様の位相保存性は体性感覚野や聴覚野でも知られる．発生段階では比較的ランダムに近い視神経投射が，誕生後に光刺激を受けることで次第に整理されることで自己組織的にレチノトピーが形成される．この自己組織現象を数理モデル的に解明することが研究者の興味の対象だった [8].

一方，V1 ではさまざまな視覚的特徴に反応するニューロンの存在が知られる．たとえば特定の傾きを持つ線分に反応するニューロンや，特定方向への動き，特定の色に反応するニューロンなどである．これらのニューロンも隣接するニューロンとは互いに似た特性を持つ．たとえば水平な線分に反応するニューロンの近くには，わずかに傾いた線分に反応するニューロンが存在する．これは視覚的特徴に基づく神経系の自己組織化現象である．この現象の数理モデル的理解も試みられた [9].

2 つの自己組織化現象は互いに似ているものの，本質的には異なる性質を持つ．前者はデータの信号ラインに関する自己組織化であり，後者はデータの特徴量に関する自己組織化である．Kohonen は前者を Type-1，後者を Type-2 の自己組織化現象と呼んで区別している [10]．そして Kohonen の SOM は Type-2 の自己組織化モデル，すなわち入力の特徴ベクトルの自己組織的分化に分類される．SOM 以外にもさまざまな自己組織化モデルがあったため，他と区別して Self-organizing feature map (SOFM) や Kohonen's map などとも呼ばれた．そして SOM のもっとも大きなエポックは，神経系における自己組織化現象の数理モデルをデータの特徴にもとづく教師なし学習という工学的応用へと転換したことである．

視覚神経モデルから出発したため，初期の SOM には神経モデルの特徴を色濃く備えていた．モデルニューロンを並べたアーキテクチャ，Winner-Take-All によるニューロン間競合，Hebb 則に基づくシナプス荷重（参照ベクトル）の逐次的学習（オンライン学習）などである．またランダムな初期状態から自己組織的にマップが形成されることも重要なポイントであった．しかし工学応用の観点からは神経回路モデルにこだわる必要がないため，より安定した学習結果が得られるようにアルゴリズムの改良が行われた．また完全にランダムな初期状態から始めるのではなく，線形モデル近似による初期状態からスタートすることで学習効率を上げる方法も採用された．最終的な SOM のスタンダードと言えるのがバッチ型 SOM である．バッチ型 SOM は旧来のオンライン学習型 SOM と比べ学習が高速かつ安定している．コホネン自身も実応用にはバッチ型を推奨しており，旧来のオンライン型 SOM は神経回路の自己組織化モデルという文脈で捉えるべきと述べている [11].

SOM はシンプルなアルゴリズムであるため，膨大なバリエーションが提案されてきた

[12, 13]．一方で 1990 年代以降は統計的機械学習の視点から SOM アルゴリズムの再導出や一般化が行われるようになった [14, 15, 5].

SOM の研究には複数の異なる立場や目的が混在することには注意が必要である．本ドキュメントでは工学的視点での SOM について述べ，神経回路の自己組織化モデルとしての SOM は取り扱わない．しかしながら、自己組織化の数理モデルという視点もまた重要であることには違いない．

1.3 アーキテクチャとアルゴリズム

1.3.1 アーキテクチャと用語

SOM で用いられる用語は統一されていない．そこで SOM のアーキテクチャを説明しながら，本ドキュメントで用いる用語を定義する．

Self-Organizing Map の日本語名は自己組織化マップもしくは自己組織化写像であり，両者ともよく使われる[*3]．英語では Self-Organizing Feature Map (SOFM) や Kohonen map/net と呼ばれることもあったが，最近は SOM が一般的である．

図 1.3 は SOM のアーキテクチャの概念図である．SOM の目的は高次元データを低次元空間へ写像することであり，本稿では高次元側を**観測空間** *(observable space)*，低次元側を低次元側を**潜在空間** *(latent space)* と呼ぶ[*4]．他の呼び方として前者はデータ空間，入力空間，後者はマップ空間，出力空間などがある．

潜在空間は 2 次元正方形や長方形がよく用いられるが，それ以外の形状であってもかまわない．潜在空間は通常，正方格子もしくは六方格子状に分割され，それぞれに**ノード** *(node)* が配置される（ユニット，グリッド，ニューロンとも呼ばれる）．ノードは潜在空間に固定されており，動くことはない．一方，各ノードは観測空間の一点を指すベクトル値を保持しており，参照ベクトルと呼ぶ（他にもコードブックベクトル，シナプス荷重などと呼ばれる）．参照ベクトルは学習によって更新される変数である．

1.3.2 SOM のタスク

SOM の目的は観測データをマップ空間へ写像してデータ分布の地図を作ることである．しかしながら，実際に SOM が学習するのはマップ空間からデータ空間への逆向き写像である．今，観測空間を $\mathcal{X}$，潜在空間を $\mathcal{Z}$ とする．SOM の真の目的はデータ分布をうまく説明する連続単射写像 $f : \mathcal{Z} \to \mathcal{X}$ を推定することである．すなわち観測データセット $X = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ に対し，$\mathbf{x}_n \simeq f(\mathbf{z}_n)$ であるような滑らかな写像 f と潜在変数セット $Z = (\mathbf{z}_1, \ldots, \mathbf{z}_N)$ を推定することが SOM の目的である．

今，第 k ノードのマップ空間での固定座標を $\boldsymbol{\zeta}_k \in \mathcal{Z}$，参照ベクトルを $\mathbf{y}_k \in \mathcal{X}$ とする．するとマップ空間の格子点 $\{\boldsymbol{\zeta}_1, \ldots, \boldsymbol{\zeta}_K\}$ に対して写像 f を $\mathbf{y}_k = f(\boldsymbol{\zeta}_k)$ と離散表現できる．f は滑らかと仮定するので，ノード数が多ければ十分な精度が得られる．SOM の学習とは適切な f が得られるように参照ベクトル $\mathbf{y}_k$ を更新することに他ならない．

[*3] Map という語には「地図 (topographic map)」と「写像 (mathematical map)」の 2 つの意味があるため訳が難しい．

[*4] 「潜在空間」「観測空間」は機械学習的用であり，SOM の研究者の中では一般的な用語ではない．

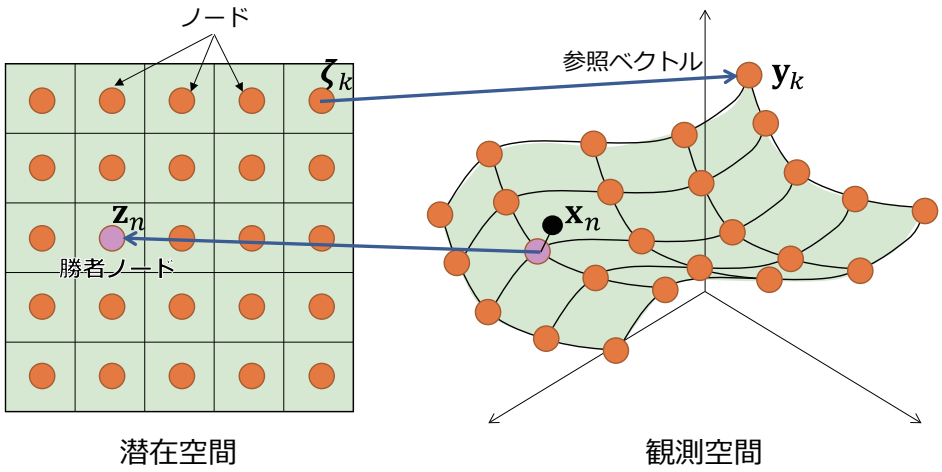

図 1.3　SOM の概念図．潜在空間（マップ空間）に等間隔に並んでいるのがノードである．各ノードは観測空間（データ空間）の 1 点を指す参照ベクトルを保持する．データ $\mathbf{x}_n$ に対してもっとも近い参照ベクトルを持つノードが勝者ノード (Best Matching Unit: BMU) になる．

1.3.3　アルゴリズム

Heykin の教科書では，SOM の学習アルゴリズムが**競合過程** *(Competitive process)*, **協調過程** *(Cooperative process)*, **適応過程** *(Adaptive process)* の 3 過程の繰り返し計算であると説明している [4]．本稿でもそれに倣ってアルゴリズムを説明する．

Step 0 (初期化):　まず参照ベクトルに初期値 $\mathbf{y}_k(0)$ を与える（後述）．そして学習回数を $t = 1$ として Step 1 に進む．

Step 1 (競合過程):　各観測データ $\mathbf{x}_n$ $(n = 1, \ldots, N)$ に対し，もっとも近い参照ベクトルを持つノードを勝者ノード (Best Matching Unit: BMU) とする．すなわち BMU のノード番号 k_n^* をすべての $\mathbf{x}_n$ に対して求める．

$$\forall n, \qquad k_n^*(t) = \arg\min_k \|\mathbf{x}_n - \mathbf{y}_k(t-1)\|^2 \tag{1.1}$$

また潜在変数の推定値 $\mathbf{z}_n(t)$ は潜在空間における BMU の位置ベクトル $\boldsymbol{\zeta}_{k_n^*(t)}$ として与えられる．

Step 2 (協調過程):　各勝者ノードが近傍ノードに分配する学習量 R_{kn} を計算する．

$$\forall n, k \qquad R_{kn}(t) = h(\mathbf{z}_n(t), \boldsymbol{\zeta}_k; \sigma(t)) \tag{1.2}$$

$h(\mathbf{z}, \boldsymbol{\zeta})$ は近傍関数と呼ばれており，BMU に近いノードほど多くの学習量の配分を受ける．近傍関数としてはガウス関数

$$h(\mathbf{z}, \boldsymbol{\zeta}; \sigma) = \exp\left[-\frac{1}{2\sigma^2}\|\mathbf{z} - \boldsymbol{\zeta}\|^2\right] \tag{1.3}$$

を用いることが多い．$\sigma(t)$ は近傍領域の大きさを決めるパラメータであり，近傍半径と呼ばれる．$\sigma(t)$ は学習回数 t とともに単調減少する（後述）．

Step 3 (適応過程): 全参照ベクトルを観測データの重みつき平均になるよう更新する．

$$\forall k, \qquad \mathbf{y}_k(t) = \frac{\sum_{n=1}^{N} R_{kn}(t)\mathbf{x}_n}{\sum_{n'=1}^{N} R_{kn'}(t)} \tag{1.4}$$

そして $t := t + 1$ として Step 1 に戻る．これを学習が収束するまで繰り返す．

最終的に得られた $\mathbf{z}_n$ がマップ上での各データの位置となる．

SOM のアルゴリズムは広義の EM アルゴリズム (Expectation Maximization algorithm: EM algorithm) と見ることができる [5]．その場合，競合過程が E ステップ，適合過程が M ステップになる．協調過程を E, M のどちらのステップに入れるかは，アルゴリズムの解釈による．

1.3.4 行列を用いた表現

SOM のアルゴリズムを行列を用いると以下のように書くことができる．まずデータ行列を $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N) \in \mathbb{R}^{D\times N}$*5，写像行列を $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_K) \in \mathbb{R}^{D\times K}$ と定義する．

競合過程では勝者行列を $\mathbf{B} = \left(\delta_{k,k_n^*}\right)$ と定義する．$\delta_{kk'}$ はクロネッカーデルタである．次の協調過程では近傍行列 $\mathbf{H} = (H_{kk'}) \in \mathbb{R}^{K\times K}$ を

$$H_{kk'} = h(\boldsymbol{\zeta}_k, \boldsymbol{\zeta}_{k'}; \sigma(t)) \tag{1.5}$$

と定義し，行列 $\mathbf{R}$ を

$$\mathbf{R} = (R_{kn}) = \mathbf{HB} \tag{1.6}$$

と求める．また $G_k = \sum_{n=1}^{N} R_{kn}$ とし，対角行列

$$\mathbf{G} = \mathrm{diag}(G_k) = \begin{pmatrix} G_1 & & 0 \\ & \ddots & \\ 0 & & G_K \end{pmatrix} \tag{1.7}$$

も求めておく．最後に適合過程では写像行列を

$$\mathbf{Y}^{\mathrm{T}} = \mathbf{G}^{-1}\mathbf{RX} = \mathbf{G}^{-1}\mathbf{HBX}^{\mathrm{T}} \tag{1.8}$$

として計算する．

1.3.5 初期化

計算に先立って全参照ベクトルを初期化する必要がある．研究目的の場合，しばしば乱数を用いて初期化する．これは SOM の自己組織化能力を検証するためであり，実応用には適さない．

実応用の場合，PCA を用いてマップ空間の次元数と等しい線形部分空間を求め，そこへ等間隔に参照ベクトルを初期配置する．この場合，各主成分方向への標準偏差（特異値）を各辺長とする長方形のマップ空間を用いるとよい．

潜在変数 $\mathbf{z}_n$ の初期値を与え，協調過程 (Step 2) から開始してもよい．この場合も乱数による初期化と PCA による初期化の双方が可能である．

*5 データ行列を $D \times N$ 行列で定義する場合と，転置をとって $N \times D$ 行列で定義する場合がある．統計では一般に後者を用いる．本ドキュメントでは前者で定義し，必要に応じて $\mathbf{X}^{\mathrm{T}}$ のように転置を明示する．

1.3.6　近傍半径のスケジュール

近傍半径 $\sigma(t)$ は学習回数 t とともに単調減少する．マップの一辺の長さを l とすれば，初期値 σ_0 は $l/2$ 程度とする．$\sigma(t)$ は一次関数もしくは指数関数で減少させ，あらかじめ定めておいた最小値 $\sigma_{\min}$ に達したところで縮小をストップする．$\sigma_{\min}$ はデータ数とノード数の比に依存し，どのノードに対しても半径 $\sigma_{\min}$ 以内に数個ないし十数個以上の勝者ノードが存在するように設定する．典型的な値は $\sigma_{\min} \sim l/10$ 程度である．また τ は近傍半径の縮小スピードを決めるパラメータである．τ が小さいほど少ない学習回数ですむが，局所解に陥りやすくなる．学習回数のめやすは数十回程度である．

一次関数で縮小する場合は

$$\sigma(t) = \max\left[\sigma_0\left(1 - t/\tau\right), \sigma_{\min}\right] \tag{1.9}$$

指数関数で縮小する場合は

$$\sigma(t) = \max\left[\sigma_0 \exp\left(-t/\tau\right), \sigma_{\min}\right] \tag{1.10}$$

とする．ここで $\max[a, b]$ は a, b のうち大きい方を返す関数である．また τ は縮小のスピードを決める時定数で，大きいほどゆっくり縮小する．

1.4　マップの彩色法

得られたマップを色付けして解析に役立てることができる [16]．代表的な方法として component plane と U-matrix がある．

Component plane は，参照ベクトルの着目する成分の大きさで色づけする方法である．今，D 次元データベクトルの第 d 成分に着目するとしよう．このとき，y_{kd} の大きさをグレースケール等で表現するのが component plane である．

一方，U-matrix は各ノードの参照ベクトルが近傍ノードと異なる度合いで色づけする方法である [17]．一般にクラスタ境界では写像の勾配が急峻になるため，参照ベクトルの差を見ることでクラスタ分析が可能になる．図 1.1 は U-matrix 法で彩色した例である．

1.5　学習理論

SOM の学習アルゴリズムは以下の目的関数から導出できる．

$$F = -\sum_{n=1}^{N}\sum_{k=1}^{K} h(\mathbf{z}_n, \boldsymbol{\zeta}_k)\,\|\mathbf{x}_n - \mathbf{y}_k\|^2 \tag{1.11}$$

目的関数 F を最大化するように $Y = (\mathbf{y}_k)$ と $U = (\mathbf{u}_n)$ を交互推定すれば（勝者決定が少し異なるものの）上述のアルゴリズムになる．このアルゴリズムは広義の EM アルゴリズムとみなせる [5]．

今，$\mathbf{Y} = (\mathbf{y}_k)$ が与えられた状況下で F を最大化する 潜在変数 $\mathbf{z}_n$ を求めて見る．すると

$$\mathbf{z}_n = \arg\min_{\boldsymbol{\zeta}_k} \sum_{k'=1}^{K} h(\boldsymbol{\zeta}_k, \boldsymbol{\zeta}_{k'})\,\|\mathbf{x}_n - \mathbf{y}_{k'}\|^2$$

となる．これは近傍関数で局所平均化した二乗誤差である．もし近傍半径が十分小さければ，

$$\sum_{k'=1}^{K} h(\boldsymbol{\zeta}_k, \boldsymbol{\zeta}_{k'}) \|\mathbf{x}_n - \mathbf{y}_{k'}\|^2 \simeq \|\mathbf{x}_n - \mathbf{y}_k\|^2$$

となる．これは SOM の競合プロセス (1.1) と同じである．

一方，潜在変数 $\mathbf{Z} = (\mathbf{z}_n)$ が与えられた状況下で F を最大化する参照ベクトル $\mathbf{y}_k$ を求めると，

$$\frac{\partial F}{\partial \mathbf{y}_k} = \sum_{n=1}^{N} h(\mathbf{z}_n, \boldsymbol{\zeta}_k)(\mathbf{x}_n - \mathbf{y}_k) = 0 \tag{1.12}$$

より

$$\mathbf{y}_k = \frac{\sum_{n=1}^{N} h(\mathbf{z}_n, \boldsymbol{\zeta}_k)\mathbf{x}_n}{\sum_{n'=1}^{N} h(\mathbf{z}_n, \boldsymbol{\zeta}_k)} \tag{1.13}$$

となる．これは適合過程 (1.4) と同じである．

1.6 SOM とは何か

SOM はさまざまな視点からその機能を解釈できる．第一の視点は次元削減による可視化である．この観点では PCA や MDS，サモンマップなどが比較対象になる．またより SOM と密接な手法としては，**主曲線法** *(Principal curve)* や**弾性ネット** *(Elastic net)* などがある．

高次元のデータ分布を多様体でモデル化するという視点で見れば，多様体学習の一種と見ることもできる．多様体の代表的な手法としては，*Isometric feature mapping: ISOMAP* や *Locally Linear Embedding (LLE)* などがある．

SOM はベクトル量子化に位相構造を導入したものとみることもできる．SOM のアルゴリズムから協調過程をなくすと，k-means 法そのものになる．また学習量の分配に近傍関数を用いた Fuzzy k-means 法と見ることもできる．また高いベクトル量子化能力を持つニューラルガス (Neural Gas: NG) は，近傍関係が動的に変わる SOM とみなすことができる．

SOM はさらに，高次元データ分布をグラフ構造でモデル化したものと見ることもできる．グラフ構造を動的にすると，さらに柔軟性を与えることができる．これは SOM から派生した一領域となっており，Growing NG や Evolving SOM などさまざまなバリエーションが提案されている．

機械学習の観点では，潜在変数モデルの一種と見ることができる．その観点では**ガウス混合分布** *(Gaussian Mixture Model: GMM)*，**因子分析** *(Factor Analysis: FA)* などとも関連する．特に**生成位写像** *(Generative Topographic Mapping: GTM)* [6] や**ガウス過程潜在変数モデル** *(Gaussian Process Latent Variable Model: GPLVM)* [7, 18] は SOM の理論的理解をする上で重要である．

1.7　リソース

SOM を利用する際は，以下のフリーパッケージを利用すると便利である．ひとつは SOM_PAK [19] であり，もうひとつは MATLAB の SOM Toolbox [20] である．これらは SOM の学習だけでなく，U-matrix 法なども利用できる．

文献としてはコホネン自身の書籍 [2] および邦訳 [3] がある．ただし内容的にはやや古い．より最近のものとしては文献 [11] がよくまとまっている．SOM に関する論文は膨大であるが，文献 [12, 13] にそれらのリストがある．

第 2 章

SOM の運用

SOM は教師なし学習であるため，得られたマップが正しいかどうかを検証する手段がほとんどない．これは実応用に際して大きな問題がある．実際，得られるマップはデータの前処理等に依存するため，実運用に際しては注意が必要である．本章では SOM の性質を理解しながら，運用上の注意点について述べる．

望ましいマップが得られない原因はさまざまである．たとえば以下のような理由が考えられる．(1) そもそも SOM が苦手とするデータ分布の場合 (2) データ分布の位相と潜在空間の位相が一致しない場合 (3) データの前処理が適切でない場合 (4) ユーザーが暗黙に望むメトリックと SOM が用いるメトリックが一致しない場合 (5) はずれ値がある場合 (6) SOM のパラメータが不適切な場合 (7) 局所最適解に陥った場合．これらすべてを解決することは不可能であるものの，SOM を慎重に運用することでリスクをかなり回避できる．

2.1　学習がうまくいかないケース

2.1.1　データ分布の形状に起因する場合

ある種のデータ分布に対して SOM はうまく学習できない．たとえばヘアピン状に急に折り返すデータ分布や螺旋状のデータ分布はうまく学習できない．これらに共通する特徴は，観測空間の同じ領域に多様体が複数回訪れることである．そのような場合，データに対する勝者の割り当てがうまくいかないことが多い．

ヘアピン多様体や螺旋多様体などの学習困難については，論点が 2 つある．第 1 の論点は潜在変数を点推定することの限界である．SOM では勝者ノードの潜在空間における座標が潜在変数の推定値となる．各データに対して勝者はただひとつに決まるが，これは最尤推定に相当する．潜在変数の事後分布がピーク値をひとつしか持たない単峰形分布の場合は，最尤推定でも悪くない結果を出す．しかしヘアピン多様体や螺旋多様体の場合，潜在変数の事後分布は複数のピーク点を持つ多峰分布になる．このような場合は最尤推定値のみを用いたアルゴリズムに限界が生じる．

第 2 の論点は，データ分布の解釈が多義的であるということである．たとえばヘアピン多様体の場合，ノイズが極めて小さいヘアピン多様体という解釈の他に，ノイズの大きい直線と解釈することもできる．すなわち最適解が複数存在することになる．これら 2 つの論点は SOM に限らず，次元削減や多様体学習に共通する普遍的問題である．

なおこのような特殊な形状を持つデータ分布が，実応用でも生じるかどうかは別の問題

である．

■**演習**　1 次元潜在空間の SOM を作成し，ヘアピン多様体や螺旋多様体の学習がうまくいかないことを試してみる．

2.1.2　データ分布と潜在空間が位相同型でない場合

データ分布と潜在空間が位相同型でない場合，当然のことながら適切な位相同型写像を推定することはできない．しかし現実のデータにはノイズが含まれており，さらに観測データの数も有限であるので，データ分布の真の位相的構造を知ることは一般に不可能である．

しかしながら，もし位相が一致しない場合に SOM はどのようなマップを生成するかを知っておくことは重要である．なぜなら，実データに対して SOM が生成したマップを見ることで，位相が一致していない可能性をある程度知ることができるからである．

■**演習**　1 次元潜在空間の SOM を用いて，2 次元に分布するデータをモデル化したらどのような写像が得られるかを確認してみる．

また 2 次元潜在空間の SOM を用いて，3 次元に分布するデータや 1 次元に分布するデータを与えたときにどのようなマップが得られるかを確認してみる．

2.1.3　潜在空間の縦横比が一致しない場合

データ分布が一方向に長く分布する場合，それに合わせて長方形の潜在空間を使うことが望ましい．もしデータ分布が比較的線形モデルに近い場合，データ行列の特異値を求めることで潜在空間の縦横比を求めることができる．

■**演習**　一方向に長く分布するデータを人工的に生成し，正方形の潜在空間を持つ SOM でマップを生成してみる．また潜在空間の縦横比をさまざまに変えてマップを比較する．特に特異値を用いた場合にどのような結果が得られるかを確認してみる．

2.1.4　データの前処理が適切でない場合

観測データをそのまま SOM に与えただけでは，望ましいマップが作られない．例として身体測定の結果（身長，体重，BMI 等）を SOM でマップする場合を考えてみよう．もし身長の単位がメートルで体重が kg だとすると，身長の誤差はおおむね 10^{-1} 程度であり，体重の誤差は 10^{1} 程度となる．したがって勝者決定に際しては体重の違いが支配的となる．しかし，もし身長の単位がミリメートル単位だとすると，身長の誤差はおおむね 10^{2} 程度となり，今度は身長の差異が勝者決定に対して支配的になる．単位の取り方を変えただけで SOM が生成するマップが変わるのである．

このような問題を回避するため，通常はデータベクトルの成分ごとに平均ゼロ，分散 1 になるように前処理をするのが一般的である．こうすれば，どの成分もおおねね同程度のスケールで分布するからである．

2.1.5 メトリックが不一致の場合

平均ゼロ，分散 1 の前処理は一般に妥当とみなされる．しかしどんな場合でも妥当であるとは言えない．たとえばある学校において，教師が生徒たちの能力を SOM で分析する場合を考えてみよう．もし身体能力に関するデータ項目（100m 走の成績など）が大半を占め，学力のデータがわずかしかない場合，SOM は明らかに身体能力にウェイトを置いたマップを生成する．しかし，もしユーザーが身体能力と学力能力を同程度に考慮したマップを望むのであれば，身体能力に関する重みを下げる必要がある．

ベクトル間の距離の測り方などを一般にメトリックと呼ぶ．ユーザーは多くの場合，好ましいメトリックを暗黙に想定している．しかし SOM は与えられたデータベクトルのユークリッド距離で勝者を決定する．両者が不一致の場合，ユーザーの期待する結果は得られない．

生徒の能力の場合，身体能力と学力能力というグループ分けが可能であった．しかし現実には，そのようなグループ分けもわからない場合も多い．たとえば会社四季報のデータを用いて上場企業のマップを作る場合を考えると，どのようなメトリックが望ましいかを知ることはほとんど不可能であることが想像できるだろう．これは教師なし学習にとって普遍的な問題である．

「何をもって誤差を測るか」という問題は他にもある．基本的な SOM アルゴリズムでは二乗誤差を用いて勝者決定を行うが，バイナリデータに対して二乗誤差が本当に適切かという疑問も生じるだろう．実際，機械学習の視点で見れば，二乗誤差はガウス分布に従うデータの対数尤度に相当する．したがってバイナリデータに対しては二乗誤差ではなく，ベルヌーイ分布の対数尤度を使う方がより妥当と言える．この問題に関しては，データタイプに応じて SOM のアルゴリズムを設計しなおすことで解決できる．

■**演習**　動物データにおいて，特定のベクトル成分を重複させた場合，動物マップがどのように変わるかを調べて見る．たとえば「狩りをする」の項目が 5 回現れるようにデータを改変する，等である．

2.1.6 はずれ値がある場合

データが本来分布するはずの多様体から大きく離れたデータ点，すなわち**はずれ値** *(outlier)* が起因してマップが崩れることもある．

はずれ値に対してロバストにするには，ガウス分布の対数尤度（すなわち二乗誤差）の代わりに，t 分布の対数尤度を使うことで対応できる．

■**演習**　はずれ値を持つデータを追加することで，マップがどれくらい影響を受けるか調べてみる．

2.1.7 SOM のパラメータが不適切な場合

学習に際して設定する必要があるパラメータはノード数，潜在空間の縦横比，近傍関数の初期値 σ_0 と最終値 σ_{mi}，および時定数 τ である．このうち，ノード数は解像度に影響を与えるが，学習結果にはあまり影響を与えない．潜在空間の縦横比についてはすでに述

べたとおりである．

近傍関数の初期値 σ_0 も，最初にある程度大きい値にしておけばあまり影響はない．一方，最終値 $\sigma_{\min}$ はある程度影響を与え，小さくしすぎると過学習が生じる．しかし $\sigma_{\min}$ についてはデータ数から見当をつけることができるため，あまり問題を生じない．

時定数 τ については，小さくしすぎると初期状態のままマップが形成されてしまい，適切なマップが作られないことがある．しかし，τ を極端に小さくしすぎない限りはあまり問題にならない．

SOM は全般にパラメータ設定に対して鈍感で，適切なパラメータの設定方法さえ知っていれば，過度に神経質になる必要はない．

■**演習** $\sigma_0, \sigma_{\min}, \tau$ をさまざまに変えて，マップがどのように生成されるかを調べてみる．

2.1.8 局所最適解に陥る場合

SOM の目的関数 (1.11) は局所最適解を持つため，得られたマップは必ずしも大域的最適解である保証はない．

局所最適解に陥る原因は 3 つ考えられる．(1) データ分布の多様体がヘアピン多様体や螺旋多様体などのように学習の困難な形状をしている場合．(2) 近傍半径を縮小する時定数 τ が小さすぎる場合．(3) 大域的最適解に近い準最適解が複数存在する場合．(1), (2) についてはすでに述べたとおりである．(1) は根本的に解決が困難である．(2) については τ を大きくすることで解決できる．問題は (3) の場合である．言い換えるならば「同程度にもっともらしいマップが複数存在する」状況と言える．

ひとつの方法として，異なる初期状態から複数回マップを求め，目的関数 (1.11) を最小化する解を選ぶという方法である．

全ノードの参照ベクトルがすべて同じ値をとってしまう場合もある．これは直前のステップにおいて，1 個のノードがすべてのデータに対して勝者になった場合に生じる．一般にこの現象は学習開始直後に発生する．その原因は，参照ベクトルの初期分布とデータ分布が一致しておらず，勝者が均等に割り振られないことによる．特にデータの規格化や参照ベクトルの初期化にバグがあると発生する．勝者ノードによる初期化法ではこの問題が発生しない．

■**演習** 動物データ（表 1.1）も複数の異なるマップが得られる．実際に F を評価し，F を最大化するマップを調べてみる．

■**演習** 参照ベクトルの初期分布をデータ分布と異なるようにすると，1 個のノードが全データに対し勝者になり，マップが 1 点に退化することを確認する．

2.2 SOM の適切な運用

SOM は教師なし学習であるため，得られたマップの「正しさ」を平均二乗誤差などの数値として評価することが難しい．そのため得られたマップが適切かどうかを知ることは重要である．

2.2.1 適切な前処理を行う

データベクトルを成分ごとに平均ゼロ，分散 1 に規格化するのは必ず行う．さらにデータベクトルがある構造を持つとき（たとえば生徒の能力が身体能力と学力能力のように複数のブロックを形成する場合）は，その構造に基づいて重みを調節してもよい．

2.2.2 観測空間での分布を見る

観測空間においてデータが実際にどう分布されるかを見るのはひとつの方法である．もし観測空間の次元が 3 次元以下ならば，データ分布を直接見ることができる．しかし SOM は高次元データの可視化が目的であり，観測空間の次元は通常，とても大きい．この場合は PCA を用いて 3 次元に次元削減してみるとよい．

PCA の計算方法はいくつか存在するが，代表的な方法として特異値分解を用いるやり方がある．今，データ行列を $\mathbf{X} \in \mathbb{R}^{D\times N}$ とする．これを**特異値分解** *(Singular Value Decomposition: SVD)* をかけた結果を

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}} \tag{2.1}$$

とする．ここで $\mathbf{U} \in \mathbb{R}^{D\times I}, \mathbf{V} \in \mathbb{R}^{N\times I}$ はそれぞれ各列ベクトルが直交する行列，$\mathbf{\Sigma} \in \mathbb{R}^{I\times I}$ は $\sigma_1, \ldots, \sigma_I$ を対角成分とする対角行列である．なお σ_i は特異値であり，$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_I$ となるように順番づけする．また $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_I)$ とし，$\mathbf{u}_i$ を第 i 固有成分と呼ぶ．また $\mathbf{U}_{123} = (\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3)$ は上位 3 主成分の作る部分空間の正規直交基底となる．そこで

$$\tilde{\mathbf{x}}_{123} = \mathbf{U}_{123}^{\mathrm{T}}\mathbf{x} \tag{2.2}$$

と変換することで，データ分布の主要な 3 次元を取り出すことができる．この 3 次元空間にデータ点をプロットすると，おおざっぱなデータ分布の構造を見ることができる．できることなら主要 3 成分だけでなく，$\mathbf{U}_{124}$ や $\mathbf{U}_{135}$ などのように第 4，第 5 成分も含む 3 次元空間も取ってみる．

なお $\hat{\mathbf{Z}} \triangleq \mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}, \hat{\mathbf{Z}} = (\hat{\mathbf{z}}_1, \ldots, \hat{\mathbf{z}}_N)$ とし，$\hat{\mathbf{z}}_n$ を潜在変数 $\mathbf{z}_n$ の初期値して用いるのが PCA による初期化である．

SOM の学習においても，写像によって作られた多様体がどのようにデータ分布をモデル化するかをプロットし，観察すると良い．

2.2.3 適切な潜在空間のデザインを行う

データ分布の位相が潜在空間の位相と一致しない場合は，潜在空間の形状を変える必要があるかもしれない．また正方形よりも長方形の方が適切かもしれない．潜在空間の縦横比を特異値で決める方が良い結果をもたらすこともある．

一部の研究者は，端点の存在しない球面状やトーラス状の潜在空間が良いと主張する．これも元のデータ分布の位相構造に依存する問題である．観測空間で歪んだ球面のようなデータ分布をしているのであれば，球面状の潜在空間を用いる意味がある．

2.2.4　初期化を変えてみる

もし SOM の計算を 1 回しか行わないのであれば，上述のように PCA による初期化が好ましい．しかし SOM の学習を何度も行えるのであれば，ランダムな初期状態からマップをいくつも生成し，一貫した結果が得られるかどうか確認するのも意味がある．また目的関数 (1.11) を評価してもっとも大きな F を与えるマップを採用する方法もある．

2.2.5　計算過程をモニターする

パラメータを決める段階では，計算結果だけでなく計算過程も必ずモニターするとよい．モニターする際は潜在空間と観測空間の双方を表示し，計算ステップごとにどのように写像と潜在変数が更新されているかを確認する．もし計算に問題がある場合は，必ず怪しい挙動が見つかるだろう．

大規模なデータで計算時間がかかる場合は，データのサブセットを作ってバラメータ等をチューニングし，全データによる計算はモニターせずに行えばよい．

2.2.6　他の次元削減手法との比較検証

実応用においては，高い信頼性が必要な場合もある．そのような場合は他の類似手法でも可視化して一貫性のある結果が得られるか確認することも大事である．初期化でも用いた PCA も比較対象のひとつである．また MDS も比較対象にするとよい．

2.3　まとめ

本章では学習が失敗する例を挙げて説明した．しかし SOM の欠点を挙げたわけではない．データの前処理などはどの手法を用いるにしても共通するし，データ分布をよく観察したり，学習過程をモニターしたりすることも基本である．要は，データを放り込めば自動的に答えを出してくれる便利な手法というのは存在しないということである．実際，SOM は他の手法と比べて使いやすく，デリケートな調整も必要ない，使い勝手の良い部類に属する．また SOM にとって困難な課題は，他の手法にとっても困難か，あるいは教師なし学習にとって本質的に重要な問題であることがほとんどである．その意味で，SOM の適切な運用を学ぶと言うことは，教師なし学習の基本的運用を学ぶことに通じる．

第 3 章

SOM のプログラミングと高速化

SOM のプログラミングは比較的容易である．もしデータ規模がそれほど大きくないのであれば，アルゴリズムどおりにプログラムしても問題なく動作する．

大規模なデータに SOM を応用する際は計算効率の良いコーディングが必要になる．また SOM のアルゴリズム自体を修正して高速化することもできる．本章では計算効率と高速化についても述べる．

3.1 SOM の計算オーダー

一般にアルゴリズムの計算オーダーは乗算の数で決まる．SOM の場合，競合プロセスでは乗算が $N \times K \times D$ 回現れる．また適応プロセスでも乗算が $N \times K \times D$ 回行われる．一方，協調プロセスでは exp 関数が $N \times K$ 回評価される（実直にプログラムした場合）が，計算速度への影響は小さい．したがって SOM の計算オーダーはおおむね $O(NKD)$ である．これはデータ規模 N およびデータ次元 D に比例して計算コストが増大することを意味する．特に N が大きいときの計算効率化は重要である．

3.2 適応過程の効率化

適応過程では参照ベクトルを更新する．そのためには N 個のデータベクトルに関して積和を取る必要があり，参照ベクトル 1 個あたりの ND 回の乗算が必要である．したがって適応過程の計算オーダーは $O(NKD)$ である．

Algorithm 1 は適応過程をそのままアルゴリズムとして表現したものであり，計算オーダーは $O(NKD)$ である．もし $N \gg K$ であるならば **Algorithm 2** のようにすることで計算オーダーを $O(K^2D)$ に抑えることができる[*1]．これは $\mathbf{Y}^{\mathrm{T}} = \mathbf{G}^{-1}(\mathbf{HB})\mathbf{X}^{\mathrm{T}}$ と計算する代わりに，$\mathbf{Y}^{\mathrm{T}} = \mathbf{G}^{-1}\mathbf{H}(\mathbf{BX}^{\mathrm{T}})$ と計算することに相当する．行列 $\mathbf{B}$ は 0 と 1 のみで構成されているため，$\mathbf{BX}^{\mathrm{T}}$ の計算は加算だけですむところがポイントである．

適応過程の計算をさらに高速化するのであれば，勝者ノードから遠く離れたノードに対して近傍関数を 0 にしてしまうことも考えられる．ガウス関数の場合，中心から 3σ より遠い点に対してはほとんど 0 と考えてよいため，積和からはずしてしまうことも可能である．2σ で打ち切ればさらに高速化できる．

[*1] このアルゴリズムでは，$N > K$ の場合には $O(K^2D)$，$N < K$ の場合には $O(NKD)$ になるため，N と K の大小に関わらず効率化される．

Algorithm 1 Cooperative and Adaptive process (1)

for $n = 1$ **to** N **do**
  for $k = 1$ **to** K **do**
    $R_{kn} \longleftarrow neighborhood(k_n^*, k, \sigma)$
  end for
end for
for $k = 1$ **to** K **do**
  $G_k \longleftarrow \sum_{n=1}^{N} R_{kn}$
  for $d = 1$ **to** D **do**
    $Y_{kd} \longleftarrow \sum_{n=1}^{N} R_{kn} X_{kd}$
    $Y_{kd} \longleftarrow Y_{kd} / G_k$
  end for
end for

Algorithm 2 Cooperative and Adaptive process (2)

for $k = 1$ **to** K **do**
  for $k' = 1$ **to** K **do**
    $H_{kk'} \longleftarrow neighborhood(k, k', \sigma)$
  end for
end for
$\forall k, \hat{\mathbf{y}}_k \longleftarrow \mathbf{0}, N_k \longleftarrow 0, G_k \longleftarrow 0$
for $n = 1$ **to** N **do**
  $\hat{\mathbf{y}}_{k_n^*} \longleftarrow \hat{\mathbf{y}}_{k_n^*} + \mathbf{x}_n$
  $N_{k_n^*} \longleftarrow N_{k_n^*} + 1$
end for
for $k = 1$ **to** K **do**
  for $k' = 1$ **to** K **do**
    if $N_{k'} \neq 0$ **then**
      $G_k \longleftarrow G_k + N_{k'} R_{kk'}$
      $\mathbf{y}_k \longleftarrow \mathbf{y}_k + H_{kk'} \hat{\mathbf{y}}_{k'}$
    end if
  end for
  $\mathbf{y}_k \longleftarrow \mathbf{y}_k / G_k$
end for

これらの高速化はたかだか数倍程度の効果しかない．したがって小規模な課題に対してはコーディング上の工夫を特にしなくても大きな問題にならない．しかし数日要する計算が 1 日で終わるのであれば，コーディングを工夫する価値がある．

3.3 競合過程の高速化

勝者ノードを決定するためには，データベクトルと参照ベクトルの二乗誤差，すなわちユークリッド距離を計算する必要があり，1 回の誤差計算につき D 回の乗算が必要となる．SOM ではデータベクトルと参照ベクトルのすべての組み合わせについて二乗誤差を求めるため，競合過程の計算オーダーは $O(NKD)$ となる．オーダーを下げるには，(1) K を下げる，(2) N を下げる，(3) D を下げる，の 3 とおりが考えられる．

3.3.1 競合ノードの限定による高速化

最初に K を下げる方法について検討する．通常の SOM では K 個のノードすべてが競合するが，いくつか選抜したノードについてのみ競合を行うことで高速化することができる．

第 1 の方法は，前回の勝者ノードから見て一定の距離内にあるノードのみを競合対象とする方法である．たとえば近傍半径 σ に対して前回の勝者ノードから見て 2σ 以内にあるノードで競合を行う．学習の後半では勝者位置がほとんど変わらないため，この方法を用いても計算結果に大きな影響は与えない．一方，勝者位置が激しく変わる学習の前半では，近傍半径も大きいため全ノードが競合対象となり，今までと同様の計算が行われる．

第 2 の方法は，学習の初期では粗いグリッドを使い，学習が進むに連れて徐々に解像度を上げることである．学習開始直後は 5×5 のノードで始め，次第に 10×10，20×20 とノード数を増やしていく[*2]．この方法と第 1 の方法を組み合わせることで，学習を通してほぼ一定の競合計算ですむようになる．

3.3.2 距離計算の簡略化による高速化

データの次元 D が大きい場合も計算コストが問題になる．D の影響を少なくするには，二乗誤差を絶対値誤差で置き換えてしまうことである．すなわちユークリッド距離（L_2 ノルム）でマンハッタン距離（L_1 ノルム）代用する．こうすると誤差計算に乗算がなくなるため，D が大きい場合は計算が一気に高速化する[*3]．この方法は SOM のハードウェア実装においてもしばしば用いられる．

3.3.3 セミバッチによる高速化

もし N が極めて多い場合，各ループで全データの勝者決定を行うのは大きな計算コストがかかる．もし一部のデータでも十分に学習可能であるならば，必ずしも全データを使う必要がない．そこでデータの一部で学習するセミバッチ法が有効になる．

セミバッチは以下のように進める．各ステップにおいて，$N' < N$ 個のデータ点をランダムに抽出する．そして抽出したデータに関して勝者ユニット更新し，またそれらのデー

[*2] コーディング上では最初から必要数をすべて用意しておき，学習開始時は休止させておけばよい．

[*3] Tensor SOM も D が大きい場合に相当する．学習の初期だけでもマンハッタン距離で代用し，学習の後半は競合ノード数を限定した上でユークリッド距離を使うことも考えられる．

タだけで参照ベクトルを更新する．

$$Y_{kd}(t+1) = \sum_{n'=1}^{N'} R_{kn'} X'_{n'd} \tag{3.1}$$

ここで $X'_{n'd}$ は抽出して作った新しいミニデータセットである．そして次のループでは新たに N' 個のデータを抽出し直す．

抽出による偏りが心配ならば，参照ベクトルの更新式 $\mathbf{Y} = \mathbf{G}^{-1}\mathbf{HBX}$ を次のようにすることも可能である．

$$Y_{kd}(t+1) = (1-\varepsilon)Y_{kd}(t) + \varepsilon \sum_{n'=1}^{N'} R_{kn'} X'_{n'd} \tag{3.2}$$

これはオンライン型とバッチ型の折衷になっている．

もし毎回ランダムにサンプルするのが面倒ならば，データを順々に N' 個ずつ取ってくるという方法も考えられる．もともとのデータが十分にシャッフルされていれば，この方法でもうまくいく．

3.4　ループ回数削減による高速化

時定数 τ を小さくすればループ回数を減らせる．計算時間はループ回数に比例するため，回数を減らせばそのまま計算時間短縮につながる．しかし計算を高速化すると不適切な解に陥るリスクも生じる．

ループ回数を減らす第 1 の方法は，PCA による初期化を行うことである．PCA 初期化の場合は線形多様体ですでにフィッティングされているため，（よほどデータ分布が複雑な形状をしていない限り）すみやかに写像の学習が行われる．

N が大きい場合は，データのサブセットを作っておき，これを用いて事前にパラメータチューニングをする．最後に全データを使って計算することで不要に遅い時定数を使うことが避けられる．

3.5　まとめ

SOM の効率的なコーディング法と，高速化のためのいくつかのアイデアを紹介した．現在のコンピュータは PC でも十分速いため，よほど大規模なデータでなければ計算できないことはないだろう．しかし次に述べる Tensor SOM ではさらに大規模な計算が要するため，これらの工夫が必要になる．

コーディングを工夫する際には，プログラムの各手続きに要する時間を調べ，もっとも時間がかかっている箇所の高速化を行うのが鉄則である．全体の所要時間に影響を与えていない箇所は，いくら速くしても効果が現れないからである．

最近の PC ではマルチコアがごく普通になった．マルチスレッドを使うことで高速化できることは言うまでもない．

参考文献

[1] T. Kohonen, Self-organized formation of topologically correct feature maps, Biological Cybernetics 43 (1) (1982) 59–69.

[2] T. Kohonen, Self-Organizing Maps, Springer-Verlag, Berlin Heidelberg, 2001.

[3] T. コホネン, 自己組織化マップ, シュプリンガーフェアラーク東京, 2005.

[4] S. Haykin, Neural Networks and Learning Machines, Prentice Hall, 2009, Ch. 9, pp. 425 – 474.

[5] J. Verbeek, N. Vlassis, B. Krose, Self-organizing mixture models, Neurocomputing 63 (2005) 99–123. doi:10.1016/j.neucom.2004.04.008.

[6] C. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006, Ch. 12, pp. 591–599.

[7] N. D. Lawrence, Gaussian process latent variable models for visualisation of high dimensional data, in: Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada], 2003, pp. 329–336.
URL http://papers.nips.cc/paper/2540-gaussian-process-latent-variable-models-for-visualisation-of-h

[8] S.-I. Amari, Topographic organization of nerve fields, Bulletin of Mathematical Biology 42 (3) (1980) 339–364. doi:10.1007/BF02460791.

[9] C. von der Malsburg, Self-organization of orientation sensitive cells in the striate cortex, Kybernetik 14 (2) (1973) 85–100. doi:10.1007/BF00288907.

[10] T. Kohonen, Self-organizing neural projections, Neural Networks 19 (6–7) (2006) 723 – 733.

[11] T. Kohonen, Essentials of the self-organizing map, Neural Networks 37 (2013) 52–65.

[12] S. Kaski, J. Kangas, T. Kohonen, Bibliopgraphy of self-organizing map (SOM) papers: 1981–1997., Neural Computing Surveys 1 (1998) 102–350.

[13] M. Oja, S. Kaski, T. Kohonen, Bibliopgraphy of self-organizing map (SOM) papers: 1998–2001 addendum, Neural Computing Surveys 3 (2003) 1–156.

[14] C. M. Bishop, M. Svensen, C. K. I. Williams, GTM: The generative topographic mapping, Neural Computation 10 (1998) 215–234.

[15] C. M. Bishop, M. Svensen, C. K. I. Williams, Developments of the generative topographic mapping., Neurocomputing 21 (1-3) (1998) 203–224.

[16] P. Stefanovic, O. Kurasova, Visual analysis of self-organzing maps, Nonlinear Analysis: Modelling and Control 16 (4) (2011) 488–504.

[17] A. Ultsch, H. P. Siemon, Kohonen's self organizing feature maps for exploratory data

analysis., in: Proc. INNC'90, Int. Neural Network Conf., 1990, pp. 305–308.

[18] N. Lawrence, Probabilistic non-linear principal component analysis with gaussian process latent variable models, Journal of Machine Learning Research 6 (2005) 1783?–1816.

[19] SOM_PAK.
URL http://www.cis.hut.fi/research/som_lvq_pak.shtml

[20] SOM Toolbox.
URL http://www.cis.hut.fi/somtoolbox/