日本神経回路学会誌　Vol. 17, No. 4（2010），187–196

解　　説

自己増殖型ニューラルネットワーク SOINN とその実践

山崎和博[*1]，巻渕有哉[*1]，申　富饒[*2]，長谷川　修[*3]

東京工業大学大学院知能システム科学専攻[*1]，中国南京大学計算機ソフトウェア新技術国家重点実験室[*2]，
東京工業大学像情報工学研究所[*3]

Self-Organizing Incremental Neural Network—SOINN— and Its Usage

Kazuhiro Yamasaki,[*1] Naoya Makibuchi,[*1] Furao Shen[*2] and Osamu Hasegawa[*3]

Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology[*1]
The State Key Laboratory for Novel Software Technology, Nanjing University[*2]
Imaging Science and Engineering Laboratory, Tokyo Institute of Technology[*3]

概要

自己増殖型ニューラルネットワーク（SOINN）は，追加学習可能なオンライン教師なし学習手法であり，事前にネットワークの構造を決定する必要がないほか，高いノイズ耐性を有し，計算が軽いなどの特長がある．SOINN は特に実世界のデータ処理に有効であり，画像や音声などのパターンの学習・認識や，実環境でオンライン・リアルタイムに稼動する知能ロボットなどに効果的に活用できる．本稿では，SOINN のアルゴリズムを紹介した後，パラメータの調整に関する指針やソフトウェア実装の解説を行なう．

1. はじめに

本稿では，著者らが提案する自己増殖型ニューラルネットワーク（Self-Organizing Incremental Neural Network: SOINN）のアルゴリズムやソフトウェア実装について解説する．

1.1 研究背景

SOINN は，Self-Organizing Map（SOM）[1,2]と Growing Neural Gas（GNG）[3]に着想を得て構築した教師なし学習手法の一種である．教師なし学習は，しばしばクラスタリングとも呼ばれ，代表的手法として k-means 法や EM アルゴリズムなどが挙げられる．

一般に，k-means 法に基づく手法では，利用者が事前にクラス数 k を決定する必要があるが，実世界のデータでは事前に適切なクラス数が決められるとは限らない．また，k-means 法ではクラスの形状が超球形であることを仮定しているが，実世界のデータはその限りではない．EM アルゴリズムも，事前にクラス数を決定する必要があるほか，初期値依存性を有している．

更に，実世界のデータは分布の形状が変化するなど非定常な場合があり，そうした分布の非定常性に対処するためには，追加学習の機能が必要となる．つまり，実世界のデータを適切に学習・認識するには，既存の学習結果に対して，新しいクラスからの入力や分布の形状変化などに対応し，動的にネットワークの構造を変化させうる学習手法でなければならない．また，実世界のデータの多くは継続的に入力されるため，オンライン学習の機能も必要となる．すなわち，多数の学習データを一括して処理するのではなく，入力されるデータを逐次学習し続けられなければならない．

以上を背景に，初期の SOINN[4] では 2 層のネットワークからなる構造を導入し，事前にその構造を決定することなく，非定常な分布から得られる入力の学習や，分布自体が複雑な形状をもつクラスに対する，適切なクラス数及び位相構造の出力などを実現した．また同時に，ノイズへの耐性も実現した[4]．

Enhanced-SOINN（ESOINN）[5]は，初期の SOINN の構造を 1 層に簡略化するとともに，事前定義パラメータの削減や，分布に高密度の重なりをもつクラスを分離できるなどの改良を施したものである．Shen らは，この ESOINN から更にパラメータを削減した構造を利用した，Adjusted SOINN Classifier（ASC）を提案している[6]．ASC は SOINN を教師あり学習手法

[*1] 〒 226–8503　神奈川県横浜市緑区長津田町 4529
[*2] 中国南京市
[*3] 〒 226–8503　神奈川県横浜市緑区長津田町 4529

へ応用した手法ともなっている．櫻井らは ESOINN 同様，SOINN を 1 層に簡略化した構造を提案し，半教師あり能動学習への拡張を行なっている[7]．

1.2 SOINN の発展的応用

SOINN を発展的に応用した研究も複数存在する．

岡田らは，SOINN を時系列データの学習・認識に利用した[8]．SOINN は分布が複雑な形状のクラスに対しても，分布を近似する位相構造を適切に獲得できる．そこで，この分布の近似機能を確率分布の近似に利用し，時系列データの学習や認識を実現した．

Sudo らは，SOINN へのデータ入力方式などを工夫することで，オンラインに学習可能な連想記憶を実現した[9]．また，これを発展的に拡張することで，パターン情報を命題とした任意の形の if-then ルールを学習し，これらの知識を利用して推論を行なうシステムも提案している[10]．パターン情報を用いた連想記憶器の更なる拡張として，Shen らによって，系列データなどの連想も可能なアーキテクチャが提案されている[11]．

転移学習への応用では，Aram らにより属性情報共有のためのオンライン学習手法が提案されている[12]．

知能ロボティクスへの応用もなされており，He らはロボットによる言語獲得や文法獲得の実現に SOINN を利用している[13,14]．巻渕らは，ロボットの知能発達のために SOINN と General Problem Solver（GPS）[15] を組み合わせたアーキテクチャを提案し[16]，ロボットによる汎用的な問題解決を実現している．

2. SOINN アルゴリズム

前述のように，SOINN には複数の拡張が存在する．ここでは，それらを総称して SOINN シリーズと呼び，本節ではそれらで共通して利用する，最も基本的なアルゴリズムである adjusted SOINN について説明する．

2.1 アルゴリズム概略

SOINN シリーズのうちいくつか[5–7]は，オンライン学習による自己増殖的なネットワークの構成と，構成されたネットワークに基づく入力分布の位相構造の近似ができる．それらに共通する，入力パターンに対する挙動は以下の通りである．

- 分布の近似が十分な領域へ入力されたパターンは，既に現在の構造によって表現されているパターンである可能性が高いため，そうした入力パターンは周辺のノードを更新するために利用される．
- 分布の近似が不十分な領域へ入力されたパターンは，ノイズの可能性がある．その一方で，現在の構造では表現が不十分なパターンの可能性もある．そこで，入力パターンをネットワーク構造に取り込むことによって，分布の近似をより精密に行なえるよう，構造を変化させる．

以上に加え，関係するノードをエッジで連結する，ノイズを除去する，などの処理を組み合わせることで，SOINN のアルゴリズムは構成される．

2.2 Adjusted SOINN

adjusted SOINN は SOINN シリーズで共通して利用するアルゴリズムであり，1 層構造をなし，2 つのパラメータを有する．また adjusted SOINN は，初期の SOINN に比べネットワーク構造やパラメータ数などが整理・統合されているが，識別能力やクラス分離能力は初期の SOINN と同等以上である[6]．

adjusted SOINN でも，他の SOINN シリーズ同様，オンライン学習による自己増殖的なネットワークの構成と，構成されたネットワーク構造に基づく入力分布の近似が目的となる．そのため各入力に対して，「ノードの挿入」，「エッジの挿入・削除」，「ノードの位置ベクトルの更新」，「ノードの削除」という処理を適宜組み合わせて実行し，適切な分布の近似を行なう．

また，SOM など多くの従来手法は事前にネットワークの構造を決定する必要があるが，adjusted SOINN をはじめとする SOINN シリーズではその必要がない．すなわち，SOINN シリーズはネットワークの構造が可変であり，追加学習に対しても適応的に対処できる．更に，GNG では事前にノード数の上限を決定しない場合，定常的な入力に対してノード数が際限なく増加してしまうという問題がある．これに対し，SOINN シリーズでは，ノード数は入力データの分布に応じて自動的に決定される．すなわち，入力分布の近似にノード数が十分であると判定した場合，ノードの追加を行なわないため，ノード数が適切に制御できる．

2.2.1 ノードの挿入

ノードの挿入は，分布の近似が十分でない領域に対して実行される．分布の近似が十分であるかどうかの判定は，既存のネットワーク構造から類似度閾値 T を算出し，これに基づいて行なわれる．Fig. 1 に示すように，入力パターンと勝者ノードおよび第二勝者ノードとの距離がいずれかの類似度閾値 T を超える場合，分布の近似が不十分であると判定され，入力パターンを新たなノードとしてネットワークに挿入する．なお，勝者ノードとは入力パターンに対する最近傍ノードであり，第二勝者ノードとは入力パターンに 2 番目に近いノードを指す．

類似度閾値 T は，Alg. 1 によって算出される．エッジで連結されたノード（以下，連結ノード）が存在する場合は，連結ノードのうち最も遠いノードとの距離

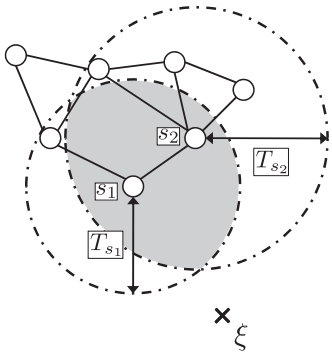

Fig. 1 Node insertion process.

を，連結ノードが存在しない場合は，最近傍ノードとの距離をそれぞれ類似度閾値とする．このとき，後述するノードの位置ベクトルの更新処理などによって，ノード間の距離は動的に変化しうるため，それに伴って類似度閾値も変化することに注意されたい．

Alg. 1 Method to calculate the similarity threshold T_i.

Require: i：ノード番号（勝者ノード，または第二勝者ノード），N_i：ノード i の連結ノード集合，A：全ノードの集合，W_i：ノード i の位置ベクトル，T_i：ノード i の類似度閾値

if $N_i \neq \phi$ **then**
  $T_i \leftarrow \max_{c \in N_i} ||W_i - W_c||$
else
  $T_i \leftarrow \min_{c \in A \setminus \{i\}} ||W_i - W_c||$
end if

2.2.2 エッジの挿入・削除

エッジの挿入・削除は，後述するノードの位置ベクトルの更新とともに，入力に応じてネットワークの位相構造を形成・更新するために実行される．エッジの挿入・削除は，ノード間の接続関係を適切に構築することを目的になされ，構築基準として competitive Hebbian rule（CHL）[17] を用いる．CHL は，各入力に対してユークリッド距離基準で最も近い 2 つのノード（勝者及び第二勝者ノード）をエッジで連結する．また，CHL によって得られたグラフ構造は分布の形状を有効に近似する[17]．一方で，後述するノードの位置ベクトルの更新や新規ノード挿入の結果，初期に構築された連結関係が学習の進行に伴い不適切になる可能性がある．

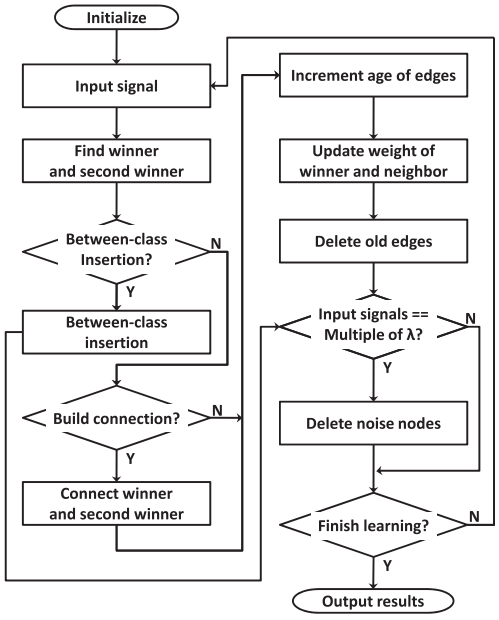

Fig. 2 Flowchart of adjusted SOINN.

そのため，エッジの加齢処理（edge aging scheme）[18] を用いて参照されなくなったエッジを削除することにより，学習の進行に合わせて連結関係の更新も行なう．

2.2.3 ノードの位置ベクトルの更新・ノードの削除

ノードの位置ベクトルの更新は，Fig. 1 の灰色で示す領域へ入力が発生した際に行なわれる．この更新処理は，入力パターンに対するネットワークの局所的な適応を目的としており，各入力パターンに対して，勝者ノードとその連結ノードの位置ベクトルが更新される．

ノードの削除は，ノイズの影響を受けたノードを削除するために，連結ノード数が 1 以下のノードはノイズの影響を受けている可能性が高いという仮定に基づいて，入力パターンが一定数を超えるたびに実行される．

以上が adjusted SOINN のアルゴリズムとなる．Alg. 2 に adjusted SOINN のアルゴリズムを，Fig. 2 にそのフローチャートを示す．

3. 事前定義パラメータ解析

adjusted SOINN には 2 つのパラメータが存在し，その調整が問題となる．本節では，それら 2 つのパラメータの役割や性質について概要を述べたのち，その調整方法の指針を示す．

3.1 パラメータの性質

Alg. 2 に示した通り，adjusted SOINN には 2 つの事前定義パラメータ λ, age_{max} が存在する．この 2

Alg.2 Learning algorithm of adjusted SOINN.

Require: A：全ノードの集合，$C \subset A \times A$：全エッジの集合，N_i：各ノード i の連結ノード集合，W_i：各ノード i の位置ベクトル，λ：ノード削除パラメータ，age_{max}：エッジ削除パラメータ

1: **if** 初回の学習である **then**
2: $A \leftarrow \{c_1, c_2\}$：学習データからランダムに選択した位置ベクトルをもつ 2 つのノード c_1, c_2 で全ノードの集合を初期化
3: $C \leftarrow \phi$
4: **end if**
5: **while** 入力パターン $\xi \in R^n$ が存在する **do**
6: $s_1 \leftarrow \arg\min_{c \in A} ||\xi - W_c||$：$\xi$ に対する勝者ノード s_1 を探索
7: $s_2 \leftarrow \arg\min_{c \in A \setminus \{s_1\}} ||\xi - W_c||$：$\xi$ に対する第二勝者ノード s_2 を探索
8: Alg. 1 によって類似度閾値 T_{s_1}, T_{s_2} を計算
9: **if** $||\xi - W_{s_1}|| > T_{s_1}$ または $||\xi - W_{s_2}|| > T_{s_2}$ **then**
10: $A \leftarrow A \cap \{\xi\}$：$\xi$ を新規ノードとして追加
11: **else**
12: **if** $(s_1, s_2) \notin C$：s_1 と s_2 の間にエッジがない **then**
13: $C \leftarrow C \cap \{(s_1, s_2)\}$：エッジ (s_1, s_2) を追加
14: **end if**
15: $a_{(s_1,s_2)} \leftarrow 0$：エッジ (s_1, s_2) の年齢を 0 にリセット
16: $a_{(s_1,i)} \leftarrow a_{(s_1,i)} + 1$ $(\forall i \in N_{s_1})$：s_1 につながる全エッジの年齢をインクリメント
17: $\Delta W_{s_1} \leftarrow \epsilon_1(t_{s_1})(\xi - W_{s_1})$：$t_i$ はノード i が勝者ノードに選択された回数，$\epsilon_1(t_i) = 1/t_i$
18: $\Delta W_i \leftarrow \epsilon_2(t_i)(\xi - W_i)$ $(\forall i \in N_{s_1})$：$\epsilon_2(t_i) = 1/100t_i$
19: 勝者ノードとその近傍ノードの重みベクトルを ΔW_{s_1} と ΔW_i によって更新 $(\forall i \in N_{s_1})$
20: 事前閾値 age_{max} に対して $a_e > age_{max}$ を満たす全エッジ $e \in C$ を削除
21: 20 において削除されたエッジにより連結されていたノードのうち，$|N_i| = 0$ となったノード i をすべて削除．ただし $|\cdot|$ で集合の要素数を表す
22: **end if**
23: **if** 入力パターン数が λ の倍数 **then**
24: $|N_i| \leq 1$ となったノード i をすべて削除
25: **end if**
26: **end while**

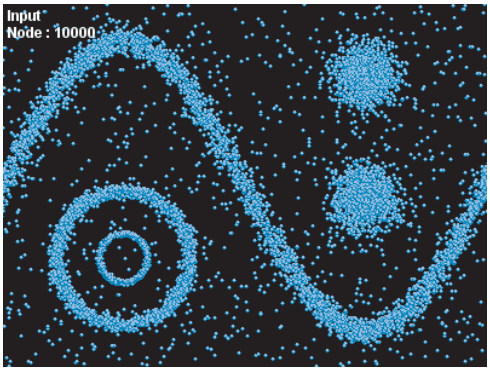

Fig. 3 Example of input data.

つのパラメータは，ノードが疎な領域における削除頻度とエッジ削除頻度にそれぞれ影響を与える．以下，λ, age_{max} を変化させた場合に SOINN が示す挙動を，実験例を交えて説明する．

今回の実験ではデータセットとして，2 つの同心円，2 つの正規分布，正弦波関数からなる 5 クラスの 2 次元の人工データ（Fig. 3）を用い，λ, age_{max} を変化させたときに SOINN が学習したノード数とクラス数を解析の対象とした．また，特に断らない限り，10% の一様ノイズが付加されている．今回の実験において学習に要した時間は，Java 言語による実装，Intel Xeon 3.0 GHz, メモリ 4 GByte の PC 環境の下で，$(\lambda, age_{max}) = (5001, 200)$ で 647.0 ノード，1,850.0 (ms), $(\lambda, age_{max}) = (501, 200)$ で 145.0 ノード，197.0 (ms) であった．

なお，今回の実験では解析の対象とするパラメータ値の範囲を制限している．これは，λ の値が入力データ数より大きな場合，Alg. 2 の L.23 以降の処理が実行されず，SOINN の動作に λ が関与しないためである．同様に，SOINN のあるひとつのエッジに注目した場合，その近傍にあるエッジに入力されるデータ数は全体の入力データ数に対して非常に少ない値であることが予想される．そのため，入力データに対して大きすぎる age_{max} の値は，エッジの削除に対して有効ではない．こうした理由や，これまでの研究から得た知見を踏まえ，λ の最大値を入力データ数，age_{max} の最大値を入力データ数の 1/100 としている．ただし追加学習などによりデータ数の増加が見込まれる場合は，増加量を想定したパラメータの設定や，動的な修正が必要であることに注意されたい．

3.1.1 ノード数との関係

Fig. 4 に，1 万データを 5 セット学習したときのノー

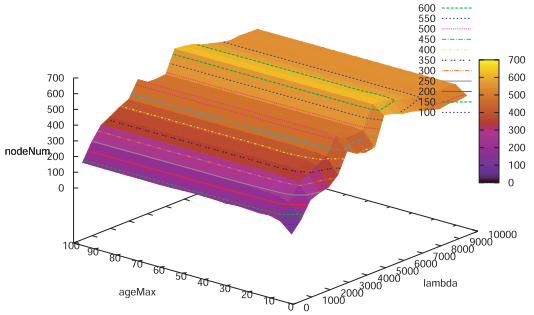

Fig. 4 Graph of average number of nodes for 10,000 artificial data. $\lambda \in [0, 10000]$, $age_{max} \in [0, 100]$.

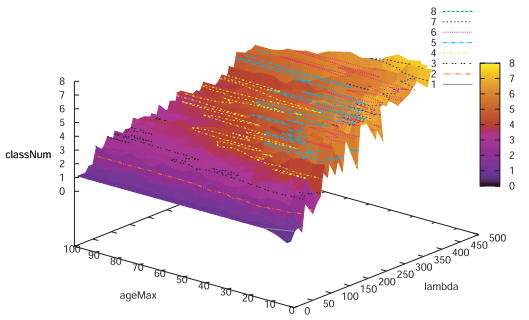

Fig. 5 Graph of average number of classes with 10% noise. $\lambda \in [0, 500]$, $age_{max} \in [0, 100]$.

ド数の平均と λ, age_{max} の関係を示す．Fig. 4 において，入力データ数を n として，まず，$\lambda = n/k$ $(k = 1, \cdots, n)$ から $\lambda = n/k + 1$ へ変化する際にノード数が増加している．これは，Alg. 2 の L.23 以降の処理が起きた回数の違いによるものと考えられる．実際，k はこの処理の回数を表している．次に，$\lambda = n/k + 1$ から $\lambda = n/(k-1)$ に向かってノード数が若干減少する傾向が見られる．これは，L.23 以降の処理が起こるまでは，ノードの削除が明示的に行なわれることはなく，SOINN は前回の削除以降に入力されたデータを保存するよう動作することに起因する．

以上より，λ を n/k ごとに増加させることで，ノード数を安定して増加させることができる．ただし，$\lambda = n/2 + 1$ まで λ を増加させると，入力パターン数が固定されている場合，基本的にそれ以上ノードを増やすことができないため，注意が必要である．

3.1.2 クラス数との関係

Fig. 5 に，1 万データを 5 セット学習した際のクラス数の平均と λ, age_{max} との関係を示す．

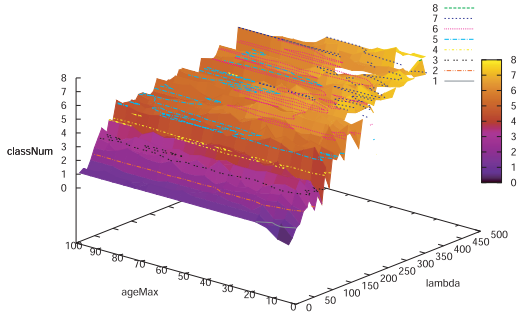

Fig. 6 Graph of average number of classes with 0% noise. $\lambda \in [0, 500]$, $age_{max} \in [0, 100]$.

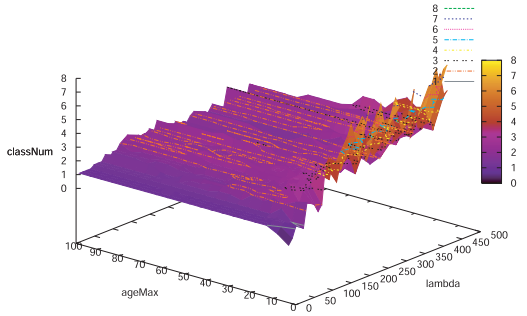

Fig. 7 Graph of average number of classes with 30% noise. $\lambda \in [0, 500]$, $age_{max} \in [0, 100]$.

λ が小さい範囲では，クラス数の変化に age_{max} が影響していない．これは，入力パターン数に対して λ が小さい範囲では，age_{max} によるエッジの削除が発生する前に，λ によるノードの削除（Alg. 2 の L.23 以降）が発生してしまい，エッジの年齢が age_{max} に達しないためと考えられる．次に，λ が大きい範囲では，λ の増加に伴ってクラス数が増加するだけでなく，age_{max} の減少に伴いクラス数が増加する．これは，λ の増加によってノード数が増加する一方で，age_{max} の減少に伴いクラスの分離が進むことで，少数ノードからなるクラスが多数形成されることによる．以上より，age_{max} を有効に機能させるためには，λ を一定値以上に設定する必要がある．

3.1.3 ノイズ量との関係

Fig. 6，7 にノイズが 0% 及び 30% の場合の結果を示す．こちらも同様に，λ の増加に伴ってクラス数が増加し，age_{max} の増加に伴いクラス数が減少する傾向が読み取れる．それぞれの結果を比較すると，ノイズが増加するごとに，同じパラメータに対して出力されるクラス数が減少していることが分かる．今回の実

Table 1 Guide for sizes of data, noise and so on.

データ量	多	数百万～数千万
	中	数千～数十万
	少	数十～数百
ノイズ量	多	全体の 30% 程度
	中	全体の 10% 程度
次元数	高次元	～数百次元
	低次元	～数十次元
クラス数	多	数十クラス程度
	少	十クラス程度

験のように一様ノイズがデータセットに対して過剰に発生する場合，各クラスの入力分布がもつ特徴が相対的に失われる．その結果，入力全体がひとつのクラスとしてまとめられる傾向が発生し，クラス数の減少を引き起こしたと考えられる．

これらの結果及びノード数と 2 つのパラメータとの関係から，例えばクラス数を維持したままノード数を増加させたければ，λ と age_{max} とを同時に増加させればよい．一方クラス数を増加させたければ，λ を増加させ，age_{max} を減少させればよい．またノイズが多い場合には，age_{max} を減少させることで，クラスの分離を適切に行なうことができる．

なお，入力パターン数が少ない場合，クラス数が安定しないことがある．そうした場合には，学習が収束するまで再学習を行なう[8]，あるいは，ガウス分布を用いて入力パターンへ摂動を加える[9] などの処理が有効である．

3.2 パラメータの調整

3.1 節では各パラメータの性質のうち，ノード数・クラス数との関係について述べた．ここではそれらの関係を踏まえ，具体的な利用例とともにパラメータの調整について指針を示す．なお，入力パターンの特性として以下を想定している．

- ノイズが多い：ノイズとして発生したパターンが全体の 3 割程度
- 高次元の場合：次元数が数百次元程度
- クラス数が多い：クラス数が数十程度

Table 1 に，これらデータ量やノイズ量の大小についての目安をまとめる．以後，Table 2 に示す 6 通りの場合における調整例を述べる．

3.2.1 データマイニングなどにおける調整例

まず SOINN をデータマイニングなどの分野で利用する状況を想定する．この場合，データ数が非常に多いことが予想される（入力パターン数 = 数百万～数千万データ）．また，実データを対象とするため，ある程度のノイズの混入が予想される（全体の 10% 程度がノイズ）．次元数やクラス数に注目すると，どちらも低次元の場合やクラス数が少ない場合，あるいは高次元，多数クラスの場合など非常に多岐に渡ることが予想される．そのため，適切にクラスを分離するためには，パラメータも問題に応じて調整する必要がある．例えば，低次元でクラス数が多い場合（Table 2 の 1）には，age_{max} を小さめに設定し，λ も比較的小さく設定すればよい．これにより，ノード数の増加を抑えつつ，適切なクラスの分離が可能である．一方，高次元の場合（Table 2 の 2）には，低次元の場合と比較して特徴空間上におけるパターンの密度が小さくなると考えられるため，λ, age_{max} ともに大きくとればよい．λ, age_{max} とクラス数との関係も踏まえ，λ, age_{max} の範囲を図示すると Fig. 8 のようになる．

Table 2 Indices for three factors.

番号	データ数	ノイズ量	次元数
1	多	中	低次元
2	多	中	高次元
3	中・少	中	低次元
4	中・少	中	高次元
5	中・少	多	低次元
6	中・少	多	高次元

3.2.2 パターン認識などにおける調整例

次に SOINN を画像認識や音声認識などのパターン認識に利用する状況を想定する．この場合，データ数がある程度確保できる場合（入力パターン数 = 数千～数十万データ）と確保できない場合（入力パターン数 = 数十～数百データ）の 2 通りが考えられる．また，こちらもノイズが一定以上含まれていることが予想され，次元数やクラス数に関しても，様々な場合が考えられる．こうした場合，例えば，ノイズが中程度で次元数が小さい（Table 2 の 3）ならば，λ, age_{max} ともに比較的小さく設定すればよい．一方でノイズが多い場合（Table 2 の 5）には，ノイズの影響を受けたエッジが削除されやすくなるよう，λ を大きく設定し，age_{max} を小さく設定すればよい．また，高次元の場合（Table 2 の 4，6）には，前述のデータマイニングの例と同様に，λ, age_{max} をともに大きくとればよい．更に，データ数が中程度の場合と，少数データしか取り扱えない場合は，基本的にデータ数が増えるごとに，λ, age_{max} の両方を大きく設定すればよい．以上の議論を受け，Fig. 9 に Fig. 8 と同様，パラメータの範囲

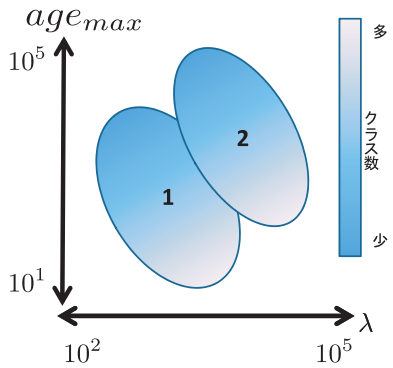

Fig. 8 Example of configuration for data mining.

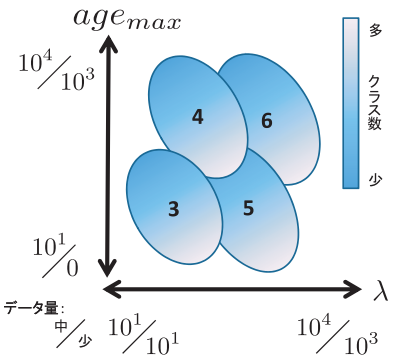

Fig. 9 Example of configuration for robotics.

を図示する．

なお，3.1 節などでも言及したとおり，データ数に応じて，λ と age_{max} のスケールが変化することに注意されたい．また，データマイニングの例のように，大規模データを扱う場合にはノード数が増加し，計算時間が問題となるが，SOINN の主要な処理は，最近傍探索における距離計算であり，GPU 実装などによる高速化が可能である．現在，著者らのグループでは SOINN の GPU 実装を進めている．

3.2.3 適用範囲に関する考察

Table 1 では，データ量やノイズ量の大小に関する目安を示したが，SOINN は原理的にこの範囲を超える状況にも対応可能である．ここでは，Table 1 の範囲を超える状況に関する考察を行なう．

ノイズ量や次元数が Table 1 の範囲内で，なおかつデータ量やクラス数が更に大きくなる場合，基本的にこれまでの議論の延長によってパラメータの調整が可能と推測される．すなわち，データが増加する場合，そのスケールに合わせて λ, age_{max} を変化させればよい．また，多数のクラスが存在し，より詳細な分離が必要であれば，こちらもそのスケールに応じて，λ を大きく，age_{max} を小さく設定すればよいと考えられる．しかしながら，ノイズ量や次元数の変化については，入力分布との関係などから挙動の予測が難しく，パラメータを試行錯誤で決定する必要がある．

なお，入力データが非常に高次元かつスパースな場合，十分に SOINN のノードが形成されないことがある．そうしたデータを SOINN により学習する場合には，入力データの次元を下げるなどの前処理を行なう必要がある．

4. SOINN Software

前節までに SOINN のアルゴリズムとパラメータの調整方法について述べた．本節では，SOINN のソフトウェア実装について解説する．

4.1 SOINN Software 概説

SOINN のソフトウェア実装は，以下の Web ページから無償で入手できる．

`http://haselab.info/downloads.html`

現在公開しているソースコードは C++ 及び Java 言語による 2 種類の実装であり，C++ による実装は Microsoft Visual Studio 2005 のソリューションとして公開している．そのため，C++ による実装を利用する場合は SOINN のソースコードのみをコピーするか，ソリューションの設定を適宜書き換える必要がある．これらの実装は adjusted SOINN[6] を実装したものであり，すべての言語で同様の機能を提供している．なお，本稿で参照しているソースコードのバージョンは 1.2.0 であり†，今後もバグフィックスや機能拡張などの更新が行なわれる可能性がある．

SOINN Software は C++，Java 言語のいずれの場合においても，基本的な構成は同じであり，以下に示す通りである．

- SOINN クラス
- Node クラス
- Edge クラス

SOINN クラスはデータの入力，学習など，基本的な機能を提供している．Node クラスと Edge クラスはそれぞれ，SOINN アルゴリズムにおけるノードとエッジを表したものである．一般的なパターン認識は，SOINN クラスに定義されている関数を複数組み合わせて呼び

† バージョン番号の形式は「メジャーアップデート．マイナーアップデート．バグフィックス」である．

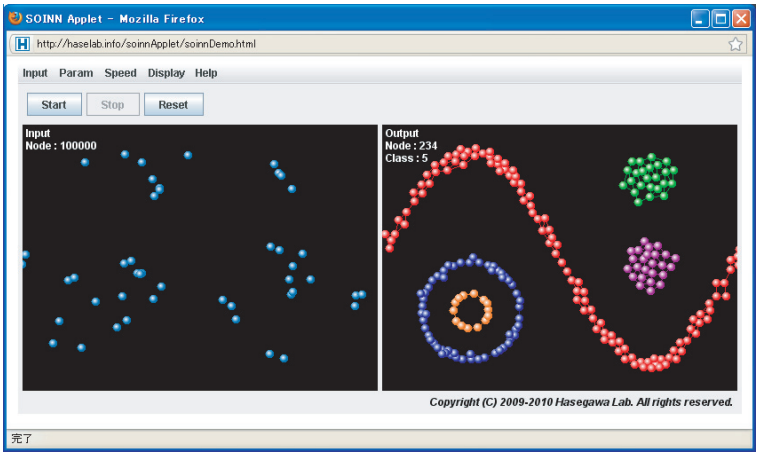

Fig. 10 Example of execution of this applet.

出すことで実現できる．SOINN Software の具体的な使用例などは，ダウンロードページにあるサンプルプログラムやユーザーガイドを参照されたい．

また，前掲の Web ページでは SOINN の動作を確認するためのデモとして，SOINN プログラムの Java アプレットも公開している．このアプレットを利用することで，入力データが SOINN によって学習される様子をグラフィカルに確認することができる．

Input メニューでは入力する人工データの種類を選択することができる．Param メニューでは，SOINN がもつ 2 つのパラメータと入力データに付加するノイズの割合を設定できる．Speed メニューは学習データの入力速度を変更する．Display メニューは表示範囲に関する設定を行なう．Fig. 10 に実行例を示す．この例の入力は前述の Fig. 3 と対応している．画像右側に SOINN の出力を示しており，色が異なるノードは異なるクラスと判定されている．この例から SOINN が適切にノイズを除去できていることが分かる．

4.2 SOINN Software ソースコード概説

ここでは SOINN Software の内部実装と，SOINN アルゴリズム（Alg. 2）との対応関係を説明する．SOINN アルゴリズムと対応している SOINN Software の関数は `CSOINN::InputSignal()` であり，この関数を用いて入力データの学習を行なう．以下，Lst. 1 に示すソースコードと比較しながら説明する．

まず，Alg. 2 の L.1～L.4 にある初期学習に対応する部分は，ソースコード中 L.12～L.16 の処理である．ソースコード L.14 の `CSOINN::AddNode()` 関数に入力パターンを渡すことにより，ネットワークに対して新規ノードを追加することができる．また，説明が前後するが，ソースコード L.9 で変数 `m_inputNum` をインクリメントすることにより，入力パターン数をカウントしている．

続いて入力データの学習についてだが，Alg. 2 は入力パターンが続く限り学習を繰り返すように設計されている．一方で，SOINN Software は，ユーザが外部で繰り返し処理部分を実装するよう設計されている．すなわち，`CSOINN::InputSignal()` 関数には Alg. 2 の L.6～L.25 に相当する，1 データごとの学習処理が実装されており，複数のデータを一括して学習する関数は提供されていない．大量のデータを学習する際に必要な関数の実装例は，ダウンロードページにあるサンプルプログラムを参照されたい．これ以降，Alg. 2 の L.6～L.25 に相当する部分の説明を行なう．

Alg. 2 の L.6, 7 に対応するソースコード中の処理は，L.19 の `CSOINN::FindWinnerAndSecondWinner()` 関数である．この関数により，現在のネットワーク構造から，入力パターンに対する勝者ノードと第二勝者ノードが探索される．Alg. 2 の L.8～L.10 に対応する処理は，ソースコード中 L.22～L.24 である．ソースコード L.22 の `CSOINN::IsWithinThreshold()` 関数によって，入力パターンと勝者ノード及び第二勝者ノード間の距離が類似度閾値に対して大きいかどうか判定され，いずれか一方でも類似度閾値が大きいと判定されると `CSOINN::AddNode()` 関数により，新規ノードとして入力パターンがネットワークに追加される．このとき，関数内部では類似度閾値の計算が行なわれている．

続いて，Alg. 2 の L.12～L.14 にはソースコード L.29 の `CSOINN::AddEdge()` 関数が，Alg. 2 の L.15 に

はソースコード L.31 の CSOINN::ResetEdgeAge() 関数が，Alg. 2 の L.16 にはソースコード L.33 の CSOINN::IncrementEdgeAge() 関数が，Alg. 2 の L.17〜L.19 にはソースコード L.35 の CSOINN::UpdateLearningTime() 関数と L.37 の CSOINN::MoveNode() 関数が，Alg. 2 の L.20，21 にはソースコード L.39 の CSOINN::RemoveDeadEdge() 関数がそれぞれ対応している．各関数ではそれぞれ順に，勝者ノードと第二勝者ノード間にエッジがなければ新規作成を行なう，勝者ノードと第二勝者ノード間のエッジの年齢をリセットする，勝者ノードと近傍ノードとの間にある全エッジの年齢をインクリメントする，勝者ノードの勝者ノード選択回数をインクリメントする，勝者ノードと近傍ノードの重みベクトルを更新する，閾値を超えた年齢のエッジを削除する，という処理が行なわれている．

最後に，Alg. 2 の L.23〜L.25 にはソースコード L.43 〜 L.49 が対応し，λ 回の学習ごとに CSOINN::RemoveUnnecessaryNode() 関数が呼び出される．この関数により，近傍ノード数が 1 以下となったノードが削除される．続いて，CSOINN::Classify() 関数により，各ノードにクラス ID が割り当てられる．この際，エッジで連結されたノードには同一のクラス ID を割り当てるものとする．なお，この関数に対応する処理は Alg. 2 中に明示的に示されていないが，本実装では便宜的に λ 回の学習ごとにこの関数を呼び出すことで，自動的にクラス ID の更新を行なっている．そのため，外部からクラス ID を参照する必要がある場合，ユーザが明示的に CSOINN::Classify() 関数を呼び出し，クラス ID の更新を行なう必要がある．また，孤立したノードをノイズとして除去したい場合には，CSOINN::Classify() 関数の呼び出しを行なう前に，CSOINN::RemoveUnnecessaryNode() 関数を呼び出す必要がある．

Lst. 1: Source code of CSOINN::InputSignal()

```
1  bool CSOINN::InputSignal(const double *
       signal)
2  {
3    int winner, secondWinner;
4
5    // If the input data is invalid value,
         return false and exit
6    if (signal == NULL) return false;
7
8    // Count up the number of input data
9    m_inputNum++;
10
11   // If number of nodes is less than 2,
       directly add the input data as new
       node
12   if (m_nodeInfo.size() < 2)
13   {
14     AddNode(signal);
15     return true;
16   }
17
18   // Find winner and runnerup
19   FindWinnerAndSecondWinner(winner,
         secondWinner, signal);
20
21   // If the input data belongs to new
         knowledge, insert it as new node
22   if (!IsWithinThreshold(winner,
         secondWinner, signal))
23   {
24     AddNode(signal);
25   }
26   else
27   {
28     // generate edge between winner and
           runnerup
29     AddEdge(winner, secondWinner);
30     // Reset the age of edge between
           winner and runnerup
31     ResetEdgeAge(winner, secondWinner);
32     // Count up the age of edges linked
           with winner
33     IncrementEdgeAge(winner);
34     // Count up the number of times been
           winner
35     UpdateLearningTime(winner);
36     // Update weight of winner and
           neighbor of winner
37     MoveNode(winner, signal);
38     // Remove edges whose age are greater
           than threshold of age
39     RemoveDeadEdge();
40   }
41
42   // If the learning times are greater
         than threshold, remove isolated
         nodes
43   if (m_inputNum%m_lambda == 0)
44   {
45     RemoveUnnecessaryNode();
46
47     // Give class label for all nodes
48     Classify();
49   }
50
51   return true;
52 }
```

5. おわりに

本稿では，著者らが提案する自己増殖型ニューラルネットワーク（SOINN）のアルゴリズムと，そのパラメータの調整に関する指針などについて述べた．また，ソフトウェアの利用方法やアルゴリズムとの対応についても言及した．更に，本稿ではパラメータとノード数・クラス数の関係について，実験をもとに定性的な評価を行なった．

近年，理論的背景が比較的明らかで，実問題に対して有効なオンライン学習手法[19] が登場している．今後はこうした手法などを参考に，SOINN の挙動の理論的解明にも取り組む予定である．

SOINN が一人でも多くの方に有効に活用して頂ければ，著者らとして望外の幸せである．

参 考 文 献

1) Willshaw, D.J., von der Malsburg, C. (1976): How patterned neural connections can be set up by self-organization, In Proc. of the Royal Society of London B, Vol.194, pp.431–445
2) Kohonen, T. (1982): Self-organized formation of topologically correct feature maps, Biological Cybernetics, Vol.43, pp.59–69
3) Fritzke, B. (1995): A growing neural gas network learns topologies, Advances in neural information processing systems, pp.625–632
4) Shen, F., Hasegawa, O. (2005): An incremental network for on-line unsupervised classification and topology learning, Neural Networks, Vol.19, No.1, pp.90–106
5) Shen, F., Ogura, T., Hasegawa, O. (2007): An enhanced self-organizing incremental neural network for online unsupervised learning, Neural Networks, Vol.20, No.8, pp. 893–903
6) Shen, F., Hasegawa, O. (2008): A fast nearest neighbor classifier based on self-organizing incremental neural network, Neural Networks, Vol.21, No.10, pp.1537–1547
7) 櫻井啓介，神谷祐樹，長谷川修（2007）：競合型ニューラルネットを用いたオンライン準教師付き能動学習手法, 電子情報通信学会論文誌 D, Vol.J90-D, No.11, pp.3091–3102
8) 岡田将吾，長谷川修（2008）：自己増殖型ニューラルネットワークを用いた時系列データの学習・認識，電子情報通信学会論文誌 D, Vol.J91-D, No.4, pp.1042–1057
9) Sudo, A., Sato, A., Hasegawa, O. (2009): Associative memory for online learning in noisy environments using self-organizing incremental neural network, IEEE Transactions on Neural Networks, Vol.20, No.6, pp.964–972
10) Shen, F., Sudo, A., Hasegawa, O. (2010): An online incremental learning pattern-based reasoning system, Neural Networks, Vol.23, Issue 1, pp.135–143
11) Shen, F., Yu, H., Kasai, W., Hasegawa, O. (2010): An associative memory system for incremental learning and temporal sequence, In Proc. of the 2010 Int'l Joint Conf. on Neural Networks, pp.698–705
12) Kawewong, A., Hasegawa, O. (2010): Fast and incremental attribute transferring and classifying system for detecting unseen object classes, 1st SOINN Workshop, Proc. Int'l Conf. on Artificial Neural Network, pp.563–568
13) He, X., Kojima, R., Hasegawa, O. (2007): Developmental word grounding through a growing neural network with a humanoid robot, IEEE Trans. SMC-Part B, Vol.37, No.2, pp. 451–462
14) He, X., Ogura, T., Satou, A., Hasegawa, O. (2007): Developmental word acquisition and grammar learning by humanoid robots through a self-organizing incremental neural network, IEEE Trans. SMC-Part B, Vol.37, No.5, pp. 1357–1372
15) Ernst, G.W., Newell, A. (1969): GPS: A case study in generality and problem solving. New York: Academic Press
16) 巻渕有哉，申　富饒，長谷川修（2010）：実世界における一般問題解決システムの提案とそのヒューマノイドロボットへの実装, 電子情報通信学会論文誌 D, Vol.J93-D, No.6, pp.960–977
17) Martinetz, T.M. (1993): Competitive hebbian learning rule forms perfectly topology preserving maps, In Proc. Int'l Conf. on Artificial Neural Networks, pp.427–434
18) Martinetz, T.M., Schulten, K.J. (1994): Topology representing networks, Neural Networks, Vol.7, No.3, pp.507–522
19) Bottou, L., Bousquet, O. (2008): Learning using large datasets, In Mining Massive Data Sets for Security, pp.15–26