情報処理学会論文誌 Vol. 49 No. 8 2878–2889 (Aug. 2008)

ニューラルネットツリーを生成する高速アルゴリズム

林 博友[†1] 趙 強福[†1]

ニューラルネットツリー（NNTree）は各中間ノードに小規模なニューラルネット（NN）を埋め込んだ決定木（DT）である．NNTree の利点としては，フル結合型 NN と比べ構造学習およびハードウェア実現に適していることと，通常の単一変量 DT と比べ汎化能力が高いこと等があげられる．しかし，NNTree の生成は難しい問題であり，NN はたとえ 1 個のニューロンしか含まないときでさえ，各中間ノードの最適なテスト関数を求めることは NP 完全問題である．この問題を解決するために我々は遺伝的アルゴリズム（GA）に基づく NNTree の生成方法を試みたが，この方法は非常に時間がかかるため容易に使用できない．NNTree を高速に生成するために，本論文で我々は新しいアルゴリズムを提案する．このアルゴリズムでは，まず各中間ノードに割り当てたデータのグループラベルを発見的な方法で定義し，そして，教師付き学習を用いてテスト関数を求める．提案方法の有効性は，複数の公開データベースを用いた実験によって実証された．

A Fast Algorithm for Inducing Neural Network Trees

Hirotomo Hayashi[†1] and Qiangfu Zhao[†1]

Neural network tree (NNTree) is a decision tree (DT) with each internal node containing a small neural network (NN). Although NNTree is a model good for structural learning and for hardware implementation, it is difficult to induce suitable structure of NNTrees. Even if each NN contains only one neuron, the problem for finding the optimal test function in each internal node is NP-complete. To solve this problem, we have tried to induce the NNTrees using genetic algorithm (GA). The GA-based approach, however, is very time consuming, and cannot be used easily. In this paper, we propose a new algorithm for inducing NNTrees quickly. The basic idea is to define the group labels for the data assigned to each internal node based on some heuristic rules, and then find the test function through supervised learning. The efficiency of the proposed algorithm is proved through experiments on several public databases.

†1 会津大学大学院コンピュータ理工学研究科
Graduate School of Computer Science and Engineering, The University of Aizu

1. はじめに

ニューラルネットワーク（NN: Neural Network）とは脳神経系をモデルとした情報処理システムのことであり，文字認識，音声認識といったパターン認識やデータマインニング等さまざまな分野において応用されている．というのも，NN は高次元で，かつ線形分離不可能な問題に対してしばしば良好な解を得られるからである．しかしながら，NN の構造をどのように決定したらよいか事前に決めることは困難で現実的には試行錯誤になることが多く，扱いにくい面がある．

これまでにネットワーク構造を自動的に決定する方法はいくつかの研究がなされている[1)–4)]．Optimal Brain Damage（OBD）[4)] に代表されるような Pruning Method，Cascade Correlation（CC）[3)] に代表されるような Incremental Method 等である．しかしながら，それらはステップ・バイ・ステップで内部構造を変化させながら学習を繰り返す方法である．Pruning Method では十分なサイズのネットワークから開始し，学習後，不必要であると判定される隠れニューロンやユニット間の重み係数を削除し，再学習を行う．この内部構造の簡素化は，通常，再学習が収束しなくなるまで繰り返される．この手法の問題点は十分大きなネットワークから始めるので，何度も再学習を行う必要があるため，学習に大変時間がかかることである．事前にある程度の予測ができれば適切なネットワークに近い状態から始めることも可能であろうが，これは経験に基づいて決めざるをえない．Incremental Method では Pruning method とは逆に最小のネットワーク構造に徐々に隠れニューロンを増やし，再学習を行う方法である．十分に学習ができるようになるまで隠れニューロンを追加していく．最小のネットワーク構造から始まるため小規模のネットワークで解決できる問題に対して学習スピードは Pruning method よりも速いと考えられるが，多くの隠れニューロンを必要とする場合やはり時間がかかる．

また NN をニューロアクセレータとして実現しようと考えたときにニューラルネットワーク本来の並列性を維持して実装するにはある程度の規模の並列化が必要となる．その手段は 1 ニューロンに 1 PE（Processing Element）を割り当てる実ニューロン方式と 1 PE に複数のニューロンを割り当て，それらを相互に複数結合し時分割して用いる仮想ニューロン方式が考えられるが，実ニューロン方式ではハードウェアの規模が大きくなるため，実ニューロン方式の方が高速に動作することが期待できるにもかかわらず仮想ニューロン方式を選択せざるをえないのが現状である．

そこで，我々はニューラルネットツリー（NNTree: Neural Network Tree）について考

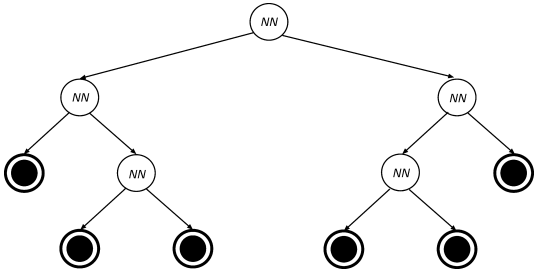

図 1　ニューラルネットツリーの例
Fig. 1　An example of neural network trees.

察する（**図 1** を参照）．NNTree は，特殊な決定木（DT: Decision Tree）であり，各中間ノードに対応するテスト関数は小規模な NN によって実現される．NNTree の利点としては，システムの構造は木の生成とともに自動的に決定されること，生成結果はハードウェアで容易に実現できること等がある．

NNTree を生成する際に，与えられた問題に対して良い解が得られるまで木を成長させていき，訓練データを十分に分割できるようになるまで木に小規模な NN を追加し枝を伸ばしていく．このように NNTree の構造は生成するときに動的に決められるので試行錯誤する必要がなくなる．各小規模な NN については，その構造をあらかじめ決めておいて，同じ構造を持つものを複数コピーし，結合係数だけを変更すれば利用可能となる．したがって，本研究では NNTree の中のモジュールとして使われている小規模な NN の構造を決定することを問題視しない．

フル結合型 MLP を実ニューロン方式でフル並列的に実現する場合，回路規模が非常に大きくなるので，ニューラルネットの実現には仮想ニューロン方式の方が現実的であると思われる．しかし，システムの規模が大きくなればそれに比例して仮想ニューロンを増やしていかなければならない．実際の PE 数は固定されているので，仮想ニューロンが増えれば，任意のパターンを認識するために必要な計算時間（計算量）はニューロン数に比例して長く（大きく）なる．これに対して，NNTree では，各中間ノードに使われる NN を同じ数の PE で実現した場合，計算時間は木の平均レベル数に比例する．クラス数を N_c とすれば，ニューロン数は通常 N_c に比例するが，木の平均レベル数は $\log_2 N_c$ に比例する．したがって，NNTree とフル結合型 MLP を同じ規模の回路で実現された場合，前者はオーダ的に速いと考えられる．

しかし，NNTree のような多変量決定木を生成するためのコストが一般的に高いため，あまり使われていないのが現状である．実際，最適な多変量テスト関数を求める問題は NP 完全である[5]．この問題を解決するために，発見的探索を利用する方法がいくつか提案され，その中で最も効率が良いとされるものは OC1[6] である．しかし，OC1 に使われている方法は，斜めの超平面を求めるのに比較的に有効であるが，NN のような一般的な多変量テスト関数を生成するためには利用できない．

多変量テスト関数を求める方法はこれまですべて「生成と評価」（Generate and Evaluate）に基づくものである．すなわち，可能な限り大量な多変量テスト関数の候補を生成して，それらをある評価関数（たとえば，情報利得率）で評価する．評価値の一番大きいものを最終結果として使う．通常の軸平行型決定木（APDT: Axis Parallel Decision Tree）を生成する場合，可能な単一変量テスト関数を全部調べ尽くしてもそれほど時間がかからない．しかし，多変量決定木を生成する場合，可能な多変量テスト関数は無限にあるため，調べ尽くすことは不可能である．

効率良く「生成と評価」を行うために，進化的アルゴリズム（EA: Evolutionary Algorithm）が考えられる．実際に，多変量決定木の一種である傾斜型決定木（ODT: Oblique Decision Tree）を生成するために EA に基づくアルゴリズムが提案されている[7]．我々も NNTree を生成するために遺伝的アルゴリズム（GA: Genetic Algorithm）に基づく方法を提案した[8]．しかし，EA に基づく方法はスケーラビリティ（Scalability）がないため，現実問題を解くためにあまり利用できない．

本論文では，従来とまったく違う方法を提案する．この方法では，まず各中間ノードに割り当てるデータのグループラベル（教師信号）を発見的手法で定義する．そして，教師付き学習アルゴリズムを使って多変数テスト関数を求める．NNTree を生成する場合，多変数テスト関数は NN であるので，それを求めるのに誤差逆伝搬法（BP: Back Propagation）が使える．提案手法は，簡単で，高速に NNTree を生成することができる．その有効性は複数のデータベースを利用した実験で確認する．

本論文の構成は次のとおりである．2 章では決定木の基礎知識と，我々が以前に提案した GA に基づく NNTree の生成方法について説明する．3 章はデータの教師信号を定義するための発見的方法を提案し，それに基づく NNTree の生成方法を紹介する．4 章では実験結果を示し，従来の方法と比較して提案方法の性能について考察する．5 章はまとめである．

2. 基 礎 知 識

2.1 決定木の定義

決定木はサイクルのない有向グラフである．木の最初のノードはルートである．通常，決定木はルートが一番上にあるように描かれる（図 1 を参照）．各ノード（ルートを除いて）の上には 1 つのノードがあり，それは親ノートである．ノードの下にあるノードは子ノードである．子ノードがないノードは終端ノード（葉）であり，それ以外のものは中間（非終端）ノードである．本論文では，ノードを以下のように定義する：

$$node = \{I, F, Y, N, L\} \tag{1}$$

ただし，I はノードの識別番号，F はテスト関数，Y は子ノードへのポインタの集合，$N = |Y|$ は子ノードの数，L はノードのクラスラベルである．なお，ノートが中間ノードである場合，L は未定義である．また，ノードが終端ノードである場合，F は未定義で，かつ Y は空集合である．

決定木を使って未知のパターン $\boldsymbol{x}$ を認識する過程は以下のようになる：

- Step 1：ルートを初期の「現在ノード」とする．
- Step 2：現在ノードが終端ノードである場合，現在ノードのラベルを $\boldsymbol{x}$ に与え，終了する．そうでなければ，テスト関数 $F(\boldsymbol{x})$ を使って，次に探索する子ノードを決める．
- Step 3：i（$= F(\boldsymbol{x})$）番目の子ノードを現在ノードとして，Step 2 に戻る．

2.2 決定木の生成

決定木を生成するためには，まず訓練集合を用意する．決定木は訓練集合を再帰的に分割することによって生成される．生成過程において主にノードの分割，終端ノードの判断，終端ノードのクラスラベルの決定の 3 つの操作がある．その中で最も重要であり，かつ時間がかかるものはノードの分割である．ノードを分割するために良いテスト関数を見つける必要がある．テスト関数の「良さ」を評価する基準がいくつか提案されている[9),10)]が，生成された決定木の性能はテスト関数の評価基準にあまり影響されないことが知られている[10)]．本研究において我々は情報利得率（IGR: Information Gain Ratio）を採用する．IGR はもともと Quinlan により提案され，よく知られている決定木生成プログラムである C4.5 で使われている[9)]．

IGR を最大にするようなテスト関数 $F(\boldsymbol{x})$ は，現在ノードに割り当てたデータを $F(\boldsymbol{x})$ を基にして分割すると，未知のデータを認識するための平均情報量は最小となる．いま，S は現在ノードに割り当てたデータの集合（$|S|$ はそのサイズである）であり，n_i は i 番目のクラスに属するデータの数（$i = 1, 2, \cdots, N_c$, N_c はクラス数である）であるとする．未知のデータを認識するための平均情報量（エントロピー）は以下のようになる：

$$info(S) = -\sum_{i=1}^{N_c} \frac{n_i}{|S|} \times \log_2 \left(\frac{n_i}{|S|} \right). \tag{2}$$

S をテスト関数 F によって N 個の部分集合 $S_1, S_2, \cdots, S_N$ に分割したとする．このときの情報利得は

$$gain(F) = info(S) - info_F(S) \tag{3}$$

で与える．ただし，

$$info_F(S) = \sum_{i=1}^{N} \frac{|S_i|}{|S|} \times info(S_i). \tag{4}$$

情報利得率（IGR）を以下のように定義する：

$$gain\ ratio(F) = gain(F)/split\ info(F) \tag{5}$$

ただし，

$$split\ info(F) = -\sum_{i=1}^{N} \frac{|S_i|}{|S|} \times \log_2 \left(\frac{|S_i|}{|S|} \right). \tag{6}$$

2.3 NNTrees の定義とその進化的生成方法

前述のように，NNTree は多変量決定木の一種であり，各中間ノードに使われているテスト関数は小規模な NN によって実現される．この小規模な NN の構造については特に制限はないが，本研究では，**図 2** に示すような階層型ニューラルネットを使う．NN の入力数と出力数はそれぞれ特徴の数 N_d と子ノードの数 N に対応する．また，本研究の主旨は複数の小さい NN を決定木に組み込むことにより複雑な問題を解決することであるため，NN の隠れニューロン数 N_h は，小さい数である．本論文においては，$N_h = 4$ とする．

NNTree を利用する場合，未知のパターン $\boldsymbol{x}$ の認識過程は以下のようになる．

- Step 1：ルートを初期の「現在ノード」とする．
- Step 2：現在ノードが終端ノードである場合，現在ノードのラベルを $\boldsymbol{x}$ に与え，終了

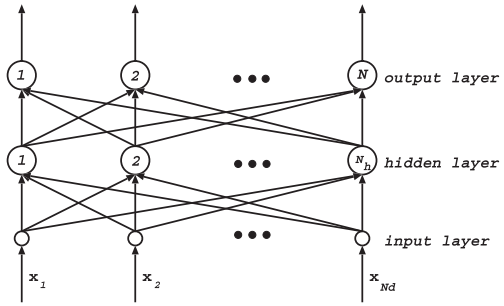

図 2　本研究に使われる NN の構造
Fig. 2　Structure of the NN used in this study.

する．そうでなければ，

$$i = F(\boldsymbol{x}) = \arg \max_{1 \le k \le N} o_k \tag{7}$$

を用いて次に探索する子ノードを見つける．ここで o_k は NN の k 番目の出力である．

- Step 3：i 番目の子ノードを現在ノードとして，Step 2 に戻る．

NNTree を生成するために，C4.5 を生成するプロセスとほぼ同じプロセスが利用できる．唯一の相違点は，テスト関数 $F(\boldsymbol{x})$ の求め方である．NNTree の場合，テスト関数は NN であるため，通常の「生成と評価」に基づく方法はあまり効率的ではない．一方，どのデータをどの子ノードに割り当てるかに関する事前情報はないため，教師付き学習もできない．したがって，我々は遺伝的アルゴリズム（GA）に基づく方法を提案した[8]．

GA には通常，選択，交叉，突然変異の 3 つの遺伝的操作がある．我々の研究では切捨て選択，一点交叉，ビットごと突然変異を使う．NN である個体の遺伝子型は 2 進数で表現した重み係数を並べたリストである．個体の適応度を IGR と定義すれば，最適なテスト関数が進化によって得られる．

NNTree を再帰的に生成する際に，GA は各中間ノードの NN の重み係数を求めるのに使う．したがって，GA は生成過程の中でサブルーチンとして呼び出される．個体を NN ではなく，NNTree 全体にすることも可能である[11]が，この場合，探索空間が非常に大きく，良い NNTree を得るために，膨大な計算時間が必要とされるので，本論文ではこの方法を割愛する．

3. 教師付き学習に基づく NNTrees の高速生成法

最適なテスト関数を実現する NN を設計するために我々が GA を使ってきた主な理由は，各中間ノードに対してデータをどのように分けたらよいかに関して，事前情報はないからである．GA に基づく方法は理論的には大変強力であるが，現実的には大変時間がかかるので，あまり実用的ではない．本論文では，データを事前に分けることによって，NN を求める問題を教師付き学習問題に帰着させる．以下は，データを分けるための一方法である[12]：

S を現在ノードに割り当てられたデータの集合であるとする．S を N 個のサブセット $S_1, S_2, \cdots, S_N$ に分けたいと仮定する．各サブセットの初期状態は空集合である．S が空集合になるまで，以下のことを繰り返す：

- Step 1：任意のデータ $\boldsymbol{x}$ を S から取り出す．
- Step 2：もし S_i に $label(\boldsymbol{y}) = label(\boldsymbol{x})$ を満たす $\boldsymbol{y}$ が存在するならば，$\boldsymbol{x}$ を S_i に割り当てる．そうでないならば，Step 3 に移動する．
- Step 3：もし空集合の S_i が存在するならば，$\boldsymbol{x}$ を S_i に割り当てる．そうでないならば，Step 4 に移動する．
- Step 4：$\cup S_i$ の中で $\boldsymbol{x}$ の最近傍である $\boldsymbol{y}$ を見つけ，$\boldsymbol{x}$ を $\boldsymbol{y}$ と同じサブセットに割り当てる．

これをフローチャートに表すと**図 3** となる．任意のデータ $\boldsymbol{x}$ の教師信号は，それが割り当てられたサブセットの番号になる．すなわち，$\boldsymbol{x}$ が S_i に割り当てれた場合，その教師信号は i となる．すべてのデータの教師信号が分かると，テスト関数を実現する NN を求める問題は教師付き学習となる．本研究では，学習のためによく知られている誤差逆伝搬法（BP）を使用する．

以上のグループ分けは 2 つの原則に基づいている．第 1 に，同じクラスのデータをできるだけ同じサブセットに分けたい．第 2 に，特徴空間において「近い」データをできるだけ同じサブセットに分けたい．データ間の近さは通常ユークリッド距離で計る（他の距離の定義を使うこともできる）．このようなグループ分けは，以下の特徴がある．まず，同じクラスのデータを同じサブセットに分けると，分割にともなう情報量の利得は大きくなる．これは，既存の評価基準に一致する．また，近いデータを同じサブセットに分けると，決定境界は簡単なものとなり，それを実現するために小さい NN が使える．当然，以上の方法は，

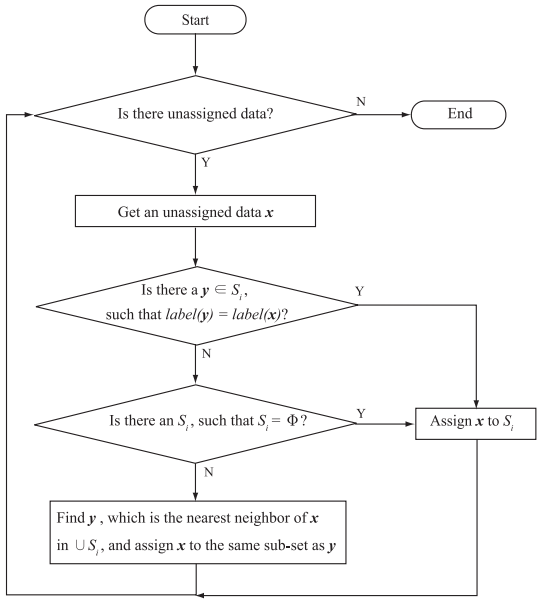

図 3　グループラベルを割り当てるために提案した方法のフローチャート
Fig. 3　Flowchart of the proposed method for assigning group labels.

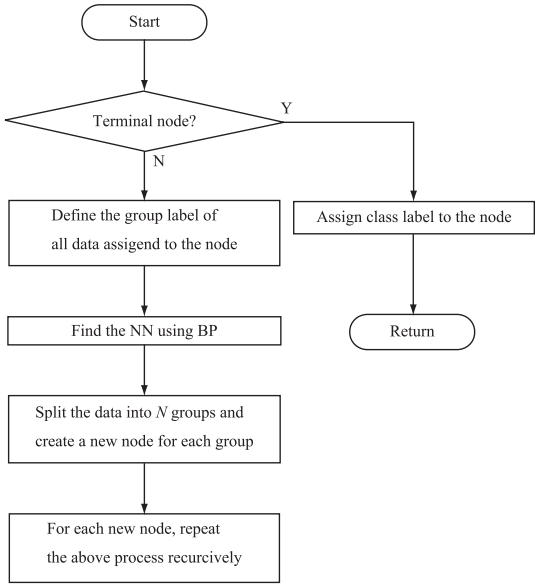

図 4　提案方法に基づく NNTree を生成するフローチャート
Fig. 4　Flowchart for inducing the NNTrees using the proposed method.

あくまでも発見的（Heuristic）方法であり，最適なものである保証はない．

図 4 は提案する方法のフローチャートである．このフローチャートに含まれている操作は現在ノードに関するものである．最初は，ルートを現在ノードとし，それにすべてのデータを割り当てる．現在ノードに割り当てられたデータがすべて同じクラスである場合，そのノードはそれ以上分ける必要がないので終端ノードである判定し，そのラベルを定義してから呼び出されたところに戻る．実際，すべてのデータが同じクラスにならなくても，大部分のデータが同じクラスに属すれば，現在ノードを終端ノードにすることができる．この場合，終端ノードのラベルは（多数決で）最もデータ数が多いクラスと決める．

また，現在ノードは終端ノードではないと判定された場合，それを中間ノードとして，そのテスト関数を求め，割り当てたデータを N 個のサブセットに分割する．そして，それぞれのサブセットに割り当てられたデータを子ノードに割り当て，それぞれの子ノードを順番に現在ノードとして同じプロセスを再帰的に実行する．子ノードがすべて終端ノードとなったら木の生成は終了である．

図 4 の各処理をアルゴリズムの形で説明すると以下のようになる．

- Step 1：現在ノードが終端ノードであるか判定する．その判定基準は現在ノードに割り当てられたデータが複数のクラスを含むかどうかである．単一のクラスであった場合，終端ノードと判定し，Step 2 へ行き，複数クラスを含むのであれば，中間ノードと判定し，Step 3 へ行く．
- Step 2：現在ノードに割り当てられたデータで最も数の多いクラスをこの終端ノードに割り当て，このパスの処理を終了する．
- Step 3：現在ノードに割り当てられたデータをどちらの子ノードに割り当てたらよいかを決める．その結果は BP の教師信号として使用する．
- Step 4：Step 3 で得られた教師信号を基に，BP アルゴリズムで NN を設計する．
- Step 5：現在ノードに割り当てられたすべてのデータを Step 4 で設計した NN に入力

させ，NN の出力に従って N 個のサブセットを求める．各サブセットに対応して 1 つの子ノードを作る．

- Step 6：現在ノードの子ノードを現在ノードとして同様の処理を再帰的に行う．

4. 実験と考察

本論文で提案した方法の有効性を確認するために，カルフォルニア大学アーヴァイン校の公開データベースから得たデータベースを使い，実験を行った．使用したデータベースは car，crx，dermatology，ecoli，housevotes84，ionosphere，iris，optdigits，pen-based，tic-tac-toe の 10 個である．**表 1** にデータベースのパラメータを示す．すべての実験において，10-fold クロスバリデーション（cross validation）を行い，それを 5 回を行った．実験に使用したコンピュータはサンワークステーションの SunJavaWorkstationW1100z モデル，1 CPU，1.8 GHz AMD Optron 144 である．

ここでは，以下の結果について比較する：

- 提案する方法で生成した NNTree
- GA に基づく方法で生成した NNTree
- C4.5 で生成した単一変量決定木（APDT）
- OC1 で生成した傾斜型決定木（ODT）
- BP で得られたフル結合型 NN

表 1　データベースのパラメータ
Table 1　Parameters of the databases.

	Number of examples (N_t)	Number of features (N_d)	Number of classes (N_c)
car	1,728	6	4
crx	690	15	2
dermatology	366	34	6
ecoli	336	7	8
housevotes84	435	16	2
ionosphere	351	34	2
iris	150	4	3
optdigits	5,620	64	10
pen-based	10,992	16	10
tic-tac-toe	958	9	2

- OBD で得られた NN
- CC で得られた NN

実験で使用したパラメータは以下のようになる．これらのパラメータは最適なものではないかもしれないが，実験を通して得られた最良のものを選択している．

- GA に基づく NNTree の生成方法に関するパラメータ
  （1） 世代数更新数 = 1,000
  （2） 個体数 = 200
  （3） 各重みのビット数 = 16
  （4） 選択率 = 0.2
  （5） 交叉率 = 0.7
  （6） 突然変異率 = 0.01
- 提案方法のパラメータ
  （1） BP に使う学習率 = 0.5
  （2） 学習回数（エポック数）= 1,000
  （3） NN の入力数 = N_d（特徴数）
  （4） 隠れ層ニューロン数 = 4
  （5） 出力ニューロン数（枝の数）= 2
- BP に基づくフル結合型 NN を設計する際のパラメータ
  （1） 入力数 = N_d
  （2） 出力ニューロン数 = N_c（クラス数）
  （3） 学習回数 = 1,000
  （4） 学習率 = 0.5
  （5） 隠れ層ニューロン数 = 提案した方法で得られた NNTree に使われる隠れニューロンの総数の平均値（隠れ層ニューロン数の求め方の一例を示す．NNTree は終端ノードに NN を持たないので，隠れニューロンの総数は中間ノード数と NNTree の NN の隠れニューロン数の積である．car の場合，NNTree の木のサイズは平均約 15 であるので，中間ノード数は約 7 となり，NNTree の NN の隠れニューロ数は 4 であるので，総隠れニューロン数 = $7 \times 4 = 28$ となる）
- Optimal Brain Damage のパラメータ
  （1） 入力数 = N_d
  （2） 初期隠れ層ニューロン数 = 提案した方法で得られた NNTree に使われる隠れ

表 2　提案した方法の結果
Table 2　Results obtained by the proposed method.

	Error Rate (%)	Tree Size	Time (second)
car	2.01 ± 0.42	15.04 ± 1.07	2.2 ± 0.23
crx	16.67 ± 1.34	16.2 ± 0.87	4.17 ± 0.16
dermatology	3.06 ± 0.78	13.84 ± 0.5	0.07 ± 0.01
ecoli	15.15 ± 1.49	15.04 ± 0.83	1.09 ± 0.04
housevotes84	5.72 ± 0.89	5.16 ± 0.47	0.25 ± 0.04
ionosphere	8.23 ± 1.23	5.32 ± 0.64	0.51 ± 0.08
iris	4.27 ± 1.38	5.56 ± 0.34	0.18 ± 0.01
optdigits	4.18 ± 0.25	30.76 ± 2.05	1.77 ± 0.19
pen-based	2.59 ± 0.16	31.0 ± 2.21	4.99 ± 1.18
tic-tac-toe	0.86± 0.23	5.2 ± 0.28	0.12 ± 0.02

表 3　GA に基づく方法の結果
Table 3　Results obtained by the GA-based approach.

	Error Rate (%)	Tree Size	Time (second)
car	5.83 ± 0.54	45.44 ± 1.74	605.46 ± 31.78
crx	20.43 ± 1.39	58.68 ± 1.52	389.53 ± 9.81
dermatology	6.61 ± 0.99	11.92 ± 0.28	13.29 ± 1.04
ecoli	21.76 ± 1.91	49.84 ± 1.13	79.79 ± 2.49
housevotes84	8.23 ± 1.14	12.56 ± 0.6	55.19 ± 3.29
ionosphere	8.74 ± 1.42	11.88 ± 0.52	82.72 ± 4.2
iris	4.4 ± 1.33	7.16 ± 0.33	4.77 ± 0.48
optdigits	5.1 ± 0.24	107.32 ± 2.03	3,231.77 ± 104.05
pen-based	2.04 ± 0.13	132.48 ± 2.99	3,682.71 ± 115.21
tic-tac-toe	5.24 ± 0.74	18.28 ± 0.67	186.43 ± 10.43

ニューロンの総数の平均値

(3)　出力ニューロン数 = N_c（クラス数）

(4)　初回学習回数 = 1,000

(5)　学習率 = 0.2

(6)　再学習回数 = 100

- Cascade-Correlation のパラメータ

(1)　入力数 = N_d

(2)　出力ニューロン数 = N_c（クラス数）

(3)　入力重み係数の学習回数 = 200

(4)　出力重み係数の学習回数 = 200

(5)　最大隠れニューロン数 = 200

表 2〜表 8 に 10 個のデータベースに対する実験結果を示す．これらの表では，学習後のテストセットに対するエラー率（%），木のサイズ（すべてのノードの数），学習に要した計算時間（秒）を示す．階層型ニューラルネットは決定木ではないので木のサイズではなく内部の隠れニューロン数を示す．また，表 9 に提案手法，OBD，CC それぞれの総ニューロン数を示す．総ニューロン数は隠れ層ニューロンと出力層ニューロンの総計である．入力層にはニューロンは存在しないので含めない．なお，訓練データを完全に識別できるように NNTree を生成しているので，訓練データに対するエラー率はここでは与えない．各項目 2 段で表されている数値は，上段の値が平均値であり，下段が 95%の信頼区間である．これらの結果から次のような結論が出されるのではないかと思われる．

(1)　まず提案した方法と GA に基づく方法を比較する．**表 2** と**表 3** から分かるように，GA に基づく方法より，提案した方法は平均で数百倍のスピードで NNTree を構築することができる．また，提案した方法で得られた NNTree はほとんどの場合にエラー率も木のサイズも小さい．これらの結果から，NNTree を生成することに関して

表 4　OC1 の結果
Table 4　Results of OC1.

	Error Rate (%)	Tree Size	Time (second)
car	5.08 ± 0.53	46.96 ± 4.32	11.73 ± 0.19
crx	15.36 ± 1.2	7.36 ± 1.71	4.29 ± 0.13
dermatology	8.33 ± 1.72	11.12 ± 0.48	1.9 ± 0.06
ecoli	19.28 ± 1.72	12.48 ± 2.01	3.63 ± 0.08
housevotes84	5.33 ± 1.00	6.72 ± 1.73	1.63 ± 0.05
ionosphere	12.25 ± 1.55	10 ± 1.66	3.77 ± 0.13
iris	4.4 ± 1.47	5.4 ± 0.37	0.22 ± 0.01
optdigits	9.04 ± 0.39	128 ± 13.44	471.45 ± 4.78
pen-based	3.03 ± 0.14	157.08 ± 10.34	551.31 ± 5.94
tic-tac-toe	9.43 ± 1.08	28.84 ± 3.2	4.17 ± 0.19

表 5　C4.5 の結果
Table 5　Results of C4.5.

	Error Rate (%)	Tree Size	Time (second)
car	7.94 ± 0.53	171.5 ± 2.15	0.01 ± 0
crx	14.03 ± 0.92	27.16 ± 2.98	0.02 ± 0
dermatology	4.05 ± 0.77	15.08 ± 0.11	0.02 ± 0
ecoli	18.84 ± 1.86	36.4 ± 1.72	0.01 ± 0
housevotes84	3.45 ± 0.74	10.64 ± 0.22	0.01 ± 0
ionosphere	9.63 ± 1.26	27.16 ± 1.03	0.08 ± 0
iris	6.14 ± 1.53	8 ± 0.38	0.01 ± 0
optdigits	9.64 ± 0.42	412.12 ± 3.93	0.96 ± 0
pen-based	3.63 ± 0.18	377.32 ± 3.32	0.66 ± 0
tic-tac-toe	14.36 ± 1.01	132.04 ± 2.87	0.01 ± 0

いえば，提案した方法が GA に基づく方法より良いといえる．

(2) 次に，提案した方法と OC1 を比較する．OC1 の結果は，傾斜型決定木（ODT）である．ODT は最も簡単な多変量決定木であり，軸平行型決定木（APDT）よりは効率的であるといわれている．表 2 と**表 4** から分かるように，データベース（たとえば，optdigits，pen-based）が大きい場合，NNTree は OC1 で得られた ODT より小さいし，エラー率も低い．また，大きいデータベースに対して，提案した方法の計算時間は OC1 より数百倍短い場合もある．ただ，データベースが小さい場合（たとえば，crx，housevotes84），多くのフリーパラメータを持つ NNTree を生成する十分な情報がない場合もあり，この場合，ODT と比べ，NNTree は必ずしも良くない．

(3) 多変量決定木（NNTree と ODT）と単一変量決定木（APDT）の性能を比較するために，表 2，表 4，**表 5** を参照して説明する．一般論としては，APDT よりも多変量決定木の方はノード数が小さい．しかし，エラー率に関しては多変量決定木は必ずしもよくない．ほとんどの場合 NNTree は APDT より良いが，多くの場合 ODT は APDT よりも悪い．一般的に，多変量決定木は中間ノードにおいてより複雑な判断をするため APDT よりも優れている可能性が高いと考えられるが，データの数が十分でなければ，良いシステムパラメータ（たとえば，NN の重み）を得ることは容易ではない．少ないデータによって多くのパラメータを推定しようとすると，オーバフィット（over-fit）となりやすい．したがって，NNTree はデータの数が十分に大きいときにのみ使用するべきであると思われる．

(4) 次に NNTree と単一フル結合型 NN を比較する．表 2 と**表 6** から分かるように，テストデータに対するエラー率に関していうと NNTree は 4 勝 6 敗であるが，NNTree とフル結合型 NN の間に大きな差なく同程度であるといえる．ただ，NNTree を使う利点として次のことがあげられる．まず，NNTree の構造は生成の過程を通して自動的に決定することができる．反対に，フル結合型 NN の構造（たとえば，隠れ層

表 6　BP 学習により得られたフル結合型 NN の結果
Table 6　Results of fully connected NNs obtained through BP learning.

	Error Rate (%)	Number of Hidden Neuron	Time (second)
car	2.21 ± 0.31	28 ± 0	1.51 ± 0.08
crx	16.29 ± 1.19	32 ± 0	6.56 ± 0.03
dermatology	2.67 ± 0.78	24 ± 0	0.11 ± 0.01
ecoli	13.88 ± 1.59	28 ± 0	3.83 ± 0
housevotes84	5.35 ± 0.95	8 ± 0	0.47 ± 0.12
ionosphere	9.26 ± 1.23	8 ± 0	0.31 ± 0.07
iris	3.2 ± 1.13	8 ± 0	0.27 ± 0.02
optdigits	2.16 ± 0.17	60 ± 0	5.27 ± 0.15
pen-based	4.51 ± 0.18	60 ± 0	303.84 ± 3.65
tic-tac-toe	0.97 ± 0.28	8 ± 0	0.04 ± 0

ニューロン数）を決めることは容易ではない．表 6 は 1 つ興味深いことを示している．前述のように，今回の実験においては，フル結合型 NN の隠れニューロン数は提案した方法で生成された NNTree に使用された隠れニューロンの総数の平均値とした．この値は最適ではないが，得られたフル結合型 NN の性能は適当に選んだにしては良いと考えられる．どうしてもフル結合型 NN を使いたい場合，提案した方法は初期隠れニューロン数を決める方法として使用することができるかもしれないと思われる．すなわち，まず提案方法で NNTree を生成し，それを基に隠れニューロン数を求め，次に BP を利用してフル結合型 NN を設計し，試行錯誤で隠れニューロン数を調整することも可能であろう．しかし，やはり試行錯誤は必要であるので積極的にフル結合型 NN を使う理由はあまりないのではないかと考えられる．

(5)　NNTree と OBD（Optimal Brain Damage）を比較するため，表 2，**表 7**，**表 9** を見る．まず表 9 の総ニューロン数に着目すると，OBD の方がすべてのデータベースで少なくなった．しかし，表 2 と表 7 のエラー率を見ると optdigits を除く 9 つのデータベースで NNTree の方が小さく，計算時間は一見して明らかなようにすべてのデータベースで OBD の方が計算時間がかかっている．

これは OBD では貢献度の低い重み係数を計算しそれを削除し，再度学習を行うため，内部のネットワークがある程度複雑であれば十分にネットワークが整理されるまで何度も繰り返して再学習を行わなければならないからであり，大変時間がかかる可能性もある．OBD の隠れニューロン数の初期値は NNTree の結果を利用して与えたので，十分に大きい値を与えて始めるよりも速くなっているはずであるが，そうでなければさらに時間が必要となる．

表 7　OBD の結果
Table 7　Result of Optimal Brain Damage.

	Error Rate (%)	Time (second)
car	3.59 ± 1.04	68.92 ± 11.88
crx	20.35 ± 3.06	18.60 ± 6.82
dermatology	18.67 ± 2.32	313.06 ± 35.30
ecoli	27.33 ± 7.68	69.02 ± 3.21
housevotes84	6.47 ± 1.13	9.71 ± 0.59
ionosphere	12.40 ± 2.14	15.79 ± 1.54
iris	4.93 ± 2.53	0.68 ± 0.09
optdigits	3.71 ± 0.32	675.82 ± 122.47
pen-based	2.70 ± 0.29	25,203.88 ± 1,032.07
tic-tac-toe	8.29 ± 2.83	4.96 ± 0.50

(6)　次に NNTree と CC（Cascade-Correlation）を比較するため，表 2，**表 8**，表 9 を見る．表 9 から総ニューロン数はすべてのデータベースで CC の方が少ないが，表 2 と表 8 からエラー率はすべてのデータベースに対して高いことが分かる．学習速度は NNTree の方が速いもの，CC の方が速いものそれぞれ 5 対 5 でデータベース

表 8　CC の結果
Table 8　Result of Cascade-correlation.

	Error Rate (%)	Time (second)
car	9.69 ± 2.24	7.04 ± 0.28
crx	22.03 ± 3.40	2.76 ± 0.16
dermatology	6.44 ± 1.39	0.06 ± 0.01
ecoli	37.21 ± 7.78	2.55 ± 0.12
housevotes84	8.14 ± 1.78	0.22 ± 0.02
ionosphere	15.71 ± 2.81	0.29 ± 0.03
iris	5.60 ± 2.26	0.12 ± 0.02
optdigits	11.13 ± 1.42	111.61 ± 2.22
pen-based	4.80 ± 0.89	334.11 ± 6.30
tic-tac-toe	8.23 ± 2.13	1.24 ± 0.07

によるところが大きいように思われるが，データ数が多いデータベース（optdigits, pen-based）についていえば，その差は明らかで，NNTree の方が速い．CC はネットワークを自動的に構築するために繰り返し学習を行うためで，データ数が多ければ学習時間は長くなる．提案手法の方は木構造をとっているためノードを再帰的に作る必要があるが，それ自体は 1 回のパスで終了するため全体で見れば繰り返し学習をする必要はない．

（7） 最後に，実験結果を基に提案手法で得られた NNTree と OBD で得られた MLP（以下，OBD-MLP と略記する）を同じ数の PE で実装した場合のそれぞれの計算回数を比較する．PE 数は NNTree に合わせるものとし，NNTree の中間ノードに割り当てられた NN を並列動作させるには，PE は 4 つあれば十分なので PE 数 $N_p = 4$ とする．また，ここでは NNTree の NN の隠れニューロン数を 4，出力ニューロンを 1 と想定する（2 分木であるので，理論上出力ニューロンは 1 でも 2 分木は構成可能であり，実際に出力ニューロンを 1 にして行った実験でも同等の結果が得られている）．

表 9　各手法の総ニューロン数の比較
Table 9　Comparison of total number of neurons obtained by each method.

	Proposed Method	OBD	CC
car	42.12 ± 3.22	27.88 ± 1.43	23.82 ± 0.70
crx	45.60 ± 2.62	32.88 ± 1.45	23.96 ± 1.29
dermatology	38.52 ± 1.53	17.62 ± 1.64	6.00 ± 0.10
ecoli	42.12 ± 2.49	22.66 ± 1.12	33.86 ± 1.07
housevotes84	12.48 ± 1.42	6.00 ± 0.39	5.04 ± 0.23
ionosphere	12.96 ± 1.92	6.18 ± 0.40	5.28 ± 0.25
iris	13.68 ± 1.02	10.08 ± 0.60	7.94 ± 0.45
optdigits	89.28 ± 6.15	68.60 ± 2.76	42.66 ± 0.67
pen-based	90.00 ± 6.64	45.76 ± 2.07	70.42 ± 0.83
tic-tac-toe	12.60 ± 0.85	8.82 ± 0.47	10.36 ± 0.36

NNTree で認識するために必要な計算回数は中間ノードに割り当てられた NN の計算回数と NNTree の平均レベル数の積であり，NN の計算回数は，中間層の計算と出力層の計算の合計 2 である（各層のニューロンが並列動作するため計算時間は 1 単位とする）ので，NNTree の計算回数は「2 × NNTree の平均レベル数」である．optdigits の場合，表 2 より NNTree の木のサイズ（総ノード数）は約 31 であるので平均レベル数は 4 である．以上より NNTree で optdigits を認識するために必要な計算回数は 8 である．

OBD-MLP の PE が並列実行されるとき，計算回数は大体「総ニューロン数/N_p」と考えられる．表 9 より総ニューロン数は平均約 69 である．よって OBD-MLP の計算回数は $69/4 = 17.25$ となる．OBD-MLP のほうが約 2 倍多いことが分かる．表 9 を見ると NNTree の総ニューロン数の方が OBD-MLP の総ニューロン数よりも多いが，実際には NNTree の場合すべてのニューロンを計算する必要がないので NNTree

の方が計算回数は少なくなる．

一般的にはソフトウェア上でシミュレーションする場合等，$N_p = 1$ であるシステムを想定するケースは少なくないと思われる．この場合，optdigits で考えると，NNTree の計算回数は中間ノードに割り当てられた NN のニューロンの総数（$4 + 1 = 5$）と NNTree のレベル数（4）の積であるので 20 となる．一方，MLP の計算回数は総ニューロン数となるので MLP と NNTree の比率は $69/(5 \times 4) = 3.45$ 倍となる．したがって，リソースが限られている場合，NNTree のほうが有利である．

もちろんすべてのデータベースで NNTree の計算回数が少なくなるというわけではないが，NNTree はクラス数が多い問題に対して有益であると考えられる．実際，MLP の計算時間はニューロン数に比例し，ニューロン数は N_c（クラス数）に比例する．NNTree の場合，計算時間は木の平均レベル数に比例し，木の平均レベル数は $\log_2 N_c$ に比例するので，NNTree の方がオーダ的に速いと考えられる．

5. おわりに

本論文で，ニューラルネットツリー（NNTree）を高速に生成できるアルゴリズムを提案した．この方法では，まず各中間ノードに割り当てたデータのグループラベルを定義し，次に教師付き学習を用いてテスト関数を求めた．これによって，汎化能力が高い NNTree を高速に生成することに成功した．このことは，複数の公開データベースを用いた実験において，既存のいくつかの方法と比較することで実証された．

NN と DT を組み合わせて学習を行う方法は，本論文で提案した方法以外にもいくつかある．たとえば，文献 13)，14) においては，NN を木の形に直し，実時間で学習できる「決定木」を提案した．実際，これらのモデルは，DT ではなく，階層型 NN である．また，自己組織のメカニズムを木構造を利用して階層的に実現する方法もいくつか提案されている[15),16)]．これらの方法は，教師なし学習であり，本論文の結果と直接比較できないが，本論文の方法と結合することにより，準教師付き学習（Semi-supervised learning）は可能になるのではないかと考えられる．これについて今後詳しく検討していきたい．

謝辞　本研究の一部は日本学術振興会科学研究費補助金（No.19500128）の補助を得て行われたものである．

参考文献

1) Ash, T.: Dynamic Node Creation in Backpropagation Networks, *Connection Science*, Vol.1, No.4, pp.365–375 (1989).
2) Mozer, M. and Smolensky, P.: Using Relevance to Reduce Network Size Automatically, *Connection Science*, Vol.1, No.1, pp.3–16 (1989).
3) Fahlman, S., Lebiere, C. and Touretzky, D.: The Cascade-Correlation Learning Architecture, *Advances in Neural Information Processing Systems*, Vol.2, pp.524–532 (1990).
4) LeCun, Y., Denker, J. and Solla, S.: Optimal brain damage, *Advances in neural information processing systems*, Vol.2, pp.598–605 (1990).
5) Heath, D., Kasif, S. and Salzberg, S.: Induction of oblique decision trees, *Proc. 13th International Joint Conference on Artificial Intelligence*, pp.1002–1007 (1993).
6) Murthy, K.: On Growing Better Decision Trees from Data, Ph.D. Thesis, University of Maryland (1997).
7) Cantu-Paz, E. and Kamath, C.: Inducing oblique decision trees with evolutionary algorithms, *IEEE Trans. Evolutionary Computation*, Vol.7, No.1, pp.54–68 (2003).
8) Zhao, Q.: Evolutionary design of neural network tree – Integration of decision-tree, neural network and GA, *Proc. IEEE Congress on Evolutionary Computation*, pp.240–244 (2001).
9) Quinlan, J.: *C4.5: Programs for machine learning*, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA (1993).
10) Breiman, L., Friedman, J.H., Olshen, R.A. and Stong, C.J.: *Classification and Regression Trees*, Wadsworth Pub. Co. (1984).
11) Hayashi, H. and Zhao, Q.: A Comparative Study on GA Based and BP Based Induction of Neural Network Trees, *IEEE International Conference on Systems, Man and Cybernetics*, pp.822–826 (2005).
12) Zhao, Q.: A New Method for Efficient Design of Neural Network Trees, Technical Report of IEICE, Vol.PRMU2004-115, pp.59–64 (2004).
13) Golea, M. and Marchand, M.: A growth algorithm for neural network decision trees, *EuroPhys. Lett.*, Vol.12, No.3, pp.105–110 (1990).
14) Basak, J.: Online adaptive decision trees, *Neural Computation*, Vol.16, No.9, pp.1959–1981 (2004).
15) Basak, J. and Krishnapuram, R.: Interpretable hierarchical clustering by constructing an unsupervised decision tree, *IEEE Trans. Knowledge and Data Engineering*, Vol.17, No.1, pp.121–132 (2005).
16) Li, T., Fang, L. and Li, K.: Hierarchical classification and vector quantization with neural trees, *Neurocomputing*, Vol.5, No.2, pp.119–139 (1993).

(平成 19 年 12 月 27 日受付)
(平成 20 年 5 月 8 日採録)

林　　博友

2004 年会津大学コンピュータ理工学部コンピュータソフトウエア学科卒業．現在，同大学大学院博士後期課程に在学中．機械学習に関する研究に従事．

趙　　強福（正会員）

1982 年中国山東大学計算機科学科卒業．1988 年東北大学大学院電子工学専攻博士課程修了．工学博士．1991 年北京理工大学副教授．1993 年東北大学助教授．1995 年会津大学助教授．1999 年同大学教授，現在に至る．機械学習，パターン認識，ニューラル計算，進化計算，画像処理等に興味を持つ．IEEE，電子情報通信学会，計測自動制御学会，日本神経回路網学会各会員．1960 年生．