ニューラルネットワークシミュレータ用ツール SONNET

7F-6

若松 真　秋山 泰　篠沢 一彦　安西 祐一郎

慶応義塾大学 理工学部

1．はじめに

ニューラルネットワークの研究を行う際、計算機シミュレーションは必要不可欠なものとなっている。著者らは、ニューラルネットワークのシミュレータをC言語を用いて簡易に構築するためのツール SONNET(Simulation Outfit for Neural NETworks)を開発した。

SONNETは、ニューラルネットワークのシミュレーションおよび結果表示において必要とされる機能を、C言語から呼び出し可能なライブラリ群の形で提供する。以下ではSONNETの構成と機能を、記述例をまじえて紹介する。

2．SONNET 開発目的

大規模なニューラルネットワークのシミュレータを作成することは容易ではなく、研究の本質とは離れたプログラミング作業に多大な時間が消費されている。特に、結果を視覚的に表示する場合には、グラフィックインタフェースに対する習熟も要求され、さらに手間が増す。

最近、ニューラルネット・シミュレーションの専用言語／システムが少しずつ発表されているが、それらにもいくつかの問題点がある。まず、記述の容易性を追求するあまり、アルゴリズムの自由度が制限されている例が多い。有名なモデルを追試するだけならば十分かもしれないが、本当に必要なのは様々な新種のアルゴリズムを試せる環境である。また、これらの専用システムでは、動作可能なハードウェアや価格の面での制約も大きい。

そこで著者らは、シミュレータを記述する際に共通して必要となる部分を、「汎用のプログラミング言語から呼び出されるライブラリの形で供給する」ことが最も実用的な効果が大きいと考えた。この方法では、ユーザにその言語の(基礎的な)文法知識が要求される反面、記述の自由度は十分に確保され、研究者それぞれの個人的計算環境との親和性が格段に優れている。

本研究では、高速性や融通性の面から、ターゲットの汎用言語としてC言語を選び、全ツールをC言語から呼び出されるライブラリおよびヘッダーとして提供する。

開発にあたっては、以下に掲げる点を特に心がけた。

①記述されたプログラムの可読性が高く、研究者間での情報交換の媒体となり得るようにする。

②C言語の文法を変更せず、あくまでもライブラリとして供給することにより、ユーザによる改造を容易にする。（これは①とは矛盾する）

③シミュレータを書く上で最も手間のかかるニューロン間の結合関係の記述をとくに簡素化する。

④パソコン上での実現性も十分に考慮する。FPPが発達しコスト性能比が向上しているパソコンを活用することは、ニューラルネットワーク研究のパーソナル化を考える上で重要である。カラーディスプレイが完全普及していることからも利用価値は大きい。

3．SONNETの構成

SONNETは、シミュレータ・ライブラリとグラフィック・ライブラリとの2つの部分から構成されている。

①主要なシミュレータ・ライブラリ

(1)Generate(型, cluster名, ニューロン数)
ある型として定義されたニューロンの配列を生成し、1つのクラスタとして取り扱えるようにする。

(2)Connect(cluster1, cluster2, 荷重変数の個数)
cluster1とcluster2の間の完全結合を行う。

(3)InitialWeight(分布型, ﾊﾟﾗﾒﾀ1, ﾊﾟﾗﾒﾀ2)
Connect時の結合強度の初期分布を指定する。

(4)Propagate(cluster)
cluster内の全ニューロンの次の出力値を計算する。

(5)Sigma(neuron)
FPP（80287か80387）を用いた入力和高速演算関数。

②主要なグラフィック・ライブラリ

(1)WeightMap [引数は省略、以下同様]
ニューロン間の結合強度を四角形や円の大きさで表示。

(2)OutMap
ニューロンの出力値を四角形や円の大きさで表示。

(3)BarGraph
棒グラフを表示する。

(4)LineGraph
折れ線グラフを表示する。

現在のところ、グラフィック・ライブラリはパソコンPC9800を対象としたもののみ実装されている。

4．SONNETの特徴

①クラスタの概念

層状ネットワークの場合など、ニューロンの集まりを1つのクラスタとして扱うことで記述が大幅に簡素化できることがある。このため、SONNETではGenerate関数によって多数のニューロンを1つのクラスタとして生成し、ConnectやPropagateの際にそのクラスタ名を指示するだけでクラスタ内の全ニューロンが一括処理されるようにした。単一ニューロン用の関数も用意されているので、クラスタを用いないモデルで効率が落ちることはない。

②ユーザによる拡張の容易性

SONNETは、C言語（および一部アセンブリ言語）のソースコードとして供給されるため、各ユーザによる改造がきわめて簡単に行える。

a)データ構造の拡張

モデルが変わると1個のニューロンを表現するために必要になるデータ構造が変化する。ニューロンのモデルは次々に新しいものが出現するので、「どのようなモデルにも耐えられる汎用の構造」を初めから用意することは不可能である。SONNETでは、ニューロンのデータ構造はヘッダー"neuron.h"の中に記述される。ユーザは、このヘッダー内にあらかじめ登録されている数種の構造を

SONNET: A Tool for Constructing Neural Network Simulators
Makoto Wakamatsu, Yutaka Akiyama, Kazuhiko Shinozawa and Yuichiro Anzai
Faculty of Science and Technology, Keio Univ.

ヒントに、新しいデータ構造を自由に定義できる。

図２は、一般的なバックプロパゲーション用のニューロンの構造を示している。この構造の特徴は、結合荷重(weight)だけではなく、その微分値(dw)を学習則の要求から保持していることと、出力側から逆伝播してくる誤差信号を合計するための変数(sum)を持つことである。

```
typedef struct _BP_NEURON {
        unsigned                count;
        double                  out;
        struct _BP_NEURON       *next;
        struct _BP_NEURON       **link;
        double                  *weight;
        double                  *dw;
        double                  *sum;
} BP_NEURON;
```

図１．バックプロパゲーション用ニューロンの構造

b)関数の拡張

シミュレータ・ライブラリおよびグラフィック・ライブラリはソースコードの形で提供されるので、Ｃ言語の知識があるユーザは容易に機能を拡張できる。ただし現在SONNETにおいては、ある関数の動作を、その引数となるニューロンの型の違いにより変化させること（メソッド・サーチ）を実現しておらず、関数名の管理が煩雑化している。関数名管理については今後の課題であるが、速度を考えると、あまり強力な機構は設けにくい。

③シミュレータ用のグラフィック・ライブラリ

SONNETでは、グラフィック・ライブラリとして、ニューラルネットワークのシミュレーションにおいて、頻繁に使用される出力形式をツールとして用意している。

5．シミュレータの記述例

ここではSONNETの使用例として、簡単なバックプロパゲーション・アルゴリズムのシミュレータ記述を示す。

例として図２のような３層のネットワークにより、ｘｏｒ論理を学習することを考える。

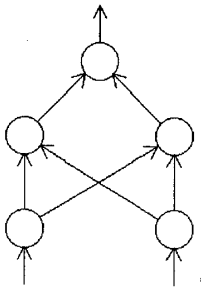

図２．ｘｏｒ

このとき、アルゴリズムの本体だけを記述してみると、図３のようなプログラムとなる。シミュレーションに必要な各種の処理は、SONNETがシミュレータ・ライブラリ関数として用意しているので、ユーザはそれらをつなぎ合わせる制御構造を、Ｃ言語の文法に従ってプログラムしていけばよい。

以下で、このプログラムを簡単に説明する。ここでは、各ニューロンにおけるバイアス値も学習の対象としている。そのために、全ニューロンは常に１を出力するダミーニューロン(bias)との間に仮想的な結合を持つ。この結合は、一般の結合と同様に学習則の適用を受け、これがバイアス値の学習にあたる。各結合の初期値は、-3～+3の間の一様乱数によって設定している。

動作としては、まずInput関数で入力層にパターンを入力させたのち、隠れ層の出力を求めるためにPropagate関数を呼んでいる。同様に出力層の値もPropagateで求める。

次に、教師データを出力層に提示し、出力層と隠れ層間のリンクを更新するためにBpOutputLearning関数を、隠れ層と入力層間のリンクの更新にはBpHiddenLearning関数を実行する。４つの入力パターンを学習させるごとに、自乗誤差を求め、学習が終了したかを判定している。

また、このプログラムでは現われていないが、SONNETのグラフィック・ライブラリを用いると、図４のようにシミュレーション結果を視覚的にとらえることができる。

```
#include <stdio.h>
#include <math.h>
#include "neuron.h"
#define  L      0.0
#define  H      1.0
double    pattern[4][2]={{L,L},{L,H},{H,L},{H,H}};
double    teacher[4][1]={{L},{H},{H},{L}};
BP_NEURON input[2], hidden[2], output[1], bias[1];

main()
{
   int    i;
   double dif, err;

   Generate( BP_NEURON, input, 2 );
   Generate( BP_NEURON, hidden, 2 );
   Generate( BP_NEURON, output, 1 );
   Generate( BP_NEURON, bias, 1 );
   InitialWeight( UNIFORM, -3.0, 3.0 );
   Connect( input, hidden, SECOND );
   Connect( hidden, output, SECOND );
   Connect( bias, hidden, SECOND );
   Connect( bias, output, SECOND );
   bias->out = 1.0;
   do{
      for( i = 0; i < 4; i++ ){
         Input( input, pattern[i] );
         Propagate( hidden );
         Propagate( output );
         BpOutputLearning( hidden, output,
                                   teacher[i]);
         BpHiddenLearning( input, hidden );
      }
      for( err = 0.0, i = 0; i < 4; i++ ){
         Input( input, pattern[i] );
         Propagate( hidden );
         Propagate( output );
         dif = teacher[i] - output->out;
         err += dif*dif;
      }
      printf( "%lf\n", err );
   }while( err > 0.01 );
}
```

図３．ｘｏｒを学習させるためのＢＰプログラム

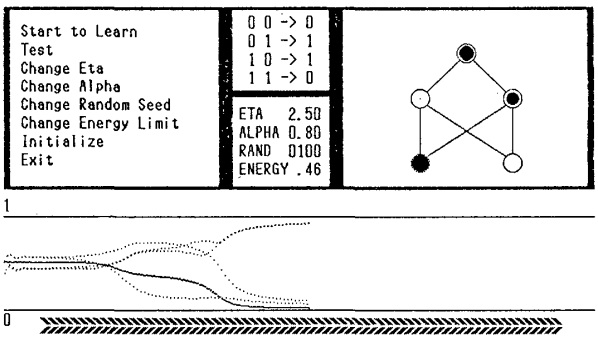

図４．グラフィック・ライブラリによる表示例

6．おわりに

ニューラルネットワーク・シミュレータの簡易作成用ツール SONNETを紹介した。本ツールを使用して現在までに、バックプロパゲーション、ボルツマンマシン、競合学習などのモデルのシミュレータを実現した。パソコン（PC-9801RA）上でも80387を用いると約0.2MFLOPSの計算能力があり、10万LPS(Link Per Second)の速度でシミュレーションを行えた。今後は、多くのマシン上にグラフィック部を移植することが課題である。

参考文献

[1] J. McClelland, D. Rumelhart and the PDP Research Group: “Parallel Distributed Processing” vol. 1, MIT Press, Cambridge, 1986

[2] N. Goddard: “The Rochester Connectionist Simulator: User Manual”, Univ. of Rochester, 1987