情報工学実験２
Neural Network 学習実験

グループ名： ぷりぷりプリン隊
メンバー： 035711B 大城堅治
035713J 小野雅俊
045743A 比嘉雅樹
035763E 饒平名隆一

実験日： 2005/01/17,24
提出日： 2004/02/04

1 実験の目的

Neural Network という人間の脳構造をモデル化したシステムについて学び，その学習アルゴリズムおよび学習の試行錯誤性について実際のシミュレーションを通して体験する。

2 課題

2.1 各種パラメータ（学習係数・慣性係数・乱数幅・乱数 seed・収束条件）を変更させて最も学習後の誤差が小さくなるパラメータの組み合わせを検討せよ。

2.1.1 検討したパラメータと値

表 1: 各種パラメータ

seed	10000,20000,....50000
学習係数	0.1〜2.0
乱数幅	0.01〜6.0
慣性係数	0.0, 1.0, 2.0

- 最も学習後の誤差が小さくなるパラメータ (平均)
  学習係数：1.29、乱数幅：3.01、慣性係数：0.0 のとき 5 回全て収束し、

  avgEpoch = 401.200000
  avgIteration = 12058.600000
  aveGError = 0.0008467 (5/5)
  Percent = 97.000000
  Square Error = 0.011895
  Absolute Error = 0.045282
  となる。（それぞれの実行結果はレポートの最後に付けてある。）

- 最も学習後の誤差が小さくなるパラメータ（単体）
  seed：50000、学習係数：1.29、乱数幅：3.01、慣性係数：0.0 のとき、結果は

  Epoch = 1242
  Iteration = 37286
  Error = 0.000290
  The whole number of recognition = 116 / 120 = 96.7
  Test Mean Square Error = 0.010319
  Test Absolute value of Error = 0.033256
  となる。

上記の２つのパラメータは、収束条件を変えずに行った結果である。
次に、収束条件を変化させた際のパタメータと値を示す。

- 最も学習時の誤差が小さくなるパラメータ（平均）
  収束条件:0.000008、学習係数：1.7、乱数幅：3.5141、
  慣性係数：0.0 のとき 5 回全て収束し

  avgEpoch = 4433.600000
  avgIteration = 133031.800000
  aveGError = 0.0000080 (5/5)
  Percent = 96.333333
  Square Error = 0.012302
  Absolute Error = 0.026200
  となる。

- 最も学習時の誤差が小さくなるパラメータ（単体）
  収束条件：0.000005、seed：10000、学習係数：1.7、乱数幅：3.514、
  慣性係数：0.0 のとき

  Epoch = 3997
  Iteration = 119919
  Error = 0.000005
  The whole number of recognition = 114 / 120 = 95.0
  Test Mean Square Error = 0.019232
  Test Absolute value of Error = 0.034239
  となる。

2.1.2 パラメータ、値を求めた方法

今回、我々の班は最適なパラメータの組み合わせを見つけるため neuralgtk.c を２タイプに改良した。一つ目は一度の実行で seed を 10000 から 50000 まで自動で増やし５つの平均した結果を出力するプログラムで、最後の調整はそれで行った。
もう一つは for 文で各種パラメータを増やしていく全自動プログラムで、学習係数、乱数幅をそれぞれ 0.05 刻みで増やしていき最適なパラメータの組み合わせを検討した。その後、さらに刻みを細かくして結果の良かったパラメータの周りを調べていくという方法で今回のパラメータの組み合わせを発見した。
なお、上記のプログラムを作成する過程で neuralgtk.c 以外の c ソースも多少変更したので以下にそれを示す。

- dynamics.c
  プログラムを実行した際に表示される「data .. NG」以下の出力を省いた。

- `data_gtk.c`、`spec_gtk.c`
  全自動プログラムを実行した際、この２つの c ソースの最後で fclose をしてないと途中エラーがおきて止まってしまう為 fclose を書き加えた。

2.2　2.1 で得られたパラメータによる学習曲線及び誤差を出力し、パラメータと収束能力の関連性について考察せよ。

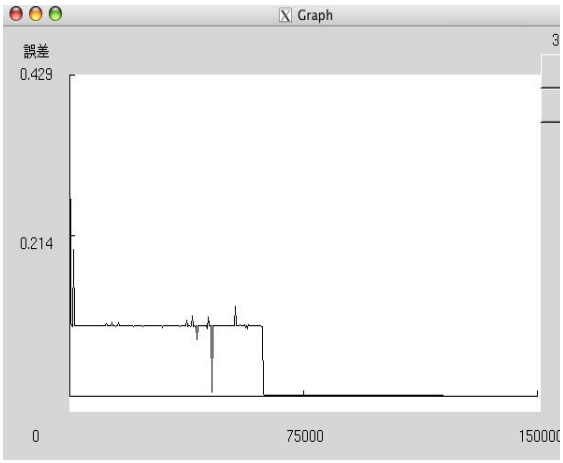

図 1: 求めた値 (0.000005) の学習曲線

- 上の図を見ると、学習を繰り返し試行していると波形は振動していて、平均的にエラーが少なくなる方向に推移して、最終的には誤差が 0 に収束している。これは、人間の脳と同様に何度も試行しているうちにその事柄についての結びつきが強くなっている（学習している）ことを示している。

- 少ない試行回数で学習できるということが収束能力の高さを示していといえる。

- 学習がなかなか終わらい場合などあるが、あれは何度も同じことを学習するが似たような値が繰り返されるためであり、人間が混乱してしまった状況に似ていると考えられる。

- 学習係数の値が少ない場合は振動はあまり起こらず、緩やかなカーブを描いて安定した学習を行う。これは学習係数が高くなるに従って、学習により得られた結果コンピュータが行う自身への修正の度合を高めるためである。

- 学習係数の値が大きくなればなるほど収束しづらい。

- 慣性係数は 0.0 でないと認識能力は上がらない。
  慣性係数とは、前回に取った行動がこれからの行動に及ぼす変化の量を決定する係数であると考える。従って、この慣性係数の値が学習係数に比べて等しいかそれ以上である場合には、コンピュータは新たに学習した事よりも今までの間違った方向に重点を置き、行動してしまう可能性が高いと考える。

2.3 2.1 と同様に各種パラメータを変更させて、未学習データに対する認識能力が最も高くなるパラメータの組み合わせを検討せよ。

- 認識能力が最も高くなる組み合わせ（平均）
  学習係数：1.95、乱数幅：1.75、慣性係数：0.0 のとき 5 回全て収束し

  Percent = 97.500000
  Square Error = 0.011278
  となる。

- 認識能力が最も高くなる組み合わせ（単体）
  seed：50000、学習係数：2.0、乱数幅：3.4、慣性係数：0.0 のとき

  The whole number of recognition = 118 / 120 = 98.3
  Test Mean Square Error = 0.009817
  となる。実際認識率が 98.3 になる組み合わせは何百個もあったが、その中で一番平均自乗誤差が小さいものを上では使用した。

2.4 パラメータの値と収束能力・未学習データに対する認識能力について考察せよ。

まず、慣性係数を変動させて起きた変化を考えてみる。慣性係数は値を少しいじっただけで認識能力や収束能力、誤差に影響が出た。これより、ニューラルネットは過去の計算結果を反映できることと、そして neuralgtk のソースコードから慣性係数は「過去の計算結果をどこまで反映させるか」を決めるものであることが分かる。そのためこれをセットすると過去に演算した「間違った結果」を多く取り入れてしまうと思われ、係数の値を 0.0 以外にした時には収束しないと考えられる。

次に、学習係数と乱数幅の値の組み合わせでの収束能力について考える。下の表は収束能力の変化について調べた表である。

表 2: 学習係数と乱数幅の値による収束能力の変化 (0.10〜1.00)

学習係数	収束回数 (4/5) 以下になる時の乱数幅	収束回数 (3/5) 以下になる時の乱数幅
0.10	3.75〜	4.85〜
0.20	3.70〜	5.25〜
0.30	3.65〜	5.25〜
0.40	3.65〜	5.25〜
0.50	3.60〜	5.25〜
0.60	3.55〜	5.25〜
0.70	3.65〜	5.25〜
0.80	3.35〜	5.25〜
0.90	3.80〜	5.25〜
1.00	3.85〜	5.20〜

表 3: 学習係数と乱数幅の値による収束能力の変化 (1.10〜2.00)

学習係数	収束回数 (4/5) 以下になる時の乱数幅	収束回数 (3/5) 以下になる時の乱数幅
1.10	3.65〜	5.20〜
1.20	3.80〜	5.00〜
1.30	3.35〜	4.75〜
1.40	3.45〜	4.70〜
1.50	3.40〜	4.45〜
1.60	3.60〜	4.75〜
1.70	3.35〜	4.45〜
1.80		3.25〜
1.90	2.75〜	3.85〜
2.00		3.00〜

この表 2 と 3 を見てみると、学習係数が小さい値から大きい値に向かうにつれて、収束しなくなる量が増えている事が読み取れる。学習係数が高いと学習が進むが認識能力と収束能力が悪くなることが分かった。これはおそらく、間違った答えをもとに学習してしまい、そのまま間違った答えを導き出すためと思われる。

次に、「neuralgtk」でこれらがどのように計算に使われているかを見てみる。慣性係数、学習係数はユニット間の「重み」を計算する「学習」にて使用。重みの計算は次のようになっている。

- 重みの微小変化=(学習係数＊出力) ＋ (慣性係数＊この前の重みの微小変化)
- 重み+=重みの微小変化

つまり学習係数は出力をどこまで考慮するか、慣性係数は前回の計算をどこまで考慮するかというものということになる。はじめのうちは計算が間違っているのは当たり前だが慣性係数を考慮すると更に計算が狂うということになる。また、乱数幅はこの構造ネットの初期化の際に重みの初期化、及びバイアス (傾斜) の初期化に使われる。が、重みもバイアスもこのあとどんどん更新されていくのと、もともとこの二つの値が完全ランダム (厳密には違う) になってしまうため計算結果に影響を与えるとは考えにくい。その時の計算は下記のようになる。

- 重みの初期値＝（ランダム＊ 2.0 ー 1.0）＊乱数幅

バイアスも同じような計算式。だが重みやバイアスが巨大になろうと微小になろうとその時はその時なりの計算が取られるためそこまでの影響はなしと思われる。未学習データの誤差判断能力は学習の回数が多くなるほど高くなる。このことは、人間の生活における経験にあたる。

2.5　2.1 で得られた最適なパラメータ値を使って、忘却付き構造学習を行い、得られる学習終了時のネット構造について考察せよ。

2.1 で得られた最適な実行データを使い、忘却付き構造学習を行った。忘却係数は 0.000100 とする。
seed 40000
学習係数：1.7
慣性係数：0.0
乱数幅：3.5141
左側の４つの○を第１列、真ん中の○を第２列、左の○を第３列とすると、以上の

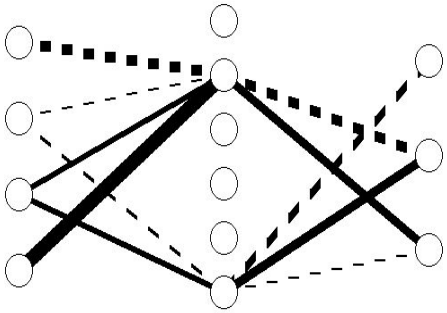

図 2: 得られたネット構造

実行結果では、第２列の上から２番目と６番目以外は使用されないらしく、第２列目で中継されているのは、上から２番目と６番目のみであり、第１列の上から１番目と４番目は第２列目の上から２番目にのみ出力され、第１列の上から２番目、３番目は第２列の２番目及び６番目に出力されている。
以上のことにより、下の表のような依存関係が成り立っている。

表 4: 第１列目と第２列目の関係

第１列目	第２列目
○ 1	○ 2
○ 2	○ 2, ○ 6
○ 3	○ 2, ○ 6
○ 4	○ 2

表 5: 第２列目と第３列目の関係

第１列目	第２列目
○ 2	○ 2, ○ 3
○ 6	○ 1, ○ 2

2.6 忘却係数を $1.0 \times 10^{-4}, 1.0 \times 10^{-5}, 5.0 \times 10^{-6}, 1.0 \times 10^{-6}$ と変えた場合の学習終了時のネット構造について調べ、忘却係数がネット構造に与える影響について考察せよ。

今回はデータは慣性係数 0.0、学習係数 1.70、乱数幅 3.5141、収束条件 0.000008 で 10000 から 50000、10000 刻みの seed、忘却係数は指定された４つに 0.000000 の五つを実行してみた。図は全て seed が 10000 のものである。また、入力ユニットは９個、中間ユニットは６個、出力ユニットは３個とする。

- 忘却係数 0.000000

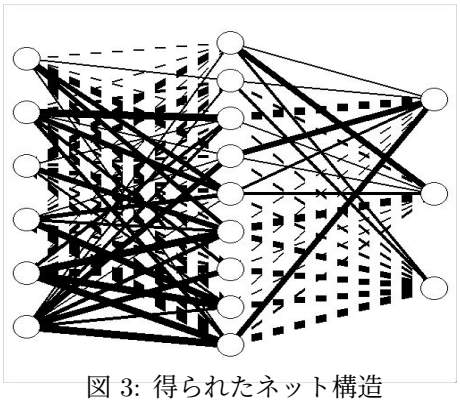

図 3: 得られたネット構造

全ての seed で共通していえることだがあるユニットは別の階層で隣り合う階層のユニットと『確実に』接続されている。これではどの入力がどこの中間をとおり、出力に関わるかということが分からない。

- 忘却係数 0.000001

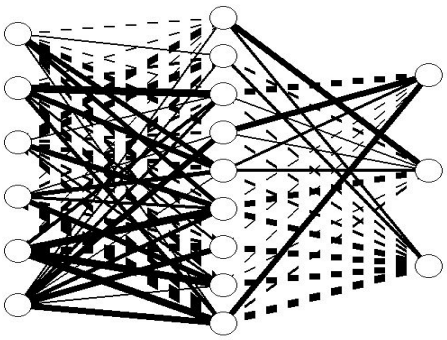

図 4: 得られたネット構造

忘却係数をとりあえず設定。すると一部の接続 (図の点線) がなくなった。これは全ての seed で同一の現象が見られる。ただ、seed によりその程度が変わる。接続が一つしか消えないものから四、五本以上消えるものもある。

- 忘却係数 0.000005

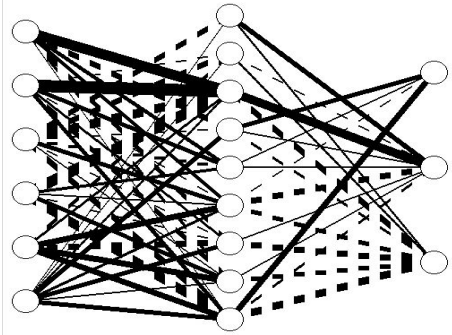

図 5: 得られたネット構造

接続切れが増え、多少なりとも最適化されてきている。seed による差もまだある。

- 忘却係数 0.000010

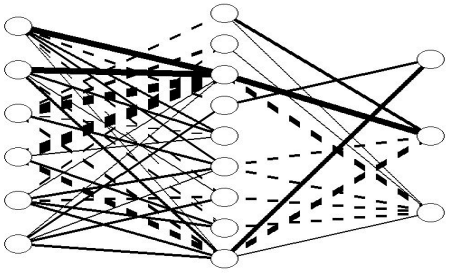

図 6: 得られたネット構造

ここまで来ると接続切れもかなりの数になり、次の階層にデータを渡さないユニットも現れた。一部、先程まで接続されていなかったユニットが接続されていたりするが誤差範囲内ということだろう。

- 忘却係数 0.000100

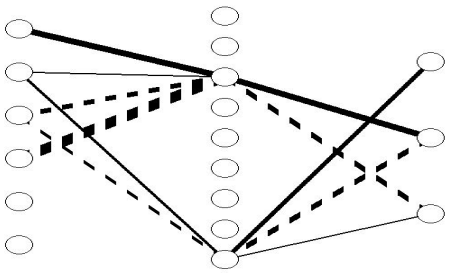

図 7: 得られたネット構造

指定された中で最大の忘却係数。この時点で接続の関係がほぼ固まり、無駄な接続が一掃される。つまり、骨格構造が現れる。

その後、忘却係数 100.000000 でも試してみたがネット構造にあまり変化が見られなかった為画像は省く。

授業資料と今回の実験の結果から考えられるのは「忘却係数」は「ネット構造」構築の際のフィルタの役目ということ。授業資料に書かれている通り「忘却学習」は結合加重の総和を抑える学習。ソースを見ても完全に理解できた訳ではないがこの忘却係数は重みを計算する際に使われる。使われ方はものすごく簡単で重みの計算の際にこの値を加えるか引くかするだけ。これを繰り返すことにより、重み全体を整えることが出来る。よって小さすぎる場合はあまり調整できない、大きすぎると逆に減らし過ぎたり増やし過ぎたりするので良くない、ということになる。忘却係数を有効にした学習の関数はソースの bp.c の中の WeightUpdateSLF() という関数である。この関数で忘却係数が使われている箇所はあまり多くなく、重さやバイアスの変化を緩やかにする目的で使われているだけである。が、実際には計算は相当数行われているのでその差はかなりのものになることが考えられる。この重さを出来る限り増やさない計算方法により、忘却係数によってはうまい具合に骨格構造 (学習会を導き出す接続のみの入出力モデル) が得られる。しかし忘却係数が小さすぎるとあまり重みは削られず骨格構造に至らない、大きすぎると重みを削り過ぎて振動させる結果となる。あまりにも大きすぎると骨格構造に至らないが誤差は安定することもある。

3 最後に

今回改良したプログラムは

`naha:/net/home/y03/j03043/neural/`

以下に置いておく。置いてあるファイルの説明として、

- `neuralgtk_seed.c`
  「neuralgtk.c」から、グラフや図を無くし、一回実行ボタンを押すと 5 つの seed 全てを実行しその平均を出力する。「参照」にてパラメータの値を設定して「保存」を押した時に出る確認のメッセージも無くした。

- `neuralgtk_kansei.c`
  上のプログラムをさらに改良し、パラメータの値を for 文で増やしていく全自動プログラム。このプログラムでは Epoch,Error, 認識率,Square Error,Absolute 〜Error の単体、平均での良い結果をファイルに保存する。

- `tigai_seed.txt`
  「`diff -c neuralgtk.c neuralgtk_seed.c > tigai_seed.txt`」を実行して作られたファイル。

- `tigai_kansei.c`
  「`diff -c neuralgtk.c neuralgtk_kansei.c > tigai_kansei.txt`」を実行して作られたファイル。