情報工学実験II レポート（探索アルゴリズム2）

月曜班 グループ7

実験実施日 2014年12月22日

グループメンバ

- 135711F:屋比久祐樹:Level1
- 135713B 天願寛之: 担当 Level3.3
- 135717E 岡田和也: 担当 Level3.1, 3.2
- 135761B 大城海斗: 担当 Level2, 3.4

提出したレポート一式について

レポート一式は‘‘shell:~tnal/2014-search2-mon/group7/’’ にアップロードした。提出したファイルのディレクトリ構成は以下の通りである。

```
./             # レポート関係ファイル
./figs         # 図ファイル, 作成したスクリプト.
./lebel3-3figs # level3-3 で使用する図ファイル等
./level2figs   # level2 で使用する図ファイル等
./levelX       # level3.1 で作成したスクリプトファイルと編集したプログラム.
```

1　Level1: 線形分離可能な OR 問題への適用

1.1　課題説明

2 入力 1 出力で構成される単純パーセプトロン（ニューラルネットワーク）を用いて、4 つの教師信号を用意した OR 問題へ適用し、重みが適切に学習可能であることを確認する。また、学習が収束する様子をグラフとして示す。

1.2　OR 問題を学習させた際の誤差収束度合いについて

1.2.1　実験結果

シード値を変える事により、収束回数が変化し 10 パターンの誤差などが図 1 で表せることで実行結果が出た。
さらに、10 パターンの結果を平均を図 2 として結果が出た。

表 1: OR 問題の学習に要した回数

シード値	収束した回数
1000	96
2000	90
3000	111
4000	109
5000	93
6000	99
7000	100
8000	114
9000	113
10000	94
10 試行の平均値	101.9

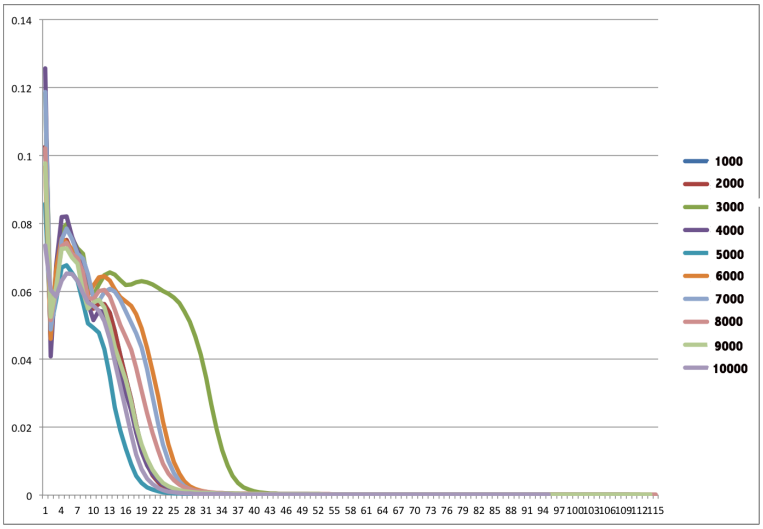

図 1: 重みを更新する様子

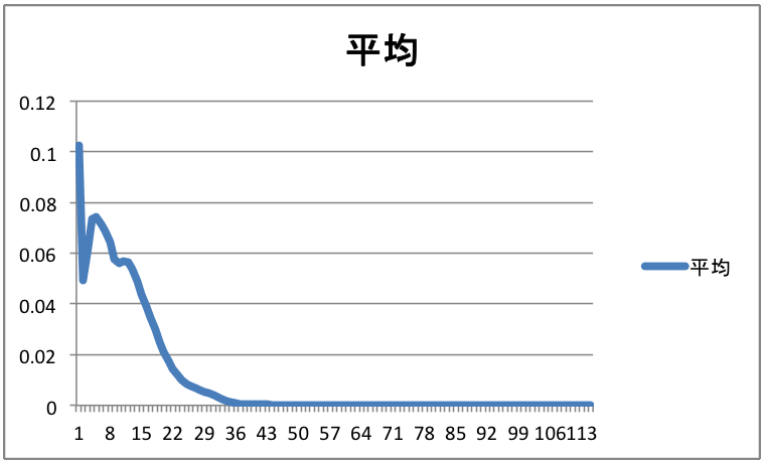

図 2: 重みを更新する様子（平均値）

1.2.2 考察

上記の二つの図を見て x 軸は収束回数を表し、y 軸は誤差を表している。図 1 は 10 パターンをまとめ収束回数と誤差を表している。図 2 は 10 パターンのグラフ平均をまとめた図である。

2 Level2: 線形分離不可能な ExOR 問題への適用

2.1 課題説明

階層型ニューラルネットワークを ExOR 問題へ適用し、線形分離できない問題においても学習可能であることを確認する。特に Level2 では、この問題を解決するために中間層を導入することで拡張した階層型ニューラルネットワークにより学習可能であることを確認する。

2.2 階層型 NN による学習

2.2.1 最適なパラメータを探すためのアプローチ

ETA, *ALPHA*, *HIDDEN* を指定された範囲内でランダムな値に設定し，プログラムを実行した．パラメータの値の決定や，iteration 数の平均値を求める処理は次のシェルスクリプト (ソースコード 1) で行った．ExOR 問題は線形分離不可であるため，HIDDEN の最小値を 2 としている．最大値は特に指定されていないが，HIDDEN を極端に大きくしすぎると各パラメータの最適な組み合わせを見つけ出すのが困難になると考えたため，最大値を 16 とした．

Listing 1: 本 level で使用したシェルスクリプト

```
#!/bin/sh
#シェルスクリプトの要件
# 1.ETA,ALPHA,の値として範囲内でランダムな組み合わせを選択肢HIDDEN
# それを元に2.値を変えてパターンの結果を得る．seed10
# 3.のの回数を抽出し，そのときのパラメータの値を一緒に表示FINISHiteration
# パターンの4.10の平均をとる．iteration
#
#bp_mo_exor.の変更c
# 文字列処理を行うためにのを以下のように変更した．fprintfstderr
# fprintf(stdout,"FINISH 2: iteration = %4d, error = %.10f\n",ite,err);
#
export LANG=C

((eta_inte=${RANDOM}%2))
#eta_deci=`echo "scale=2; ($RANDOM*3) / 100000" | bc`
((eta_deci=${RANDOM}%100))
ETA=`expr ${eta_inte}.${eta_deci}`
sed -i -e "s:#define ETA.*:#define ETA $ETA:g" bp_mo_exor.c

#alpha_deci=`echo "scale=2; ($RANDOM*3) / 100000" | bc`
((alpha_deci=${RANDOM}%100))
ALPHA=`expr 0.$alpha_deci`
sed -i -e "s:#define ALPHA.*:#define ALPHA $ALPHA:g" bp_mo_exor.c

((HIDDEN=${RANDOM}%15+2))
sed -i -e "s:#define HIDDEN.*:#define HIDDEN $HIDDEN:g" bp_mo_exor.c

sed -i -e "s/
//g" bp_mo_exor.c

gcc bp_mo_exor.c -o bp_mo_exor
echo "ETA:$ETA  ALPHA:$ALPHA    HIDDEN:$HIDDEN"
./bp_mo_exor 1000 > fuga.txt 2> log.txt
./bp_mo_exor 2000 >> fuga.txt 2>> log.txt
./bp_mo_exor 3000 >> fuga.txt 2>> log.txt
./bp_mo_exor 4000 >> fuga.txt 2>> log.txt
./bp_mo_exor 5000 >> fuga.txt 2>> log.txt
./bp_mo_exor 6000 >> fuga.txt 2>> log.txt
./bp_mo_exor 7000 >> fuga.txt 2>> log.txt
./bp_mo_exor 8000 >> fuga.txt 2>> log.txt
./bp_mo_exor 9000 >> fuga.txt 2>> log.txt
./bp_mo_exor 10000 >> fuga.txt 2>> log.txt

sum=0
cat log.txt | awk '{print $5}' | tr -d "," > figures.txt
exec < figures.txt
while read line
do
    ((sum=${sum}+${line}))
done

((avarage=${sum}/10))

cat log.txt
echo "the avarage is ${avarage}."
```

このシェルスクリプトを数十回実行し，iteration 数の平均値が比較的小さい実行結果を 10 個分記録した．その中で平均値が最小なものを選択し，グラフ化することにした．

2.2.2 実行結果

表 2 にシード値 10 パターンで試した際の収束に要した学習回数と，その最小の平均回数を示す．

シード値	収束した回数
1000	37
2000	45
3000	52
4000	44
5000	50
6000	53
7000	39
8000	29
9000	38
10000	39
10 試行の平均値	42

表 2: 階層型 NN による ExOR 問題の学習に要した回数

各パラメータが $ETA = 1.26, ALPHA = 0.94, HIDDEN = 16$ の時，表 2 のような結果が得られた．その時の学習曲線は図 3 のようになる．図 3 は gnuplot を用いて seed 値別にプロットしたグラフを元に smooth unique オプションで平均化を行い，得られたものである．

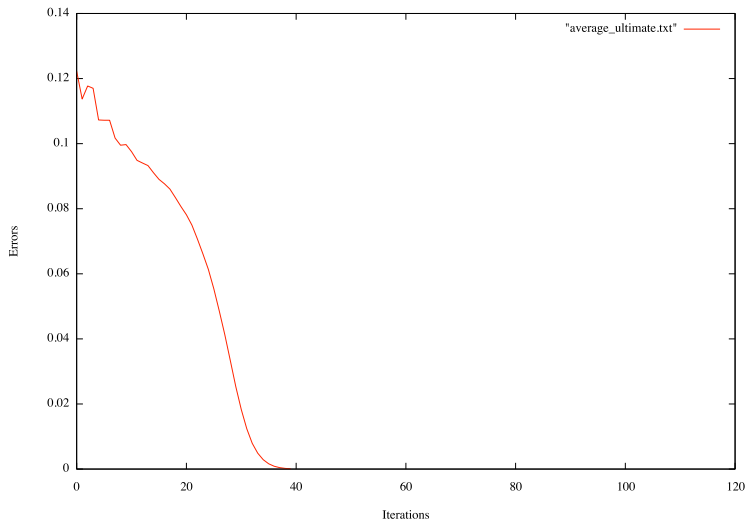

図 3: 重みを更新する様子（平均値）

2.2.3 考察

ここでは，各パラメータと iteration 数の関係について考察する．
下記に平均 iteration 数が比較的小さい実行結果とそのときのパラメータの値を示す．

平均 iteration 数が小さい実行結果

```
/Users/e135761/search/info2/nn/bp_mo% cat suitable.txt | grep -e 'ETA' -e 'the'
ETA:1.89  ALPHA:0.82    HIDDEN:14
the avarage is 102.
ETA:1.79  ALPHA:0.80    HIDDEN:10
the avarage is 120.
ETA:1.20  ALPHA:0.90    HIDDEN:5
the avarage is 115.
ETA:1.22  ALPHA:0.86    HIDDEN:14
the avarage is 114.
ETA:0.85  ALPHA:0.94    HIDDEN:15
the avarage is 55
ETA:1.80  ALPHA:0.83    HIDDEN:10
the avarage is 102.
ETA:1.74  ALPHA:0.77    HIDDEN:14
the avarage is 137.
ETA:1.26  ALPHA:0.94    HIDDEN:16
the avarage is 42.
ETA:1.60  ALPHA:0.93    HIDDEN:10
the avarage is 46.
ETA:0.84  ALPHA:0.93    HIDDEN:16
the avarage is 70.
ETA:0.69  ALPHA:0.94    HIDDEN:13
the avarage is 78.
```

10 回の実行結果における共通点は，いずれも ALPHA 値が 1 に近い値をとるという点だった.

続いて，平均 iteration 数が大きい実行結果を示す.

平均 iteration 数が大きい実行結果

```
/Users/e135761/search/info2/nn/bp_mo% cat unsuitable.txt | grep -e 'ETA' -e 'the'
ETA:0.14  ALPHA:0.29    HIDDEN:9
the avarage is 5344.
ETA:0.38  ALPHA:0.25    HIDDEN:2
the avarage is 12870.
ETA:0.37  ALPHA:0.94    HIDDEN:2
the avarage is 20202.
ETA:0.0  ALPHA:0.78    HIDDEN:9
the avarage is 100000.
ETA:0.81  ALPHA:0.49    HIDDEN:2
the avarage is 10938.
ETA:0.16  ALPHA:0.22    HIDDEN:4
the avarage is 6317.
ETA:0.57  ALPHA:0.25    HIDDEN:2
the avarage is 11891.
ETA:1.70  ALPHA:0.27    HIDDEN:2
the avarage is 33521.
ETA:1.49  ALPHA:0.44    HIDDEN:2
the avarage is 30505.
ETA:1.68  ALPHA:0.72    HIDDEN:2
the avarage is 40170.
```

平均 iteration 数が小さいときのパラメータと平均 iteration 数が大きいときのそれとを比較すると，HIDDEN の値が小さいと平均 iteration 数が大きくなり，反対に値が大きいと平均試行回数が小さくなるという結果になった．また *ETA* に関しては，0 を設定しない限り iteration 数に大きな影響を与えないことが伺える．これら実行結果より，最も効率良く学習が収束するパラメータの組み合わせは，表 3 のようになると仮説を立てた.

ETA	*ALPHA*	*HIDDEN*
0 以外の数値	1 に近い値	できるだけ大きい数値

表 3: 最適と思われるパラメータの組み合わせ

しかし，$ETA = 1.26, ALPHA = 0.94$ のままで $HIDDEN$ 値を 100 にすると，平均試行回数は大きくなり 12711 回となった．単純に $HIDDEN$ 値を大きくすればよいわけではないことが分かる．次に，$ETA = 1.26, HIDDEN = 16$ のままで $ALPHA$ を 0.99 に設定して実行したところ，これも平均試行回数が 32999 となり，1 に近い値を設定すればよいという仮説は棄却された．他の数値も試したが，$ALPHA = 0.94$ がベストな値であった．続いて，$ALPHA = 0.94, HIDDEN = 16$ のままで ETA の値を変更してみたところ，$0.8 \leq ETA \leq 1.6$ の範囲では平均試行回数は 51 や 45 などどれも少なかった．その中で，平均試行回数がもっとも少なかったのは 40 回で，$ETA = 1.5$ のときであった．このことから最適なパラメータの組み合わせは表 4 のようになる．

ETA	$ALPHA$	$HIDDEN$	平均試行回数
1.50	0.94	16	40

表 4: 最適なパラメータの組み合わせ

3 Level3: 応用事例：文字認識問題への適用

3.1 課題説明

階層型 NN を文字認識に適用し、考察する。特に、用意された教師データと認識のしやすさに関する関係性や、学習最適化のためのパラメータのチューニングおよび、より柔軟性の高い認識方法に関する検討を行う。

3.2 Level3.1: パラメータのチューニング

3.2.1 最適なパラメータを探すためのアプローチ

指定された条件下において学習が効率良く行われるパラメータの組み合わせを探すため、hidden を 10 から 100 まで 10 づつ増やしていき,eta を 0.01 から 1.98 まで 0.01 づつ増やし,alpha を 0.01 から 0.99 まで 0.01 づつ増やしていきその中から一番値が小さい物を探した. その動作をスクリプトを書いて実行させた. その後その値の近辺をスクリプトで探した.

3.2.2 実行結果

各パラメータが eta = 1.95, ALPHA = 0.62, HIDDEN 30 の時, 表 5 のような結果が得られた. このときの学習曲線の平均はスクリプトを書いて出した. ソースは./figs/の中に置いておく.

表 5: 階層型 NN による文字認識問題の学習に要した回数

シード値	収束した回数
1000	105
2000	75
3000	62
4000	126
5000	74
6000	87
7000	85
8000	78
9000	122
10000	61
10 試行の平均値	87.5

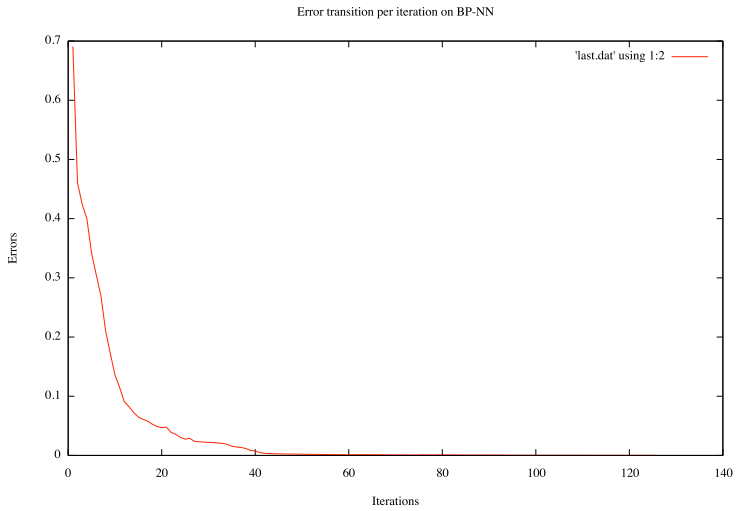

図 4: 重みを更新する様子（平均値）

3.3 Level3.2: パラメータと収束能力の関連性について

3.3.1 関係性を確認するためのアプローチ

3 つのパラメータがどのような関係にあるかを検証するため、ケース 1,2,,, を設定し、学習曲線からその関係性について考察する。

3.3.2 結果

```
--------------------------------
10000,hidden=10,alpha=0.2,eta=0.1
10000,hidden=10,alpha=0.1,eta=0.1
9996.7,hidden=10,alpha=0.6,eta=0.1
9919.8,hidden=10,alpha=0.3,eta=0.1
9879,hidden=10,alpha=0.5,eta=0.1
9856.8,hidden=10,alpha=0.4,eta=0.1
9671.3,hidden=10,alpha=0.1,eta=0.2
9670.8,hidden=90,alpha=0.8,eta=1.7
9660.6,hidden=10,alpha=0.2,eta=0.2
9424.7,hidden=10,alpha=0.3,eta=0.2
-----------------------------------
109,hidden=40,alpha=0.4,eta=1.4
108.4,hidden=30,alpha=0.5,eta=1.5
108.1,hidden=40,alpha=0.5,eta=1.4
106.9,hidden=30,alpha=0.5,eta=1.8
106.2,hidden=30,alpha=0.6,eta=1.8
105.6,hidden=30,alpha=0.6,eta=1.5
105.6,hidden=30,alpha=0.5,eta=1.7
105.3,hidden=30,alpha=0.6,eta=1.6
104.9,hidden=30,alpha=0.5,eta=1.6
101.3,hidden=30,alpha=0.6,eta=1.7
```

上記の実行結果はスクリプトでの出力を sort コマンドでソートし, 結果が悪いものと良いものを抜き出してきたものである. 左から seed 値 1000— —10000 までの収束した回数の平均値,hidden の値,alpha の値,eta の値となっている.

3.3.3 考察

悪いところの共通点は hidden の値が 10 か 90 のどちらかになっている. 結果が良かった方を見ると hidden は 30 か 40 を取っているので hidden の値は極端に大きい物や小さい物は良くない. また,alpha 値は結果が良い方を見ると,0.5,0.6,0.4 に分けられる. この事から alpha の最適な値は 0.5±1 だという事が分かる.eta の値は大きい方が良いが結果の悪い方の, 下から三行目の結果から hidden 値と eta が両方とも大きいと結果が悪くなる事が分かる.

3.4 Level3.3: 任意の評価用データを用いた評価

3.4.1 アプローチ

学習時のデータ (教師データ) との違いが少ないほど認識率が高く、逆に教師データとの違いが多いほど認識率が低くなるとの仮定のもと、以下に示す評価用データを用意した。
今回は違いを分かり易くするため、0 と 1 で表現された教師用データから、1 の位置をズラす範囲とその数によって違いを作っている。つまり、より多くの 1 がより大きくズレていれば、より違いが多いということである。

```
1
000000000000
000011100000
000001000000
000001000000
000010000000
000010000000
000001000000
000001000000
000001000000
000001000000
000011100000
000000000000
```

(1) 違いが 1 番少ないデータ

```
1
000000000000
000011100000
000001000000
000010000000
000100000000
000100000000
000100000000
000010000000
000010000000
000001000000
000011100000
000000000000
```

(2) 違いが 2 番目に少ないデータ

```
1
000000000000
000111000000
000000100000
000010000000
001000000000
100000000000
001000000000
000010000000
000000100000
000001000000
000001110000
000000000000
```

(3) 違いが最も多いデータ

図 5: 任意の評価用データ

3.4.2 結果

最初に「1」という文字を学習させた。学習したことによる学習度合いを図 6 に示す。

用意した任意の評価用データに対する適応度合いを確認する。

3.4.3 考察

上記の結果で注目すべきは EVAo[1] という項目であり、この値が 0.9 に近ければ近いほど、より「1」と認識された事を表している。違いが多いほどこの値が小さくなり、「1」と認識されにくいことが分かる。
したがって、仮説通り教師データとの違いが少ないほど認識率が高く、逆に教師データとの違いが多いほど認識率が低いといえる。

```
err=0.000090, MIN_ERR=0.000100
CHECK ctg[0] :
CHECK o[0] = 0.09983, t[0] = 0.1
CHECK o[1] = 0.89983, t[1] = 0.9
CHECK o[2] = 0.10014, t[2] = 0.1
CHECK o[3] = 0.09841, t[3] = 0.1
CHECK o[4] = 0.09992, t[4] = 0.1
CHECK o[5] = 0.09986, t[5] = 0.1
CHECK o[6] = 0.09994, t[6] = 0.1
CHECK o[7] = 0.10017, t[7] = 0.1
CHECK o[8] = 0.10029, t[8] = 0.1
CHECK o[9] = 0.09991, t[9] = 0.1
CHECK sum_error = 0.00292
```

図 6: 学習事例に対する学習度合いを確認

```
EVA o[0] = 0.12511, correct[0] = 0.1
EVA o[1] = 0.81676, correct[1] = 0.9
EVA o[2] = 0.26431, correct[2] = 0.1
EVA o[3] = 0.18871, correct[3] = 0.1
EVA o[4] = 0.07906, correct[4] = 0.1
EVA o[5] = 0.08593, correct[5] = 0.1
EVA o[6] = 0.10824, correct[6] = 0.1
EVA o[7] = 0.08488, correct[7] = 0.1
EVA o[8] = 0.15333, correct[8] = 0.1
EVA o[9] = 0.09219, correct[9] = 0.1
EVA sum_error = 0.48087
```

(1) 違いが 1 番少ないデータ

```
EVA o[0] = 0.12202, correct[0] = 0.1
EVA o[1] = 0.45512, correct[1] = 0.9
EVA o[2] = 0.47307, correct[2] = 0.1
EVA o[3] = 0.22733, correct[3] = 0.1
EVA o[4] = 0.16363, correct[4] = 0.1
EVA o[5] = 0.39274, correct[5] = 0.1
EVA o[6] = 0.09856, correct[6] = 0.1
EVA o[7] = 0.06884, correct[7] = 0.1
EVA o[8] = 0.25194, correct[8] = 0.1
EVA o[9] = 0.12242, correct[9] = 0.1
EVA sum_error = 1.53062
```

(2) 違いが 2 番目に少ないデータ

```
EVA o[0] = 0.32506, correct[0] = 0.1
EVA o[1] = 0.44897, correct[1] = 0.9
EVA o[2] = 0.13983, correct[2] = 0.1
EVA o[3] = 0.15143, correct[3] = 0.1
EVA o[4] = 0.24578, correct[4] = 0.1
EVA o[5] = 0.08240, correct[5] = 0.1
EVA o[6] = 0.09852, correct[6] = 0.1
EVA o[7] = 0.02110, correct[7] = 0.1
EVA o[8] = 0.23263, correct[8] = 0.1
EVA o[9] = 0.23425, correct[9] = 0.1
EVA sum_error = 1.27799
```

(3) 違いが最も多いデータ

図 7: 任意の評価用データ

ところで、違いが 2 番目に少ないデータと 1 番多いデータの認識率はあまり変わらない。教師データに用いられた「1」を表現する 0 と 1 の羅列を「1」という形を保ったまま、重なるところがないようにズラしたところ、認識率は 0.03 以下になった。一方で、「1」の上端下端のみにしたところ認識率は 0.6 以上になった。もちろん認識率 0.6 では「1」と呼べないが、「1」と認識できるデータよりも認識率が大きいため、教師データに対してより重なる部分が多ければ、認識され易いと考えることが出来る。

3.5 Level3.4: 認識率を高める工夫

3.5.1 対象とする問題点

以下に対象とする問題点を列挙する.

- 与えられた数字のサイズが相対的に小さい
- 与えられた数字のサイズが相対的に大きい
- 与えられた数字の位置がずれている
- 与えられた数字の周りにノイズが入っている

3.5.2 改善方法の提案

1. 数字のサイズが相対的に大きいもしくは小さいなどの場合は，与えられた数字に対して拡大・縮小を行い，学習用の数字に近づけた上で認識させる方法がある.
2. 数字の位置がずれている場合も同様に，与えられた数字を中央に移動させたものを認識させる方法が考えられる.
3. 数字の周りにノイズが入っている場合は，学習用の数字と与えられた数字とを重ね合わせて AND をとったものを認識させる方法がある.

3.5.3 考察

与えられた数字に対して拡大・縮小を行う方法の問題点として，どの程度の拡大および縮小を行えばよいのかが不明な点である．少しずつ拡大・縮小の度合いを大きくしていき，それを認識させることを繰り返すという方法が考えられるが，これはやや非効率なやり方である．

数字を中央に移動させる方法は，数字の縦横の比率から中央となる場所を割り出し，数字を移動させるため，比較的単純な処理となる．

AND をとる方法では，AND をとることで与えられた数字と学習用の数字の共通部分のみを抽出できるため，認識率を格段に向上させることができる．しかし，与えた数字と学習用の数字のどちらかが相対的に大きかったり，互いの位置がずれていたりすると，この手法はあまり効果を発揮しない．そのため，改善方法の提案で述べた 1 と 2 の方法と組み合わせることで，認識率を高めることができると考える．

4 levelX オプション課題

4.1 Level 3.1, 3.2 における複数シミュレーションを効率良く行い、集計するスクリプト作成やプログラム修正等。

num.c を修正し実行すると,iteration の値だけが出力するようにした. 以下のスクリプトは./levelX/src/に置いてある

- *fuga.perl*
  パラメータの値と seed 値を変更しながら nn_num を実行していく. 結果は標準出力として出る.
- *hoge*2.*perl*
  fuga.perl で出た出力をファイルに保存しておきその中から一番収束するのが速い結果の行を出力する (sort コマンドのおかげで使用はしていない)
- *average.perl*
  学習曲線の平均値を出すプログラム. このプログラムを使用する際には num.c を修正して err の値だけが出力されるようにし, その出力を seed 値毎にファイルを分けて保存すると使用出来る. その際ファイルの名前は'rep[seed 値].txt'にする.

5 その他: 実験の内容・進め方に関するコメント等

参考文献

[1] 情報工学実験 2: 探索アルゴリズムその 2（當間）
http://www.eva.ie.u-ryukyu.ac.jp/~tnal/2011/info2/search2/ perlplus
http://www.perlplus.jp/