情報工学実験II

-探索アルゴリズム 2-

グループ：A

メンバー：j05002　池野谷克俊 (Level 0,1,2,3 担当)

j05019　鴻池宏輝 (Level 0,1,4 担当)

j05066　與久田龍一 (Level 0,4 担当)

提出日:2006 年 11 月 29 日 水曜日

1 Level 0

使用した pw およびアカウントは以下の通り。

池野谷克俊:pw002,j05002
鴻池宏輝 :pw019,j05019
與久田龍一:pw026,j05066

2 Level 1

2.1 Level 1.1

乱数シード値を変更して実行した結果を以下に示す。

- 乱数シード値 10000

```
省略
iteration =   91, error = 0.00010
iteration =   92, error = 0.00010
iteration =   93, error = 0.00010
iteration =   94, error = 0.00010
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10471, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.89009, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.89009, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.92378, t{0] = 0.9
iteration =   94, error = 0.00010
```

- 乱数シード値 20000

```
省略
iteration =   99, error = 0.00010
iteration =  100, error = 0.00010
iteration =  101, error = 0.00010
iteration =  102, error = 0.00010
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10475, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.88999, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.89015, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.92388, t{0] = 0.9
iteration =  102, error = 0.00010
```

- 乱数シード値 30000

```
省略
iteration =   95, error = 0.00010
iteration =   96, error = 0.00010
iteration =   97, error = 0.00010
iteration =   98, error = 0.00010
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10473, t{0] = 0.1
```

```
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.89016, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.89001, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.92381, t{0] = 0.9
iteration =   98, error = 0.00010
```

- 乱数シード値 40000

```
省略
iteration =  104, error = 0.00010
iteration =  105, error = 0.00010
iteration =  106, error = 0.00010
iteration =  107, error = 0.00010
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10660, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.88994, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.88993, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.92310, t{0] = 0.9
iteration =  107, error = 0.00010
```

- 乱数シード値 50000

```
省略
iteration =  115, error = 0.00010
iteration =  116, error = 0.00010
iteration =  117, error = 0.00010
iteration =  118, error = 0.00010
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10661, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.88992, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.88994, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.92311, t{0] = 0.9
iteration =  118, error = 0.00010
```

実行結果より, 重みが変わっても約 100 回程度の学習で収束する事が分かる。

2.2 Level 1.2

5回の実行結果の平均を取るため以下のスクリプトを作成した。

```
#!/bin/sh

#このスクリプトは実行結果の内, グラフ作成に必要な部分を
#抜き出して平均を計算してくれるが, 最後の数行に不要な部分
#が含まれてしまうためその数行は手動で削除しないといけない ····

#引数チェック
if [ $# -lt 5 ] ; then
  echo "引数を 5 つにしてください"
  exit
elif [ $# -gt 5 ] ; then
  echo "引数を 5 つにしてください"
  exit
fi

#引数で指定したファイルが存在するか判定
if [ ! -e $1 ] ; then
  echo "$1 は存在しません"
  exit
fi
```

```
#引数で指定したものがファイルかどうか判断
if [ ! -f $1 ] ; then
  echo "$1 はファイルではありません"
  exit
fi


##実行結果を
##ファイル"learn.data"に格納する
learn=‘ cut -c 26-36 $1‘
learn2=‘ cut -c 26-36 $2‘
learn3=‘ cut -c 26-36 $3‘
learn4=‘ cut -c 26-36 $4‘
learn5=‘ cut -c 26-36 $5‘

echo " " > learn.data
flag=1

count=0

for learn in $learn
do

ARRAY[$count]=$learn
#echo ${ARRAY[$count]}
count=‘expr $count + 1‘

done

count=0

for learn6 in $learn2
do

ARRAY2[$count]=$learn6
#echo ${ARRAY2[$count]}
count=‘expr $count + 1‘

done

count=0

for learn7 in $learn3
do

ARRAY3[$count]=$learn7
#echo ${ARRAY3[$count]}
count=‘expr $count + 1‘

done

count=0

for learn8 in $learn4
do

ARRAY4[$count]=$learn8
#echo ${ARRAY4[$count]}
count=‘expr $count + 1‘

done

count=0

for learn9 in $learn5
do

ARRAY5[$count]=$learn9
#echo ${ARRAY5[$count]}
count=‘expr $count + 1‘

done
```

```
i=0

while [ $i -le $count ]
do
  if [ $flag -eq 1 ] ; then
    j=‘echo ${ARRAY[$i]}‘
k=‘echo ${ARRAY2[$i]}‘
l=‘echo ${ARRAY3[$i]}‘
m=‘echo ${ARRAY4[$i]}‘
n=‘echo ${ARRAY5[$i]}‘

#echo $j
#echo $k

    data=‘echo "scale=7; $j +  $k + $l + $m + $n" | bc‘
data1=‘echo "scale=5; $data / 5" | bc‘
    flag=0

i=‘expr $i + 1‘
  else
echo "$i  0$data1" >> learn.data
flag=1
  fi
done
```

実際に実行して平均を取ってみた。

```
[j05002@bp_mo]% ./a.out 10000 > test.txt
[j05002@bp_mo]% ./a.out 20000 > test1.txt
[j05002@bp_mo]% ./a.out 30000 > test2.txt
[j05002@bp_mo]% ./a.out 40000 > test3.txt
[j05002@bp_mo]% ./a.out 50000 > test4.txt
ここで生成したテキストの不要な数行を削除しなければいけない。
[j05002@bp_mo]% sh count2.sh test.txt test1.txt test2.txt test3.txt test4.txt
[j05002@bp_mo]% gnuplot
省略
gnuplot> plot "learn.data" w l
```

以下の図が学習結果のグラフである。

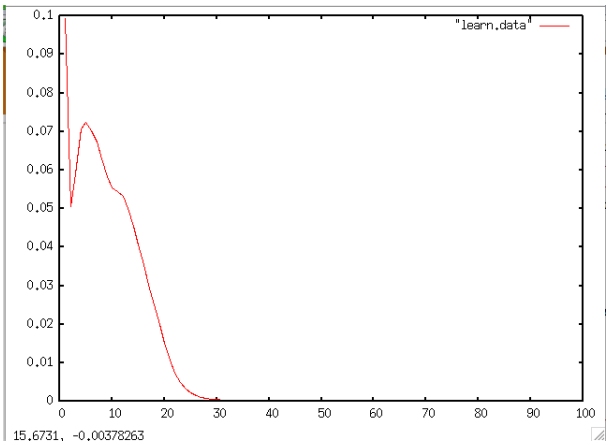

図 1: グラフ

3 Level 2

3.1 Level 2.1

bp_mo_exor.c のソースの抜粋を以下に示す

```
省略
int main(argc,argv)
        int     argc;
        char **argv;
{
  int   i,j,ite,ctg;
  double err,sum;
  int seed;
省略
  /* ExOR Problem */
  i_lay[0][0]=OFF; i_lay[0][1]=OFF; i_lay[0][2]=ON; teach[0][0]=OFF;
  i_lay[1][0]=ON;  i_lay[1][1]=OFF; i_lay[1][2]=ON; teach[1][0]=ON;
  i_lay[2][0]=OFF; i_lay[2][1]=ON;  i_lay[2][2]=ON; teach[2][0]=ON;
  i_lay[3][0]=ON; i_lay[3][1]=ON;  i_lay[3][2]=ON; teach[3][0]=OFF;
省略
```

シード値を変更して複数回実行しても以下の実行結果のように, 学習が適切に収束しない。

```
iteration = 99996, error = 0.05679
iteration = 99997, error = 0.05679
iteration = 99998, error = 0.05679
iteration = 99999, error = 0.05679
iteration = 100000, error = 0.05679
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.36648, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.36635, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.90262, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.36647, t{0] = 0.1
iteration = 100000, error = 0.05679
```

ExOR 問題が OR より困難な理由として上げられるのが以下の図に示したように On と Off の線引きが OR では一本で十分なのに対して,ExOR では少なくとも二本必要であり, 中間層のユニット数が少ないのが理由として上げられる。

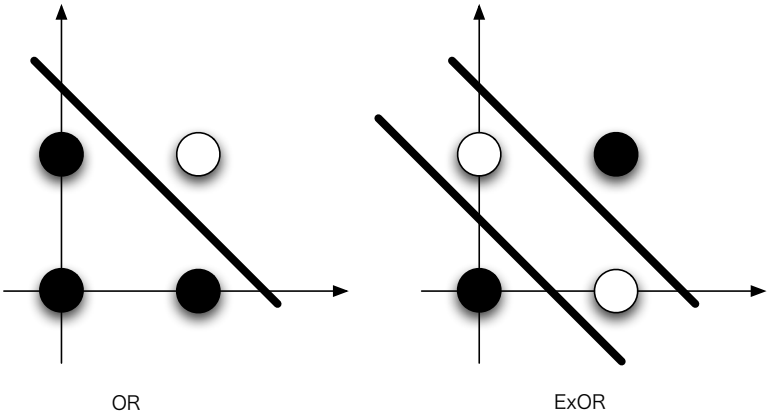

図 2: OR と ExOR

3.2 Level 2.1

各パラメータを以下のように変更した

学習係数 ETA	1.99
慣性項 ALPHA	0.94
中間数のユニット数 HIDDEN	15

実行結果を以下に示す。

シード値	収束したときの回数
10000	37
20000	35
30000	51
40000	30
50000	32

4 Level 3

4.1 Level 3.1

実行結果 (eva1-1.txt)

```
filename? --> data/eva1-1.txt
correct = 0100000000
0000010000
0000110000
0000010000
```

```
0000010000
0000010000
0000010000
0000010000
0000010000
0000010000
0001111000
CHECK filename data/eva1-1.txt
EVA o[0] = 0.00423, correct[0] = 0.0
EVA o[1] = 0.97945, correct[1] = 1.0
EVA o[2] = 0.00004, correct[2] = 0.0
EVA o[3] = 0.00928, correct[3] = 0.0
EVA o[4] = 0.00736, correct[4] = 0.0
EVA o[5] = 0.00041, correct[5] = 0.0
EVA o[6] = 0.00011, correct[6] = 0.0
EVA o[7] = 0.00218, correct[7] = 0.0
EVA o[8] = 0.00322, correct[8] = 0.0
EVA o[9] = 0.00158, correct[9] = 0.0
EVA sum_error = 0.04895
```

eva1-1.txt での実行結果は, o[1] = 0.97945 で,”1”に一番近いと判断されている。

実行結果 (eva1-2.txt)

```
filename? --> data/eva1-2.txt
correct = 0100000000
0000000000
0000000000
0000000100
0000000100
0000000100
0000000100
0000000100
0000000100
0000000100
0000000000
CHECK filename data/eva1-2.txt
EVA o[0] = 0.00152, correct[0] = 0.0
EVA o[1] = 0.00044, correct[1] = 1.0
EVA o[2] = 0.00181, correct[2] = 0.0
EVA o[3] = 0.38005, correct[3] = 0.0
EVA o[4] = 0.00168, correct[4] = 0.0
EVA o[5] = 0.06357, correct[5] = 0.0
EVA o[6] = 0.00226, correct[6] = 0.0
EVA o[7] = 0.00421, correct[7] = 0.0
EVA o[8] = 0.00190, correct[8] = 0.0
EVA o[9] = 0.00030, correct[9] = 0.0
EVA sum_error = 1.45685
```

eva1-2.txt での実行結果は, o[3] = 0.38005 が一番高い値となっており, ”3”に一番近いと判断されている。

4.2 Level 3.2

まず HIDDEN の値を 10 と 15 に変更し, 両者を比較してみた (その他のパラメータは変更していない)

1. HIDDEN が 10 の時の実行結果

```
省略
iteration = 2500 error = 0.00010
iteration = 2501 error = 0.00010
iteration = 2502 error = 0.00010
iteration = 2503 error = 0.00010
iteration = 2504 error = 0.00010
iteration = 2505 error = 0.00010
iteration = 2506 error = 0.00010
iteration = 2507 error = 0.00010
nn>
```

2. HIDDEN 15

```
iteration = 1205 error = 0.00010
iteration = 1206 error = 0.00010
iteration = 1207 error = 0.00010
iteration = 1208 error = 0.00010
iteration = 1209 error = 0.00010
iteration = 1210 error = 0.00010
iteration = 1211 error = 0.00010
nn>
```

以上の結果より HIDDEN の値が 10 の時より 15 の時の方が少ない学習回数ですんだ。
HIDDEN の値は高ければ高い方がよいと予測できるので, 今度は HIDDEN の値を 70 としてみた。

```
iteration = 3996 error = 0.05001
iteration = 3997 error = 0.05001
iteration = 3998 error = 0.05001
iteration = 3999 error = 0.05001
iteration = 4000 error = 0.05001
nn>
```

この結果を見ると,iteration = 4000 になってもまだ error = 0.05001 で学習回数が増えている。以上のことより,HIDDEN の値は高い方がよいが高すぎると逆に学習回数が増えるということが分かる。

次に ETA の値を 1.50 と 1.70 の場合で試してみた。

1. ETA の値が 1.50 の場合

```
iteration = 8417 error = 0.00010
iteration = 8418 error = 0.00010
iteration = 8419 error = 0.00010
iteration = 8420 error = 0.00010
iteration = 8421 error = 0.00010
iteration = 8422 error = 0.00010
nn>
```

2. ETA の値が 1.70 の場合

```
iteration = 6588 error = 0.00010
iteration = 6589 error = 0.00010
iteration = 6590 error = 0.00010
iteration = 6591 error = 0.00010
iteration = 6592 error = 0.00010
iteration = 6593 error = 0.00010
iteration = 6594 error = 0.00010
iteration = 6595 error = 0.00010
nn>
```

以上の結果より ETA の値は高い方がよいと予測した。

次は,ALPHA の値が 0.5 と 0.7 の場合を比較してみた。

1. ALPHA の値が 0.5 の場合

```
iteration = 5678 error = 0.00010
iteration = 5679 error = 0.00010
iteration = 5680 error = 0.00010
iteration = 5681 error = 0.00010
iteration = 5682 error = 0.00010
iteration = 5683 error = 0.00010
iteration = 5684 error = 0.00010
nn>
```

2. ALPHA の値が 0.7 の場合

```
iteration = 4582 error = 0.00010
iteration = 4583 error = 0.00010
iteration = 4584 error = 0.00010
iteration = 4585 error = 0.00010
iteration = 4586 error = 0.00010
iteration = 4587 error = 0.00010
iteration = 4588 error = 0.00010
nn>
```

以上の結果より,ALPHA の値は高いほうがよいと予測した。

最後にこの 3 つのパラメータの値を変更して少ない学習回数で済むようにしてみた。

我々のグループでは以下のパラメータの時に最小だと判断した

学習係数 ETA	1.79
慣性項 ALPHA	0.44
中間数のユニット数 HIDDEN	40

上記のパラメータにおいて,5 回実行したときの結果を以下に示す。

実行回数	収束したときの回数
1 回目	710
2 回目	673
3 回目	728
4 回目	831
5 回目	634

4.3 Level 3.3

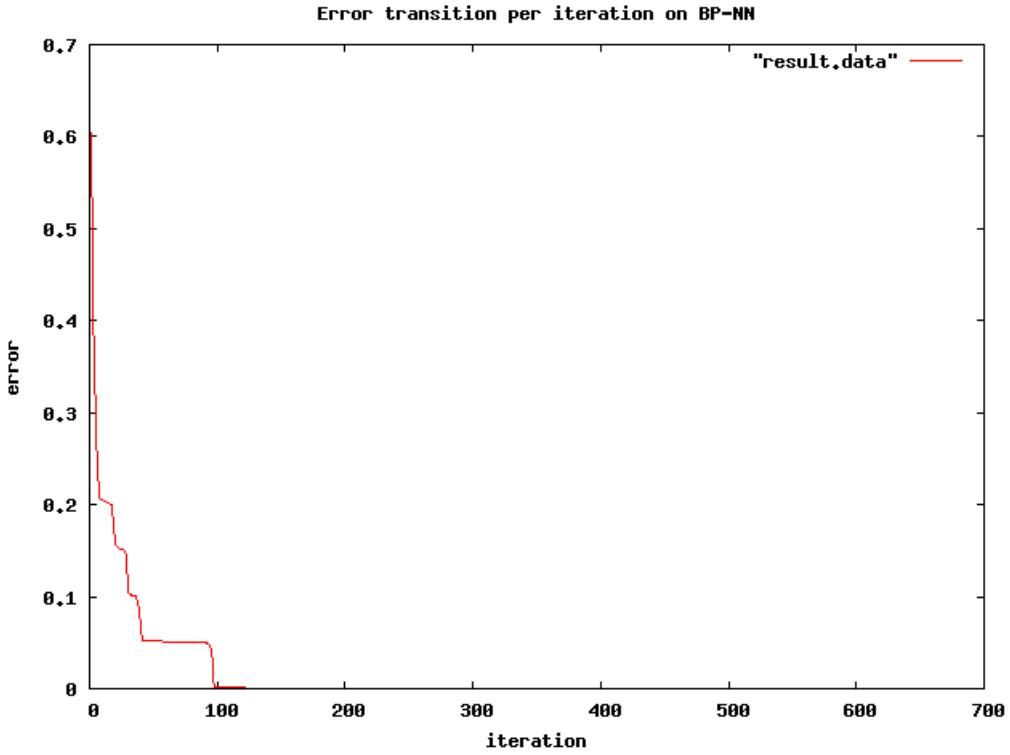

図 3: 学習曲線 (5 回目のもの)

中間層の数は Level3.2 で述べたように, 少なくても大きすぎても, 収束能力

は落ちてしまう。このことより, 入力層と出力層の数に対する最適な中間層の値が存在することが分かる。
中間層の求め方としては主に以下の 7 種類の方法があることが分かった。

1. 入力層の数と出力層の数との平均を取り調節する方法

   この方法は入力層の数と出力層の数を足して 2 で割った値を中間層の数とする方法である。

2. 中間層の数を入力層および出力層と同数とする方法

   この方法は、中間層の数を入力層および出力層と同数とする方法である。

3. 中間層の数を入力層の数の 70~80 パーセントにする方法

   この方法は、中間層の数を入力層の数に百分率で 70 %ぐらいから 80 ぐらいの数として、 次第に増加させていき最もコスト関数が理想の値に近づいたものを採用する方法である。

4. 中間層の数を入力ベクトル次元に依存する方法

   この方法は、中間層の数を入力するパターンと同数にする方法である。例えば ExOR 問題をニューラルネットワークに解かすのであれば、入力パターンは 4 つなので中間層の数を 4 つにする。

5. 入力層の数と出力層の数の積の平方根を中間層の数とする方法

   入力層が 3 つ出力層 1 つの場合、その積は 3 で平方根は 1.732 であるから中間層を 2 にする。

6. 入力層の数の 2 倍に 1 を加えたものを中間層の数とする方法

   入力層が 3 つ出力層 1 つの場合、入力層の 2 倍は 6 で 1 を加えると 7 となる。よって、中間層を 7 とする。

7. 試行錯誤的に求める方法

   この方法は試行錯誤的に中間層の数を決める方法である。まず、中間層の数を少なく設定して学習を行い、 最小二乗誤差が大きければその数を増やして誤差が減っていくなら中間層を増やす。そうでなければ数を減らといった方法で中間層の数を決める。

Level3.2 で色々パラメータを変えて実験したところ, 学習係数と慣性項の間には何らかの関係があり, この 2 つのパラメータのバランスが崩れると, 学習回数が上がったり, ある値 (今回の場合 0.05 付近など) でなかなか収束しなかったりして, 収束能力が落ちることが分かった。以下は,0.05 付近で収束しづらくなった時の実行結果である。

```
省略
iteration =  182 error = 0.05023
iteration =  183 error = 0.05023
省略
iteration =  469 error = 0.05008
iteration =  470 error = 0.05008
省略
iteration =  761 error = 0.05005
iteration =  762 error = 0.05005
省略
iteration = 1100 error = 0.05003
iteration = 1101 error = 0.05003
省略
iteration = 1705 error = 0.05002
iteration = 1706 error = 0.05002
省略
iteration = 2333 error = 0.05000
iteration = 2334 error = 0.04999
省略
```

4.4 Level 3.4

以下の 2 つの評価用データを作成した。

- eva1-3.txt

```
3
0000000000
0000000000
0111111110
0000000010
0000000010
0111111110
0000000010
0000000010
0111111110
0000000000
```

- eva1-7.txt

```
7
0000000000
0111111110
0100000010
0100000010
0000000010
0000000010
0000000010
0000000010
0000000010
0000000000
```

以下はその実行結果である。

- eva1-3.txt

```
correct = 0001000000
0000000000
0000000000
0111111110
0000000010
0000000010
0111111110
0000000010
0000000010
0111111110
0000000000
CHECK filename data/eva1-3.txt
EVA o[0] = 0.00011, correct[0] = 0.0
EVA o[1] = 0.00247, correct[1] = 0.0
EVA o[2] = 0.12670, correct[2] = 0.0
EVA o[3] = 0.00064, correct[3] = 1.0
EVA o[4] = 0.17308, correct[4] = 0.0
EVA o[5] = 0.04819, correct[5] = 0.0
EVA o[6] = 0.00000, correct[6] = 0.0
EVA o[7] = 0.00058, correct[7] = 0.0
EVA o[8] = 0.00771, correct[8] = 0.0
EVA o[9] = 0.35606, correct[9] = 0.0
EVA sum_error = 1.71425
```

o[9] = 0.35606 が一番高い値なので,”9”に一番近いと判断されている。

- eva1-7.txt

```
correct = 0000000100
0000000000
0111111110
0100000010
0100000010
0000000010
0000000010
0000000010
0000000010
0000000010
0000000000
CHECK filename data/eva1-7.txt
EVA o[0] = 0.04301, correct[0] = 0.0
EVA o[1] = 0.00050, correct[1] = 0.0
EVA o[2] = 0.11218, correct[2] = 0.0
EVA o[3] = 0.00026, correct[3] = 0.0
EVA o[4] = 0.06873, correct[4] = 0.0
EVA o[5] = 0.00010, correct[5] = 0.0
EVA o[6] = 0.00289, correct[6] = 0.0
EVA o[7] = 0.13413, correct[7] = 1.0
EVA o[8] = 0.06157, correct[8] = 0.0
EVA o[9] = 0.00073, correct[9] = 0.0
EVA sum_error = 1.15583
```

o[7] = 0.13413 が一番高い値なので, 今回は正しく”7”と判断された。

以上の結果より, 学習用のデータとのズレが大きかったり, 他のデータと似通っているデータは正しく判断できないことが分かる。

4.5 Level 3.5

0 と 1 で数字を描いて認識させているので,1 が連続している部分や 1 が最も固まっている部分を学習の対象として認識させ, サイズが小さい場合は対象部分の周囲の 0 を 1 として認識させるようにすれば上手くいくと考えた。例えば次のようなものである。

```
1
0000000000          0000000000
0000000000          0000111000
0000100000          0000111000
0000100000          0000111000
0000100000    →     0000111000
0000100000          0000111000
0000000000          0000111000
0000000000          0000000000
0000000000          0000000000
0000000000          0000000000
```

位置がずれている場合は, 対象部分のズレを計算させ, そのズレ分だけ補正をかけるようにプログラムすれば上手くいくと考えた。(全体の中心の位置を定義し, 学習対象と思われる範囲を包囲できる最小の長方形を計算し, その中心を求めその 2 つの中心の位置の差を修正する, という計算を行えればよい。)

5 Level 4

5.1 解説

ニューラルネットワークを用いて、株価の予測を行う。
過去のデータを学習データとして用いて、今後どのように株価が変動していくかを予測させる。
予測の制度を上げるためには、入力データを株価のみでなく、その日に起きた出来事も考慮しなければいけない。これは、コンピュータで扱うにはあまりにも曖昧なデータであると考えられる。そこで、どの出来事がどの程度株価に影響を与えるかを過去のデータからその関係を求め、その情報をもとに、ある出来事が株価に影響があるかどうかはユーザーが判断し、影響があると考えられる場合には、その影響の程度をシステムに入力し株価を予測する。

5.2 入力層・出力層の構成

- 入力
  予測したい銘柄の過去 10 日間のデータを入力とする。
  その日に起きた出来事の株価に対する重要度。

- ニューロン数
  入力したデータに対応する重要度を決定するために、ニューロン数は入力データと同じ数とする。

- 出力
  過去のデータにより予測された、結果を出力とする。