認識行動システムの基礎　演習問題１

341052G　(計数)
さいたまおとこ

2005 年 1 月 26 日提出

1.

パターンはその要素が時間経過の次元を含め n 個の標本点から構成されるとすれば、n 次元のベクトルで表すことができる。パターン認識とはこの空間 $\boldsymbol{R}^n$ 内での各点のクラス分け・カテゴリ分けに相当する。

次元 n は平面上の画素数のように通常巨大な数になるので、n 次元ベクトルを考える時には考えうるベクトルの組み合わせはさらに莫大な数になる。これは

- 計測・入力時の手間や
- 情報の保持・処理機構自体にかかる手間を大きくし、
- 判別・カテゴリ分けを困難にする。

2.

パターンの属する空間を 2 クラスに分離できる問題の中でも、特にクラス判別の条件式が $\boldsymbol{R}^n$ 内での明示的な $\boldsymbol{R}^{n-1}$ での分割を表している場合に線形識別可能であるという。非線形な 2 クラス識別問題であってもデータ空間内で適当に基底を取り直して簡単な線形識別の問題に帰着させることができる。

3.

特定のトレーニングセットに依存しない識別を可能にすること。学習時に精密な識別境界を作っておくことで、未知のデータに対する誤判別を減らし安定した識別結果を得る事ができる。ニューロンモデルならば隠れ層のニューロンを増やしてやる事にあたる。

4.

(I)　クラス内で平均を取って、それぞれの代表ベクトル $(\frac{1}{2}, \frac{3}{2})$ と $(3, \frac{3}{2})$ の作る識別面は 2 点から等距離の $x = \frac{7}{4}$

(II)　$x = 2$ をとればクラスＢ各点と $\boldsymbol{a}_2$ の間でマージン最小化がなされるので適する。

5.

多層ニューラルネットワークで、教師データ d と出力層からの出力の誤差 E を

$$E_c = \frac{1}{2}\sum_j (y_{j,c} - d_{j,c})^2, \quad E = \sum_c E_c$$

と決め、これを最小化するように各層での重みを

$$w_{n+1} = w_n + \Delta w = w_n - \eta \frac{\partial E}{\partial w(n)}$$

として逐次更新する。精確な $\boldsymbol{w}$ を得るには係数 η を小さくとるが、このとき $\frac{\partial E}{\partial w(n)} = 0$ となるような E 上の極小値で更新が終了してしまうローカルミニマムの問題がある。これを避けるために BP 法では η は割と大きくとって

$$w_{n+1} = w_n + \Delta w = w_n + \left(-\eta \frac{\partial E}{\partial w(n)} + \alpha \Delta w_{n-1} \right)$$

のように、前回の更新動作を $\alpha > 0$ を係数とする慣性項として反映し落ち込みを回避している。

※生物の見る**夢**もまた、起きている間の記憶 (教師データ) を再生する事により至るところにある局所解つまり偽記憶を除去して、記憶がローカルミニマムに陥るのを避けているのだと言われる。一方で最近の量子場脳理論では、夢は虚数空間からの記憶の演算子 (インスタントン) の染み出しで、単なる非現実的な記憶の生成にすぎないともされる。どちらをとってもおもしろい。

6.

以下のプログラムでモデルを立て計算した。

```
#include <stdio.h>
#include <math.h>

double f(double x); //素子内の入出力特性

//ニューロン
struct neuron{
 double value;
 double w[3];
 double dw[3];//10 行目
};

int main(void){
 int i,c,sk,j,tj,h;
 double idt[8]={1,0,2,3,0,1,3,0},ddt[8]={1,0,1,0,0,1,0,1},
  odt[8],mdt[8],tdelc,jsum;
 struct neuron n[6]; //生成

 //初期化
 for(i=0;i<6;i++){ //20 行目
  n[i].w[1]=0;
  n[i].w[2]=0;
 }
```

```
n[0].w[1]=1;
n[2].w[1]=1;
for(i=0;i<6;i++){
 printf( "(%d)w1:%f w2:%f\n", i+1,n[i].w[1],n[i].w[2] );
}

//データ入力 //30 行目
for(c=0;c<4;c++){ //c(パターン番号) ごと
 n[0].value = idt[2*c];   //入力素子に入力
 n[1].value = idt[2*c + 1];
 for(sk=1;sk<3;sk++)
 for(j=0;j<2;j++){  //sk(層番号) ごとに重みつき和
  n[2*sk+j].value = n[2*sk-2].w[j+1] * f( n[2*sk-2].value )
   + n[2*sk-1].w[j+1] * f( n[2*sk-1].value );
 }
 mdt[2*c]=n[2].value;mdt[2*c+1]=n[3].value;//層ごとの出力を保存
 odt[2*c]=n[4].value;odt[2*c+1]=n[5].value;  //40 行目
 printf( "out(%d):%f %f\n", c+1,odt[2*c],odt[2*c+1] );
}

//ih 間で最急降下法による重みの更新量を求める
for(i=0;i<2;i++)
for(h=0;h<2;h++){
 tdelc=0;
 for(c=0;c<4;c++){
  jsum=0;
  for(tj=0;tj<2;tj++){ //50 行目
   jsum+=(odt[2*c+tj]-ddt[2*c+tj])*odt[2*c+tj]*(1-odt[2*c+tj])
    *n[2+tj].w[1+tj];
  }
  tdelc +=
  jsum * mdt[2*c+i]*(1-mdt[2*c+i])* idt[2*c+h];
 }
 n[i].dw[j]+=tdelc;
 printf("w%d:dif=%f\n",1+2*i+h,tdelc);
}
//60 行目
 //ji 間
 for(i=0;i<2;i++)
 for(j=0;j<2;j++){
```

```
  tdelc=0;
  for(c=0;c<4;c++){
   tdelc += (odt[2*c+j] - ddt[2*c+j]) *odt[2*c+j]*(1-odt[2*c+j])*mdt[2*c+i];
  }
  n[2+i].dw[j]+=tdelc;
  printf("w%d:dif=%f\n",5+2*i+j,tdelc); //70 行目
 }
 return 0;
}

double f(double x){
 return (1/(1+exp(-x))); //シグモイド関数
}
```

結果：

```
w1:dif=-0.007115
w2:dif=0.017451
w3:dif=0.000000
w4:dif=0.000000
w5:dif=0.105649
w6:dif=0.000000
w7:dif=0.000000
w8:dif=0.000000
```

線形のネットワークにしたときは

```
w1:dif=11.000000
w2:dif=3.000000
w3:dif=11.000000
w4:dif=3.000000
w5:dif=11.000000
w6:dif=-3.000000
w7:dif=0.000000
w8:dif=0.000000
```

感想：

多層でシグモイド関数を使った手計算は大変です。去年、期末試験では 1 層線形モデルの計算が出題されたそうな・・・。