

電子情報工学専攻 特別実験（I）

『ニューラルネットワークによるパターン認識』

担当：桐島俊之

1 本実験の目的

我々人間に限らずあらゆる生命体は自己とその環境との関わりの中で生きている。このため、外部環境を常に把握し、生存に最適な行動を選択しなければならない。この際に不可欠となるのが「パターン認識」能力である。生命体はこの能力によってのみ外界を認識し、敵・味方の区別、食物と毒物を区別することができる。コンピュータ技術の著しい向上は、こうしたパターン認識能力を限定的であるにせよ模倣することを可能としている。このことは、生命体以外の存在である機械に知能を付与することに他ならない。機械に知能を与えることで、従来よりも高度な作業を柔軟かつ高速に実行させることが可能になる。

しかしながら、我々の住む世界は極めて複雑な要因・要素により構成されているため、このような世界（これを実世界と呼ぶ）で生じる複雑かつ多様な事象を機械に認識させることは容易でない。具体的には、曖昧さやノイズを許容する柔軟な認識アルゴリズムが不可欠となるが、ニューラルネットワークはこうした要求を背景として、近年盛んに応用・研究されているパターンの学習と認識を行う手法の一つである。

本実験は、ニューラルネットワークを利用したパターン認識実験を行い、その基本的性質と特性を明らかにすることを目的とする。

2 パターン認識とは

パターン認識 (pattern recognition) は、認識対象がいくつかの概念に分類できる時、観測されたパターンをそれらの概念のうちのひとつに対応させる処理である。この概念をクラス (class) あるいは類 (category) と呼ぶ。例えば、数字の認識は、入力パターンを 10 種類の数字のいずれかに対応させることが目的である。

パターン認識における最も基本的な課題は、未知の認識対象を計測して得られた特徴ベクトルからその対象がどのクラスに属するかを判定する識別方法を開発することである。そのためには、まず、クラスの帰属が既知の学習用のサンプル集合から特徴ベクトルとクラスとの確率的な対応関係を知識として学習することが必要である。本実験では、近年、パターン認識にも盛んに利用されるようになったニューラルネットワークに焦点を当てる。

3 ニューラルネットワークの歴史的背景

現在のニューラルネットワークの原点は、脳におけるパターン学習と識別法のモデルとして、1962 年にローゼンブラット (Rosenblatt) が提案した単純パーセプトロンにあると言われる。単純パーセプトロンの結合荷重を推定するための学習アルゴリズムとしては、いくつかの方法が提案されているが、Rosenblatt らの方法は、ネットワークにあるパターンを分類させてみて間違っていたら結合荷重を修正する誤り訂正の方法であった。

しかし、この学習規則は、線形分離可能でない場合、すなわち、誤識別 0 にする線形識別関数が存在しない場合には、誤り訂正の手続きを無限に繰り返しても解に到達できない可能性がある。また、学習を途中で打ち切った場合に得られるパラメータが最適であるという保証がない。このように、古典パーセプトロンには、パターンの線形分離条件が満たされない限り、解が収束しないという問題があり、1960 年代のニューラルネットワーク研究ブームに水を差す結果となった。

そこで、識別関数が線形に留まるのを止め、例えば楕円などの 2 次以上の非線形な識別関数へ拡張を図る方策が見出された。それは、取り扱いに不都合なパターン空間をもたらした根本的原因を取り除くように識別過程の前段階である特徴抽出にも並列かつ学習の概念を持ち込むというものである。これは言わば、パーセプトロンを多層化した構成のシステムとなる。

4 階層型ニューラルネットワーク

本節では、階層型ニューラルネットワークにおける多層パーセプトロンの学習法として重要な位置付けにある誤差逆伝播法（B P 法：Back-Propagation method）について説明する。古典パーセプトロンの拡張としての多層パーセプトロンは、Rumelhart らが誤差逆伝播法を提案して以来、パターン認識や制御などのさまざまな問題に適用され、その有用性が実証されている。

4.1 誤差逆伝播法によるパターン学習

多層パーセプトロンは任意の連続関数を近似するのに十分な表現能力をもっているが、そうしたネットワークに望みの情報処理をさせるためにはユニット間の結合荷重を適切なものに設定しなければならない。ユニットの数が増えると結合荷重の数も増え、それらをいちいち設定することは難しい。一般に、それらは利用可能なデータからの学習によって求められる。そのためのアルゴリズムとしては、最急降下法に基づく誤差逆伝播学習法が最も一般的に利用されている。

ここでは、中間層のユニットの入出力関数がロジスティック関数で、出力層のユニットの入出力関数が線形であり、さらに中間層が n 層の場合のネットワークにおける誤差逆伝播法について説明する。図 1 に誤差逆伝播法の説明図を示す。

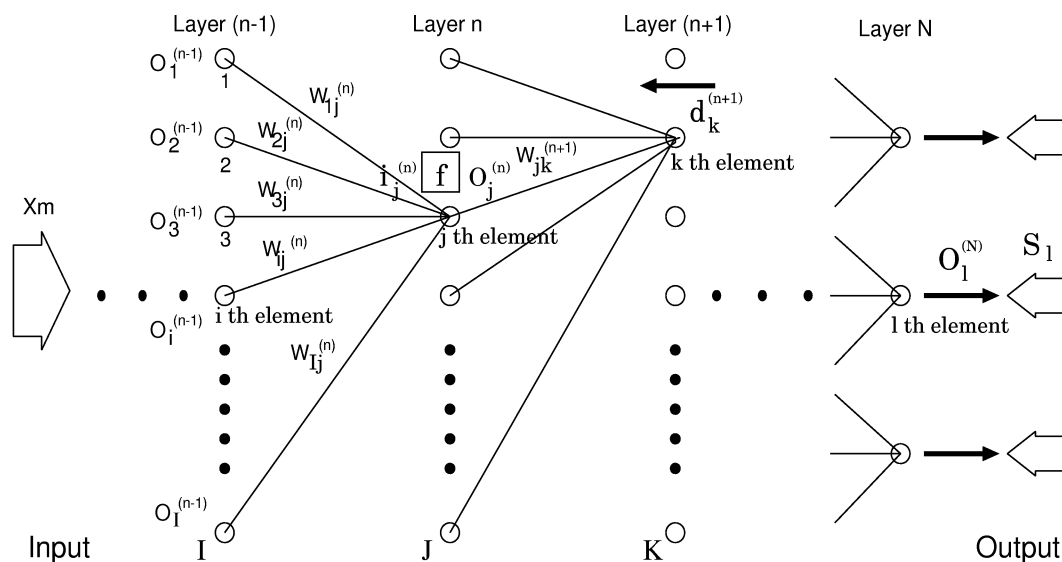


図 1: 誤差逆伝播法説明図

誤差逆伝播法では、単純パーセプトロンによる学習とは異なり、各層を構成する要素の入出力特性に非減少で微分可能（すなわち準線形；完全な線形では等価的に一層で表現できるため階層化の意味がない）な関数の使用を必要とする．図 1 の第 n 層の要素 j について入出力特性の式は、

$$\text{入力} : i_j^{(n)} = \sum_i w_{ij}^{(n)} o_i^{(n-1)} \quad (1)$$

により表すことができる．ここで、 $w_{ij}^{(n)}$ は第 $(n-1)$ 層の要素 i から第 n 層の要素 j を結ぶ重みを意味する．今、出力を

$$\text{出力} : o_j^{(n)} = f \left(i_j^{(n)} \right) \quad (2)$$

により定義する．関数 f の具体的な関数型としては、通常、式 (3) に示すようなロジスティック関数が用いられる．

$$f(x) = \frac{1}{1 + \exp^{-x}} \quad (3)$$

4.2 誤差逆伝播法における学習則の導出

誤差逆伝播法による学習は、教師付き学習に属し、システムの識別出力と教師信号が一致するように重みを変えて学習を進める．誤差逆伝播法とは、この誤差を可能な限り小さくするための数学的手法である．誤差逆伝播法における学習則の導出方法を以降に示す．

パターン x_m が提示されたときの出力誤差を

$$E_{x_m} = \sum_{k=1}^K \frac{\left(S_k - o_k^{(n+1)} \right)^2}{2} \quad (4)$$

とする．最急降下法を使って E_{x_m} を可能な限り小さくする．まず、

$$w_{jk}^{(n+1)} = w_{jk}^{(n)} + \Delta w_{jk}^{(n+1)} \quad (5)$$

ここで、

$$\begin{aligned} \Delta w_{jk}^{(n+1)} &= -\varepsilon \frac{\partial E_{x_m}}{\partial w_{jk}^{(n+1)}} \\ &= -\varepsilon \frac{\partial E_{x_m}}{\partial i_k^{(n+1)}} \cdot \frac{\partial i_k^{(n+1)}}{\partial w_{jk}^{(n+1)}} \end{aligned} \quad (6)$$

$$= -\varepsilon \left(\frac{\partial E_{x_m}}{\partial o_k^{(n+1)}} \cdot \frac{\partial o_k^{(n+1)}}{\partial i_k^{(n+1)}} \right) O_j^{(n)} \quad (7)$$

$$= \varepsilon \left(S_k - o_k^{(n+1)} \right) f' \left(i_k^{(n+1)} \right) O_j^{(n)} \quad (8)$$

次に、

$$w_{ij}^{(n)} = w_{ij}^{(n-1)} + \Delta w_{ij}^{(n)} \quad (9)$$

ここで、

$$\begin{aligned} \Delta w_{ij}^{(n)} &= -\varepsilon \frac{\partial E_{x_m}}{\partial w_{ij}^{(n)}} \\ &= -\varepsilon \frac{\partial E_{x_m}}{\partial i_j^{(n)}} \cdot \frac{\partial i_j^{(n)}}{\partial w_{ij}^{(n)}} \end{aligned} \quad (10)$$

$$= -\varepsilon \left(\frac{\partial E_{x_m}}{\partial o_j^{(n)}} \cdot \frac{\partial o_j^{(n)}}{\partial i_j^{(n)}} \right) O_i^{(n-1)} \quad (11)$$

$$= -\varepsilon \frac{\partial E_{x_m}}{\partial o_j^{(n)}} f' \left(i_j^{(n)} \right) O_i^{(n-1)} \quad (12)$$

$$= -\varepsilon \left(\sum_{k=1}^K \frac{\partial E_{x_m}}{\partial i_k^{(n+1)}} \cdot \frac{\partial i_k^{(n+1)}}{\partial o_j^{(n)}} \right) f' \left(i_j^{(n)} \right) O_i^{(n-1)} \quad (13)$$

$$= \varepsilon \left(\sum_{k=1}^K d_k^{(n+1)} w_{jk}^{(n+1)} \right) f' \left(i_j^{(n)} \right) O_i^{(n-1)} \quad (14)$$

$$(15)$$

このような最急降下法を用いた学習法では、学習率をどのように決めるかによってアルゴリズムの収束の速さが影響を受けるので、学習率を適切な値に設定するための方法がいくつかの提案されている（例えば [1]）。また、学習の高速化に関しては、多くの方法が提案されている。例えば、Quick Prop[2] は、多くのヒューリスティックを組み合わせて学習を高速化している。

誤差逆伝播法は、学習をパターンの特徴抽出レベルまで拡張することにより、古典パーセプトロンの欠点を解決する画期的なものである。この点において興味深い事柄は、ニューラルネットワークの研究者がシステム全体の入出力特性だけに注目するのではなく、学習終了時における中間層（隠れ層：Hidden Layer）の状態に非常な関心を抱いているということである。この理由としては、システムのこの部分において学習による最適な特徴抽出処理が実行されているとの推測が挙げられる。実際に、幾つかの画像処理において動物の脳視覚経路が持つ細胞の機能と同様な特性が現れたという報告がある。また、理論的には、ネットワークの能力、学習アルゴリズムの高速化、多変量データ解析との関係、汎化能力の高いネットワークを構成するための方法などに関する多くの知見が既に得られている。

4.3 階層型ニューラルネットワークの実装

前節で導かれた学習則を整理すると以下ようになる。

$$w_{ij}^{(n)}(t+1) = w_{ij}^{(n)}(t) + \Delta w_{ij}^{(n)}(t) \quad (16)$$

$$\Delta w_{ij}^{(n)}(t) = \varepsilon d_j^{(n)} o_i^{(n-1)} + \alpha \Delta w_{ij}^{(n)}(t-1) \quad (17)$$

$$d_j^{(n)} = \left(\sum_k w_{jk}^{(n+1)}(t) d_k^{(n+1)} \right) \cdot f' \left(i_j^{(n)} \right) \quad (n \neq N) \quad (18)$$

$$d_l^{(N)} = \left(S_l - o_l^{(N)} \right) \cdot f' \left(i_l^{(N)} \right) \quad (n = N) \quad (19)$$

ここで、 $d_i^{(\cdot)}$ ：学習信号、 ε ：学習定数、 α ：安定化係数である。この学習則をC言語を用いてプログラミングすることにより、階層型ニューラルネットワークを実装する。プログラム例を以下に示す。なお、以下の例は、 n 層の階層型ニューラルネットワークを実現するプログラムである。

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

#define MAX_LAYER 3
#define MAX_ELEMENT 12      +1

double output[ MAX_LAYER ][ MAX_ELEMENT ];
double input[ MAX_LAYER ][ MAX_ELEMENT ];
```

```

double weight[ MAX_LAYER ][ MAX_ELEMENT ][ MAX_ELEMENT ];
double dweight[ MAX_LAYER ][ MAX_ELEMENT ][ MAX_ELEMENT ];
double back_output[ MAX_LAYER ][ MAX_ELEMENT ];

/*
double input[ MAX_LAYER ][ MAX_ELEMENT ];
double output[ MAX_LAYER ][ MAX_ELEMENT ];

double weight[ MAX_LAYER ][ MAX_ELEMENT ][ MAX_ELEMENT ];
double dweight[ MAX_LAYER ][ MAX_ELEMENT ][ MAX_ELEMENT ];

double back_output[ MAX_LAYER ][ MAX_ELEMENT ];
*/

int layer_structure[ MAX_LAYER ]={ 4,10,12 }; /* 各構成 < MAX_ELEMENT */

double pattern[12][4]={
    { 0,0,0,1 },
    { 0,0,1,0 },
    { 0,0,1,1 },
    { 0,1,0,0 },
    { 0,1,0,1 },
    { 0,1,1,0 },
    { 0,1,1,1 },
    { 1,0,0,0 },
    { 1,0,0,1 },
    { 1,0,1,0 },
    { 1,0,1,1 },
    { 1,1,0,0 }
};

double rnd()
{
    return (random() & 0xffffffff)/(255.*65535.);
}

double calc_f( double x )
{
    return 1./ ( 1.+exp( -x ) );
}

void Neuron_weight_init( void )
{
    int i,i2,i3;

    for( i=0 ; i<MAX_LAYER ;++i )
    {
        for( i2=0; i2<MAX_ELEMENT ;++i2 ) /* from layer n */
        {
            for( i3=0; i3<MAX_ELEMENT ;++i3 ) /* to layer n+1 */
            {
                weight[ i ][ i2 ][ i3 ]= rnd();
            }
        }
    }

    return;
}

```

```

void Forward_propagation( int pattern_no,int layer_no )
/* layer_no must be above 1 */
{
    int i,i2,i3;
    double sum;

    /*
    output[ layer_no-1 ][ layer_structure[ layer_no-1 ] ]= -1;
    */

    /* 出力値の計算 */
    for( i=0; i < layer_structure[ layer_no ] ;++i )
    {
        sum=0;
        for( i2=0; i2<= layer_structure[ layer_no-1 ] ; ++i2 )
        {
            sum+= weight[ layer_no ][ i2 ][ i]*output[ layer_no-1 ][ i2 ];
        }
        input[ layer_no ][ i ]= sum;
        output[ layer_no ][ i ]= calc_f( sum );
    }
    return ;
}

void Back_propagation( int pattern_no, int layer_no )
{
    int i,i2,i3,ii,j;
    double sum,sum2;

    if ( layer_no== MAX_LAYER-1 )
    /* 終端出力での誤差逆伝搬 */
    {
        for( j=0 ; j < layer_structure[ layer_no ] ;++j )
        {
            back_output[ layer_no ][ j ]=
                ( ( j==pattern_no ? 1. : 0 ) - output[ layer_no ][ j ] )
                *calc_f( input[layer_no][j] )*( 1.-calc_f( input[layer_no][j] ) );
            for( ii=0; ii <= layer_structure[ layer_no-1 ] ;++ii )
            {
                dweight[ layer_no ][ ii ][ j ]=
                    0.25*back_output[ layer_no ][ j ]*output[ layer_no-1 ][ ii ]
                    +0.9*dweight[ layer_no ][ ii ][ j ];
                weight[ layer_no ][ ii ][ j ] += dweight[ layer_no ][ ii ][ j ];
            }
        }
    }
    else
    {
        /* 中間出力での誤差逆伝搬 */
        for( j=0 ; j < layer_structure[ layer_no ] ;++j )
        {
            sum=0;
            for( i=0; i<layer_structure[ layer_no+1 ] ;++i )
            {
                sum+= weight[ layer_no+1 ][ j ][ i ]*back_output[ layer_no+1 ][ i ];
            }
            back_output[ layer_no ][ j ]= sum*
                calc_f( input[layer_no][j] )*( 1.-calc_f( input[layer_no][j] ) );
        }
    }
}

```

```

        for( ii=0; ii <= layer_structure[ layer_no-1 ] ;++ii )
        {
            dweight[ layer_no ][ ii ][ j ]=
                0.25*back_output[ layer_no ][ j ]*output[ layer_no-1 ][ ii ]
                +0.9*dweight[ layer_no ][ ii ][ j ];
            weight[ layer_no ][ ii ][ j ] += dweight[ layer_no ][ ii ][ j ];
        }
    }
    return;
}

```

```

void main()
{
    int i,i2,c,tg,win_count,max_no;
    double max;

    Neuron_weight_init();

    c=0;
    tg=0;
    win_count=0;

    printf("Layer Structure is as follows : ");
    for( i=0; i<MAX_LAYER ;++i )
        printf("%d ",layer_structure[ i ]);
    printf("\n");

    printf("Followings are the contents of neurons within each layer.\n" );

    while( 1 )
    {
        for( i=0; i<layer_structure[ 0 ] ;++i ) output[ 0 ][ i ]= pattern[ tg ][ i ];
        for( i=1; i<MAX_LAYER ;++i ) Forward_propagation( tg,i );

        max= -1.0;
        max_no = -1;
        for( i=0; i<MAX_ELEMENT ;++i )
            if ( output[ MAX_LAYER-1 ][ i ]>max )
            {
                max = output[ MAX_LAYER-1 ][ i ];
                max_no = i;
            }
        if ( max_no == tg ) win_count++;

        if( c % 10==0 && 0)
        {
            printf("-----\n");
            printf("Learning iteration : %5d    target : %2d\n",c,tg+1);
            for( i=0; i<MAX_LAYER ;++i )
            {
                printf("Layer %d : ",i+1 );
                for( i2=0; i2<MAX_ELEMENT ;++i2 )
                    printf("%5.2lf ",output[ i ][ i2 ] );
                printf("\n");
            }
        }
    }
}

```

```

for( i=MAX_LAYER-1 ; i>0 ; --i ) Back_propagation( tg,i );
tg++;
if ( tg>11 )
{
    c++;
    tg=0;
    if ( win_count==MAX_ELEMENT -1 )
    {
        printf("Finished iteration at %d.\n",c);
        exit(0);
    }
    win_count=0;
}

}

return ;
}

```

5 ニューラルネットワークによる数字パターンの学習と認識実験

表 1 に示す階層構造を持つ 4 種類の階層型ニューラルネットワークを実装し、数字パターンの学習および認識実験を行う。本実験で対象とするのは、図 2 に示すような 10 種類の縦 8 ドット横 8 ドットのマトリクスパターンである。ニューラルネットワークの入力素子数は 64、出力素子数は 10 である。

表 1: 実験で用いるニューラルネットワークの素子構造

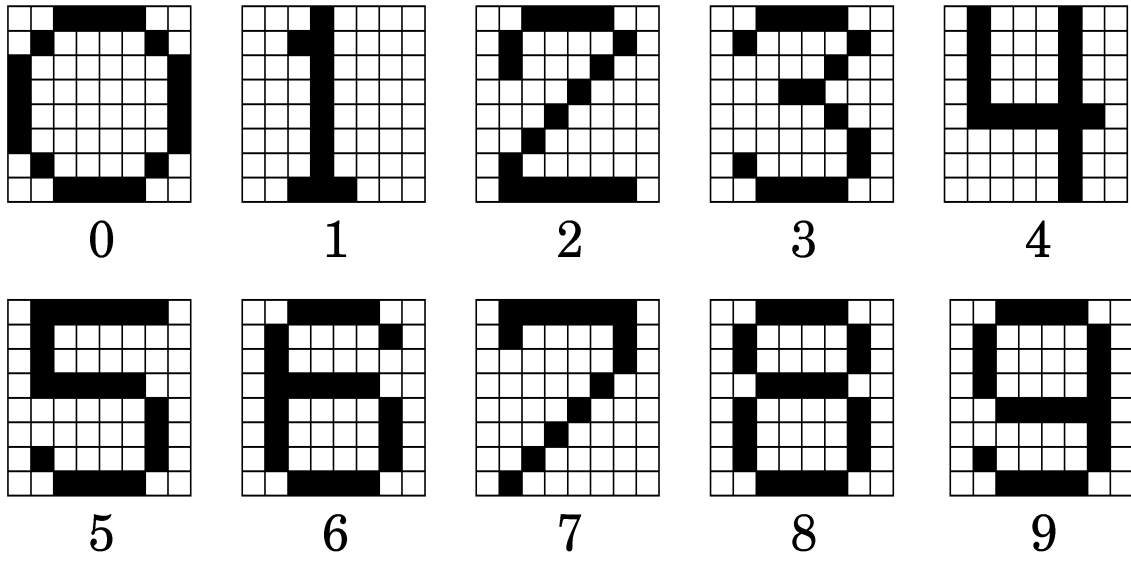
TYPE	入力層	中間層		出力層	備考
I	64	10		10	3 層ニューラルネットワーク
II	64	30		10	3 層ニューラルネットワーク
III	64	5	5	10	4 層ニューラルネットワーク
IV	64	15	15	10	4 層ニューラルネットワーク

実装したニューラルネットワークを使用して数字パターンの認識実験を行う。図 2 に学習に用いる標準パターンを示し、図 3 ～図 5 に実験で用いるテストサンプルを示す。

5.1 標準パターンのみを学習させた場合

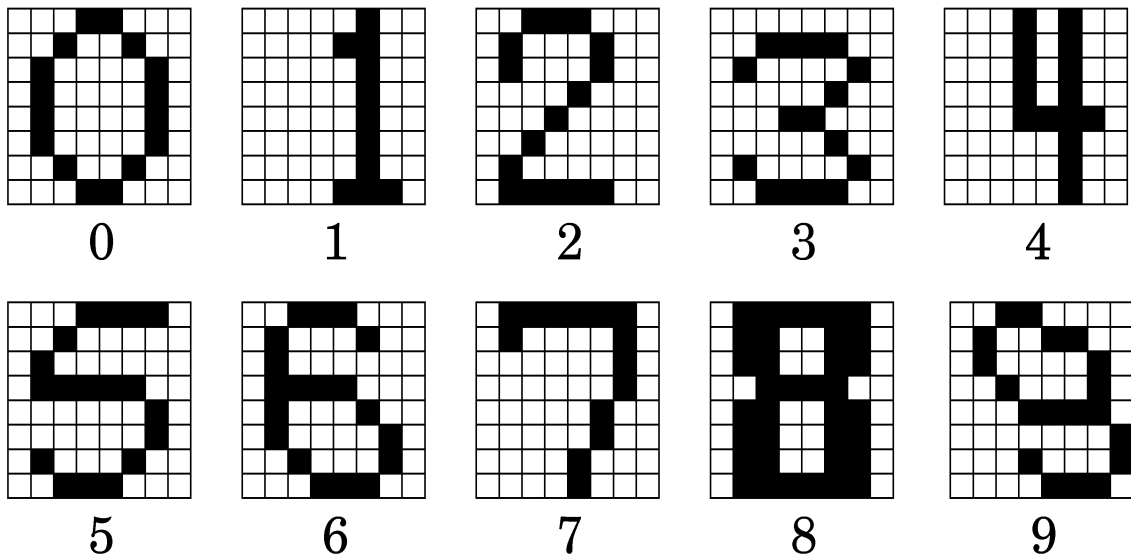
ここでは、ニューラルネットワークに標準的な数字パターンを学習させることができるかどうかを実験により調べる。実験手順は以下の通りである。

- (1) 図 2 に示す標準パターンをニューラルネットワーク (TYPE I ～ IV) に学習させる。この際に各出力素子の応答をファイルに記録する。
- (2) “0” から “9” までの標準パターンを誤りなく識別できるようになった時点で学習を打ち切り、その際の学習回数をファイルに記録する。
- (3) 図 2 に示す訓練サンプルを学習後のニューラルネットワークに入力した時の各出力素子の応答をファイルに記録する。



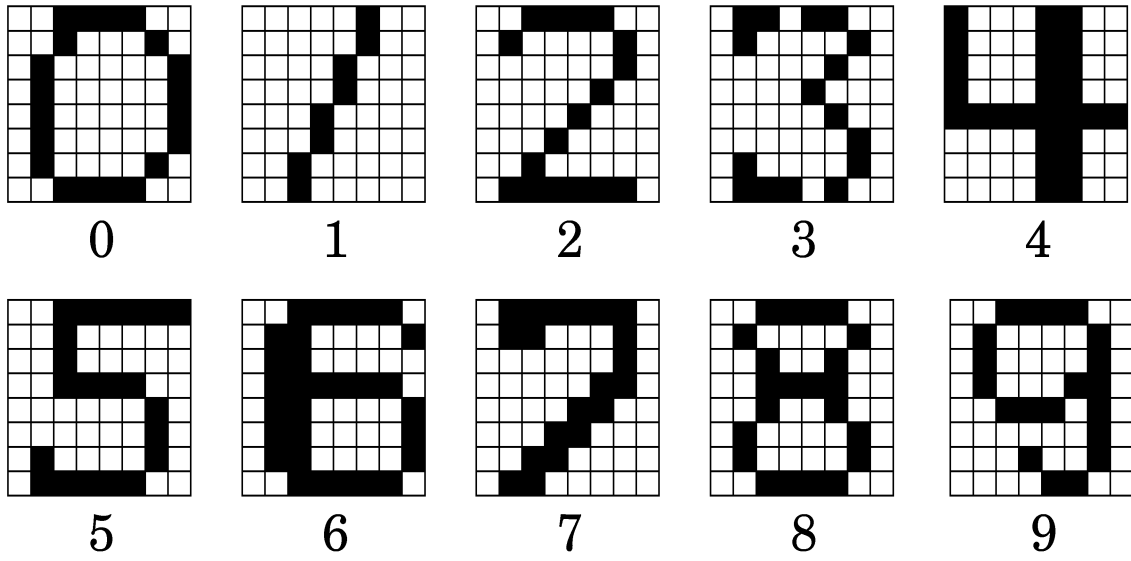
Standard pattern for digit recognition experiments (Each digit consists of 8 by 8 dots).

図 2: 学習に用いる標準パターン (訓練サンプル)



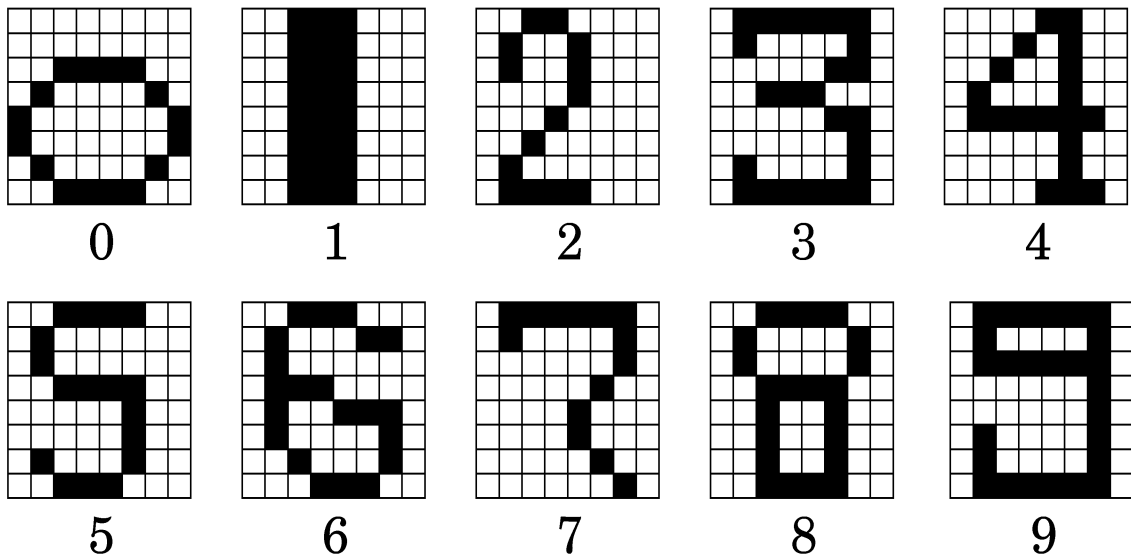
Test samples for digit recognition experiments (Each digit consists of 8 by 8 dots).

図 3: 実験に用いるテストサンプル (その1)



Test pattern for digit recognition experiments (Each digit consists of 8 by 8 dots).

図 4: 実験に用いるテストサンプル (その2)



Test pattern for digit recognition experiments (Each digit consists of 8 by 8 dots).

図 5: 実験に用いるテストサンプル (その3)

- (4) 図 3 ～図 5 に示すテストサンプルを学習後のニューラルネットワークに入力した時の各出力素子の応答をファイルに記録する。

5.2 標準パターンとテストサンプルの一部を学習させた場合

本節では、ニューラルネットワークに標準的な数字パターンのみならず類似した数字パターンを学習させることにより、汎化能力を獲得させることが可能かどうかを実験により調べる。実験手順は以下の通りである。

- (1) 図 2 に示す標準パターンと図 3 ～図 5 のテストサンプルをニューラルネットワーク (TYPE I ～ IV) に学習させる。この際に各出力素子の応答をファイルに記録する。
- (2) “0” から “9” までの標準パターンを誤りなく識別できるようになった時点で学習を打ち切り、その際の学習回数をファイルに記録する。
- (3) 図 2 に示す訓練サンプルを学習後のニューラルネットワークに入力した時の各出力素子の応答をファイルに記録する。
- (4) 図 3 ～図 5 に示すテストサンプルを学習後のニューラルネットワークに入力した時の各出力素子の応答をファイルに記録する。さらに、テストサンプルを識別させた結果から各数字パターンに対する認識率を算出する。

5.3 実験結果

実験結果を以下の要領でまとめる。

- (1) 標準パターンを学習させる際に記録した各出力素子の応答について、横軸を学習回数としてグラフにまとめる。
- (2) 訓練サンプルをニューラルネットワークに入力したときの応答をグラフにまとめる。
- (3) テストサンプルをニューラルネットワークに入力したときの応答をグラフにまとめる。

6 考察・検討

- (1) ニューラルネットワークの中間素子構造と学習回数との関係について考察せよ。また、ニューラルネットワークにおける隠れ層の役割について調べよ。
- (2) 学習係数 (ε : 学習定数, α : 安定化係数) はニューラルネットワークの認識性能にどのような影響を与えるのかを考察せよ。
- (3) ニューラルネットワークによる識別処理が、図形の平行移動・大きさ・回転などの変動に対してどのような影響を受けるかについて考察せよ。
- (4) 階層型ニューラルネットワークにおいて、階層数と識別性能の関係について考察せよ。
- (5) ニューラルネットワークの学習法について、BP 法以外の手法について調べよ。
- (6) ニューラルネットワークの具体的な応用例を挙げ、ニューラルネットワークを用いる利点と欠点について考察せよ。

参考文献

- [1] Hush,D.R. and Horne,B.G.: “Progress in supervised neural networks,” IEEE Signal Processing Magazine, pp.8-39, 1993.
- [2] Fahlman,S.E.: “An empirical study of learning speed in back-propagation networks,” Tech. Report, CMU-CS-88-162, 1988.
- [3] 舟久保 登: “パターン認識”, 共立出版, 1991.
- [4] 栗田多喜夫: “パターン認識とニューラルネットワーク”, 電子技術総合研究所ホームページ (<http://www.aist.go.jp/ETL/kurita/lecture/prnn/prnn.html>)
- [5] “FUZZY MODELLING Paradigms and Practice,” Kluwer Academic Publishers, 1996.