2018.05.15-05.24.HI

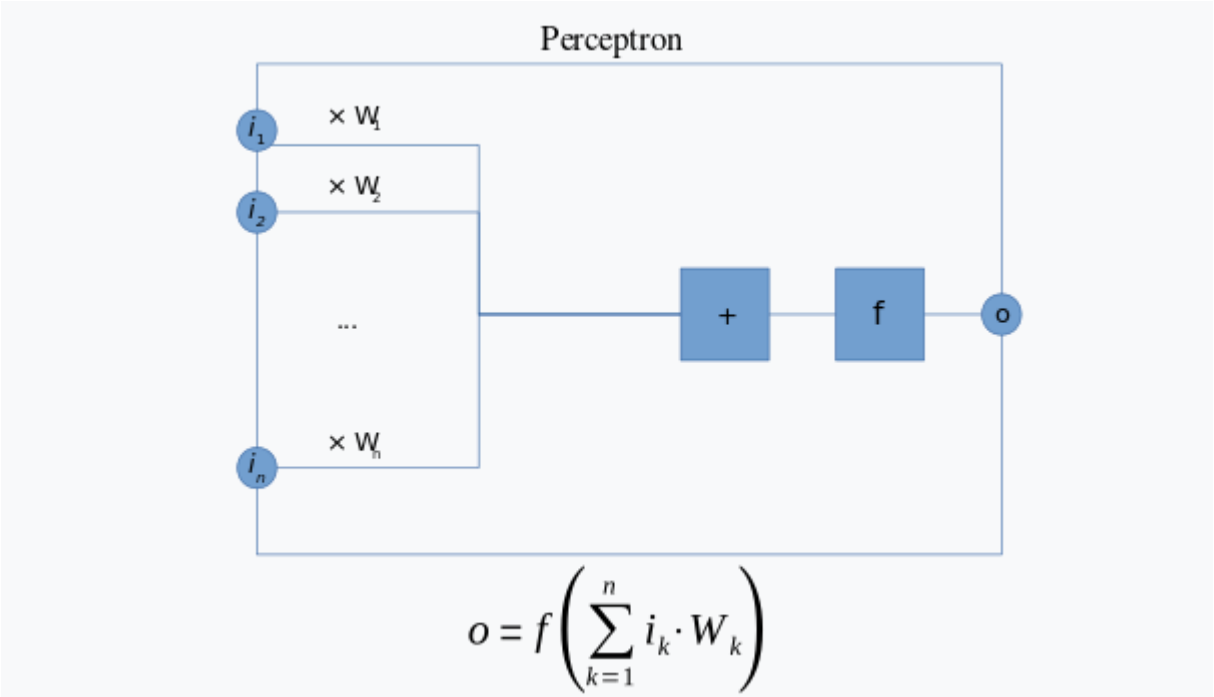
一色 浩

# バックプロパゲーション（誤差逆伝播法）

㈲数理解析研究所　一色 浩

## １．XOR問題

図1のような単純パーセプトロンでは，排他的論理和（XOR）問題は絶対に解けない．排他的論理和とは，2 つの入力が（1，0）または（0，1）のときのみ1を出力する問題である。図2左を見るとわかるとおり、XOR 問題は一本の判別直線で白マルと黒マルを分離できない、すなわち線形分離不可能な問題で ある。




$$o = f\left(\sum_{k=1}^{n} i_k \cdot W_k\right)$$

図 1 単純パーセプトロン（https://en.wikipedia.org/wiki/Perceptron）

図 1 に示されるように，単純パーセプトロンで示されるのは，1 次式であり 2 次元の場合は一本の直線である．この直線の上か下かで出力の値が異なる．これを点の判別に利用できるが，XOR の場合には直線が 2 本いることになり，一本の直線しか使えない単純パーセプトロンでは，XOR の判別はできないことになる．

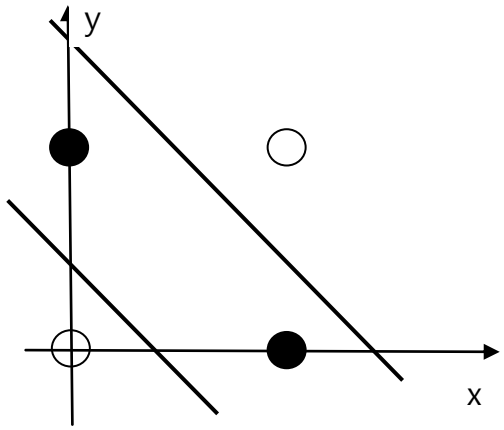



図2 線形分離不可能な問題

## 2．バックプロパゲーション

$m$層からなるニューラルネットワークを考える．第$m$層のニューロンの数を$N_m$とする．図1にニューロンのネットワークを示す.

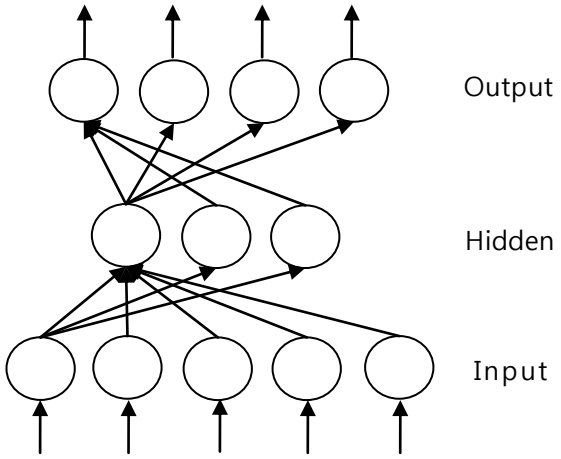



図3 ニューラルネットワーク(バイアスなし)

図3 ニューラルネットワーク（煩雑になるので，層の上下のニューロンは通常すべて結線される．また、簡単のためにバイアスを与えるニューロンも省略してある）

図2に一つのニューロンの入出力を示す．第$k$層のニューロンの数を$N_k$とし，第$k$層の番号$j$のニューロン$C_j^k$の第$k-1$層の番号$i$の出力$y_j^{k-1}$からの入力を$x_{ij}^k$，第$k$層のニューロンの出力を$y_j^k$とする．$x_{ij}^k$をすべて足し合わせたものが,

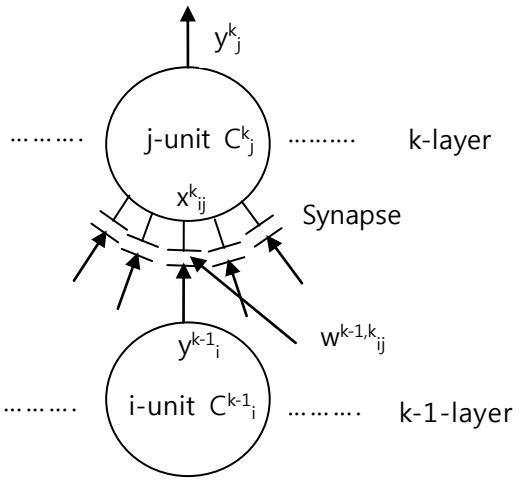



図4 ニューロンの入出力と変数

$$y_j^k = f(x_j^k) = \frac{1}{1+e^{-x_j^k}} \tag{1}$$

$$x_j^k = \sum_i^{N_{k-1}} x_{ij}^k = \sum_i^{N_{k-1}} w_{ij}^{k-1,k} y_i^{k-1} \tag{2}$$

ここで，（1）式の最右辺の関数はシグモイド関数と呼ばれる非線形関数である．第$k$層の第$j$番目のニューロン$C_j^k$の入力と第$k-1$層の第$i$番目のニューロン$C_i^{k-1}$の出力$y_i^{k-1}$を結ぶ軸索のウェイトを$w_{ij}^{k-1,k}$としている．添字の変域を以下に示す．

$$k = 2,\cdots,N_k, \quad i = 1,2,\cdots,N_{k-1}, \quad j = 1,2,\cdots,N_k \tag{ISK1}$$

出力層である第$m$層の番号$j$のニューロン$C_j^m$に与えられる教師データを$t_j$とすると，誤差$\delta_j^m$とコスト関数$E$は下式で定義される．

$$\delta_j^m = y_j^m - t_j \tag{3a}$$

$$E = \frac{1}{2}\sum_j \left(\delta_j^m\right)^2 \tag{3b}$$

コスト関数$E$は，第$k$層の第$j$番目のニューロン$C_j^k$の入力と第$k-1$層の第$i$番目のニューロン$C_i^{k-1}$の出力を結ぶ軸索のウェイトを$w_{ij}^{k-1,k}$とし，コスト関数$E$を最小にする最急降下法の算式は$\varepsilon$をパラメーターとして

$$\Delta w_{ij}^{k-1,k} = -\varepsilon \frac{\partial E}{\partial w_{ij}^{k-1,k}} \tag{4}$$

により与えられる．ウェイト $w_{ij}^{k-1,k}$ の更新式は

$$\left[ w_{ij}^{k-1,k} \right]_{\text{new}} = \left[ w_{ij}^{k-1,k} \right]_{\text{old}} + dw_{ij}^{k-1,k} \tag{ISK2}$$

となる．

(4)式の微分をチェインルールを使って書き直すと

$$\frac{\partial E}{\partial w_{ij}^{m-1,m}} = \sum_{l=1}^{N_k} \frac{\partial E}{\partial y_l^m} \frac{\partial y_l^m}{\partial w_{ij}^{m-1,m}} = \frac{\partial E}{\partial y_j^m} \frac{\partial y_j^m}{\partial w_{ij}^{m-1,m}} = \delta_j^m \frac{\partial y_j^m}{\partial w_{ij}^{m-1,m}} \tag{*5}$$

のように書き直せる．まず次の微分を考える． $y_j^m$ は $x_j^m$ のみの関数であるから

$$\frac{\partial y_j^m}{\partial w_{ij}^{m-1,m}} = \frac{\partial y_j^m}{\partial x_j^m} \frac{\partial x_j^m}{\partial w_{ij}^{m-1,m}} \tag{6}$$

ここで，出力関数がシグモイド関数の場合には，すなわち， $y = 1/(1+e^{-x})$ のとき

$$\frac{\partial y}{\partial x} = \frac{\partial}{\partial x} \frac{1}{1+e^{-x}} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} \left( 1 - \frac{1}{1+e^{-x}} \right) = y(1-y) \tag{7}$$

に注意しておく．

つぎに， $\Delta w_{ij}^{m-1,m}$ を書き直す．

$$\Delta w_{ij}^{m-1,m} = -\varepsilon \frac{\partial E}{\partial w_{ij}^{m-1,m}} \tag{8}$$

$$= -\varepsilon \sum_{l=1}^{N_k} \frac{\partial E}{\partial y_l^m} \frac{\partial y_l^m}{\partial w_{ij}^{m-1,m}} = -\varepsilon \frac{\partial E}{\partial y_j^m} \frac{\partial y_j^m}{\partial w_{ij}^{m-1,m}} = -\varepsilon \frac{\partial E}{\partial y_j^m} \frac{\partial y_j^m}{\partial x_j^m} \frac{\partial x_j^m}{\partial w_{ij}^{m-1,m}} \tag{9}$$

$$\begin{aligned} &= -\varepsilon \frac{\partial E}{\partial y_j^m} y_j^m (1 - y_j^m) \frac{\partial \sum_{k=1}^{N_{m-1}} w_{kj}^{m-1,m} y_k^{m-1}}{\partial w_{ij}^{m-1,m}} \\ &= -\varepsilon \frac{\partial E}{\partial y_j^m} y_j^m (1 - y_j^m) \sum_{k=1}^{m-1} d_k^i y_k^{m-1} = -\varepsilon (y_j^m - t_j) y_j^m (1 - y_j^m) y_i^{m-1} \end{aligned} \tag{*10}$$

$$= -\varepsilon \delta_j^m y_j^m (1 - y_j^m) y_i^{m-1} \tag{*11}$$

ここで，$d_j^i$はChroneckerのデルタで

$$d_j^i = \begin{cases} 1 & \text{when } i = j \\ 0 & \text{otherwise} \end{cases} \tag{ISK3}$$

である．

つぎに

$$\delta_j^n = \frac{\partial E}{\partial y_j^n} \tag{ISK4}$$

と書いて，第$n$層の誤差と考える．すると

$$\delta_j^n = \frac{\partial E}{\partial y_j^n} = \sum_{k=1}^{N_{n+1}} \frac{\partial E}{\partial y_k^{n+1}} \frac{\partial y_k^{n+1}}{\partial y_j^n} = \sum_{k=1}^{N_{n+1}} \frac{\partial E}{\partial y_k^{n+1}} \frac{\partial y_k^{n+1}}{\partial x_k^{n+1}} \frac{\partial x_k^{n+1}}{\partial y_j^n} \tag{12}$$

$$= \sum_{k=1}^{N_{n+1}} \frac{\partial E}{\partial y_k^{n+1}} \frac{\partial y_k^{n+1}}{\partial x_k^{n+1}} w_{jk}^{n,n+1} = \sum_{k=1}^{N_{n+1}} \frac{\partial E}{\partial y_k^{n+1}} y_k^{n+1} (1 - y_k^{n+1}) w_{jk}^{n,n+1} \tag{*13}$$

$$= \sum_{k=1}^{N_{n+1}} \delta_k^{n+1} y_k^{n+1} (1 - y_k^{n+1}) w_{jk}^{n,n+1} \tag{14}$$

が得られる．ウェイトの修正量$\Delta w_{ij}^{k-1,k}$については，(11)式と同様にして

$$\Delta w_{ij}^{k-1,k} = -\varepsilon \frac{\partial E}{\partial w_{ij}^{k-1,k}} = -\varepsilon \delta_j^k y_j^k (1 - y_j^k) y_i^{k-1} \tag{15}$$

となる．

以上をまとめると，(3a)式，(14)，(15)式から

$$\delta_j^k = \begin{cases} y_j^k - t_j & \text{if } k = m \\ \sum_{l=1}^{N_{k+1}} \delta_l^{k+1} y_l^{k+1} (1 - y_l^{k+1}) w_{jl}^{k,k+1} & \text{otherwise} \end{cases} \tag{*16}$$

$$\Delta w_{ij}^{k-1,k} = -\varepsilon \delta_j^k y_j^k (1 - y_j^k) y_i^{k-1} \tag{ISK5}$$

というアルゴリズムが得られる．

(2)式と(16)式を比べると，信号$y_j^k$の伝播は入力層から出力層に，すなわち

$k=1\to m$ に起こる．一方，$\delta_j^k$ 誤差はこれと逆方向に起こることが分かる．これが誤差逆伝播と呼ばれる所以である．

## ３．MatLabによるコード

図4には常にバイアス1を出力するニューロンが追加されている．

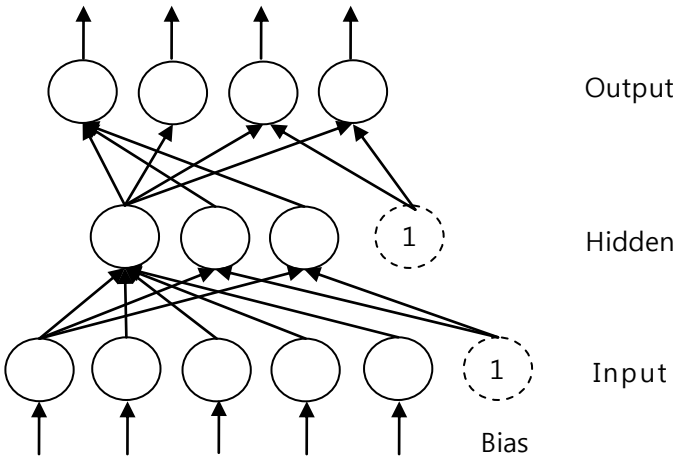



図4 ニューラルネットワーク(バイアスあり)

しかし、下記のコードでは，簡単のためバイアスを与えるニューロンは入っていない．詳しくコメントを付けてあるので，あへて説明をしない．具体的な問題としては２数の積を学習する．しかし，一般的に書いてあるので，コードを少し書き直せば，他の問題へ適用できる．

```
%{
// ------------------------------------------------------------- //
//                                                               //
//    File Name: BP3.m                          2018.05.16-2018.05.24 //
//                                                               //
//    Backward Propagation                                       //
//    y = x1 * x2                                                //
//                                                               //
// ------------------------------------------------------------- //
%}

function BP3()

    clear;

    %%%% preparation
```

```
    %% Output file
    fp_out = fopen('BP3_out.csv', 'w');

    %% Input Data

    Ninp = input("Input Ninp (eg. 2) = ")               % No. of neurons in input la
yer
    Nhid = input("Input Nhid (eg. 3) = ")               % No. of neurons in hidden l
ayer
    Nout = input("Input Nout (eg. 1) = ")               % No. of neurons in output l
ayer
    Ntrn = input("Input Ntrn (eg. 100) = ")             % No. of training data
    Nstd = input("Input Nstd (eg. 500) = ")             % No. of training or study
    eps = input("Input eps (eg. 0.2) = ")               % step in steepest descent m
ethod
    val = input("Input val (seed of rand ... eg. 1) = ") % value of seed of random nu
mber
    fprintf("₩n");

    fprintf("Ninp = %d₩n", Ninp);
    fprintf("Nhid = %d₩n", Nhid);
    fprintf("Nout = %d₩n", Nout);
    fprintf("Ntrn = %d₩n", Ntrn);
    fprintf("Nstd = %d₩n", Nstd);
    fprintf("eps = %12.6f₩n", eps);
    fprintf("val = %12.6f₩n", val);
    fprintf("₩n");

    fprintf(fp_out, "Ninp =, %d₩n", Ninp);
    fprintf(fp_out, "Nhid =, %d₩n", Nhid);
    fprintf(fp_out, "Nout =, %d₩n", Nout);
    fprintf(fp_out, "Ntrn =, %d₩n", Ntrn);
    fprintf(fp_out, "Nstd =, %d₩n", Nstd);
    fprintf(fp_out, "eps =, %12.6f₩n", eps);
    fprintf(fp_out, "val =, %12.6f₩n", val);
    fprintf(fp_out, "₩n");

    rand ("seed", val);                         % value of seed of random number

    %% input and output vectors

    xinp = zeros(Ninp,1);                       % input to neurons in input layer
    yinp = zeros(Ninp,1);                       % output from neurons in input layer
    xhid = zeros(Nhid,1);                       % input to neurons in hidden layer
    yhid = zeros(Nhid,1);                       % output from neurons in input layer
    xout = zeros(Nout,1);                       % input to neurons in output layer
    yout = zeros(Nout,1);                       % output from neurons in output layer

    %% weight matrices
```

```
    wIH = zeros(Ninp,Nhid);                    % weight between input and hidden neur
ons
    wHO = zeros(Nhid,Nout);                    % weight between hidden and output neu
rons
    dwIH = zeros(Ninp,Nhid);                   % increment of wIH
    dwHO = zeros(Nhid,Nout);                   % increment of wHO

    %% initial value of weights

    for i = 1: Ninp
        for j = 1: Nhid
            wIH(i,j)=0.1*rand();               % initial value of wIH
        end
    end

    for i = 1: Nhid
        for j = 1: Nout
            wHO(i,j) = 0.1*rand();             % initial value of wHO
        end
    end

    %% print out of initial value of weights

    fprintf(fp_out, "initial value of weight₩n")
    fprintf(fp_out, "i, j, wIH₩n")
    for i = 1: Ninp
        for j = 1: Nhid
            fprintf(fp_out, "%d, %d, %12.6f₩n", i, j, wIH(i,j))
        end
    end
    fprintf(fp_out, "₩n")

    fprintf(fp_out, "i, j, wHO₩n")
    for i = 1: Nhid
        for j = 1: Nout
            fprintf(fp_out, "%d, %d, %12.6f₩n", i, j, wHO(i,j))
        end
    end
    fprintf(fp_out, "₩n")

    %% train data

    xinpTD = zeros(Ntrn,Ninp);                 % No. of input data for training
    tTD = zeros(Ntrn,Nout);                    % No. of teach data for training

    % generation of training data
    %%%%%%%%%%%%%%%%%%%%%%
    % input data
    for p = 1: Ntrn
```

```
    for (i = 1: Ninp)
        xinpTD(p,i) = rand();                   % random number between 0 and 1
    end
end

% teach data
for p = 1: Ntrn
    for (i = 1: Nout)
        tTD(p,i) = xinpTD(p,1)*xinpTD(p,2);    % y = x1 * x2
    end
end
%%%%%%%%%%%%%%%%%%%%%

%% print out of train data

fprintf(fp_out, "train data¥n")
fprintf(fp_out, "input data¥n")
fprintf(fp_out, "p, ")
for i = 1: Ninp
    fprintf(fp_out, "i = %d, ", i)
end
fprintf(fp_out, "¥n")
for p = 1: Ntrn
    fprintf(fp_out, "%d, ", p)
    for i = 1: Ninp
        fprintf(fp_out, "%d, ", xinpTD(p,i))
    end
    fprintf(fp_out, "¥n")
end
fprintf(fp_out, "¥n")

fprintf(fp_out, "teach data¥n")
fprintf(fp_out, "p, ")
for i = 1: Nout
    fprintf(fp_out, "i = %d, ", i)
end
fprintf(fp_out, "¥n")
for p = 1: Ntrn
    fprintf(fp_out, "%d, ", p)
    for i = 1: Nout
        fprintf(fp_out, "%d, ", tTD(p,i))
    end
    fprintf(fp_out, "¥n")
end
fprintf(fp_out, "¥n")


%%%% training

dltO = zeros(Nout,1);
```

```
dltH = zeros(Nhid,1);
dltI = zeros(Ninp,1);

fprintf(fp_out, "stdy, err_trnWn");
xstd = zeros(Nstd/10,1);
yerr = zeros(Nstd/10,1);
for istd = 1: Nstd
    itmp = mod(istd,10);
    if (itmp == 0)
        itmp1 = floor(istd/10);
        xstd(itmp1) = istd;
    end
    esum = 0.0;
    for p = 1: Ntrn
        for i = 1: Ninp
            xinp(i) = xinpTD(p,i);
        end

        for i = 1: Ninp
            yinp(i) = xinp(i);
        end

        xhid = wIH' * yinp;

        yhid = fout(xhid);    % colum vector is transfered to row vecter
        yhid = yhid';         % return to column vector

        xout = wHO' * yhid;
        yout = fout(xout);    % colum vector is transfered to row vecter
        yout = yout';         % return to column vector

        %% backward error propagation

        % output layer
        for j = 1: Nout
            dltO(j) = yout(j) - tTD(p,j);
        end

        % hidden layer
        for j = 1: Nhid
            dltH(j) = 0;
            for l = 1: Nout
                dltH(j) = dltH(j) + dltO(l)*yout(l)*(1-yout(l))*wHO(j,l);
            end
        end

        %% correction of weight

        % correction of weight weight wHO
        for i = 1: Nhid
```

```
            for j = 1: Nout
                dwHO(i,j) = -eps*dltO(j)*yout(j)*(1-yout(j))*yhid(i);
                wHO(i,j) = wHO(i,j) + dwHO(i,j);
            end
        end

        % correction of weight weight wIH
        for i = 1: Ninp
            for j = 1: Nhid
                dwIH(i,j) = -eps*dltH(j)*yhid(j)*(1-yhid(j))*yinp(i);
                wIH(i,j) = wIH(i,j) + dwIH(i,j);
            end
        end

        esum = esum  + dltO(1)*dltO(1);    % sumof errors
    end

    err_trn = sqrt(1/Ntrn*esum);               % root mean square error
    if (itmp == 0)
        yerr(itmp1) = err_trn;
        fprintf("%d, %12.6f¥n", istd, err_trn);
        fprintf(fp_out, "%d, %12.6f¥n", istd, err_trn);
    end
end
fprintf("¥n")
fprintf(fp_out, "¥n")

%% plotting of error
plot(xstd', yerr');    % error plotong
xlabel('Repetition of Study'); ylabel('Root Mean Square Error');

fprintf("err = %12.6f¥n", err_trn)
fprintf(fp_out, "err = %12.6f¥n", err_trn)
fprintf("¥n")
fprintf(fp_out, "¥n")


%%%% testing

%%% testing 1 ... testing of 1000 data

esum = 0;
fprintf(fp_out, "ii, x1, x2, y, y_exct¥n")
for ii = 1: 1000

    % generation of test data
    xinp(1) = rand();    % input of x1
    xinp(2) = rand();    % input of x2

    % input layer
```

```
        for i = 1: Ninp
            yinp(i) = xinp(i);
        end

        % hidden layer
        xhid = wIH' * yinp;
        yhid = fout(xhid);
        yhid = yhid';

        % output layer
        xout = wHO' * yhid;
        yout = fout(xout);
        yout =yout';        % estimation of x1 and x2

        esum = esum  + (yout-xinp(1)*xinp(2))*(yout-xinp(1)*xinp(2));    % sumof erro
rs

        % print out of result
        fprintf("%d, %12.6f * %12.6f = %12.6f₩n", ii, xinp(1), xinp(2), yout(1))
        fprintf(fp_out, "%d, %12.6f, %12.6f, %12.6f, %12.6f₩n",
                ii, xinp(1), xinp(2), yout(1), xinp(1)*xinp(2))
    end
    fprintf(fp_out, "₩n")

    err_test = sqrt(1/1000*esum);               % root mean square error

    fprintf("testing 1 ... 1000 input data₩n")
    fprintf(fp_out, "testing 1 ... 1000 input data₩n")
    fprintf("err_test = %12.6f₩n", err_test)
    fprintf(fp_out, "err_test = %12.6f ... 1000 data₩n", err_test)
    fprintf("₩n")
    fprintf(fp_out, "₩n")


    %%% testing 2 ... input from keyboard

    fprintf("testing 2 ... input from keyboard₩n")
    for ii = 1: 1000

        % input of test data from keyboard
        xinp(1) = input("Input x1 (eg. 0.2 ... 1000 to quit) = ");    % input of x1
        if (xinp(1) >= 1000)    % breake of keyboard input
            break
        end
        xinp(2) = input("Input x2 (eg. 0.8) = ");    % input of x2

        % input layer
        for i = 1: Ninp
            yinp(i) = xinp(i);
        end
```

```
        % hidden layer
        xhid = wIH' * yinp;
        yhid = fout(xhid);
        yhid = yhid';

        % output layer
        xout = wHO' * yhid;
        yout = fout(xout);
        yout =yout';        % estimation of x1 and x2

        % print out of result
        fprintf("%12.6f * %12.6f = %12.6f¥n", xinp(1), xinp(2), yout(1))
        fprintf(fp_out, "%12.6f * %12.6f = %12.6f¥n", xinp(1), xinp(2), yout(1))
    end

    fclose(fp_out);

endfunction

%% ------------------------------------------------------------------ %%

function retval = fout(x)
    % Sigmoid function
    retval = 1/(1+exp(-x));
endfunction

%% ------------------------------------------------------------------ %%
```

## ４．計算例

計算の一例を図5に示す．計算に用いたパラメータを表1に示す．図5(b)を見ると随分精度が悪いようであるが，ディープラーニングを取り入れる以前のニューロの実力はこんなものであった．

表1 計算パラメータ

| 項目 | 記号 | 値 |
|---|---|---|
| 入力層のニューロン | Ninp | 2 |
| 隠れ層のニューロン | Nhid | 3 |
| 出力層のニューロン | Nout | 1 |
| 教師パターン数 | Ntrn | 100 |
| 学習回数 | Nstd | 1000 |
| 山下りのステップ | eps | 0.2 |
| 乱数の種 | val | 1 |

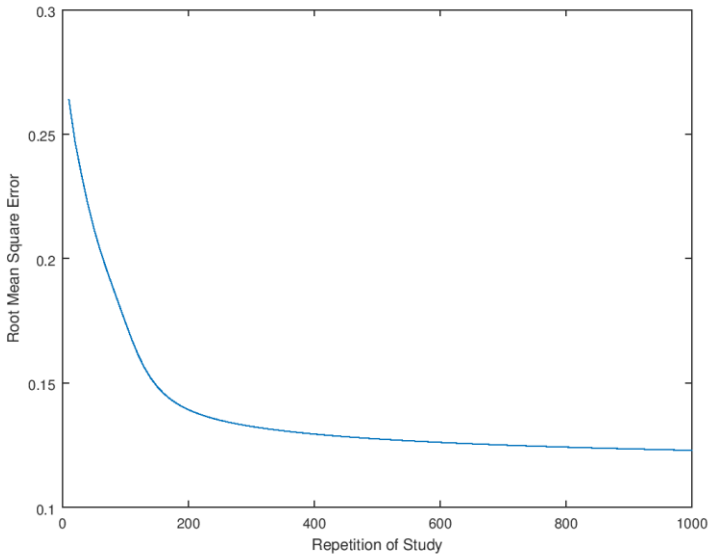
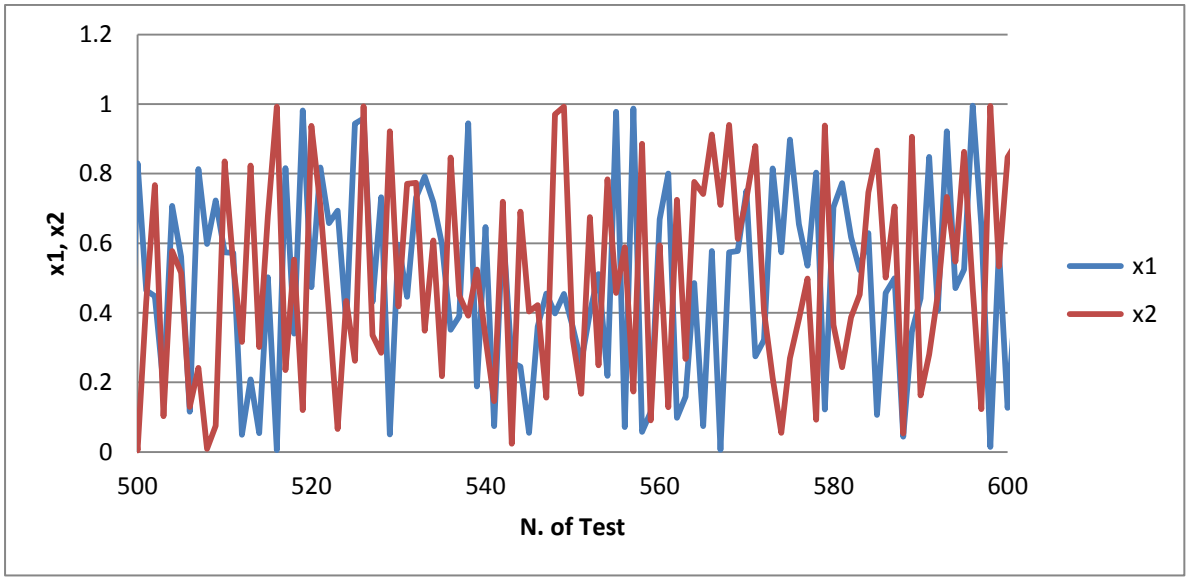
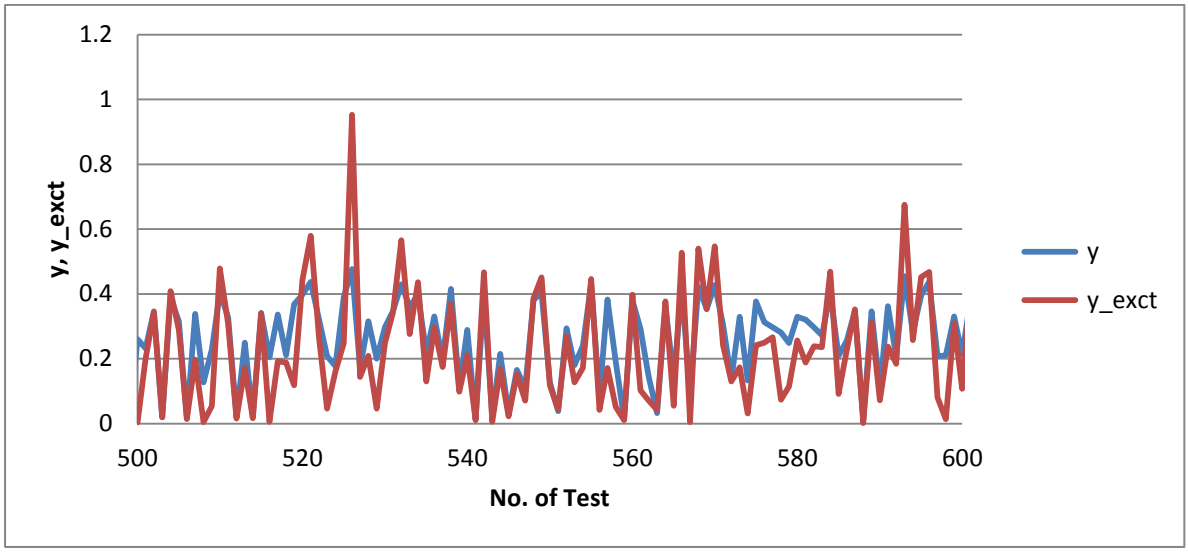



図6 学習曲線




(a) $x_1$ and $x_2$




(b) $y = x_1 * x_2$

図6 計算結果

## ５．結言

教師付き学習ニューラルネットワークの基礎である誤差逆伝播法(エラーバックプロパゲーション法)を解説した．深層学習に用いられている最新の手法の理解に，本解説が役立てば幸いである．

参考文献

[1] 浅川伸一，バックプロパゲーション，pp.1-20,

http://www.cis.twcu.ac.jp/~asakawa/waseda2002/bp.pdf