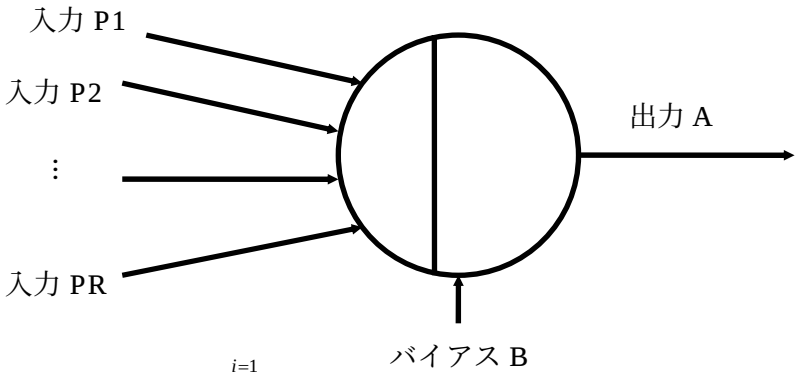
Artificial Neural Net Work とは、

ニューロ研究の目的は、人間に近い能力を持つ情報処理システムを実現することにある。そのため、ニューロの手本として人間などの脳の情報処理のメカニズムをもとに様々な情報処理システムが提案されてきている。この研究さきがけとして、1953 年に提案されたパーセプトロンが有名である。パーセプトロンによる応用例としてパターン認識装置などあり、その能力の可能性が注目されたが、1969 年のミンスキー、パパートらによりパーセプトロンの能力の限界が指摘されその後、研究は下火になった。しかし、1980 年代には、パーセプトロンの欠点を解決した多重ニューラルネットワークモデル、ニューラルネットワークモデルと物理学で磁気的性質を説明するスピンのモデルの類似性から考えられたホップフィールド型ネットワーク、それに学習効果とスピンモデルの温度低下を対応させたボルツマンマシンなど様々なタイプのネットワークと処理アルゴリズムが提案された。1990 年代では、生物の進化の過程をモデルに考慮に入れた学習アルゴリズムとして、GA（Genetic Algorithm）や Heuristic Search アルゴリズムやファジィロジックを応用したモデル、Radial Based ニューラルネットワークなど様々なものが提案されおり現在も進行中である。ここでは、特に、ある入力と出力が与えられたときの関係関数を求めるような目的に用いられるネットワークアルゴリズム。具体的にはパーセプトロン、Widrow-Hoff のモデル、バックプロパゲーションの３種類について例題として、AND 論理の学習を取り上げ説明を行う。

ニューロンモデル（Unit）

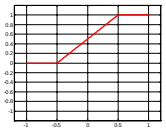
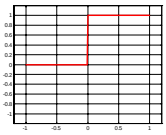
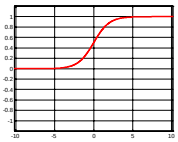
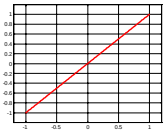
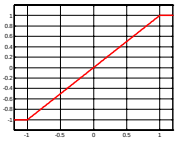
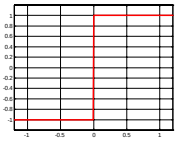
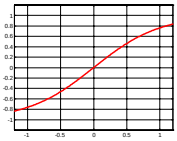
ニューラルネットワークの基本となるニューロンの人工的なモデルをユニット（Unit）とよぶ。一般的にニューロンユニットは、以下の様な構造を持つ。



$$A = NeuralUnit(\sum_{R}^{i=1} P_i + B)$$

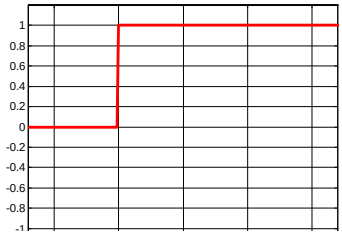
多入力１出力関数で表され、Neural Unit として、これまで様々な人工的なモデルが提案されている。ニューラルネットワークでは、この単純化されたニューロンユニットがいくつ

も組合わさり構成されている。組み合わせ方は、単層型、階層型、星型など様々である。

	satlin	hardlim	logsig
			
purelin	satlins	hardlims	**tansig**
			
リニア近似をしたい場合	リニア近似に非線形要素を入れたい場合	パーセプトロン	バックプロパゲーションなど

ここでのニューロンユニットは、MATLAB の Neural Network Toolbox の定義に準じて説明を行う。MATLAB に実装されているニューロンモデル(UNIT)は、以下のものが定義されている。ここで、_____,_____,_____ の出力はいづれも０～１の間であるのに対し、_____,satlins,_____, tansig は、-1～1 の範囲で変化することができる。また、下の表では、０を中心にしたグラフを表示したが、ｘ軸に方向（左右方向）に対しバイアスを加えることで出力のスレッショルドを変更することができる。（スレッショルドを変更することで入力値により変化する出力が 1 になるタイミングを変更することが出来る。）例えば、以下のように hardlim(input,b)にバイアス b=0.5 を加えると入力が　-0.5 より大きくなった場合、出力が 1 になる。ここでのバイアスは、直感と符号が異なるので注意する。

```
plot(-1.2:0.01:1.2,hardlim(-1.2:0.01:1.2,0.5),'r','linewidth',3);axis([-1.2 1.2 -1.2 1.2]);grid
```

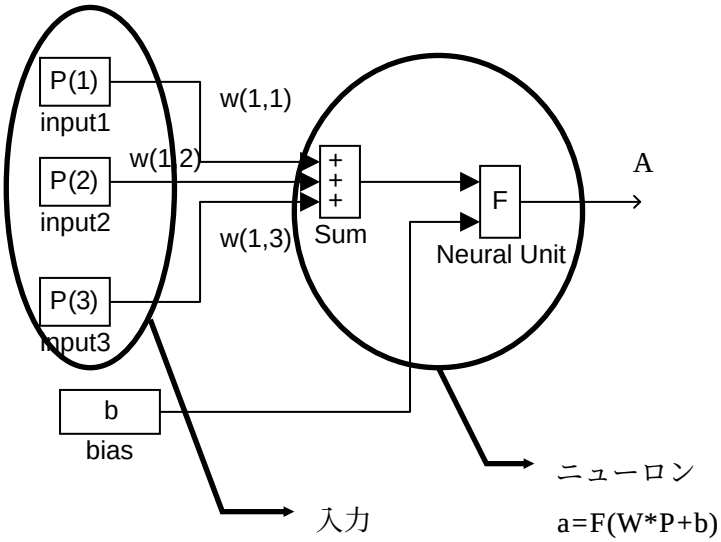


同様に、他のニューロンユニットに対してもバイアスをかけることができる。

多入力１出力ニューロンユニット

ニューラルネットワークに用いるニューロンは、一般的に、多入力１出力の関数として使われる。一般的に、それぞれの入力には、重み係数がありそれらの合計がニューロンに入

りその結果として出力がでる。



Neural Unit は行列表記を活用することでコンパクトでかつわかりやすい表記で定義できる。まず、入力 P、重み係数 W を以下のように定義すると

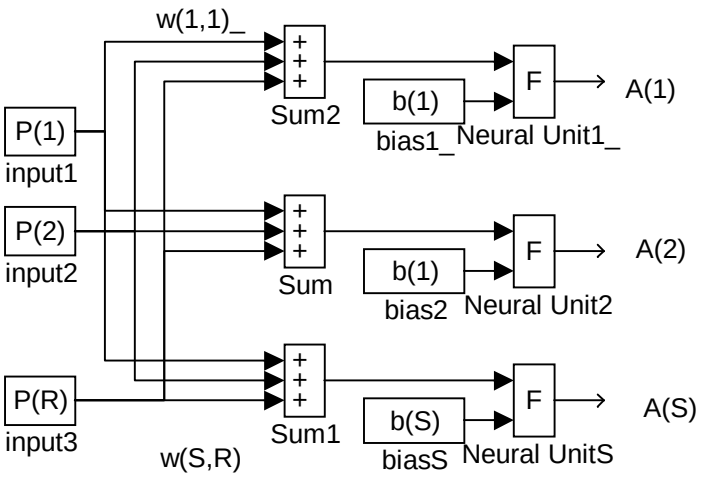
$$\mathbf{P}=\begin{bmatrix}P(1)\\P(2)\\P(3)\end{bmatrix}、\quad \mathbf{W}=\begin{bmatrix}W(1,1) & W(1,2) & W(1,3)\end{bmatrix}$$

$$A=F\left(\begin{bmatrix}W(1,1) & W(1,2) & W(1,3)\end{bmatrix}\cdot\begin{bmatrix}P(1)\\P(2)\\P(3)\end{bmatrix}+b\right)$$

$$A=F\left(\mathbf{W}\cdot\mathbf{P}+b\right)$$

多入力多出力ニューロンユニット

また、この表記を若干変更するだけで多入力多出力ニューロンも定義することができる。



上の図のように入力の数を R、出力の数を S とすると

$$\mathbf{P}=\begin{bmatrix}P(1)\\P(2)\\\vdots\\P(R)\end{bmatrix},\ \mathbf{W}=\begin{bmatrix}W_{11}&W_{12}&\cdots&W_{1R}\\W_{21}&W_{22}&\cdots&W_{2R}\\\vdots&\vdots&\ddots&\vdots\\W_{S1}&W_{S2}&\cdots&W_{SR}\end{bmatrix}、\ \mathbf{b}=\begin{bmatrix}b(1)\\b(2)\\\vdots\\b(S)\end{bmatrix}$$

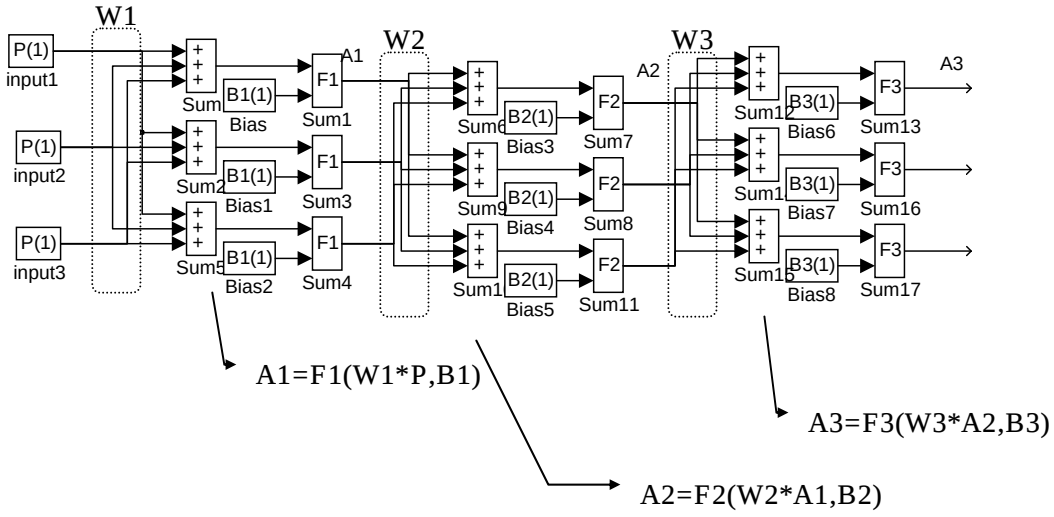
$$\begin{bmatrix}A(1)\\A(2)\\\vdots\\A(S)\end{bmatrix}=F\left(\begin{bmatrix}W_{11}&W_{12}&\cdots&W_{1R}\\W_{21}&W_{22}&\cdots&W_{2R}\\\vdots&\vdots&\ddots&\vdots\\W_{S1}&W_{S2}&\cdots&W_{SR}\end{bmatrix}\cdot\begin{bmatrix}P(1)\\P(2)\\\vdots\\P(R)\end{bmatrix}+\begin{bmatrix}b(1)\\b(2)\\\vdots\\b(S)\end{bmatrix}\right)$$

$$A=F\left(\mathbf{W}\cdot\mathbf{P}+\mathbf{b}\right)$$

と１出力の場合と同じ表記になる。

多層ニューラルネットワーク

ニューラルネットワークは、いくつかの層を持つことができる。それぞれの層は、重み行列 W とバイアス行列 b、入力ベクトル P、出力ベクトル A を持つ。それぞれの層のニューロンユニットを区別するため、それぞれの行列に添え数字を付けることで区別をする。以下に３層ニューラルネットワークの例を示す。



A3=F3(__*__(W2*F1(W1*P,B1),__),B3)

と、このように表現できる。同様にさらに多層のニューラルネットも容易に構成可能である。

多入力ベクトルの行列表現

通常、入力と出力はベクトルで表されるが、MATLAB、Neural Net Toolbox では、行列の機能を活用し、複数の入力ベクトルと出力ベクトルを行列で表現することで一括して処理をすることができる。例えば、

$$\text{入力}\ \mathbf{P}(:,1)=\begin{bmatrix}P(1,1)\\P(2,1)\\\vdots\\P(R,1)\end{bmatrix}\ \text{と出力}\ \mathbf{A}(:,1)=\begin{bmatrix}A(1,1)\\A(2,1)\\\vdots\\A(S,1)\end{bmatrix}$$

$$\text{入力}\ \mathbf{P}(:,2)=\begin{bmatrix}P(1,2)\\P(2,2)\\\vdots\\P(R,2)\end{bmatrix}\ \text{と出力}\ \mathbf{A}(:,2)=\begin{bmatrix}A(1,2)\\A(2,2)\\\vdots\\A(S,2)\end{bmatrix}$$

. . .

$$\text{入力}\ \mathbf{P}(:,Q)=\begin{bmatrix}P(1,Q)\\P(2,Q)\\\vdots\\P(R,Q)\end{bmatrix}\ \text{と出力}\ \mathbf{A}(:,Q)=\begin{bmatrix}A(1,Q)\\A(2,Q)\\\vdots\\A(S,Q)\end{bmatrix}\ \text{のパターンの組み合わせを行列で表}$$

表現し直すと

入力 $\mathbf{P}=\begin{bmatrix} P(1,1) & P(1,2) & \cdots & P(1,Q) \\ P(2,1) & P(2,2) & \cdots & P(2,Q) \\ \vdots & \vdots & \ddots & \vdots \\ P(R,1) & P(R,2) & \cdots & P(R,Q) \end{bmatrix}$ と出力 $\mathbf{A}=\begin{bmatrix} A(1,1) & A(1,2) & \cdots & A(1,Q) \\ A(2,1) & A(2,2) & \cdots & A(2,Q) \\ \vdots & \vdots & \ddots & \vdots \\ A(S,1) & A(S,2) & \cdots & A(S,Q) \end{bmatrix}$、

を用い表すためには、

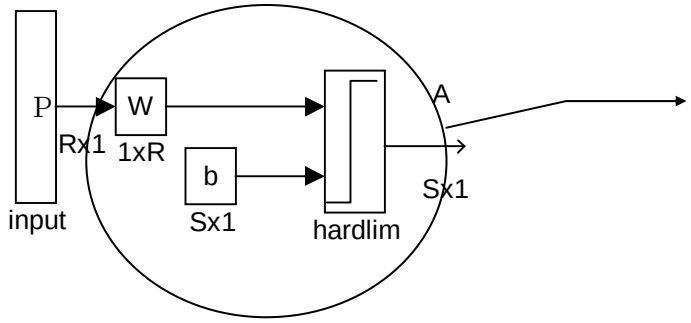
$\mathbf{A}=F(\mathbf{W}\cdot\mathbf{P}+\mathbf{b})$ の形式で表現したいわけであるが、バイアス行列 b のサイズが異なるため、うまく行かない。そこで Neural Network Toolbox では、＋演算の代わりに新たにバイアスパラメータを入力できるように Neural Unit 関数を拡張してある。

$A=F(\mathbf{W}\cdot\mathbf{P},\mathbf{b})$

このように行列表現をフルに利用することで入出力パターンを一括して表現することができる。

パーセプトロンの学習アルゴリズム

パーセプトロンとは、**1957 年に** Rosenblatt が提案した Artificial Neural Network の一つである。もっとも単純なパーセプトロンは、1 層の Neuron で構成され Neuron の____係数と_____係数を調整することで学習させる。パーセプトロンのニューロンのタイプは、_____タイプのニューロンユニットを用いる。



hardlim ニューロンは、２つの異なる信号のクラス分けをするような目的に便利である。

それでは、ディジタル回路の単純な問題 AND 回路を例にパーセプトロンの学習の手順を説明する。AND 回路の入力と出力の関係は、

P(1)	P(2)	A
1	1	1
1	0	0
0	1	0
0	0	0

となっている。まず入力行列 P と目標出力行列 T(Target)を以下の様に定義する。

```
>> P=[0 0;0 1;1 0;1 1]'
```

```
P =
     0     0     1     1
     0     1     0     1
>> T=[0 0 0 1]
T=
     0     0     0     1
```

ここでは、P(:,1)が T(1)に P(:,2)が T(2)に対応している。
次に重み係数をランダムに代入する。Neural Network Toolbox では、rands という関数が用意されている。ここでの入力数 R は 2 で出力数 S は 1 なので

```
R=2;S=1;
 [W,B]=rands(S,R)
W =
    -0.9846    -0.2332
B =
    -0.8663
```

となるここで与えられた重み係数 W とバイアス係数 B の値は、ランダムなので実際上の値とは異なる。
学習アルゴリズムは以下の通りである。

```
ニューロンユニット hardlim により A 行列を値を求める。
A=hardlim(W*P,B);
learnp 関数によりターゲット行列との偏差を求める。
[dW dB]=learnp(P,A,T);
```

learnp 関数の中では以下の計算を行っている。

```
E=T-A      %hardlim で計算されたターゲット T から A を差し引き誤差を計算する。
dW = E*P'; % 差し引き誤差に P の転値行列を掛けて W の誤差成分 dW を計算する。
dB = E;   %差し引き誤差 E が直接 B の誤差成分 dB となる。
```

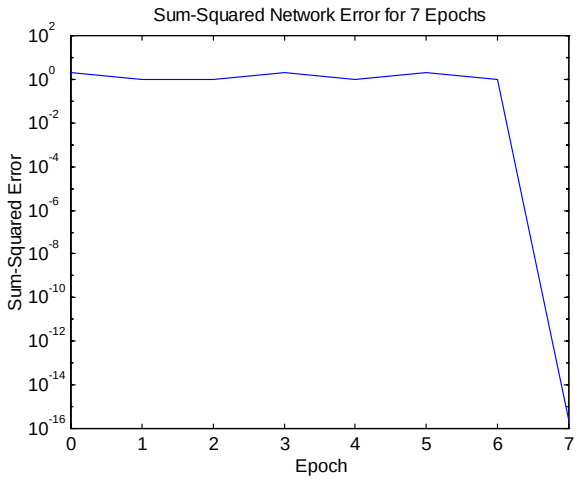
誤差成分 dW,dB を基に W,B を更新する。

```
W=W+dW
B=B+dB
```

あとは、以上の手順を誤差 E が十分小さくなるまで計算を繰り返す。
ちなみに Neural Network Toolbox では、これらの一連の学習計算をする関数 M ファイルとして trainp 関数を用意している。

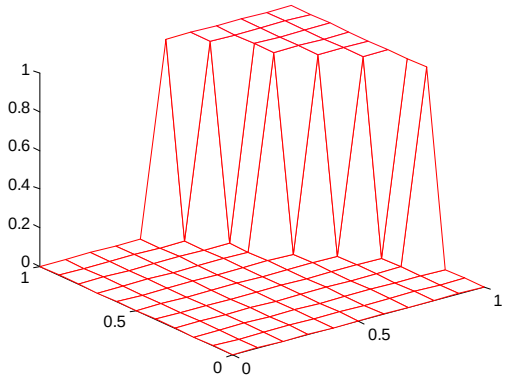
TP=[disp_freq max_epoch];
[W,B,epochs]=trainp(W,B,P,T,TP)
ここで定義している変数 disp_freq は、途中の計算結果を表示する周期である。max_epoch は、最大繰り返し演算回数である。
以上の手順をまとめると以下のようなスクリプトで書くことができる。

```
P=[0 0;0 1;1 0;1 1]';
T=[0 0 0 1];
R=2;S=1;
[W,B]=rands(S,R)
disp_freq=1;
max_epoch=100;
TP=[disp_freq max_epoch];
[W,B,epochs]=trainp(W,B,P,T,TP)
```



ここでは、7 回の繰り返しで収束している。以下に入力 P(1),P(2)をそれぞれ x,y 軸に出力 A を z 軸にとった 3 次元プロットを示す。座標(1,1)の時 z の値も 1 になっているのかわかる。

```
for PP1=0:0.1:1
for PP2=0:0.1:1
PPP(PP1*10+1,PP2*10+1)=hardlim(W*[PP1;PP2],B);
end;
end
mesh(0:0.1:1,0:0.1:1,PPP)
```



パーセプトロンは、シンプルでかつ直感的に分かりやすいアルゴリズムである。しかしパーセプトロンにはいくつかの制限がある。ニューロンユニットとして Hardlim を用いたパーセプトロンの出力は、0 か 1 の２つのクラス分けにしか用いることができない。計算でチューニングするパラメータは、重み係数とバイアス係数だけであるためパーセプトロンは、線形分離可能な入出力関係しか扱えないなどの問題点がある。

Widrow-Hoff 学習アルゴリズム

Widrow-Hoff のアルゴリズムは、基本的に、パーセプトロンのネットワーク構造とほぼ同じである。違いは、パーセプトロンは、非線形関数である Hardlim 関数を用いたのに対し、Widrow-Hoff では、_____関数を用いている。そのため出力の値は、0,1 だけでなくその間の数も表現可能になる。重み係数 W とバイアス係数 B の計算には最小自乗法を用い出力の誤差の最小にするように調整する。この方法もパーセプトロンと同じく重み係数 W とバイアス係数 B のみの調整であるため線形分離可能な入出力関係しか扱うことはできない。

Widrow-Hofff の学習アルゴリズムでは、パーセプトロンとは、異なりいくつかのパラメータを追加する必要がある。パーセプトロンの場合、出力は 1 または 0 であったため最終出力誤差が０になるまで繰り返し演算をしたが、Widrow-Hoff の学習アルゴリズムの場合、出力は連続的に変化するため許容誤差変数 err_goal を指定する必要がある。この Widrow-Hoff では、線形ニューロンユニット(purelin)の重み係数 W とバイアス係数 B をチューニングする。チューニングする指標は E=T-A を用い、その誤差に対してもっとも急峻に変化する方向に係数 W、B をチューニングする方法を採っている。なぜなら、線形ネットワークの誤差曲線は、パラボリック関数で表現でき、かつ最小値は１つしかないためである。ここで重要なのは、学習レート変数 lr である。学習レート変数 lr を大きくすると学習が速くなり小さい値にすれば学習は遅くなる。学習レート lr の値は 1/（$(P^T \cdot P)$ の最大固有値）より小さい値をとるとよい。そのため Neural Network toolbox では、lr を計算する関数 maxlinlr を用意している。

```
lr=0.99*maxlinlr(P,'bias');
```

２番目のパラメータは、ニューロンユニットにバイアス係数がある場合には以下の計算式を用いている。

$$\max linlr = \frac{1}{\max\left(\left[P^T \vdots\ 1\right]\cdot\begin{bmatrix} P \\ \cdots \\ 1 \end{bmatrix}\right)}$$

パーセプトロンと Widrow-Hoff アルゴリズムの違いは、

```
E=T-A
dW = E*P';
dB = E;
```

learnp

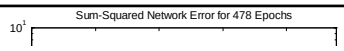
→

```
E=lr*(T-A)
dW = E*P';
dB = E;
```

learnwh

であり、エラーの計算の際に学習レート係数を乗算するか否かの違いである。
以上のアルゴリズムを用い学習する関数Mファイルが関数 trainwh として用意されている。
パーセプトロンと同様にディジタル回路の AND 論理の学習を例に考える。
Hidrow-Hoff の学習アルゴリズムのスクリプトは以下の様になる。

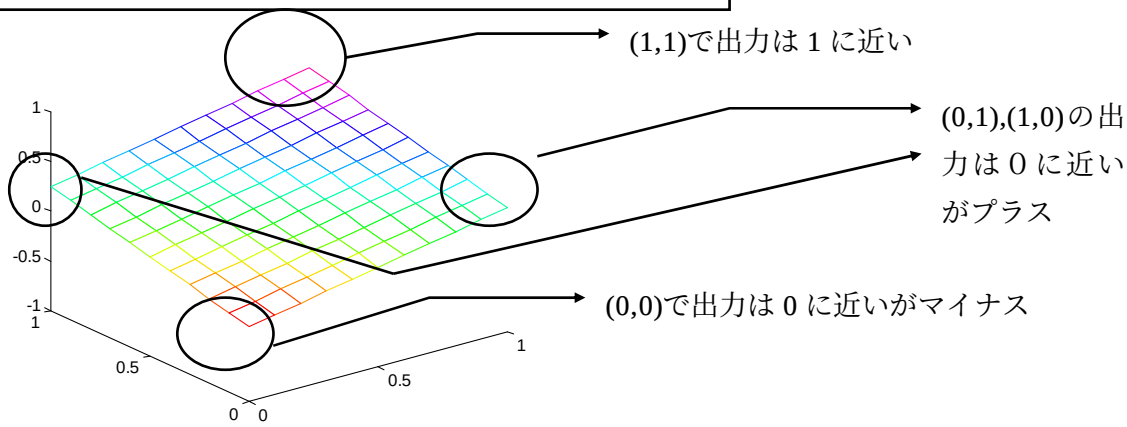
```
P=[0 0;0 1;1 0;1 1]';
T=[0 0 0 1];
R=2;S=1;
[W,B]=rands(S,R)
disp_freq=10;
max_epoch=1000;
err_goal=0.25;
lr=0.4* maxlinlr(P,'bias');
TP=[disp_freq max_epoch err_goal lr];
[W,B,epochs]=trainwh(W,B,P,T,TP)
```



この場合、入出力関係は、線形ではないため、誤差 err_goal は 0.25 で収束してしまっている。

なぜならば、線形ネットワークを適用しているため推定される変換面は曲面が構成できず平面であるためである。

```
for PP1=0:0.1:1
for PP2=0:0.1:1
PPP(PP1*10+1,PP2*10+1)=purelin(W*[PP1;PP2],B);
end;
end
mesh(0:0.1:1,0:0.1:1,PPP); axis([0 1 0 1 -1 1])
```



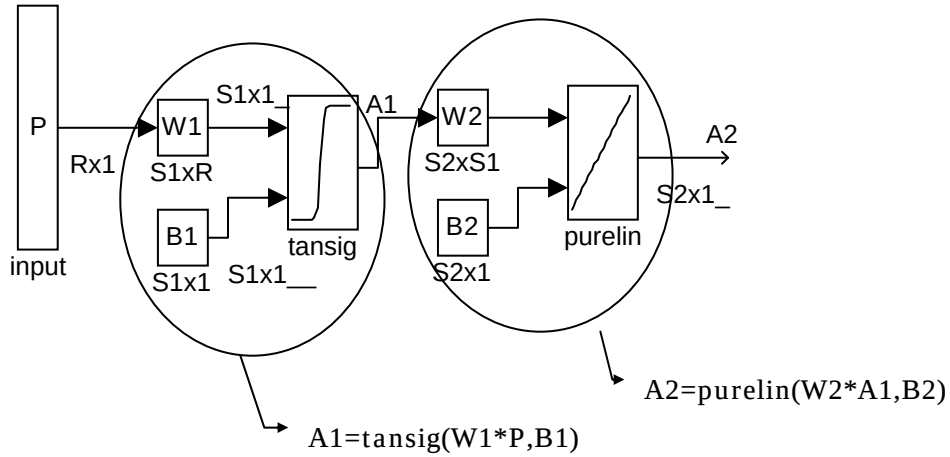
バックプロパゲーション

バックプロパゲーションとは、Widrow-Hoff の学習アルゴリズムを多層ネットワークに、そして、ニューロンユニットに微分可能な関数を使ったものである。。

バックプロパゲーション学習ルールは、重み行列とバイアス行列の大きさを調整することで、ネットワークの入力による出力と教師出力の差の２乗を最小にする。いわゆる非線形最自乗法の一種である。ネットワークの重み行列とバイアス行列の値を変化させたときの最も誤差を最小にする勾配方向にパラメータを調整するため、最急勾配法といわれている。学習されたニューラルネットワークは、たとえ一度も入力が与えられていない入力を代入してもそれなりの合理的な出力を得ることができる。このことをニューラルネットの汎化能力と呼びこれが、ニューラルネットワークを使う理由として最大の魅力でもある。

バックプロパゲーションで使われるニューロンユニットのタイプは、中間層のニューロンユニットとしてシグモイド関数（tansig や logsig）、出力ユニットとしてリニア関数（purelin）がよく用いられる。なぜならば、このコンビネーションを用いると容易に中間層の数次第で、どのような曲面の関数でも表現可能になるからである。

バックプロパゲーションに用いる階層ニューラルネットワーク



この階層ネットワークの出力を式にしてまとめて表現すると以下の様になる。

A2=purelin(__*tansig(W1*P,__),B2)

まずニューラルネットを学習させるためにパラメータの設定を行う。具体的には、重み係数 W1,W2 とバイアス係数 B1,B2 の設定をする。ここでは、中間層の数 S1 を 4 すると以下のように他の重み係数を設定することができる。

```
[R,Q]=size(P);
S1=4;
[S2,Q]=size(T);
[W1,B1]=rands(S1,R);
[W2,B2]=rands(S2,S1);
```

バックプロパゲーションアルゴリズムでは、ニューロンユニットの微分を求める必要性からニューロンユニットに_____や_____や_____関数などの連続関数を用いる。微分値を求めるためそれぞれのニューロンユニット関数に対して deltalin, deltatan, deltalog 関数が定義されている。基本的なアルゴリズムは、Widrow-Hoff の学習アルゴリズムと似ており違いは、E を計算する際に差分を使うのではなく微分関数によりその誤差の勾配を求める点が異なる。

```
E=lr*(T-A)
dW = E*P';
dB = E;
```

learnwh


```
E=lr*D;
dW = E*P';
dB = E;
```

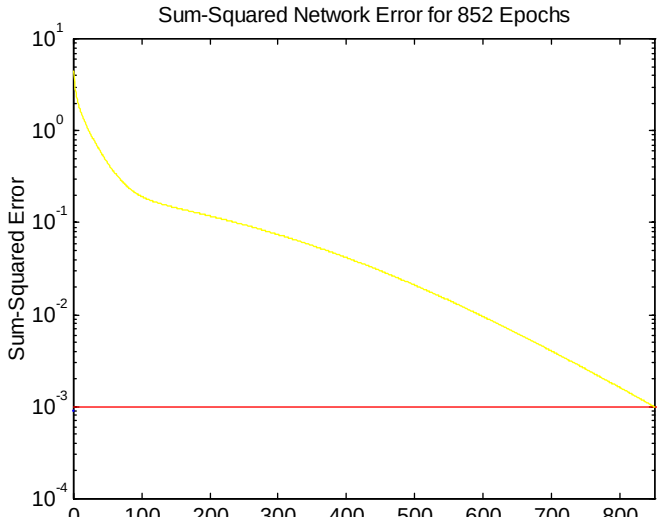
learnbp

ここで D は deltalin,deltatan,deltalog 関数などニューロンユニットのタイプの違いにより求められる微分されたベクトルを入力する。
以下に AND 論理を例にした learnbp を用いたスクリプトを示す。

```
P=[0 0;0 1;1 0;1 1]';
T=[0 0 0 1];
[R,Q]=size(P);
S1=6;
[S2,Q]=size(T);
[W1,B1]=rands(S1,R);
[W2,B2]=rands(S2,S1);
lr=0.02;
err_goal=0.02;
A1=logsig(W1*P,B1);A2=purelin(W2*A1,B2);
E=T-A2;
SSE=sumsqr(E);
for epoch=1:5000
if(SSE < err_goal) epoch=epoch-1;break; end
D2=deltalin(A2,E);D1=deltatan(A1,D2,W2);
[dW1,dB1]= learnbp(P,D1,lr);
[dW2,dB2]=learnbp(A1,D2,lr);
W1=W1+dW1;B1=B1+dB1;
W2=W2+dW2;B2=B2+dB2;
A1=logsig(W1*P,B1);A2=purelin(W2*A1,B2);
E=T-A2;
SSE=sumsqr(E);
end
```

以上のアルゴリズムを用い学習する関数 M ファイルが関数 trainbp として用意されている。

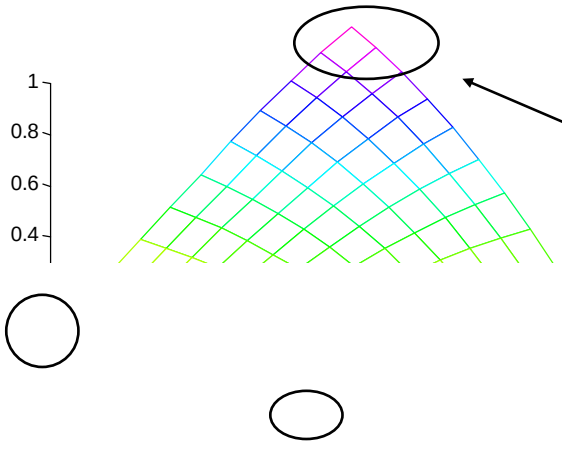
```
clear
close all
P=[0 0;0 1;1 0;1 1]'
T=[0 0 0 1];
[R,Q]=size(P);
S1=5;
[S2,Q]=size(T);
[W1,B1]=rands(S1,R);
[W2,B2]=rands(S2,S1);
disp_freq=10;
max_epoch=8000;
err_goal=0.001;
lr=0.02;
TP=[disp_freq max_epoch err_goal lr];
[W1,B1,W2,B2,epoch,TR]=trainbp(W1,B1,'tansig',W2,B2,'purelin',P,T,TP);
```



重み行列 W1,W2 とバイアス行列 B1,B2 は乱数で与えているため同じデータでも毎回異なった収束回数になるので上記のグラフと全く同じになるとは限らない。
以下は、入力と出力の関係を元に 3 次元グラフを作るスクリプトである。

このスクリプトを用いるとマウスにより 3 次元グラフを回転させることができる。

```
figure
for P1=0:0.1:1;
for P2=0:0.1:1
aaa(P1*10+1,P2*10+1)=purelin(W2*tansig(W1*[P1;P2]+B1)+B2);
end
end;
%%%%%%%
a=mesh(0:0.1:1,0:0.1:1,aaa);
d=0;
AZ=-37.5;EL=30;
pint=0;ppint=0;
b=gca;c=gcf;
%set(a,'erasemode','xor');
set(c,'windowbuttondownfcn','d=1;pint=get(c,"currentpoint");')
set(c,'windowbuttonmotionfcn',...
['if(d==1) ppint=get(c,"currentpoint");',...
'diffpint=pint-ppint;pint=ppint;',...
'AZ=AZ+diffpint(1);EL=EL+diffpint(2);view([AZ,EL]);end'])
set(c,'windowbuttonupfcn','d=0;')
```



(1,1)のみ出力は 1

(0,0),(0,1),(1,0)　ともに全て出力は 0 になっている。

作り出された平面は、曲面であり学習した AND 論理の点は全てみたしその間の値も滑らかに補間されている。

バックプロパゲーションアルゴリズムの高速化

バックプロパゲーションアルゴリズムは、xor 問題の様に_____できない場合でも学習させることが可能である。しかし、バックプロパゲーションの学習は、多層ネットワークということもあり大きいパターンを学習させようとすると全く実用にならないほど計算処理量が要求されてしまう。ここでは、バックプロパゲーションアルゴリズムにおいて高速化する 3 つの手法について紹介しその解説を行う

ここで述べる高速化手法は、以下の３つである。

1.モーメンタム修正法

2.よりよい初期値の与える

3.学習レートの適応化

モーメンタム修正法

この方法のアイデアは、通常のバックプロパゲーションアルゴリズムでは、重み行列 W の値とバイアス行列 B の修正値 dW,dB の値は、誤差の大きさ E によってのみ決まる。しかし誤差にノイズが入る様な場合、つまりローカルミニマムがたくさんあった場合などには、収束する際のそのローカルミニマムにある勾配の影響をそのまま受けてしまう。そこで、このモーメンタム修正法では、重み行列 W とバイアス行列 B の修正の際求めていた dW と dB に 1 次のローパスフィルタを付け加えることでローカルミニマムに陥らない様にする方法である。

```
dW = E*P';
dB = E;
```

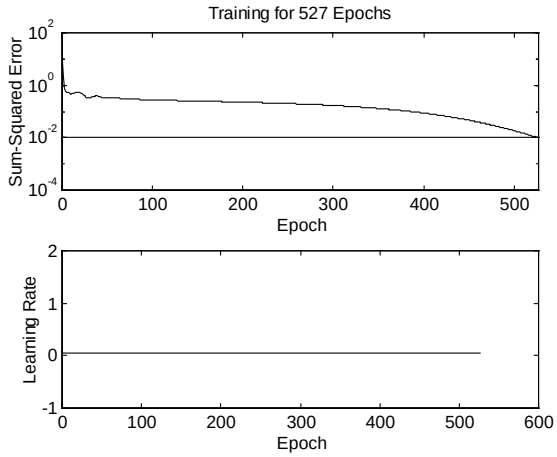


```
dW = mc*dW+(1-mc)*E*P';
dB = mc*dB+(1-mc)*E;
```

ここで用いる係数 mc は 0 から 1 までのフィルタの係数でもし mc=0 の場合には、従来のバックプロパゲーションアルゴリズムと変わりない。例えば、mc=.95 の場合には、現在の誤差の勾配よりも過去の勾配を重視する設定となる。この Neural Network Toolbox では、そのための学習関数として trainbpm が用意されている。

```
  err_goal=0.01;
  lr=0.05;
  momentum=0.95;
  err_ratio=1.04;
  TP=[disp_freq max_epoch err_goal lr momentum err_ratio];
[W1,B1,W2,B2,epoches,TR]=trainbpm(W1,B1,'tansig',W2,B2,'purelin',P,T,TP);
```

なおここで指定している変数 err_ratio は、モーメンタム係数 momentum によってフィルタリングされた値と、フィルタリングされていない値との比のことである。この値が大きい場合には、収束しない可能性があるためモーメンタム法を適用しない。



よりよい初期値の与える

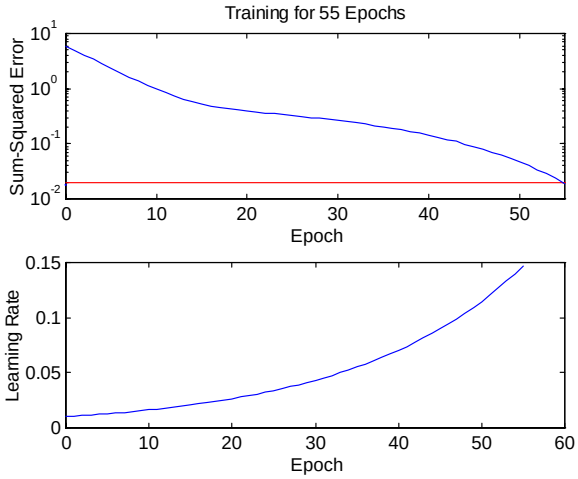
この方法は、Nguyen,Widrow らが提案した方法で、初期の重み係数 W とバイアス係数 B に与える乱数にある性質を持たすことでニューラルネットの収束を速くする手法である。この方法では、中間層のニューロンユニットの性質により例えば tansig または logsig の場合で初期値の与え方がことなるためそれぞれ nwtan、nwlog が用意されている。次段のニューロンユニットの係数は、通常の乱数を与えるが偏差は通常より少な目にする。Neural Network Toolbox では乱数を自動的に生成する initff 関数が定義されている。

```
[W1,B1,W2,B2] = initff(P,S1,'tansig',T,'purelin');
```

学習レートの適応化

バックプロパゲーションアルゴリズムでは、通常学習レート lr は固定である。この学習レートを可変にすることでより速い収束を実現することができる。以下にそのアルゴリズムを使った学習関数 M ファイル trainbpa 関数の例を示す。

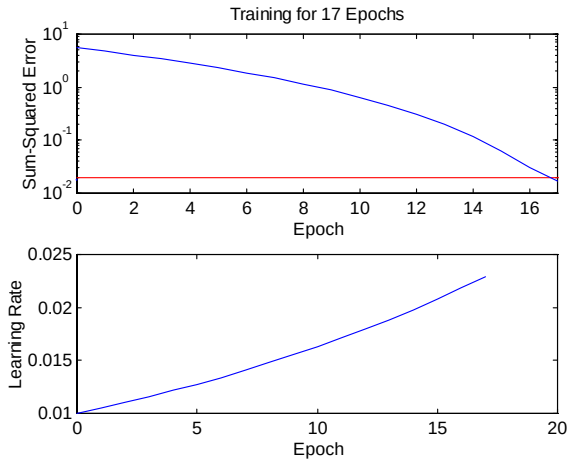
```
P=[0 0;0 1;1 0;1 1]'
T=[0 0 0 1];
S1=6;
[W1,B1,W2,B2] = initff(P,S1,'tansig',T,'purelin');
disp_freq=100;
max_epoch=8000;
err_goal=0.02;
lr=0.01;lr_inc = 1.05;lr_dec = 0.7;
err_ratio=1.04;
TP=[disp_freq max_epoch err_goal lr lr_inc lr_dec err_ratio];
[W1,B1,W2,B2,epoches,TR]=trainbpa(W1,B1,'tansig',W2,B2,'purelin',P,T,TP);
```



高速化手法を駆使したバックプロパゲーション

これら高速化手法モーメンタム法と適法学習レート法を同時におこなうための関数 M ファイルが trainbpx として用意されている。

```
P=[0 0;0 1;1 0;1 1]'
T=[0 0 0 1];
S1=6;
[W1,B1,W2,B2] = initff(P,S1,'tansig',T,'purelin');
disp_freq=100;
max_epoch=8000;
err_goal=0.02;
lr=0.01;lr_inc = 1.05;lr_dec = 0.7;;
err_ratio=1.04;
momentum = 0.9;
TP=[disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_ratio];
[W1,B1,W2,B2,epoches,TR]=trainbpx(W1,B1,'tansig',W2,B2,'purelin',P,T,TP);
```



乱数により学習回数が毎回変わるので何ともいえないが少なくともオリジナルのバックプロパゲーションアルゴリズムよりかなり速く収束する。

ここで扱ったニューラルネットワークの種類と関数

タイプ		ユニット	学習・トレイン関数	
単層ネットワーク	パーセプトロン	hardlim	learnp	trainp
	Widrow-Hoff	purelin	learnwh	tranwh
多層ネットワーク	Back propagation	tansig/logsig purelin	learnbp	trainbp
	モーメント		learnbpm	trainpbm
	学習レート		learnbpa	trainbpa
	学習+モーメント		learnbpx	trainbpx