# SAPT
# 第7回知能化分科会

一色 浩
（SAPT知能化分科会長）

2018.07.14

SAPT電子会議室C

# 目　　次

0.　復習…誤差逆伝播学習とは

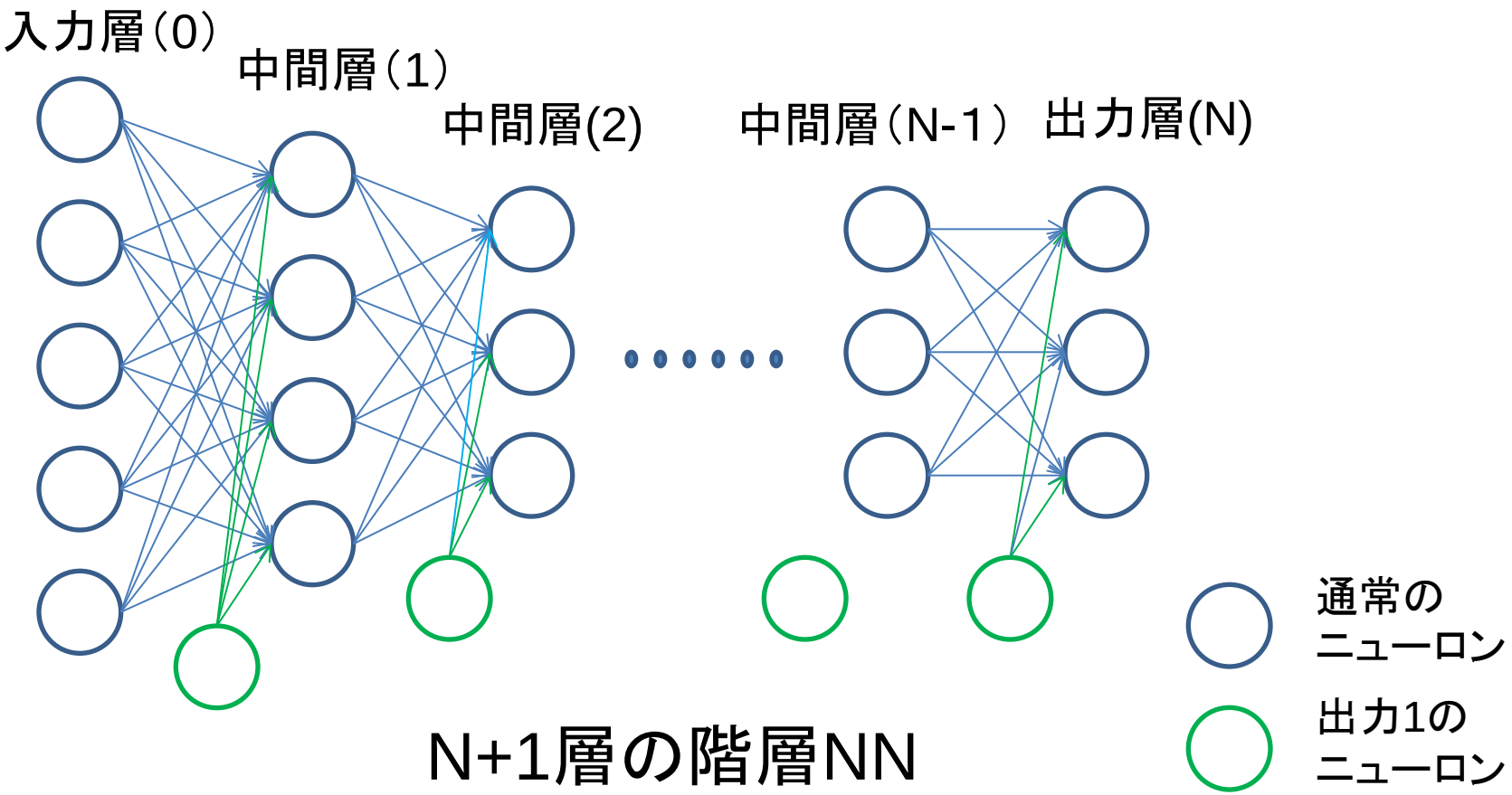
1. 瀧 雅人著「**これならわかる深層学習**」

2. 飲み会の代わり…自由討論

# 復習…誤差逆伝播学習とは

## 学習理論

計算機アルゴリズム研究会，「C言語による実用ニューロ・コンピューティング」，ラッセル社，(1992)




N+1層の階層NN

$p\,(p=0,1,2,\ldots,P-1)$: 学習パターン

$v_i^{(0)}(p)$: 入力層ユニット$i$のパターン$p$に対する出力
入力層では，入力＝出力

$u_i^{(n)}(p)$: 中間層入力　$v_i^{(n)}(p)$: 中間層出力

学習誤差:

$$E=\sum_p E(p) \qquad (3.1.1)$$

$$E(p)=\frac{1}{2}\sum_j\left[t_j(p)-v_j^{(N)}(p)\right]^2 \qquad (3.1.2)$$

$w_{ji}^{(n)}$：チャンネル$j$-$i$のウェイト　$-\theta_j^{(n)}$：ユニット$j$の閾値

第$n$+1層のユニット$j$の入力:

$$u_j^{(n+1)}(p)=\sum_i w_{ji}^{(n)}v_i^{(n)}(p)+\theta_j^{(n)} \qquad (3.1.3)$$

第$n$+1層のユニット$j$の出力:

$$v_j^{(n+1)}(p) = h\left(u_j^{(n+1)}(p)\right) \qquad (3.1.4)$$

式(3.1.3)より，$w_{ji}^{(n)}$ は $u_j^{(n+1)}$ にしか含まれないので

$$\frac{\partial E(p)}{\partial w_{ji}^{(n)}} = \frac{\partial E(p)}{\partial u_j^{(n+1)}(p)} \frac{\partial u_j^{(n+1)}(p)}{\partial w_{ji}^{(n)}} = -\delta_j^{(n+1)}(p) v_i^{(n)}(p) \qquad (3.1.5)$$

$$\delta_j^{(n+1)}(p) = -\frac{\partial E(p)}{\partial u_j^{(n+1)}(p)} \qquad (3.1.6)$$

$\delta_j^{(n+1)}(p)$: 第$n$+1層のユニット$j$のパターン$p$に対する誤差と呼ぶ.

同様に:

$$\frac{\partial E(p)}{\partial \theta_j^{(n)}} = \frac{\partial E(p)}{\partial u_j^{(n+1)}(p)} \frac{\partial u_j^{(n+1)}(p)}{\partial \theta_j^{(n)}} = -\delta_j^{(n+1)}(p) \cdot 1 \qquad (3.1.7)$$

式(3.1.4)より

$$\delta_j^{(n+1)}(p) = -\frac{\partial E(p)}{\partial v_j^{(n+1)}(p)} \frac{dv_j^{(n+1)}(p)}{du_j^{(n+1)}(p)} = -h'\left(u_j^{(n+1)}(p)\right) \frac{\partial E(p)}{\partial v_j^{(n+1)}(p)} \quad (3.1.8)$$

2乗誤差を仮定してるので，出力層のときには

$$\frac{\partial E(p)}{\partial v_j^{(N)}(p)} = -\left[t_j(p) - v_j^{(N)}(p)\right] \quad (3.1.9)$$

式(3.1.9)を式(3.1.8)に代入すれば

$$\delta_j^{(N)}(p) = h'\left(u_j^{(N)}(p)\right)\left[t_j(p) - v_j^{(N)}(p)\right] \quad (3.1.10)$$

中間層の場合には式(3.1.6)より

$$\frac{\partial E(p)}{\partial v_j^{(n+1)}(p)} = \sum_k \frac{\partial E(p)}{\partial u_k^{(n+2)}(p)} \frac{\partial u_k^{(n+2)}(p)}{\partial v_j^{(n+1)}(p)} = -\sum_k \delta_k^{(n+2)}(p) w_{kj}^{(n+2)} \quad (3.1.11)$$

なので，式(3.1.6)より

$$\delta_j^{(n+1)}(p) = h'\left(u_j^{(n+1)}(p)\right)\sum_k \delta_k^{(n+2)}(p) w_{kj}^{(n+2)} \quad (3.1.12)$$

となる．すなわち，パターン$p$に対する第$n$+1層のユニット$j$の誤差$\delta_j^{(n+1)}(p)$ は第$n$+2層の$\delta_j^{(n+1)}(p)$ から伝播されたもの．これが誤差逆伝播法の名前の由来．

最急降下法による修正：

$$\Delta_p w_{ji}^{(n)} = -\frac{\partial E(p)}{\partial w_{ji}^{(n)}}\eta_w \quad \Delta_p \theta_i^{(n)} = -\frac{\partial E(p)}{\partial \theta_i^{(n)}}\eta_o \quad (3.1.13,14)$$

一括修正の場合は：

$$\Delta w_{ji}^{(n)} = -\frac{1}{P}\left(\sum_p \frac{\partial E(p)}{\partial w_{ji}^{(n)}}\right)\eta_w \quad \Delta_p \theta_i^{(n)} = -\frac{1}{P}\left(\sum_p \frac{\partial E(p)}{\partial \theta_i^{(n)}}\right)\eta_o \quad (3.1.15,16)$$

さらに，1ステップ前の修正量$\Delta^{(-1)}w_{ji}^{(n)}$と$\Delta^{(-1)}\theta_j^{(n)}$の影響を加味すると，効果的な修正が期待できる．ただし，修正の最終段階では，前回と同じように修正する．

$$\Delta w_{ji}^{(n)} = -\frac{1}{P}\left(\sum_p \frac{\partial E(p)}{\partial w_{ji}^{(n)}}\right)\eta_w + \left(\Delta^{(-1)}w_{ji}^{(n)}\right)\alpha \quad (3.1.17)$$

$$\Delta \theta_i^{(n)} = -\frac{1}{P}\left(\sum_p \frac{\partial E(p)}{\partial \theta_i^{(n)}}\right)\eta_o + \left(\Delta^{(-1)}\theta_i^{(n)}\right)\alpha \quad (3.1.18)$$

$\alpha$は慣性係数と呼ばれる定数であるが，前ステップの値を$\alpha^{(-1)}$，増分を$\Delta\alpha$として

$$\alpha = \alpha^{(-1)} + \Delta\alpha \quad \text{when} \quad \alpha^{(-1)} + \Delta\alpha < A \quad (3.1.19a)$$

$$\alpha = A \quad \text{when} \quad \alpha^{(-1)} + \Delta\alpha \geq A \quad (3.1.19b)$$

最急降下法は一定ステップの探索法であるので，以下のような欠点が指摘されている.

(1) 局所的最小値に捕まると，そこから抜け出せない. 大域的最小値に収束する保証がない.

(2) 最小値近傍では修正量がほとんど0となるので，収束が悪い. 一括完成修正法も十分な解決を与えていない.

ユニットの出力関数としては，シグモイド関数がある:

$$h(u) = 1 \Big/ \left(1 + e^{-u}\right) \qquad -\infty < u < \infty \qquad (3.1.20)$$

シグモイド関数には以下のような便利な関係がある:

$$h(u)(1 + e^{-u}) = 1 \qquad (3.1.21)$$

$$h'(u) = h(u)(1 - h(u)) \qquad (3.1.22)$$

# C言語によるコード例（RTINY.c … 来週配布）

```
void main()                                  メイン関数
{

    read_system_data();                      システムデータの読み込み
    read_param();                            パラメータの読み込み


    read_input_data();                       入力データの読み込み
    read_teach_data();                       教師データの読み込み

    initialize_weight();                     ウェイト初期値の設定
    initialize_offset();                     オフセット初期値の設定

    fp_out = open_output_file();    出力ファイルの設定

    save_all_1(fp_out);                      計算条件を出力ファイルへ記録
```

```
loop = 0;
while (loop < iteration_lmt) {          繰り返しループ開始
    error = 0.0;

    for (p = 0; p < NT; p++) {          教師データの数だけ繰り返し
        forward(p);                     NN順方向計算
        delta(p);                       NN逆方向誤差計算
    }

    correct_weight();                   ウェイトの修正

    print_error(loop);
    save_all_2(fp_out, loop);           エラーの記録

    if ((error / NT / OU < error_lmt))
        alpha += dalpha;                慣性収束の手続き
    loop++;
}       /* LOOP END */                  繰り返しループ終了
```

```
                                                    結果を計算

for (p = 0; p < NT; p++)
        forward(p);                                 NN順方向計算

    save_all_3(fp_out);                             計算結果を出力ファイルへ記録

    close_output_file(fp_out); /                    出力ファイルを閉じる

    quit();                                         終了
}
```

```
// ----------------------------順方向の計算---------------------------- //

void forward(p)
int  p;
{
    int i, j;

    for (i = 0; i < IU; i++) {
        O_I[p][i] = input_data[p].input[i];                    入力層出力
    }

    for (j = 0; j < HU; j++) {
        NET_H[p][j] = 0.0;                                    隠れ層入力の計算
        for (i = 0; i < IU; i++)
            NET_H[p][j] += W_H_I[j][i] * O_I[p][i];
        NET_H[p][j] += OFF_H[j];
        O_H[p][j] = sigmoid(NET_H[p][j]);                     隠れ層出力の計算
    }
```

```
for (j = 0; j < OU; j++) {
        NET_O[p][j] = 0.0;                                  出力層入力の計算
        for (i = 0; i < HU; i++)
            NET_O[p][j] += (W_O_H[j][i] * O_H[p][i]);
        NET_O[p][j] += OFF_O[j];
        O_O[p][j] = sigmoid(NET_O[p][j]);                   出力層出力の計算
    }
}
```

```
/// ----------------------------誤差の計算---------------------------- //

void delta(p)
int  p;
{
    int     j, k;        double  sum;

    for (j = 0; j < OU; j++) {                                  出力層の誤差計算
        DELTA_O[p][j] = (teach_data[p].teach[j] - O_O[p][j])
                * u0 * O_O[p][j] * (1.0 - O_O[p][j]);
        error += (fabs(teach_data[p].teach[j] - O_O[p][j]));
    }

    for (j = 0; j < HU; j++) {                                  隠れ層の誤差計算
        sum = 0.0;
        for (k = 0; k < OU; k++)
            sum += DELTA_O[p][k] * W_O_H[k][j];
        DELTA_H[p][j] = 2.0 / u0 * O_H[p][j] * (1.0 - O_H[p][j]) * sum;
    }
}
```

```
// -------------------------慣性法／荷重更新------------------------ //
void correct_weight()
{
    int  i, j, p;        double sum;

    for (j = 0; j < OU; j++) {
        for (i = 0; i < HU; i++) {
            sum = 0.0;
            for (p = 0; p < NT; p++)
                sum += DELTA_O[p][j] * O_H[p][i];      一括修正
            DW_O_H[j][i] = eta_wgt * sum
                        + alpha * DW_O_H[j][i];          慣性修正
            W_O_H[j][i] += DW_O_H[j][i];       出力層･隠れ層間のウェイト修正
        }
        sum = 0.0;
        for (p = 0; p < NT; p++)
            sum += DELTA_O[p][j] * 1.0;                  一括修正
        DOFF_O[j] = eta_wgt * sum
                  + alpha * DOFF_O[j];                   慣性修正
        OFF_O[j] += DOFF_O[j];                 出力層のオフセット修正
    }
```

```
    for (j = 0; j < HU; j++) {
        for (i = 0; i < IU; i++) {
            sum = 0.0;
            for (p = 0; p < NT; p++)
                sum += DELTA_H[p][j] * O_I[p][i];          一括修正
            DW_H_I[j][i] = eta_wgt * sum
                        + alpha * DW_H_I[j][i];            慣性修正
            W_H_I[j][i] += DW_H_I[j][i];                   隠れ層･入力層間のウェイト修正
        }
        sum = 0.0;
        for (p = 0; p < NT; p++)
            sum += DELTA_H[p][j] * 1.0;                    一括修正
        DOFF_H[j] = eta_wgt * sum
                  + alpha * DOFF_H[j];                     慣性修正
        OFF_H[j] += DOFF_H[j];                             隠れ層のオフセット修正
    }
}
```

# RTINY.c による計算結果1

## AND, OR, XORの計算

計算条件

システムデータ

IU = 2
HU = 2
OU = 3
NT = 4

計算パラメータ

| | |
|---|---|
| u0 | 0.6 |
| inival_wgt | 0.3 |
| inival_off | 0.2 |
| eta_wgt | 0.8 |
| eta_off | 0.6 |
| alpha | 0.3 |
| dalpha | 0.01 |
| alpha_lmt | 0.9 |
| error_lmt | 0.005 |
| iteration_lmt | 1000 |

入力データ

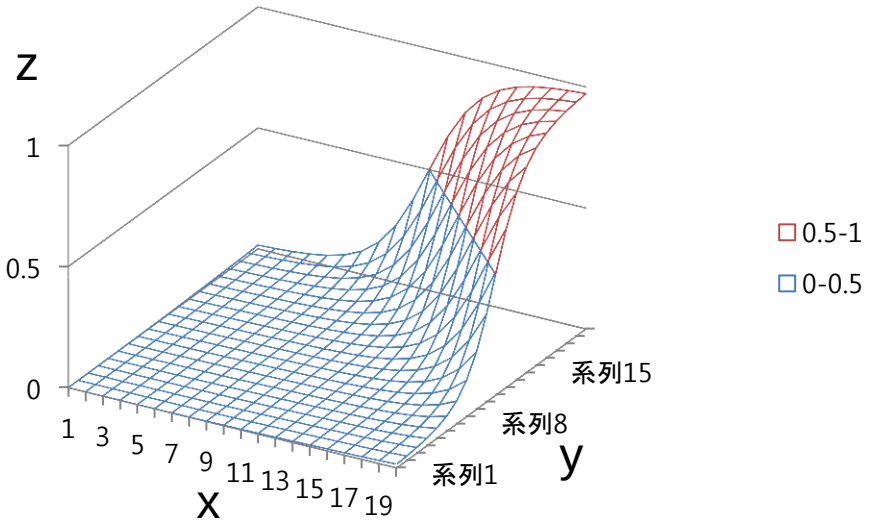
(1) 0, 0
(2) 1, 0
(3) 0, 1
(4) 1, 1

教師データ

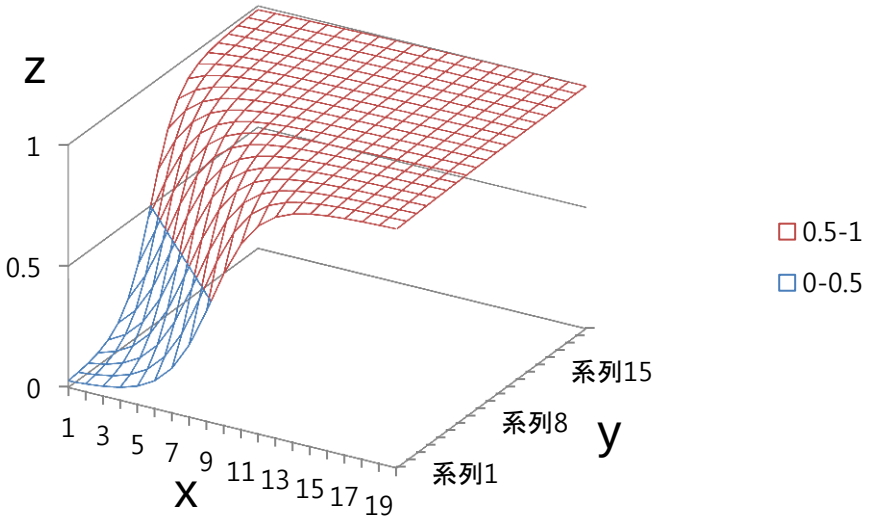
(1) 0, 0, 0
(2) 0, 1, 1
(3) 0, 1, 1
(4) 1, 1, 0

# 出力結果

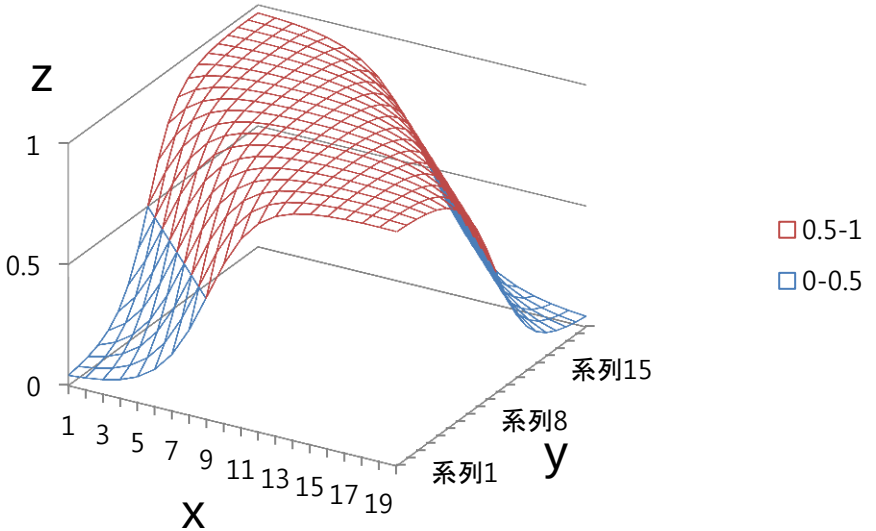
| 入力値 | 入力値 | 出力値 | 出力値 | 出力値 |
|---|---|---|---|---|
| ユニット0 | ユニット1 | ユニット0 | ユニット1 | ユニット2 |
| | | AND | OR | XOR |
| 0 | 0 | 0 | 0.02 | 0.04 |
| 1 | 0 | 0.02 | 0.98 | 0.97 |
| 0 | 1 | 0.02 | 0.98 | 0.97 |
| 1 | 1 | 0.98 | 1 | 0.04 |




(a) AND




(b) OR




(c) XOR

AND, OR, XOR でできる曲面

# RTINY.c による計算結果2(シグモイド→ReLU)

XOR(線形不分離)が解けなくなる.

| 入力値 | 入力値 | 出力値 | 出力値 | 出力値 |
|---|---|---|---|---|
| ユニット0 | ユニット1 | ユニット0 | ユニット1 | ユニット2 |
| | | AND | OR | XOR |
| 0 | 0 | 0 | 0.29 | 0 |
| 1 | 0 | 0.19 | 0.73 | 0 |
| 0 | 1 | 0.04 | 0.84 | 0 |
| 1 | 1 | 0.88 | 1.21 | 0 |

# 瀧 雅人著「これならわかる深層学習」

今回の講義では，以下の各章を取りあへず省略する．

4　勾配降下法による学習

5　深層学習の正規化

6　誤差逆伝播法

# 7　自己符号化器

## 7.1　データ圧縮と主成分分析

64×64ピクセルの小さな白黒画像でも4096次元．ただし，データ点の分布は均一ではなく，粗密がある．

図7.1は単純なケースで，(a)では$x_1$方向には分散が大きく，$x_2$方向には分散が小さい．

データが広がっている方向は分散で特定できる．

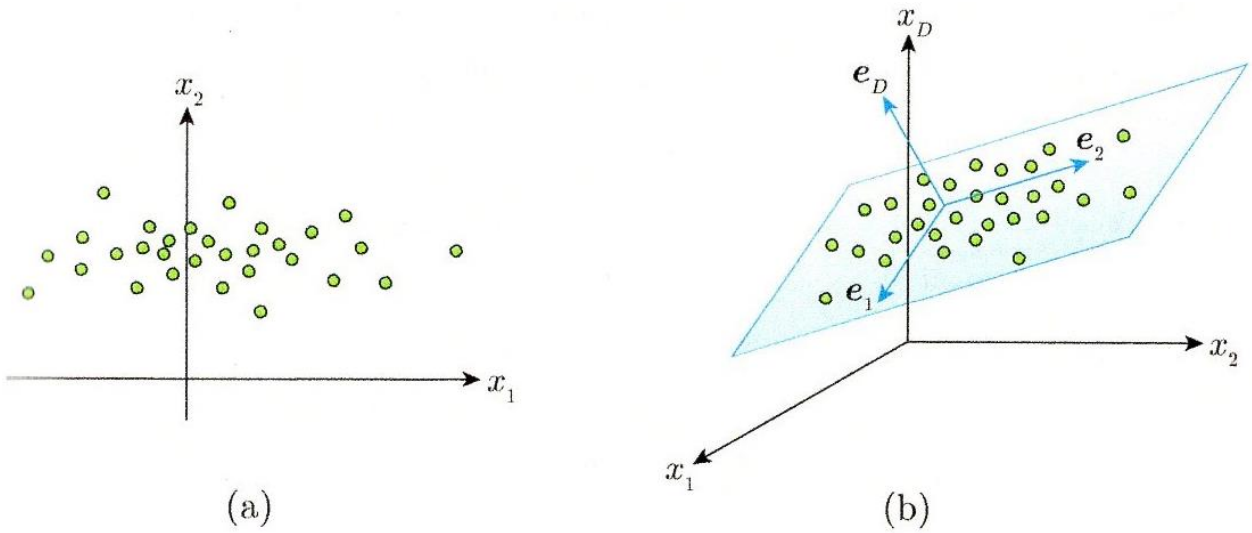



図7.1　(a)は2次元空間中のデータの偏り. (b)は高次元空間中での，平坦部分空間への散布の偏り

このような解析法を主成分分析（principal component analysis, PCA）という.

データ点をベクトル.

$$\mathbf{x} = \begin{pmatrix} x_1 & x_2 & \cdots & x_D \end{pmatrix}^T \qquad (7.1)$$

とする. 図7.1bのように平坦な部分空間は，元来の座標軸の方向を向いていない.

この方向とそれに直交する方向の正規直行基底を

$$\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d \qquad (7.2)$$

とする. これらは分散の大きい順に取られているものとする. この基底ベクトルの決定法を述べる.

任意のデータ点$\mathbf{x}_n$をこの部分空間に射影する.

$$\mathbf{x}_n \approx \mathbf{c}_0 + \sum_{h=1}^{d} \left( \mathbf{e}_h{}^T (\mathbf{x}_n - \mathbf{c}_0) \right) \mathbf{e}_h \qquad (7.3)$$

ここで, $\mathbf{c}_0 = \sum_n \mathbf{x}_n / N$ とする.

右辺で与えた射影がなるべく左辺を近似するためには, 二乗誤差を最小にすれば良い. すなわち

$$E(\mathbf{c}_0, \mathbf{e}_h) = \sum_{n=1}^{N} \left( (\mathbf{x}_n - \mathbf{c}_0) - \sum_{h=1}^{d} \left( \mathbf{e}_h{}^T (\mathbf{x}_n - \mathbf{c}_0) \right) \mathbf{e}_h \right)^2 \qquad (7.4)$$

として以下の最小値問題を解くことになる:

$$\min_{\mathbf{c}_0,\mathbf{e}_h} E(\mathbf{c}_0,\mathbf{e}_h) \qquad (7.5)$$

簡単のために, $\Delta\mathbf{x}_n = \mathbf{x}_n - \mathbf{c}_0$ とする. さらに

$$\mathbf{\Gamma}^T \equiv (\mathbf{e}_1 \ \ \mathbf{e}_2 \ \ \cdots \ \ \mathbf{e}_d) \qquad (7.6)$$

という$D \times d$行列を導入し,$D \times D$射影行列を定義する:

$$\mathbf{P} \equiv \mathbf{\Gamma}^T \mathbf{\Gamma} \qquad (7.7)$$

基底ベクトルが正規直交であることを用いると,$\mathbf{I}$ を単位行列として

$$\mathbf{P}^T = \mathbf{P}$$

$$\mathbf{P}^2 = \mathbf{\Gamma}^T \mathbf{\Gamma}\mathbf{\Gamma}^T \mathbf{\Gamma} = \mathbf{\Gamma}_{D\times d}{}^T \mathbf{I}_{d\times d} \mathbf{\Gamma}_{d\times D} = \mathbf{\Gamma}^T \mathbf{\Gamma} = \mathbf{P}$$

であるので, 式(7.4)の第2項は

$$\sum_{h=1}^{d}(\mathbf{e}_h{}^T\Delta\mathbf{x}_n)\mathbf{e}_h=\sum_{h=1}^{d}\mathbf{e}_h(\mathbf{e}_h{}^T\Delta\mathbf{x}_n)=\mathbf{\Gamma}^T\mathbf{\Gamma}\Delta\mathbf{x}_n=\mathbf{P}\Delta\mathbf{x}_n \quad (7.8)$$

と書ける．式(7.4)に代入すると

$$E=\sum_{n=1}^{N}(\Delta\mathbf{x}_n-\mathbf{P}\Delta\mathbf{x}_n)^2=\sum_{n=1}^{N}\Delta\mathbf{x}_n{}^T(\mathbf{I}-\mathbf{P})^T(\mathbf{I}-\mathbf{P})\Delta\mathbf{x}_n$$

$$=\sum_{n=1}^{N}\Delta\mathbf{x}_n{}^T(\mathbf{I}-\mathbf{P})\Delta\mathbf{x}_n \quad (7.9)$$

$\Delta\mathbf{x}_n=\mathbf{x}_n-\mathbf{c}_0$ であるので $\mathbf{c}_0$ に関する最小化は

$$0=\frac{\partial E}{\partial c_0}=-2\sum_{n=1}^{N}(\mathbf{I}-\mathbf{P})\Delta\mathbf{x}_n=-2(\mathbf{I}-\mathbf{P})\sum_{n=1}^{N}\Delta\mathbf{x}_n \quad (7.10)$$

であるので，式(7.4)に代入すると $\sum_{n=1}^{N}\Delta\mathbf{x}_n=0$

より，$c_0=\sum_{n=1}^{N}\mathbf{x}_n\Big/N$ となる．

つぎに，$\mathbf{e}_h$ について考える.

簡単のために以下では$\mathbf{c}_0 = 0$とする. すると誤差関数は

$$E = \sum_{n=1}^{N} (\mathbf{x}_n)^2 - \sum_{n=1}^{N} \mathbf{x}_n{}^T \mathbf{P} \mathbf{x}_n = \sum_{n=1}^{N} (\mathbf{x}_n)^2 - N \sum_{h=1}^{d} \mathbf{e}_h{}^T \mathbf{\Phi} \mathbf{e}_h \qquad (7.11)$$

と書ける. ここで $\mathbf{\Phi}$ は共分散行列:

$$\mathbf{\Phi} = \frac{1}{N} \sum_{n=1}^{N} x_{ni} x_{nj}$$

∵)

$$\frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n{}^T \mathbf{P} \mathbf{x}_n = \frac{1}{N} \sum_{n=1}^{N} \sum_{h'=1}^{d} \sum_{h=1}^{d} x_{nh'} x_{nh} \mathbf{e}_{h'}{}^T \mathbf{P} \mathbf{e}_h \qquad (a)$$

$$\mathbf{e}_{h'}{}^T \mathbf{P} \mathbf{e}_h = \mathbf{e}_{h'}{}^T \mathbf{\Gamma}^T \mathbf{\Gamma} \mathbf{e}_h \qquad (b)$$

$$\mathbf{\Gamma e}_h = \begin{pmatrix} \delta_{1h} \\ \delta_{2h} \\ \vdots \\ \delta_{dh} \end{pmatrix} \qquad \mathbf{e}_{h'}{}^T \mathbf{\Gamma}^T = \begin{pmatrix} \delta_{1h'} & \delta_{2h'} & \cdots & \delta_{dh'} \end{pmatrix} \quad \text{(c)}$$

$$\mathbf{e}_{h'}{}^T \mathbf{\Gamma}^T \mathbf{\Gamma e}_h = \left( \delta_{ih'} \delta_{jh} \right) \quad \text{(d)}$$

$$\sum_{n=1}^{N} \mathbf{x}_n{}^T \mathbf{P x}_n = \sum_{n=1}^{N} \sum_{h'=1}^{d} \sum_{h=1}^{d} x_{nh'} x_{nh} \delta_{ih'} \delta_{jh} = \sum_{n=1}^{N} x_{ni} x_{nj} \quad \text{(e)}$$

QED

したがって主成分分析とは

$$\max_{\mathbf{e}_h} \sum_{h=1}^{d} \mathbf{e}_h{}^T \mathbf{\Phi e}_h \quad \text{subject to} \quad (\mathbf{e}_h)^2 = 1 \qquad (7.12)$$

ラグランジュの未定乗数法を用いると

$$L(\mathbf{e}_h, \lambda_h) = \sum_{h=1}^{d} \mathbf{e}_h{}^T \mathbf{\Phi} \mathbf{e}_h - \sum_{h=1}^{d} \lambda_h \left( (\mathbf{e}_h)^2 - 1 \right) \qquad (7.13)$$

これより

$$0 = \frac{1}{2} \frac{\partial L}{\partial \mathbf{e}_h} = \mathbf{\Phi} \mathbf{e}_h - \lambda_h \mathbf{e}_h \qquad (7.14)$$

すなわち，共分散行列 $\mathbf{\Phi}$ の固有値問題に他ならない．

式(7.11)から明らかなように $\mathbf{\Phi}$ は $D \times D$ なので，固有ベクトルは$D$個あり得るが，大きい方から$d$個を取る．

## 7.2 自己符号化器

主成分分析はデータが散在する平坦な(すなわち線型の)部分空間を探すが，実際のデータは平坦な部分空間ではなく曲面に散布している．このような部分空間を多様体と呼び，多様体上へデータの次元削減をするアプローチを多様体学習という．

汎用的な表現能力を持つニューラルネットは多様体学習に役立つ．

### 7.2.1 砂時計型ニューラルネット

階層型ニューラルネットで有用な表現を生み出す簡単な方法がある．

図7.2(左)に示される左右対象な砂時計型NNである．

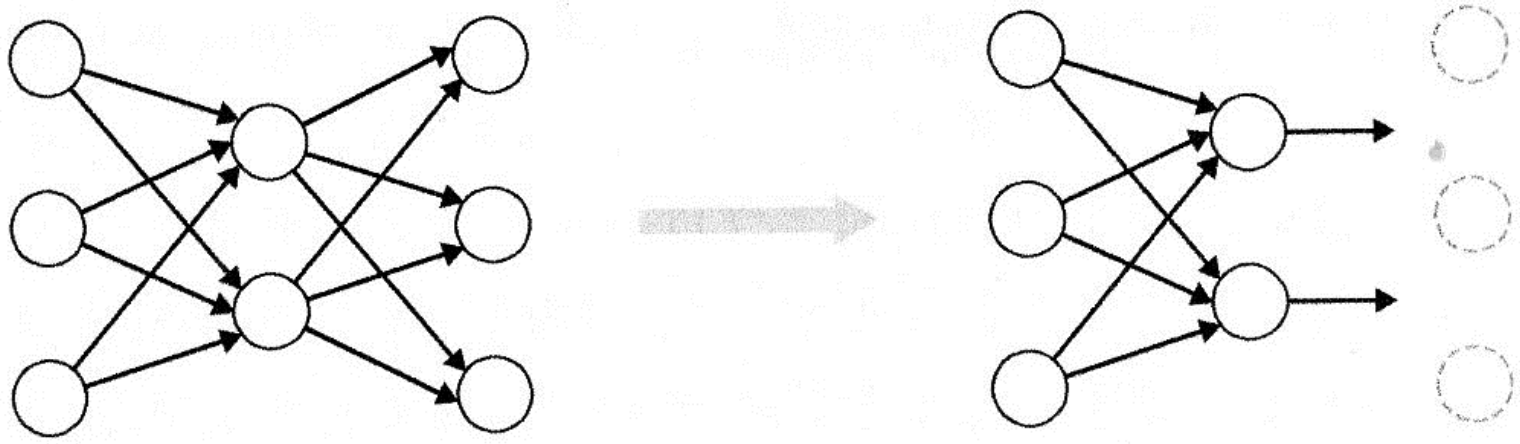


図7.2　砂時計型ニューラルネットによる自己符号化器の作成

出力層には入力データと同じ教師データを与える.

$$\mathbf{x} \to \mathbf{y} = f(\mathbf{W}\mathbf{x} + \mathbf{b}) \to \hat{\mathbf{x}} = \tilde{f}(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{7.16}$$

前半の変換操作$\mathbf{x} \to \mathbf{y}$を符号化と呼び，前半2層を符号化器と呼ぶ．符号化器の出力$\mathbf{y}$は符号と呼ばれる.

この砂時計型NNの学習は，入力$\mathbf{x}_n$が出力$\hat{\mathbf{x}}(\mathbf{x}_n)$ができる限り$\mathbf{x}_n$に近づくように行われる.

入力自身の符合を作りそれを復号するので自己符号化器と呼ばれる.

## 7.2.2 再構成誤差による学習

（1） 実数値入力

入力 $\mathbf{x}$ が実数値の場合には，出力層は線形ユニットを用いる．したがって活性化関数 $\tilde{f}$ は恒等演算です．

誤差関数は平均二乗誤差です．

$$E(\mathbf{W}, \tilde{\mathbf{W}}) = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{x}_n - \hat{\mathbf{x}}(\mathbf{y}_n))^2 \qquad (7.17)$$

（2） 2値入力

$\mathbf{x} = (x_1, \ldots, x_{d_1})^T$ の成分が0か1の場合，出力層にはシグモイドユニットを用いる．図7.2（左）に示される左右対象な砂時計型NNである．$\tilde{f}$ は入力成分数個のシグモイド関数である．

そのそれぞれが $x_i$ の予測値

$$\hat{x}_i(\mathbf{x}) = P(\hat{x}_i = 1 | \mathbf{x}) \ldots ? \quad (7.18)$$

を出力する. 学習は対数尤度関数:

$$P(\hat{\mathbf{x}} | \mathbf{x}) = \prod_{i=1}^{d_1} P(\hat{x}_i = 1 | \mathbf{x})^{\hat{x}_i} (1 - P(\hat{x}_i = 1 | \mathbf{x}))^{1-\hat{x}_i} \quad (7.19)$$

により行われる. つまり交差エントロピーの和を用いる:

$$E(\mathbf{W}, \tilde{\mathbf{W}}) = -\sum_{n=1}^{N} \sum_{i=1}^{d_1} \left( x_{ni} \log \hat{x}_i(\mathbf{x}_n) + (1 - x_{ni}) \log(1 - \hat{x}_i(\mathbf{x}_n)) \right) \quad (7.20)$$

自己符号化器の誤差は再構成誤差と呼ばれる.

すべての重みを自由パラメータとすると過剰なので, 正則化として下記の重み共有を課す.

$$w_{ji} = w_{ij} \quad (7.21)$$

## 7.2.3 符号化器の役割

符号化器の役割についての考察が述べられているが，仮定が大き過ぎてよく理解できない．

## 7.2.4 自己符号化器と主成分分析

活性化関数 $f$ と $\tilde{f}$ が恒等写像のとき，再構成誤差(7.26)が主成分分析の誤差(7.9)と同様な形になる．

すなわち，この場合の自己符号化器はまさに主成分分析を行っているという主張であるが，仮定が大き過ぎる．

理論的検討は重要であるが，数値計算などで数値的に裏付ける方が分かり易いと思われる．

## 7.3 スパース自己符号化器

### 7.3.1 自己符号化器のスパース化

これまでは入力層のユニット数$d_1$が中間層のユニット数$d_2$よりも大きな場合($d_1 > d_2$)を述べたが，逆の場合($d_1 \leq d_2$)にも，ある条件の下でデータ圧縮が可能.

安易に行うと恒等写像を学習してしまい無意味になるが，中間層の大半がどんな入力に対しても0しか出力しないスパース化というアイデアがある.

以下，我々のような初心者には馴染みにくい理論展開になるので省略する.

### 7.3.2 スパース自己符号化器の誤差逆伝播法

省略

# 7.4 積層自己符号化器と事前学習

## 7.4.1 積層自己符号化器

NNには勾配消失問題が起こるため，図7.3のような多層の深層自己符号化器を学習させることは困難.

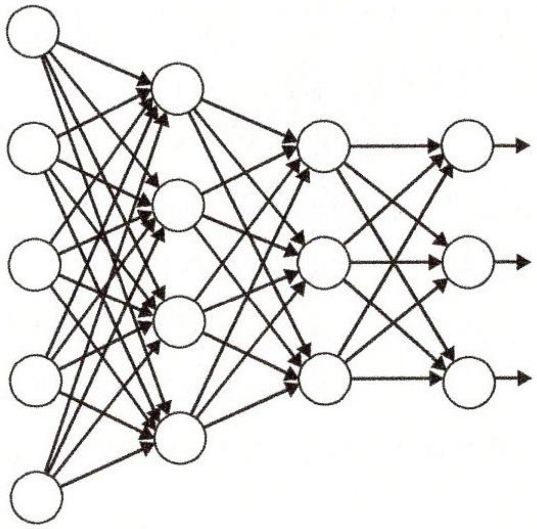


図7.3 4層からなる多層の符号化器

中間層が1層の場合は勾配消失問題の害を受けない.

問題なく学習できる浅い符号化器を活用を考える.

そこで2つの隣接層 $\mathbf{z}^{(l)} \rightarrow \mathbf{z}^{(l+1)}$ を取り出す.

そして多層の符号化器を2層の符号化器の集まりとみなす.

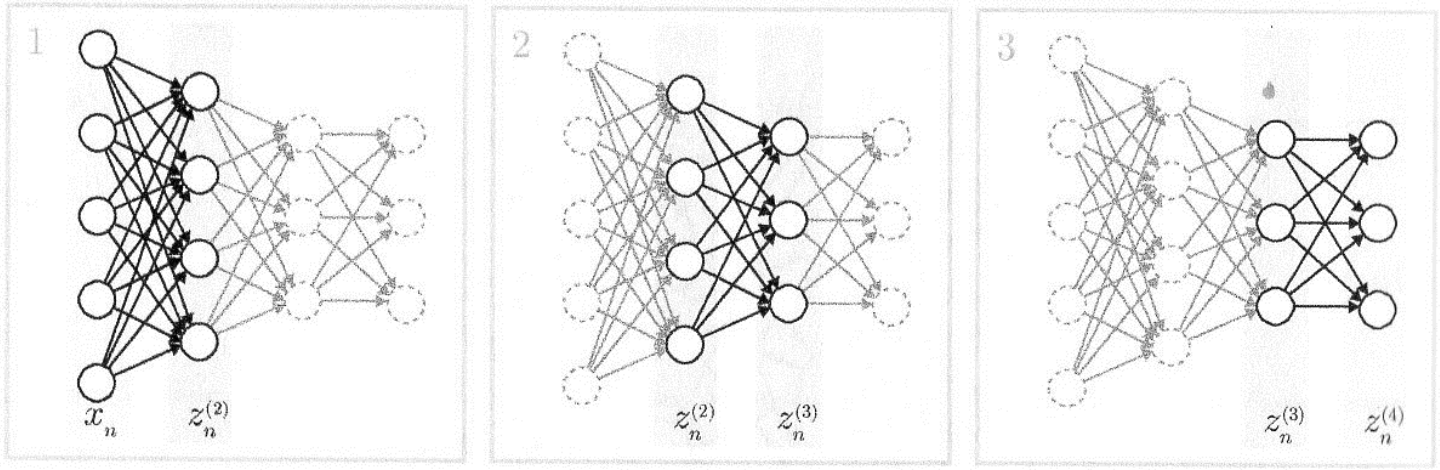



図7.4　多層符号化器の層ごとの貪欲法による学習

（1） まず1，2層だけを取り出し，これに入力層と同じユニット数の層を加える．これを訓練データ$\{\mathbf{x}_n\}$で学習させる．これで得られた符号を$\{\mathbf{z}_n^{(2)}\}$とする．

（2） つぎに2，3層を取り出し，出力層をおいて自己符号化の学習をさせる．その際の訓練データは$\{\mathbf{z}_n^{(2)}\}$で，新たにできた符号を$\{\mathbf{z}_n^{(3)}\}$とする．

（3） この操作を上層に向かって繰り返す．

（4） 出力層まで学習が終わったら，各段階で得られた学習済重みを全体の重みとして採用する．

この学習法を層ごとの貪欲法といい，貪欲法でできた自己符号化器を積層自己符号化器（SAE）という．

SAEは入力データの良い表現（符合）を獲得していると期待できる.

例えばエンの率いるGoogleチームが，SAEを用いた画像の教師なし学習で，猫などの個別の対象のみに反応する「おばあさん細胞」を実現した. いわゆるGoogleの猫である.

## 7.4.2 事前学習

多層NNの学習では，勾配法を適用する誤差関数がうねうねしているので，なかなか学習が進まない.

他にもプラトーや勾配消失問題がある.

学習を進める重要なトリックがパラメータの初期値選択である. SAEはよい初期値を与ると考えられる.

*L*層の深層NNを考える．出力層は回帰分析をするので，*L*-1層までSAEとして貪欲法で学習させ，その重みパラメータを深層NNの初期値とする．

その上で*L*-1層と*L*層の間の重みはランダムに選び，深層NN全体を通常の誤差逆伝播法で学習させる．

すると学習が大幅に加速することが経験的に知られている．

この事前学習はベンジオ等により開発され，よく使われたが，最近では事前学習に頼らない深層学習法が登場したので，必ずしも最先端の研究現場で用いられていない．

# 7.5 デノイジング自己符号化器

NNはノイズに対してロバストではない.

ノイズにより入力データを拡張した上で，自己符号化器を学習させると，ロバストになることが期待される.

これをデノイジング自己符号化器（DAE）と呼ぶ.

入力サンプルの各成分$x_{ni}$が実数値を取るとして，各訓練サンプル$\mathbf{x}_n$ごとに，適当なガウスノイズ $\delta\mathbf{x}_n \sim N(0,\sigma^2)$ を乗せて，訓練データ$\{\mathbf{x}_n+\delta\mathbf{x}_n, \mathbf{x}_n\}$で自己符号化器を学習させる.

ノイズが加わった入力 $\mathbf{x}_n+\delta\mathbf{x}_n$ から，ノイズのない $\mathbf{x}_n$ を復元させるものである.

図7.5にDAEを示す.

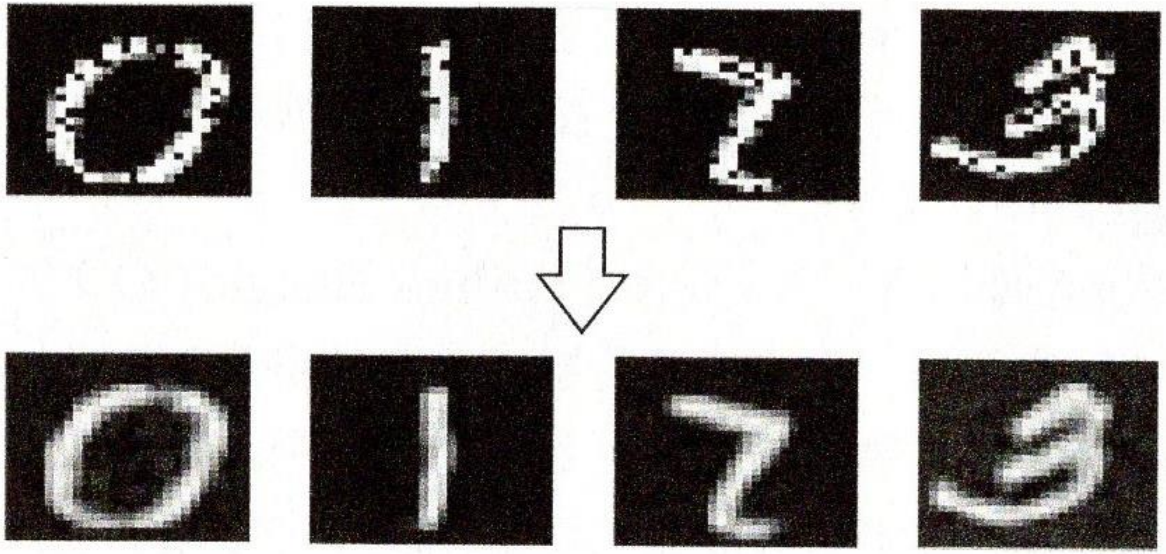


図7.5　欠落ノイズを加えた上段の画像をシンプルな単層DAEで処理した結果が下の画像.

## 7.6　収縮自己符号化器

### 7.6.1　収縮自己符号化器と多様体学習

省略

### 7.6.2　他の自己符号化器との関係

省略

# 飲み会の代わり…自由討論

分科会運営方法に関するコメント

最近の世相，時代の潮流

年寄りは無条件にモノづくりは良いもの、楽しいものと信じている？若者の考え方は？

本田宗一郎や松下幸之助の生き方に共感？

夢：遠隔地音楽アンサンブル

時間遅れをどう克服するか？

音楽合奏に関する鍋島さんの経験

ユニソンからアンサンブルへ

ノイズキャンセラーが悪さをする？

A, Bの2信号があると、一方が信号，他方がノイズになり、抑えられる。

夢：遠隔地音楽アンサンブル

ヤマハNETDUETTO（宮竹さんからの情報）

現行の通信回線でよい.

各パーツの時間遅れを極力小さくする.

ダウンロードの仕方
NETDUETTOラボ http://netduetto.net/manual/

７月４日（水）に前田，澤井，鍋島の諸氏と一色で WiFi ではなくて，有線接続で実験しました.

音の強弱，テンポが安定して，ようやく合奏出来る可能性が出てきました. 遅れ時間80ms.

新井先生から，オーディオ･インターフェースの仕組みのご報告をいただけるのを期待しています.