パターン認識と学習
教師あり学習(2)

管理工学科
篠沢佳久

資料の内容①

- 教師あり学習(2)
  - 線形識別関数の学習
  - デルタルール(Widrow-Hoffの学習規則)
  - パーセプトロン
- 教師なし学習
  - 固有ベクトルの学習
  - ヘッブの学習則(Generalized Hebbian Algorithm)

資料の内容②

- 実習①（デルタルール）
- 実習②（ヘッブの学習則）

線形判別関数の学習

デルタルール

線形識別関数

- クラスω_i（i=1,2,・・・,c）
- 各クラスに対応した関数 g_i
- 特徴ベクトル**x**（d次元）の属するクラスを調べる

$$\boxed{\max_{i=1,2\cdots,c}\{g_i(\mathbf{x})\} = g_k(\mathbf{x}) \Rightarrow \mathbf{x} \in \omega_k}$$

$$\mathbf{x} = (1, x_1, \cdots, x_d)^t$$

$$\mathbf{w}_i = (w_{i0}, w_{i1}, \cdots, w_{id})^t$$

$$g_i(\mathbf{x}) = \mathbf{w}^t\mathbf{x}$$

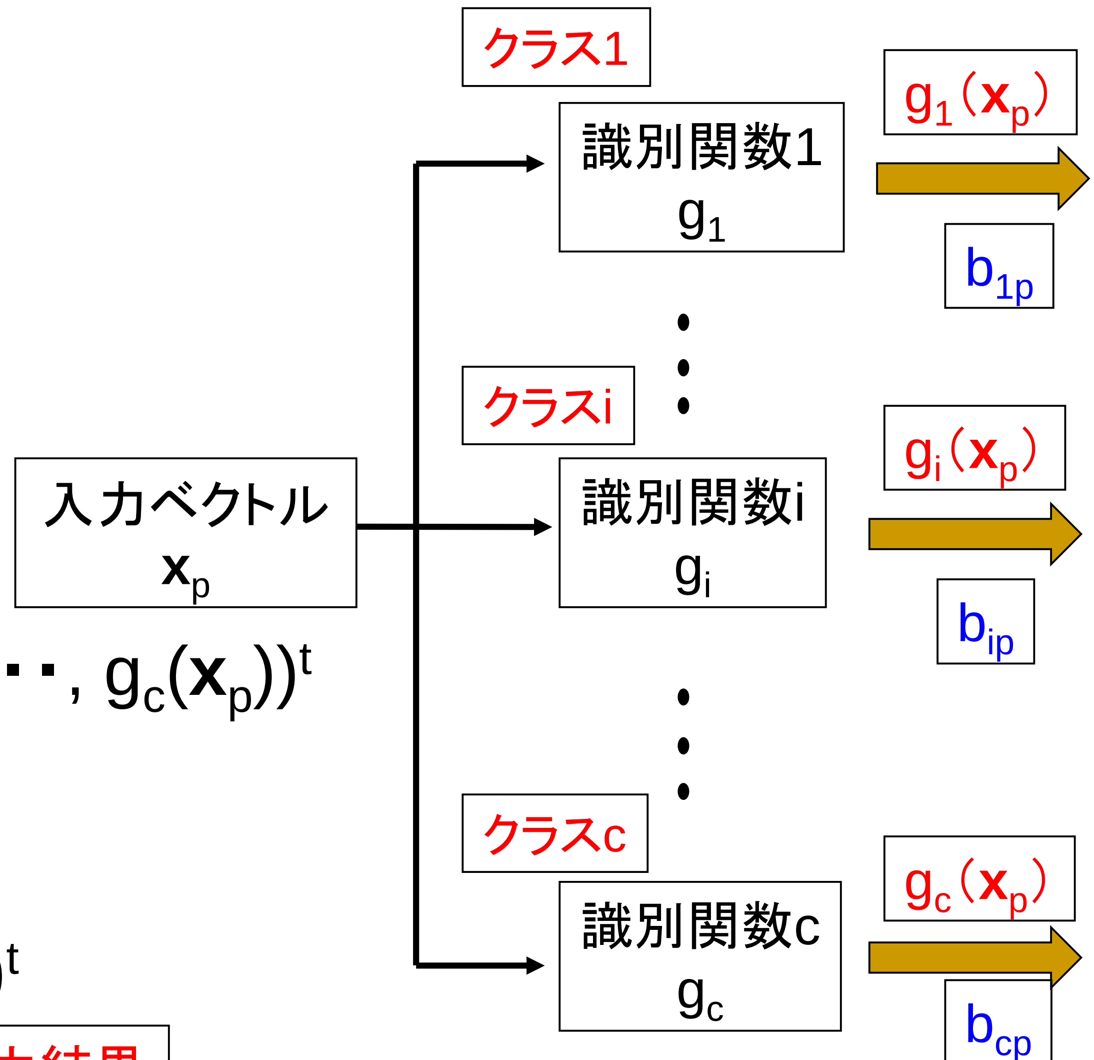
教師信号(ベクトル)①

- 学習パターンχ
  - $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$
  - 入力ベクトル$\mathbf{x}_p$

出力

  - $(g_1(\mathbf{x}_p), g_2(\mathbf{x}_p), \cdots, g_c(\mathbf{x}_p))^t$
  - $(b_{1p}, b_{2p}, \cdots, b_{cp})^t$

望ましい出力結果



教師信号(ベクトル)②

- 教師信号
  - 出力 $g_i(\mathbf{x}_p)$ に対して望ましい出力 b_{ip}

$$\mathbf{x}_p \in \omega_i \Rightarrow b_{ip} > b_{jp}$$

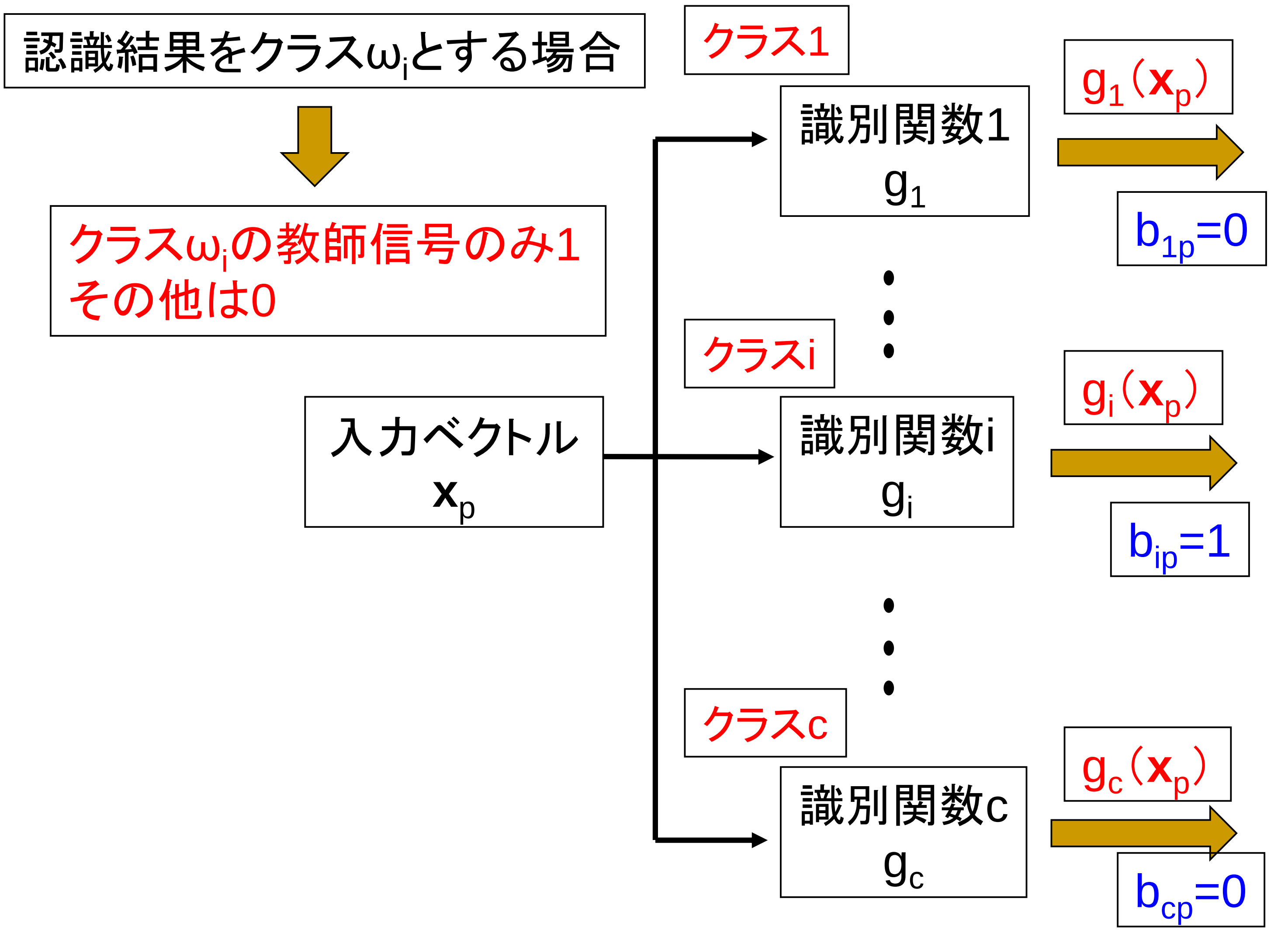
- 教師信号ベクトル
  - $(b_{1p}, b_{2p}, \cdots, b_{cp})^t$
  - (例)

i番目の要素

$$\mathbf{x}_p \in \omega_i \Rightarrow (0,0,\ldots,0,1,0,\cdots,0)^t$$

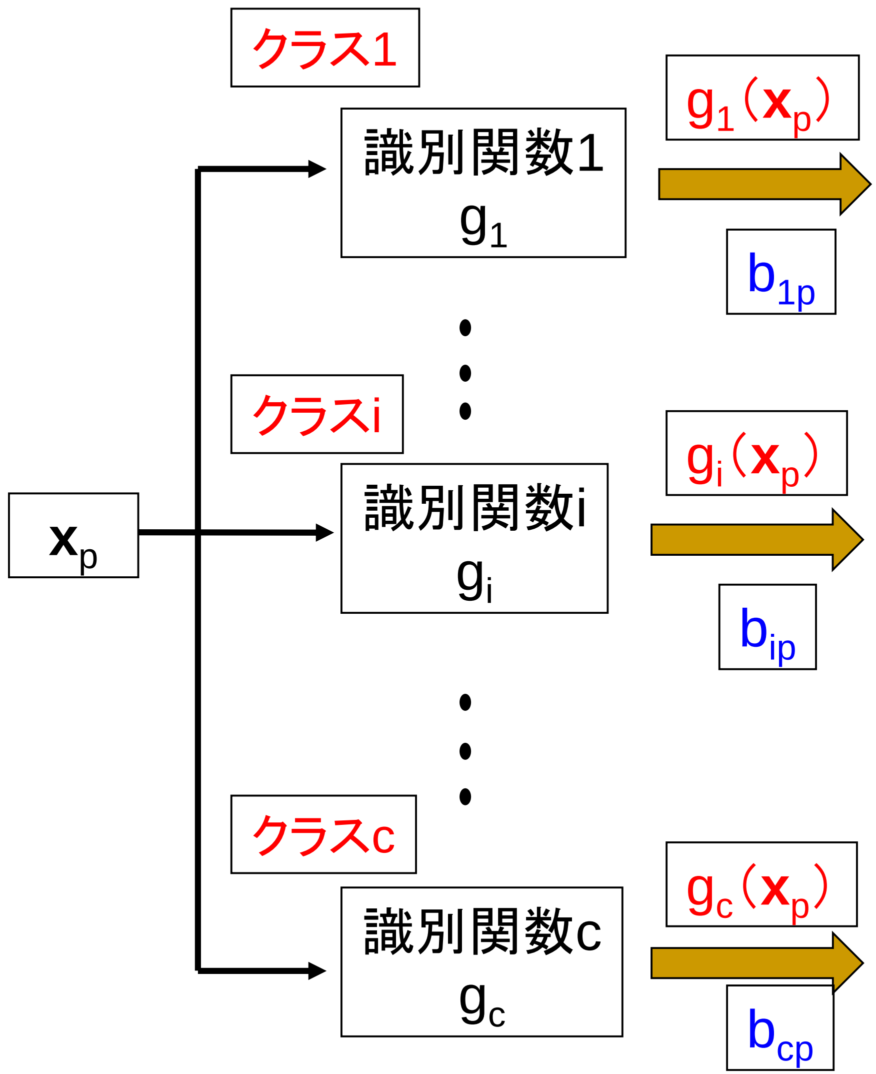
  - $b_{ip}=1$, 他は0とする

教師信号（ベクトル）の一例



重みの決め方①

全ての学習パターンにおいて，教師ベクトルと同じように出力されることが望ましい



学習パターン$\mathbf{x}_p$に対する各識別関数での教師信号との誤差

$$\varepsilon_{ip} = g_i(\mathbf{x}_p) - b_{ip}$$

全クラスでの誤差の自乗和

$$J_p = \sum_{i=1}^{c} \varepsilon_{ip}^2$$

全ての学習パターンの誤差の自乗和

$$J = \sum_{p=1}^{n} J_p = \sum_{p=1}^{n} \sum_{i=1}^{c} \varepsilon_{ip}^2$$

重みの決定②

- 全ての学習パターンの誤差の自乗和が最小となるように重み係数を決定

$$J=\sum_{p=1}^{n}\sum_{i=1}^{c}\varepsilon_{ip}^{2}=\sum_{p=1}^{n}\sum_{i=1}^{c}(g_i(\mathbf{x}_p)-b_{ip})^2$$

最小

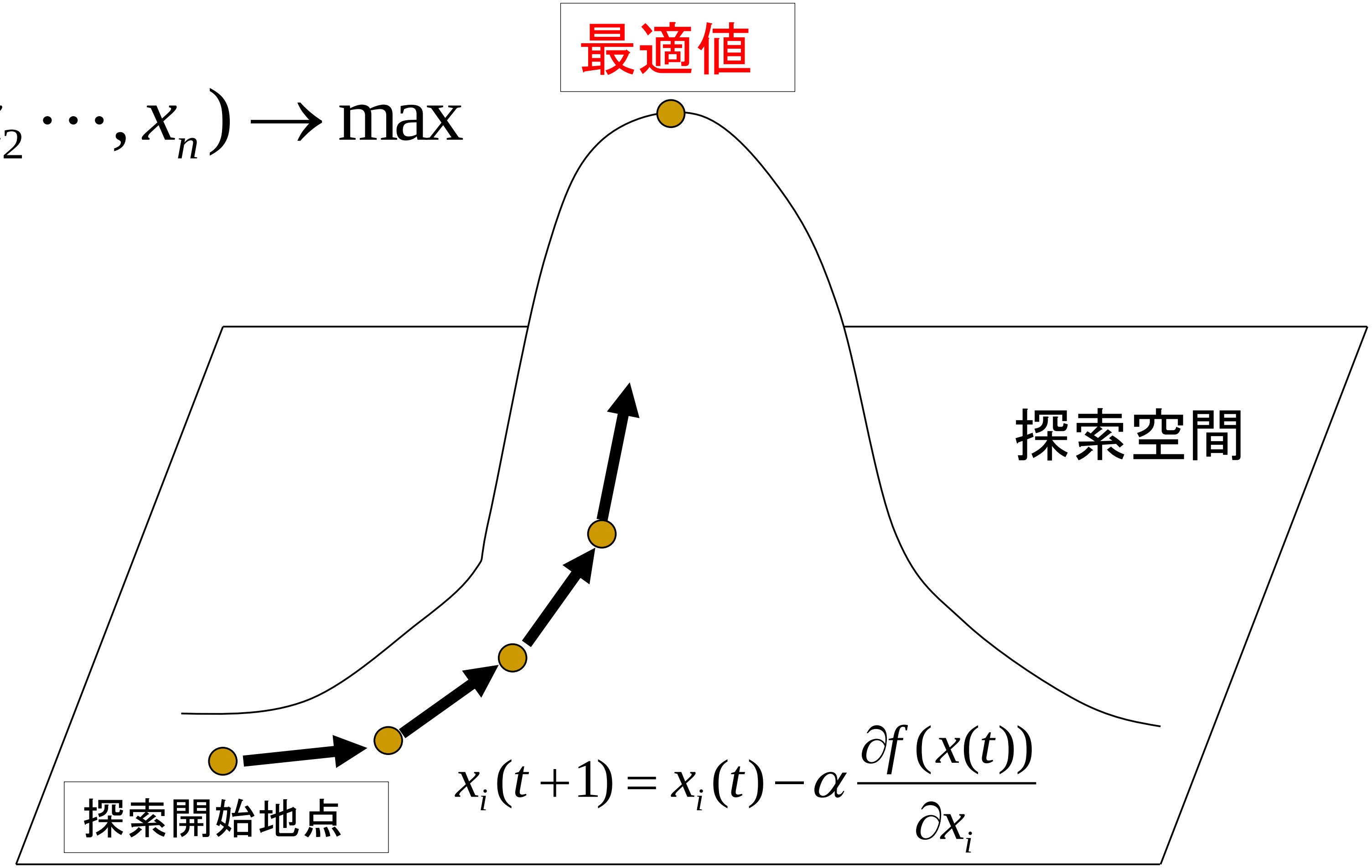
各パターンごとの誤差の自乗和

$$J_p=\sum_{i=1}^{c}\varepsilon_{ip}^{2}=\sum_{i=1}^{c}(g_i(\mathbf{x}_p)-b_{ip})^2$$

最小

最急降下法

$$f(x_1, x_2 \cdots, x_n) \to \max$$



$$x_i(t+1) = x_i(t) - \alpha \frac{\partial f(x(t))}{\partial x_i}$$

α：学習係数（0<α<1）

最急降下法による重みの決定

- 各学習パターンx_pが示される度に重みを修正

$$J_p = \sum_{i=1}^{c} \varepsilon_{ip}^2 = \sum_{i=1}^{c} (g_i(\mathbf{x}_p) - b_{ip})^2$$

$$\mathbf{w}_i' = \mathbf{w}_i - \alpha \frac{\partial J_p}{\partial \mathbf{w}_i}$$

$$\boxed{\frac{\partial J_p}{\partial \mathbf{w}_i} = \frac{\partial J_p}{\partial g_i(\mathbf{x}_p)} \frac{\partial g_i(\mathbf{x}_p)}{\partial \mathbf{w}_i}}$$

$$\frac{\partial J_p}{\partial g_i(\mathbf{x}_p)} = 2(g_i(\mathbf{x}_p) - b_{ip})$$

$$\frac{\partial g_i(\mathbf{x}_p)}{\partial \mathbf{w}_i} = \frac{\partial(\mathbf{w}_i^t \mathbf{x}_p)}{\partial \mathbf{w}_i} = \mathbf{x}_p$$

$$\Downarrow$$

$$\mathbf{w}_i' = \mathbf{w}_i - \alpha(g_i(\mathbf{x}_p) - b_{ip})\mathbf{x}_p$$

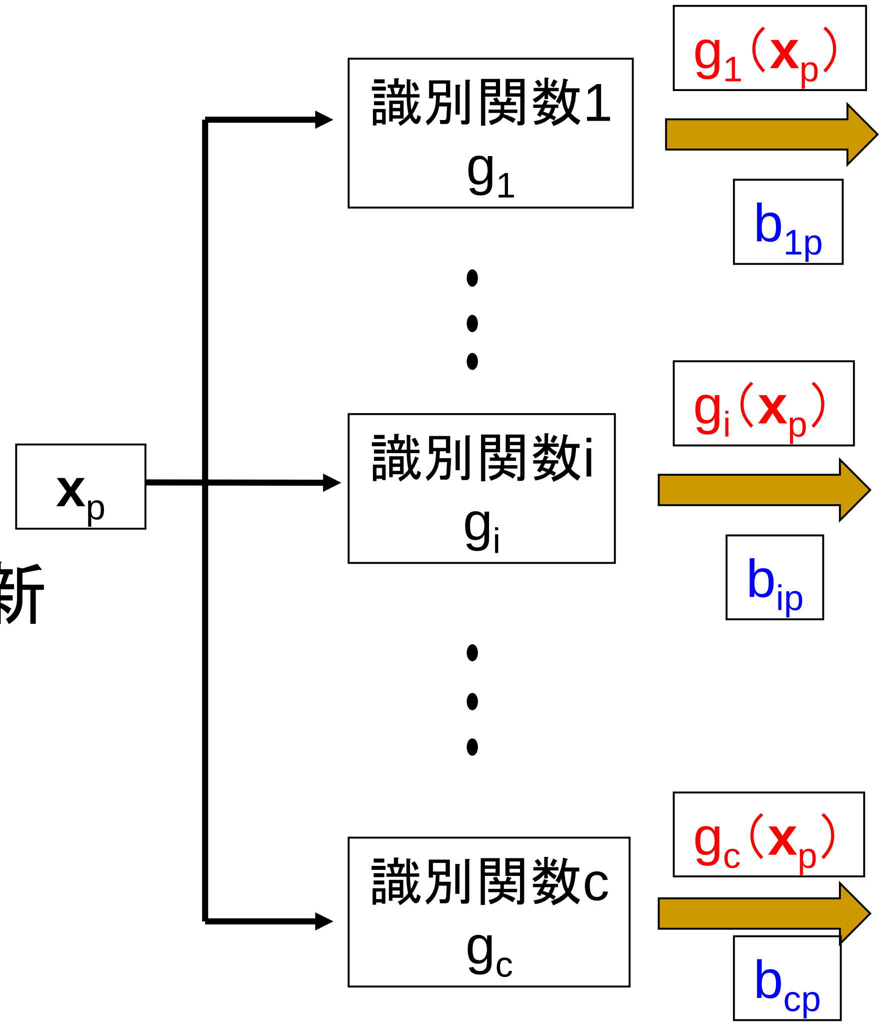
$$= \mathbf{w}_i - \alpha(\underbrace{\mathbf{w}_i^t \mathbf{x}_p - b_{ip}}_{\text{間違えた量}})\underbrace{\mathbf{x}_p}_{\text{入力ベクトル}}$$

デルタルール*による重みの決定

- 重みの更新方法

$$\mathbf{w}_i^{'} = \mathbf{w}_i - \alpha(g_i(\mathbf{x}_p) - b_{ip})\mathbf{x}_p$$
$$= \mathbf{w}_i - \alpha(\mathbf{w}_i^t\mathbf{x}_p - b_{ip})\mathbf{x}_p$$

  - 全ての$\mathbf{w}_i$において更新
  - (i=1,2,･･･,c)



*Widrow-Hoffの学習規則（ADALINEと呼ばれるニューラルネットワークの学習規則）とも呼ばれる

デルタールールのアルゴリズム

```
while() {
    Error = 0
    for( p = 0 ; p < n ; p++ ) {
        for( i = 0 ; i < c ; i++ ) {
            g_i(x_p) の計算
            e_ip = ( g_i(x_p) – b_ip )
            for( j = 0 ; j < d ; j++ ){
                w_ij -= alpha * e_ip * x_pj
            }
            Error += e_ip * e_ip
        }
    }
    if( Error < ε ) break
}
```

重みベクトルは乱数によって事前に初期化しておく

誤差の計算

重みの修正

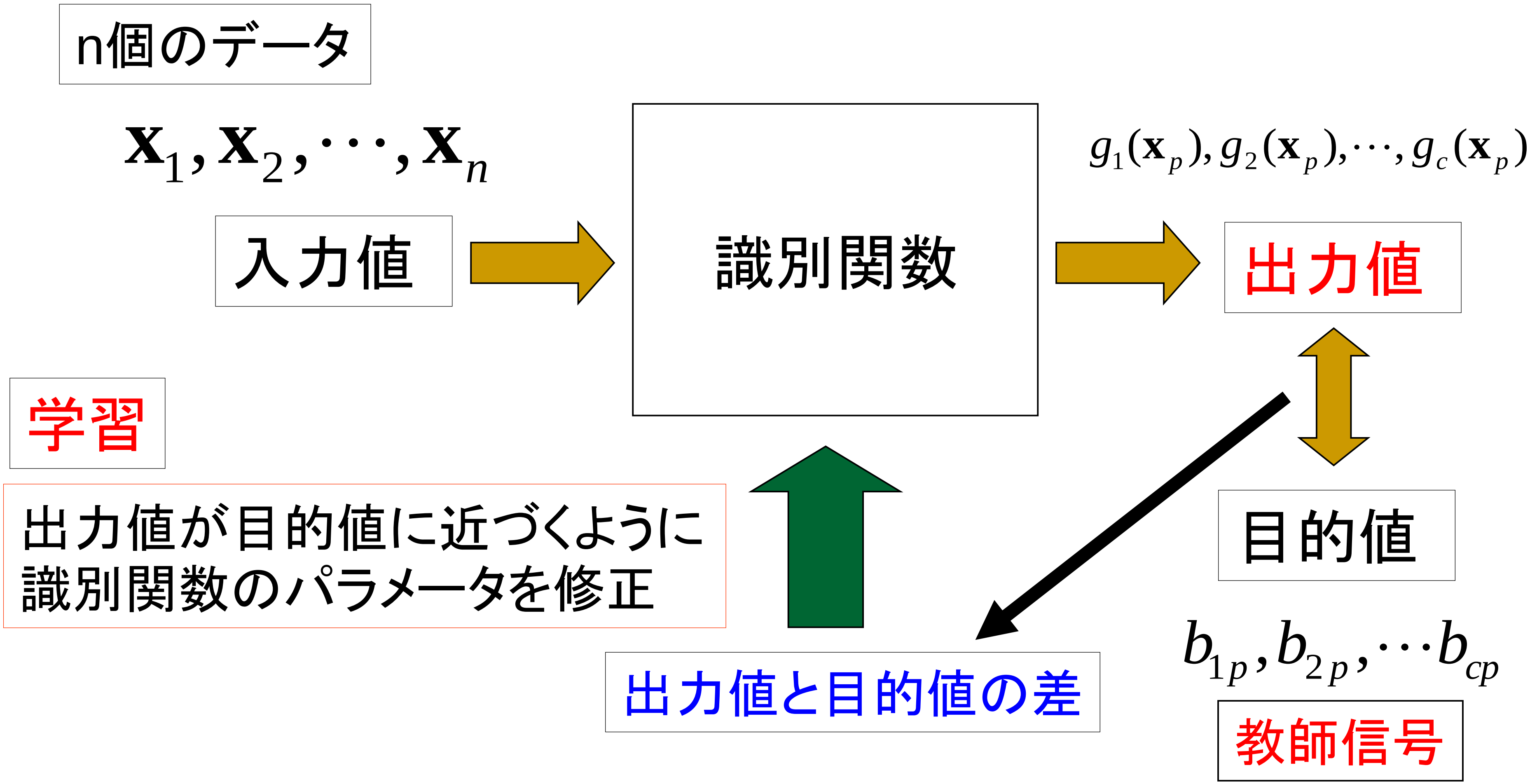
c ： クラスの総数　　　d ： 次元数

w_{ij} ： クラス i の識別関数の j番目の重み係数

x_{pj} ： 特徴ベクトル$\mathbf{x}_p$の j番目の要素

誤差自乗和が一定値以下となったら停止

学習による識別関数の構築



パーセプトロン

デルタルールの改良

線形分離

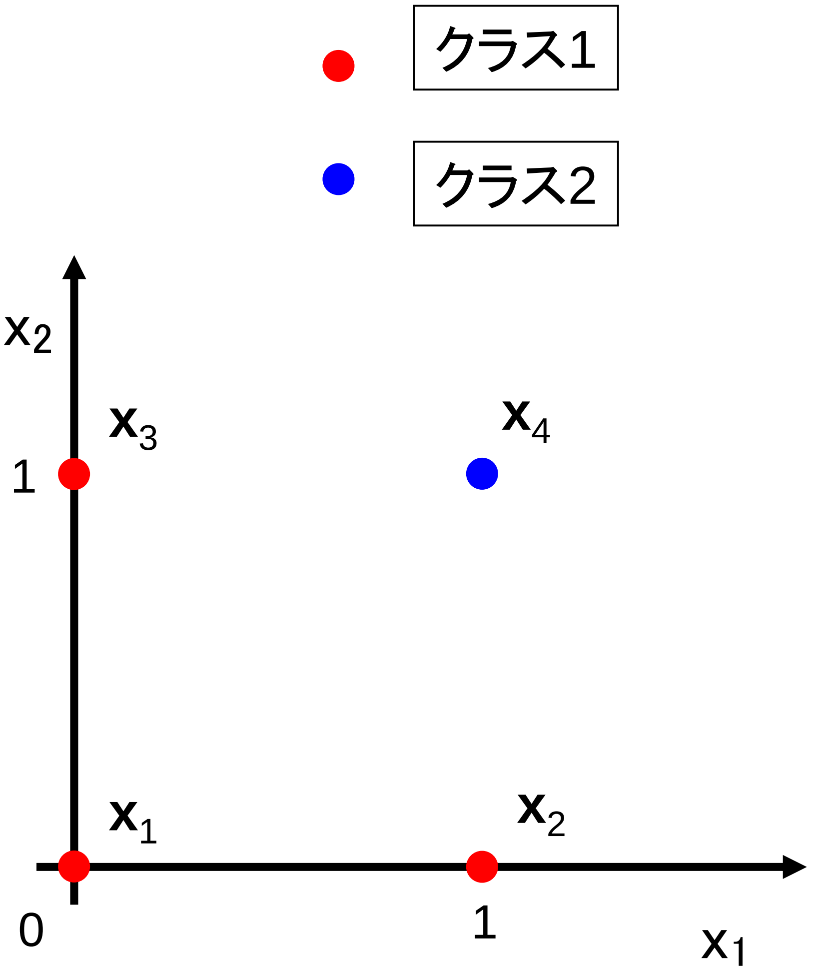
- クラスω_i（i=1,2,･･･,c）
- クラスω_iの学習パターンの集合χ_i
- χ_i に属する全ての学習パターン**x**

$$g_i(\mathbf{x}) > g_j(\mathbf{x}) \quad (j = 1,2,\cdots,c \quad j \neq i)$$

- 上記の式を満たす重みベクトル**w**が存在する場合，線形分離可能と呼ぶ

線形分離可能

	x_1	x_2	
$\mathbf{x}_1$	0	0	クラス1
$\mathbf{x}_2$	1	0	クラス1
$\mathbf{x}_3$	0	1	クラス1
$\mathbf{x}_4$	1	1	クラス2



数値例①

	x_1	x_2	
$\mathbf{x}_1$	0	0	クラス1
$\mathbf{x}_2$	1	0	クラス1
$\mathbf{x}_3$	0	1	クラス1
$\mathbf{x}_4$	1	1	クラス2

識別関数1

g_1

$w_{10}=1.25$
$w_{11}=-0.50$
$w_{12}=-0.50$

識別関数2

g_2

$w_{10}=-0.25$
$w_{11}=0.50$
$w_{12}=0.50$

識別関数

$$g_i(\mathbf{x}) = w_{i0} + \sum_{j=1}^{2} w_{ij} x_j$$

$\mathbf{x}_1=(0,0)^t$

$g_1(\mathbf{x}_1)=1.25 \;>\; g_2(\mathbf{x}_1)=-0.25$

$\mathbf{x}_2=(1,0)^t$

$g_1(\mathbf{x}_2)=0.75 \;>\; g_2(\mathbf{x}_2)=0.25$

数値例②

	x_1	x_2	
$\mathbf{x}_1$	0	0	クラス1
$\mathbf{x}_2$	1	0	クラス1
$\mathbf{x}_3$	0	1	クラス1
$\mathbf{x}_4$	1	1	クラス2

識別関数1
g_1

$w_{10}=1.25$
$w_{11}=-0.50$
$w_{12}=-0.50$

識別関数2
g_2

$w_{10}=-0.25$
$w_{11}=0.50$
$w_{12}=0.50$

識別関数

$$g_i(\mathbf{x}) = w_{i0} + \sum_{j=1}^{2} w_{ij} x_j$$

$\mathbf{x}_3=(0,1)^t$

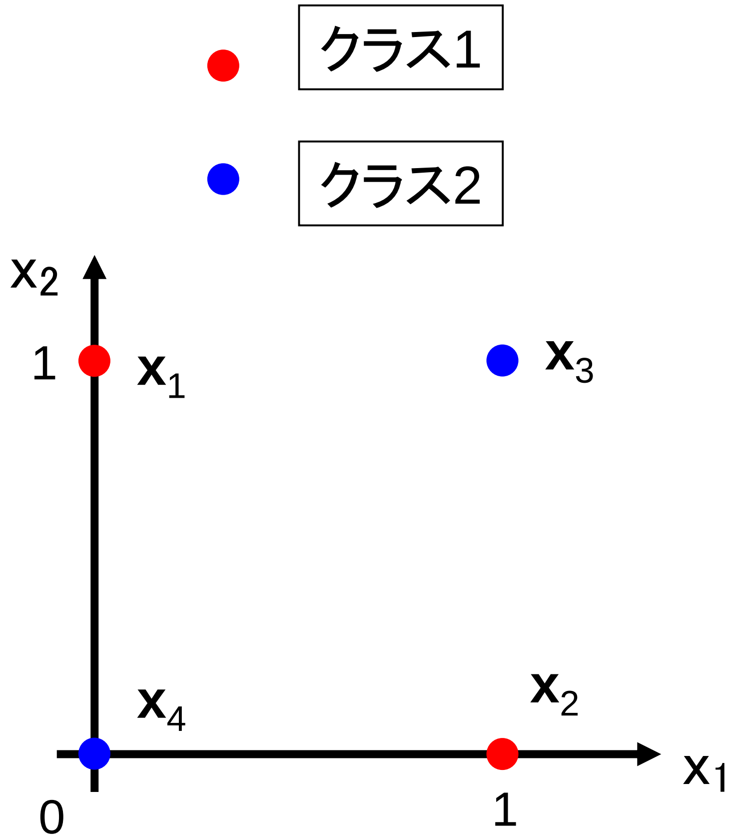
$g_1(\mathbf{x}_3)=0.75 > g_2(\mathbf{x}_3)=0.25$

$\mathbf{x}_4=(1,1)^t$

$g_1(\mathbf{x}_4)=0.25 < g_2(\mathbf{x}_4)=0.75$

線形分離不可能

	x_1	x_2	
$\mathbf{x}_1$	0	1	クラス1
$\mathbf{x}_2$	1	0	クラス1
$\mathbf{x}_3$	1	1	クラス2
$\mathbf{x}_4$	0	0	クラス2



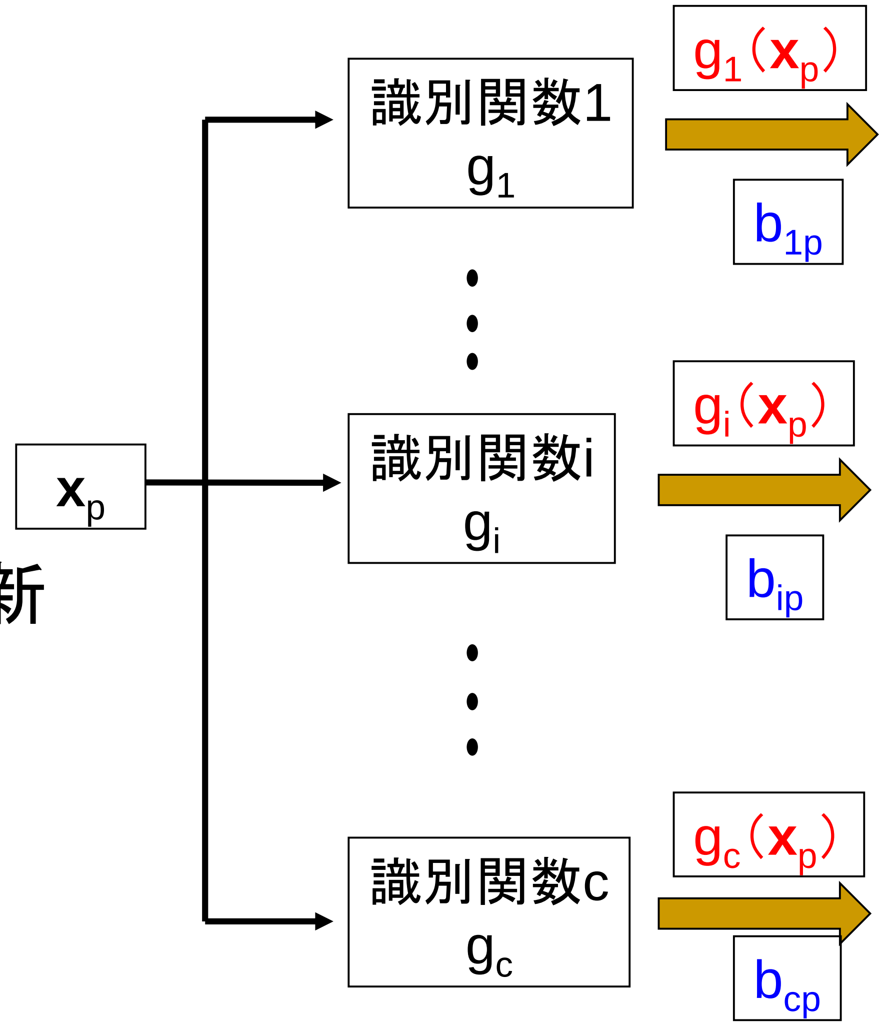
クラス1とクラス2を識別できる重みは存在しない
→ 線形分離不可能

デルタルールによる重みの決定（復習）

- 重みの更新方法

$$\mathbf{w}_i' = \mathbf{w}_i - \alpha(g_i(\mathbf{x}_p) - b_{ip})\mathbf{x}_p$$
$$= \mathbf{w}_i - \alpha(\mathbf{w}_i^t \mathbf{x}_p - b_{ip})\mathbf{x}_p$$

  - 全ての$\mathbf{w}_i$において更新
  - （i=1,2,・・・,c）



閾値関数の導入①

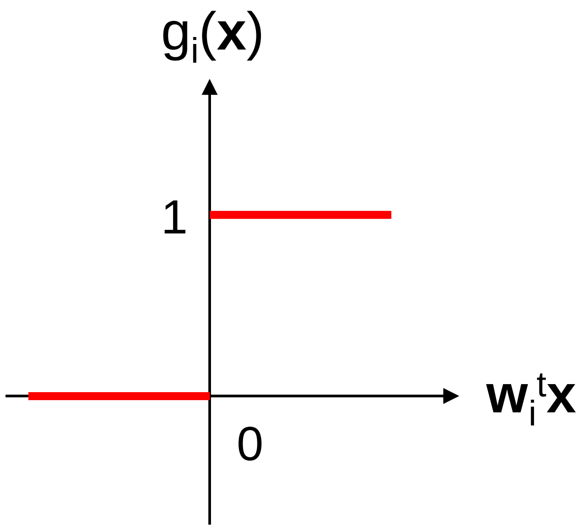
- 識別関数g_iの重みベクトル$\mathbf{w}_i$
  - 以下の式を満たすように学習する

$$\begin{cases} \mathbf{w}_i^t\mathbf{x} > 0 & (x \in \omega_i) \\ \mathbf{w}_i^t\mathbf{x} < 0 & (x \notin \omega_i) \end{cases}$$

$$i = 1, 2, \cdots, c$$

- 閾値関数

$$\boxed{g_i(\mathbf{x}) = T_i(\mathbf{w}_i^t\mathbf{x}) = \begin{cases} 1 & \mathbf{w}_i^t\mathbf{x} > 0 \\ 0 & \mathbf{w}_i^t\mathbf{x} < 0 \end{cases}}$$



閾値関数の導入②

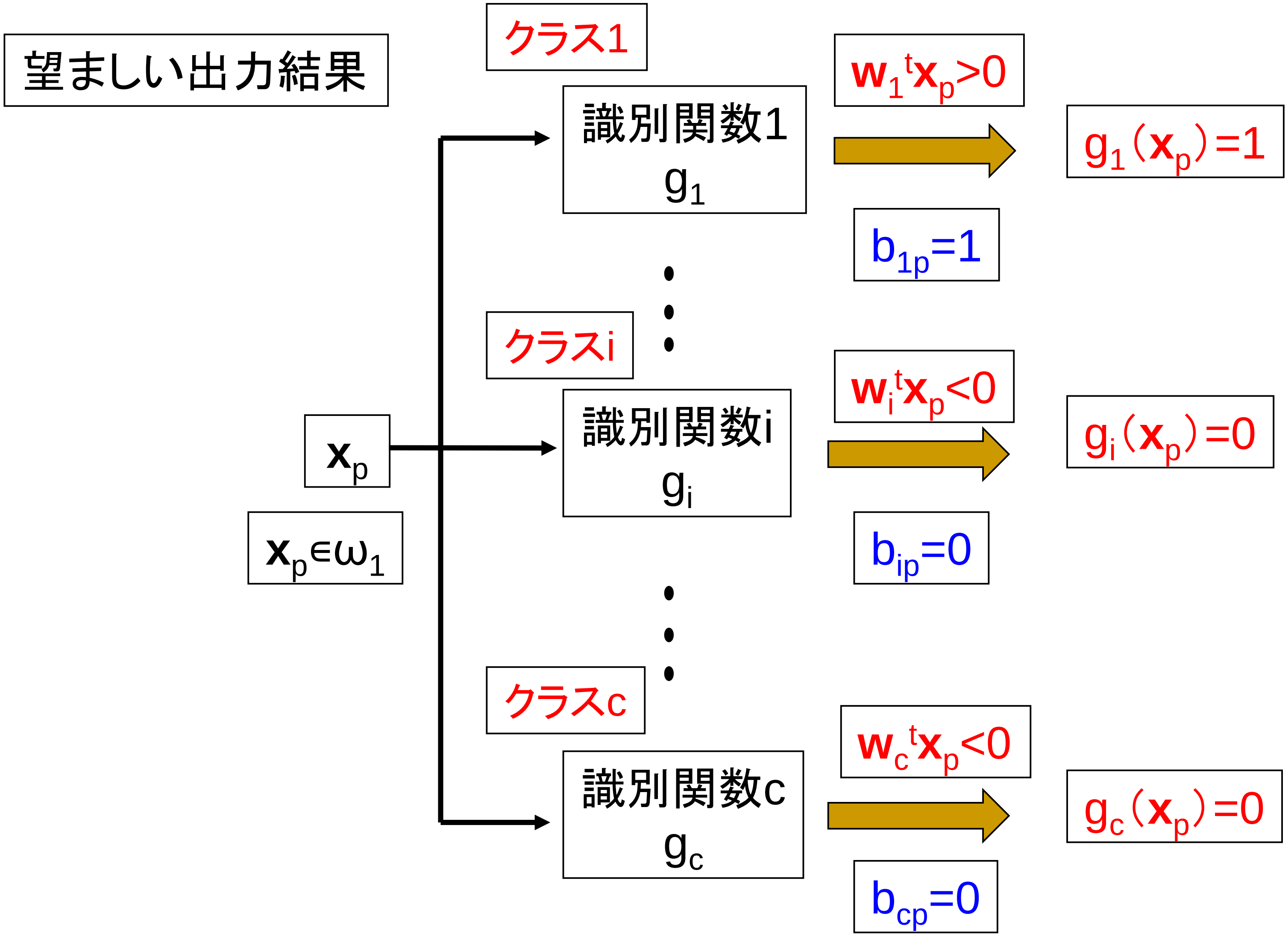
- 識別関数の出力値

$$\begin{cases} g_i(\mathbf{x}) = 1 & (x \in \omega_i) \\ g_i(\mathbf{x}) = 0 & (x \notin \omega_i) \end{cases}$$

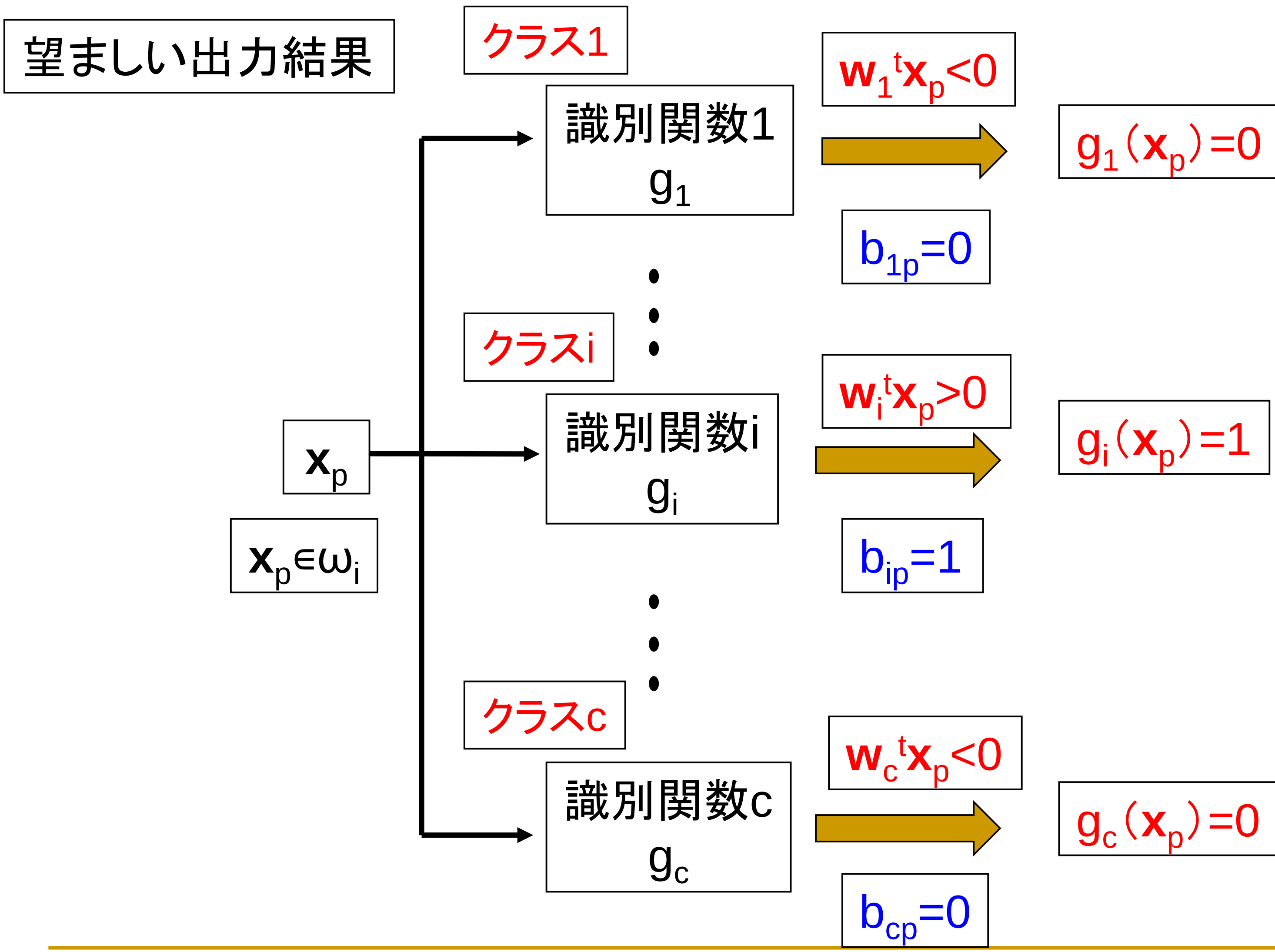
- 教師信号

$$\begin{cases} b_{ip} = 1 & (x \in \omega_i) \\ b_{ip} = 0 & (x \notin \omega_i) \end{cases}$$

閾値関数を導入した場合①



閾値関数を導入した場合②



デルタルールの変更①

- デルタルール

$$\boxed{\mathbf{w}_i^{'} = \mathbf{w}_i - \alpha(g_i(\mathbf{x}_p) - b_{ip})\mathbf{x}_p}$$

- 識別関数g_i
  - $\mathbf{x}_p \in \omega_i$ をω_iと正しく識別した場合

$$\boxed{g_i(\mathbf{x}_p) = 1 \quad b_{ip} = 1} \Rightarrow \boxed{\mathbf{w}_i^{'} = \mathbf{w}_i}$$

  - $\mathbf{x}_p \in \omega_i$ をω_iではないと誤って識別した場合

$$\boxed{g_i(\mathbf{x}_p) = 0 \quad b_{ip} = 1} \Rightarrow \boxed{\mathbf{w}_i^{'} = \mathbf{w}_i + \alpha \mathbf{x}_p}$$

デルタルールの変更②

- デルタールール

$$\boxed{\mathbf{w}_i^{'} = \mathbf{w}_i - \alpha(g_i(\mathbf{x}_p) - b_{ip})\mathbf{x}_p}$$

- 識別関数g_j(i≠j)
  - $\mathbf{x}_p \in \omega_i$ をω_jと誤って識別した場合

$$\boxed{g_j(\mathbf{x}_p) = 1 \quad b_{jp} = 0} \Rightarrow \boxed{\mathbf{w}_j^{'} = \mathbf{w}_j - \alpha \mathbf{x}_p}$$

  - $\mathbf{x}_p \in \omega_i$ をω_jではないと正しく識別した場合

$$\boxed{g_j(\mathbf{x}_p) = 0 \quad b_{jp} = 0} \Rightarrow \boxed{\mathbf{w}_j^{'} = \mathbf{w}_j}$$

重みベクトルの更新方法

- 識別関数g_iにおいて，$\mathbf{x}_p$をω_iと認識しなければならないのに，ω_iではないと認識してしまった場合

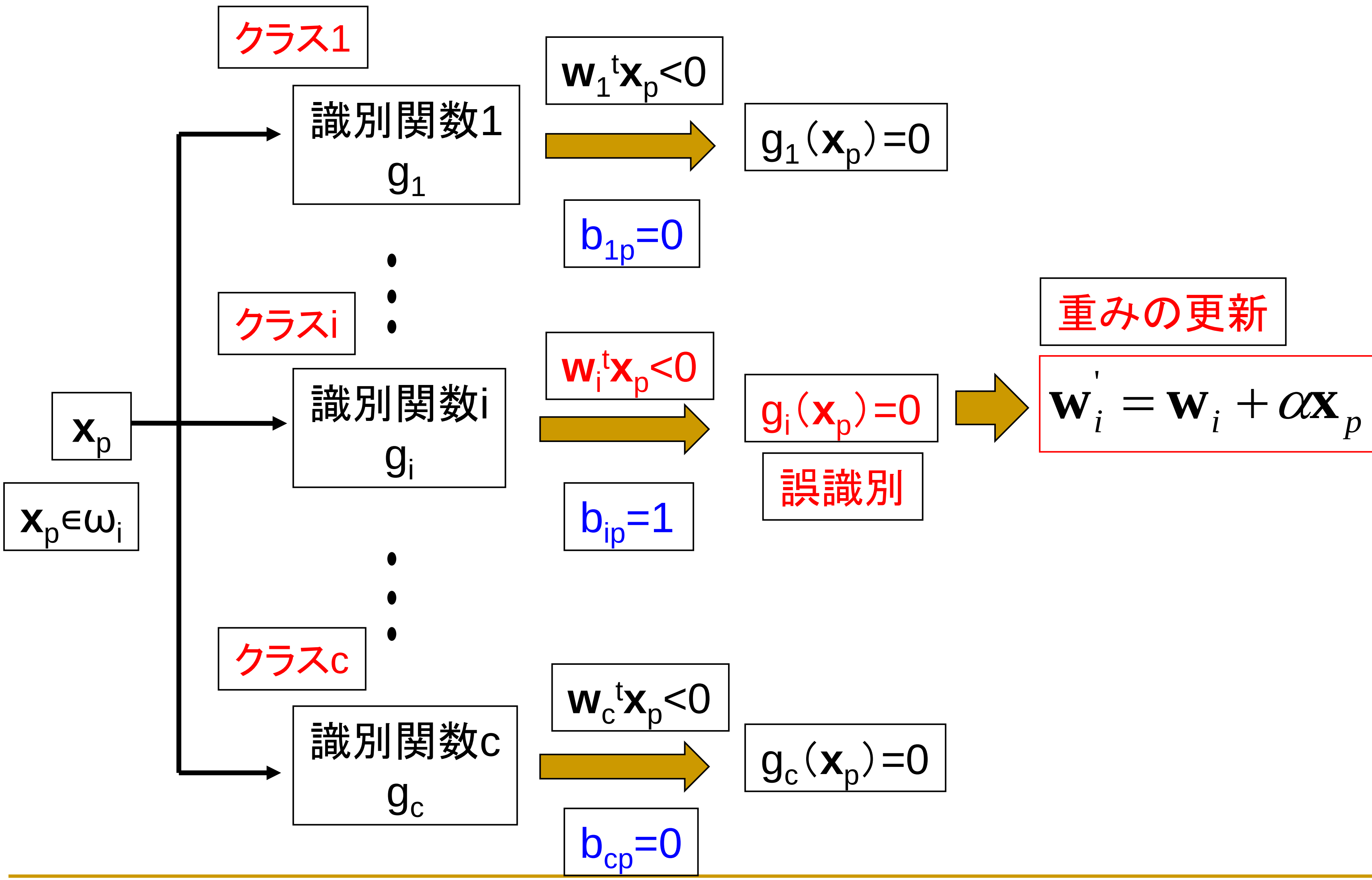
$$\boxed{g_i(\mathbf{x}_p)=0 \quad b_{ip}=1} \Rightarrow \boxed{\mathbf{w}_i^{'}=\mathbf{w}_i+\alpha\mathbf{x}_p}$$

- 識別関数g_jにおいて，$\mathbf{x}_p$をω_j と認識してはいけないのに，ω_j と認識してしまった場合

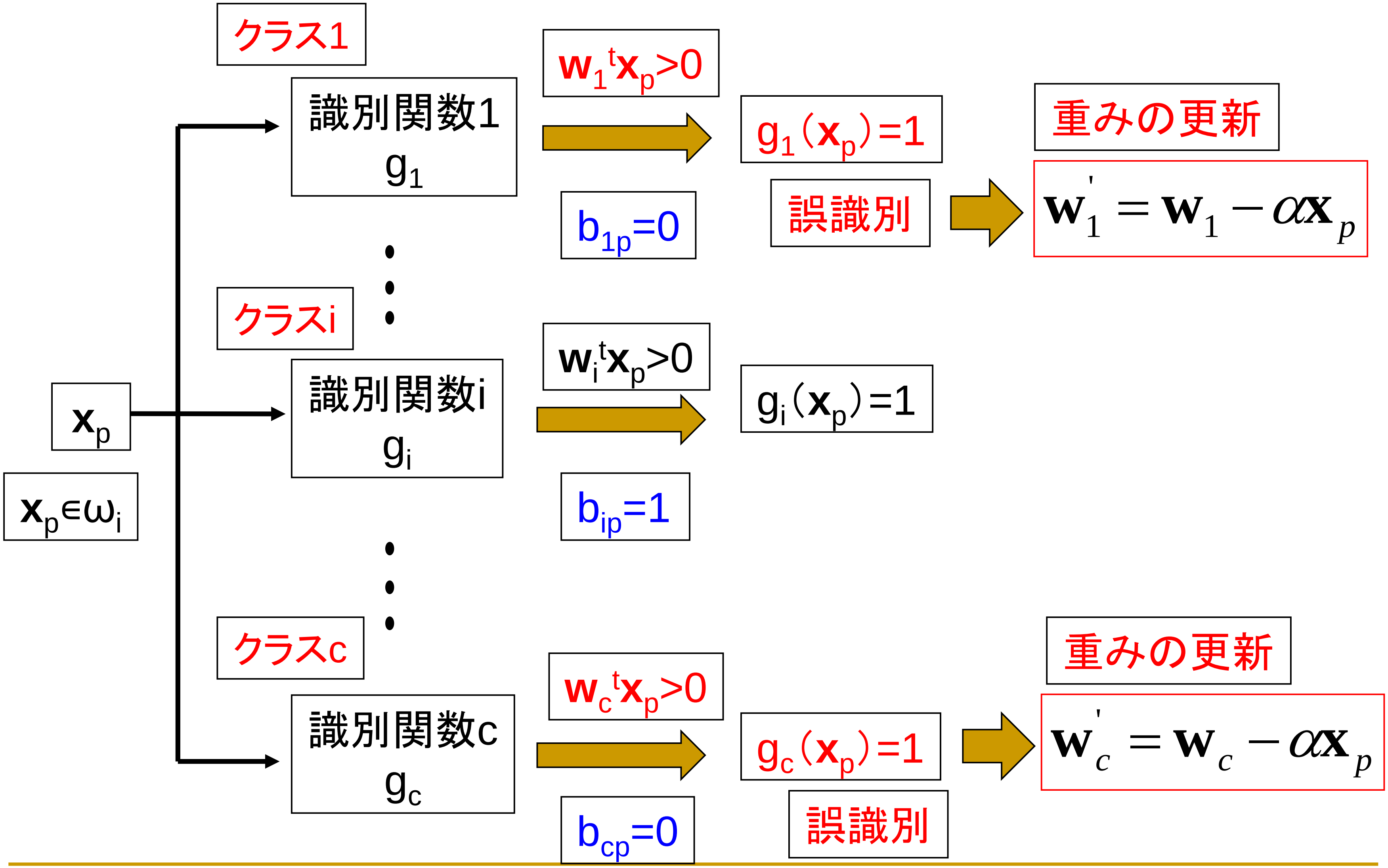
$$\boxed{g_j(\mathbf{x}_p)=1 \quad b_{jp}=0} \Rightarrow \boxed{\mathbf{w}_j^{'}=\mathbf{w}_j-\alpha\mathbf{x}_p}$$

パーセプトロンの学習規則

パーセプトロンの学習規則①



パーセプトロンの学習規則②



パーセプトロンの学習規則

1. 重みベクトル$\mathbf{w}_i$を乱数にて初期化
   - クラス数はc個（i=1,2,･･･,c）
2. 学習パターン$\mathbf{x}_p$ を選択
3. 全ての $g_i(\mathbf{x})$ を計算，正しく判別できなかった識別関数の重みベクトル$\mathbf{w}_i$を修正
   - $\mathbf{x}_p$をω_i と認識しなければならないのに，ω_iではないと認識してしまった場合→ $\mathbf{w}_i' = \mathbf{w}_i + \rho\mathbf{x}$
   - $\mathbf{x}_p$をω_i と認識してはいけないのに，ω_i と認識してしまった場合→ $\mathbf{w}_i' = \mathbf{w}_i - \rho\mathbf{x}$
4. 全ての学習パターンについて，正しく判別できるまで，2と3を繰り返す

パーセプトロン①

- Frank Rosenblatt, 1957
- 線形判別関数における重み係数の学習アルゴリズム
- 線形分離可能な問題であれば，有限回の繰り返しで，解領域の重みを求めることが可能（パーセプトロンの収束定理）

パーセプトロンの収束定理

- 線形分離可能な問題のみに対応
  - 重み空間上に解領域が存在する
- パーセプトロンによる重みの更新
  - 学習係数ρが適切であれば，誤って識別した場合，g(**x**) の符号を反転させることが可能
  - 有限回の繰り返し回数にて解領域に達することが可能

パーセプトロン②

- パーセプトロンの学習規則
  - 誤識別をした場合のみ，修正
  - 誤り訂正法と呼ばれる

- ニューラルネットワーク（人工的神経回路網）の代表的なモデルの一つ

- 問題点
  - 線形分離可能な問題のみ対応
  - 線形分離可能かどうかを調べることは困難
  - 学習した重みベクトルによって，未知データの識別が可能かどうか

パーセプトロン③

- Marvin Minsky
  - 線形分離不可能な問題には対応できないことを指摘（1969）
- 誤差逆伝播則*
  - Error Back propagation Algorithm（1986）
  - David Rumelhart，Geoffrey Everest Hinton
  - パーセプトロンの学習アルゴリズムを改良
  - 線形分離不可能な問題に対応可能

*一般化デルタルールとも呼ばれる

教師なし学習

ヘッブの学習則

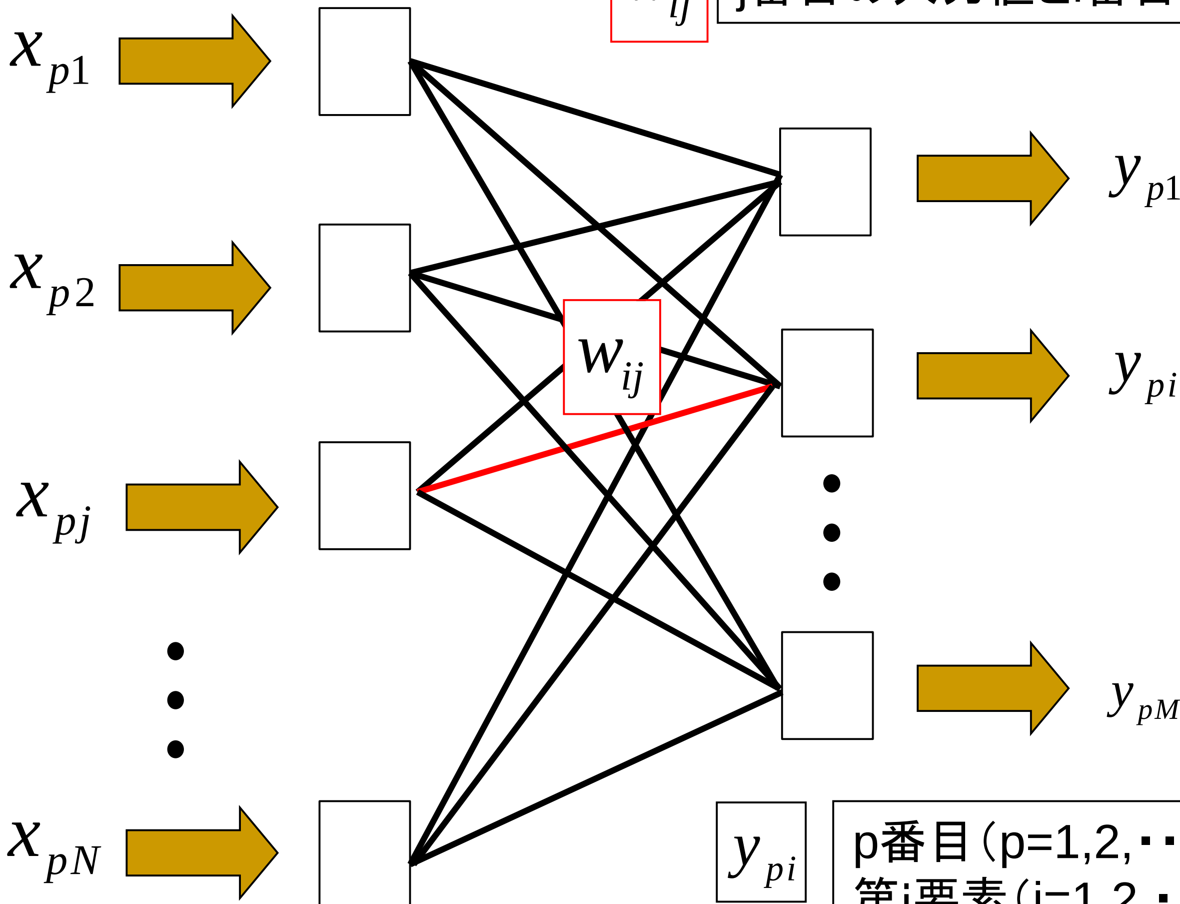
教師あり学習と教師なし学習

- 教師あり学習（Supervised Learning）
  - 教師信号を用いて学習
  - デルタルール
  - パーセプトロン

- 教師なし学習（Unsupervised Learning）
  - 教師信号を用いず学習
  - クラスタリング（EMアルゴリズム）
  - 主成分分析（ヘッブの学習則）

線形識別関数のグラフ化

w_{ij}　j番目の入力値とi番目の出力値の重み



$$y_{pi} = \sum_{j=1}^{N} w_{ij} x_{pj}$$

y_{pi}　p番目（p=1,2,･･･,P）のデータの第j要素（j=1,2,･･･,N）の出力値

x_{pj}　p番目（p=1,2,･･･,P）のデータの第j要素（j=1,2,･･･,N）の入力値

重みの学習

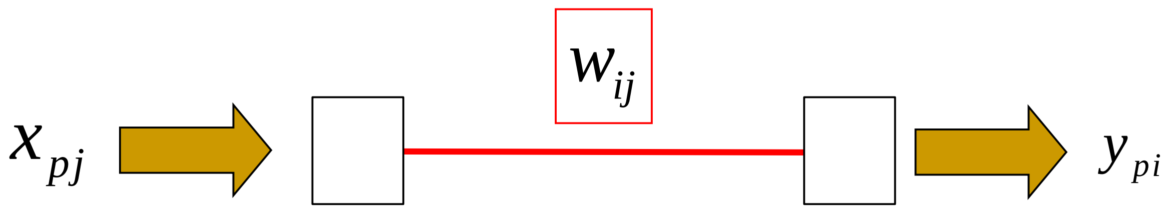
- デルタルール

教師信号を与える

$$w'_{ij} = w_{ij} - \alpha\left(\sum_{i=1}^{N} w_{ij} x_{pi} - \boxed{b_{pi}}\right) x_{pi}$$

- ヘッブの学習則

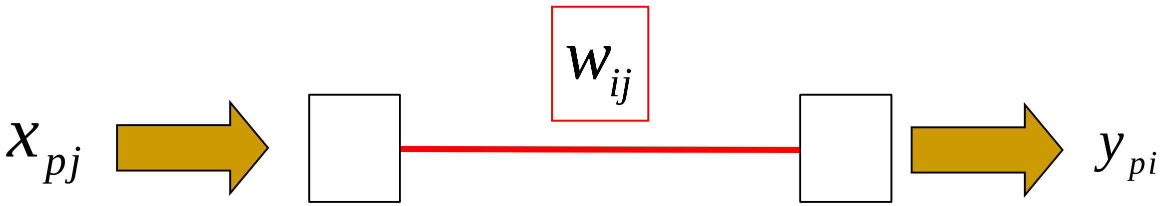
$$w'_{ij} = w_{ij} + \alpha x_{pj} y_{pi}$$



教師信号は与えない

ヘッブの学習則（Donald Hebb）

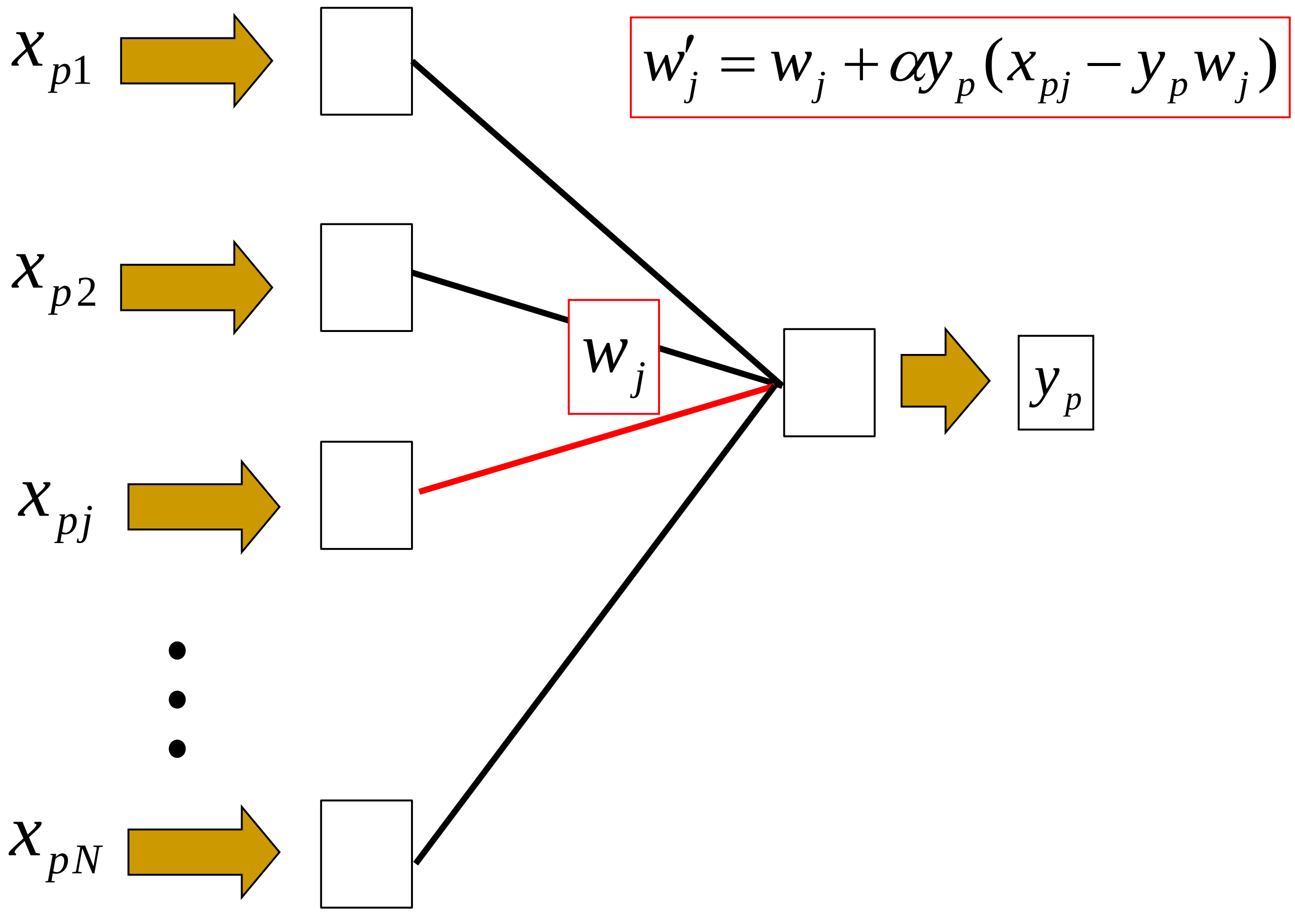
$$\boxed{w'_{ij} = w_{ij} + \alpha x_{pj} y_{pi}}$$



入力値	出力値	重み
＋	＋	強める
―	―	強める
＋	―	弱める
―	＋	弱める

Ojaの学習則

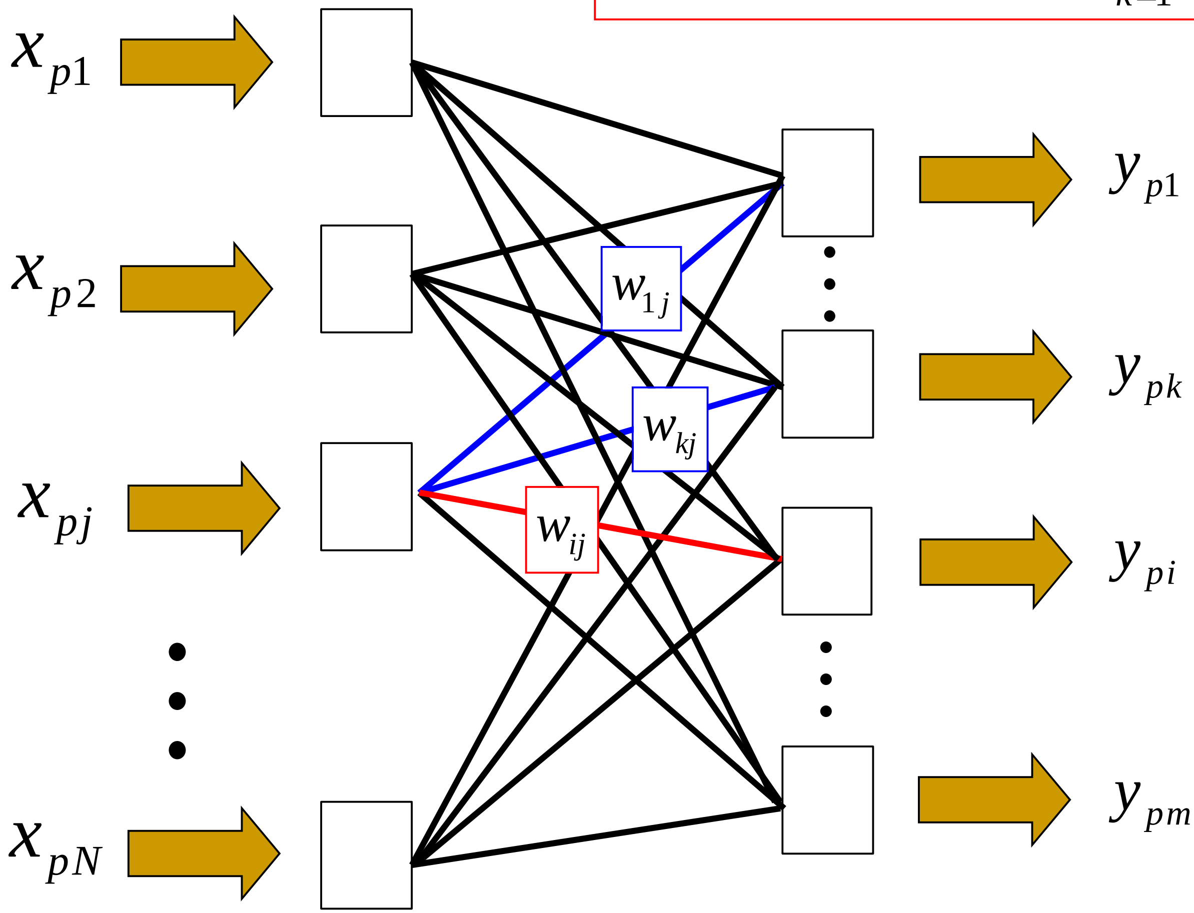
- 第一主成分の学習



Sangerの学習則*

- 第m主成分までの学習

$$w'_{ij} = w_{ij} + \alpha y_{pi}(x_{pj} - \sum_{k=1}^{i} y_{pk} w_{kj})$$



* 一般化ヘッブ学習則（Generalized Hebbian Algorithm）と呼ばれる

ヘッブの学習則

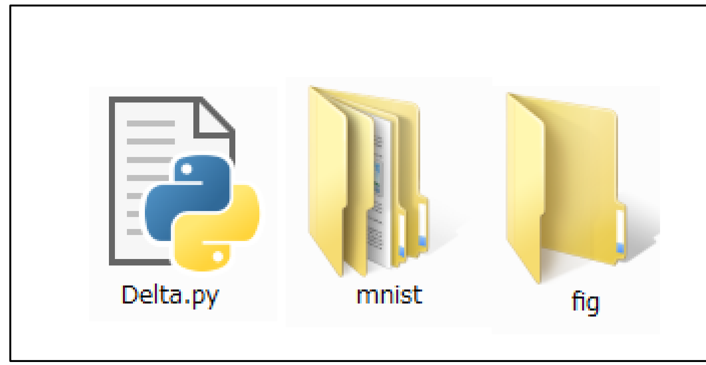
- シナプスの可塑性の法則
  - Donald Hebb, 1949
  - 神経細胞が興奮→つながる神経細胞も興奮→その間の伝達効率が強まる
  - 逆の場合，伝達効率が弱まる

実習①（デルタルール）

デルタルール（Delta.py）

- MNISTの数字画像認識

- MNISTのデータがあるフォルダーにプログラムは置いて下さい
- 「fig」という名前のフォルダーを作成して下さい



- 実行方法
  - 学習
  - > python Delta.py t
  - 認識
  - > python Delta.py p

引数をつけて下さい

Delta.pyの関数

- Read_train_data()
  - 学習データの読み込み
- Train()
  - 学習
- Load_Weight()
  - 重みの読み込み
- Predict()
  - テストデータの認識

変数の定義①

```
# クラス数
class_num = 10

# 画像の大きさ
size = 14

# 学習データ数
train_num = 100

# 学習データ
train_vec =
np.zeros((class_num,train_num,size*size+1), dtype=np.float64)
```

size×size+1
→ 閾値も必要

$$\mathbf{x} = (\boxed{1}, x_1, \cdots, x_d)^t$$

$$\mathbf{w}_i = (\boxed{w_{i0}}, w_{i1}, \cdots, w_{id})^t$$

$$g_i(\mathbf{x}) = \mathbf{w}^t \mathbf{x}$$

変数の定義②

重みは乱数(-0.5～0.5)によって初期化

```
# 重み
weight = np.random.uniform(-0.5,0.5,(class_num,size*size+1))
one = np.array([1])

# 学習回数，学習係数
LOOP = 100
alpha = 0.05
```

weight[i][j]
j番目の入力とクラスiとの重み

size×size+1
→ 閾値も必要

学習データの読み込み

Read_train_data

```
def Read_train_data():
    # 学習データの読み込み
    for i in range(class_num):
        for j in range(1,train_num+1):
            train_file = "mnist/train/" + str(i) + "/" + str(i) + "_" + str(j) + ".jpg"
            work_img = Image.open(train_file).convert('L')
            resize_img = work_img.resize((size, size))
            temp = np.asarray(resize_img).astype(np.float64).flatten()

            temp = temp / np.sum( temp )
            train_vec[i][j-1] = np.hstack( (temp , one ) )
```

読み込む画像のファイル名

グレースケール画像として読み込み→大きさの変更→numpyに変換，ベクトル化

入力値の合計を1に正規化

閾値に対応する入力値（1で固定）を追加

デルタールールのアルゴリズム

while() {

  Error = 0

  for($p = 0$; $p < n$; p++) {

    for($i = 0$; $i < c$; i++) {

      $g_i(\mathbf{x}_p)$ の計算

      $e_{ip} = (g_i(\mathbf{x}_p) - b_{ip})$

      for($j = 0$; $j < d$; j++){

        w_{ij} -= alpha * e_{ip} * x_{pj}

      }

      Error += e_{ip} * e_{ip}

    }

  }

  if(Error < ε) break

}

重みベクトルは乱数によって事前に初期化しておく

誤差の計算

重みの修正

c ： クラスの総数　　d ： 次元数

w_{ij} ： クラス i の識別関数の j番目の重み係数

x_{pj} ： 特徴ベクトル$\mathbf{x}_p$の j番目の要素

誤差自乗和が一定値以下となったら停止

学習①

```
# 学習
def Train():
    for loop in range(LOOP):
        error = 0
        for t in range(class_num*train_num):
            # ランダムにj番目の数字iを選択
            i = np.random.randint(0,class_num)
            j = np.random.randint(0,train_num)

            # 教師信号の設定
            teach = np.zeros(class_num, dtype=np.float64)
            teach[i] = 1
```

誤差二乗和が一定値以下となったら停止
→ LOOP回繰り返す

i:ランダムに数字iを選択

j:ランダムに数字iのj番目の文字を選択

数字iのみ1，他は0

学習②

```
for k in range(class_num):
    # 出力値の計算
    out = np.dot( train_vec[i][j] , weight[k] )

    # 重みの修正
    weight[k] -= alpha * ( out - teach[k] ) * train_vec[i][j]

    # 誤差二乗和の計算
    error += ( out - teach[k] ) * ( out - teach[k] )

# 誤差二乗和の出力
print( loop , " Error : " , error )
```

$$\boxed{g(\mathbf{x}_p) = \mathbf{w}_i^t \mathbf{x}_p}$$

$$\boxed{\mathbf{w}_i' = \mathbf{w}_i - \alpha(\mathbf{w}_i^t \mathbf{x}_p - b_{ip})\mathbf{x}_p}$$

重みの画像化①

```
# 重みの画像化
for i in range(class_num):
    temp = weight[i][0:size*size]
    a = np.reshape( temp , (size,size) )

    plt.imshow(a , interpolation='nearest')
    file = "fig/weight-" + str(i) + ".png"
    plt.savefig(file)
    plt.close()
```

閾値に対応した重み以外を対象

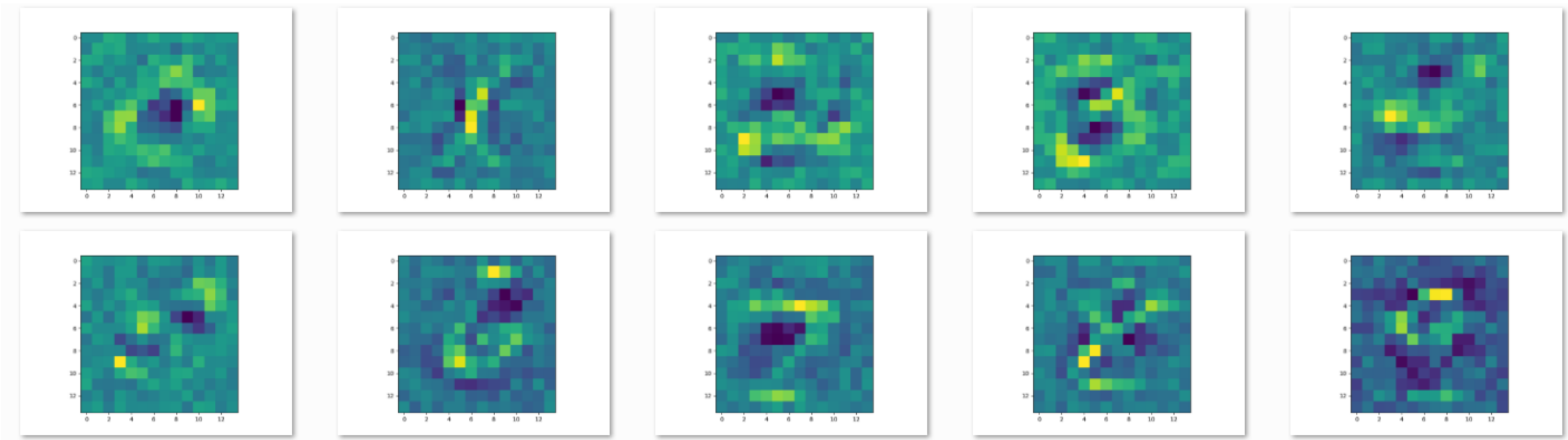
（size×size）の大きさに変更

二次元マップ化

保存するファイル名の指定→保存→閉じる

重みの画像化②

0 1 2 3 4



5 6 7 8 9

重みの保存

```
# 重みの保存
with open("weight.txt", mode='w') as f:
    for i in range(class_num):
        for j in range(size*size+1):
            f.write(str(weight[i][j])+"¥n")
```

「weight.txt」に重みを書き込む

重みの読み込み

Load_Weight

```
# 重みの読み込み
def Load_Weight():
    with open("weight.txt", mode='r') as f:
        for i in range(class_num):
            for j in range(size*size+1):
                weight[i][j] = float( f.readline().strip() )
```

「weight.txt」から重みを読み込む

予測(テストデータの読み込み)

Predict

```
def Predict():
    # 混合行列
    result = np.zeros((class_num,class_num), dtype=np.int32)
    for i in range(class_num):
        for j in range(1,train_num+1):
            # テストデータの読み込み
            pat_file = "mnist/test/" + str(i) + "/" + str(i) + "_" + str(j) + ".jpg"
            work_img = Image.open(pat_file).convert('L')
            resize_img = work_img.resize((size, size))
            temp = np.asarray(resize_img).astype(np.float64).flatten()

            temp = temp / np.sum( temp )
            pat_vec = np.hstack( ( temp , one ) )
```

読み込む画像のファイル名

グレースケール画像として読み込み→大きさの変更→numpyに変換，ベクトル化

入力値の合計を1に正規化

閾値に対応する入力値(1で固定)を追加

テストデータの予測

$$g(\mathbf{x}_p) = \mathbf{w}_i^t \mathbf{x}_p$$

```
    # 出力値の計算
    out = np.dot( weight , np.resize( pat_vec , (size*size+1) ) )
```

（class_num，size×size+1）

（size×size+1,1）

```
    # 予測
    ans = np.argmax( out )
```

np.argmax（配列）
配列の最大値の要素番号を返す

```
    result[i][ans] +=1
    print( i , j , "->" , ans )

print( "¥n [混合行列]" )
print( result )
print( "¥n 正解数 ->" ,  np.trace(result) )
```

認識結果の出力

メインメソッド

```
if __name__ == '__main__':

    argvs = sys.argv        引数

    # 引数がtの場合
    if argvs[1] == "t":

        # 学習データの読み込み
        Read_train_data()

        # 学習
        Train()

    # 引数がpの場合
    elif argvs[1] == "p":

        # 重みの読み込み
        Load_Weight()

        # テストデータの予測
        Predict()
```

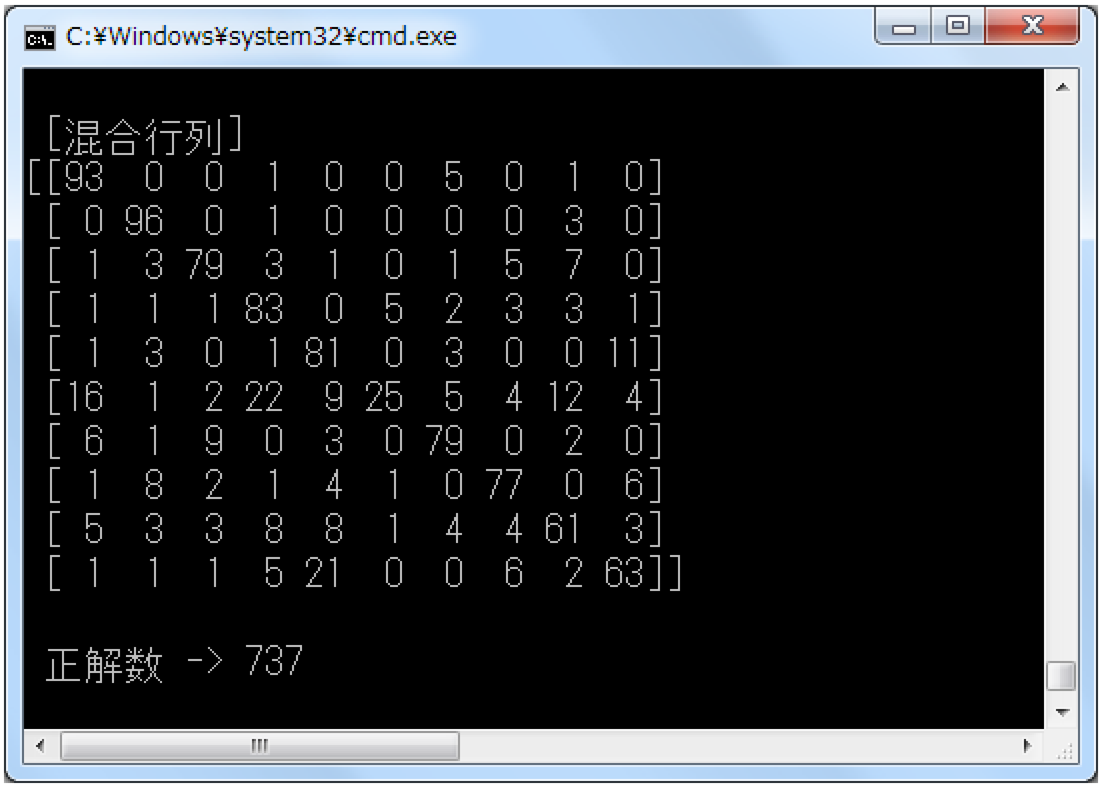
実行結果①(学習データの学習)

> python Delta.py t

誤差二乗和が小さくなっていくことを確認

実行結果②(テストデータの予測)

> python Delta.py p



宿題⑦

- デルタルールのプログラム（Delta.py）をパーセプトロンに変更しなさい.

- 線形分離可能な問題かどうかは分からないので，学習は，一定回数繰り返した後，停止してかまいません.

実は線形分離可能？

学習後の学習データの判別

```
9 91 -> 9
9 92 -> 9
9 93 -> 9
9 94 -> 9
9 95 -> 9
9 96 -> 9
9 97 -> 9
9 98 -> 9
9 99 -> 9
9 100 -> 9

 [混合行列]
[[100   0   0   0   0   0   0   0   0   0]
 [  0 100   0   0   0   0   0   0   0   0]
 [  0   0 100   0   0   0   0   0   0   0]
 [  0   0   0 100   0   0   0   0   0   0]
 [  0   0   0   0 100   0   0   0   0   0]
 [  0   0   0   0   0 100   0   0   0   0]
 [  0   0   0   0   0   0 100   0   0   0]
 [  0   0   0   0   0   0   0 100   0   0]
 [  0   0   0   0   0   0   0   0 100   0]
 [  0   0   0   0   0   0   0   0   0 100]]

 正解数 -> 1000

C:¥home¥shino¥prml-2018¥11-5¥program>
```

ヘッブの学習

固有ベクトルの学習

ヘッブの学習

- PCA.py
  - 固有ベクトルの学習（顔画像）

faceを同じフォルダー
に入れて下さい

figという名前のフォル
ダーを作成して下さい

- 実行方法
  - > python PCA.py
  - （時間がかかります）

変数の定義

```
# クラス数
class_num = 2

# 画像の大きさ
size = 16

# 学習データ
train_num = 100

# 学習データ
train_vec = np.zeros((class_num,train_num,size*size), dtype=np.float64)
```

```
# 出力の個数(固有ベクトルの個数)
output_size = 5

# 入力の個数
input_size = size*size

# 重みの初期化
weight = np.random.uniform( -0.5 , 0.5, (output_size,input_size) )

# 重みの変更値
d_weight = np.zeros( (output_size,input_size) )

# 学習回数，学習係数
LOOP = 1000
alpha = 0.01

# fig以下の画像を削除(MS-Windows)
os.system("del /Q fig¥*")
```

MacOS，UNIXの場合
rm fig/*

学習データの読み込み

```
# 学習データの読み込み
dir = [ "Male" , "Female" ]
for i in range(class_num):
    for j in range(1,train_num+1):
        train_file = "face/" + dir[i] + "/" + str(j) + ".png"
        work_img = Image.open(train_file).convert('L')
        resize_img = work_img.resize((size, size))
        train_vec[i][j-1] = np.asarray(resize_img).astype(np.float64).flatten()

        train_vec[i][j-1] = train_vec[i][j-1] / np.linalg.norm(train_vec[i][j-1])
```

読み込む画像のファイル名

グレースケール画像として読み込み→大きさの変更→numpyに変換，ベクトル化

入力ベクトルx, ||x||=1に正規化

学習①

```
# 学習
for o in range(class_num):
    for loop in range(LOOP):
        print( loop )

        for t in range(0,train_num):
            # 出力値の計算
            e = train_vec[o][t].reshape( (input_size,1) )
            V = np.dot( weight , e )
```

$$y_{pi} = \sum_{j=1}^{N} w_{ij} x_{pj}$$

学習②

$$w'_{ij} = w_{ij} + \alpha y_{pi}(x_{pj} - \sum_{k=1}^{i} y_{pk} w_{kj})$$

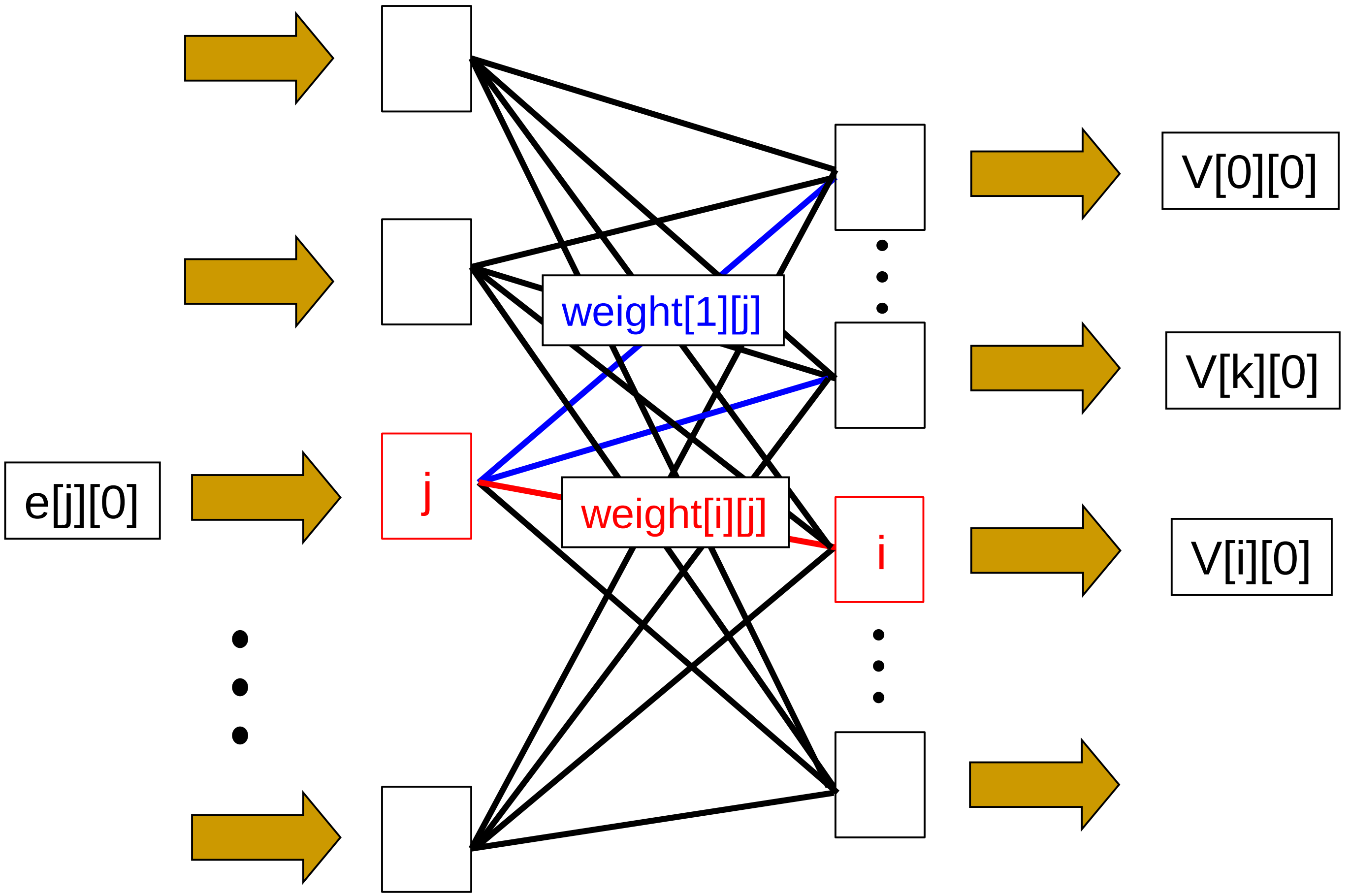
```
# 重みの更新値の計算
for i in range( output_size ):
    for j in range( input_size ):

        sum_o = 0
        for k in range(i+1):
            sum_o += V[k][0] * weight[k][j]

        d_weight[i][j] = alpha * V[i][0] * ( e[j][0] - sum_o )

# 重みの更新
weight += d_weight
```

$$\sum_{k=1}^{i} y_{pk} w_{kj}$$

$$\alpha y_{pi}(x_{pj} - \sum_{k=1}^{i} y_{pk} w_{kj})$$

学習③

$$w'_{ij} = w_{ij} + \alpha y_{pi}(x_{pj} - \sum_{k=1}^{i} y_{pk} w_{kj})$$



$$y_{pi} = \sum_{j=1}^{N} w_{ij} x_{pj}$$

重みベクトルの画像化

```
# 重みの画像化
for j in range(output_size):
    a = np.reshape( weight[j], (size,size) )
    plt.imshow(a , interpolation='nearest')
    plt.colorbar()
    file = "fig/weight-" + dir[o] + "-" + str(j) + ".png"
    plt.savefig(file)
    plt.clf()

# 検算
for i in range(output_size):
    print( np.dot( weight[1].T , weight[i] ) )
```

（size×size）の大きさに変更

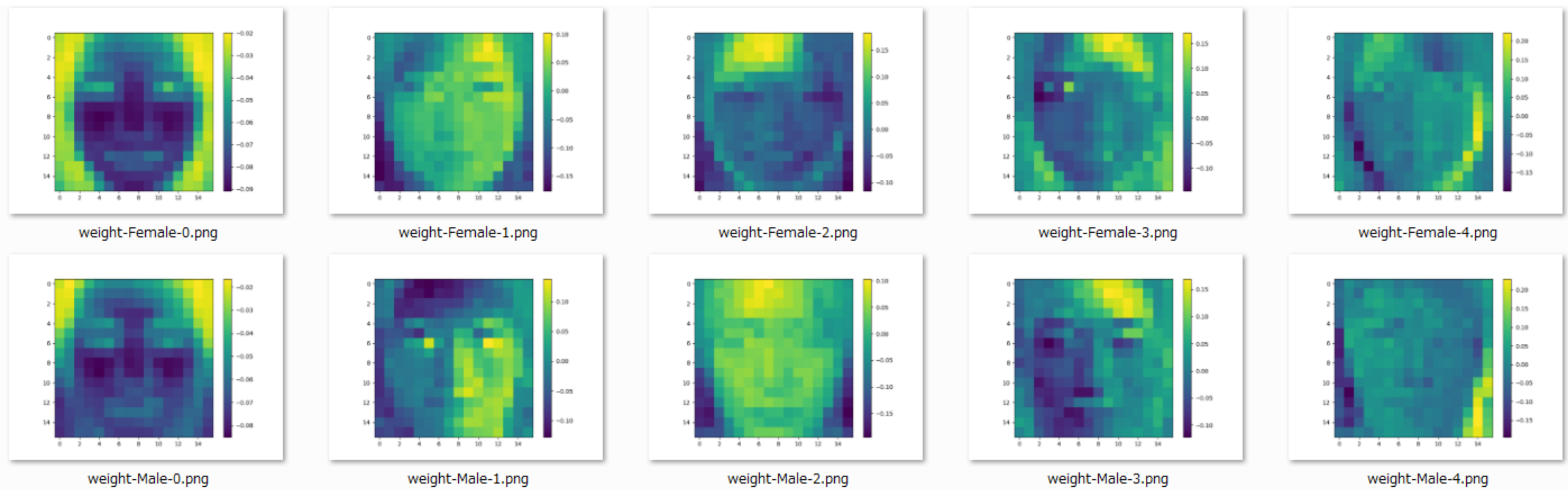
二次元マップ化

保存するファイル名の指定→保存→閉じる

第二固有ベクトルとそれ以外の固有ベクトルとの内積

重みベクトルの表示

女性画像を学習対象とした重みベクトル



男性画像を学習対象とした重みベクトル

重みベクトルの保存

```
# 重みベクトルの保存
filename = "weight-pca-" + dir[o] + ".txt"
f = open( filename , "w" )
for i in range( output_size ):
    for j in range( input_size ):
        f.write( str( weight[i][j] ) + "¥n" )
f.close()
```

実行結果

第一固有ベクトルの内積

第一固有ベクトルとその他の固有ベクトルの内積

（本日の）参考文献

- 石井健一郎他：わかりやすいパターン認識，オーム社（1998）
- C.M.ビショップ：パターン認識と機械学習（上），シュプリンガー・ジャパン（2007）
- 平井有三：はじめてのパターン認識，森北出版（2012）