パターン認識と学習
ニューラルネットワーク(2)

管理工学科
篠沢佳久

資料の内容

- ニューラルネットワーク(2)

- 多層パーセプトロン
  - 誤差逆伝播則

- 問題点と(深層学習のための)工夫

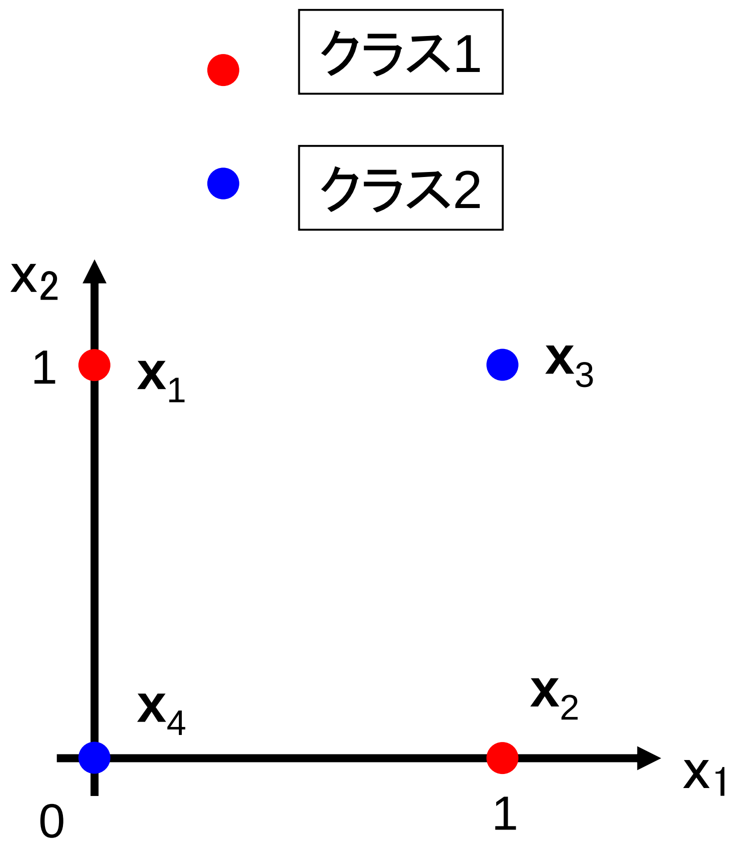
- 実習(誤差逆伝播則)

多層パーセプトロン

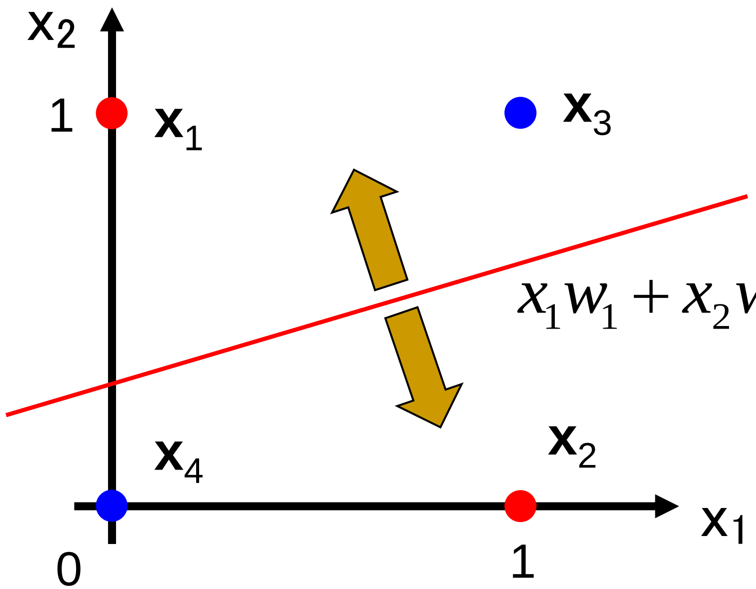
線形分離不可能な問題への対応

線形分離不可能①

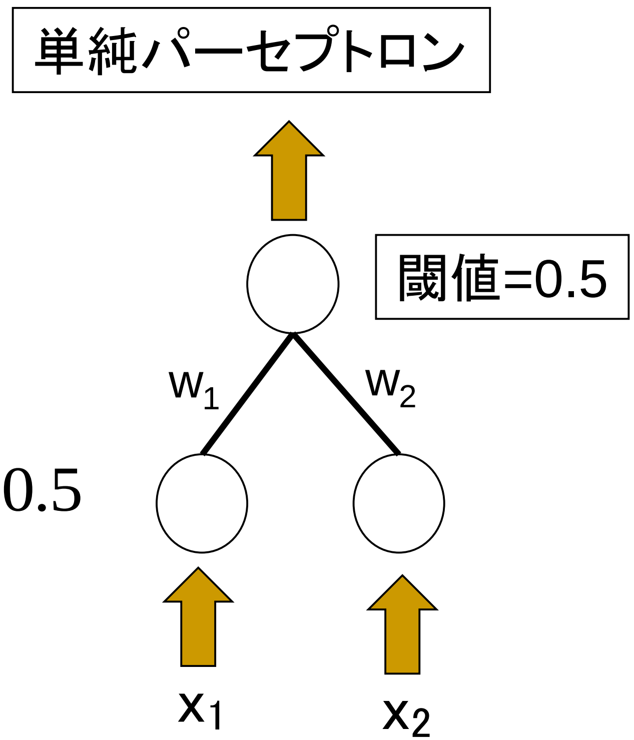
	x_1	x_2	
$\mathbf{x}_1$	0	1	クラス1
$\mathbf{x}_2$	1	0	クラス1
$\mathbf{x}_3$	1	1	クラス2
$\mathbf{x}_4$	0	0	クラス2



線形分離不可能②



$$x_1 w_1 + x_2 w_2 = 0.5$$

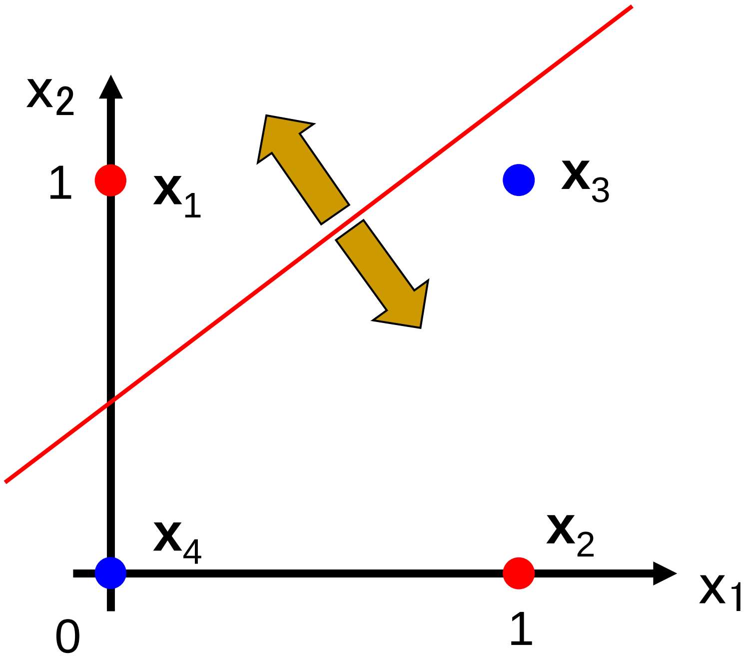


クラス1とクラス2を識別できる重みは存在しない
→ 線形分離不可能
→ パーセプトロンでは解けない問題

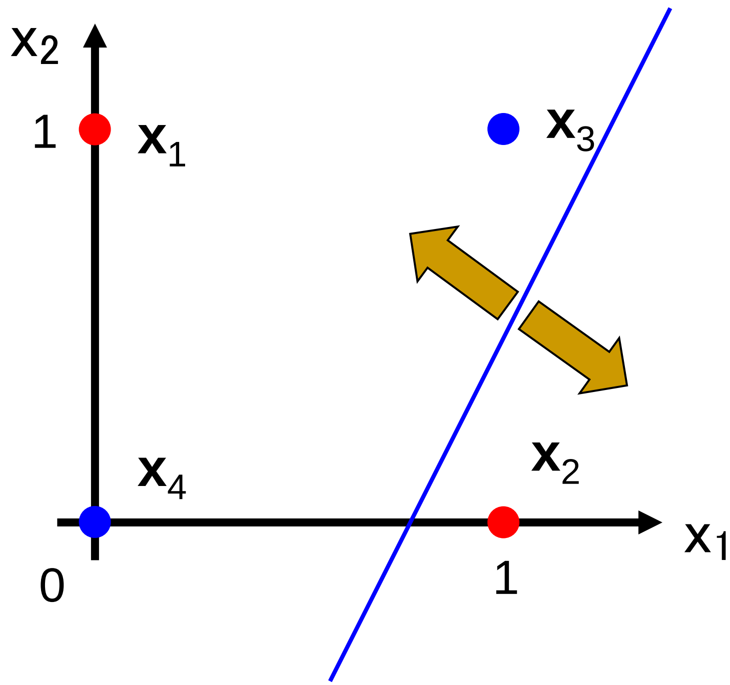
線形識別関数を組み合わせた解法①

識別関数1

$$x_1 w_{11} + x_2 w_{12} = \theta_1$$



識別関数2

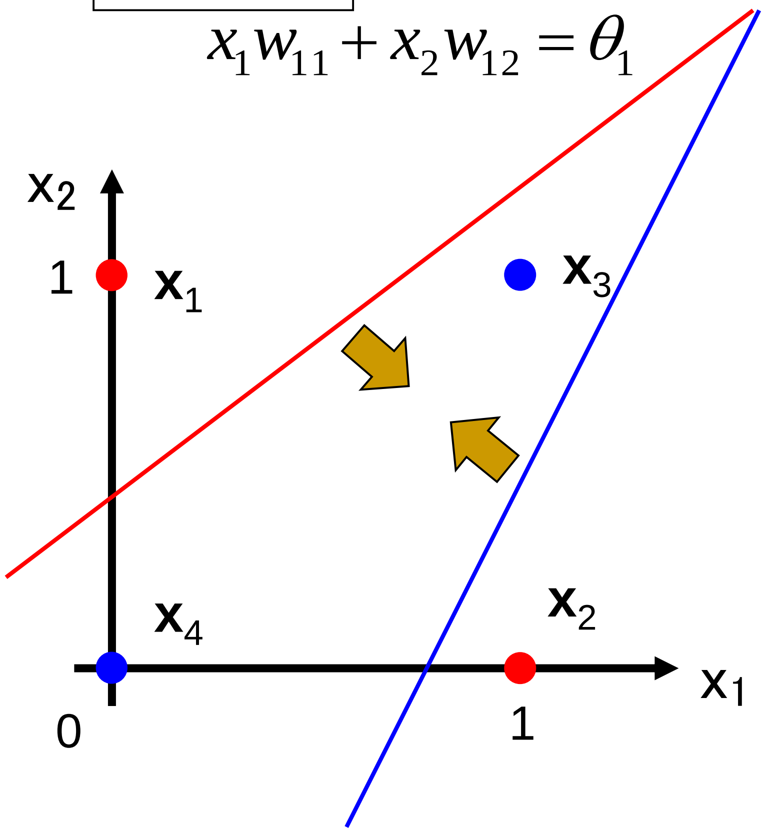


$$x_1 w_{21} + x_2 w_{22} = \theta_2$$

線形識別関数を組み合わせた解法②

識別関数1

$$x_1 w_{11} + x_2 w_{12} = \theta_1$$



識別関数2 $x_1 w_{21} + x_2 w_{22} = \theta_2$

新しい識別関数

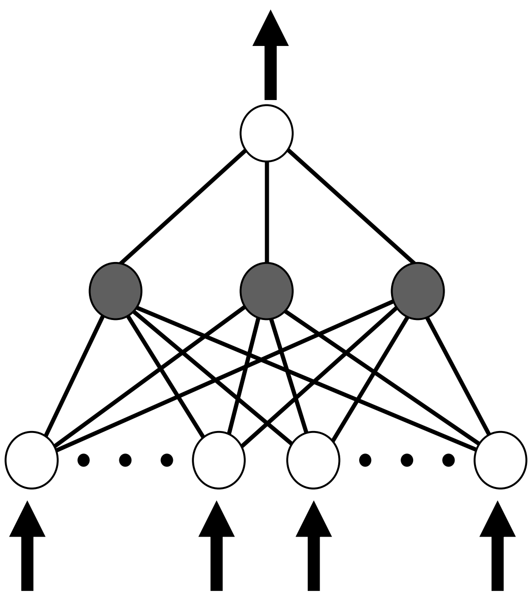
$$F(\mathbf{x}) = \alpha_1 f_1(\mathbf{x}) + \alpha_2 f_2(\mathbf{x})$$

識別関数1　　識別関数2

線形識別関数を組み合わせること
によって新しい識別関数を構築

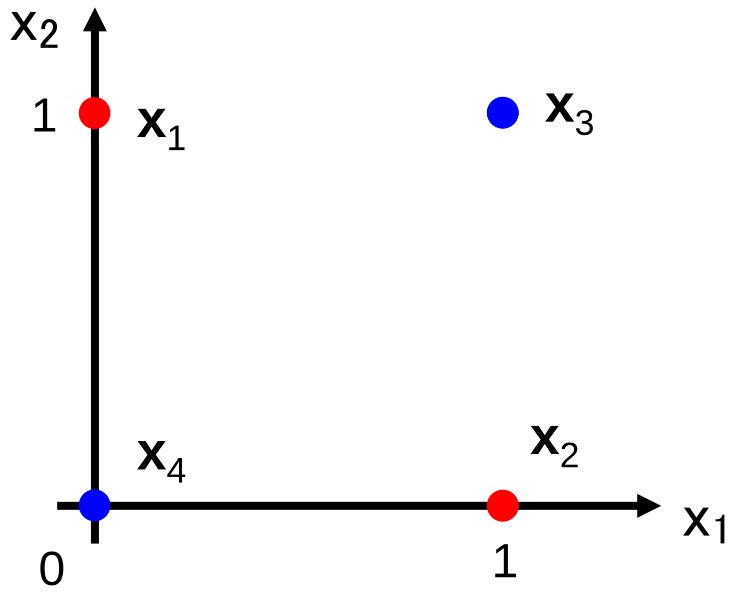
ネットワークの多層化

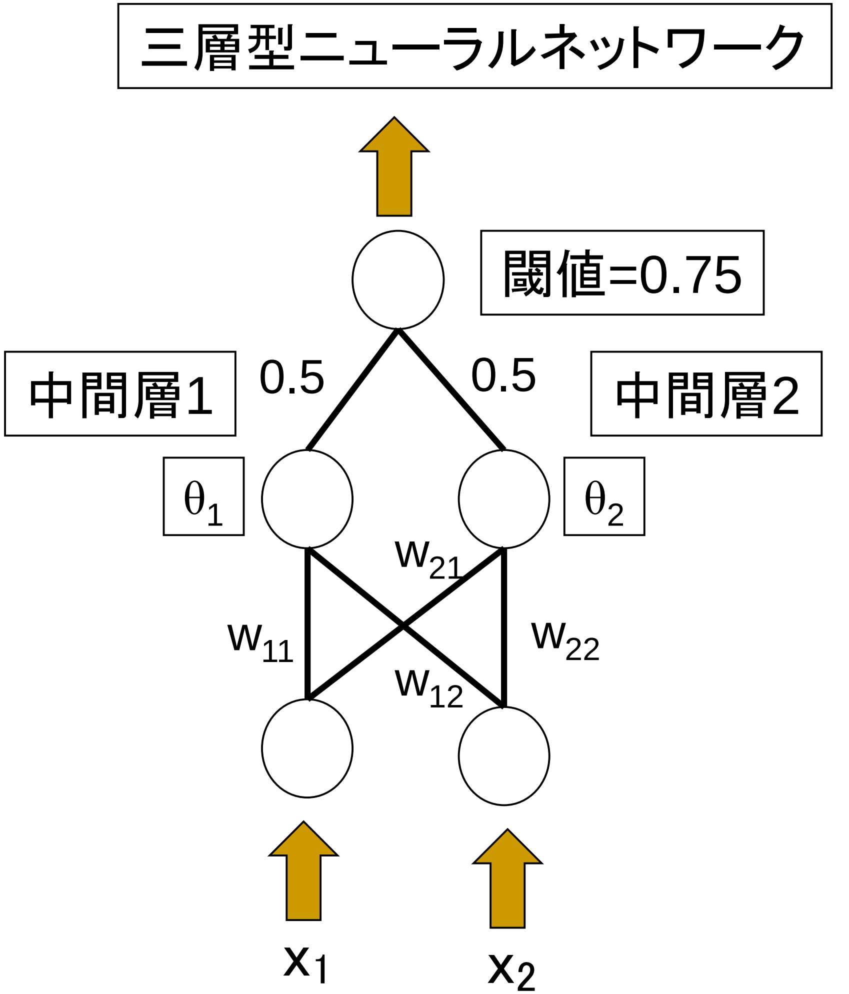
- 階層型ニューラルネットワーク
  - 「多層パーセプトロン」とも呼ばれる
  - Multi-Layer Perceptron（MLP）



ネットワークを多層化することによって線形分離不可能問題が解決できるか

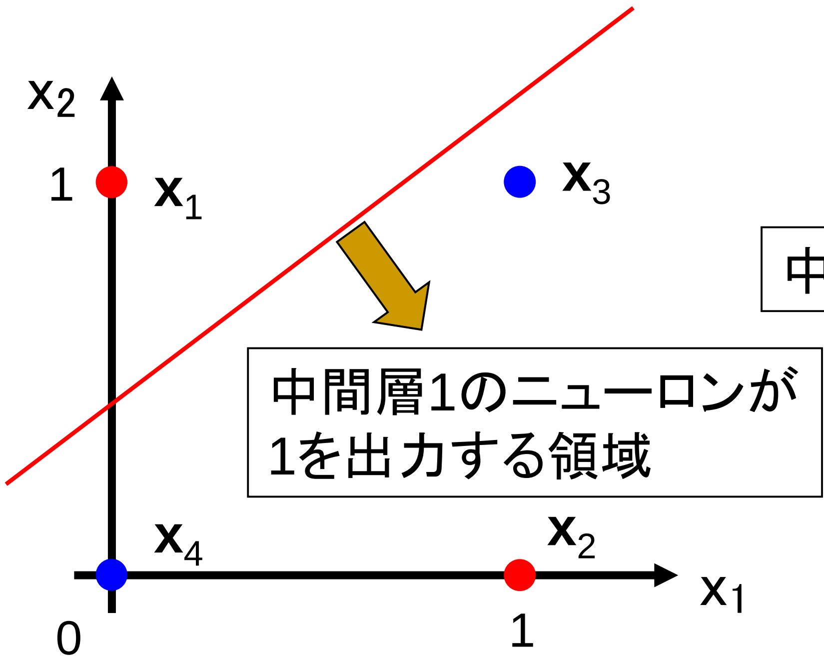
（直感的ですが…）多層化の意味①

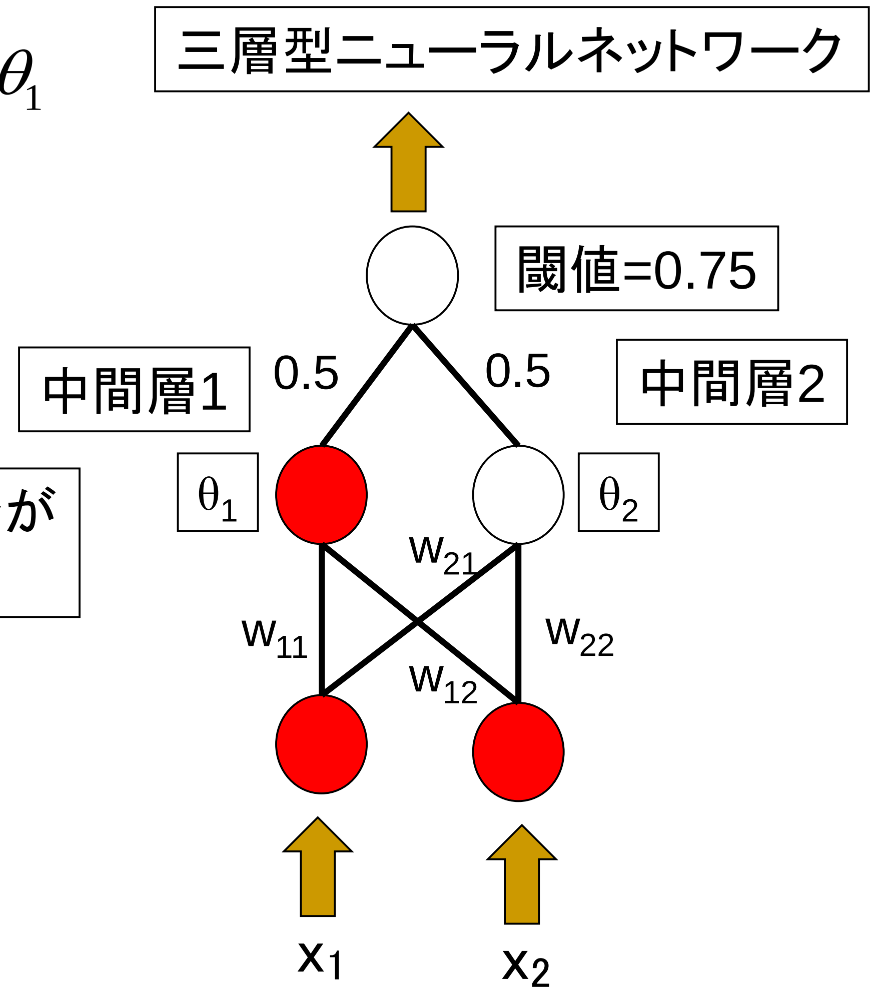




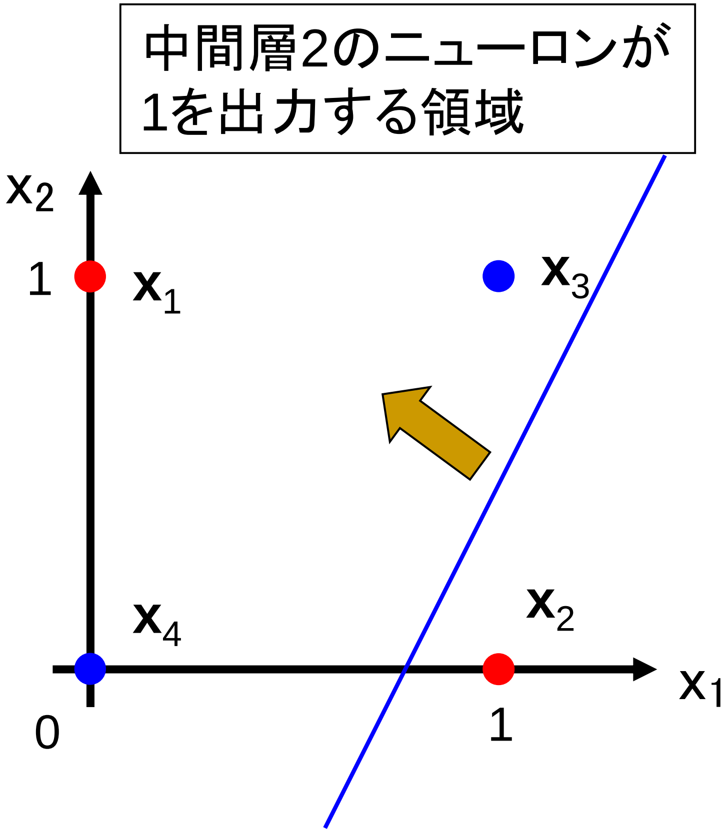
多層化の意味②

$$x_1 w_{11} + x_2 w_{12} = \theta_1$$

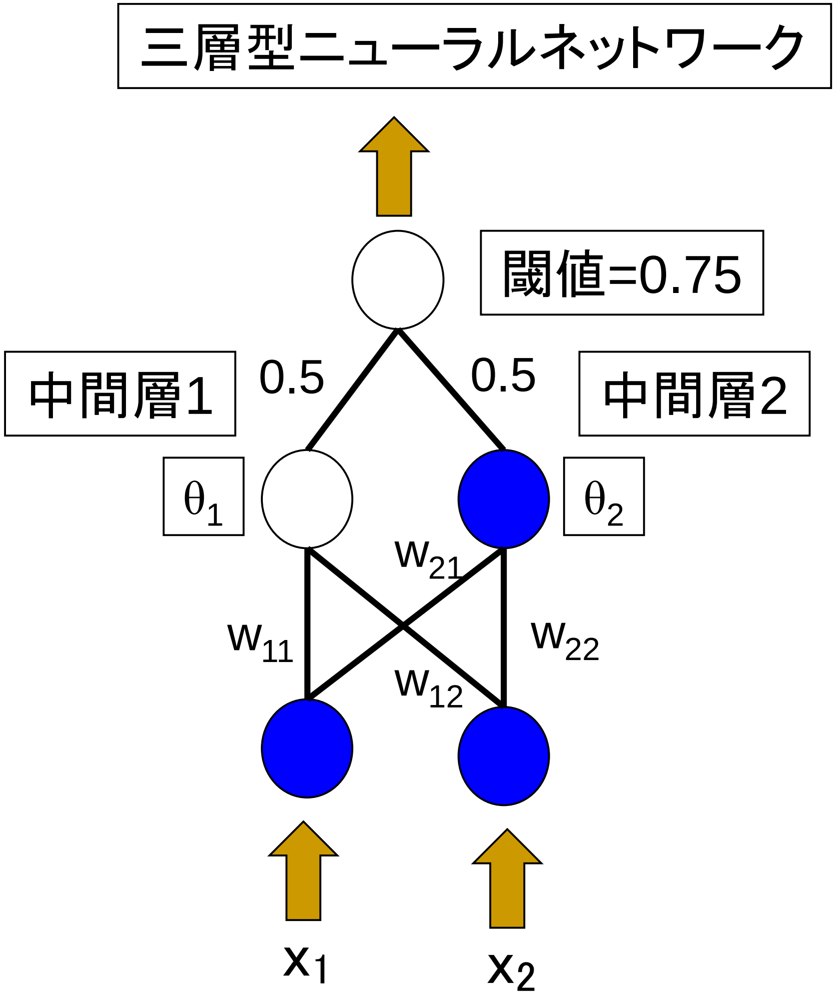




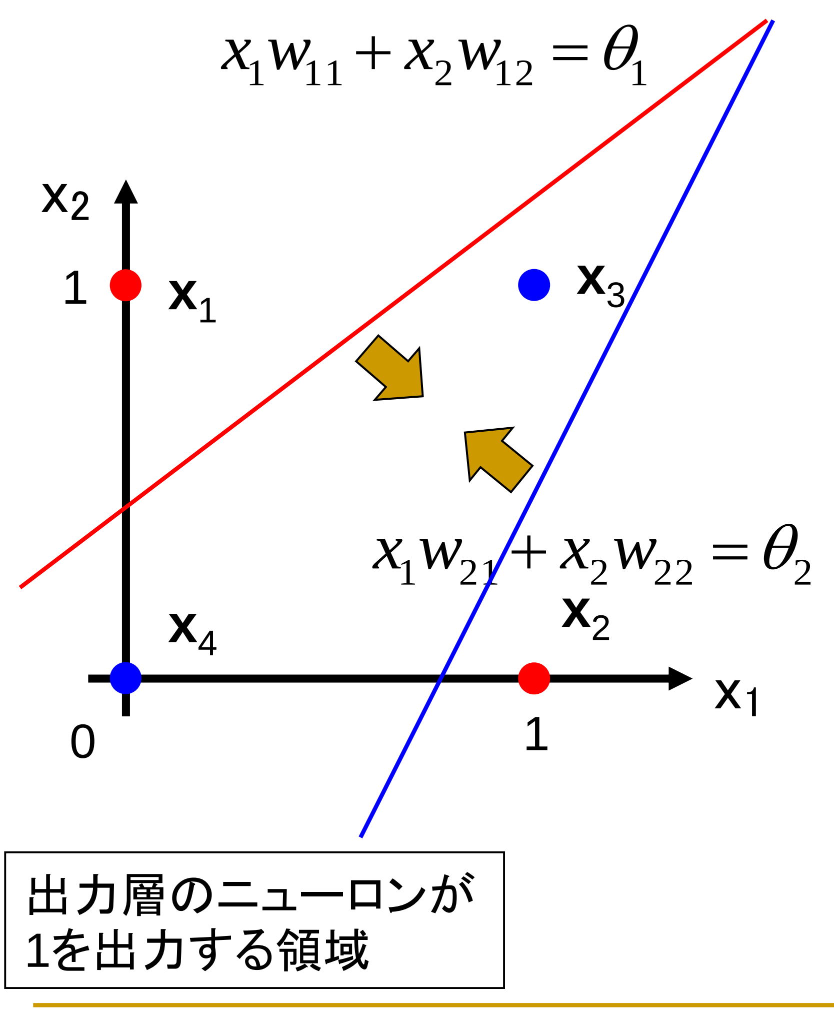
多層化の意味③



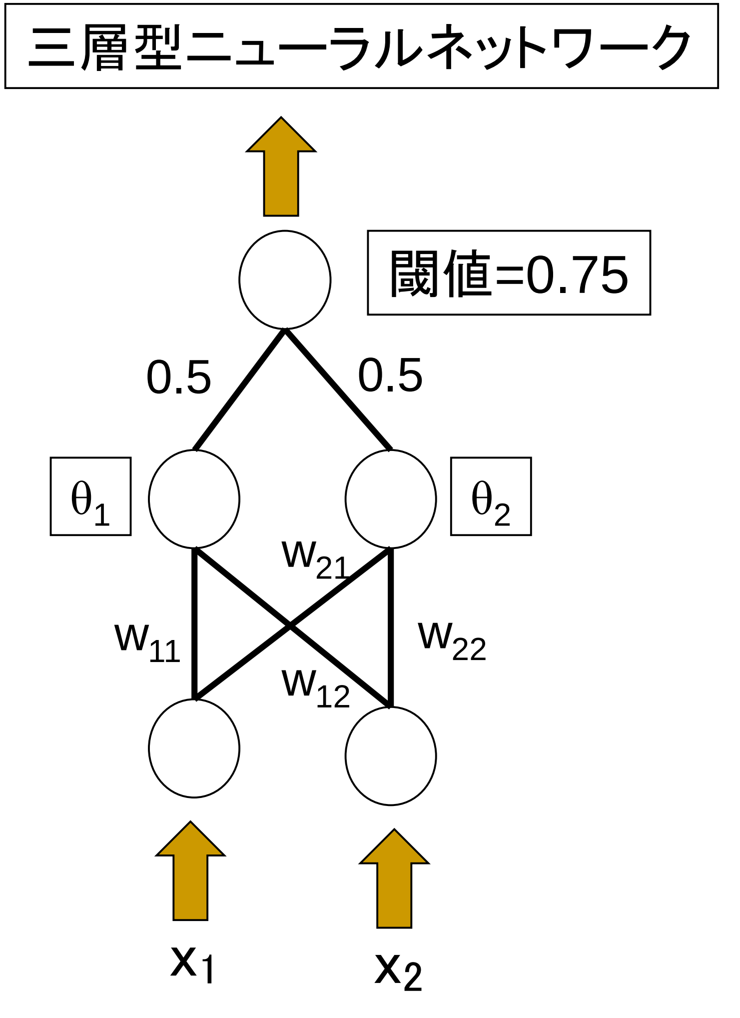
$$x_1 w_{21} + x_2 w_{22} = \theta_2$$



多層化の意味④

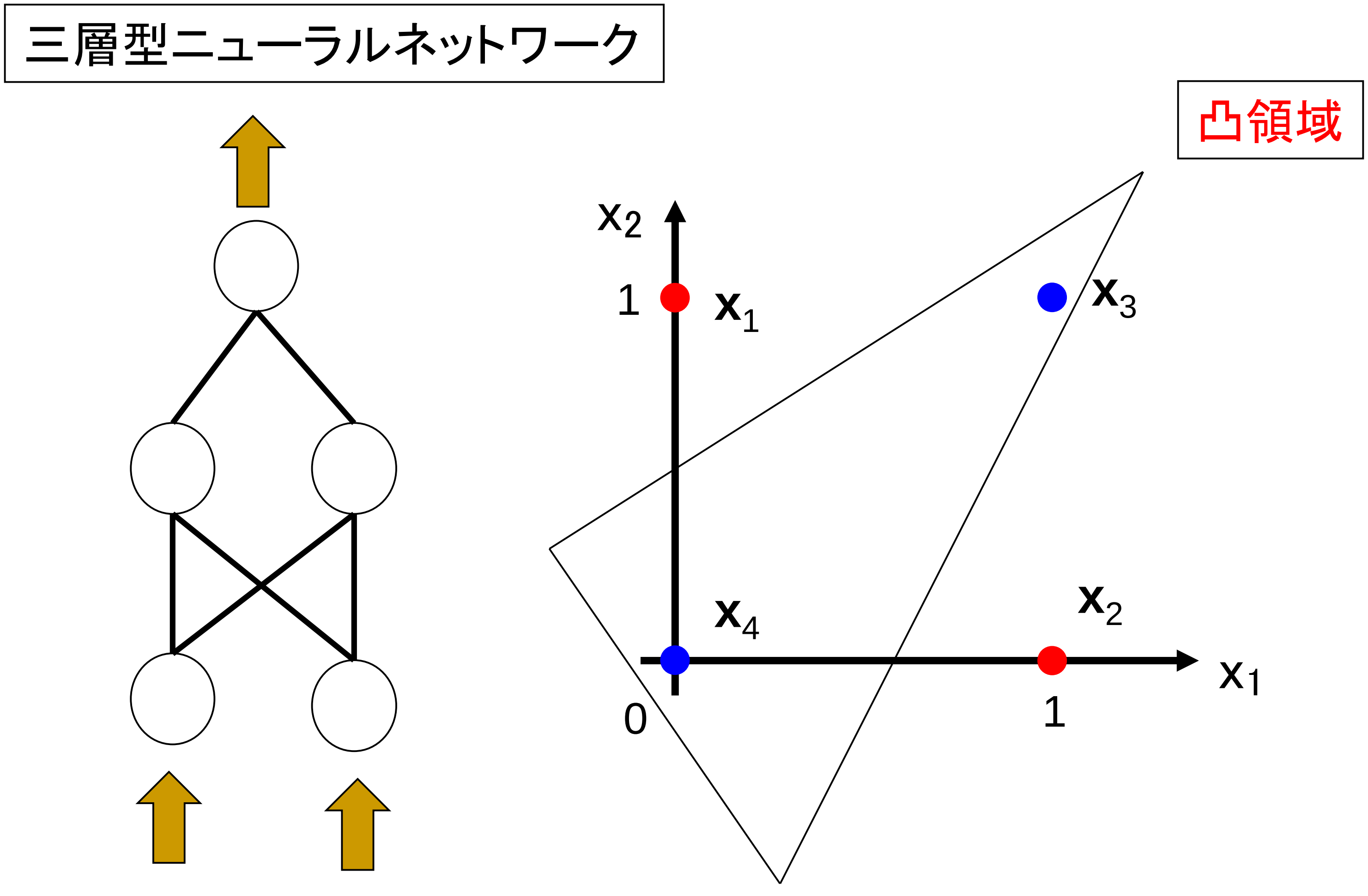


$x_1 w_{11} + x_2 w_{12} = \theta_1$
$x_1 w_{21} + x_2 w_{22} = \theta_2$
x2
1
x1
x3
x4
x2
0
1
x1
出力層のニューロンが
1を出力する領域



三層型ニューラルネットワーク
閾値=0.75
0.5
0.5
θ1
θ2
w21
w11
w22
w12
x1
x2

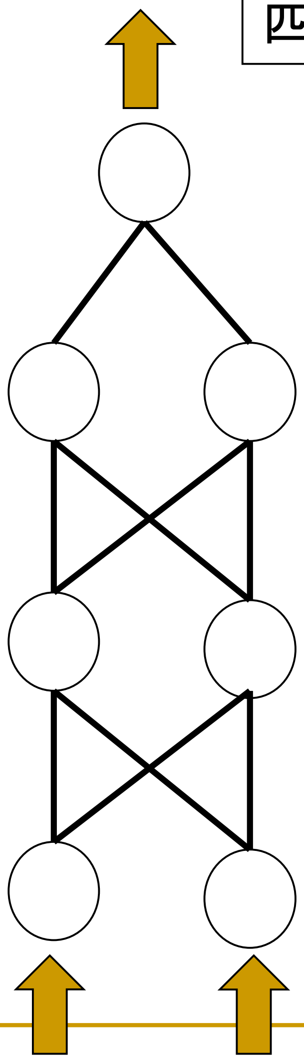
三層型ニューラルネットワーク

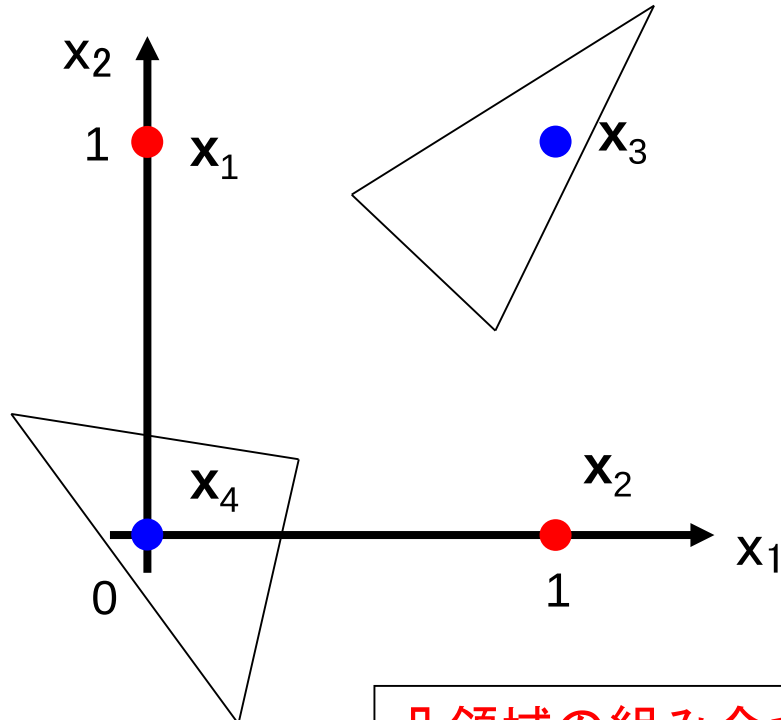


*凸領域を囲む直線（超平面）は中間層のニューロン数の影響を受ける

四層型ニューラルネットワーク

四層型ニューラルネットワーク





凸領域の組み合わせ

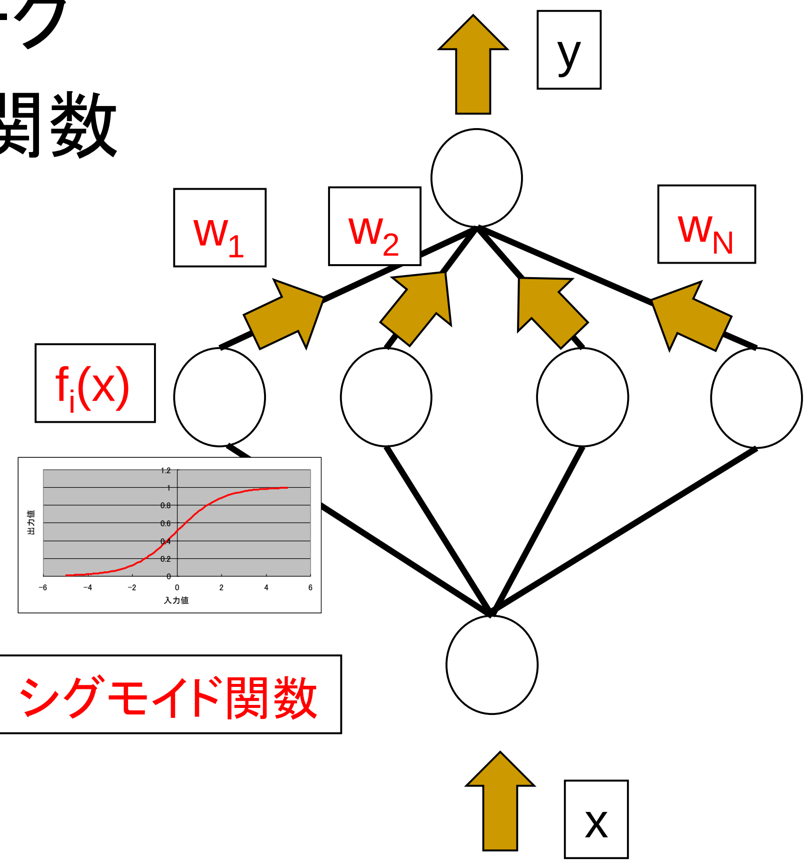
Universal Approximation Theorem （G. Cybenko）

- 三層型ニューラルネットワーク
  - 活性化関数はシグモイド関数
  - 有限個の中間層の個数

⇓

- 連続関数の近似が可能

$$y \approx \sum_{i=1}^{N} w_i f_i(x)$$



学習アルゴリズムの改良

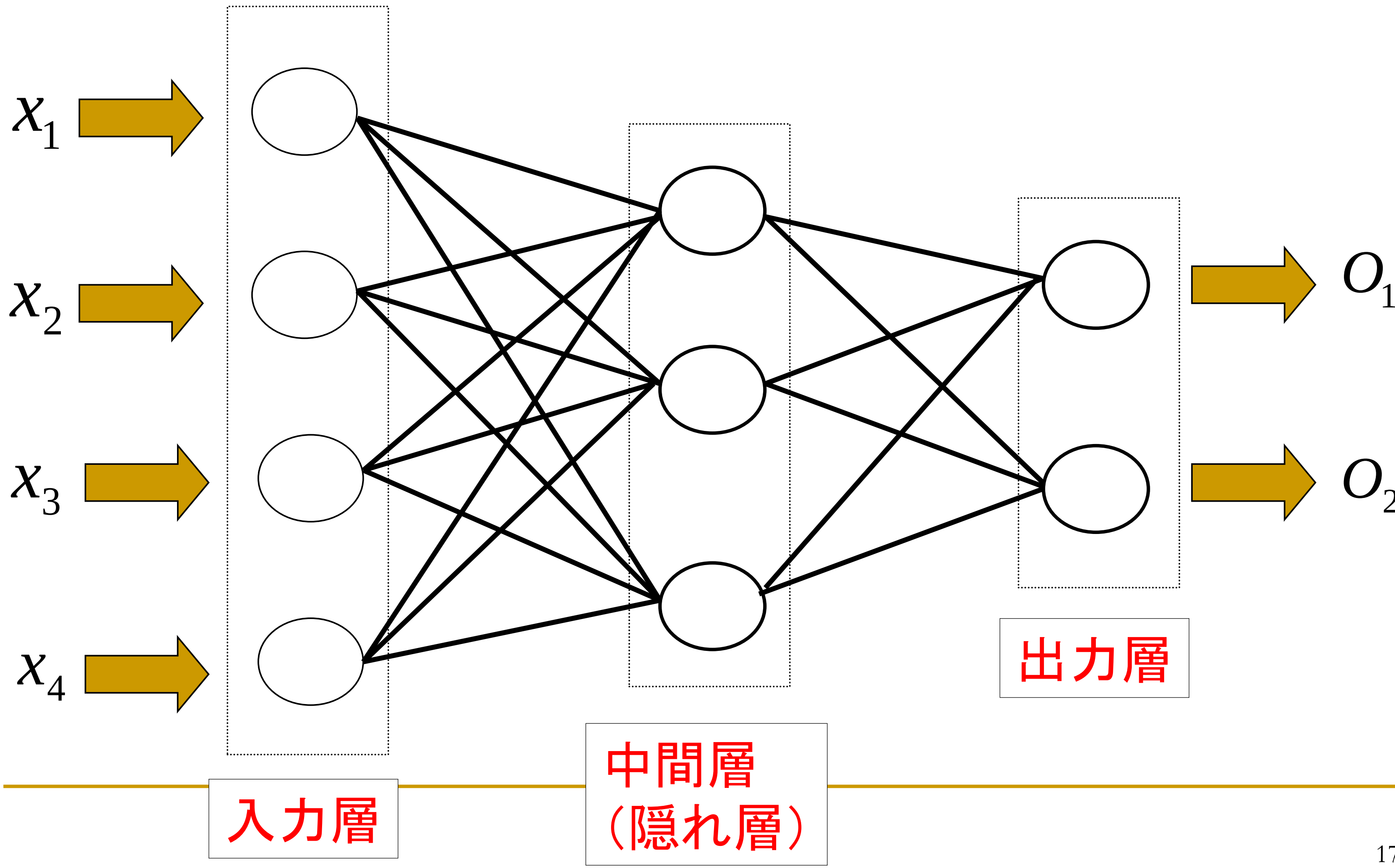
- 単純パーセプトロンでは原理的に線形分離不可能な問題には対応できない

↓

- ネットワークの階層を三層以上とする
- 多層型に対応した学習アルゴリズムの改良
  - デルタルールの改良
  - 微分可能な活性化関数(例えばシグモイド関数を用いる)

階層型ニューラルネットワーク①

三層型ニューラルネットワーク



階層型ネットワーク②

- 入力層のニューロン
  - ネットワークの外部からの入力を受け取る
- 中間層のニューロン
  - 入力層のニューロンから信号（値）を受け取り，出力層のニューロンへ信号を送る
- 出力層のニューロン
  - 中間層のニューロンから信号を受け取り，外部へ出力値を送る

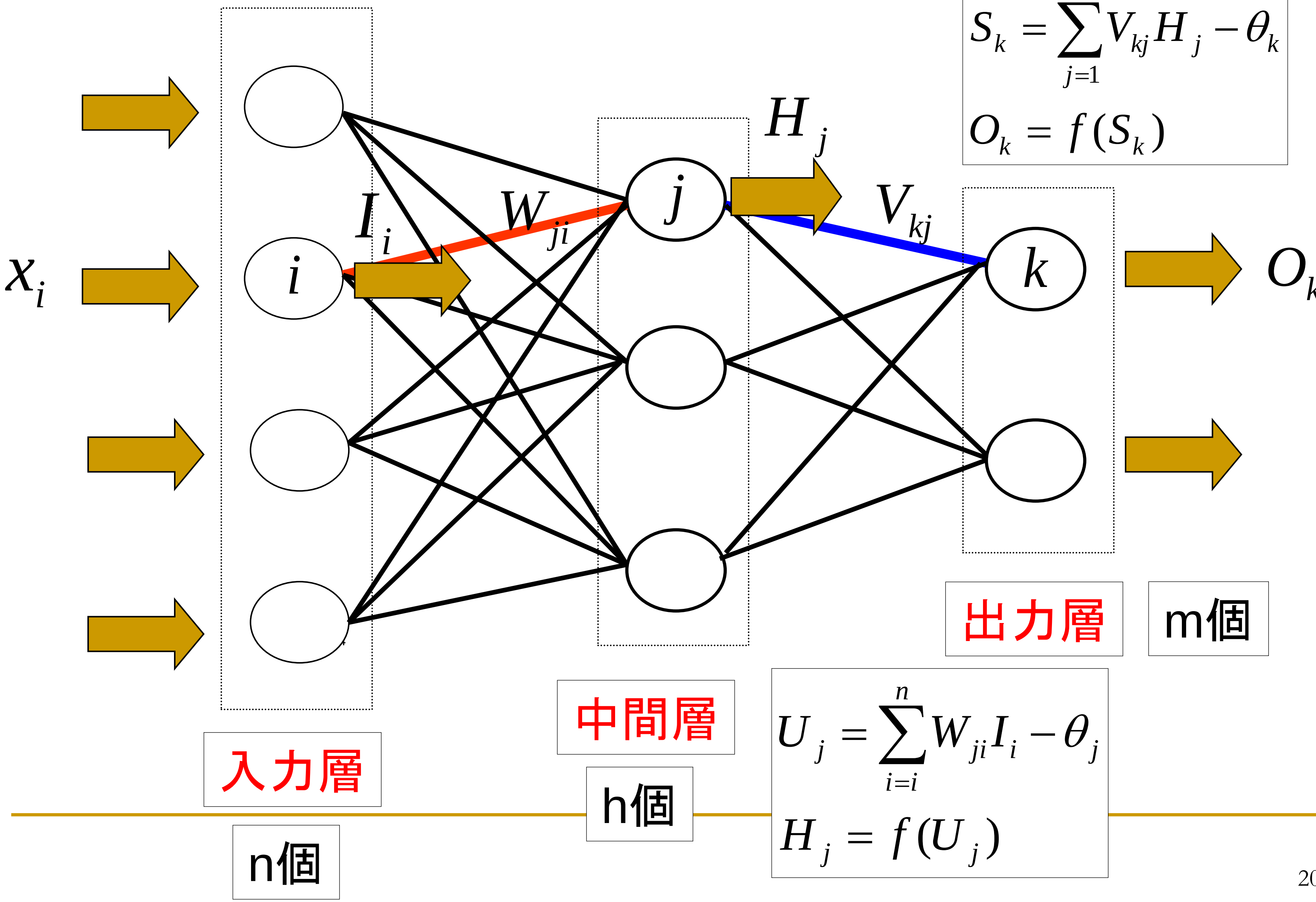
ネットワークの表記①

- i 番目の入力層の値（入力信号 x_i と同じ値） I_i
- j 番目の中間層の出力値　H_j　内部状態　U_j
- k 番目の出力層の出力値　O_k　内部状態　S_k

- 入力層のニューロン数　n
- 中間層のニューロン数　h
- 出力層のニューロン数　m

- j 番目の中間層と i 番目の入力層との結合係数　W_{ji}
- k 番目の出力層と j 番目の中間層との結合係数　V_{kj}

ネットワークの表記②



$$S_k = \sum_{j=1}^{h} V_{kj} H_j - \theta_k$$

$$O_k = f(S_k)$$

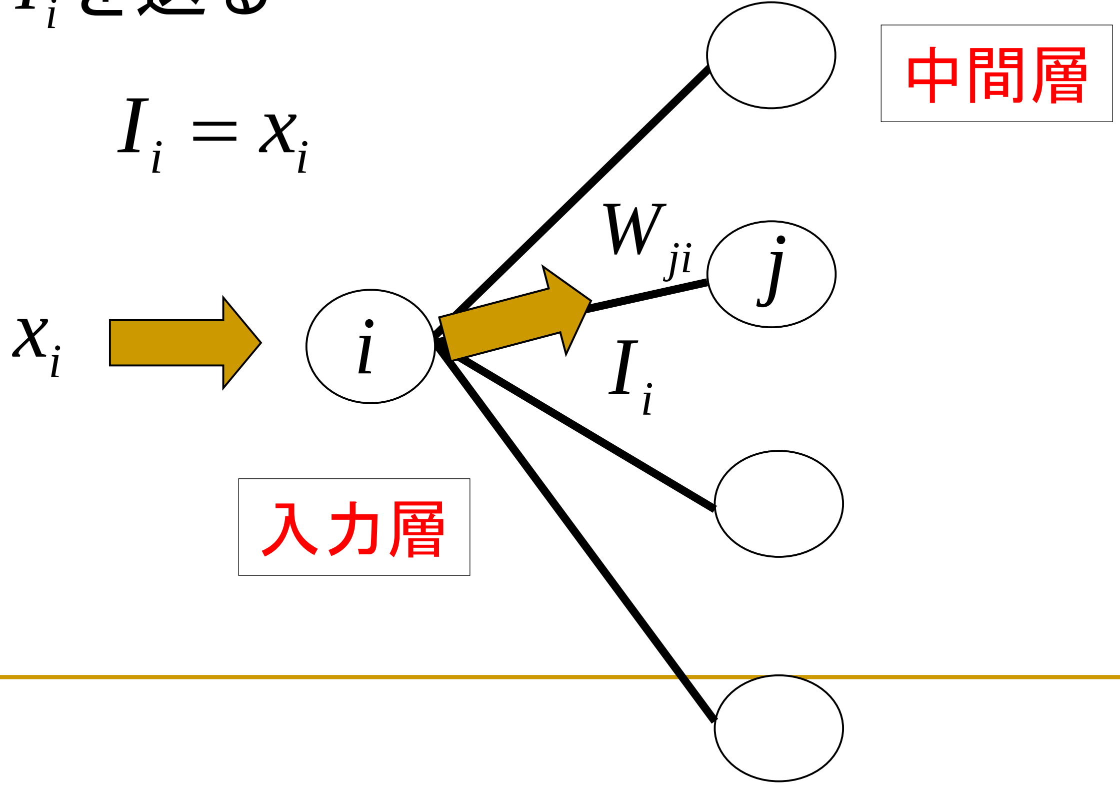
$$U_j = \sum_{i=i}^{n} W_{ji} I_i - \theta_j$$

$$H_j = f(U_j)$$

ネットワークの動作①

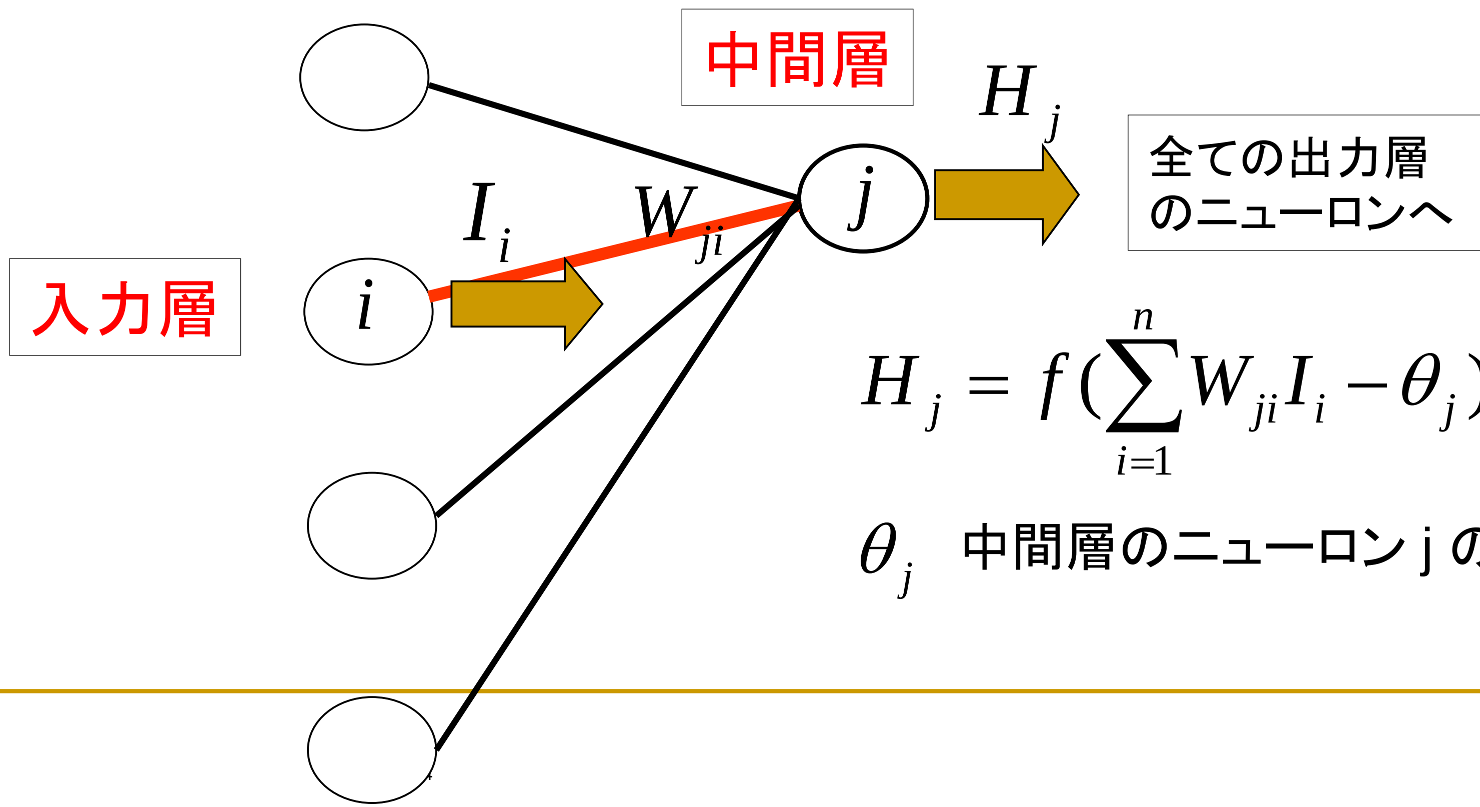
- 入力層のｉ番目のニューロン
  - 入力信号 x_i を受け取り，全ての中間層へ出力値 I_i を送る

$$I_i = x_i$$



ネットワークの動作②

- 中間層のｊ番目のニューロン
  - 全ての入力層のニューロンから値を受け取り，出力値を計算

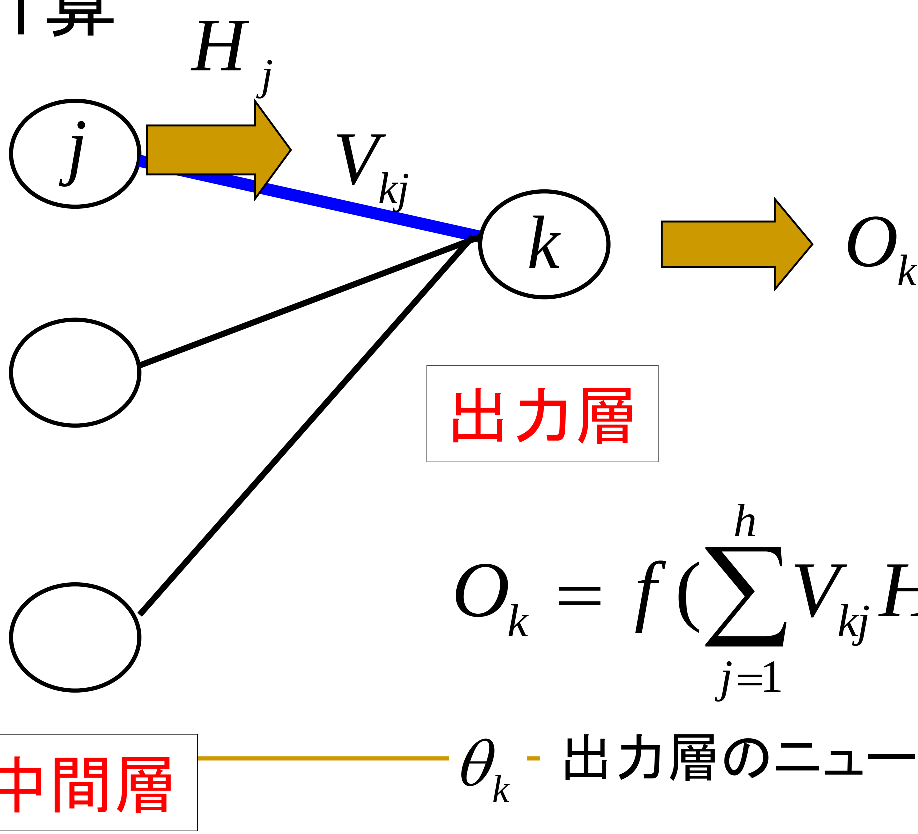


$$H_j = f\left(\sum_{i=1}^{n} W_{ji} I_i - \theta_j\right)$$

θ_j　中間層のニューロンｊのしきい値

ネットワークの動作③

- 出力層の k 番目のニューロン
  - 全ての中間層のニューロンから値を受け取り，出力値を計算



$$O_k = f\left(\sum_{j=1}^{h} V_{kj} H_j - \theta_k\right)$$

θ_k - 出力層のニューロン k のしきい値

誤差逆伝播則

多層パーセプトロンの学習

誤差逆伝播(バックプロパゲーション)アルゴリズム

- Error Back-Propagation Algorithms
- Rumelhart, D.E., McCeland, J.L.：*Parallel Distibuted Processing*, Vol.1, pp.318-362, MIT Press（1986）
- 階層型ニューラルネットワーク(三階層以上)の代表的な学習アルゴリズムの一つ
  - 一般化デルタルールとも呼ばれる

教師信号

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

出力

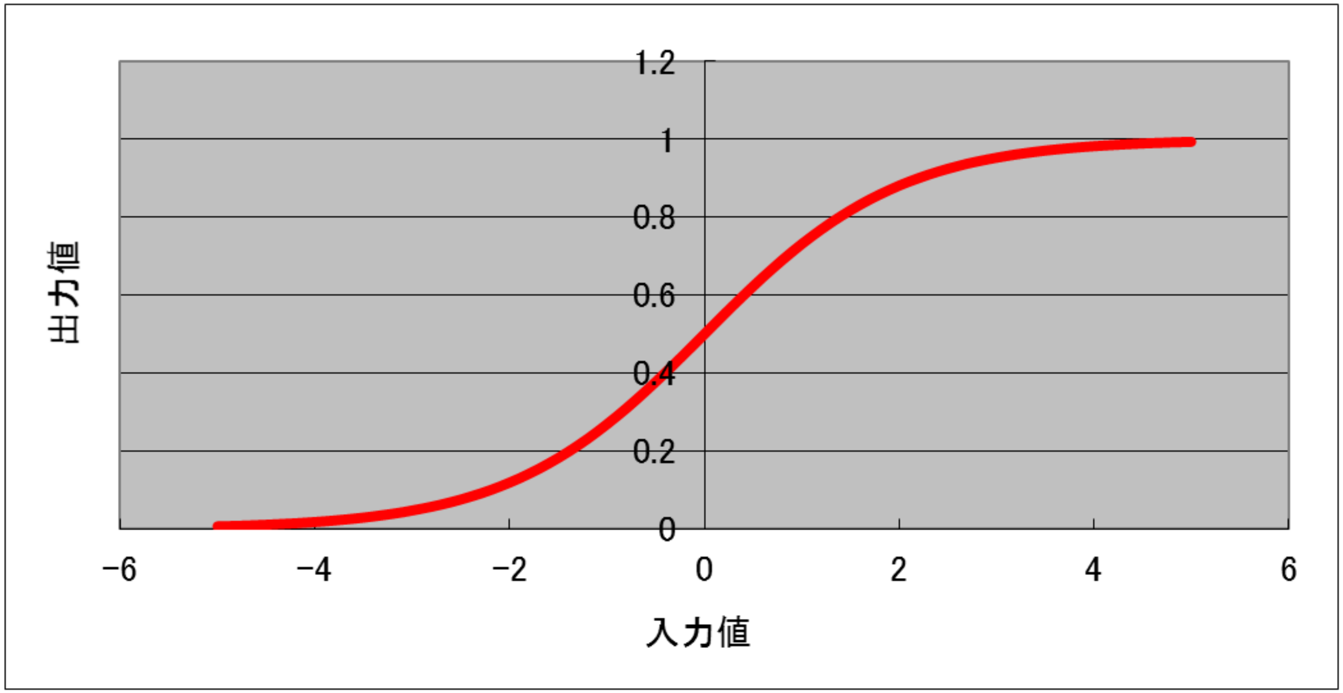
ニューラルネットワーク

  - $O_{p1}, O_{p2},\cdots, O_{pm}$
  - $t_{p1},t_{p2},\cdots,t_{pm}$

望ましい出力結果
→ 教師信号（ベクトル）

出力値の制約

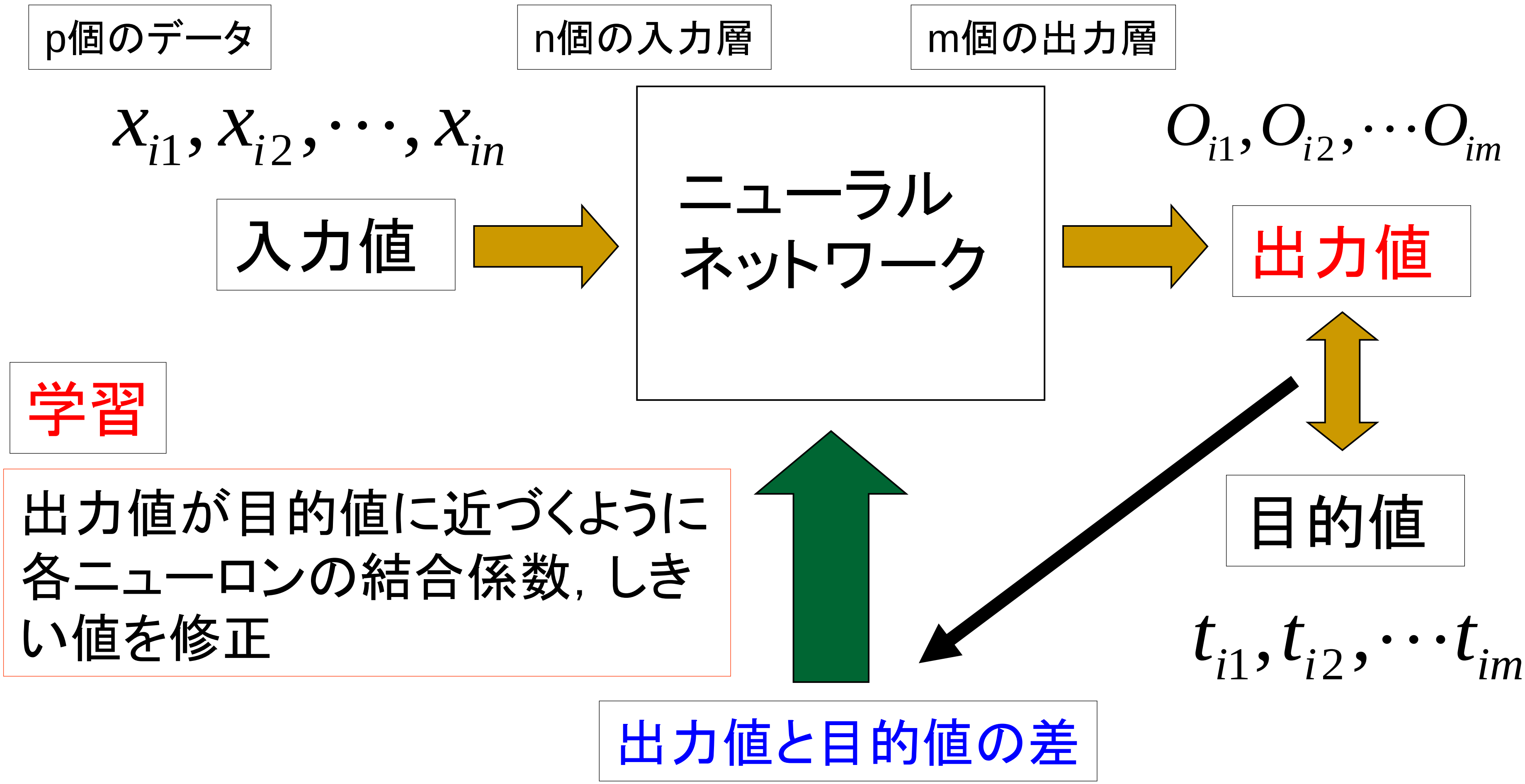
シグモイド関数の場合



出力値の範囲 $O_{i1}, O_{i2}, \cdots O_{im}$ $0 \le O_{ij} \le 1$

教師信号 t_{ij} も0から1の範囲とする

ニューラルネットワークの学習



誤差二乗和最小学習

- 外部から教師信号を与える教師あり学習

方針

- 出力値と教師信号の差の自乗和を最小とするように結合係数(しきい値)を決める

定式化*

$$E_p = \sum_{k=1}^{m} (O_{pk} - t_{pk})^2$$

$$E = \sum_{p=1}^{P} E_p \rightarrow \min$$

損失(誤差)関数

*E_pの場合，バッチサイズが1のオンライン学習，Eの場合，バッチ学習と呼ばれます

（データ数 i=1,2,･･･,P）

入力値

$x_{11}, x_{12}, \cdots, x_{1n}$

$x_{i1}, x_{i2}, \cdots, x_{in}$

$x_{P1}, x_{P2}, \cdots, x_{Pn}$

n個の入力層

ニューラルネットワーク

m個の出力層

出力値

$O_{11}, O_{12}, \cdots O_{1m}$

$O_{i1}, O_{i2}, \cdots O_{im}$

$O_{P1}, O_{P2}, \cdots O_{Pm}$

目的値（教師信号）

$t_{11}, t_{12}, \cdots t_{1m}$

$t_{i1}, t_{i2}, \cdots t_{im}$

$t_{P1}, t_{P2}, \cdots t_{Pm}$

損失関数

$$E_p = \sum_{k=1}^{m} (O_{pk} - t_{pk})^2$$

$$E = \sum_{p=1}^{P} E_p \rightarrow \min$$

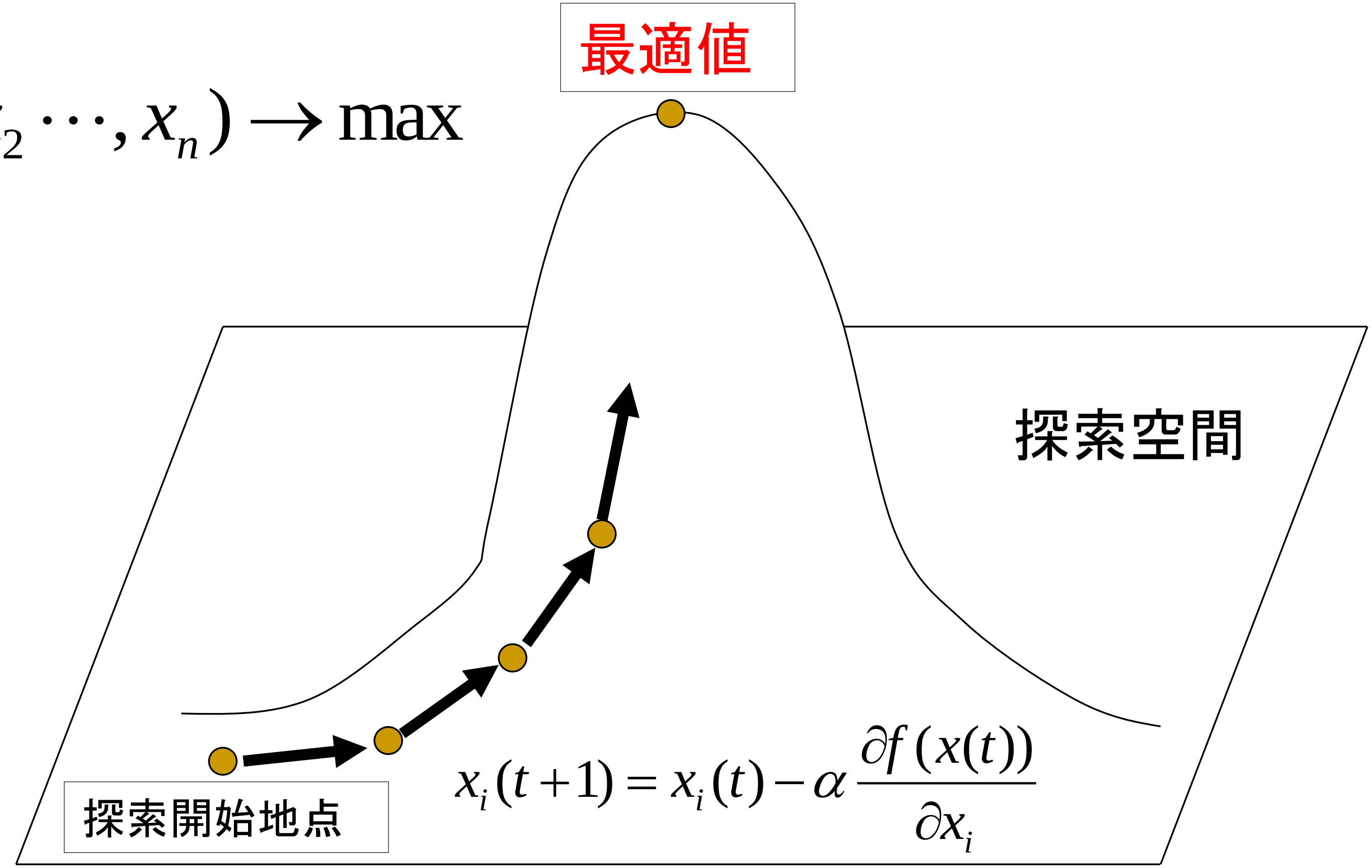
学習アルゴリズム

ニューラルネットワークを動作
プロパゲーションとも呼ぶ

while (true) {
差の合計値 = 0
for(i = 1 ; i <= データ数 ; i++) {
$\boxed{O_{i1}, O_{i2}, \cdots O_{im} \longleftarrow f(x_{i1}, x_{i2}, \cdots, x_{in})}$
$O_{i1}, O_{i2}, \cdots O_{im}$ と $t_{i1}, t_{i2}, \cdots t_{im}$ との差を求める
差の合計値 += 差
差が小さくなるようにニューラルネットワークの
パラメーターを修正（学習）
}
if 差の合計値 < ε break
}

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（α<1）

最急降下法による解法

入力値

$$x_{p1}, x_{p2}, \cdots, x_{pn}$$

ニューラルネットワーク

出力値

$$O_{p1}, O_{p2}, \cdots O_{pm}$$

教師信号

$$t_{p1}, t_{p2}, \cdots t_{pm}$$

誤差の自乗和の計算

$$E_p = \sum_{k=1}^{m} (O_{pk} - t_{pk})^2$$

入力層から中間層への結合係数の修正

$$W_{ji} \leftarrow W_{ji} - \alpha \frac{\partial E_p}{\partial W_{ji}}$$

中間層から出力層への結合係数の修正

$$V_{kj} \leftarrow V_{kj} - \alpha \frac{\partial E_p}{\partial V_{kj}}$$

それぞれのE_pにおいて最小化するように結合係数を修正する

*確率的勾配降下法（SGD：Stochastic Gradient Descent）と呼ばれます

結合係数の修正の計算

- 中間層から出力層への結合係数の修正

$$\frac{\partial E_p}{\partial V_{kj}}$$

- 入力層から中間層への結合係数の修正

$$\frac{\partial E_p}{\partial W_{ji}}$$

*次頁以降のスライドは，活性化関数がシグモイド関数の場合です

中間層から出力層への結合係数の修正①

$$\frac{\partial E}{\partial V_{kj}} = \boxed{\frac{\partial E}{\partial O_k}}\boxed{\frac{\partial O_k}{\partial V_{kj}}}$$

$$O_k = f(S_k)$$

$$S_k = \sum_{j=1}^{h} V_{kj} H_j$$

$$E = \sum_{k=1}^{m} (O_k - t_k)^2$$

$$= \sum_{k=1}^{m} (f(S_k) - t_k)^2$$

$$= \sum_{k=1}^{m} (f(\sum_{j=1}^{h} V_{kj} H_j) - t_k)^2$$

$$E = \sum_{k=1}^{m} (O_k - t_k)^2$$

$$\boxed{\frac{\partial E}{\partial O_k}} = 2(O_k - t_k)$$

$$\boxed{\frac{\partial O_k}{\partial V_{kj}}} = \frac{\partial O_k}{\partial S_k} \frac{\partial S_k}{\partial V_{kj}}$$

シグモイド関数の微分

$$= \frac{\partial f(S_k)}{\partial S_k} H_j = f'(S_k) H_j = f(S_k)(1 - f(S_k)) H_j$$

$$= O_k (1 - O_k) H_j$$

データpの添え字は省略

しきい値については結合係数として扱う

中間層から出力層への結合係数の修正②

$$\frac{\partial E}{\partial O_k} = 2(O_k - t_k) \qquad \frac{\partial O_k}{\partial V_{kj}} = O_k(1 - O_k)H_j$$

$$\frac{\partial E}{\partial V_{kj}} = \frac{\partial E}{\partial O_k}\frac{\partial O_k}{\partial V_{kj}} = \boxed{(O_k - t_k)O_k(1 - O_k)}H_j$$

$$= \delta_k H_j$$

$$\boxed{\delta_k = (O_k - t_k)O_k(1 - O_k)}$$

「誤差」と定義する

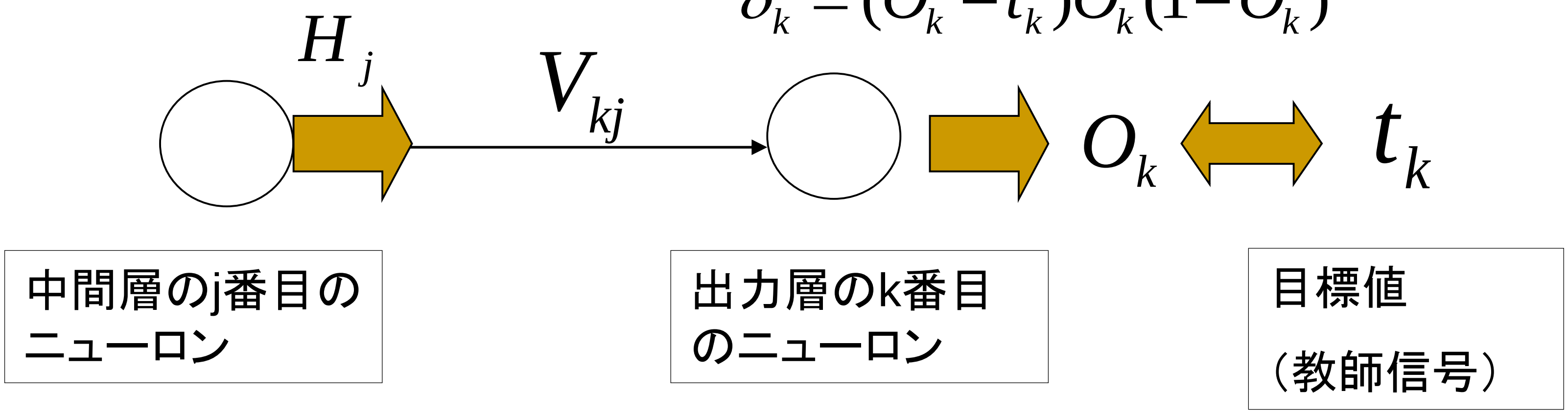
中間層から出力層への結合係数の修正方法

$$V_{kj} \leftarrow V_{kj} - \alpha \frac{\partial E}{\partial V_{kj}}$$

$$\frac{\partial E}{\partial V_{kj}} = H_j \delta_k$$

「誤差」を求める

$$\delta_k = (O_k - t_k) O_k (1 - O_k)$$



入力層から中間層への結合係数の修正①

$$E = \sum_{k=1}^{m} (O_k - t_k)^2$$

$$O_k = f(S_k) \qquad H_j = f(U_j)$$

$$S_k = \sum_{j=1}^{h} V_{kj} H_j \qquad U_j = \sum_{i=i}^{n} W_{ji} I_i$$

$$\frac{\partial E}{\partial W_{ji}} = \frac{\partial E}{\partial O_k}\frac{\partial O_k}{\partial W_{ji}} = 2\sum_{k=1}^{m}(O_k - t_k)\boxed{\frac{\partial O_k}{\partial W_{ji}}}$$

$$\boxed{\frac{\partial O_k}{\partial W_{ji}}} = \frac{\partial f(S_k)}{\partial W_{ji}} = \boxed{\frac{\partial f(S_k)}{\partial S_k}}\boxed{\frac{\partial S_k}{\partial W_{ji}}}$$

$$\boxed{f'(S_k) = f(S_k)(1 - f(S_k)) = O_k(1 - O_k)}$$

入力層から中間層への結合係数の修正②

$$\boxed{\frac{\partial S_k}{\partial W_{ji}}} = \frac{\partial \sum_{j=1}^{h} V_{kj} H_j}{\partial W_{ji}} = \frac{\partial \sum_{j=1}^{h} V_{kj} f(\sum_{i=1}^{n} W_{ji} I_i)}{\partial W_{ji}}$$

$$= V_{kj} \frac{\partial f(\sum_{i=1}^{n} W_{ji} I_i)}{\partial W_{ji}} = V_{kj} \frac{\partial f(U_j)}{\partial W_{ji}}$$

$$= V_{kj} \boxed{\frac{\partial f(U_j)}{\partial U_j}} \frac{\partial U_j}{\partial W_{ji}} = V_{kj} f(U_j)(1 - f(U_j)I_i = V_{kj} H_j (1 - H_j) I_i$$

$$\boxed{f'(U_j) = f(U_j)(1 - f(U_j)) = H_j (1 - H_j)}$$

入力層から中間層への結合係数の修正③

$$\frac{\partial E}{\partial W_{ji}} = 2\sum_{k=1}^{m}(O_k - t_k)\boxed{\frac{\partial O_k}{\partial W_{ji}}}$$

$$\boxed{\frac{\partial O_k}{\partial W_{ji}}} = O_k(1-O_k)\frac{\partial S_k}{\partial W_{ji}}$$

$$= 2\sum_{k=1}^{m}(O_k - t_k)O_k(1-O_k)\boxed{\frac{\partial S_k}{\partial W_{ji}}}$$

$$\boxed{\frac{\partial S_k}{\partial W_{ji}}} = V_{kj}H_j(1-H_j)I_i$$

$$= 2\sum_{k=1}^{m}(O_k - t_k)O_k(1-O_k)V_{kj}H_j(1-H_j)I_i$$

出力層での「誤差」 $\delta_k = (O_k - t_k)O_k(1-O_k)$

$$= 2\Big(\sum_{k=1}^{m}\delta_k V_{kj}\Big)H_j(1-H_j)I_i$$

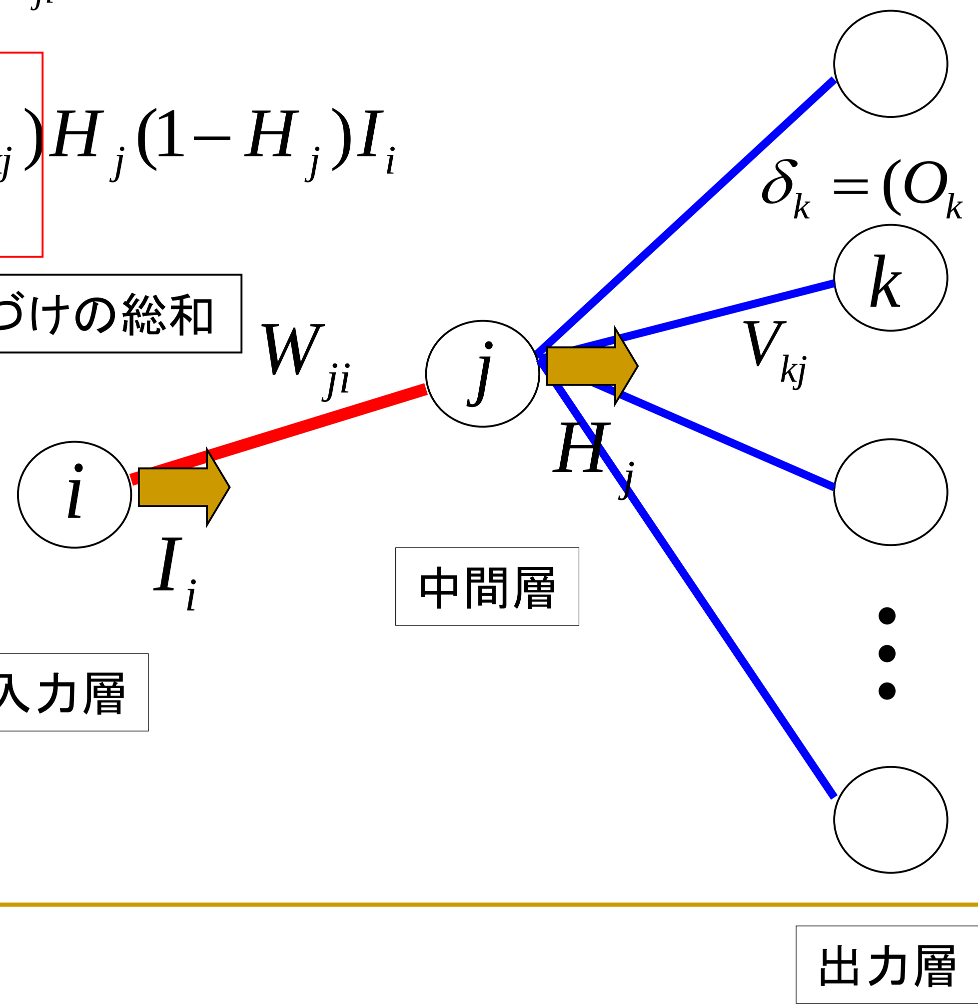
入力層から中間層への結合係数の修正方法

$$W_{ji} \leftarrow W_{ji} - \alpha \frac{\partial E}{\partial W_{ji}}$$

$$\frac{\partial E}{\partial W_{ji}} = \boxed{\left(\sum_{k=1}^{m} \delta_k V_{kj}\right)} H_j (1 - H_j) I_i$$

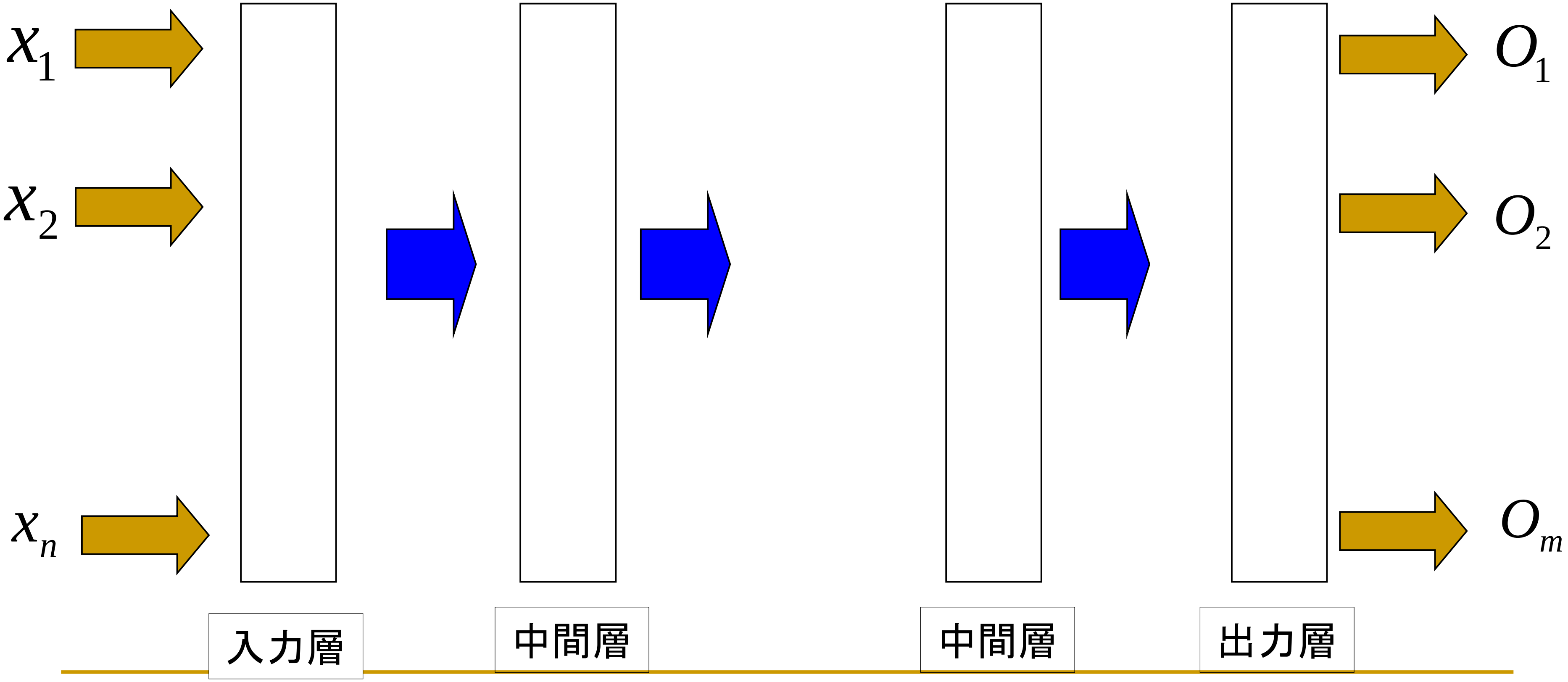
出力層の誤差の重みづけの総和

$$\delta_k = (O_k - t_k) O_k (1 - O_k)$$



一般化①

ネットワークが3層以上（中間層が複数層）の構造になった場合



一般化②

ネットワークが3層以上（中間層が複数層）の構造になった場合

$$V_{kj} \leftarrow V_{kj} - \alpha \frac{\partial E}{\partial V_{kj}}$$

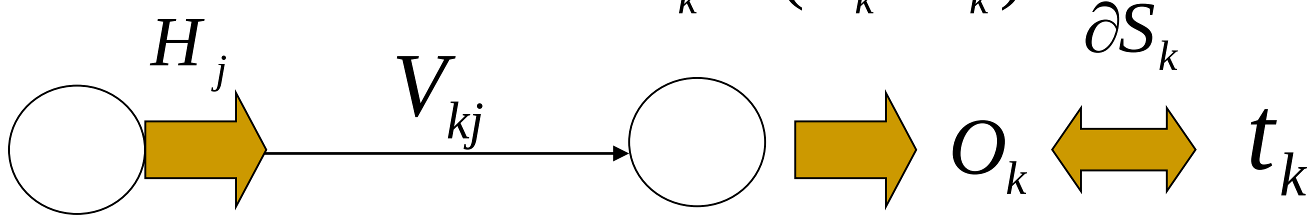
$$S_k = \sum_{j=1}^{h} V_{kj} H_j$$

$$O_k = f(S_k)$$

$$\frac{\partial E}{\partial V_{kj}} = H_j \delta_k$$

「誤差」と定義

$$\delta_k = (O_k - t_k) \frac{\partial f(S_k)}{\partial S_k}$$



中間層のj番目のニューロン

出力層のk番目のニューロン

一般化③

ネットワークが3層以上（中間層が複数層）の構造になった場合

$$W_{ji} \leftarrow W_{ji} - \alpha \frac{\partial E}{\partial W_{ji}}$$

$$\delta_j = \boxed{\left(\sum_{k=1}^{m} \delta_k V_{kj}\right)} \frac{\partial f(U_j)}{\partial U_j}$$

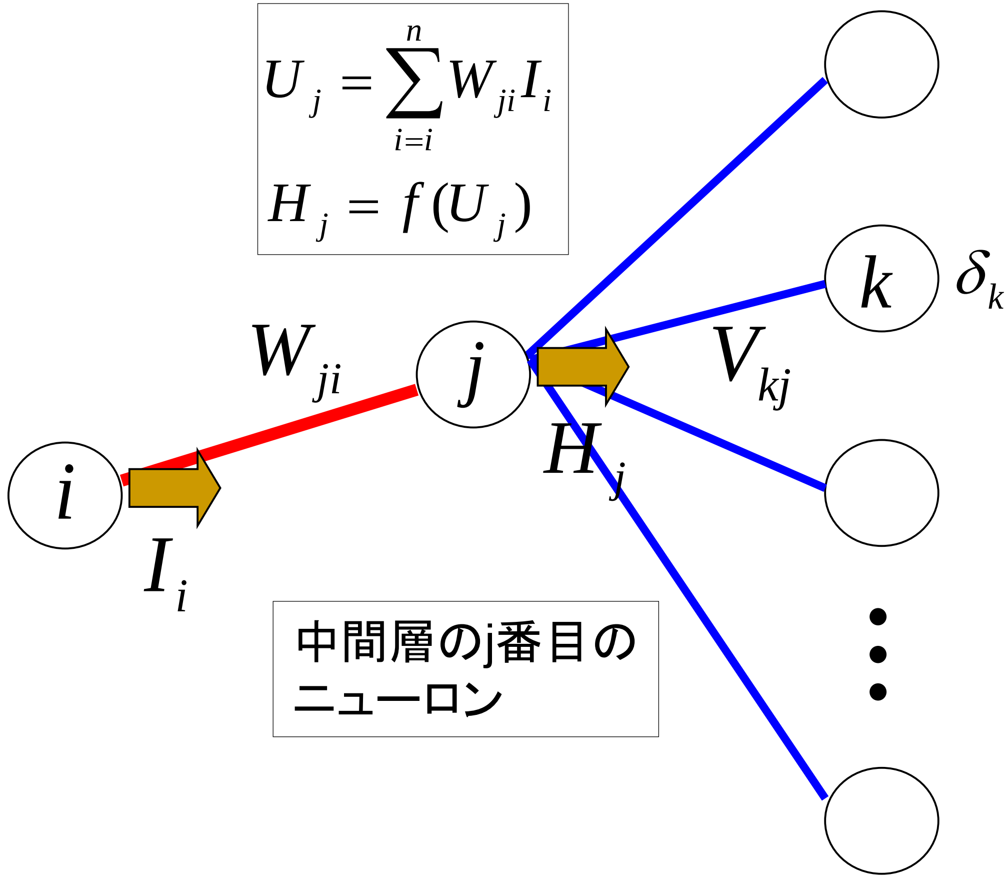
一つ上の層の誤差の重み付けの総和

「誤差」と定義

$$\frac{\partial E}{\partial W_{ji}} = \delta_j I_i$$

$$U_j = \sum_{i=i}^{n} W_{ji} I_i$$

$$H_j = f(U_j)$$



出力層→中間層→…というように逆向きに誤差を計算する

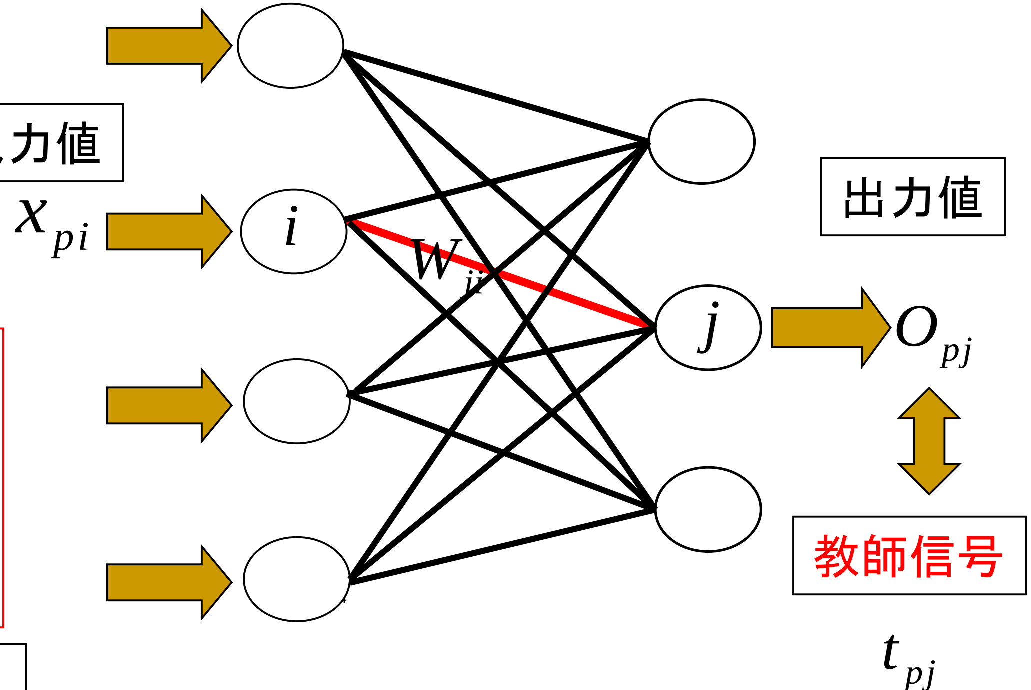
アダラインの学習則（デルタルール）

- 重みの更新方法

$$W_{ji}^{'} = W_{ji} - \alpha(O_{pj} - t_{pj})x_{pi}$$

$$= W_{ji} - \alpha(\boxed{\sum_{i=1}^{n} W_{ji}x_{pi} - t_{pj}})x_{pi}$$

アダラインの場合の誤差



3層型のニューラルネットワークの場合①

① 入力層の i 番目のニューロンに入力値を入力

$$I_i = x_i \qquad i = 1,2,\cdots,n$$

② 中間層の j 番目のニューロンの出力値を計算

$$H_j = f(\sum_{i=1}^{n} W_{ji} I_i) \qquad j = 1,2,\cdots,h$$

③ 出力層の k 番目のニューロンの出力値を計算

$$O_k = f(\sum_{j=1}^{h} V_{kj} H_j) \qquad k = 1,2,\cdots,m$$

3層型のニューラルネットワークの場合②

④ 出力層の k 番目のニューロンでの誤差を計算

$$\delta_k = (O_k - t_k) O_k (1 - O_k)$$

⑤ 中間層の j 番目のニューロンでの誤差を計算

$$\delta_j = \left(\sum_{k=1}^{m} \delta_k V_{kj}\right) H_j (1 - H_j)$$

3層型のニューラルネットワークの場合③

⑥ 中間層の j 番目のニューロンから出力層の k 番目のニューロンへの結合係数V_{kj}を修正

$$V_{kj} \leftarrow V_{kj} - \alpha \frac{\partial E}{\partial V_{kj}} \qquad \frac{\partial E}{\partial V_{kj}} = \delta_k H_j$$

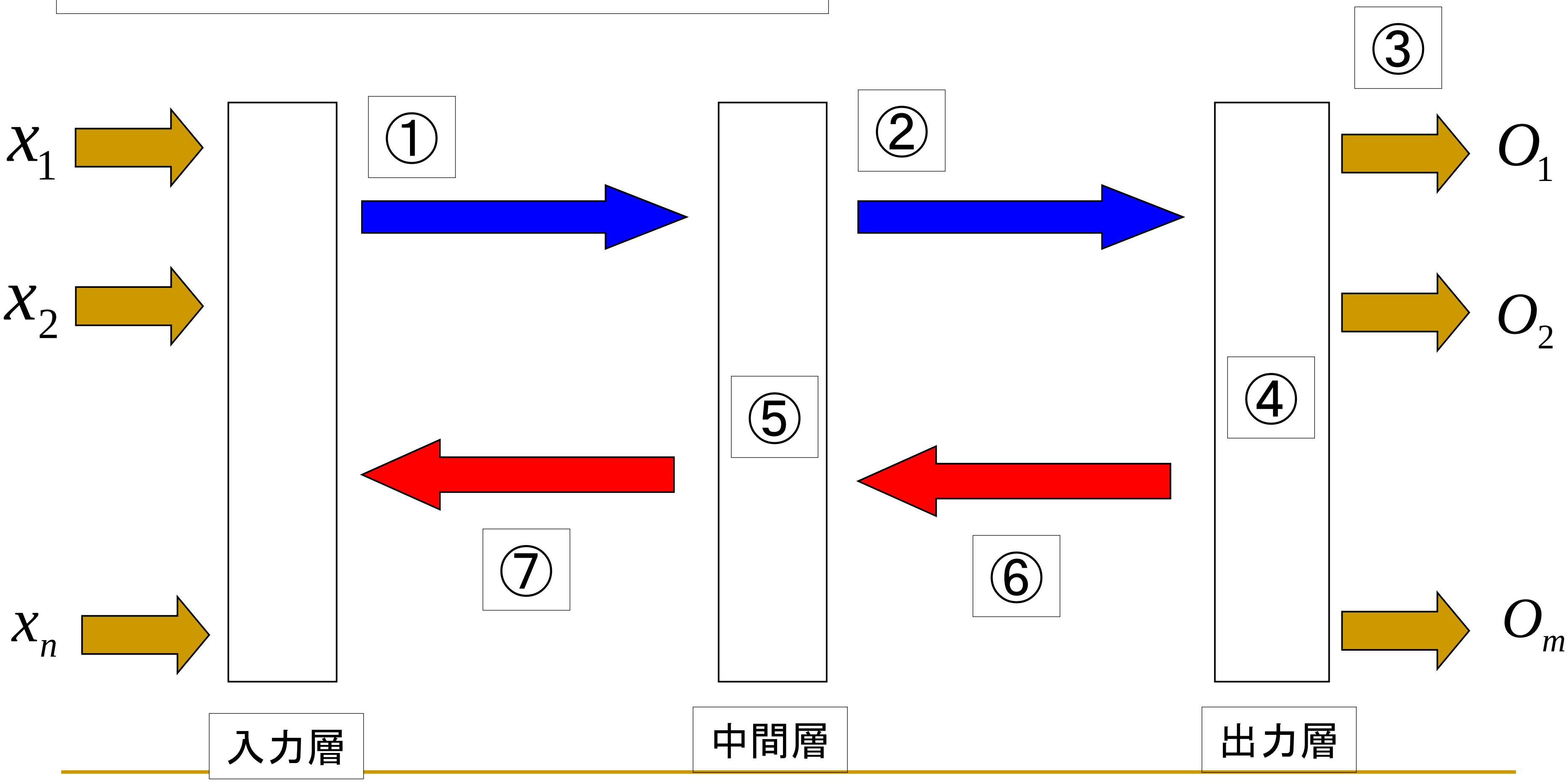
α:学習係数

⑦ 入力層の i 番目のニューロンから中間層の j 番目のニューロンへの結合係数W_{ji}を修正

$$W_{ji} \leftarrow W_{ji} - \alpha \frac{\partial E}{\partial W_{ji}} \qquad \frac{\partial E}{\partial W_{ji}} = \delta_j I_i$$

バックプロパゲーションの流れ

ネットワークが3層の構造の場合



学習アルゴリズム

*一般的には誤差二乗和を用いる

while (true) {

差の合計値 = 0

for(i = 1 ; i <= データ数 ; i++) {

$$O_{i1}, O_{i2}, \cdots O_{im} \leftarrow f(x_{i1}, x_{i2}, \cdots, x_{in})$$

プロパゲーション（①～③）

$O_{i1}, O_{i2}, \cdots O_{im}$ と $t_{i1}, t_{i2}, \cdots t_{im}$ との差*を求める

差の合計値 += 差

誤差逆伝播アルゴリズムを用いてニューラルネットワークのパラメーターを修正

バックプロパゲーション（④～⑦）

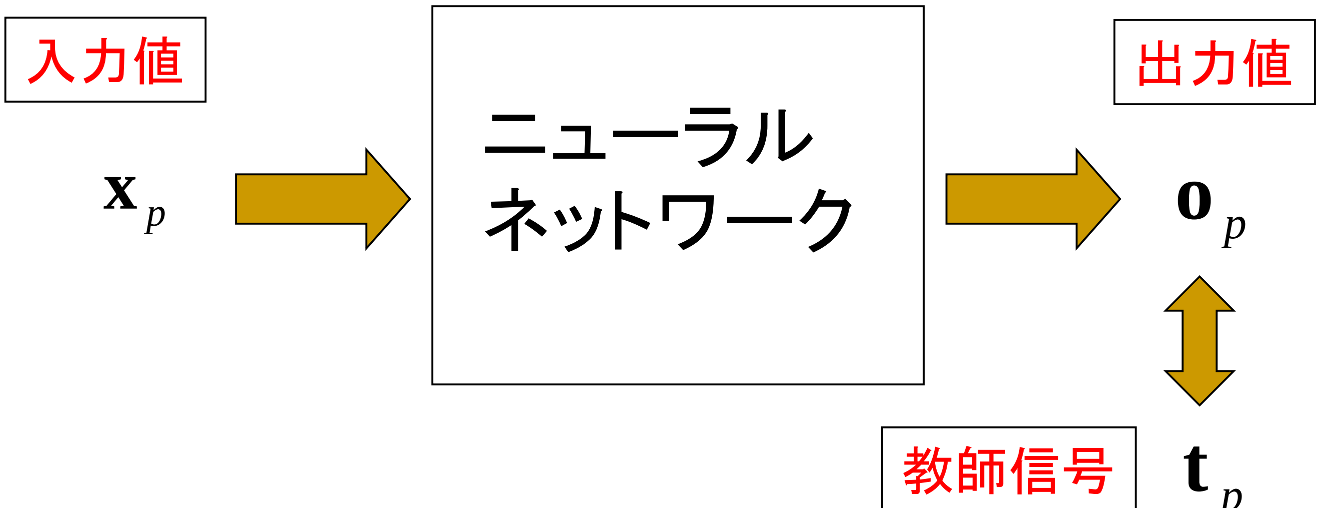
}

if 差の合計値 < ε break

}

問題設定：回帰と分類①

- 回帰の場合



損失関数 $$E = \sum_{k=1}^{m} (O_k - t_k)^2$$

問題設定：回帰と分類②

- 回帰の場合
  - 損失関数：誤差二乗和 $E=\sum_{k=1}^{m}(O_k-t_k)^2$

出力層と中間層との結合係数

S_k：出力層の内部状態

$$\frac{\partial E}{\partial V_{kj}}=\frac{\partial E}{\partial O_k}\frac{\partial O_k}{\partial V_{kj}}=\frac{\partial E}{\partial O_k}\frac{\partial O_k}{\partial S_k}\frac{\partial S_k}{\partial V_{kj}}=(O_k-t_k)f^{'}(S_k)H_j$$

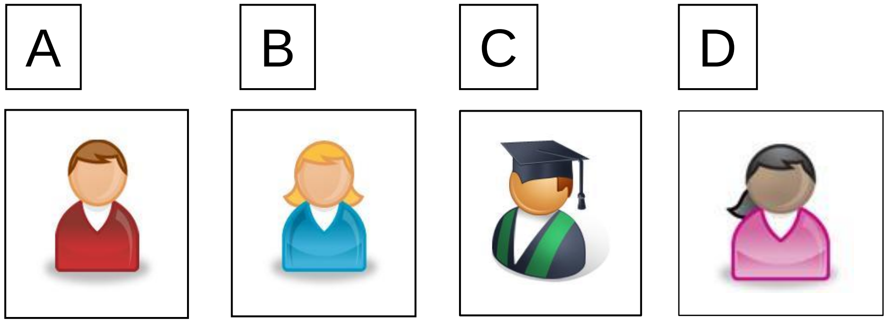
活性化関数に恒等関数を用いた場合

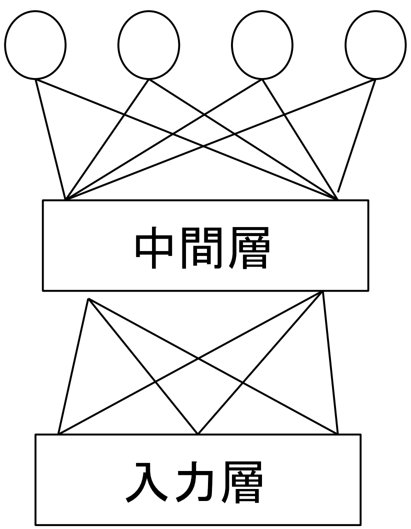
$$f^{'}(S_k)=1$$

$$\frac{\partial E}{\partial V_{kj}}=(O_k-t_k)H_j$$

H_j：中間層からの出力値

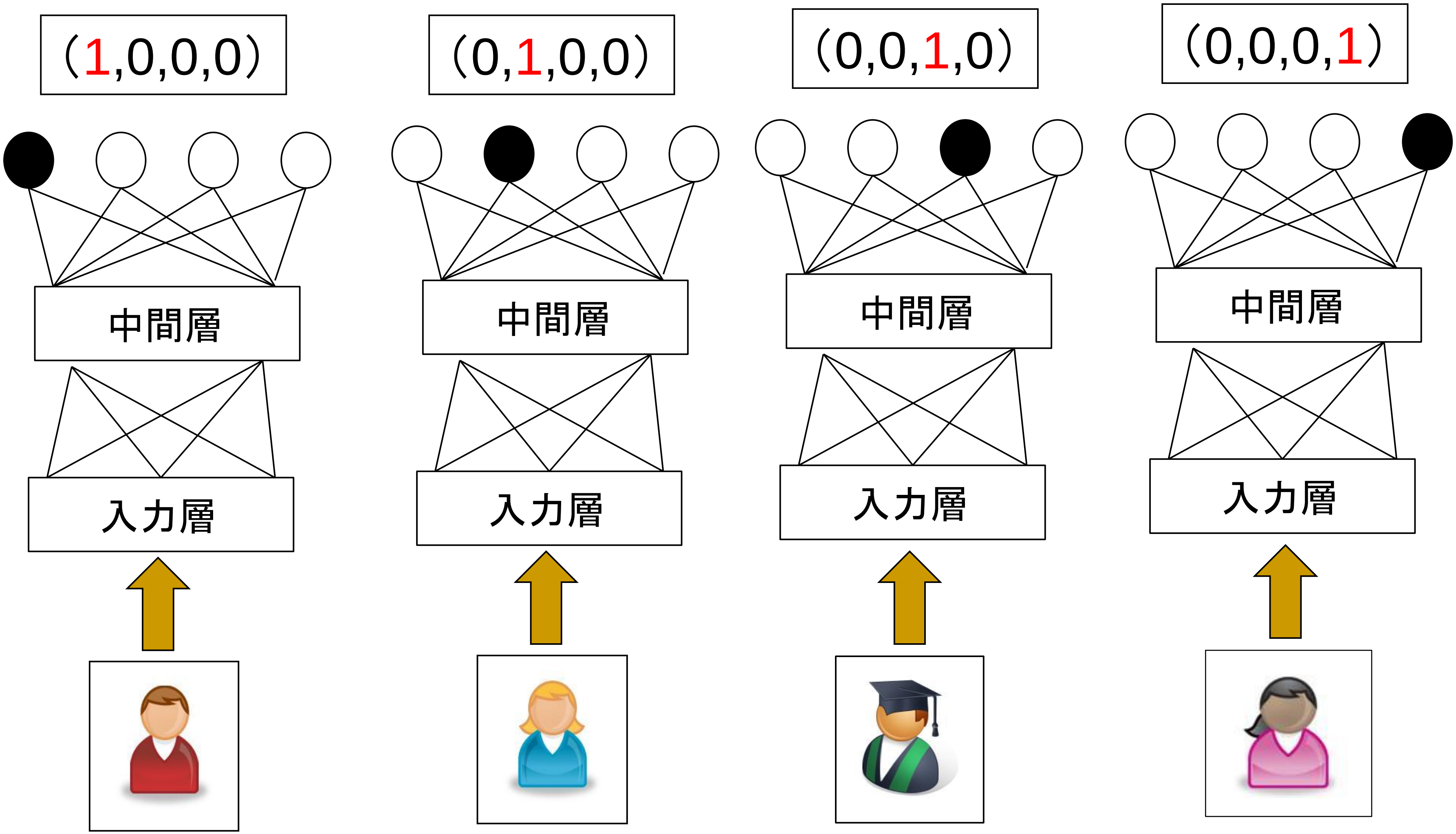
分類問題の場合①





出力層 → 4個
中間層 → 任意
入力層 → 顔画像の特徴数

分類問題の場合②



問題設定：回帰と分類③

- 分類の場合
  - 活性化関数：ソフトマックス関数

$$O_k = \frac{\exp(S_k)}{\sum_{k=1}^{m} \exp(S_k)}$$

S_k：出力層の内部状態

  - 損失関数：交差（クロス）エントロピー誤差

$$E = -\sum_{k=1}^{m} t_k \log O_k$$

問題設定：回帰と分類④

$$E = -\sum_{k=1}^{m} t_k \log O_k = -\sum_{k=1}^{m} t_k \log \frac{\exp(S_k)}{\sum_{k=1}^{m} \exp(S_k)}$$

$$= -\sum_{k=1}^{m} (t_k \log(\exp(S_k)) - t_k \log(\sum_{k=1}^{m} \exp(S_k))$$

$$= -\sum_{k=1}^{m} (t_k \log(\exp(S_k)) + \sum_{k=1}^{m} t_k \log(\sum_{k=1}^{m} \exp(S_k))$$

$$= -\sum_{k=1}^{m} t_k S_k + \log(\sum_{k=1}^{m} \exp(S_k))$$

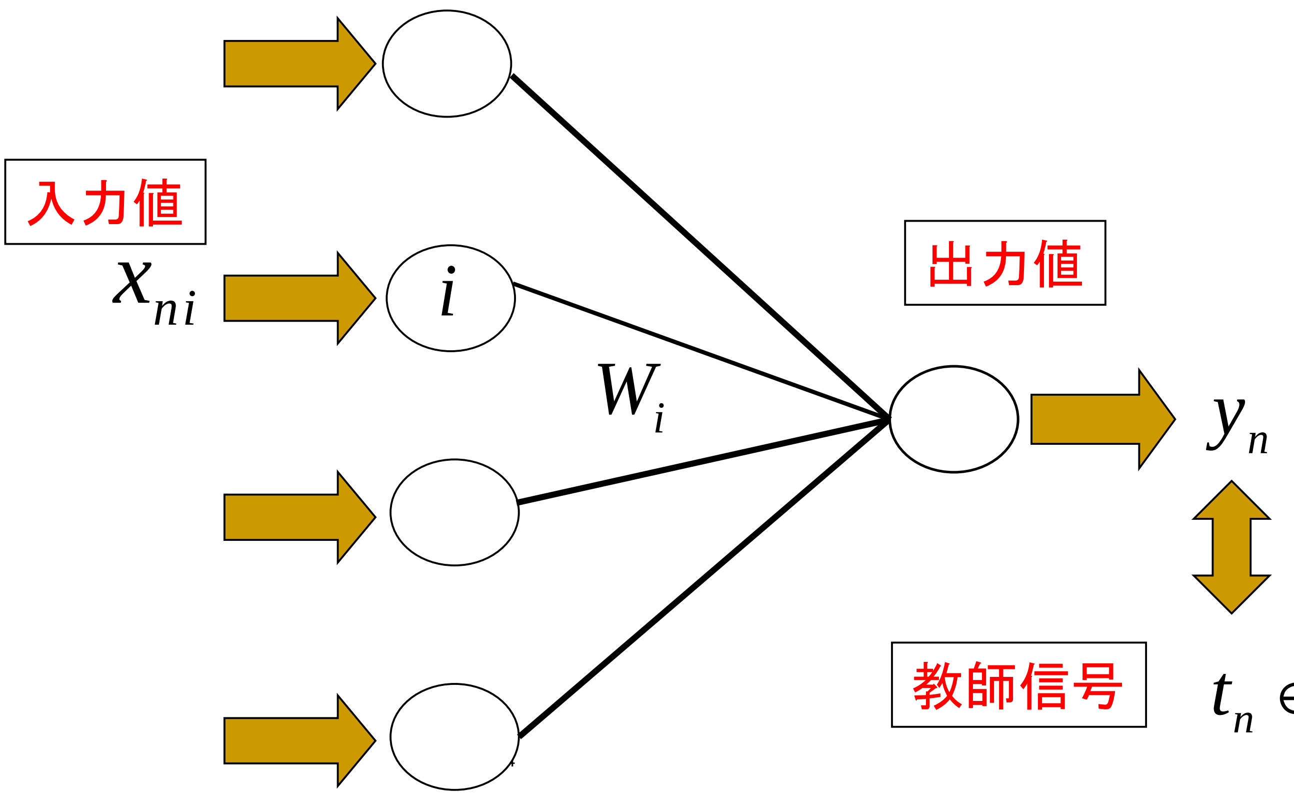
出力層と中間層の結合係数の更新

$$\frac{\partial E}{\partial S_k} = -t_k + \frac{\exp(S_k)}{\sum_{k=1}^{m} \exp(S_k)}$$

$$= -t_k + O_k$$

$$\frac{\partial E}{\partial V_{kj}} = \frac{\partial E}{\partial S_k} \frac{\partial S_k}{\partial V_{kj}} = (O_k - t_k) H_j$$

二値分類(ロジスティック回帰)①



$$u_n = \sum_i W_i x_{ni}$$

$$y_n = \frac{1}{1+\exp(-u_n)}$$

$y_n > 0.5 \rightarrow$ クラス1

$y_n < 0.5 \rightarrow$ クラス2

二値分類(ロジスティック回帰)②

$$p(t_n \mid \mathbf{x}) = y_n^{t_n}(1-y_n)^{(1-t_n)} \qquad t_n \in \{0,1\}$$

尤度関数

$$L(\mathbf{w}) = \prod_{n=1}^{N} P(t_n \mid \mathbf{x}) = \prod_{n=1}^{N} y_n^{t_n}(1-y_n)^{(1-t_n)}$$

負の対数尤度(損失関数)

$$E(\mathbf{w}) = -\log L(\mathbf{w}) = -\sum_{n=1}^{N}(t_n \log y_n + (1-t_n)\log(1-y_n))$$

$$w_i \leftarrow w_i - \frac{\partial E}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = -\sum_{n=1}^{N}(t_n - y_n)x_{ni}$$

問題設定：回帰と分類⑤

- 二値分類の場合
  - 活性化関数：シグモイド関数

$$O = \frac{1}{1+\exp(-S)}$$

S：出力層の内部状態

  - 損失関数：交差（クロス）エントロピー誤差

$$E = -\sum_{n=1}^{N}(t_n \log O_n + (1-t_n)\log(1-O_n))$$

問題点と(深層学習のための)工夫

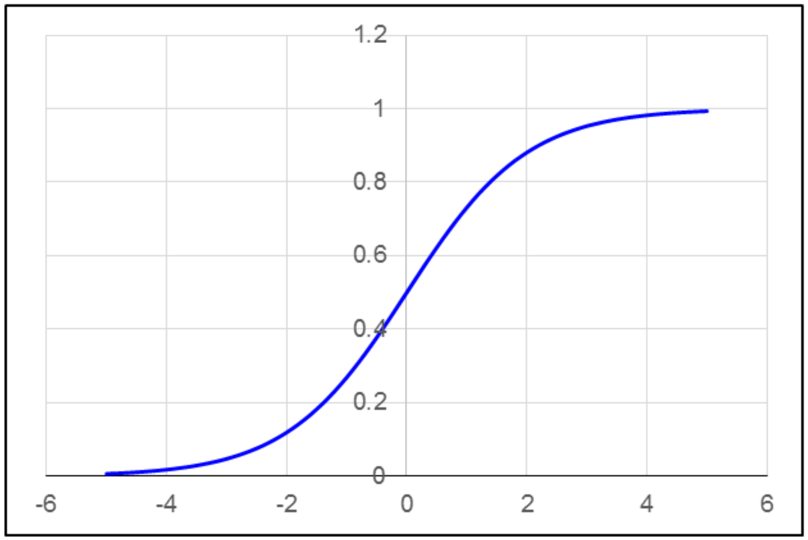
問題点

- 勾配消失問題
- ローカルミニマム問題
- 学習に多くの時間が必要
- ハイパーパラメータの設定が困難
- 過適合，過学習
- 学習後のネットワークの解析が困難

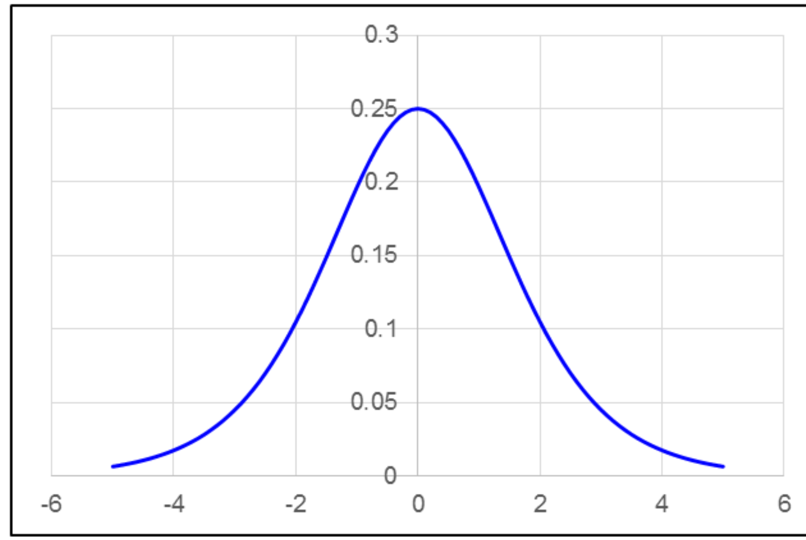
勾配消失問題①

- 以前，活性化関数にシグモイド関数を利用

シグモイド関数



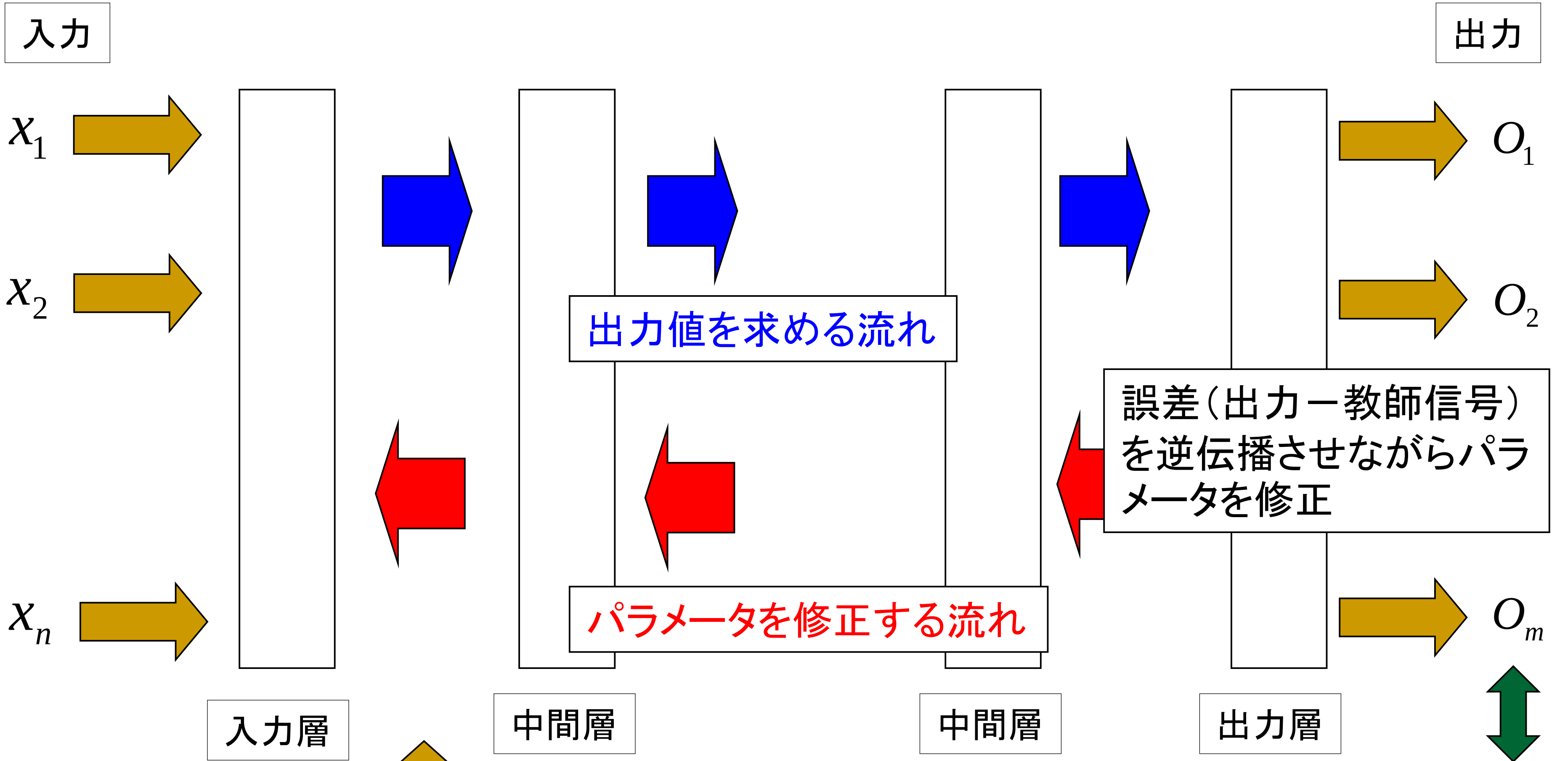
シグモイド関数の微分



$$\delta_j = \left(\sum_{k=1}^{m} \delta_k V_{kj}\right) \boxed{\frac{\partial f(U_j)}{\partial U_j}}$$

入力層に逆伝播するにつれ，誤差が小さくなる

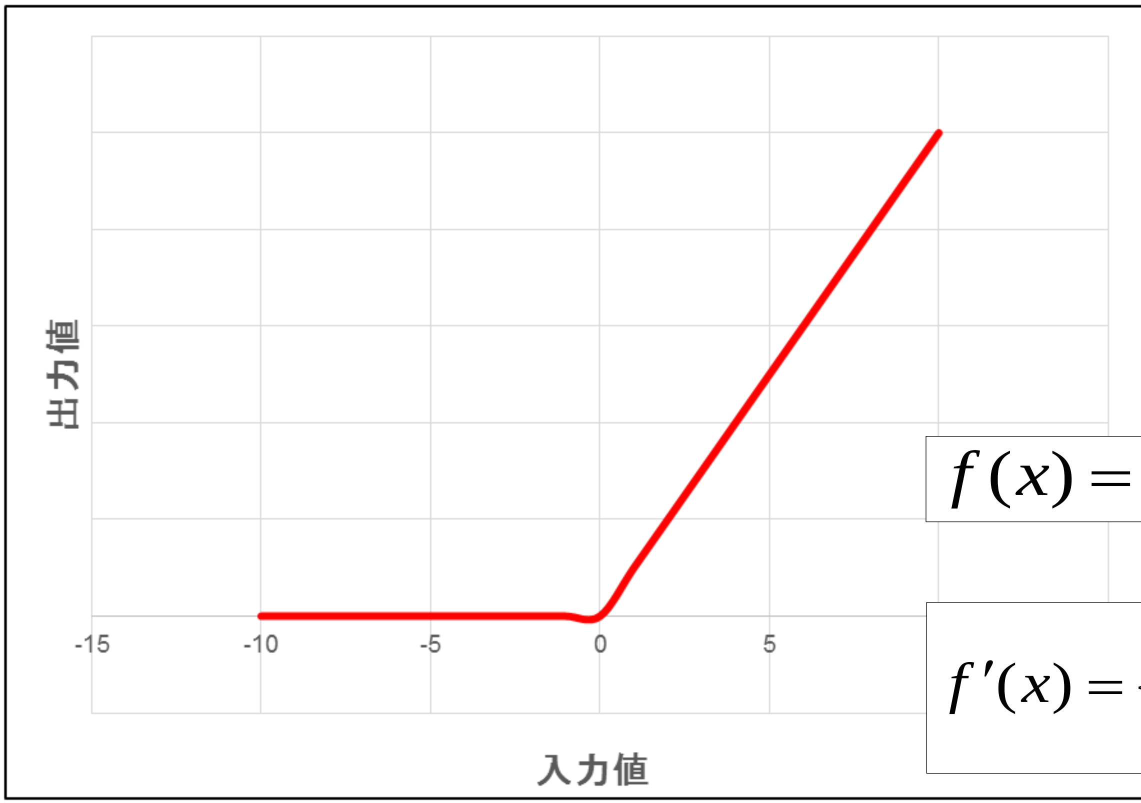
勾配消失問題②



下層ほどパラメータの修正が困難

勾配消失問題③

- 正規化線形関数（Rectified Linear Unit）



$$f(x) = \max(0, x)$$

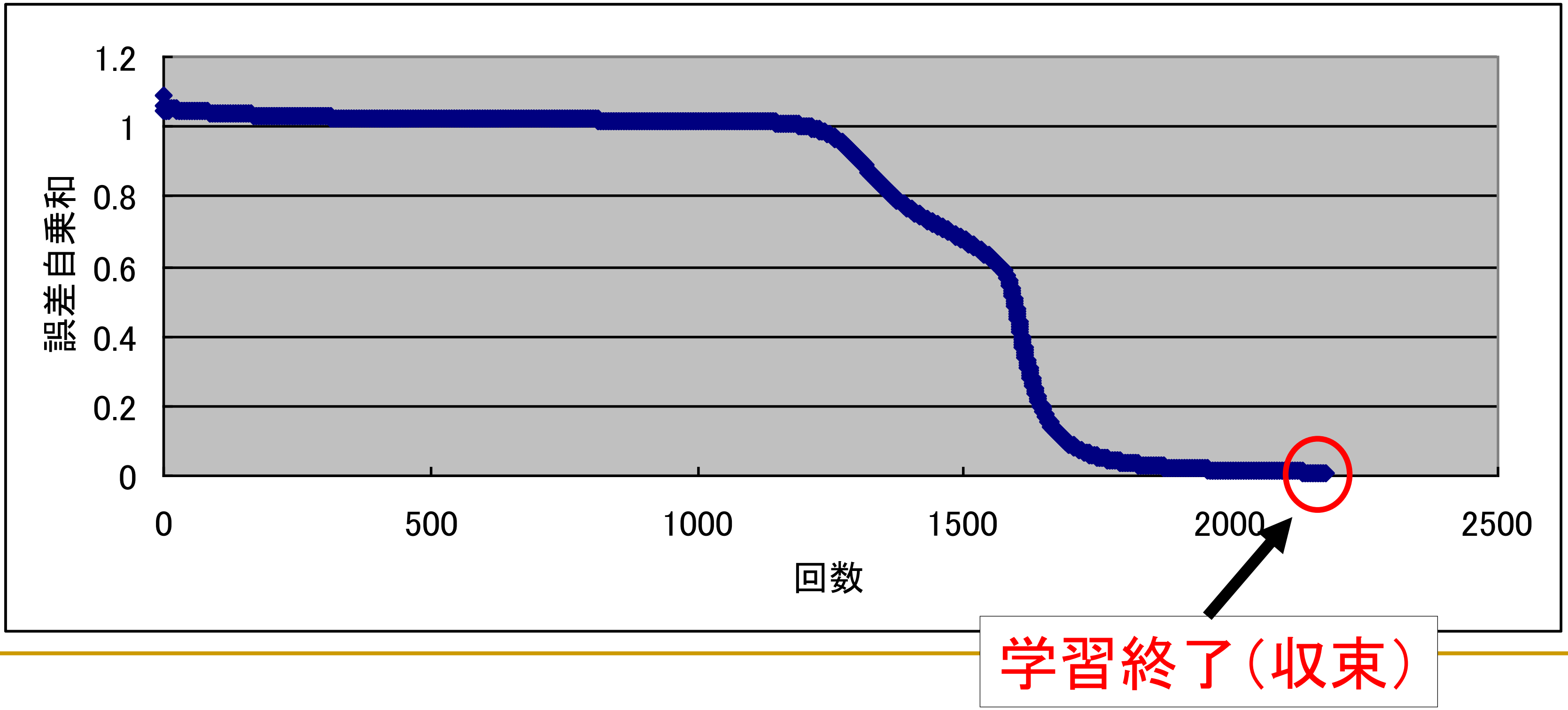
$$f'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

ローカルミニマム問題①

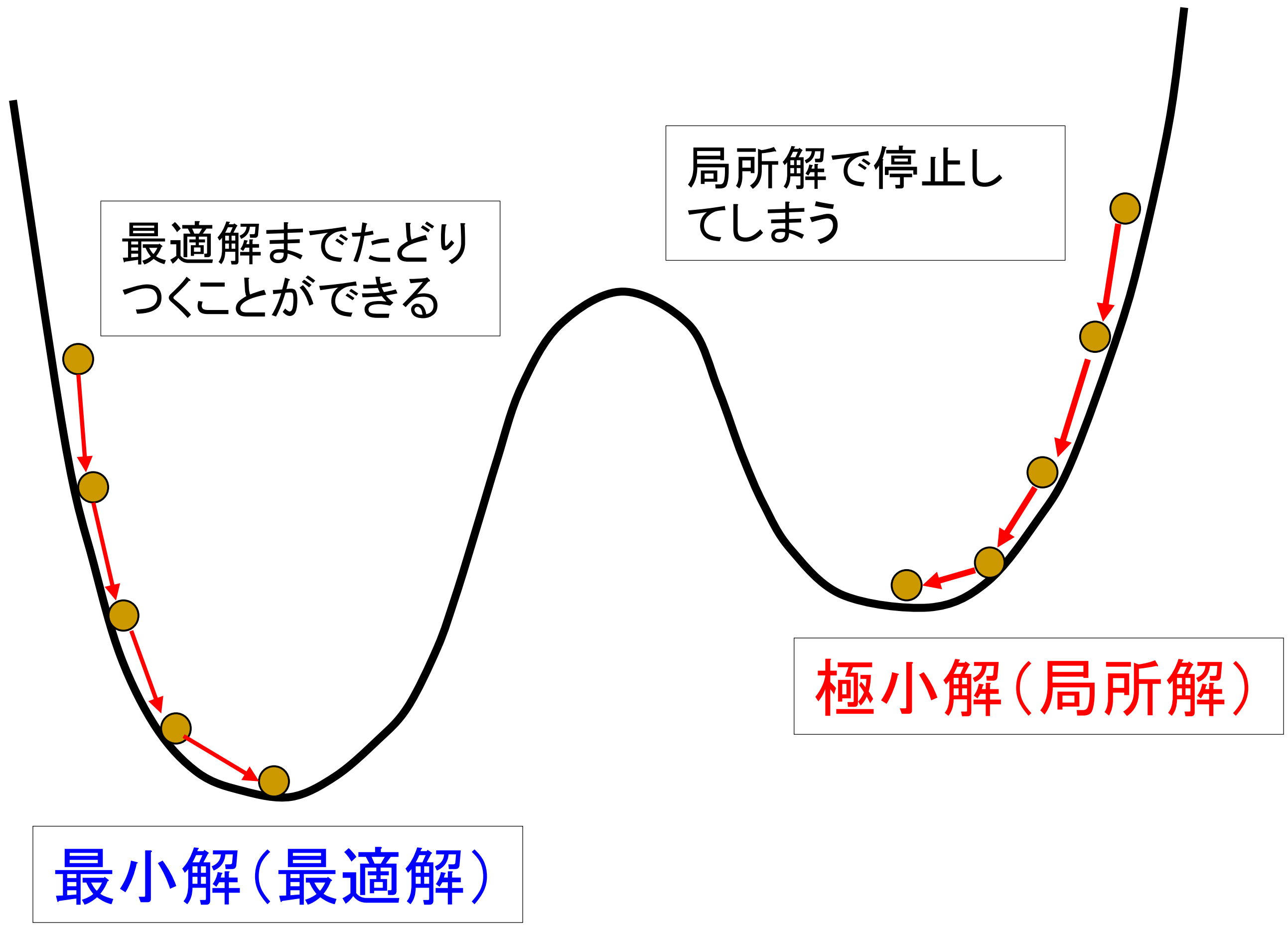
誤差二乗和

$$E=\sum_{p=1}^{P}\sum_{k=1}^{m}(O_{pk}-t_{pk})^2$$

誤差二乗和が0となれば，学習は適切に行なえたと判断し，終了する



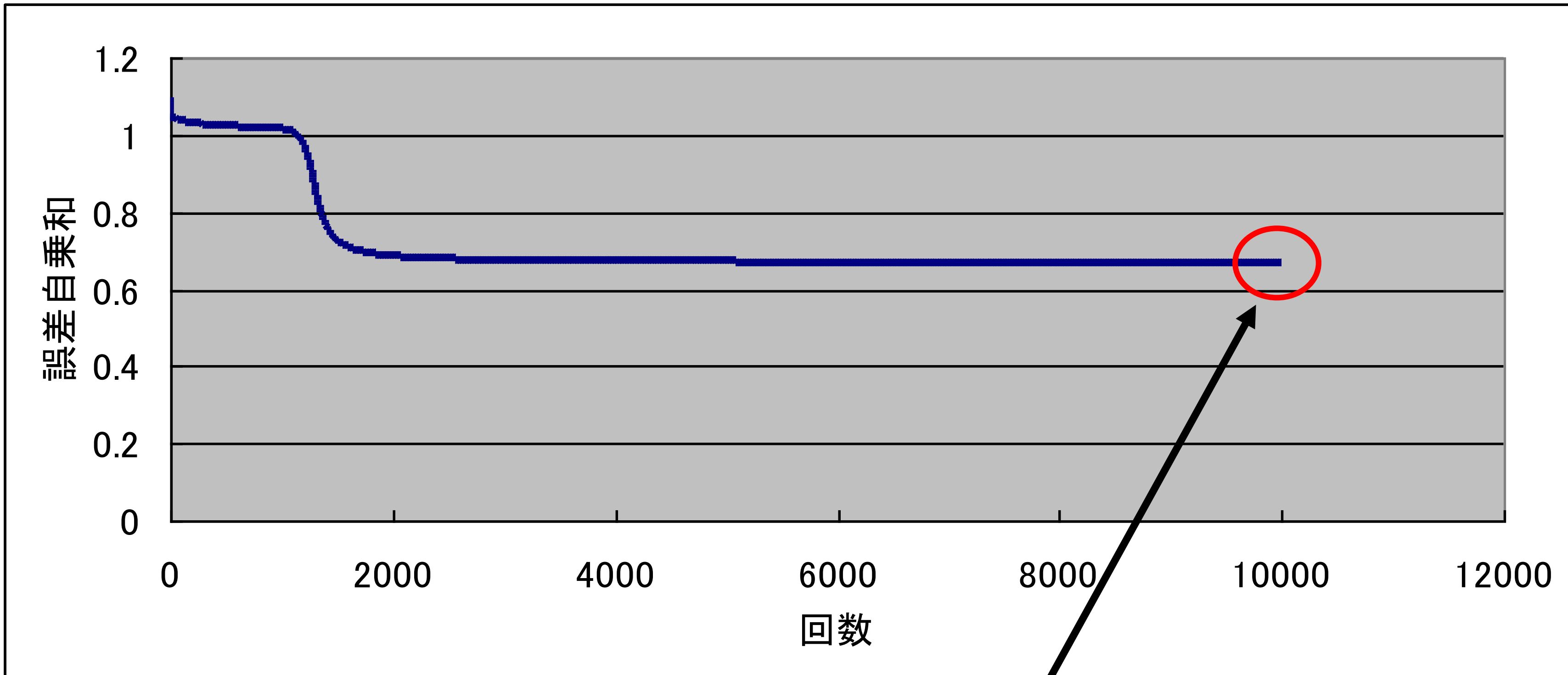
ローカルミニマム問題②



目的関数の複雑化（多峰性），初期値（開始位置）によっては最適解を求めることができず，局所解しか求まらない

ローカルミニマム問題③

学習の失敗例



誤差二乗和が0に近づかない（収束しない）

学習に多くの時間が必要
ハイパーパラメータの設定が困難

- 多層になるにつれ学習しなければならないパラメータ数（結合係数，閾値）が増加
- 学習に多くの時間が必要
- ハイパーパラメータ（学習係数，バッチサイズ，エポック数など）の調整が困難

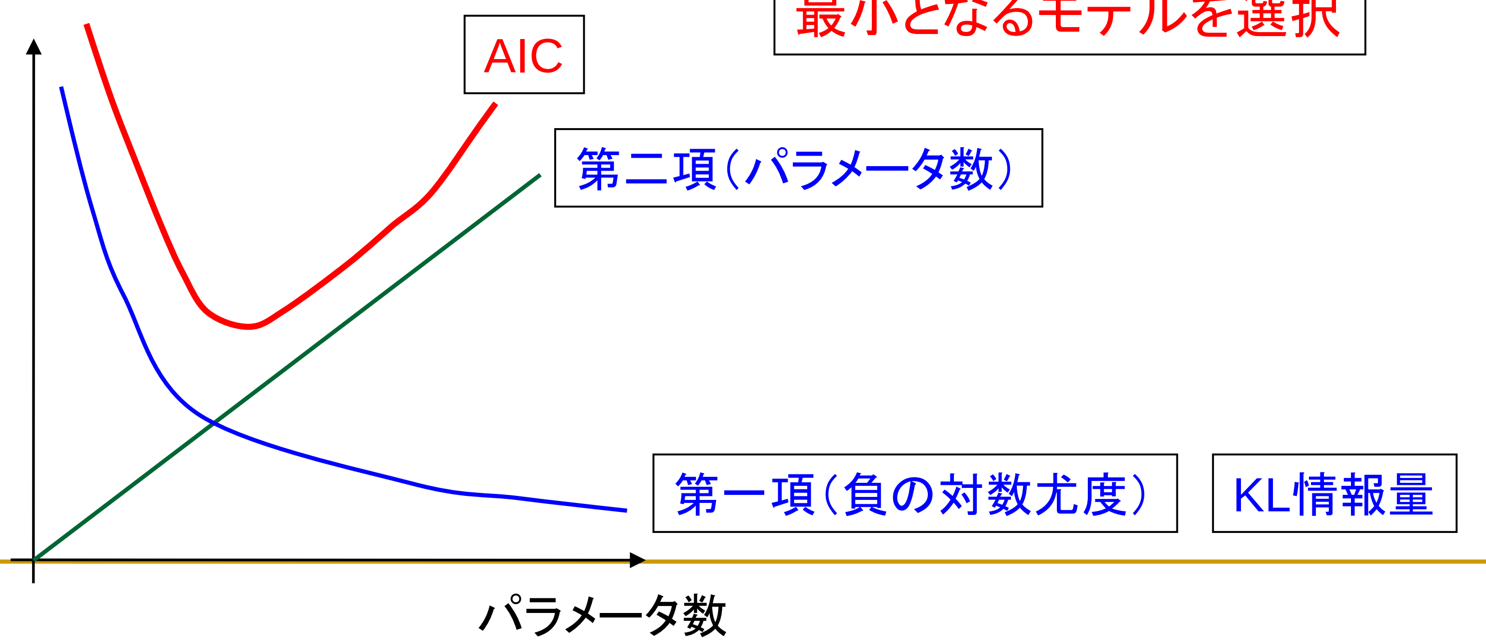
オッカムの剃刀（けちの原理）

- 現象を正しく説明できる仮説が複数ある場合，単純な仮説（パラメータの少ない）を選択
- AIC（Akaike's Information Criterion）

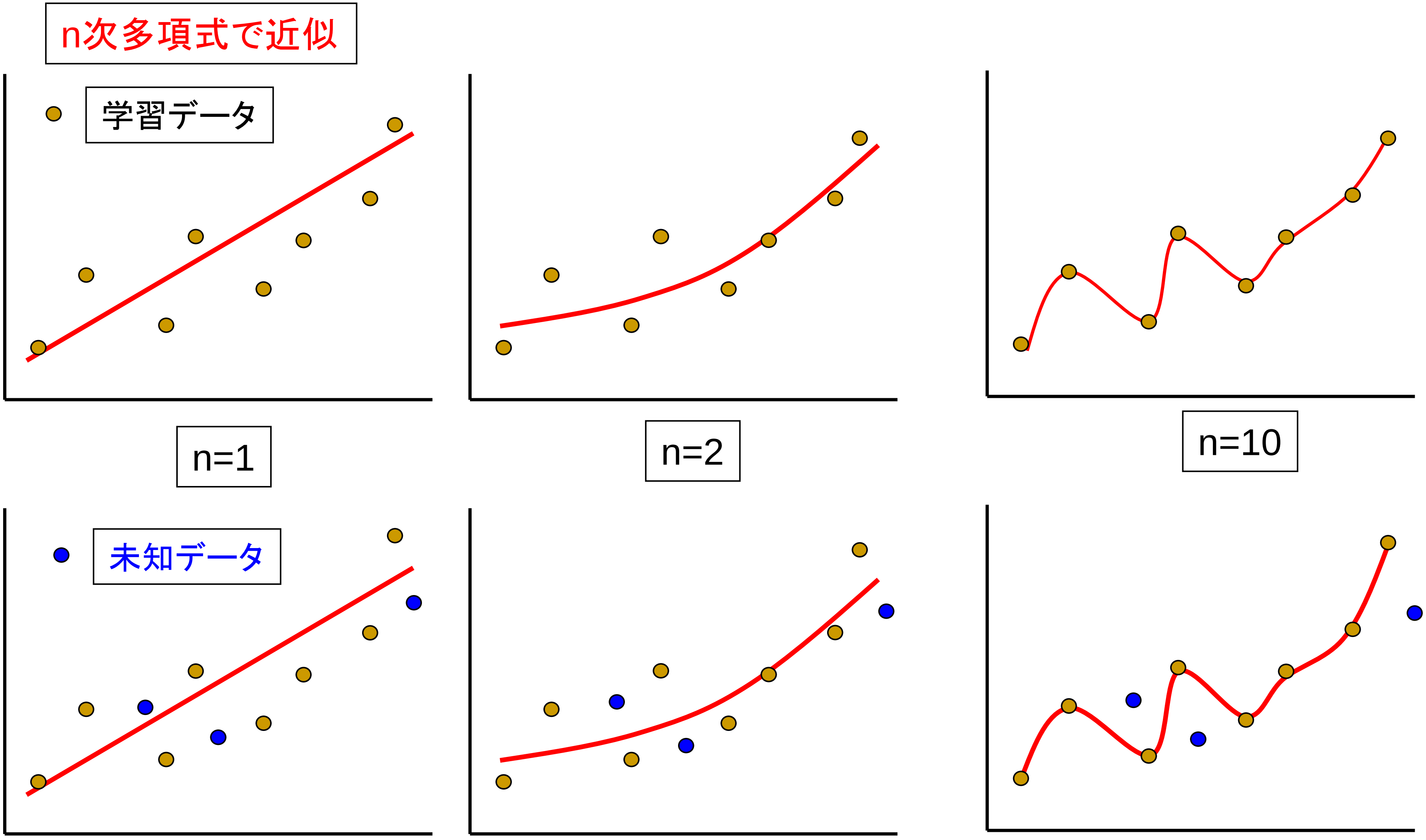
$$AIC = -\sum_i \log p(x_i;\widehat{\boldsymbol{\theta}}) + t$$

t : パラメータ$\hat{\boldsymbol{\theta}}$の次元数

最小となるモデルを選択

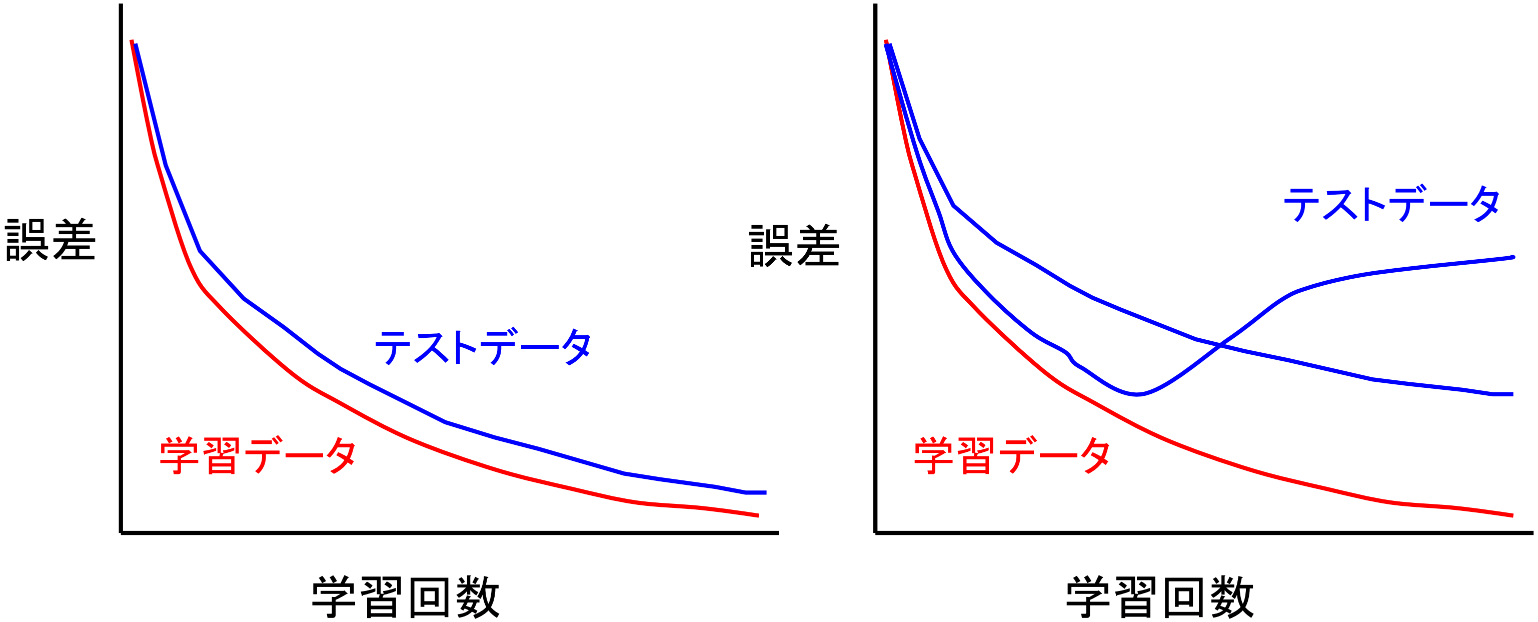


過適合(Overfitting)



過学習（Overtraining）

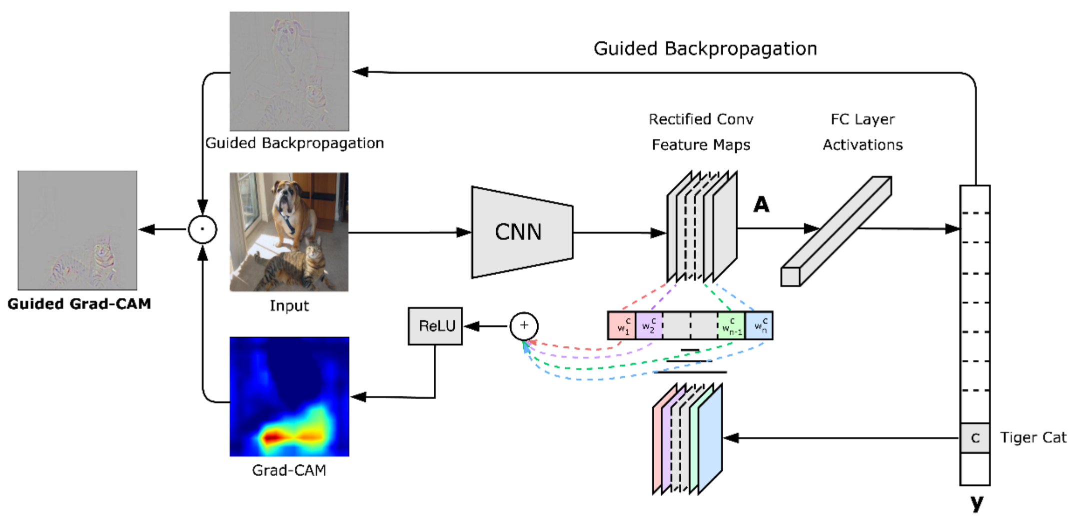
- 機械学習の最大の問題
- 学習データにのみ適用したモデルが学習され，テストデータに適用できない（汎化性の欠如）



学習後のネットワークの解析が困難

- なぜ上手くいったのか，説明が困難
- ネットワークの内部がブラックボックス

Grad-CAM



Ramprasaath R，Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization，2017

問題点のための工夫

- 結合係数の更新
- バッチサイズ
- 正則化
- ドロップアウト
- 早期終了
- 結合係数の初期化
- データの正規化

結合係数の更新①

- 確率的勾配降下法（SGD：Stochastic Gradient Descent）
  - ランダムに学習データを選択し，更新

$$W_{ji} \leftarrow W_{ji} - \alpha \frac{\partial E}{\partial W_{ji}}$$

- モーメント法

$$W_{ji} \leftarrow W_{ji} - \alpha \frac{\partial E}{\partial W_{ji}} + \beta \Delta W_{ji}$$

前回の更新値
→これまでの更新の影響を受ける

結合係数の更新②

- AdaGrad

更新量を自動的に調整

これまでの更新量が多い → 更新値は小さい
これまでの更新量が少ない → 更新値は大きい

$$h \leftarrow h + \left(\frac{\partial E}{\partial W_{ji}} \right)^2$$

$$W_{ji} \leftarrow W_{ji} - \alpha \frac{1}{\sqrt{h}} \frac{\partial E}{\partial W_{ji}}$$

- RMSProp

$$h \leftarrow \rho h + (1 - \rho) \left(\frac{\partial E}{\partial W_{ji}} \right)^2$$

$$W_{ji} \leftarrow W_{ji} - \alpha \frac{1}{\sqrt{h}} \frac{\partial E}{\partial W_{ji}}$$

結合係数の更新③

- AdaDelta

$$h_t \leftarrow \rho h_{t-1} + (1-\rho)\left(\frac{\partial E}{\partial W_{ji}}\right)^2$$

$$v_t \leftarrow \frac{\sqrt{s_t + \varepsilon}}{\sqrt{h_t + \varepsilon}} \frac{\partial E}{\partial W_{ji}}$$

$$s_{t+1} \leftarrow \rho s_t + (1-\rho) v_t^2$$

$$W_{ji} \leftarrow W_{ji} - v_t$$

結合係数の更新④

- Adam

$$m_t \leftarrow \alpha m_{t-1} + (1-\alpha)\frac{\partial E}{\partial W_{ji}}$$

$$v_t \leftarrow \beta v_{t-1} + (1-\beta)\left(\frac{\partial E}{\partial W_{ji}}\right)^2$$

$$\hat{m}_t \leftarrow \frac{m_t}{1-\alpha^t}$$

$$\hat{v}_t \leftarrow \frac{v_t}{1-\beta^t}$$

$$W_{ji} \leftarrow W_{ji} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$$

学習データ（バッチ）

学習データ

入力値1	教師信号1
入力値2	教師信号2
入力値3	教師信号3
入力値4	教師信号4
入力値N-2	教師信号N-2
入力値N-1	教師信号N-1
入力値N	教師信号N

入力層への入力→バッチ

バッチサイズ＝1

オンライン学習（SGD）

ミニバッチ学習

バッチ学習

バッチサイズ＝全データ

バッチサイズ①

学習データ

	入力値1	教師信号1
	入力値2	教師信号2
バッチ	入力値3	教師信号3
	入力値4	教師信号4
	入力値N-2	教師信号N-2
	入力値N-1	教師信号N-1
	入力値N	教師信号N

確率的勾配降下法（SGD）

$$E = E_p$$

$$W_{ji} \leftarrow W_{ji} - \alpha \frac{\partial E}{\partial W_{ji}}$$

バッチサイズ=1

オンライン学習

バッチサイズ②

学習データ

バッチ

入力値1	教師信号1
入力値2	教師信号2
入力値3	教師信号3
入力値4	教師信号4
入力値N-2	教師信号N-2
入力値N-1	教師信号N-1
入力値N	教師信号N

バッチ学習

$$E = \frac{1}{N}\sum_{p=1}^{N} E_p$$

$$W_{ji} \leftarrow W_{ji} - \alpha \sum_{p=1}^{N} \frac{\partial E}{\partial W_{ji}}$$

バッチサイズ＝全学習データ

バッチサイズ③

学習データ

バッチ

入力値1 教師信号1

入力値2 教師信号2

入力値3 教師信号3

入力値4 教師信号4

入力値N-2 教師信号N-2

入力値N-1 教師信号N-1

入力値N 教師信号N

ミニバッチ学習

$$E = \frac{1}{n}\sum_{p=1}^{n} E_p$$

$$W_{ji} \leftarrow W_{ji} - \alpha \sum_{p=1}^{n} \frac{\partial E}{\partial W_{ji}}$$

バッチサイズ=n

バッチサイズ④

- エポック
  - 全ての学習データを1回学習・・・1エポック

  - （例）学習データ数1,000個
  - オンライン学習　1エポックにつき1,000回更新
  - バッチ学習　　　1エポックにつき1回更新
  - ミニバッチ学習　 バッチサイズ=100の場合，10回更新

学習するパターンの順序

- 偏りのある順序

「0」のニューロンの発火が強まる

「1」のニューロンの発火が強まる

- 偏りのない順序

パラメーター数

- 多層になるにつれ学習しなければならないパラメータ数（結合係数，閾値）が増加

- 結合係数の値，結合方法に制約を設ける
  - 正則化
  - ドロップアウト

正則化①

- 正則化
  - 結合係数に制約を与える

$$E'(\mathbf{w}) = E(\mathbf{w}) + \lambda R(\mathbf{w})$$

$$E'(\mathbf{w}) = E(\mathbf{w}) + \lambda \mathbf{w}^t \mathbf{w}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \left(\alpha \frac{\partial E}{\partial w} + \lambda \mathbf{w} \right)$$

λは極めて小さい値

重み減衰（weight decay）

正則化②

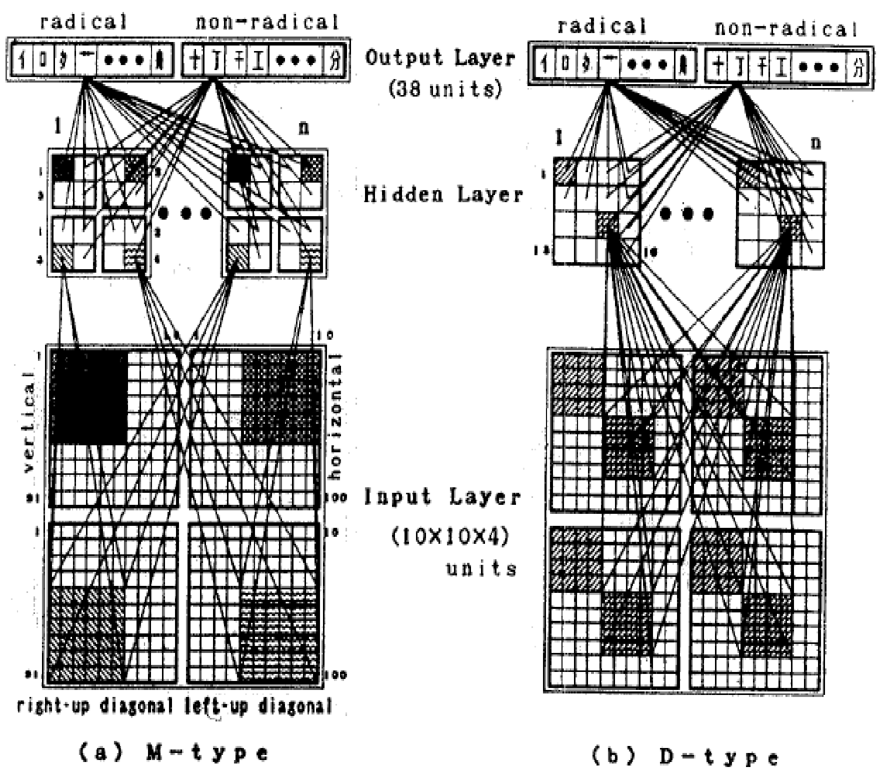
- 重み上限
  - ニューロンjへの結合係数の二乗和の上限を制限

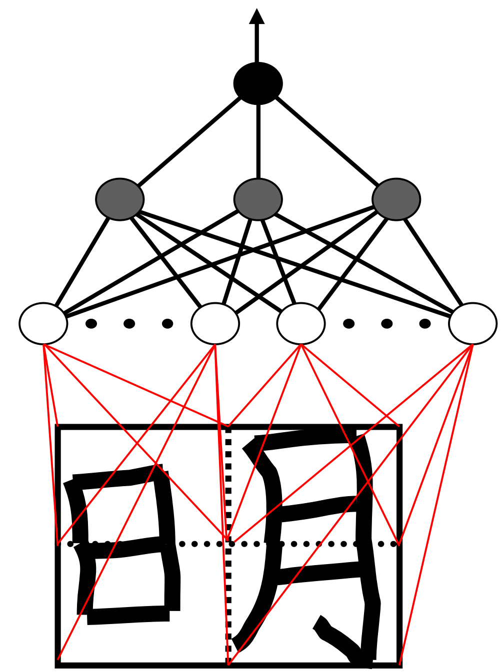
$$\sum_j w_{ji}^2 < d$$

  - 超えた場合は，定数倍し，d以下とする

局所結合

- 結合を制約→層間の結合を限定

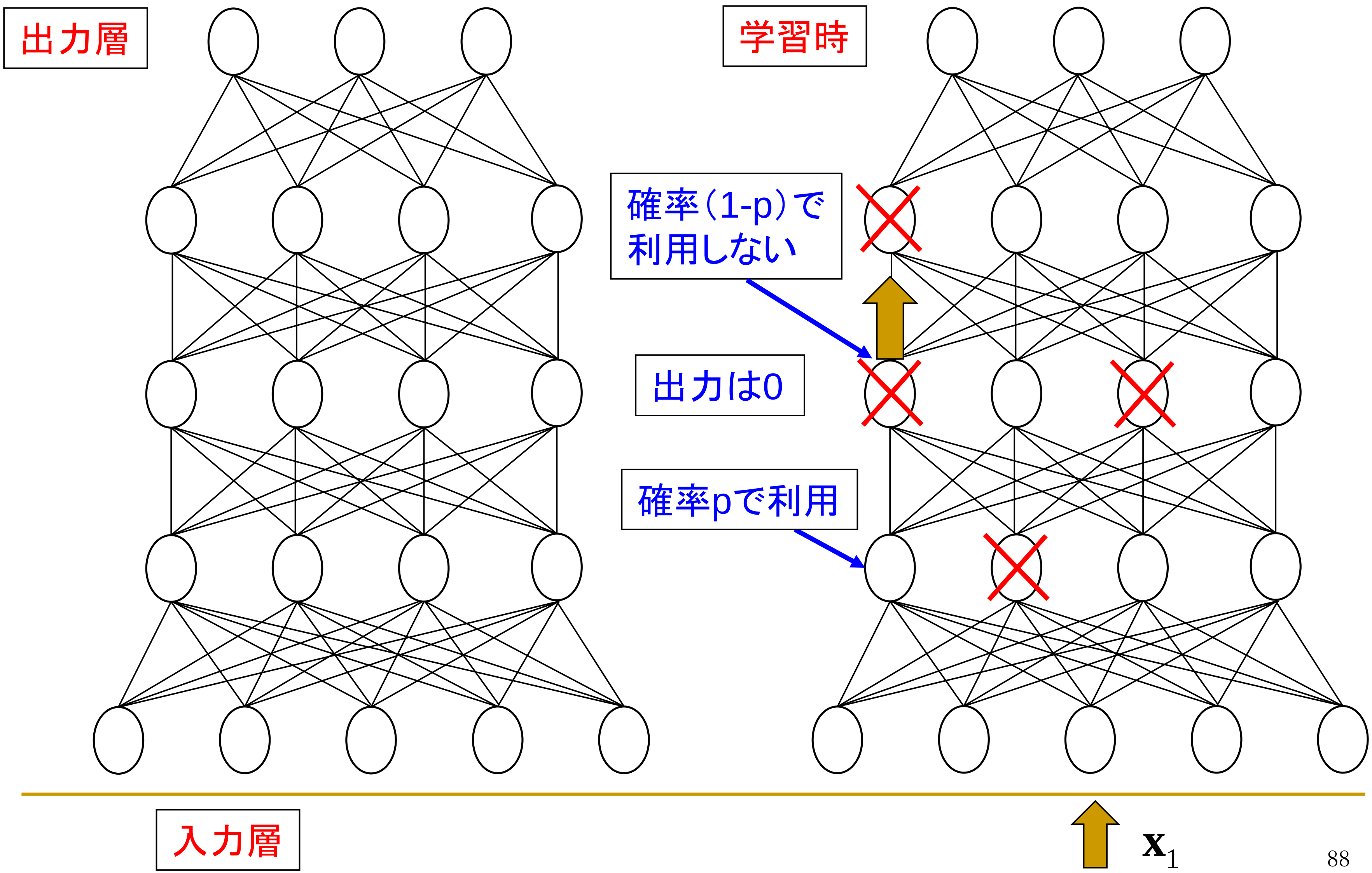




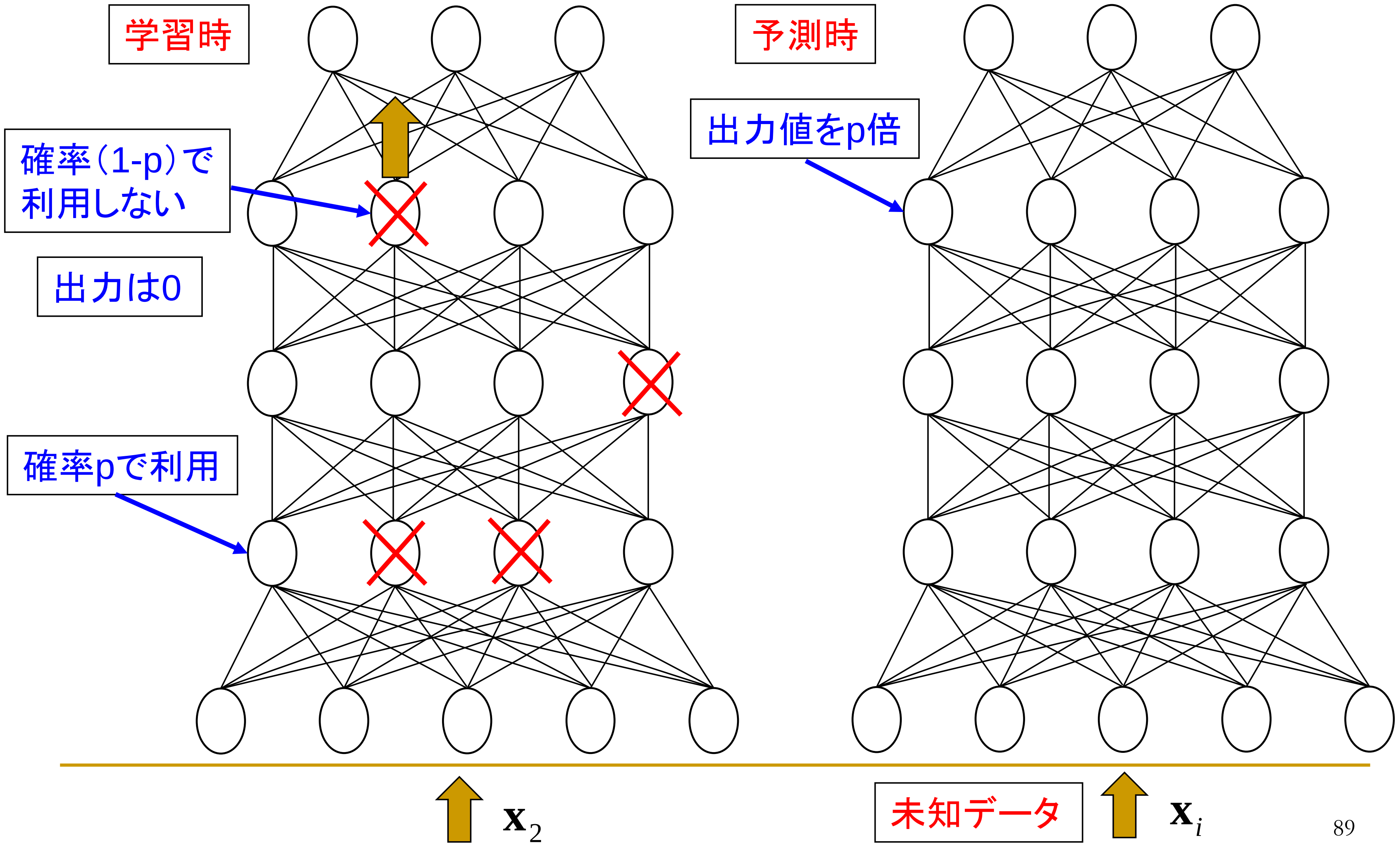
局所結合型ネットワーク

図:大友,局所結合型神経回路網モデルによる手書き漢字の効率的認識法,1994

ドロップアウト①

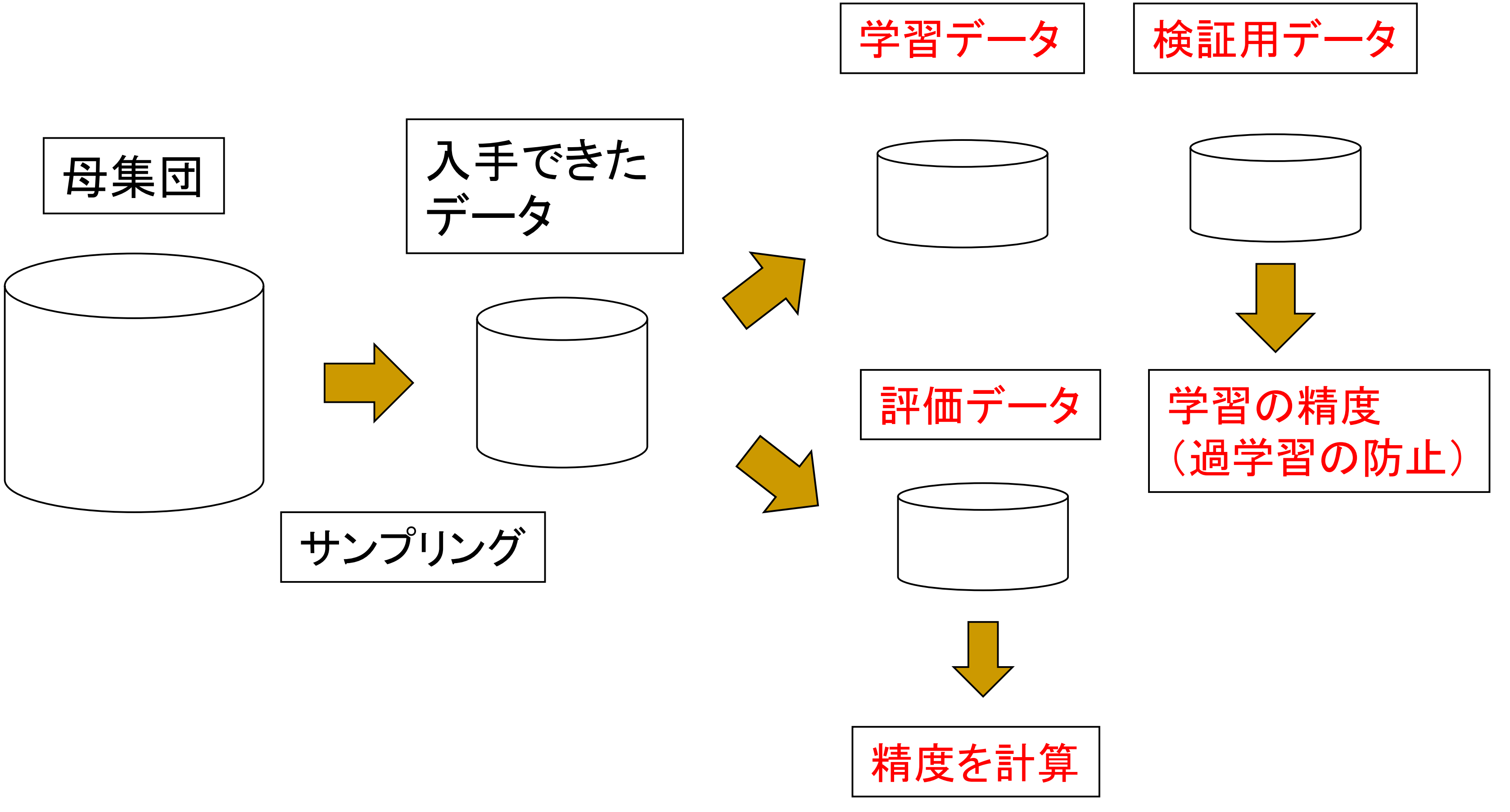


ドロップアウト②

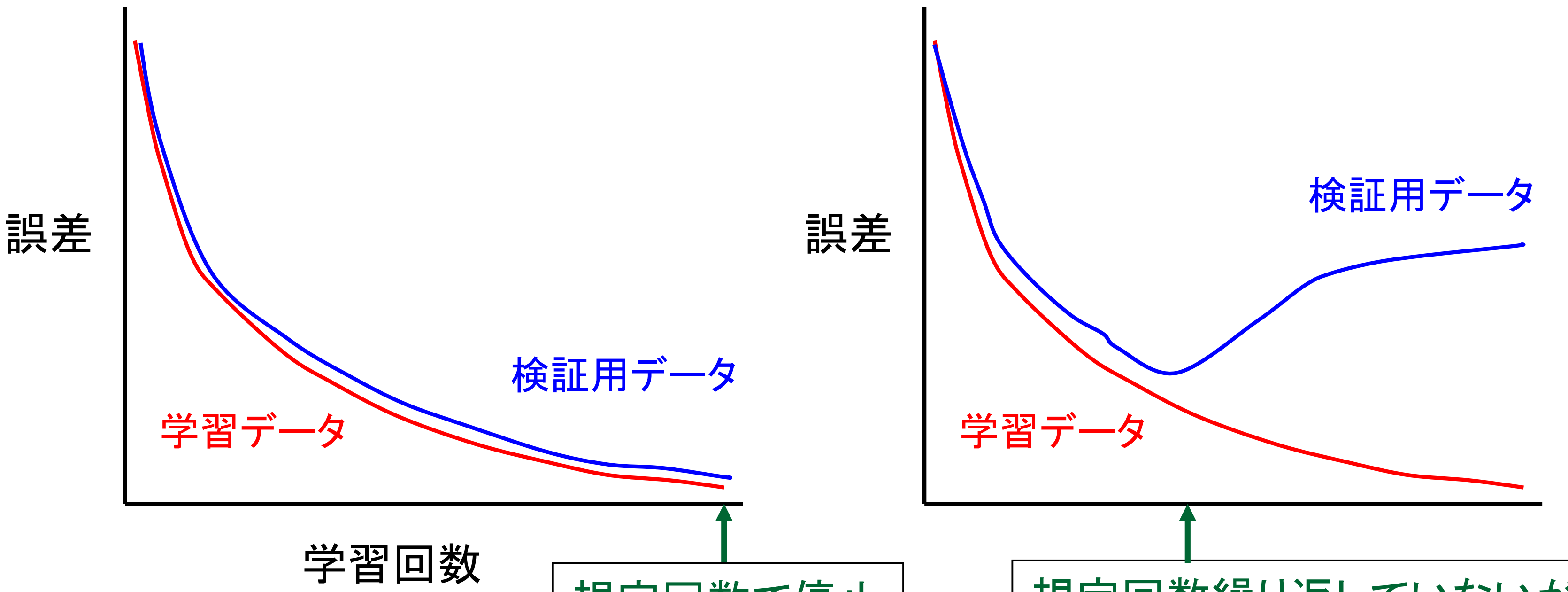


学習時
確率(1-p)で
利用しない
出力は0
確率pで利用
予測時
出力値をp倍
$\mathbf{x}_2$
未知データ
$\mathbf{x}_i$

検証用データの利用



早期終了（Early Stopping）



規定回数で停止

規定回数繰り返していないが，検証用データの誤差が大きくなって来たので，早期に終了

結合係数の初期化①

- 結合係数の初期化
  - 全て同じ値ではならない
  - 一様分布
  - 正規分布

- LeCunの初期化
  - L層目の結合係数

$$u\left(-\sqrt{\frac{3}{d_{L-1}}},\sqrt{\frac{3}{d_{L-1}}}\right)$$

$$N\left(0,\sqrt{\frac{1}{d_{L-1}}}\right)$$

d_{L-1} :（L-1）層のニューロン数

一様分布　　正規分布

結合係数の初期化②

- Xavierの初期化

$$u\left(-\sqrt{\frac{6}{d_{L-1}+d_L}},\sqrt{\frac{6}{d_{L-1}+d_L}}\right) \qquad N\left(0,\sqrt{\frac{2}{d_{L-1}+d_L}}\right)$$

d_{L-1} : (L-1)層のニューロン数

d_L : L層のニューロン数

一様分布　　正規分布

- 事前学習の結果を再学習（後述）
  - Stacked オートエンコーダによる事前学習（pre-training）
  → 再学習（fine-tuning）

データの正規化①

- 学習データ x_{pi}（p=1,2,・・・,P, i=1,2,・・・,N）
- 平均 $$\bar{x}_i = \frac{1}{P}\sum_{p=1}^{P} x_{pi}$$
- 標準偏差 $$\sigma_i = \frac{1}{P}\sum_{p=1}^{P}(x_{pi} - \bar{x}_i)^2$$

$$\boxed{y_i = \frac{(x_{pi} - \bar{x}_i)}{\sigma_i}}$$

標準化

データの正規化②

- 学習データ $\mathbf{x}_p$
- 平均ベクトル$\boldsymbol{\mu}$, 分散共分散行列Σ

$$P = (\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_N)$$

固有ベクトル

$$D = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_N \end{pmatrix}$$

固有値

$$P^{-1}\Sigma P = D$$

$$\boxed{\mathbf{y} = P\mathbf{x}}$$

無相関化

$$E[\mathbf{y}] = E[P\mathbf{x}] = P\boldsymbol{\mu}$$

$$Var[\mathbf{y}] = E[(\mathbf{y} - E[\mathbf{y}])(\mathbf{y} - E[\mathbf{y}])^t]$$

$$E[(P\mathbf{x} - P\boldsymbol{\mu})(P\mathbf{x} - P\boldsymbol{\mu})^t]$$

$$= P^t E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^t] P$$

$$= P^t \Sigma P = D$$

データの正規化③

- 学習データ $\mathbf{x}_p$
- 平均ベクトル$\boldsymbol{\mu}$, 分散共分散行列Σ

$$P^{-1}\Sigma P = D$$

$$P = (\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_N)$$

$$D^{-\frac{1}{2}} = \begin{pmatrix} \frac{1}{\sqrt{\lambda_1}} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sqrt{\lambda_2}} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \frac{1}{\sqrt{\lambda_N}} \end{pmatrix}$$

$$\boxed{\mathbf{y} = D^{-\frac{1}{2}} P(\mathbf{x} - \boldsymbol{\mu})}$$

白色化

$$E[\mathbf{y}] = D^{-\frac{1}{2}} P(E[\mathbf{x}] - \boldsymbol{\mu}) = \mathbf{0}$$

$$Var[\mathbf{y}] = E[\mathbf{y}\mathbf{y}^t]$$

$$= D^{-\frac{1}{2}} P^t E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^t] P D^{-\frac{1}{2}}$$

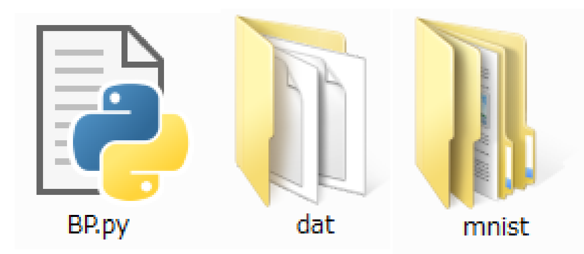
$$= D^{-\frac{1}{2}} P^t \Sigma P D^{-\frac{1}{2}} = D^{-\frac{1}{2}} D D^{-\frac{1}{2}} = I$$

実習（誤差逆伝播則）

誤差逆伝播則(BP.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
- 「dat」というフォルダーを作成して下さい



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

引数をつけて下さい

ニューラルネットワークの構造

- 出力層
  - 損失関数：交差エントロピー
  - 活性化関数：ソフトマックス関数
  - 個数：クラス数（class_num）

- 中間層（1層）
  - 活性化関数：ReLU関数
  - 個数：任意（hunit_num）

- 入力層
  - 個数：特徴数（feature）

変数の定義

```
# クラス数
class_num = 10

# 画像の大きさ
size = 14
feature = size * size

# 学習データ数
train_num = 100

# データ
data_vec = np.zeros((class_num,train_num,feature), dtype=np.float64)

# 学習係数
alpha = 0.1
```

feature
入力層の個数（特徴数）

data_vec
（クラス数，学習（テスト）データ数，特徴数）

活性化関数①

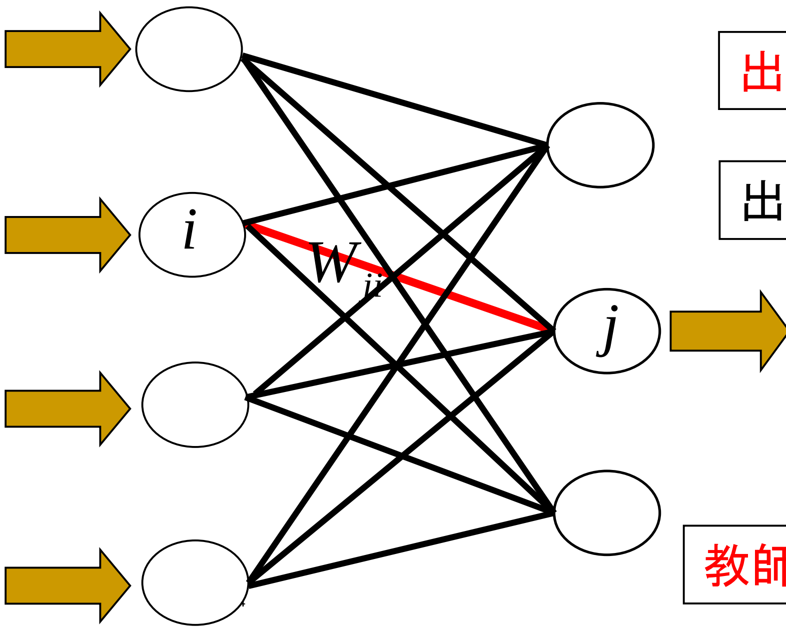
```
# シグモイド関数
def Sigmoid( x ):
    return 1 / ( 1 + np.exp(-x) )

# シグモイド関数の微分
def Sigmoid_( x ):
    return ( 1-Sigmoid(x) ) * Sigmoid(x)

# ReLU関数
def ReLU( x ):
    return np.maximum( 0, x )

# ReLU関数の微分
def ReLU_( x ):
    return np.where( x > 0, 1, 0 )
```

活性化関数②

```
# ソフトマックス関数
def Softmax( x ):
    return np.exp(x)/np.sum(np.exp(x), axis=1, keepdims=True)
```

中間層　n個

入力値　x_{pi}



出力層　m個

出力値

o_{pj}　$\mathbf{o}_p = \left(o_{p1}, o_{p2}, \cdots, o_{pm}\right)$

教師信号

t_{pj}　$\mathbf{t}_p = \left(t_{p1}, t_{p2}, \cdots, t_{pm}\right)$

$$\mathbf{x}_p = \left(x_1, x_2, \cdots, x_n\right)$$

結合係数

$$W = \begin{pmatrix} w_{11} & w_{21} & \cdots & w_{m1} \\ w_{12} & w_{22} & & w_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1n} & w_{2n} & \cdots & w_{mn} \end{pmatrix}$$

$$\boxed{\mathbf{u}_p = \mathbf{x}_p W + \mathbf{b}_p}$$

閾値

$$\mathbf{b} = \left(b_1, b_2, \cdots, b_m\right)$$

$$\boxed{\mathbf{o}_p = f\left(\mathbf{u}_p\right)}$$

出力層のクラス①

```
class Outunit:
    def __init__(self, n, m):
        # 重み
        self.w = np.random.uniform(-0.5,0.5,(n,m))
        # 閾値
        self.b = np.random.uniform(-0.5,0.5,m)

    def Propagation(self, x):
        self.x = x

        # 内部状態
        self.u = np.dot(self.x, self.w) + self.b

        # 出力値（ソフトマックス関数）
        self.out = Softmax( self.u )
```

m：出力層の個数
n: 一つ下の層（中間層）の個数

重み：（n×m）の行列
→ 0.5から0.5の乱数で初期化

閾値：m次元のベクトル
→ 0.5から0.5の乱数で初期化

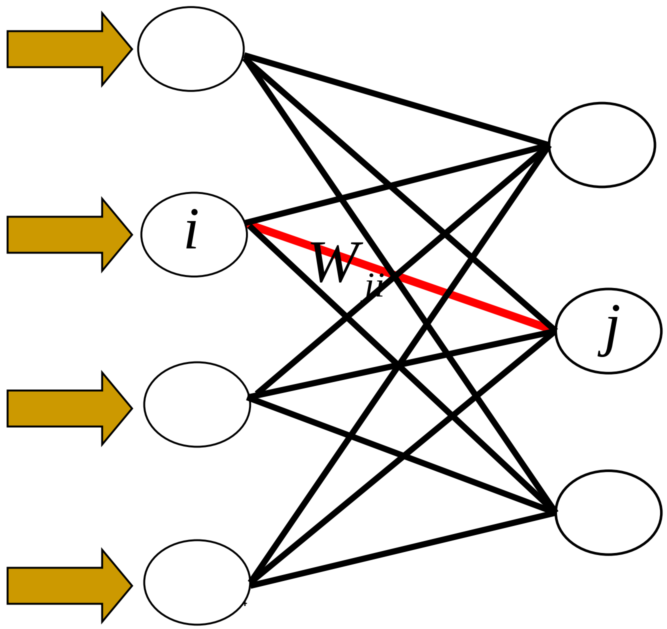
$$\mathbf{u}_p = \mathbf{x}_p W + \mathbf{b}_p$$

$$\mathbf{o}_p = f(\mathbf{u}_p)$$

出力層の重みの更新(行列計算)①

中間層　n個

入力値　x_i



出力層　m個

出力値

O_j　　$\mathbf{o} = (o_1, o_2, \cdots, o_m)$

教師信号

t_j　　$\mathbf{t} = (t_1, t_2, \cdots, t_m)$

$\mathbf{x} = (x_1, x_2, \cdots, x_n)$

誤差　$\delta_j = (O_j - t_j) f'(u_j)$

$$\boxed{\mathbf{\Delta} = (\delta_1, \delta_2, \cdots, \delta_m)}$$

出力層の重みの更新(行列計算)②

$$\partial W = \begin{pmatrix} \partial w_{11} & \partial w_{21} & \cdots & \partial w_{m1} \\ \partial w_{12} & \partial w_{22} & & \partial w_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ \partial w_{1n} & \partial w_{2n} & \cdots & \partial w_{mn} \end{pmatrix} = \begin{pmatrix} \delta_1 x_1 & \delta_2 x_1 & \cdots & \delta_m x_1 \\ \delta_1 x_2 & \delta_2 x_2 & & \delta_m x_2 \\ \vdots & \vdots & \ddots & \vdots \\ \delta_1 x_n & \delta_2 x_n & \cdots & \delta_m x_n \end{pmatrix}$$

$$= \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \begin{pmatrix} \delta_1 & \delta_2 & \cdots & \delta_m \end{pmatrix} = \mathbf{x}^t \mathbf{\Delta}$$

結合係数 $\boxed{W' = W - \alpha \partial W = W - \alpha \mathbf{x}^t \mathbf{\Delta}}$

閾値* $\boxed{\mathbf{b}' = \mathbf{b} - \alpha \partial \mathbf{b} = \mathbf{b} - \alpha 1^t \mathbf{\Delta}}$

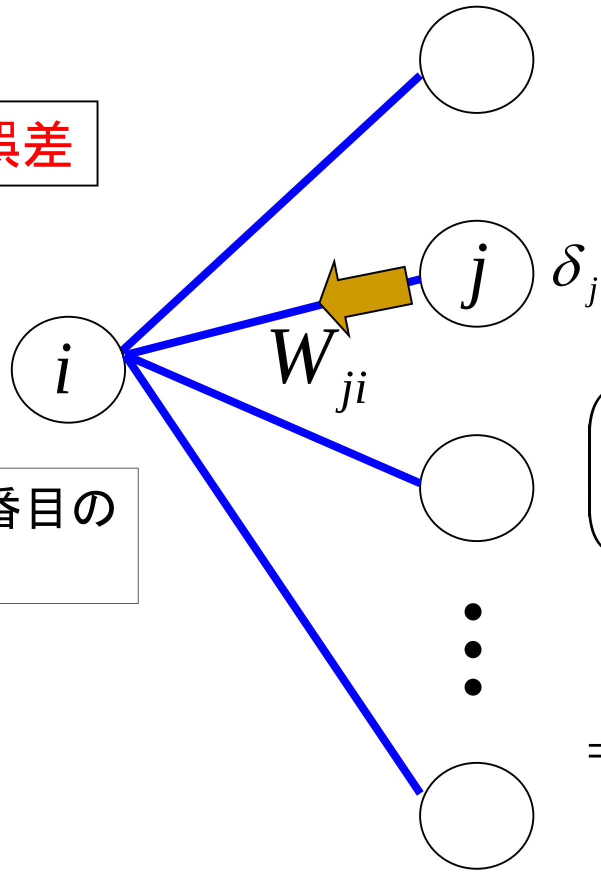
*閾値をn+1番目の重みと考えた場合，入力値は1

誤差の逆伝播（行列計算）

逆伝播する誤差

$$\sum_{j=1}^{m} W_{ji}\delta_j$$

中間層のi番目のニューロン



出力層のj番目のニューロン

$$\left(\sum_{j=1}^{m} W_{j1}\delta_j, \sum_{j=1}^{m} W_{j2}\delta_j, \cdots, \sum_{j=1}^{m} W_{jn}\delta_j\right)$$

$$= (\delta_1, \delta_2, \cdots, \delta_m)\begin{pmatrix} w_{11} & w_{21} & \cdots & w_{n1} \\ w_{12} & w_{22} & & w_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1m} & w_{2m} & \cdots & w_{mn} \end{pmatrix}$$

$$= \mathbf{\Delta} W^t$$

```
def Error(self, t):
    # 誤差
    f_ = 1
    delta = ( self.out - t ) * f_

    # 重み，閾値の修正値
    self.grad_w = np.dot(self.x.T, delta)
    self.grad_b = np.sum(delta, axis=0)

    # 前の層に逆伝播する誤差
    self.error = np.dot(delta, self.w.T)

def Update_weight(self):
    # 重み，閾値の修正
    self.w -= alpha * self.grad_w
    self.b -= alpha * self.grad_b
```

f_：活性化関数の微分

損失関数が交差エントロピー
活性化関数がソフトマックス関数の場合

$$\frac{\partial E}{\partial V_{kj}} = \frac{\partial E}{\partial S_k}\frac{\partial S_k}{\partial V_{kj}} = \boxed{(O_k - t_k)} H_j$$

誤差

$$\boxed{\partial W = \mathbf{x}^t \mathbf{\Delta}}$$

$$\boxed{\partial \mathbf{b} = 1^t \mathbf{\Delta}}$$

$$\boxed{\mathbf{\Delta} W^t}$$

$$\boxed{W' = W - \alpha \partial W}$$

$$\boxed{\mathbf{b}' = \mathbf{b} - \alpha \partial \mathbf{b}}$$

他の活性化関数の場合①

- 出力層
  - 損失関数:誤差二乗和
  - 活性化関数:シグモイド関数

```
def Propagation(self, x):


    # 出力値
    self.out = Sigmoid( self.u )


def Error(self, t):
    # 誤差
    f_ = Sigmoid_( self.u )
    delta = ( self.out - t ) * f_
```

f_ = self.out * (1 - self.out)
でもよい

他の活性化関数の場合②

- 出力層
  - 損失関数:誤差二乗和
  - 活性化関数:恒等関数

```
def Propagation(self, x):

    # 出力値
    self.out = self.u

def Error(self, t):
    # 誤差
    f_ = 1
    delta = ( self.out - t ) * f_
```

出力層のクラス②

```
def Save(self, filename):
    # 重み，閾値の保存
    np.savez(filename, w=self.w, b=self.b)

def Load(self, filename):
    # 重み，閾値のロード
    work = np.load(filename)
    self.w = work['w']
    self.b = work['b']
```

np.savez
numpy形式のデータの保存（バイナリィ）
np.savez（ファイル名，変数名）
→ ファイル名.npzとして保存

重み→キー「w」
閾値→キー「b」

np.load
numpy形式のデータのロード
np.load（ファイル名）

キー「w」→重み
キー「b」→閾値

中間層のクラス①

```
class Hunit:
    def __init__(self, n, m):
        # 重み
        self.w = np.random.uniform(-0.5,0.5,(n,m))
        # 閾値
        self.b = np.random.uniform(-0.5,0.5,m)

    def Propagation(self, x):
        self.x = x

        # 内部状態
        self.u = np.dot(self.x, self.w) + self.b

        # 出力値（ReLU関数）
        self.out = ReLU( self.u )
```

m：中間層の個数
n: 一つ下の層（入力層）の個数

重み：（n×m）の行列
→ 0.5から0.5の乱数で初期化

閾値：m次元のベクトル
→ 0.5から0.5の乱数で初期化

x：入力ベクトル（n次元）

$$\mathbf{u}_p = \mathbf{x}_p W + \mathbf{b}_p$$

$$\mathbf{o}_p = f(\mathbf{u}_p)$$

中間層と入力層の結合係数の修正

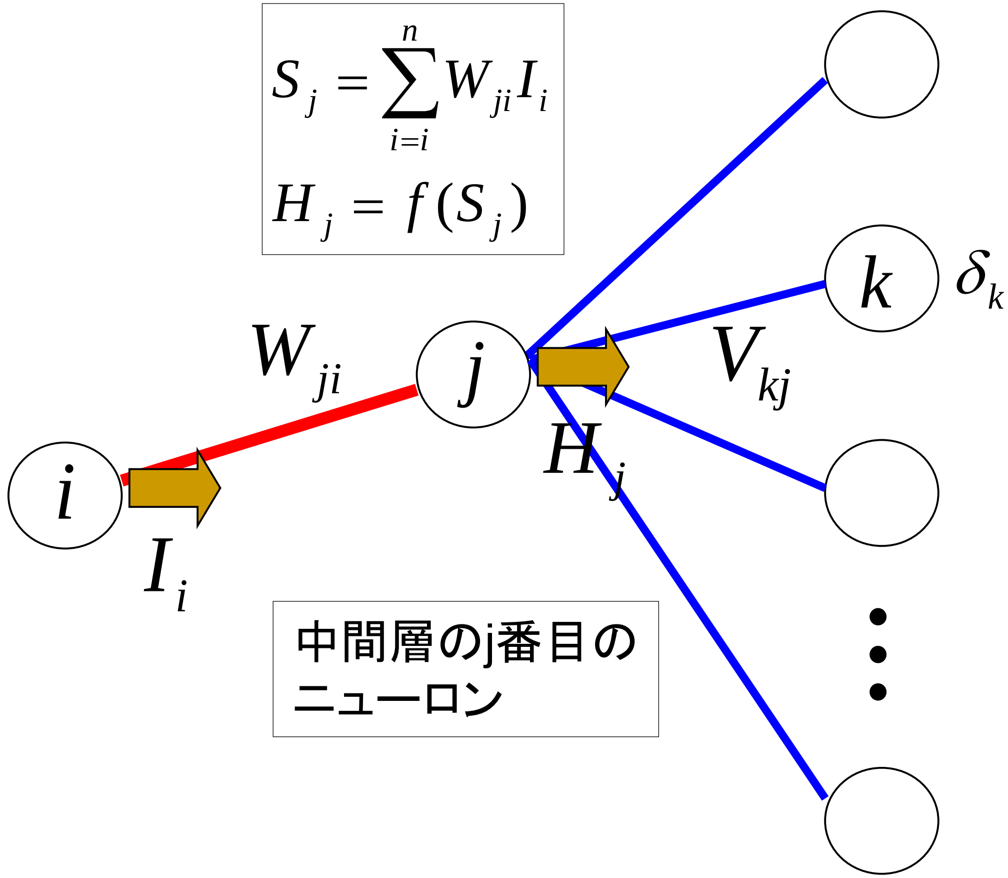
$$W_{ji} \leftarrow W_{ji} - \alpha \frac{\partial E}{\partial W_{ji}}$$

$$\delta_j = \boxed{(\sum_k V_{kj}) f'(S_j)}$$

「誤差」と定義

$$S_j = \sum_{i=i}^{n} W_{ji} I_i$$

$$H_j = f(S_j)$$



中間層のj番目の
ニューロン

$$\frac{\partial E}{\partial W_{ji}} = \delta_j I_i$$

```
def Error(self, p_error):
    # 誤差
    f_ = ReLU_( self.u )
    delta = p_error * f_

    # 重み, 閾値の修正値
    self.grad_w = np.dot(self.x.T, delta)
    self.grad_b = np.sum(delta, axis=0)

    # 前の層に逆伝播する誤差
    self.error = np.dot(delta, self.w.T)

def Update_weight(self):
    # 重み, 閾値の修正
    self.w -= alpha * self.grad_w
    self.b -= alpha * self.grad_b
```

p_error：
一つ上の層（出力層）逆伝播してきた誤差（m次元）

$$\delta_j = \boxed{\left(\sum_k V_{kj}\delta_k\right)} f'(S_j)$$

$$\boxed{\partial W = \mathbf{x}^t \mathbf{\Delta}}$$

$$\boxed{\partial \mathbf{b} = 1^t \mathbf{\Delta}}$$

$$\boxed{\mathbf{\Delta} W^t}$$

$$\boxed{W' = W - \alpha \partial W}$$

$$\boxed{\mathbf{b}' = \mathbf{b} - \alpha \partial \mathbf{b}}$$

中間層のクラス②

```
def Save(self, filename):
    # 重み，閾値の保存
    np.savez(filename, w=self.w, b=self.b)

def Load(self, filename):
    # 重み，閾値のロード
    work = np.load(filename)
    self.w = work['w']
    self.b = work['b']
```

np.savez
numpy形式のデータの保存（バイナリィ）
np.savez（ファイル名，変数名）
→ ファイル名.npzとして保存

重み→キー「w」
閾値→キー「b」

np.load
numpy形式のデータのロード
np.load（ファイル名）

キー「w」→重み
キー「b」→閾値

データの読み込み

flagが0の場合→学習データ(「mnist/train/」)
flagが1の場合→テストデータ(「mnist/test/」)
からデータを読み込む

```
def Read_data( flag ):

    dir = [ "train" , "test" ]
    for i in range(class_num):
        for j in range(1,train_num+1):
            # グレースケール画像で読み込み→大きさの変更→numpyに変換，ベクトル化
            train_file = mnist/" + dir[ flag ] + "/" + str(i) + "/" + str(i) + "_" + str(j) + ".jpg"
            work_img = Image.open(train_file).convert('L')
            resize_img = work_img.resize((size, size))
            data_vec[i][j-1] = np.asarray(resize_img).astype(np.float64).flatten()

            # 入力値の合計を1とする
            data_vec[i][j-1] = data_vec[i][j-1] / np.sum( data_vec[i][j-1] )
```

メインメソッド

```
if __name__ == '__main__':
    # 中間層の個数
    hunit_num = 32

    # 中間層のコンストラクター
    hunit = Hunit( feature , hunit_num )

    # 出力層のコンストラクター
    outunit = Outunit( hunit_num , class_num )

    # 引数
    argvs = sys.argv
```

中間層の個数:hunit_num
一つ前の層(入力層)の個数:feature

コンストラクター(__init__)により，配列を確保
→初期化
hunit.w:feature×hunit_num
hunit.b:hunit_num

出力層の個数:class_num
一つ前の層(中間層)の個数:hunit_num

コンストラクター(__init__)により，配列を確保
→初期化
outunit.w:hunit_num×class_num
outunit.b:class_num

```
# 引数がtの場合
if argvs[1] == "t":

    # 学習データの読み込み
    flag = 0
    Read_data( flag )

    # 学習
    Train()

# 引数がpの場合
elif argvs[1] == "p":

    # テストデータの読み込み
    flag = 1
    Read_data( flag )

    # テストデータの予測
    Predict()
```

flag=0
→学習データの読み込み

flag=1
→テストデータの読み込み

学習①

```
def Train():
    # エポック数
    epoch = 1000
    for e in range( epoch ):
        error = 0.0
        for i in range(class_num):
            for j in range(0,train_num):
                # 入力データ
                rnd_c = np.random.randint(class_num)
                rnd_n = np.random.randint(train_num)
                input_data = data_vec[rnd_c][rnd_n].reshape(1,feature)
```

学習回数：（epoch×class_num×train_num）回

入力データ
（1×feature）に変形

ランダムにクラス（rad_c），データ（rnd_n）を選択し，入力

伝播

```
hunit.Propagation( input_data )
outunit.Propagation( hunit.out )
```

hunit.Propagationメソッド
入力値（input_data）を渡す

outunit.Propagationメソッド
中間層の出力値（hunit.out）を渡す

教師信号

```
teach = np.zeros( (1,class_num) )
teach[0][rnd_c] = 1
```

教師信号
（1×class_num）
→ rnd_c番目の要素は1
それ以外は0

誤差

```
outunit.Error( teach )
hunit.Error( outunit.error )
```

outunit.Errorメソッド
教師信号（teach）を渡す

hunit.Errorメソッド
出力層から逆伝播される誤差
（outunit.error）を渡す

重みの修正

```
outunit.Update_weight()
hunit.Update_weight()
```

outunit.Update_Weight
hunit.Update_Weight
出力層，中間層での重みの更新

学習②

```
    # 誤差二乗和
    error += np.dot( ( outunit.out - teach ) , ( outunit.out - teach ).T )
  print( e , "->" , error )

# 重みの保存
outunit.Save( "dat/BP-out.npz" )
hunit.Save( "dat/BP-hunit.npz" )
```

outunit.Saveメソッド
出力層の保存
保存ファイル名("dat/BP-out.npz")を渡す

hunit.Saveメソッド
出力層の保存
保存ファイル名("dat/BP-out.npz")を渡す

予測

```
def Predict():
    # 重みのロード
    outunit.Load( "dat/BP-out.npz" )
    hunit.Load( "dat/BP-hunit.npz" )

    # 混合行列
    result = np.zeros((class_num,class_num), dtype=np.int32)

    for i in range(class_num):
        for j in range(0,train_num):
            # 入力データ
            input_data = data_vec[i][j].reshape(1,feature)
```

outunit.Loadメソッド
出力層のロード
ロードしたいファイル名
（"dat/BP-out.npz"）を渡す

hunit.Loadメソッド
中間層のロード
ロードしたいファイル名
（"dat/BP-hunit.npz"）を渡す

入力データ
（1×feature）に変形

```
        # 伝播
        hunit.Propagation( input_data )
        outunit.Propagation( hunit.out )

        # 教師信号
        teach = np.zeros( (1,class_num) )
        teach[0][i] = 1

        # 予測
        ans = np.argmax( outunit.out[0] )

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

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

hunit.Propagationメソッド
入力値（input_data）を渡す

outunit.Propagationメソッド
中間層の出力値（hunit.out）を渡す

教師信号
（1×class_num）
→ i番目の要素は1
それ以外は0

outunit.out
（1×class_num）

np.argmax（配列）
配列中，最大値の要素番号を返す

混合行列の表示
正解数の表示

実行(学習)①

> python BP.py t

```
C:¥home¥shino¥prml-2018¥11-26¥program¥program-8>python
0 -> [[896.22492261]]
1 -> [[844.63715633]]
2 -> [[768.87077864]]
3 -> [[699.04521139]]
4 -> [[632.92440643]]
5 -> [[560.96506969]]
6 -> [[521.60840263
7 -> [[470.92258713
8 -> [[443.42355033
9 -> [[415.00834894
10 -> [[418.68048934]]
11 -> [[402.34039729]]
12 -> [[355.5891132]]
13 -> [[382.76935936]]
14 -> [[336.80532076]]
15 -> [[355.40512914]]
16 -> [[340.69961647]]
17 -> [[338.92476113]]
18 -> [[290.33176374]]
19 -> [[260.666932]]
20 -> [[302.14776959]]
21 -> [[301.38086801]]
22 -> [[271.3730143]]
23 -> [[281.25325509]]
24 -> [[280.94993617]]
25 -> [[239.78700504]]
26 -> [[263.35697445]]
27 -> [[255.22081702]]
28 -> [[241.83171621]]
29 -> [[277.39219753]]
```

誤差二乗和が下がっていくことを確認

C:¥Windows¥system32¥cmd.exe

```
970 -> [[0.00368219]]
971 -> [[0.00398791]]
972 -> [[0.00382463]]
973 -> [[0.0043231]]
974 -> [[0.00377467]]
975 -> [[0.0035302]]
976 -> [[0.00329981]]
977 -> [[0.00333118]]
978 -> [[0.00374665]]
979 -> [[0.00393334]]
980 -> [[0.0044436]]
981 -> [[0.00287657]]
982 -> [[0.00398843]]
983 -> [[0.00338561]]
984 -> [[0.00360893]]
985 -> [[0.00415361]]
986 -> [[0.00384015]]
987 -> [[0.00334106]]
988 -> [[0.00328837]]
989 -> [[0.00385494]]
990 -> [[0.00393933]]
991 -> [[0.00330935]]
992 -> [[0.00322384]]
993 -> [[0.00431188]]
994 -> [[0.00381021]]
995 -> [[0.003618]]
996 -> [[0.00296805]]
997 -> [[0.0029336]]
998 -> [[0.0049623]]
999 -> [[0.00406488]]

C:¥home¥shino¥prml-2018¥11-26¥program¥program-8>
```

誤差二乗和が0に近づく

実行(予測)

> python BP.py p

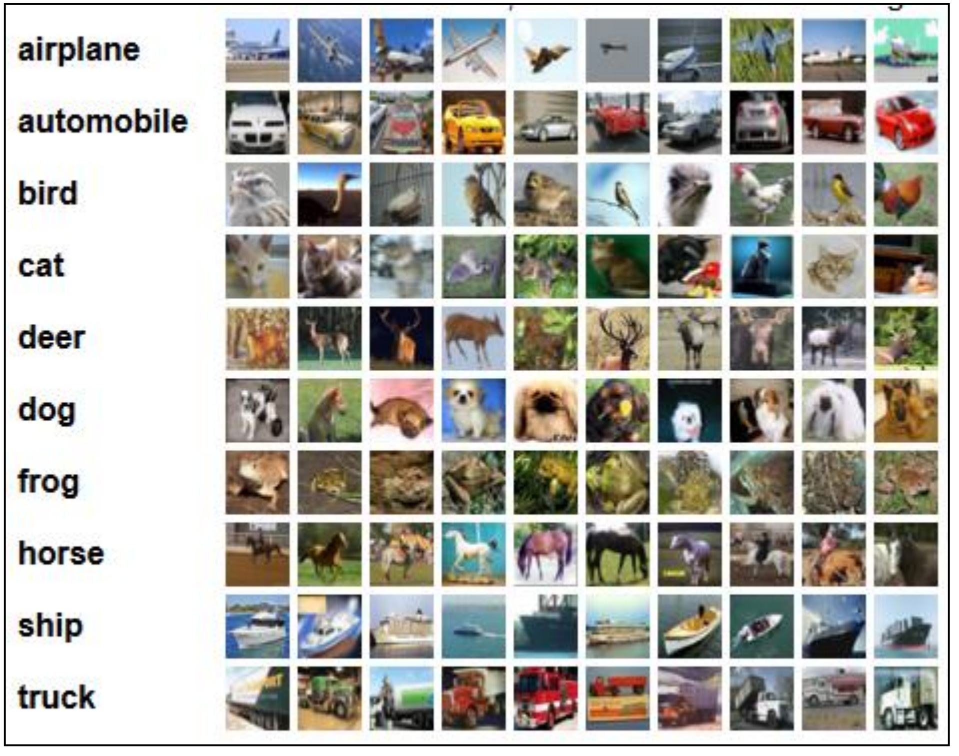
```
9 94 -> 9
9 95 -> 9
9 96 -> 9
9 97 -> 9
9 98 -> 9
9 99 -> 3

 [混合行列]
[[92  0  2  0  0  2  2  0  0  2]
 [ 0 95  1  1  0  1  0  2  0  0]
 [ 0  0 87  0  1  0  1  3  7  1]
 [ 0  1  1 71  0 19  2  1  5  0]
 [ 0  1  0  0 78  1  3  0  0 17]
 [ 4  0  0  3  3 76  1  1  6  6]
 [ 2  0  1  0  2  8 85  0  2  0]
 [ 0  1  6  6  1  0  0 82  1  3]
 [ 1  1  3  3  4  4  2  2 74  6]
 [ 0  1  1  7  1  2  0  3  1 84]]

 正解数 -> 824

C:¥home¥shino¥prml-2018¥11-26¥program¥program-8>
```

混合行列

正解数

CIFAR-10①

- https://www.cs.toronto.edu/~kriz/cifar.html

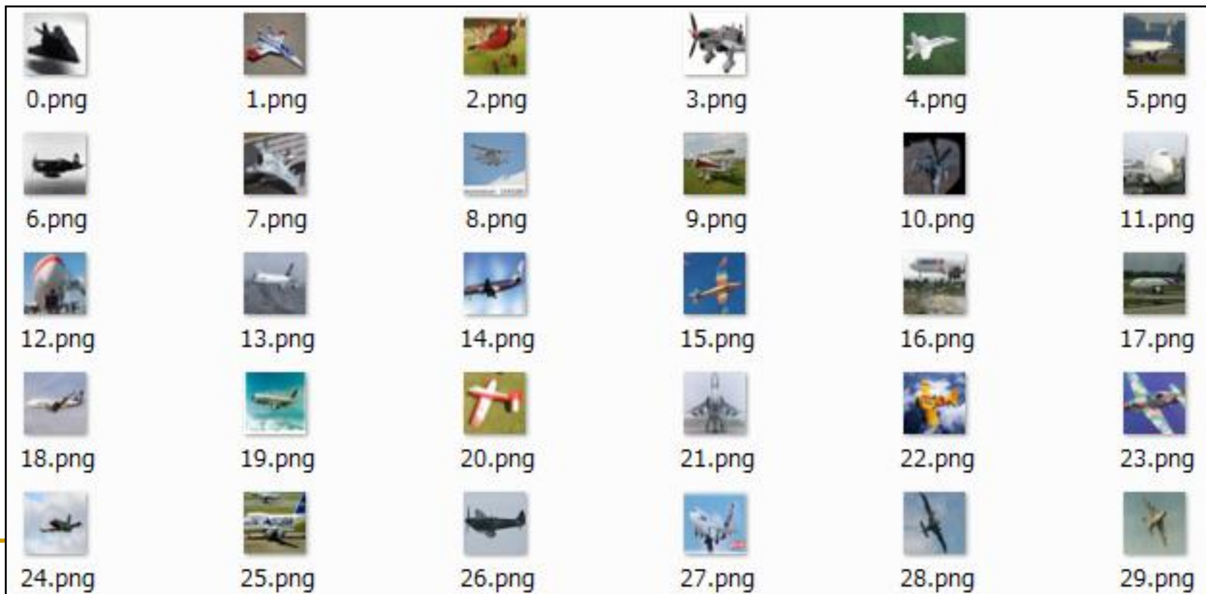


クラス数　10クラス
学習データ　50,000枚
テストデータ 10,000枚
画像の大きさ　32×32

CIFAR-10②

- cifar-10/train/ ・・・ 学習データ
- cifar-10/test/ ・・・ テストデータ
- 各クラスごとに200枚
- 学習データ，テストデータとも2,000枚
- 大きさ 32×32（RGB画像）

二次（三次?）配布はしないで下さい



train/airplane/

宿題⑨

- 3層型ニューラルネットワークを用いて，
  - cifar-10/train/以下の画像(2,000枚)を学習
  - cifar-10/train/以下の画像(2,000枚)を認識

  しなさい．

- 問題点と工夫で述べた改良方法を追加し，（できれば）精度の向上を試みて下さい．

（本日の）参考文献

- J.デイホフ：ニューラルネットワークアーキテクチャ入門，森北出版（1992）
- P.D.Wasserman：ニューラル・コンピューティング，理論と実際，森北出版（1993）
- M.L.ミンスキー，S.A.パパート：パーセプトロン，パーソナルメディア（1993）
- 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（2013）

（本日の）参考文献

- 岡谷貴之：深層学習，講談社（2015）
- 瀧雅人：これならわかる深層学習入門，講談社（2017）
- 原田達也：画像認識，講談社（2017）