機械学習
ニューラルネットワーク(1)

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(1)
  - ニューロンのモデル化
  - フィードフォワード型ネットワーク
  - 誤差逆伝播則(バックプロパゲーション)
- 実習
  - XOR問題*
    - パーセプトロンの場合
    - 三層型の場合
  - 文字認識(Digitsデータベース)

*COM実験の資料も参考にして下さい

ニューラルネットワーク

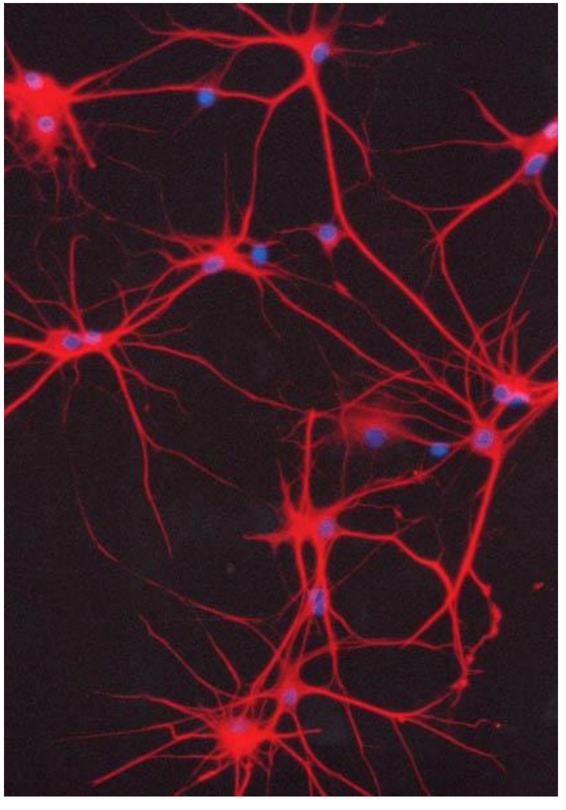
ニューロンのモデル化

フィードフォワード型ネットワーク

パーセプトロン

神経細胞

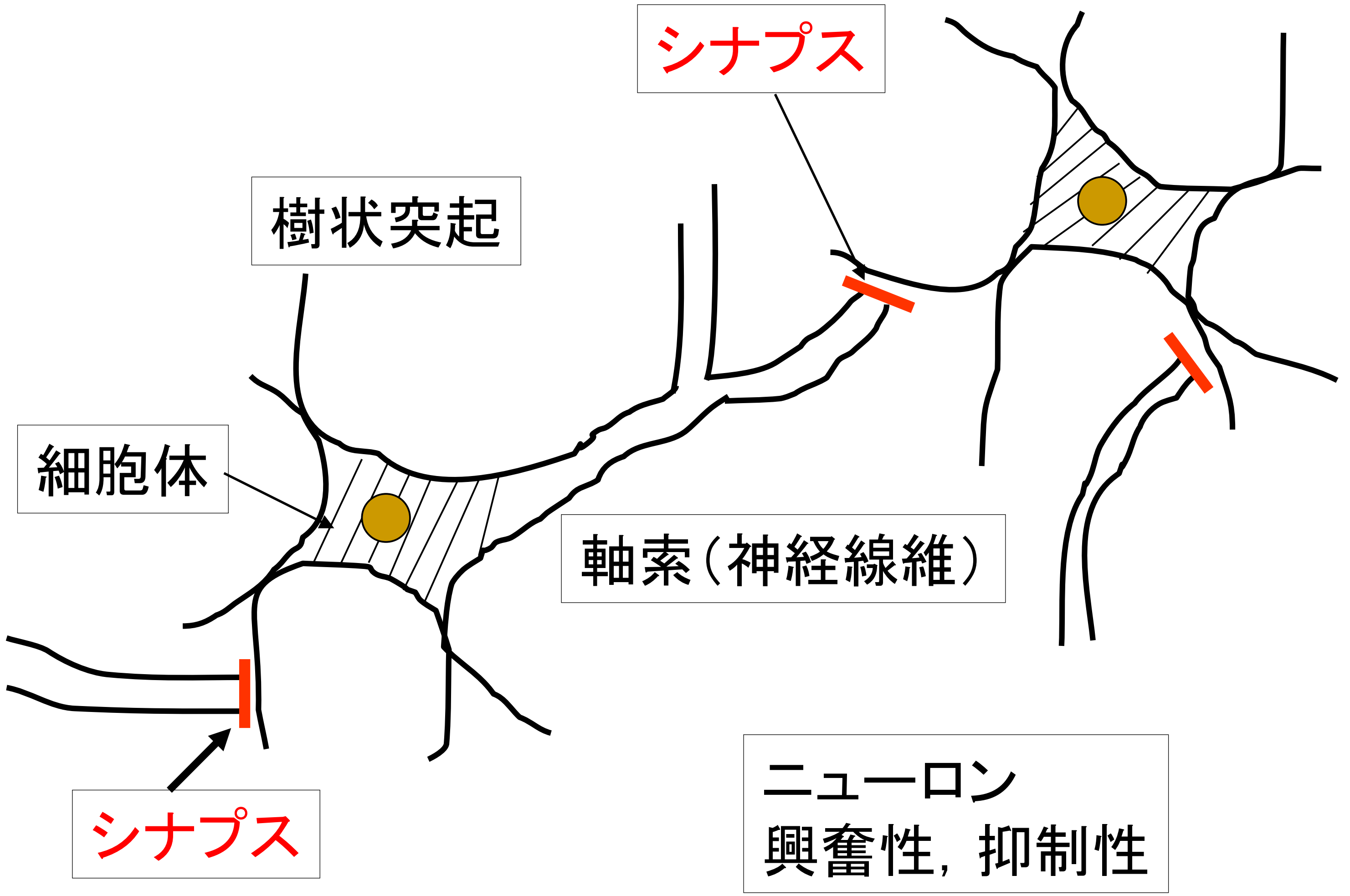
- 神経細胞（ニューロン）



細胞体の大きさ
10マイクロメートル程度

http://compe.designtope.net/nextmaruni2004/biishiki_image/02-02.jpg

神経細胞（イメージ図）

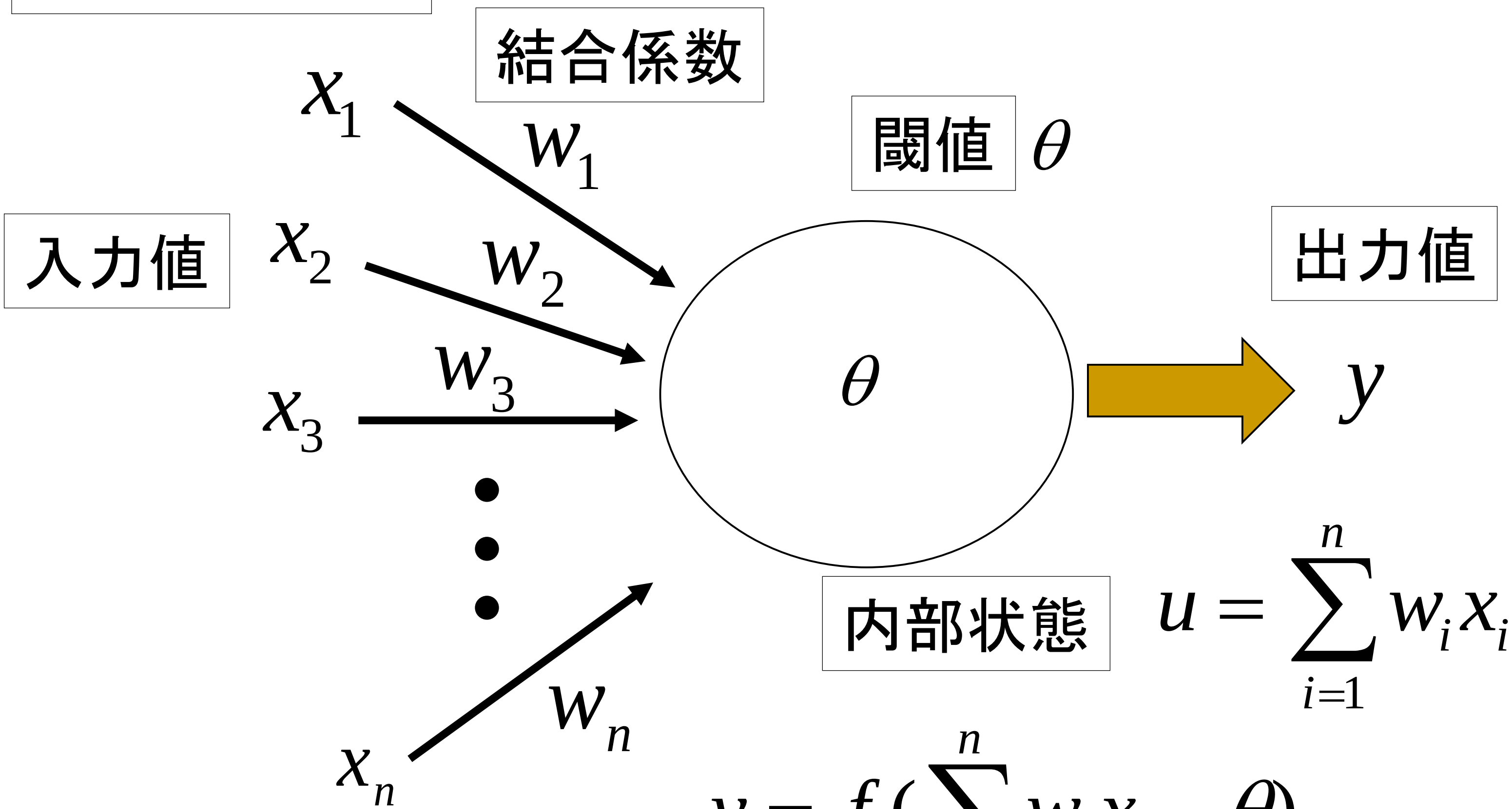


ニューロンのモデル化（概念）

- 神経線維を通して（電気）信号を送る（一出力）
- 樹状突起部のシナプスを介して信号が伝わる
- 複数のシナプスからの信号により内部状態が定まる（多入力）
- 内部状態がしきい値を超えると信号を送る

ニューロンのモデル化①

多入力一出力



内部状態 $u = \sum_{i=1}^{n} w_i x_i - \theta$

$$y = f(\sum_{i=1}^{n} w_i x_i - \theta)$$

f　活性化関数(activation function)

ニューロンのモデル化②

出力値の計算

ニューロンの内部状態ともいう

$$y = f(\boxed{\sum_{i=1}^{n} w_i x_i - \theta})$$

⇕

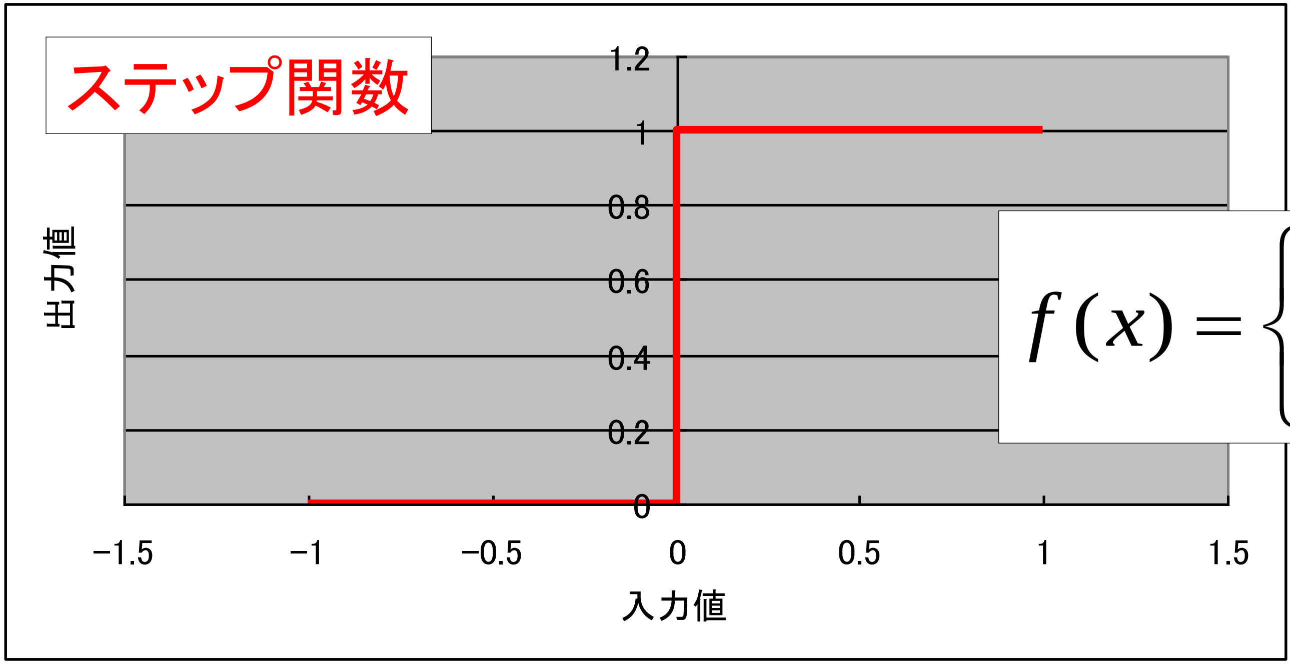
$x_0 = -1, w_0 = \theta$　とすると

$$y = f(\sum_{i=0}^{n} w_i x_i)$$

活性化関数①

- 離散型の場合

マカロック･ピッツモデル（McCulloch-Pitts, 1943）



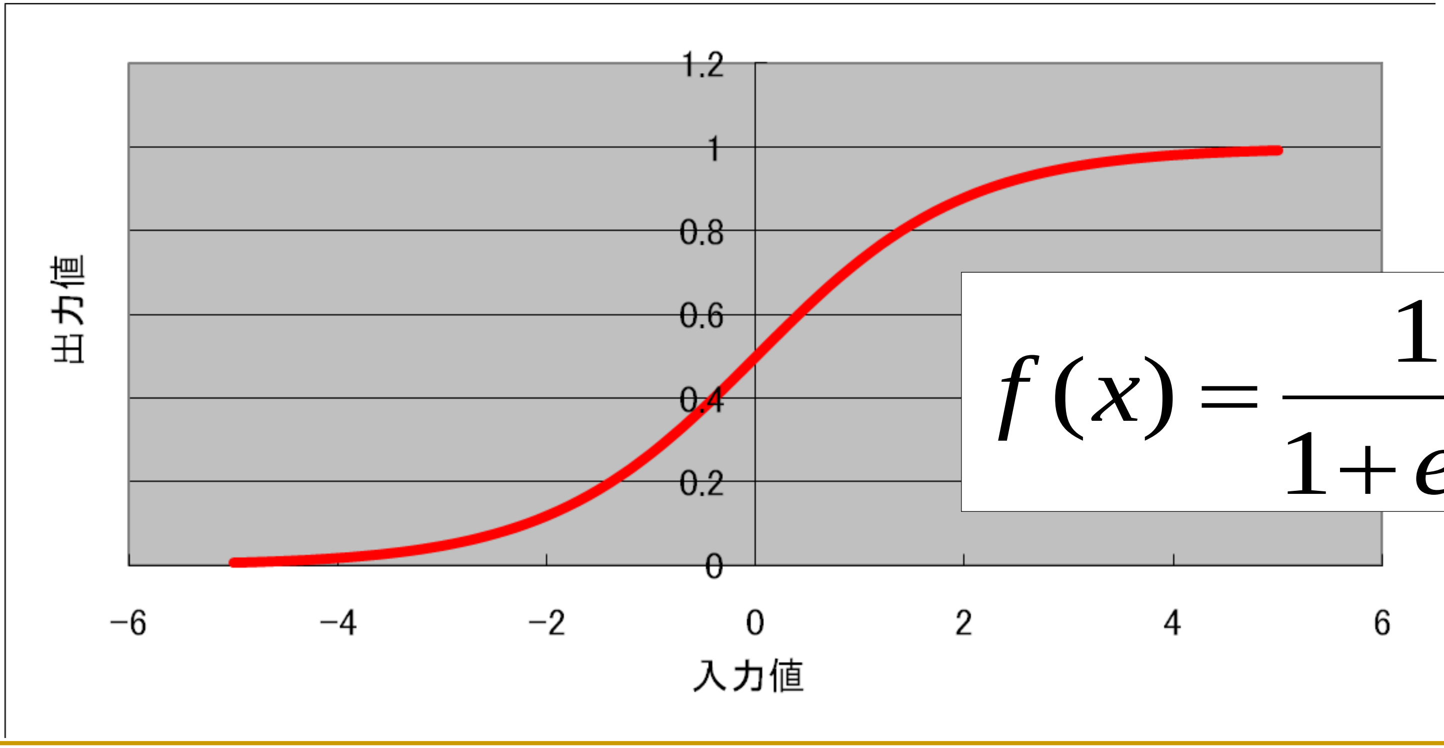
$$f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

しきい値を超えた場合出力値は1（発火）, 超えなければ出力値は0

活性化関数②

- 連続型の場合

シグモイド関数



$$f(x) = \frac{1}{1+e^{-x}}$$

シグモイド関数

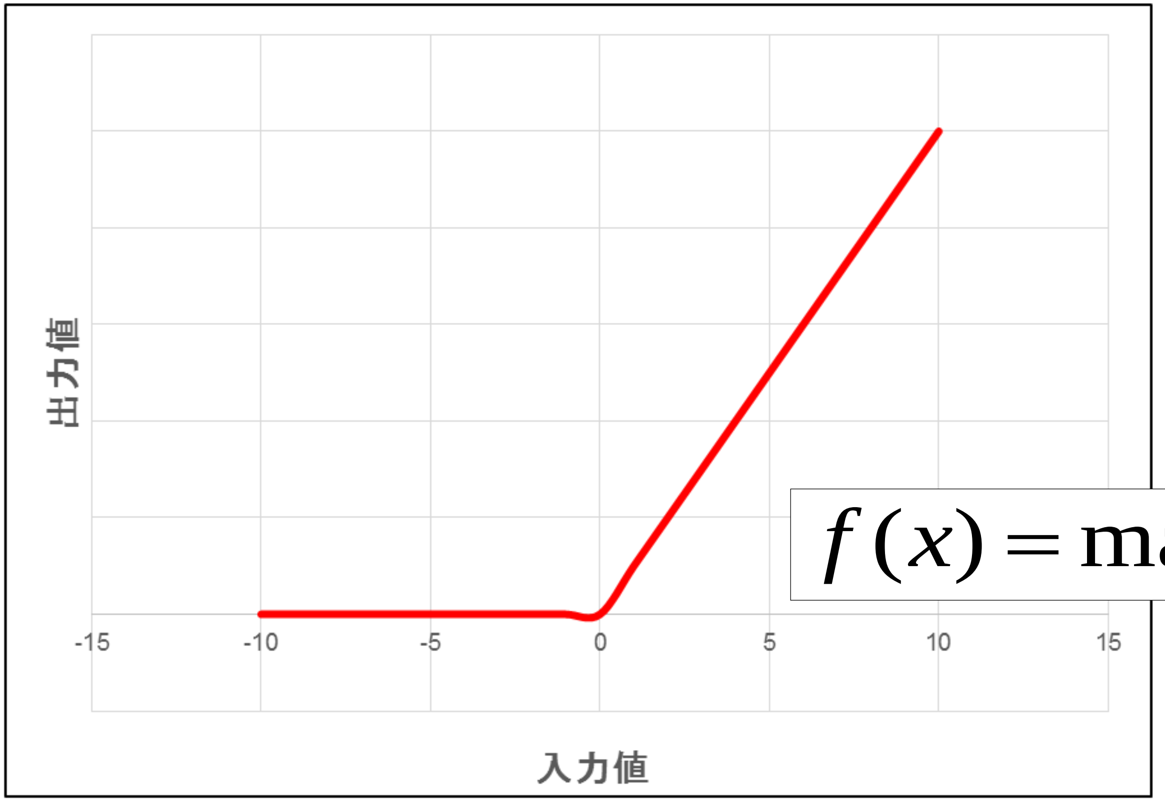
- シグモイド関数の特徴

$$f(x) = \frac{1}{1+e^{-x}}$$

$$f^{'}(x) = \frac{e^{-x}}{(1+e^{-x})^2} = f(x)(1-f(x))$$

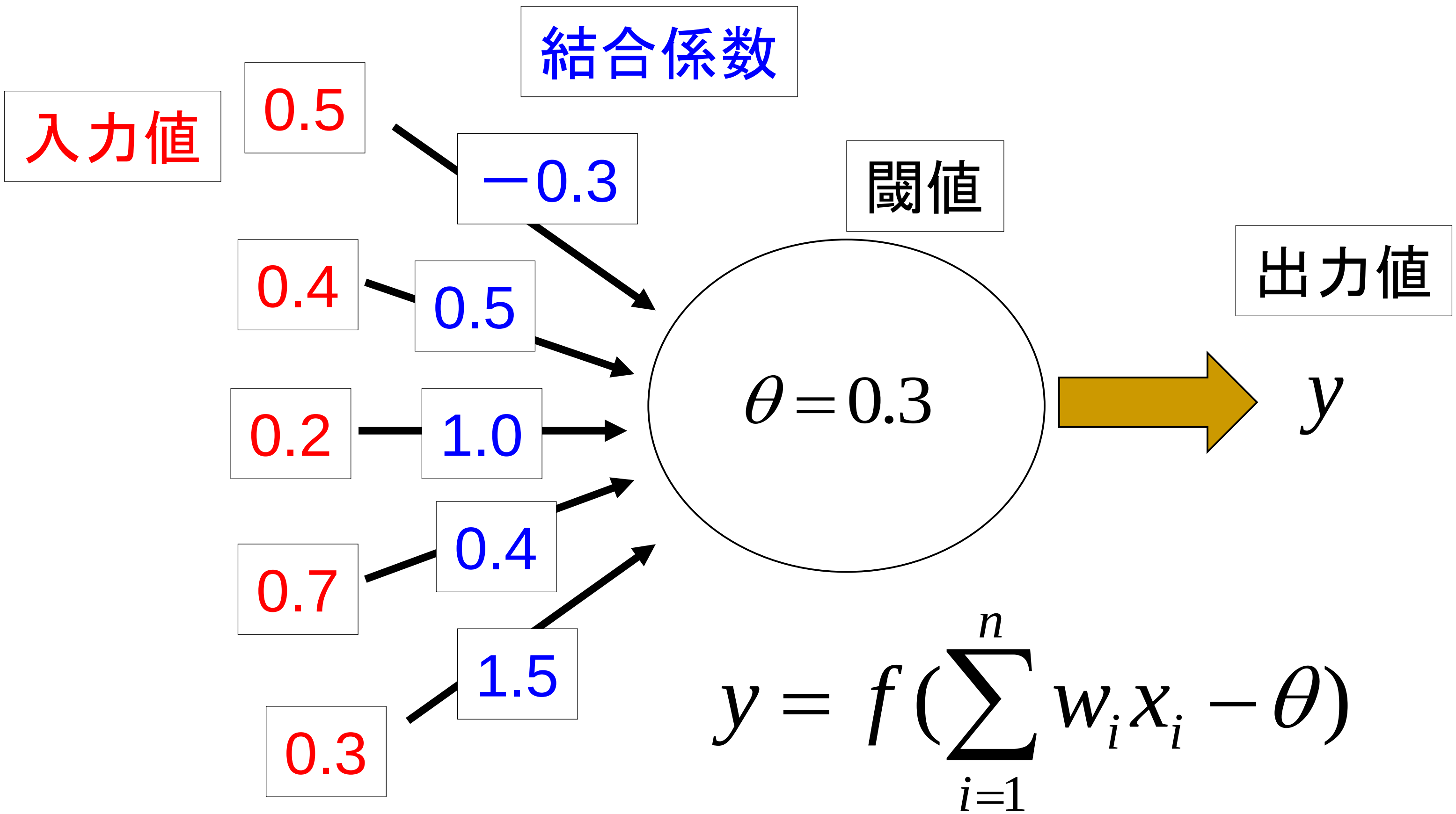
活性化関数③

- 正規化線形関数（Rectified Linear Unit）



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

出力値の計算①



$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

出力値の計算②

- 内部状態

0.5×－0.3 + 0.4×0.5 + 0.2 ×1.0 + 0.7×0.4 +0.3×1.5 - 0.3 = 0.68

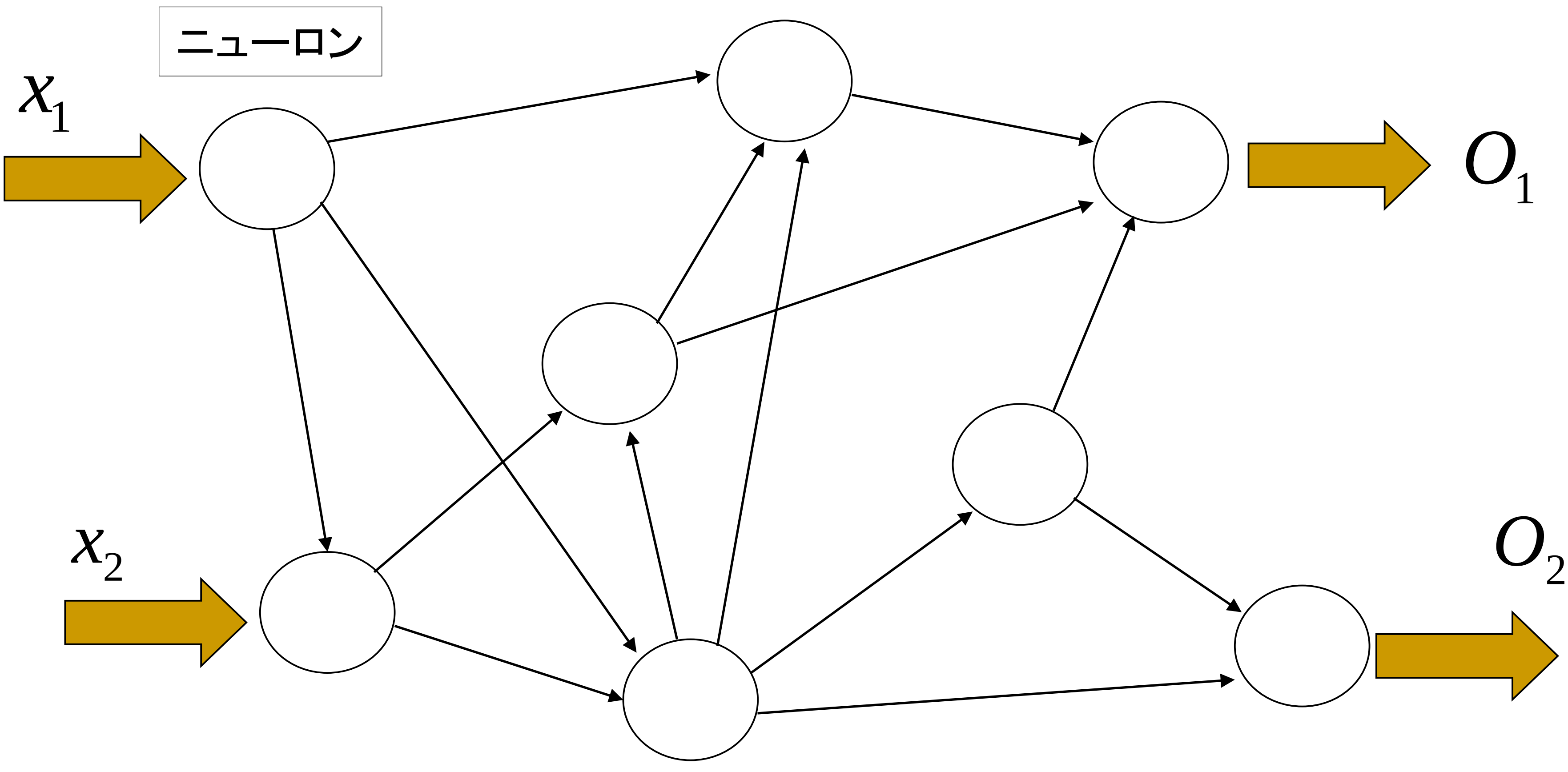
- ステップ関数の場合
  - 内部状態は正 → 出力値は1

- シグモイド関数の場合

1 / (1 + exp(-0.68)) =0.663

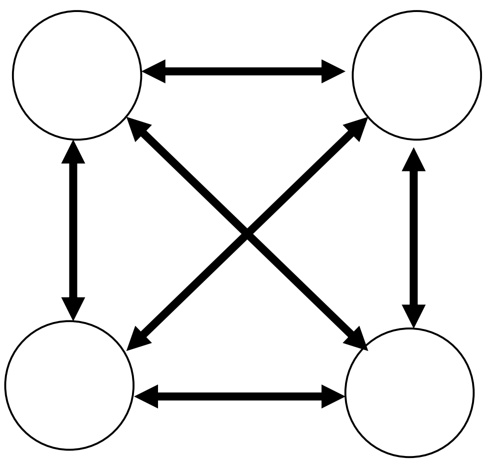
ニューラルネットワーク

ニューロンを互いに結合（ネットワーク化）

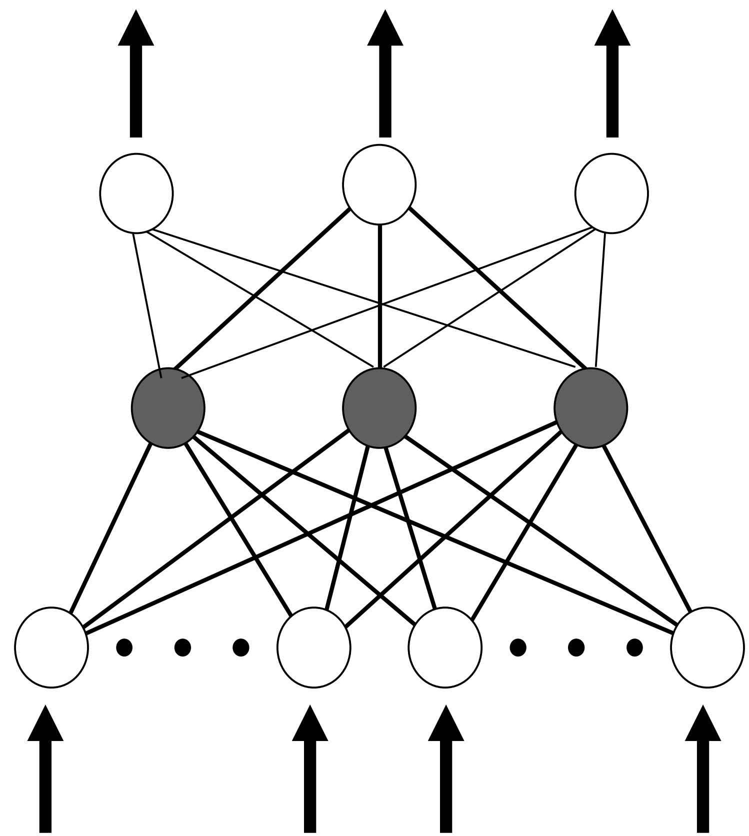


ネットワークの形態

- 相互結合型

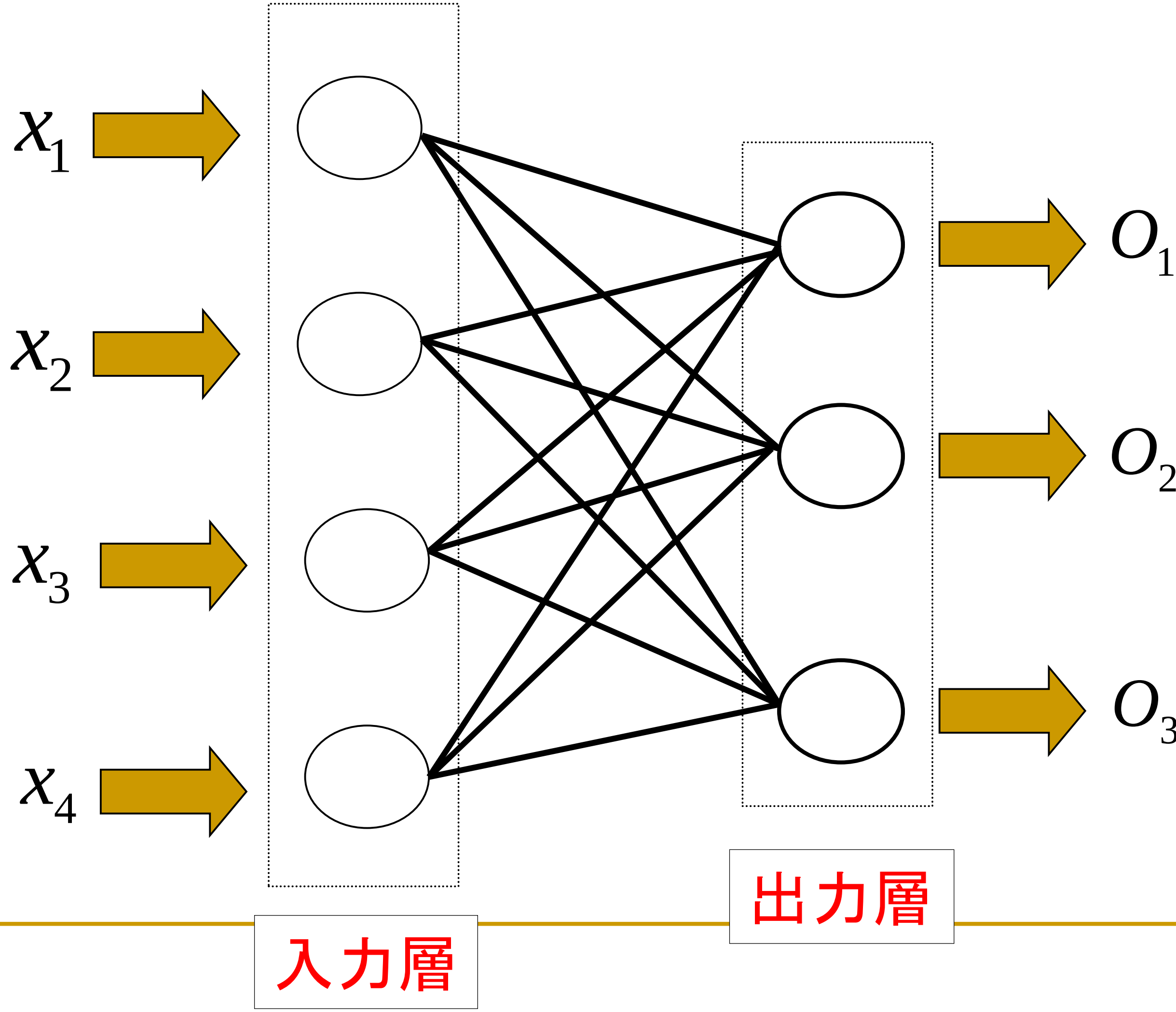


- 階層（フィードフォワード）型



階層型ネットワーク①

二層の階層型（フィードフォワード）の構造



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
- 出力層のニューロン
  - 入力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

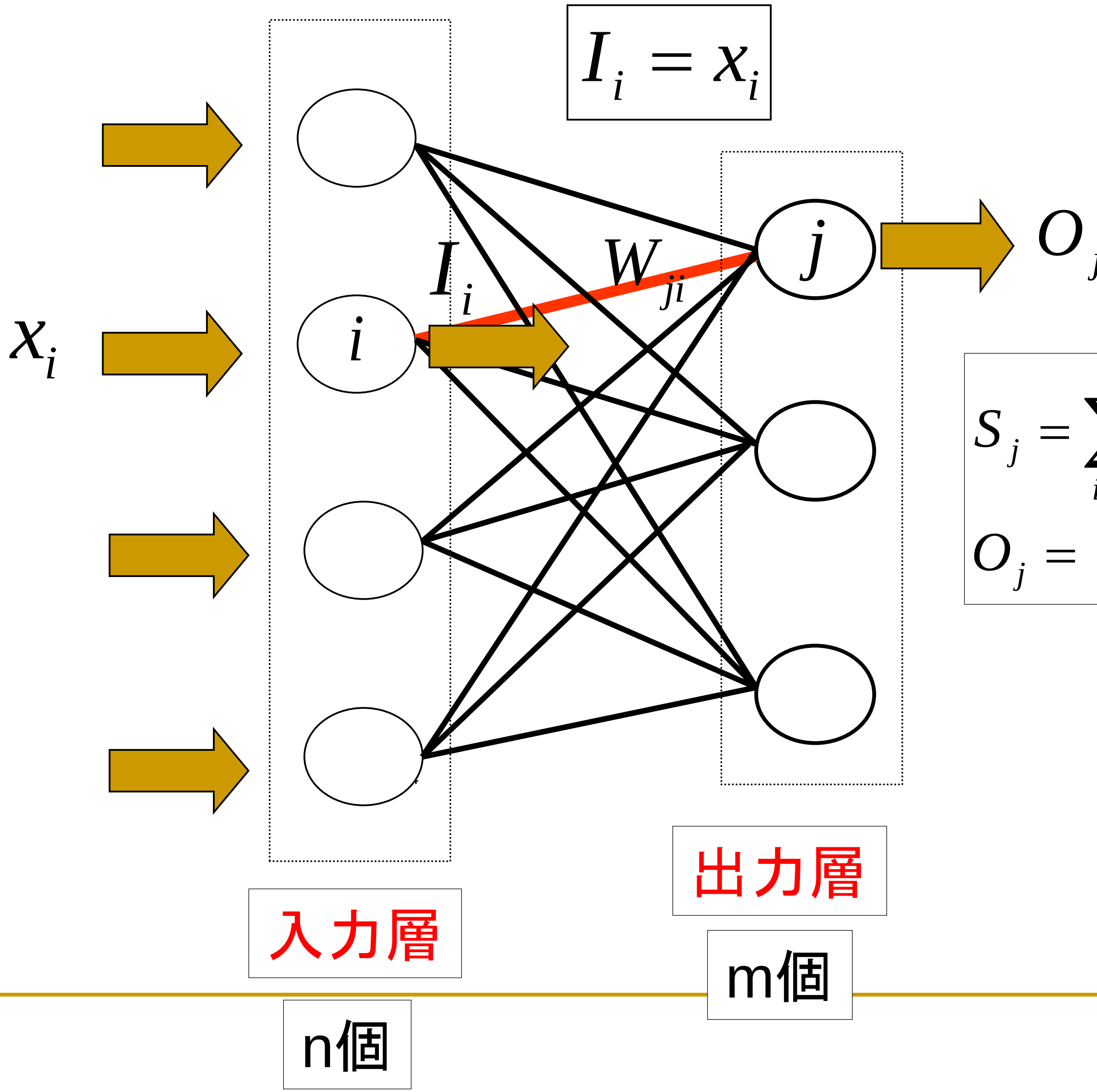
ネットワークの表記①

- i 番目の入力層の値（入力信号x_iと同じ値） I_i
- j 番目の出力層の出力値　O_j　内部状態　S_j

- 入力層のニューロン数　n
- 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}

ネットワークの表記②

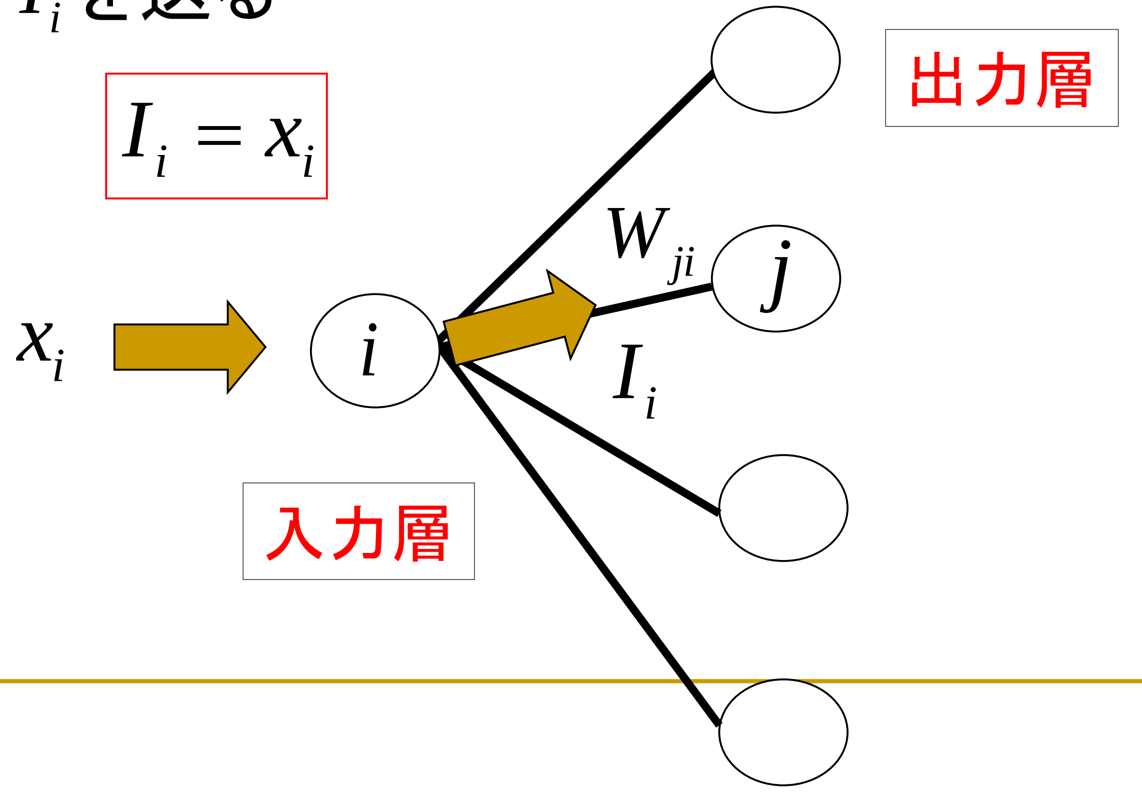


$$S_j = \sum_{i=i}^{n} W_{ji} I_i - \theta_j$$

$$O_j = f(S_j)$$

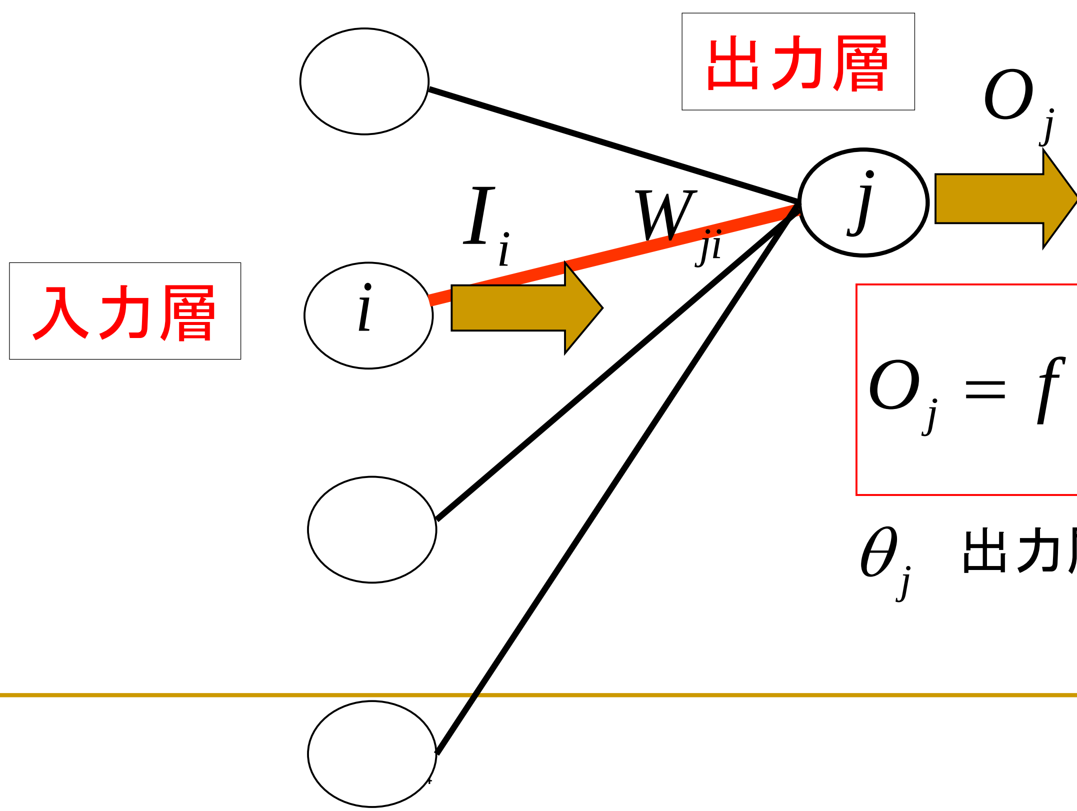
ネットワークの動作①

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



ネットワークの動作②

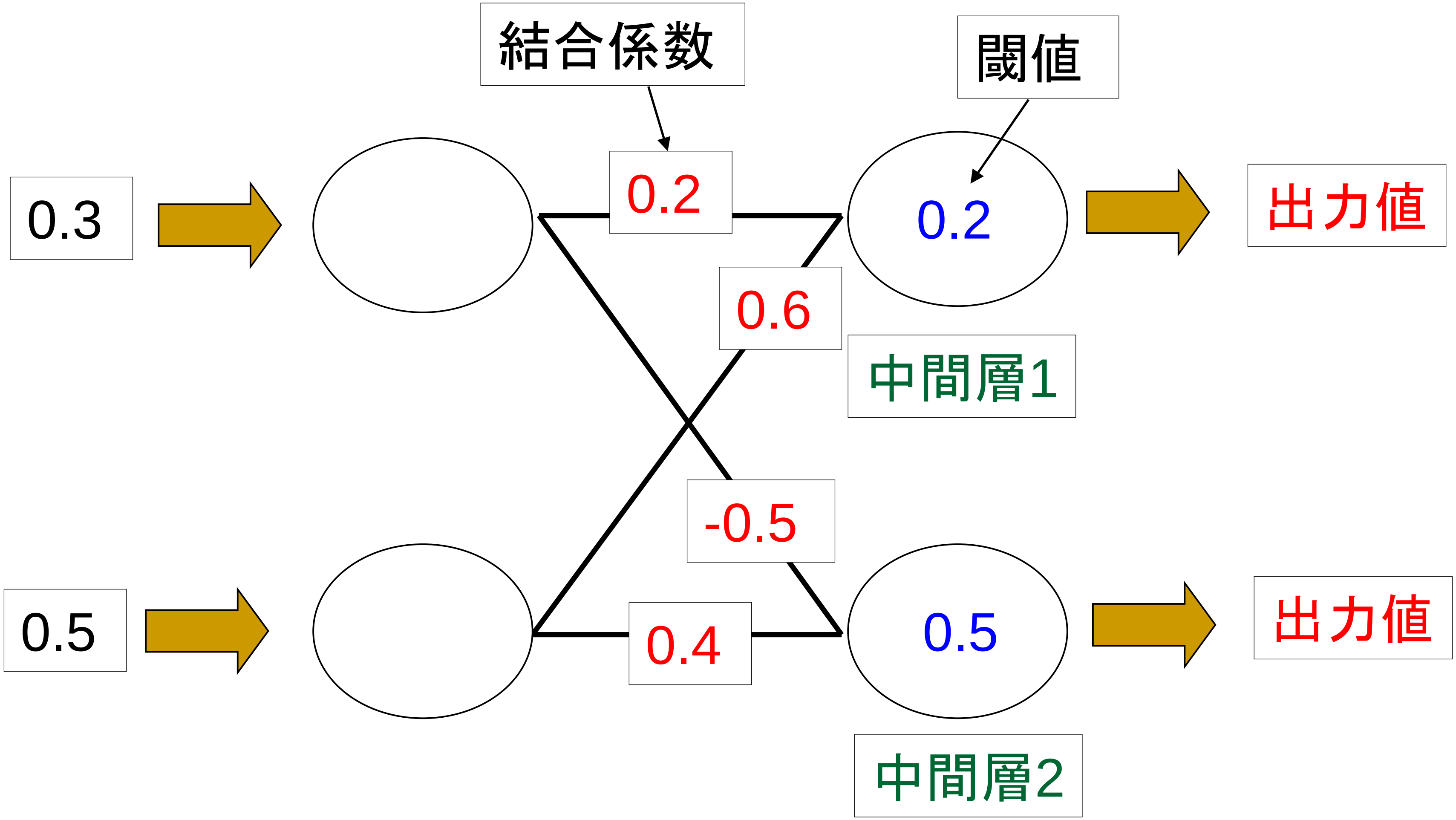
- 出力層のｊ番目のニューロン
  - 全ての入力層のニューロンから値を受け取り，出力値を計算



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

θ_j　出力層のニューロンｊの閾値

ネットワークの動作例①



ネットワークの動作例②

- 活性化関数をステップ関数とした場合
- 出力層1
  - 内部状態
  - 0.3 × 0.2 + 0.5 ×0.6 – 0.2 = 0.16
  - 内部状態は正 → 出力値は1

- 出力層2
  - 内部状態
  - 0.3 × -0.5 + 0.5 × 0.4 ー 0.5 = -0.45
  - 内部状態は負 → 出力値は0

ネットワークの動作例②

- 活性化関数をシグモイド関数とした場合
- 出力層1
  - 内部状態
  - 0.3 × 0.2 + 0.5 ×0.6 – 0.2 = 0.16
  - 1 / (1 + exp(-0.16)) = 0.54
- 出力層2
  - 内部状態
  - 0.3 × -0.5 + 0.5 × 0.4 ― 0.5 = -0.45
  - 1 / (1 + exp(0.45)) = 0.39

パーセプトロン

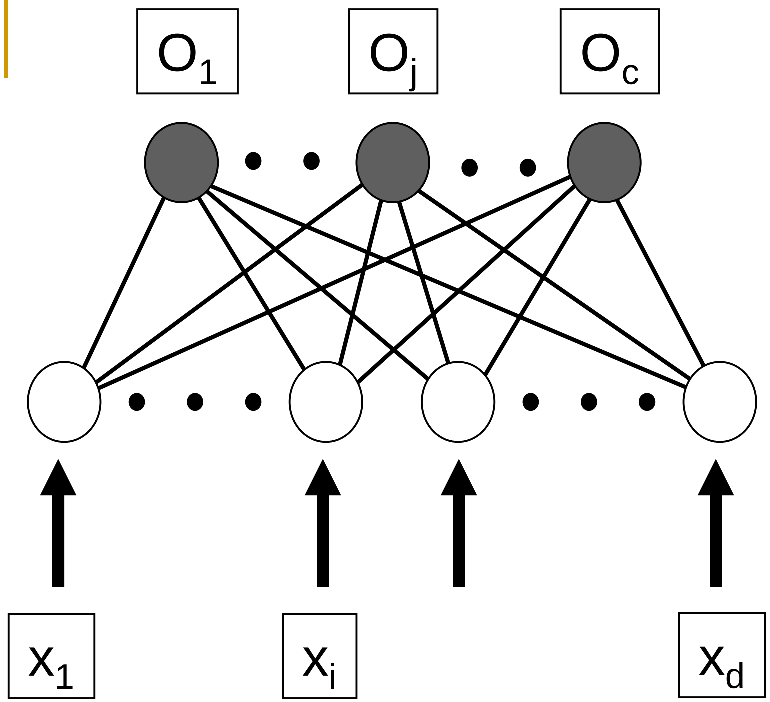
- 二層（入力層，出力層）のフィードフォワード型ネットワーク
- 活性化関数をステップ関数とした場合
- パーセプトロンと等価
- 二層のフィードフォワード型ネットワークをパーセプトロンと呼ぶ

パーセプトロンで解決できる問題

- クラスω_j（$j=1,2,\cdots,c$）
- 特徴ベクトル**x**（d次元）
- 各クラスに対応した（線形）識別関数 g_j を構築

↓

- 入力層のニューロン数 → d個
  - 特徴との対応づけ
- 出力層のニューロン数 → c個
  - 各クラスとの対応づけ



出力層のj番目のニューロン
→ クラスω_jと対応づけ

入力した特徴xがクラスω_jに属する場合
→出力層のj番目のニューロンの出力値は1

入力した特徴xがクラスω_jに属さない場合
→出力層のj番目のニューロンの出力値は0

$$\mathbf{x} = (x_1, x_2, \cdots, x_d)^t$$

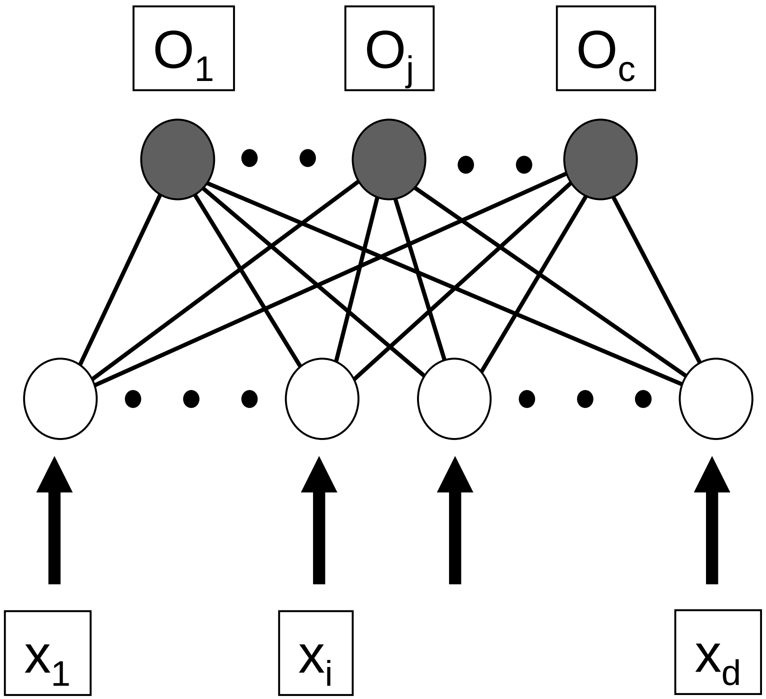
$$\mathbf{x} \in \omega_j$$

各結合係数を調整（学習）
方法は「パーセプトロンの学習規則」

入力層のi番目のニューロン
→ 特徴ベクトルのi番目の要素を入力

1番目の出力のみ1，他は0

$$o = (1,0,\cdots,0)^t$$

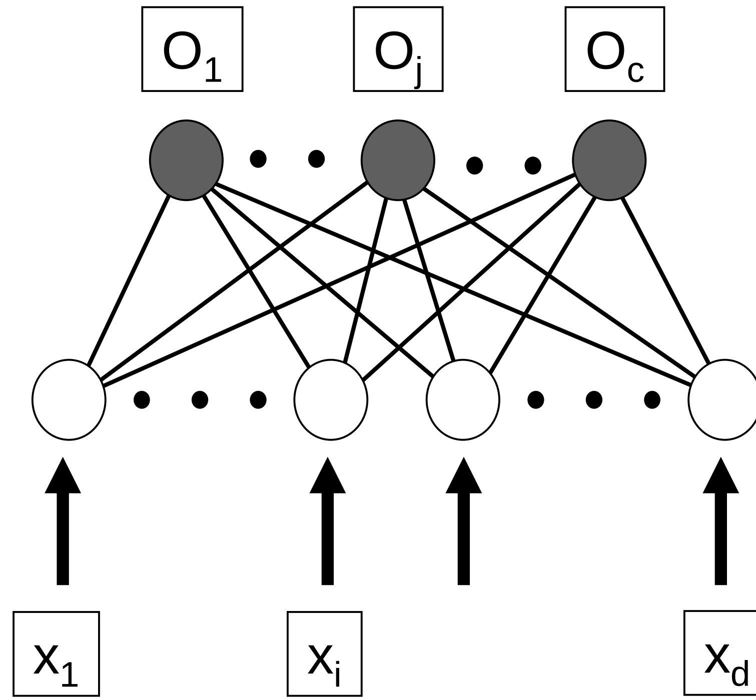


$$\mathbf{x} = (x_1, x_2, \cdots, x_d)^t$$

$$\mathbf{x} \in \omega_1$$

j番目の出力のみ1，他は0

$$o = (0,0,\cdots 1,\cdots,0)^t$$

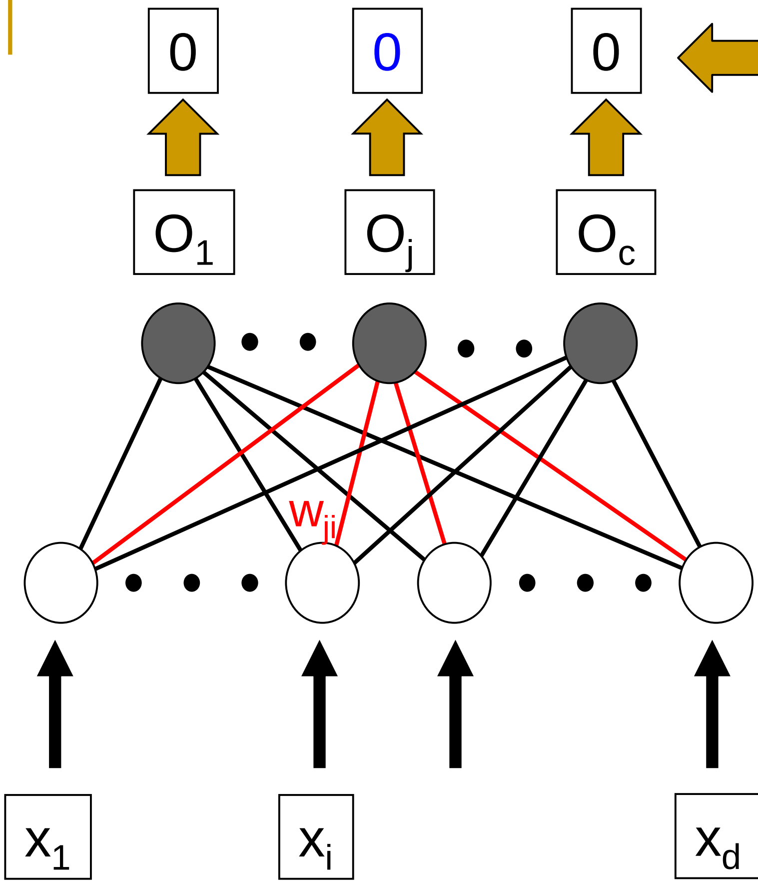


$$\mathbf{x} = (x_1, x_2, \cdots, x_d)^t$$

$$\mathbf{x} \in \omega_j$$

パーセプトロンの学習規則

1. 結合係数$\mathbf{w}_j$を乱数にて初期化
   - クラス数はc個（j=1,2,・・・,c）
2. 学習パターン$\mathbf{x}$ を選択
3. 全ての出力値を計算，正しく出力できなかったニューロンの結合係数$\mathbf{w}_j$を修正
   - $\mathbf{x}_p$をω_j と認識しなければならないのに，ω_jではないと認識してしまった場合→ $\mathbf{w}_j' = \mathbf{w}_j + \rho\mathbf{x}$
   - $\mathbf{x}_p$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→ $\mathbf{w}_j' = \mathbf{w}_j - \rho\mathbf{x}$
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



j番目の出力のみ1，他は0を出力しなければならない

$\mathbf{w}_j$を更新

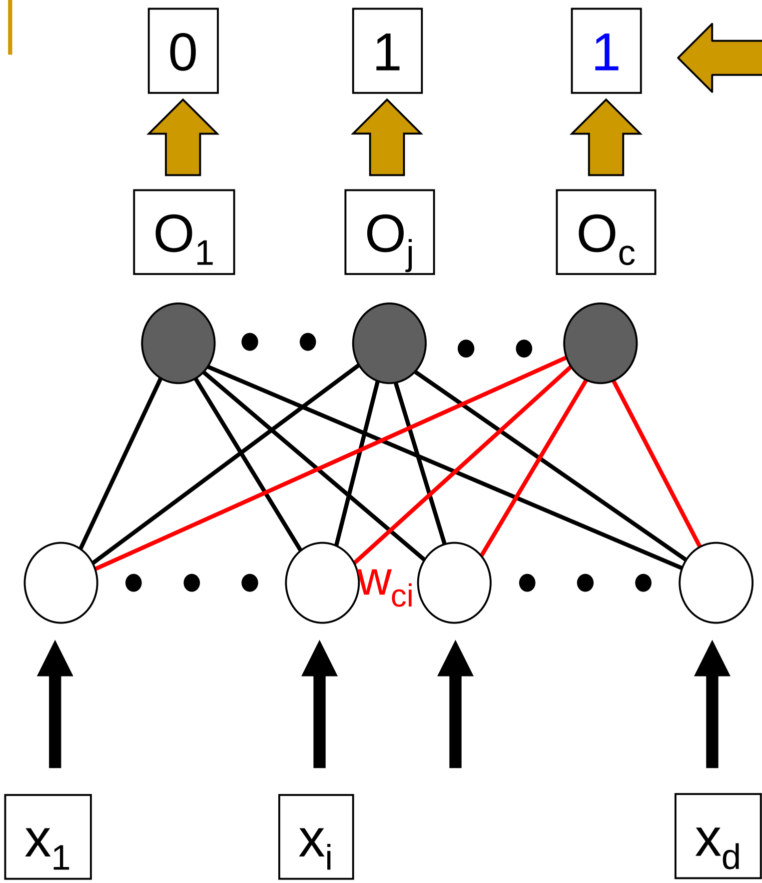
$$\mathbf{w}'_j = \mathbf{w}_j + \rho\mathbf{x}$$

$$w'_{ji} = w_{ji} + \rho x_i$$

$$(i = 1, 2, \cdots, d)$$

$$\mathbf{x} = (x_1, x_2, \cdots, x_d)^t$$

$$\mathbf{x} \in \omega_j$$



j番目の出力のみ1，他は0を出力しなければならない

$\mathbf{w}_c$を更新

$$\mathbf{w}'_c = \mathbf{w}_c - \rho\mathbf{x}$$

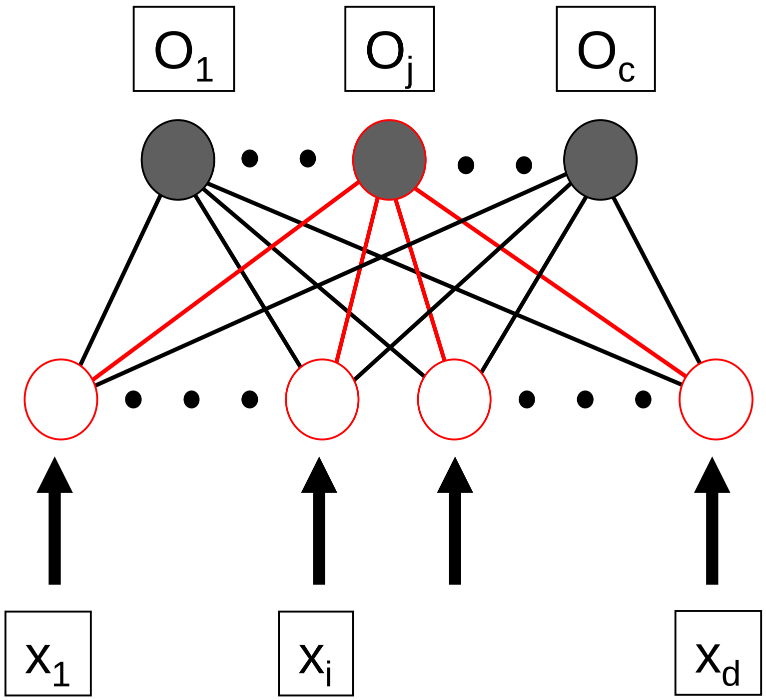
$$w'_{ci} = w_{ci} - \rho x_i$$

$$(i = 1, 2, \cdots, d)$$

$$\mathbf{x} = (x_1, x_2, \cdots, x_d)^t$$

$$\mathbf{x} \in \omega_j$$

学習後のパーセプトロン



結合係数w_jは線形判別関数g_jの重みベクトルと等価

↓

線形分離可能な問題のみに対応

ロジスティック回帰

- 二層（入力層，出力層）のフィードフォワード型ネットワーク
- 出力層が1個の場合→活性化関数をシグモイド関数
- 出力層が2個以上の場合→活性化関数はソフトマックス関数

↓

- ロジスティック回帰と等価
- デルタルールを用いて結合係数を学習

ロジスティック回帰（出力層が1個の場合）

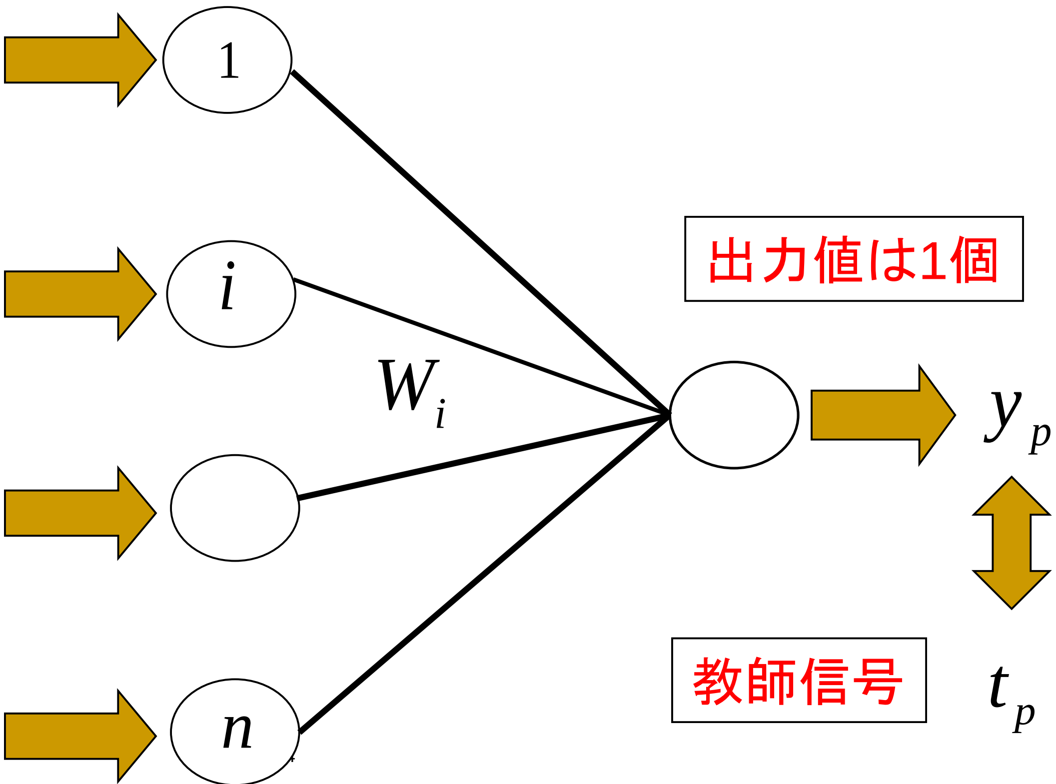
活性化関数はシグモイド関数

$$u_p = \sum_i W_i x_{pi}$$

$$y_p = \frac{1}{1+\exp(-u_p)}$$

入力値

x_{pi}



出力値は1個

$y_p > 0.5 \rightarrow$ クラス1

$y_p < 0.5 \rightarrow$ クラス2

教師信号

$$t_p \in \{0,1\}$$

$$E_p = -(t_p \log y_p + (1-t_p)\log(1-y_p))$$

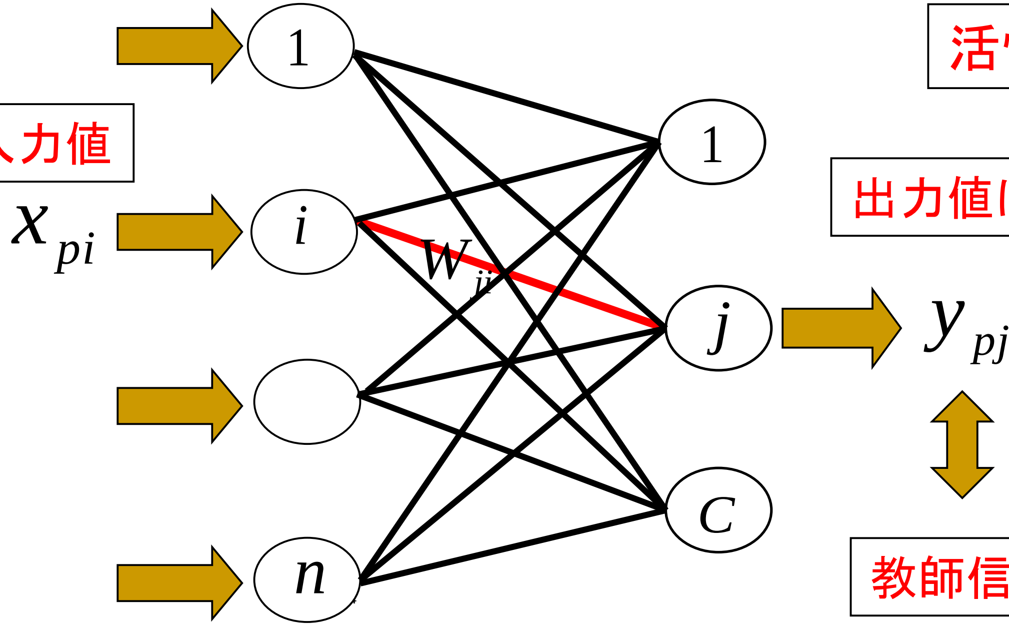
誤差関数はクロスエントロピー

結合係数
デルタルールで学習

$$W_i' = W_i - \alpha(y_p - t_p)x_{pi}$$

ロジスティック回帰(出力層が2個以上の場合)

入力値



活性化関数はソフトマックス関数

$$u_{pj} = \sum_i W_{ji} x_{pi}$$

$$y_{pj} = \frac{\exp(u_{pj})}{\sum_{k=1}^{C} \exp(u_{pk})}$$

出力値はC個

教師信号

$$t_{pj} \in \{0,1\}$$

$$E_p = -\sum_{k=1}^{C} t_{pk} \log y_{pk}$$

誤差関数はクロスエントロピー

結合係数
デルタルールで学習

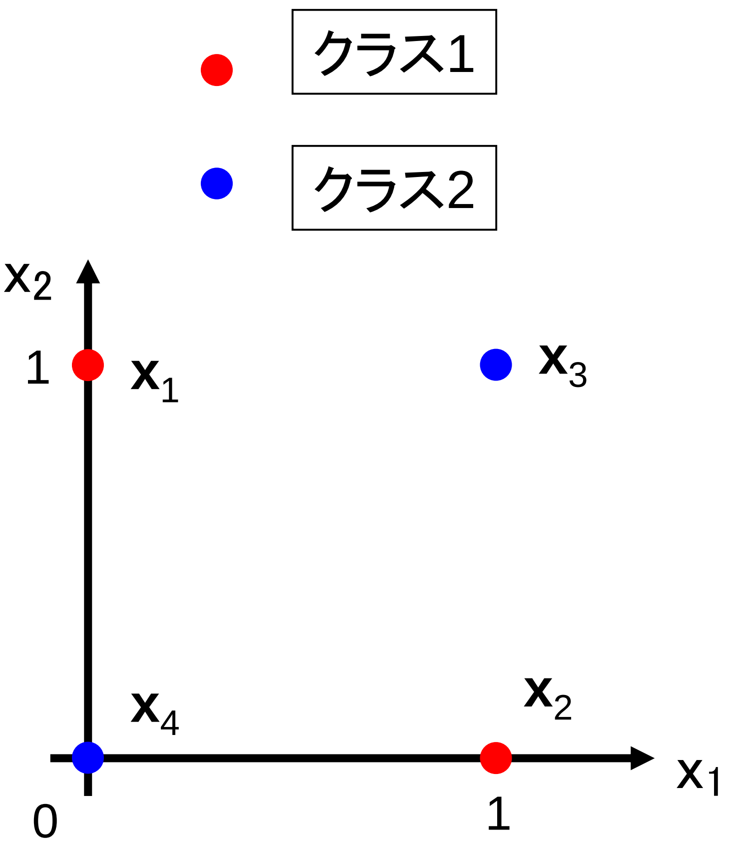
$$W'_{ji} = W_{ji} - \alpha (y_{pj} - t_{pj}) x_{pi}$$

多層パーセプトロン

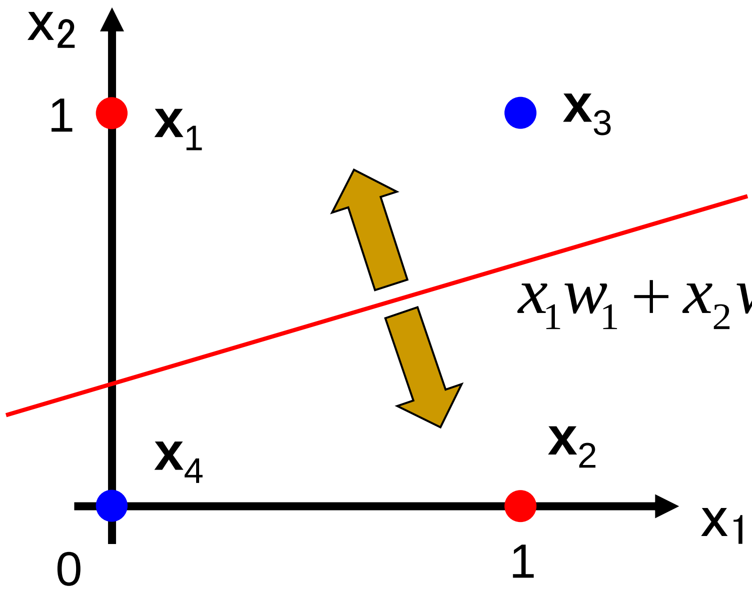
線形分離不可能な問題への対応

線形分離不可能①

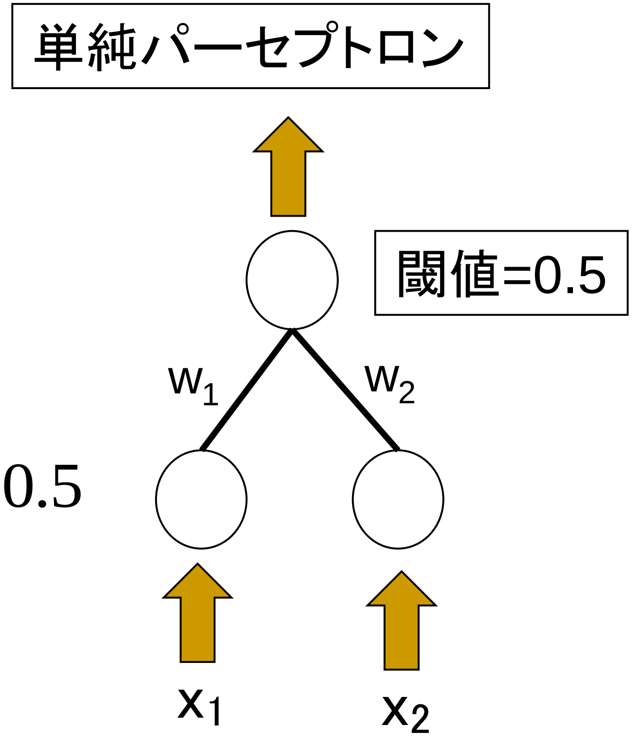
	x_1	x_2	
$\mathbf{x}_1$	0	1	クラス1
$\mathbf{x}_2$	1	0	クラス1
$\mathbf{x}_3$	1	1	クラス2
$\mathbf{x}_4$	0	0	クラス2



線形分離不可能②



$$x_1 w_1 + x_2 w_2 = 0.5$$

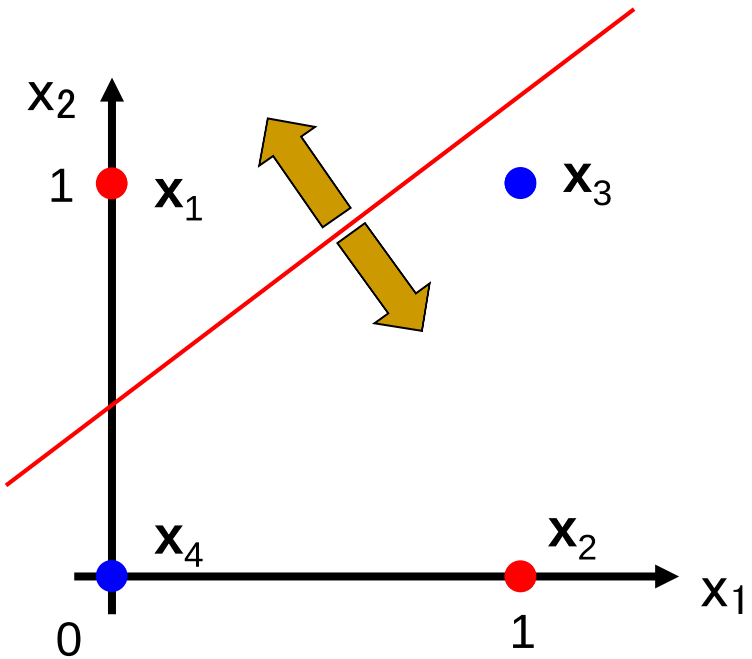


クラス1とクラス2を識別できる重みは存在しない
→ 線形分離不可能
→ パーセプトロンでは解けない問題

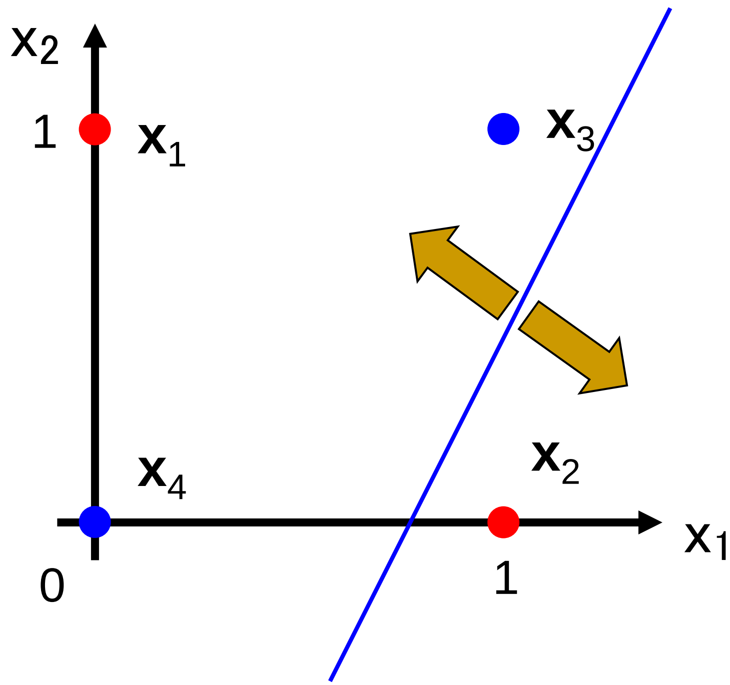
線形識別関数を組み合わせた解法①

識別関数1

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

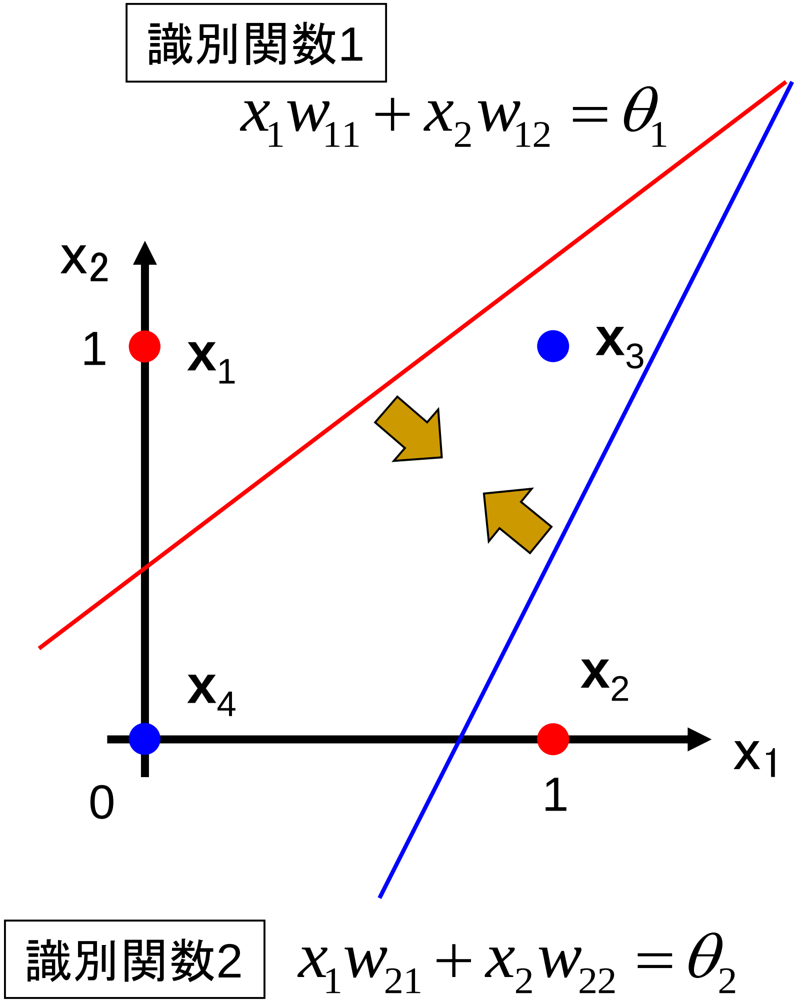


識別関数2



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

線形識別関数を組み合わせた解法②



新しい識別関数

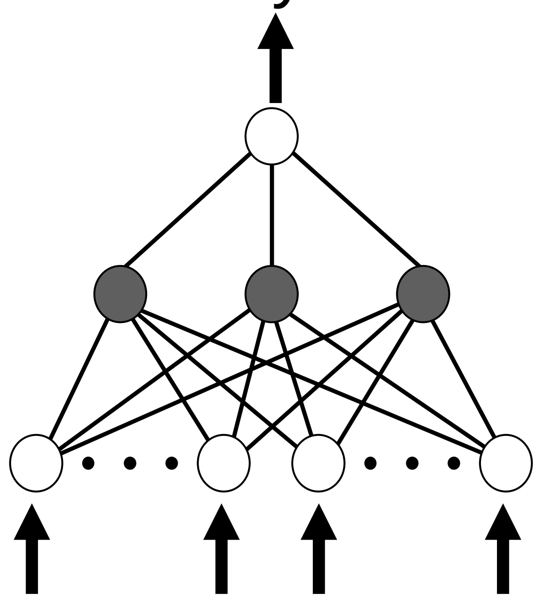
$$F(\mathbf{x}) = \alpha_1 f_1(\mathbf{x}) + \alpha_2 f_2(\mathbf{x})$$

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

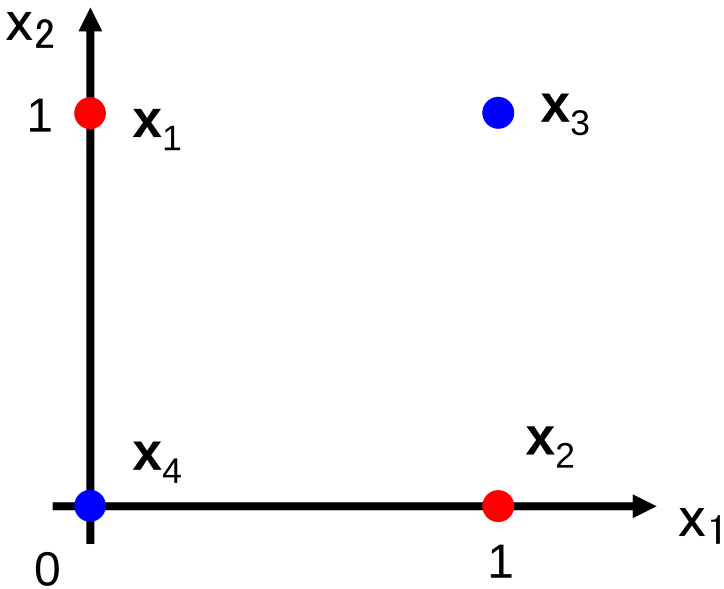
ネットワークの多層化

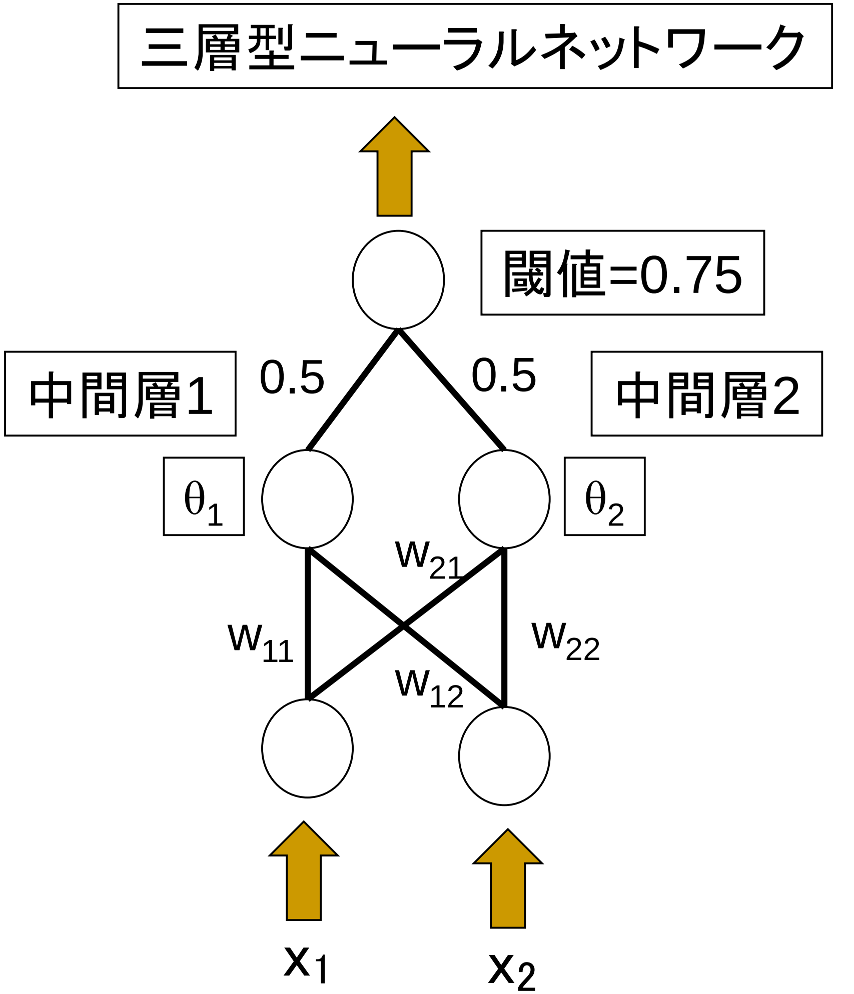
- 階層型ニューラルネットワーク
  - 「多層パーセプトロン」とも呼ばれる
  - Multi-Layer Perceptron（MLP）



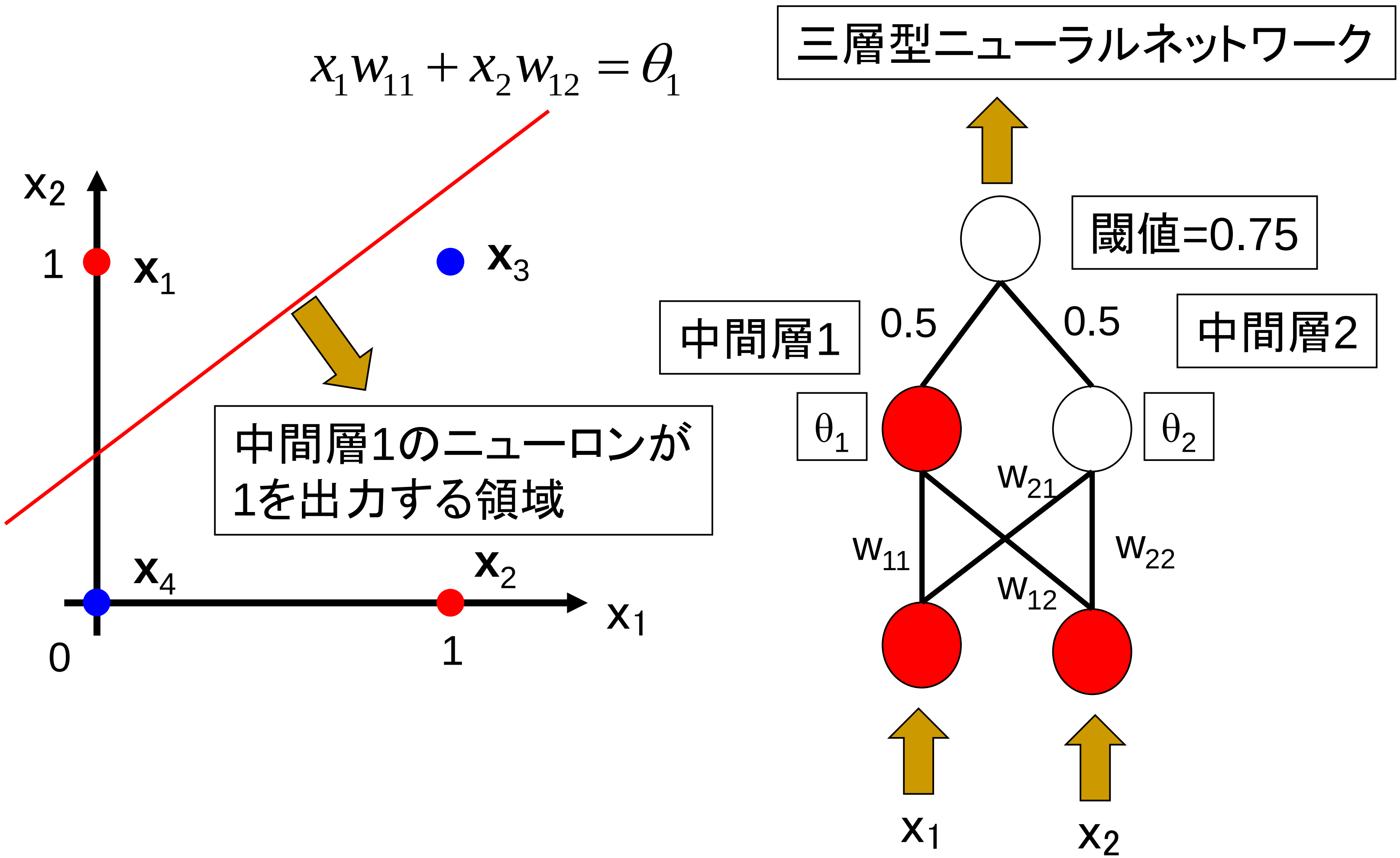
ネットワークを多層化することによって線形分離不可能問題が解決できるか

（直感的ですが…）多層化の意味①

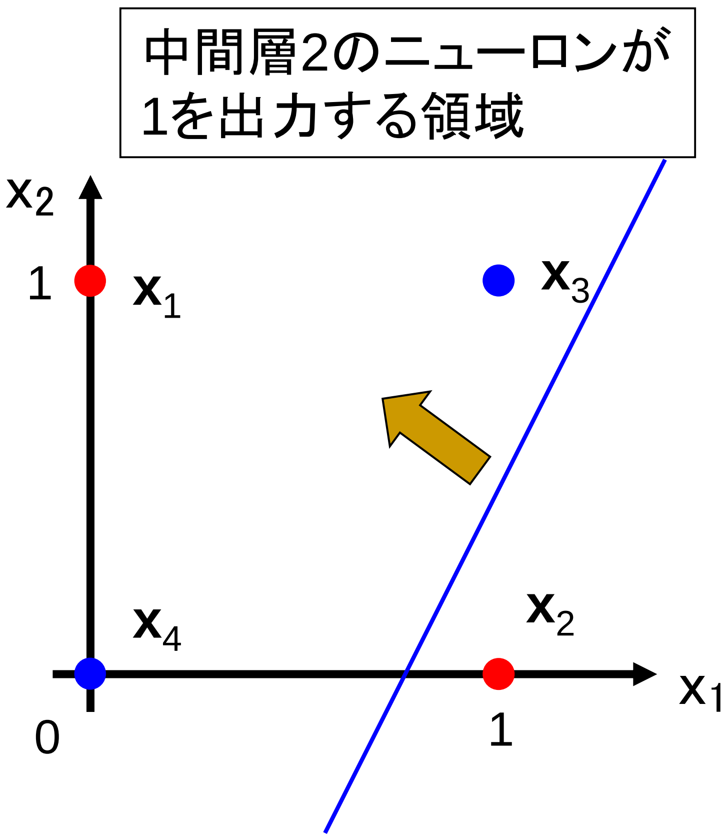




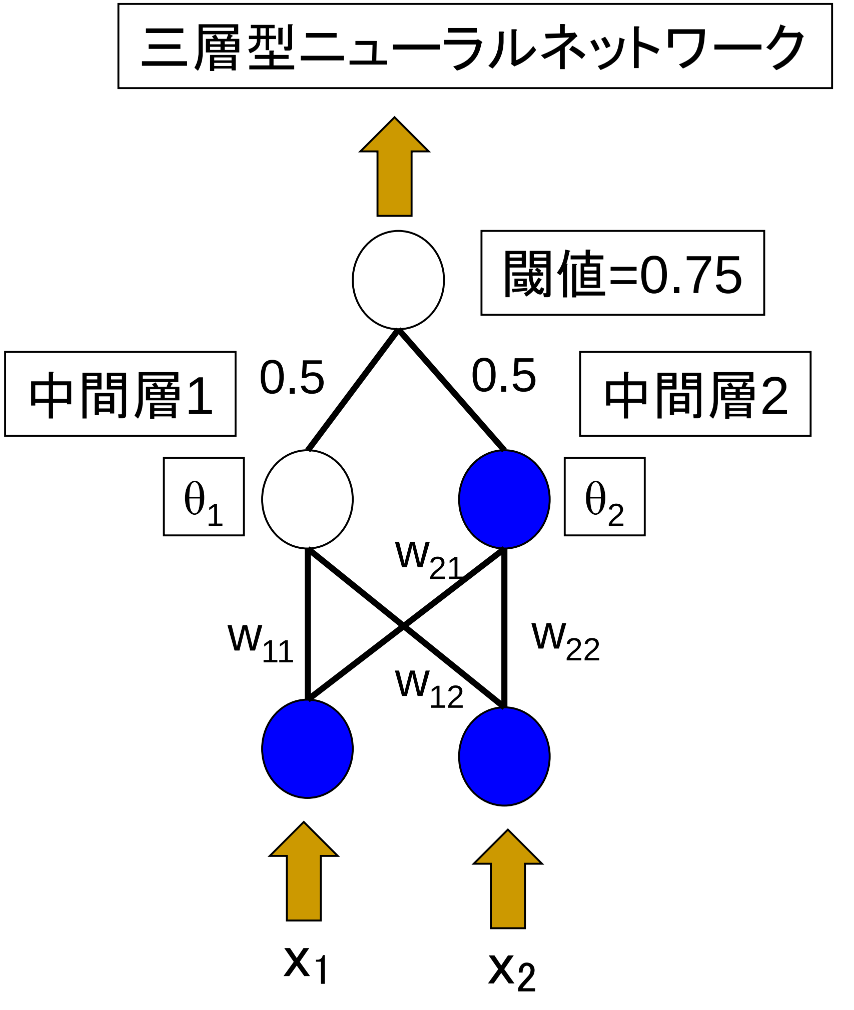
多層化の意味②


$x_1 w_{11} + x_2 w_{12} = \theta_1$
x2
1
x1
x3
中間層1のニューロンが
1を出力する領域
x4
x2
0
1
x1
三層型ニューラルネットワーク
閾値=0.75
中間層1
0.5
0.5
中間層2
θ1
θ2
w21
w11
w22
w12
x1
x2

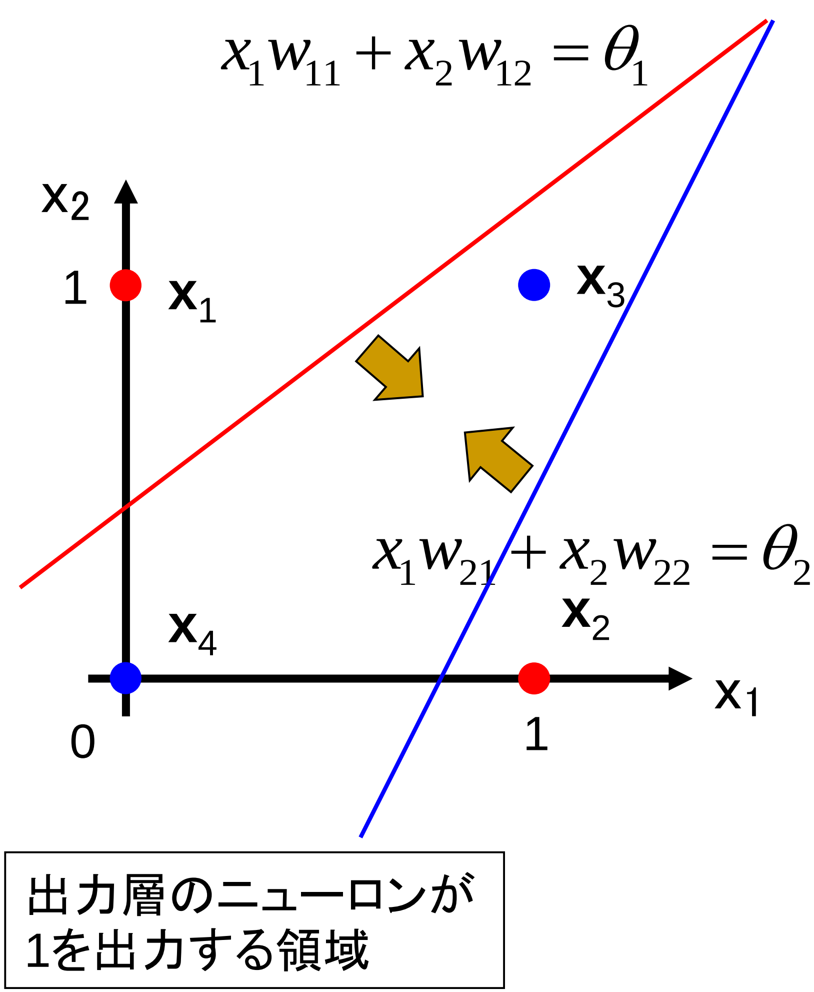
多層化の意味③



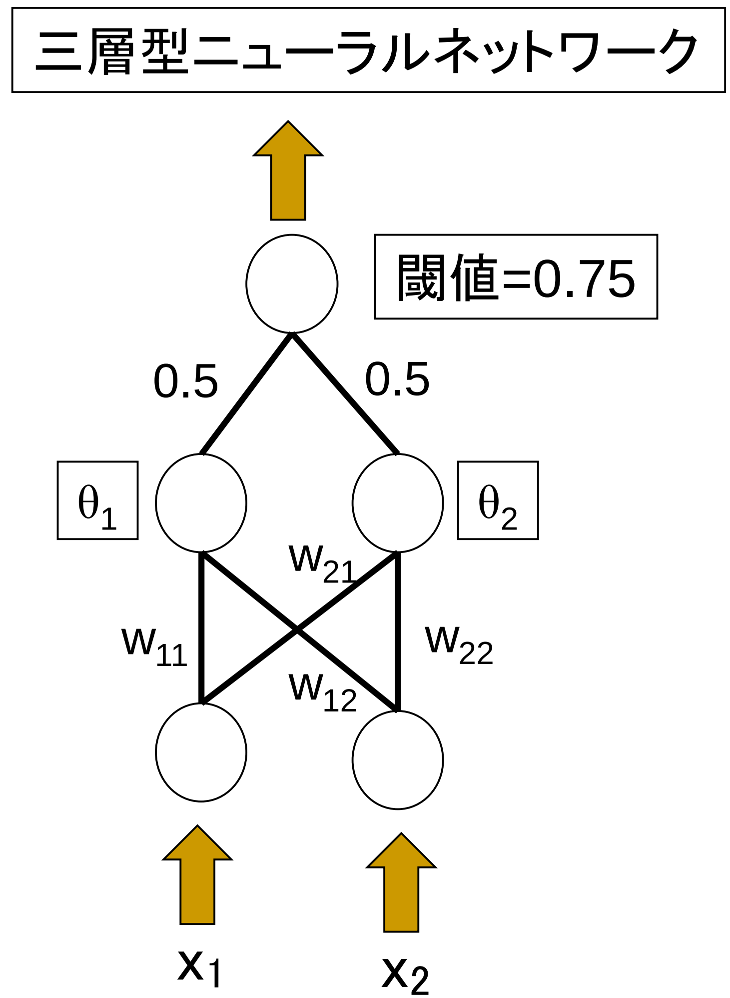
$$x_1 w_{21} + x_2 w_{22} = \theta_2$$



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

学習アルゴリズムの改良

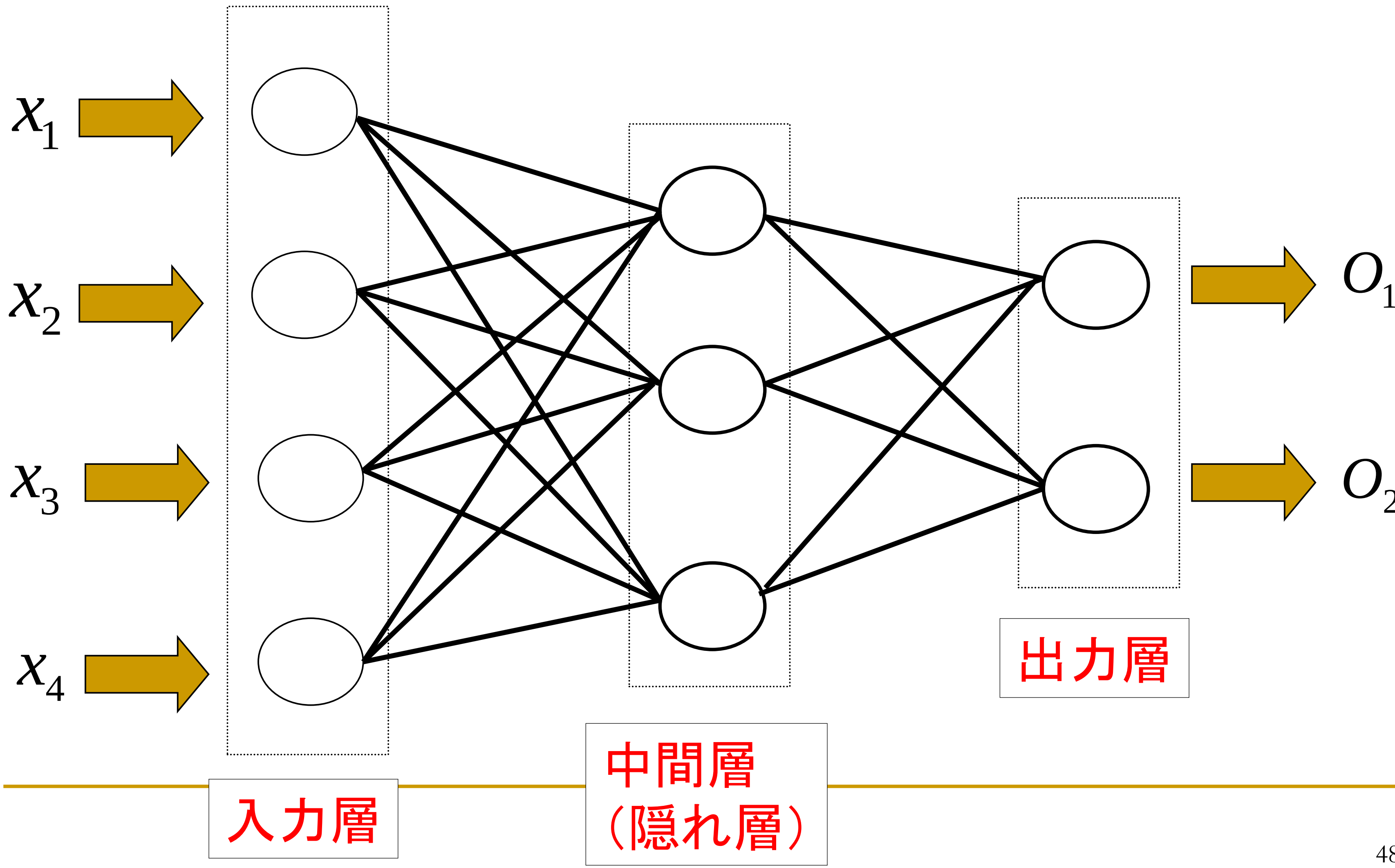
- 単純パーセプトロンでは原理的に線形分離不可能な問題には対応できない

↓

- 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
  - 微分可能な活性化関数（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

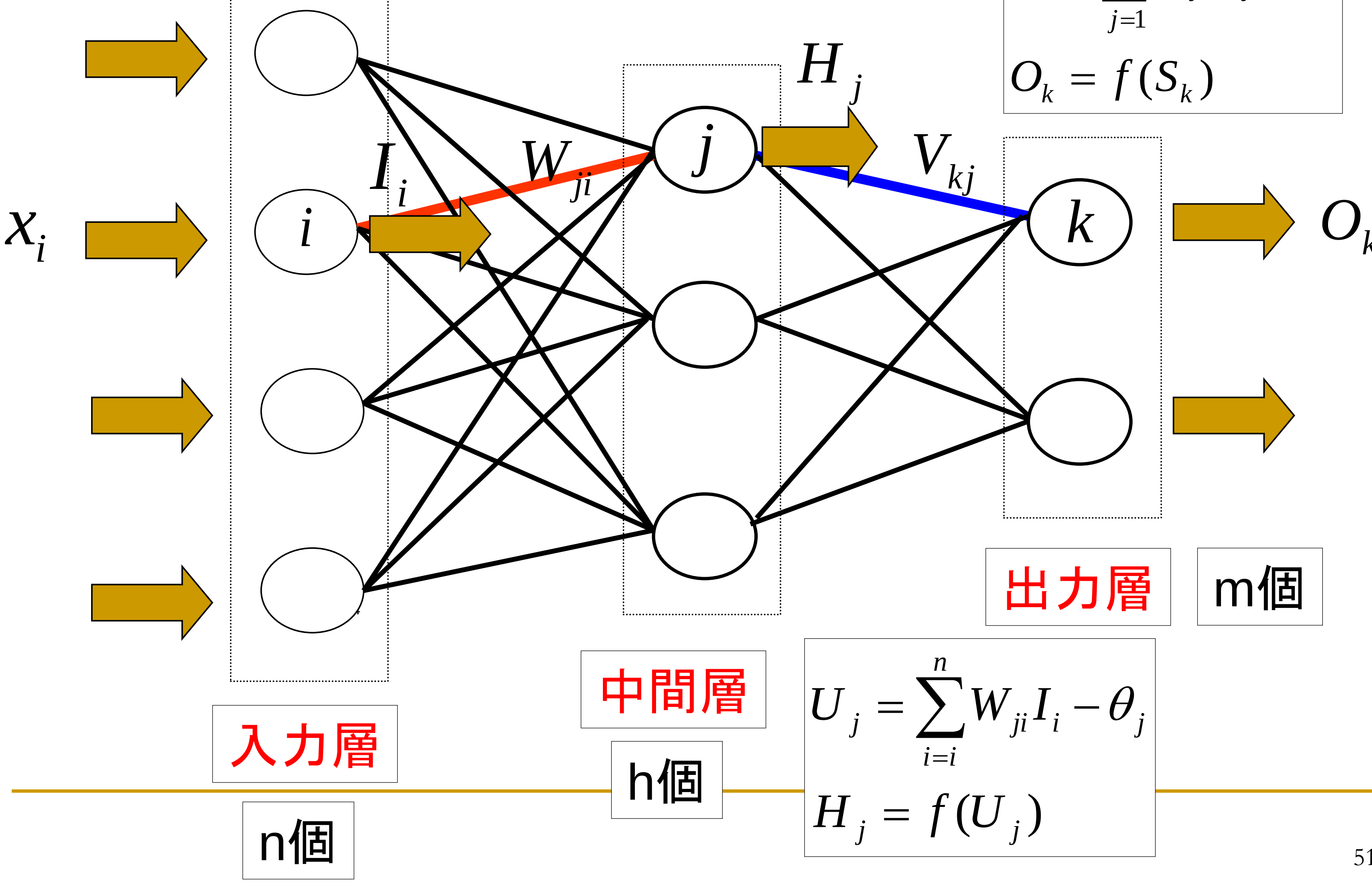
- 入力層のニューロン数　n
- 中間層のニューロン数　h
- 出力層のニューロン数　m

- 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

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

$$O_k = f(S_k)$$

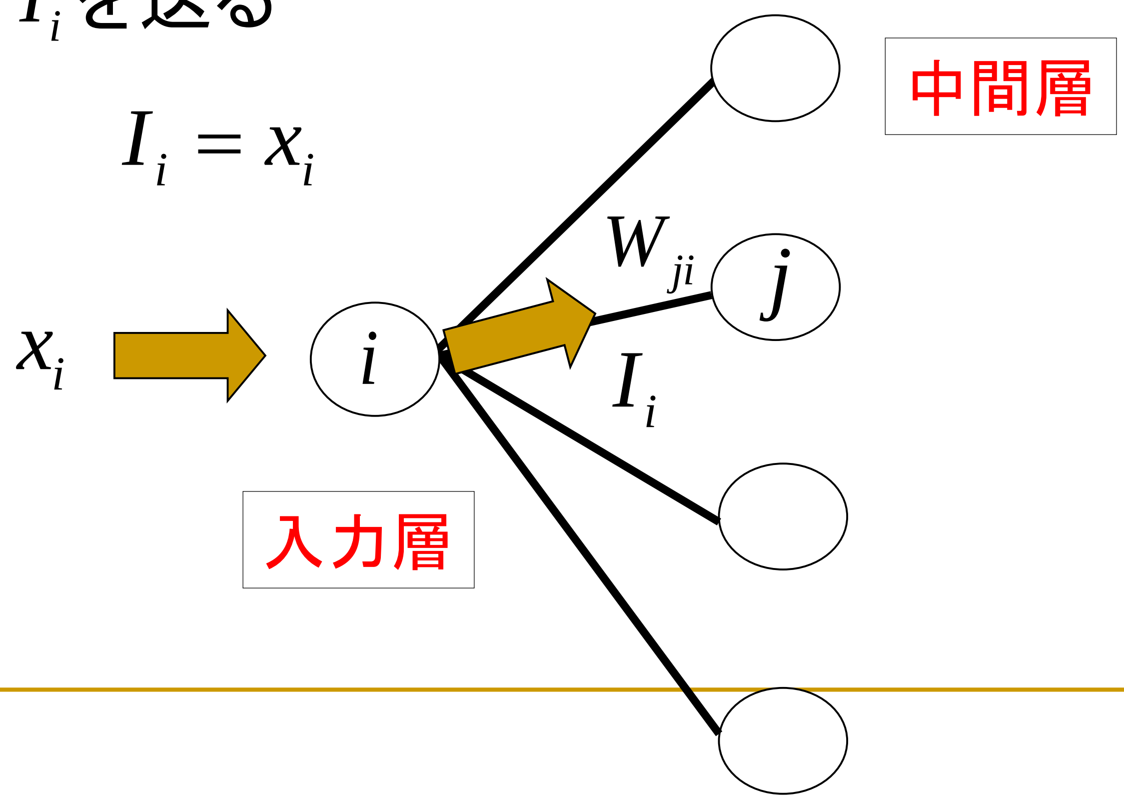


$$U_j = \sum_{i=i}^{n} W_{ji} I_i - \theta_j$$

$$H_j = f(U_j)$$

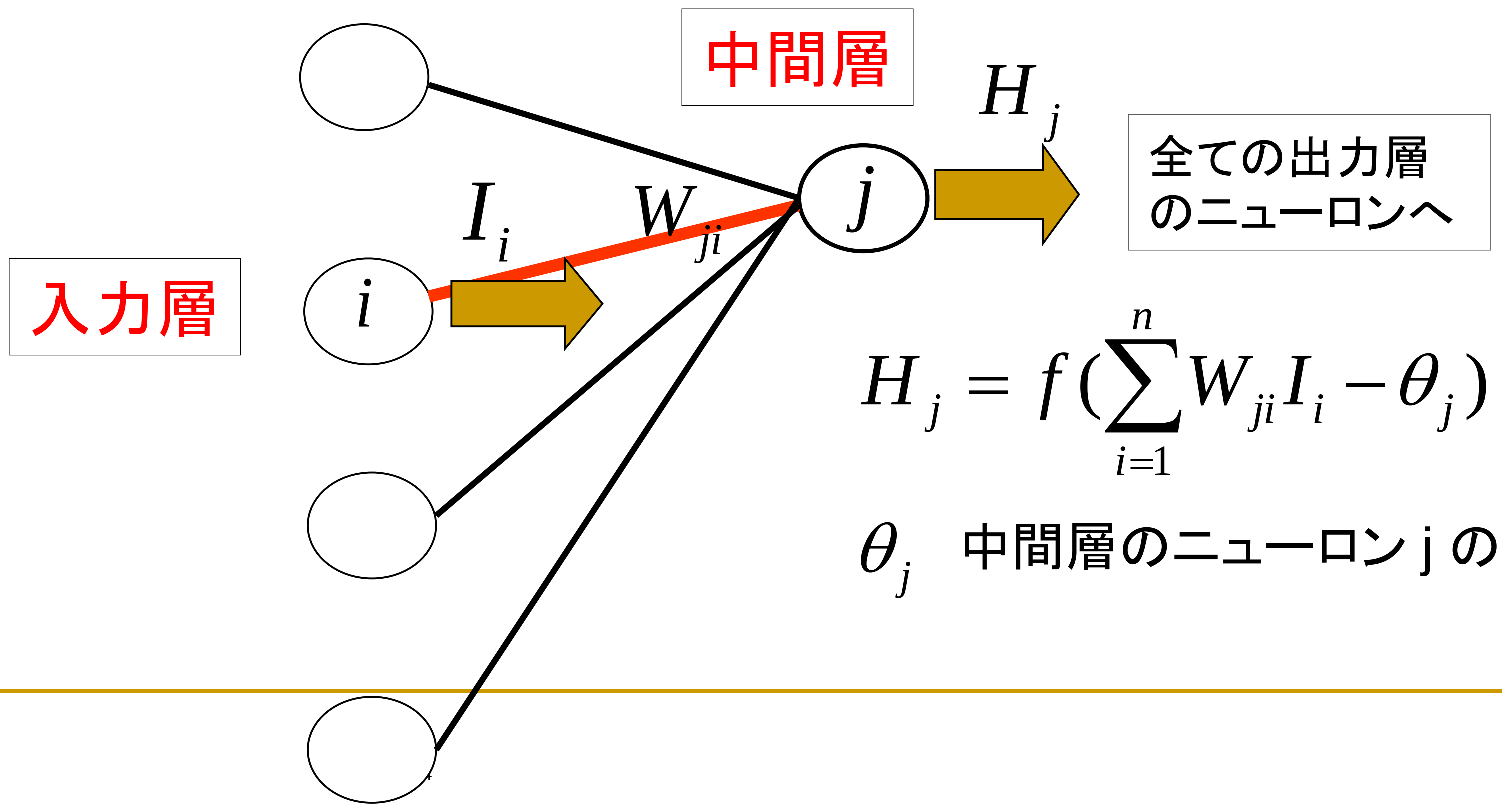
ネットワークの動作①

- 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



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

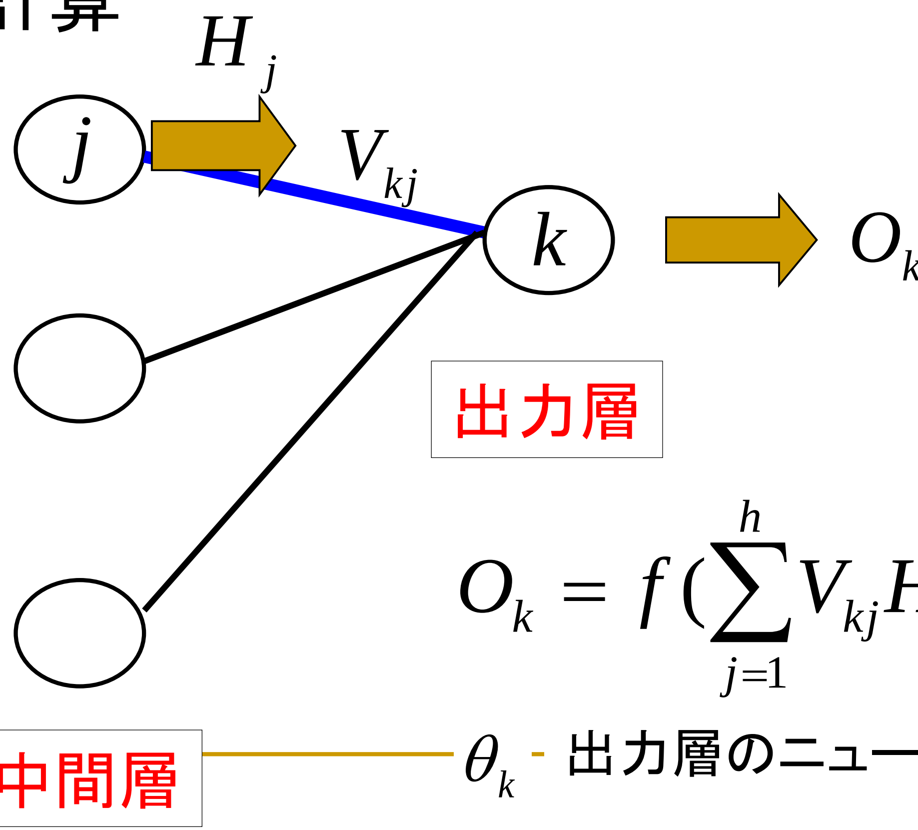


$$H_j = f\left(\sum_{i=1}^{n} W_{ji} I_i - \theta_j\right)$$

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

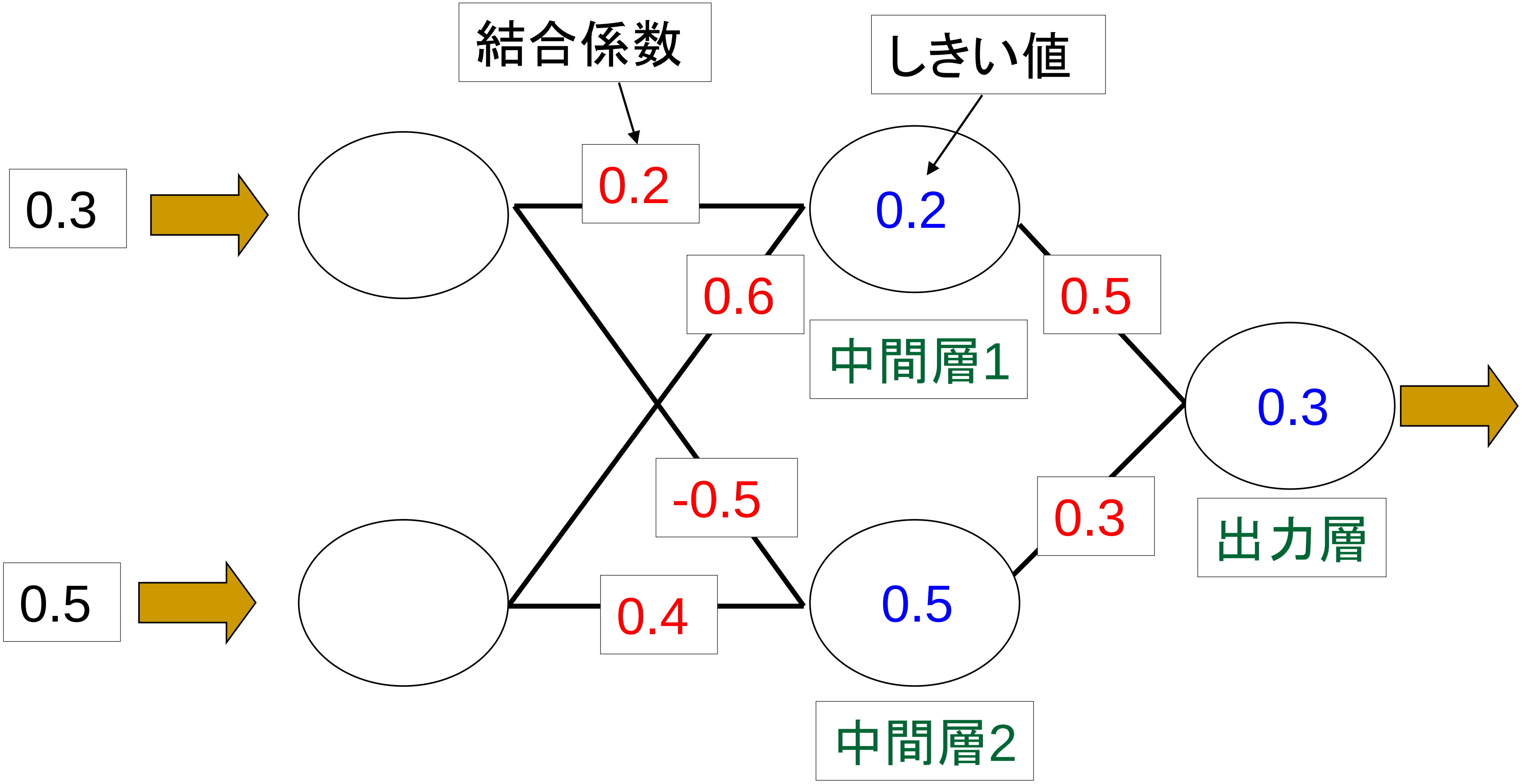
- 出力層の k 番目のニューロン
  - 全ての中間層のニューロンから値を受け取り，出力値を計算



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

θ_k - 出力層のニューロン k のしきい値

ネットワークの動作例①



ネットワークの動作例②

シグモイド関数

$$f(x) = \frac{1}{1+e^{-x}}$$

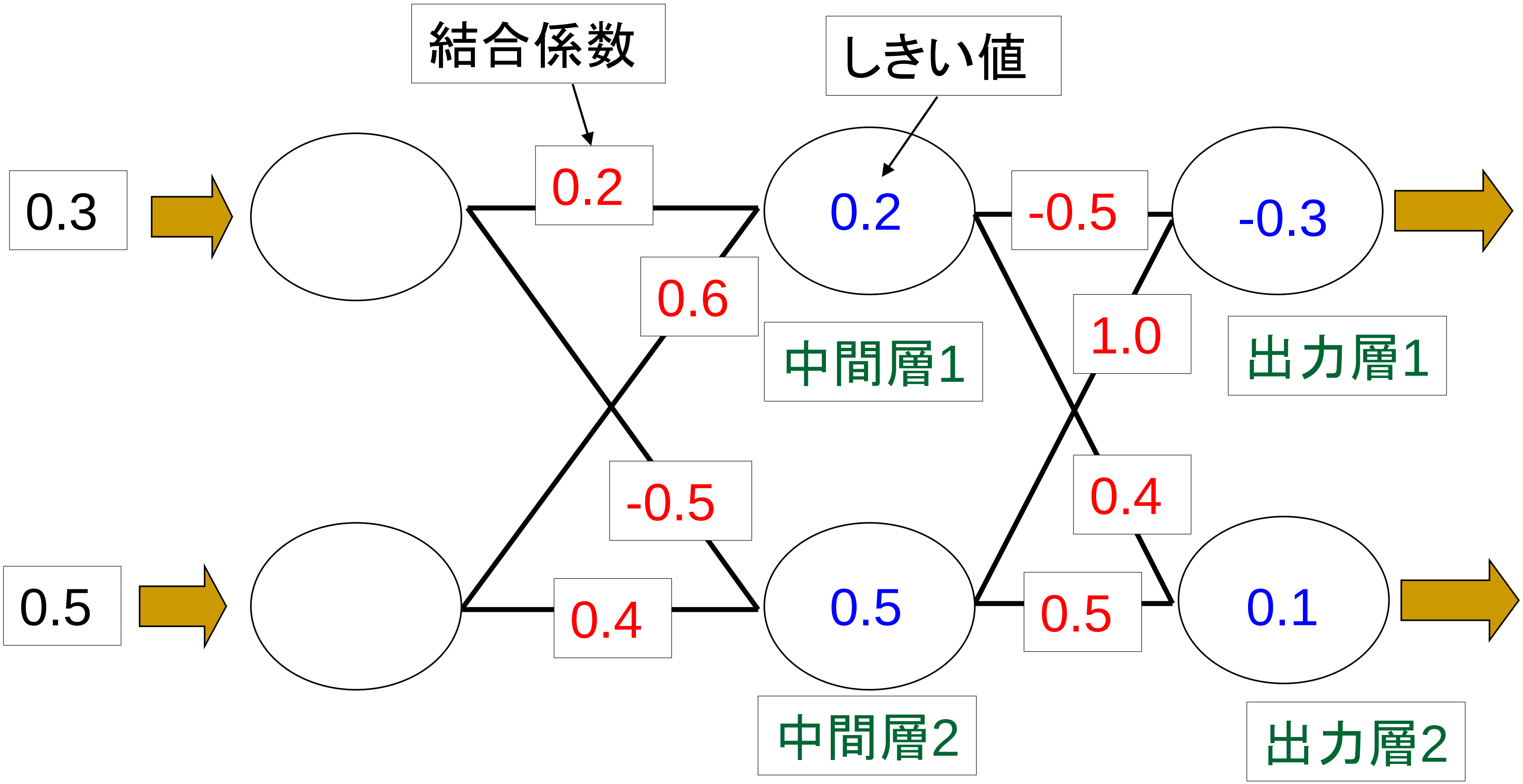
- 中間層1
  - 内部状態
  - 0.3 × 0.2 + 0.5 ×0.6 – 0.2 = 0.16
  - 1 / (1 + exp(-0.16)) = 0.54

- 中間層2
  - 内部状態
  - 0.3 × -0.5 + 0.5 × 0.4 ー 0.5 = -0.45
  - 1 / (1 + exp(0.45)) = 0.39

ネットワークの動作例③

- 出力層
  - 内部状態
  - 0.54 × 0.5 + 0.39 × 0.3 – 0.3 = 0.087
  - 1 / (1 + exp(- 0.087)) = 0.522

ネットワークの動作例④



中間層の出力値までは前のモデルと同じ

ネットワークの動作例⑤

- 出力層1
  - 内部状態
  - 0.54 × -0.5 + 0.39 × 1.0 + 0.3 = 0.42
  - 1 / (1 + exp(-0.42)) = 0.603
- 出力層2
  - 内部状態
  - 0.54 × 0.4 + 0.39 × 0.5 – 0.1 = 0.311
  - 1 / (1 + exp(-0.311)) = 0.577

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$\mathbf{x}_p$

ニューラルネットワーク

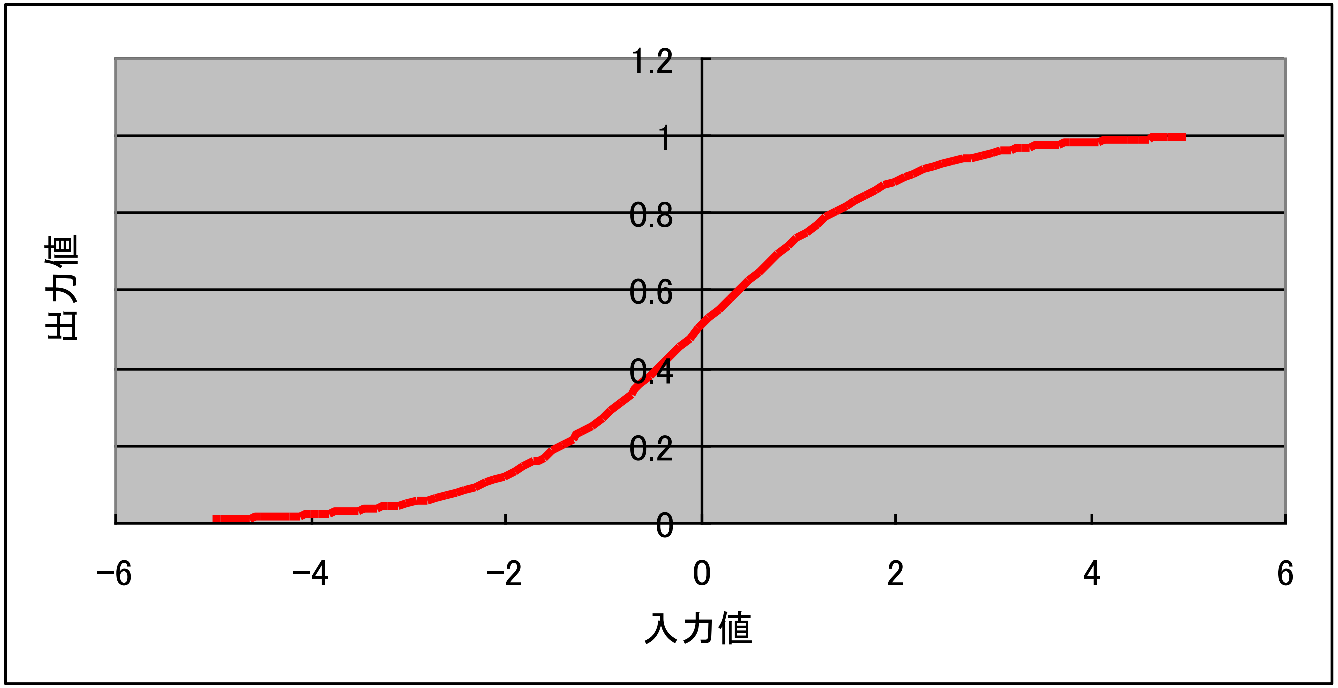
出力

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

望ましい出力結果
→ 教師信号（ベクトル）

出力値の制約

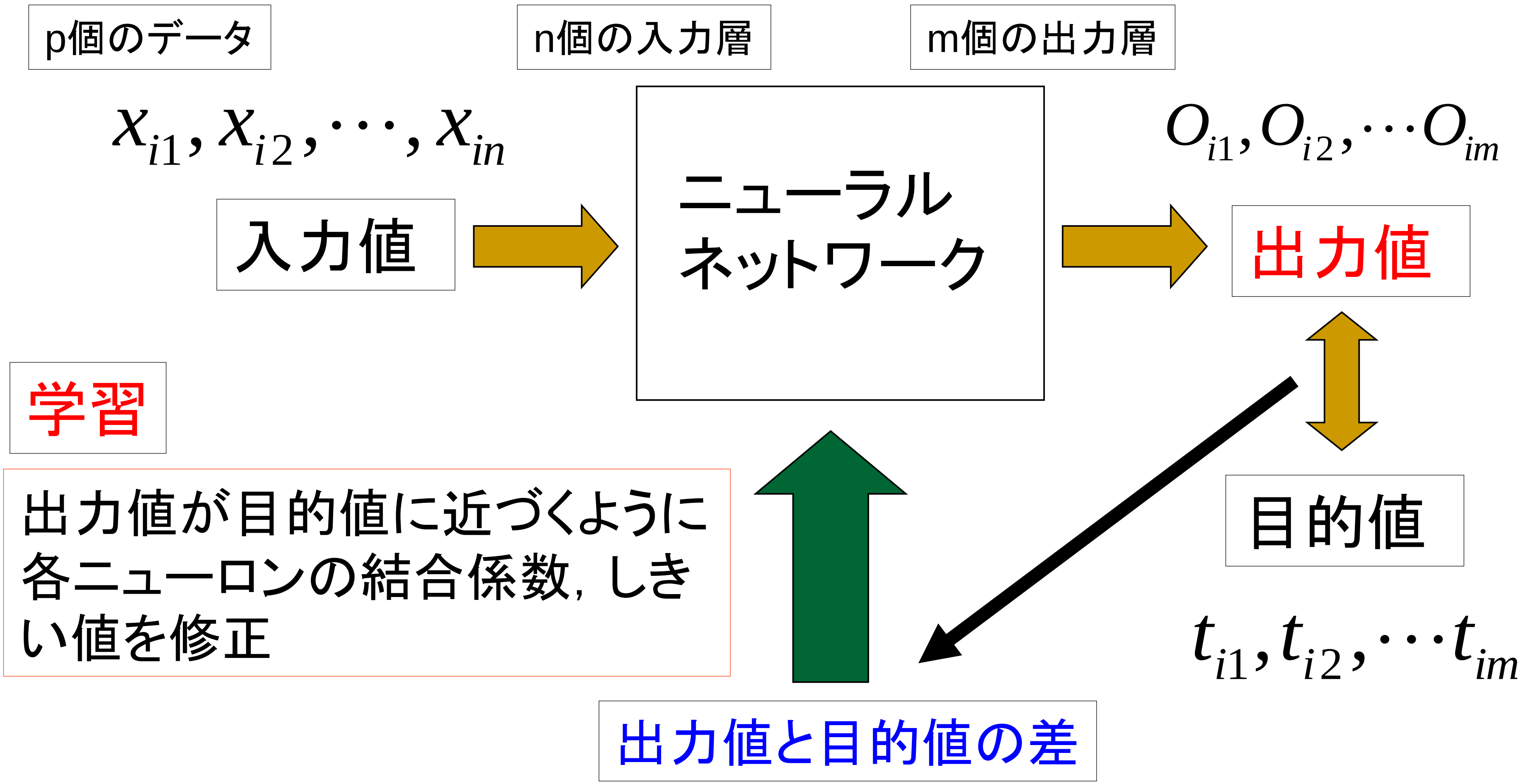
シグモイド関数



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二乗和を最小とするように結合係数(閾値)を決める

定式化

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

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

（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

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

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

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

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

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

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

目的値
（教師信号）

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

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

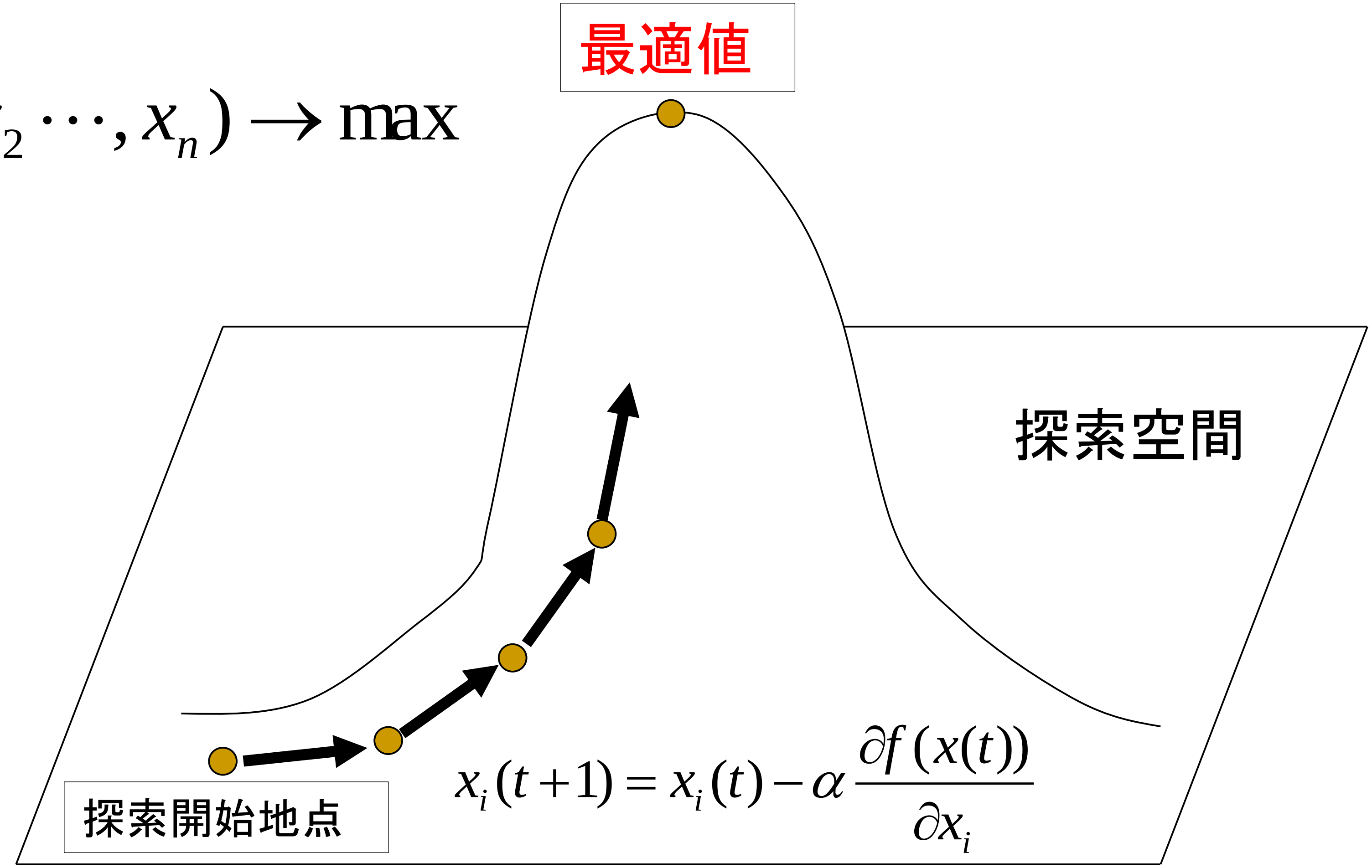
学習アルゴリズム

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 in range(データ数):
　　$O_{i1}, O_{i2}, \cdots O_{im} \longleftarrow f(x_{i1}, x_{i2}, \cdots, x_{in})$
　　$O_{i1}, O_{i2}, \cdots O_{im}$ 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

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



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

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

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

誤差の二乗和の計算

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

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

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

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

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

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

中間層から出力層への結合係数の修正①

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

$$O_k = f(S_k)$$

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

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

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

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

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

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

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

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

シグモイド関数の微分

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

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

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

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

$$= \delta_k H_j$$

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

「誤差」と定義する

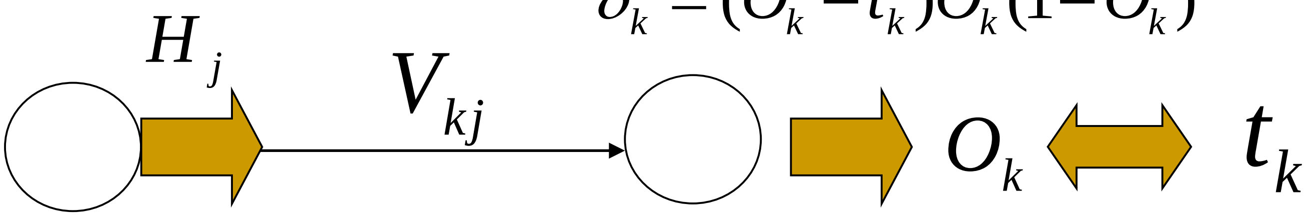
中間層から出力層への結合係数の修正方法

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

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

「誤差」を求める

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



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

目標値
（教師信号）

入力層から中間層への結合係数の修正①

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

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

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

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

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

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

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

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

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

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

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

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

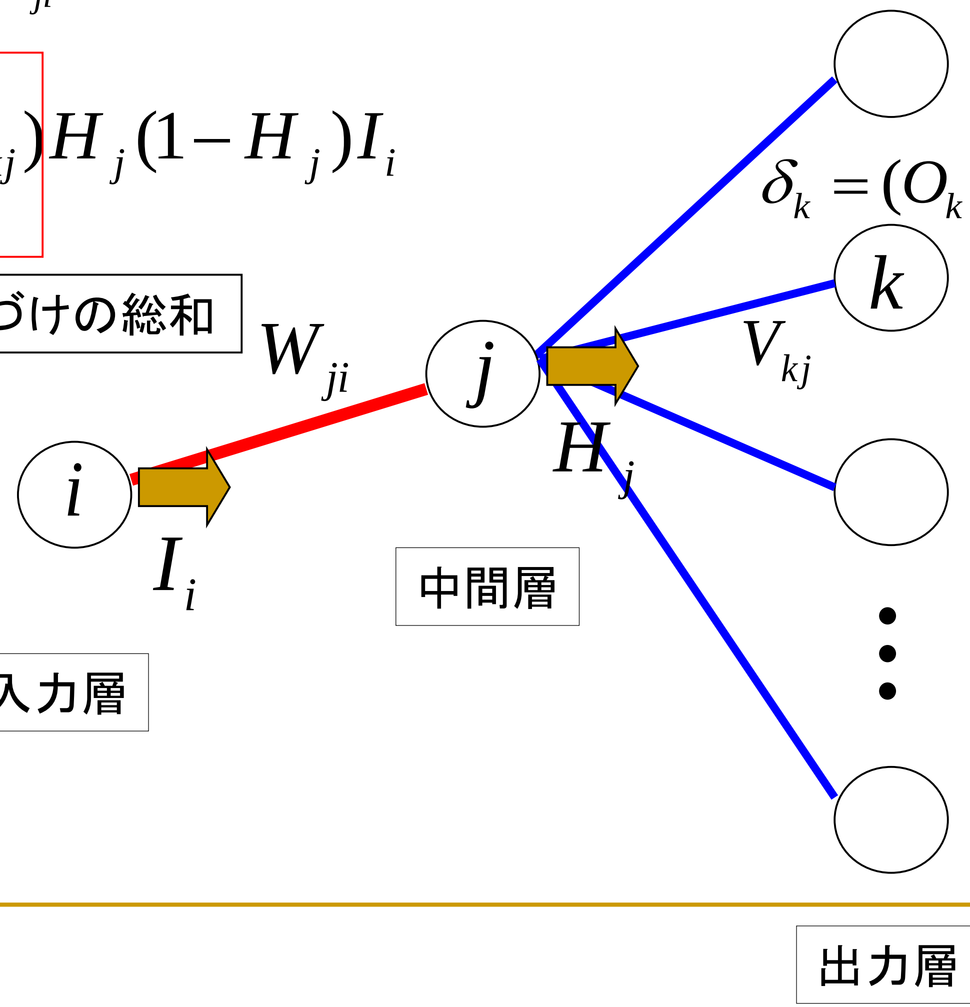
入力層から中間層への結合係数の修正方法

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

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

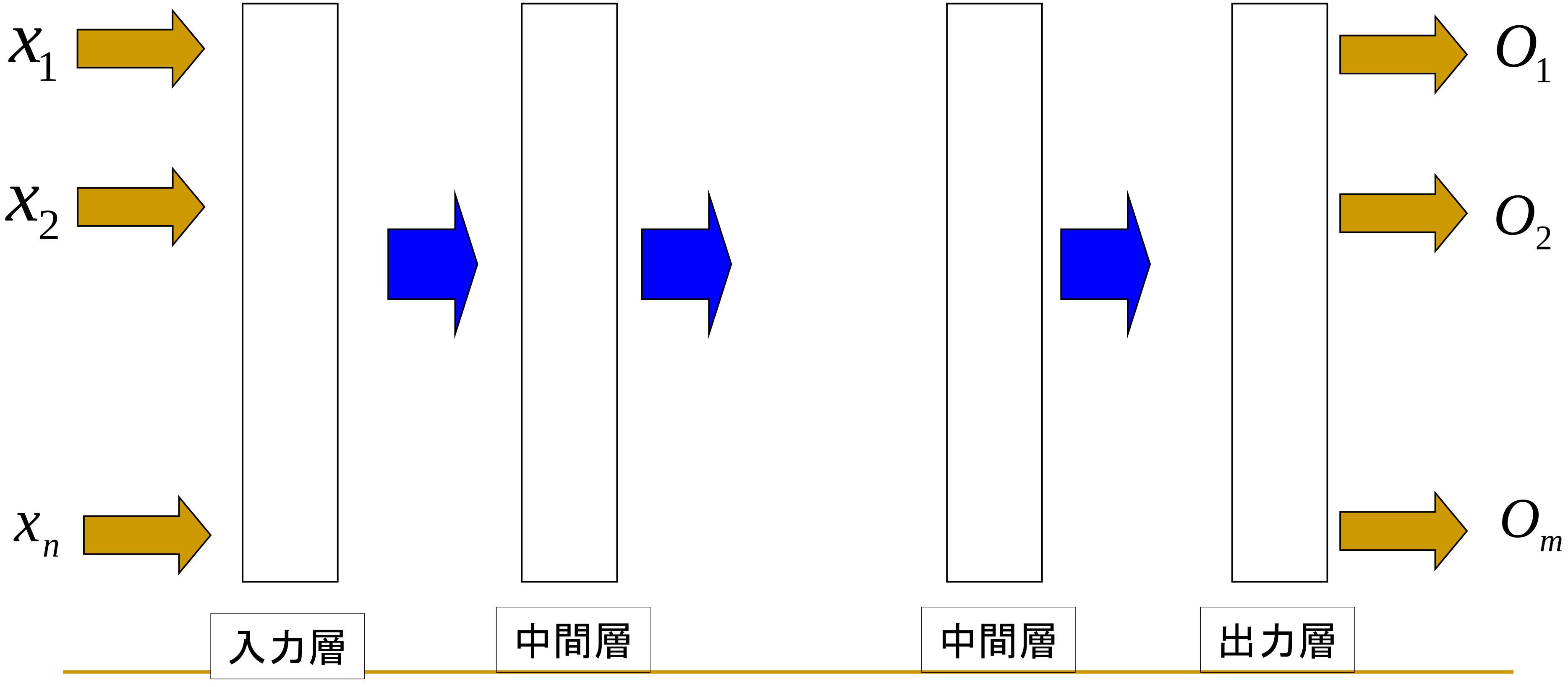
出力層の誤差の重みづけの総和

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



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

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

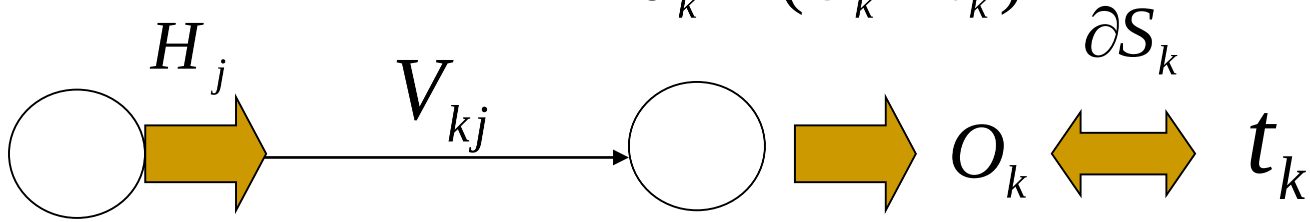
$$S_k = \sum_{j=1}^{h} V_{kj} H_j$$

$$O_k = f(S_k)$$

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

「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

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

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

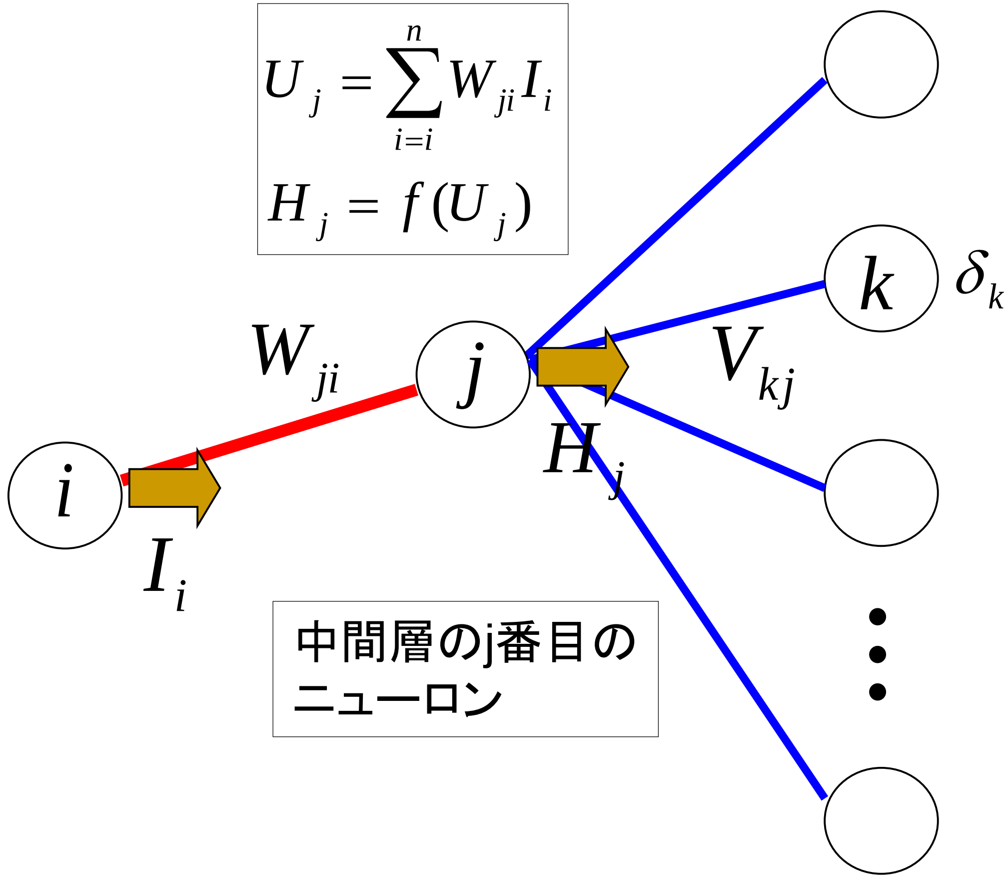
$$H_j = f(U_j)$$

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

一つ上の層の誤差の重み付けの総和

「誤差」と定義

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



出力層→中間層→…というように逆向きに誤差を計算する

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

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

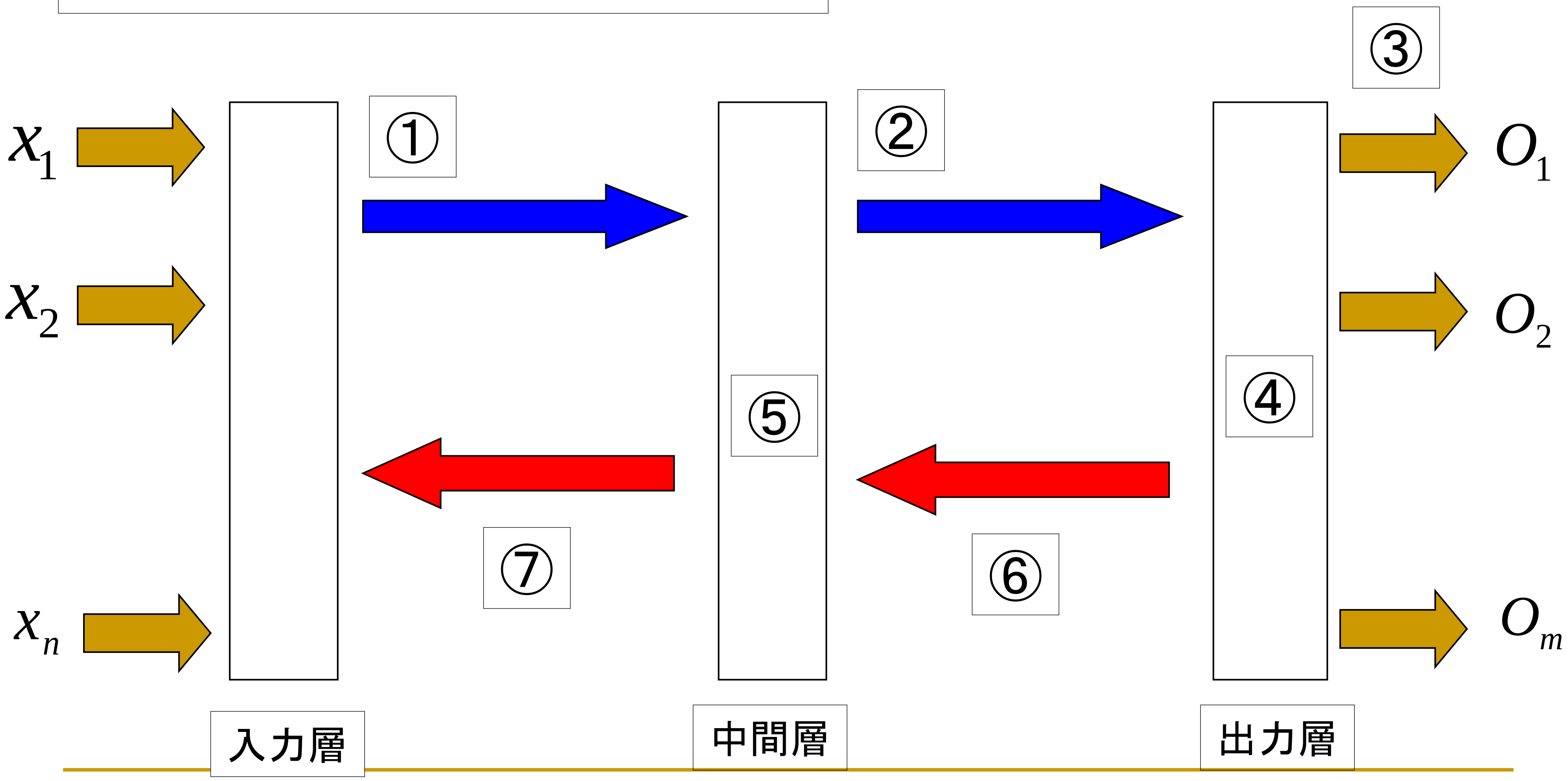
$\boxed{\alpha\text{:学習係数}}$

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

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

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 in range(データ数):

プロパゲーション（①～③）

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

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

差の合計値 += 差

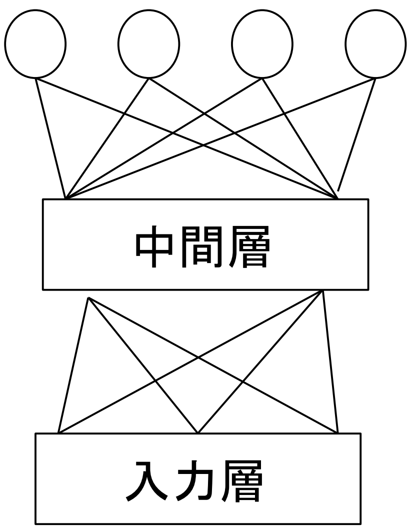
誤差逆伝播アルゴリズムを用いてニューラルネットワークのパラメーターを修正

バックプロパゲーション（④～⑦）

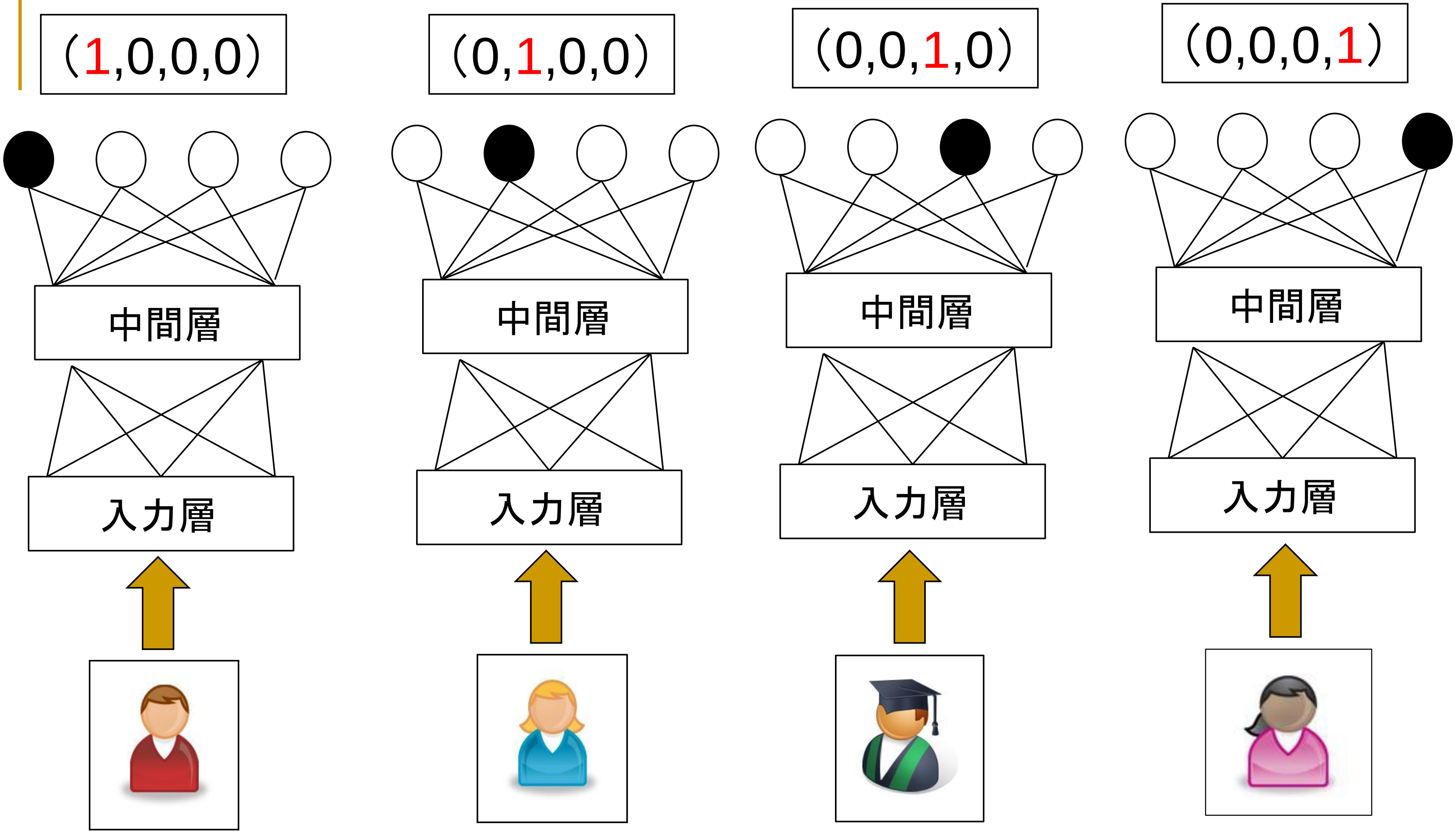
if 差の合計値 < ε :break

（応用例①）顔画像認識

A　B　C　D



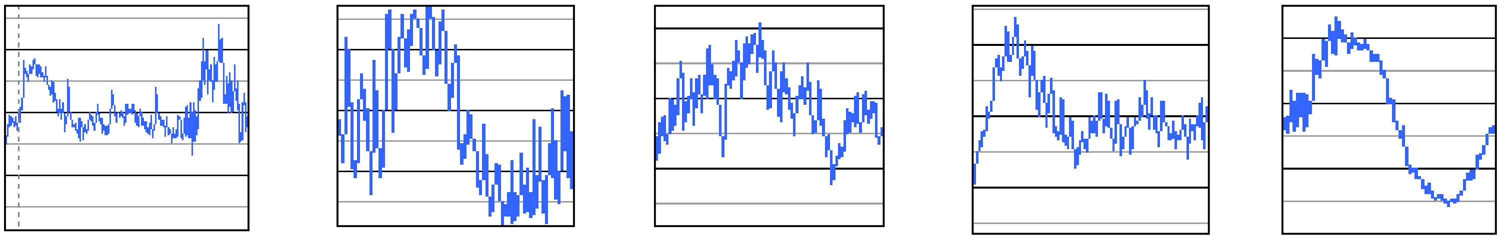
出力層 → 4個
中間層 → 任意
入力層 → 顔画像の特徴数



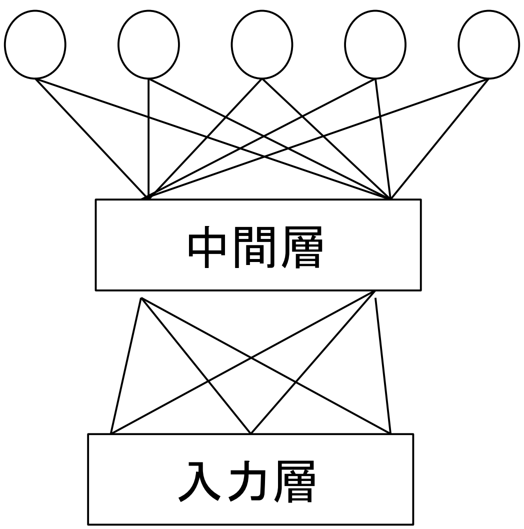
入力する顔画像ごとに，教師信号に従って出力するようにネットワークを学習

（応用例②）音声認識

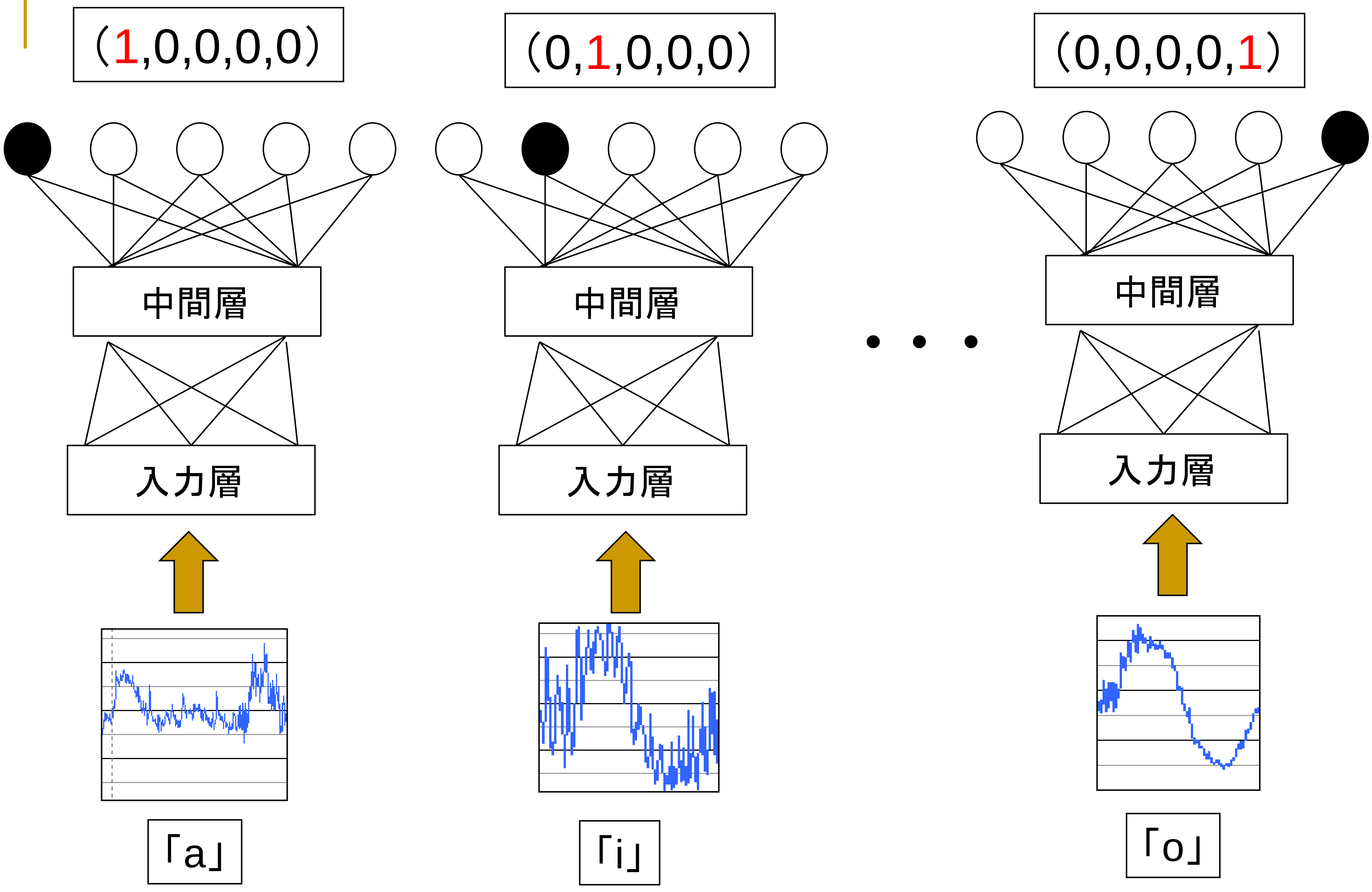
母音の解析



「a」 「i」 「u」 「e」 「o」



出力層 → 5個
中間層 → 任意
入力層 → 音声（波形）の特徴数


(1,0,0,0,0)
(0,1,0,0,0)
(0,0,0,0,1)
中間層
中間層
中間層
入力層
入力層
入力層
・・・
「a」
「i」
「o」

問題点

- 学習の収束(誤差二乗和*が0となること)が遅い
  - さまざまな改良方法が考案されている
- ローカルミニマム問題
- 過学習
- 汎化性

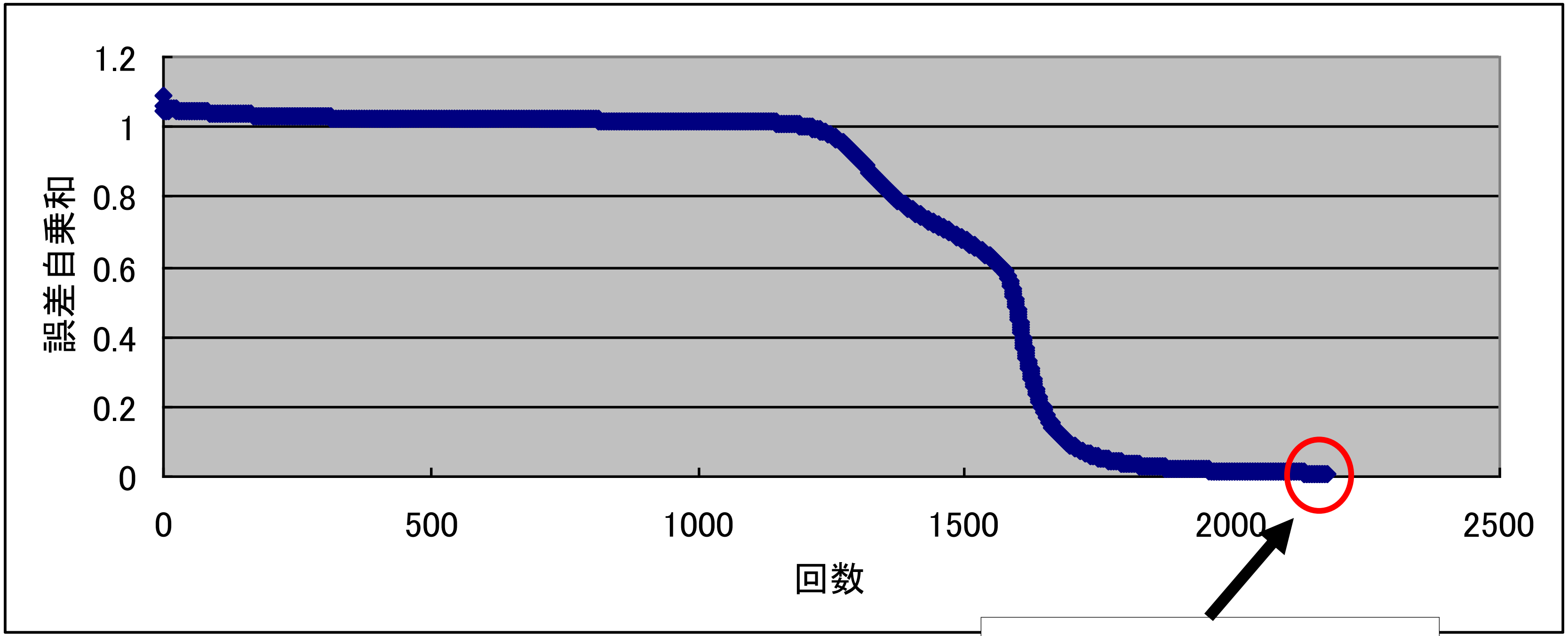
*損失関数がクロスエントロピーの場合もあります

学習回数と誤差二乗和の関係①

誤差二乗和

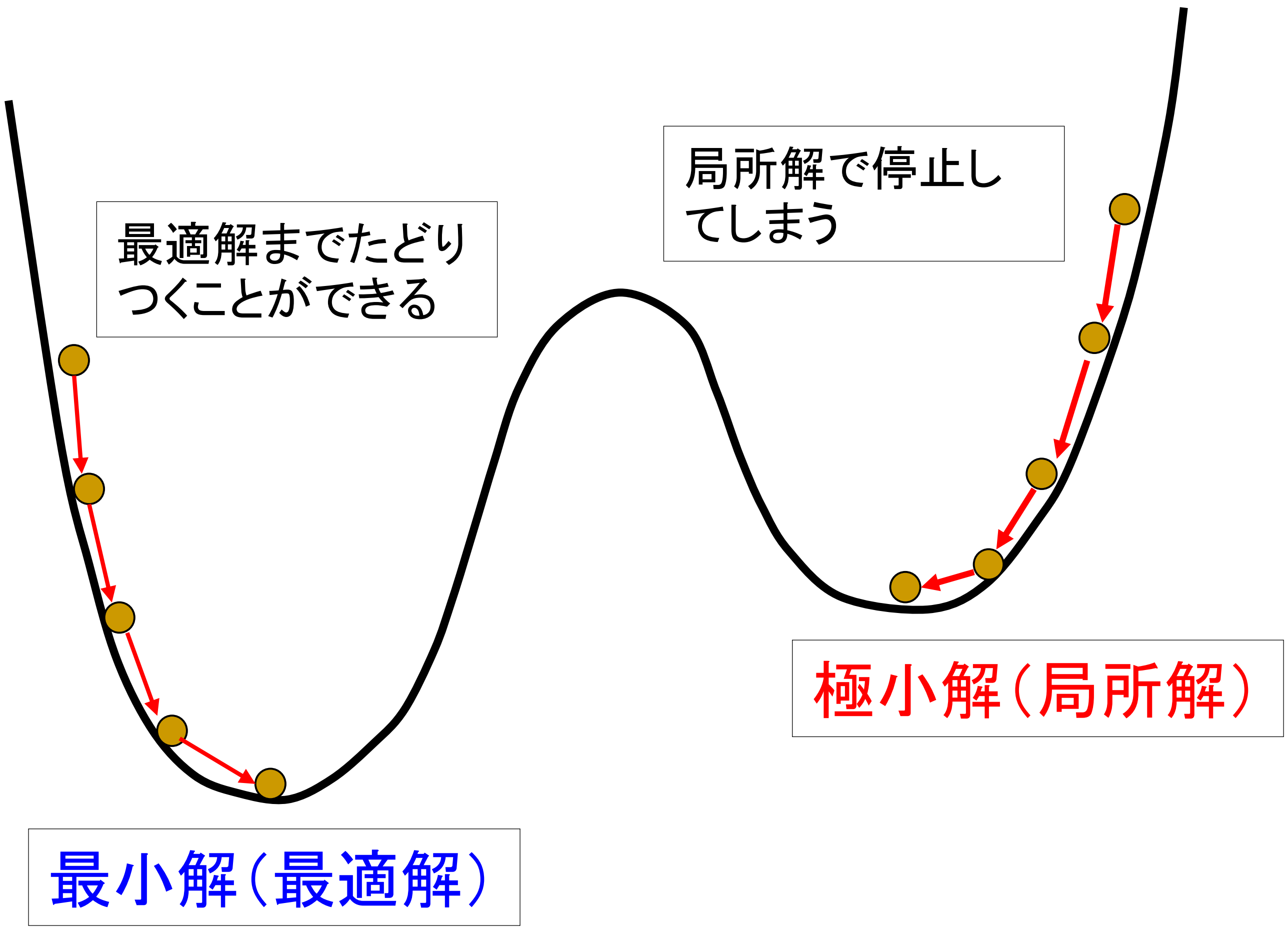
$$E=\sum_{p=1}^{P}\sum_{k=1}^{m}(O_{pk}-t_{pk})^2$$

誤差二乗和が0となれば，学習は適切に行なえたと判断する



学習終了（収束）

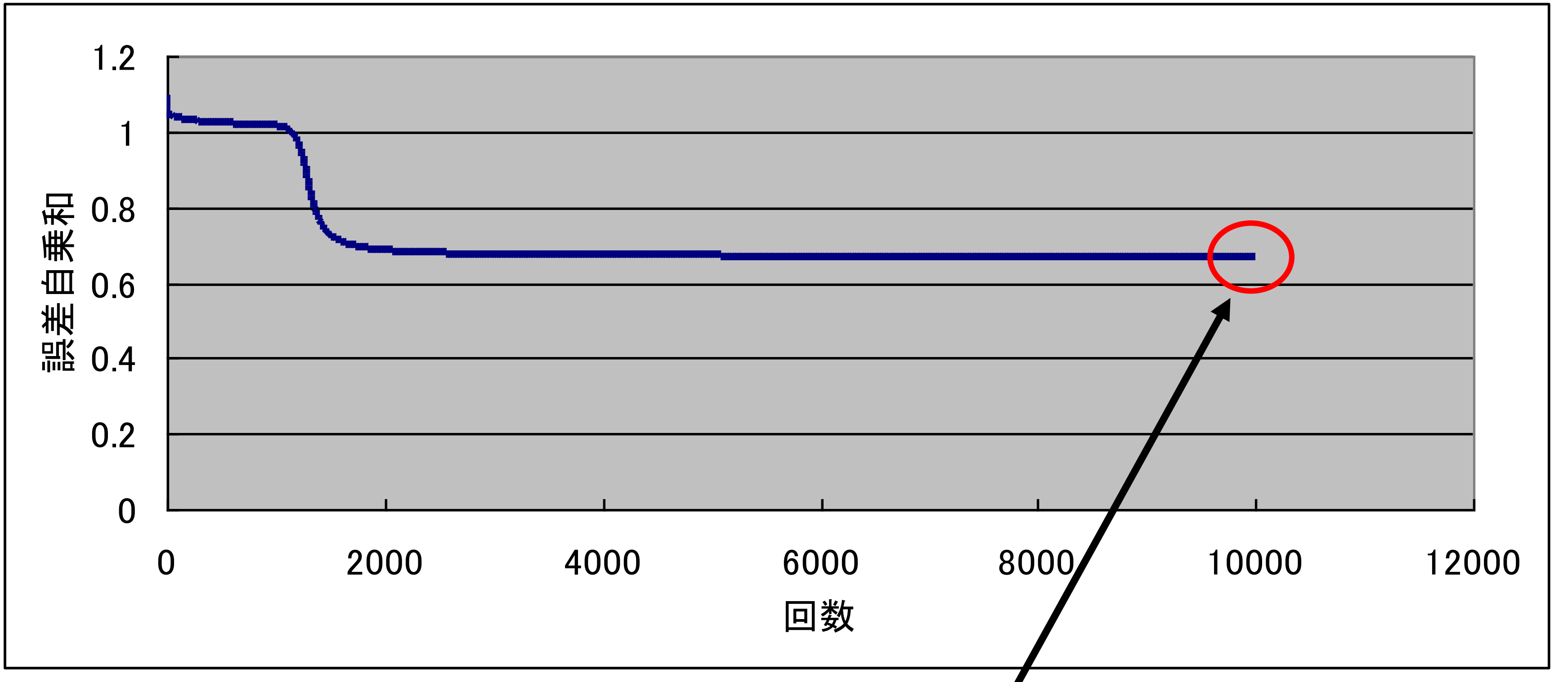
ローカルミニマム問題



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

学習回数と誤差二乗和の関係②

学習の失敗例



誤差二乗和が0に近づかない（収束しない）

汎化と過学習

- 汎化性
  - 学習データより，クラス内のパターンに内在する規則性を学習
  - 学習データ以外のパターンも適切に予測可能となる
- 過学習
  - 学習データより，学習データ固有の規則性を学習
  - 学習データ以外のパターンに対しては適切に予測ができない（予測精度が低い）

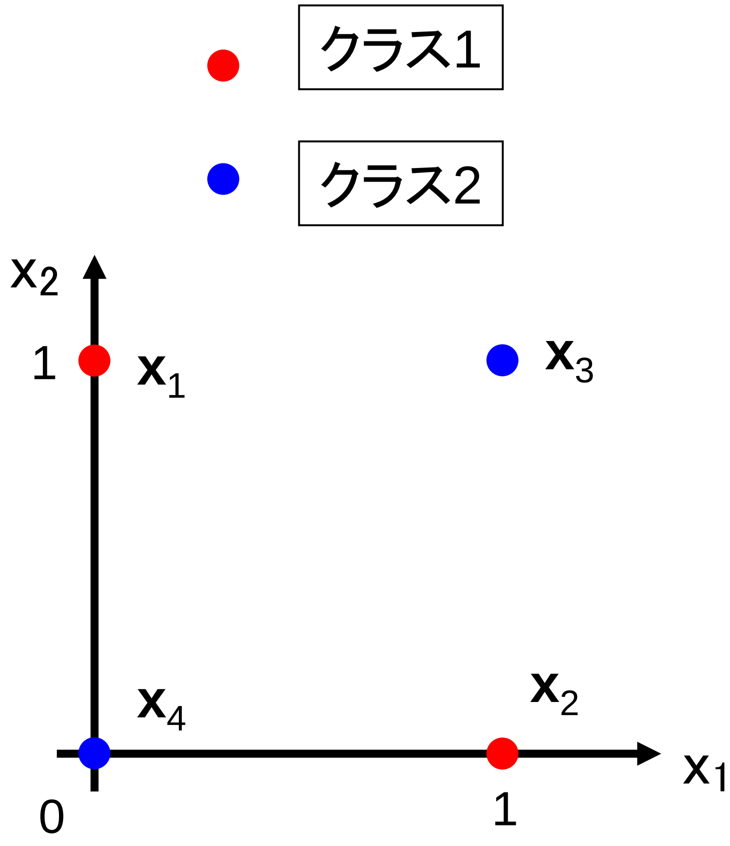
ニューラルネットワークのプログラム

XOR問題

文字認識（Digitsデータベース）

XOR問題

	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



パーセプトロン（XOR_Perceptron.py）

```
import numpy as np
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import sys

# 学習データ
train_data = np.array( [[0,1],[1,0],[1,1],[0,0]] )

# 正解ラベル
train_label = np.array( [0,0,1,1] )

# パーセプトロン
model = Perceptron()
```

train_data　train_label

	x_1	x_2	ラベル
0	0	1	0
1	1	0	0
2	1	1	1
3	0	0	1

```
# 学習
model.fit(train_data, train_label)

# 予測
predict = model.predict(train_data)
df = model.decision_function(train_data)

# 予測，正解値の表示
for i in range(len(train_data)):
   print( train_data[i][0] , train_data[i][0] , "->" , df[i] , "(", train_label[i] , ")" )

print( "¥n [ 予測結果 ]" )
print( classification_report(train_label, predict) )

print( "¥n [ 正解率 ]" )
print( accuracy_score(train_label, predict) )

print( "¥n [ 混同行列 ]" )
print( confusion_matrix(train_label, predict) )
```

予測確率
（1である確率）

正解ラベル

実行結果（パーセプトロンの場合）

x1　x2　予測値

```
0 0 -> 0.0 ( 0 )
1 1 -> 1.0 ( 0 )
1 1 -> 1.0 ( 1 )
0 0 -> 0.0 ( 1 )

 [ 予測結果 ]
              precision    recall  f1-score   support

           0       0.50      0.50      0.50         2
           1       0.50      0.50      0.50         2

    accuracy                           0.50         4
   macro avg       0.50      0.50      0.50         4
weighted avg       0.50      0.50      0.50         4


 [ 正解率 ]
0.5

 [ 混同行列 ]
[[1 1]
 [1 1]]
```

正解ラベル

パーセプトロンでは学習できない

三層型の場合（XOR_NN.py）

```
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix
import sys


# 学習データ
train_data = np.array( [[0,1],[1,0],[1,1],[0,0]] )


# 正解ラベル
train_label = np.array( [0,0,1,1] )


# 階層型ニューラルネットワーク
model = MLPClassifier(hidden_layer_sizes=128,activation='tanh',max_iter=500)
```

max_iter
繰り返し回数

hidden_layer_sizes
中間層のニューロン数

activation
活性化関数

```
from sklearn.neural_network import MLPClassifier
MLPClassifier(hidden_layer_sizes=中間層の個数, activation='活性化関数', max_iter=学習回数)
```

```
model = MLPClassifier(hidden_layer_sizes=128,activation='tanh',max_iter=500)
```

hidden_layer_sizes：中間層のニューロン数
三層型の場合（個数は128個）→ hidden_layer_sizes=128

```
model = MLPClassifier(hidden_layer_sizes=(32,32),activation='tanh',max_iter=500)
```

四層型の場合（第一中間層，第二中間層のニューロン数とも32個の場合）

```
model = MLPClassifier(hidden_layer_sizes=(32,32,32),activation='tanh',
max_iter=500)
```

五層型の場合

```
model = MLPClassifier(hidden_layer_sizes=128,activation='tanh',max_iter=500)
```

activation：活性化関数
tanh

- activation：活性化関数
  - 恒等関数 ： identity
  - ハイポブリックタンジェント ： tanh
  - シグモイド関数 ： logistic
  - ReLU関数 ： relu

$$\tanh x = \frac{e^{x} - e^{-x}}{e^{x} + x^{-x}}$$

```
model = MLPClassifier(hidden_layer_sizes=128,activation='relu',max_iter=500)
```

活性化関数はReLU

```
# 学習
model.fit(train_data, train_label)

# 予測
predict = model.predict(train_data)
proba = model.predict_proba(train_data)

# 予測，正解値の表示
for i in range(len(train_data)):
   print( train_data[i][0] , train_data[i][0] , "->" , proba[i] , "(", train_label[i] , ")" )

print( "¥n [ 予測結果 ]" )
print( classification_report(train_label, predict) )

print( "¥n [ 正解率 ]" )
print( accuracy_score(train_label, predict) )

print( "¥n [ 混同行列 ]" )
print( confusion_matrix(train_label, predict) )
```

クラスごとの予測確率

正解ラベル

実行結果(三層型の場合)

0である予測確率

1である予測確率

コマンド プロンプト

```
0 0 -> [0.98339937 0.01660063]  ( 0 )
1 1 -> [0.98300134 0.01699866]  ( 0 )
1 1 -> [0.01589127 0.98410873]  ( 1 )
0 0 -> [0.01610347 0.98389653]  ( 1 )

 [ 予測結果 ]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         2

    accuracy                           1.00         4
   macro avg       1.00      1.00      1.00         4
weighted avg       1.00      1.00      1.00         4


 [ 正解率 ]
1.0

 [ 混同行列 ]
[[2 0]
 [0 2]]
```

正解ラベル

三層型の場合，学習が可能

文字認識（Digitsデータベース）

- digitsデータベース

用途	クラス分類
データ数	1797
特徴量	画素数：64（8×8） 値：0～16
目的変数	10

数字	データ数
0	178
1	182
2	177
3	183
4	181
5	182
6	181
7	179
8	174
9	180

ニューラルネットワークの構造

- 入力層
  - ニューロン数は64個
- 中間層（二層）
  - 第一中間層のニューロン数は32個
  - 第二中間層のニューロン数は32個
- 出力層
  - ニューロン数は10個

NN_digits.py

ニューラルネットワークのために
importが必要

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

# データのロード
digits = datasets.load_digits()

# 特徴量 (1797, 8, 8)
image = digits.images
total , x_size , y_size = image.shape
image = np.reshape( image , [total,x_size*y_size] )

# 目的変数
label = digits.target
```

数字画像(digits)の読み込み

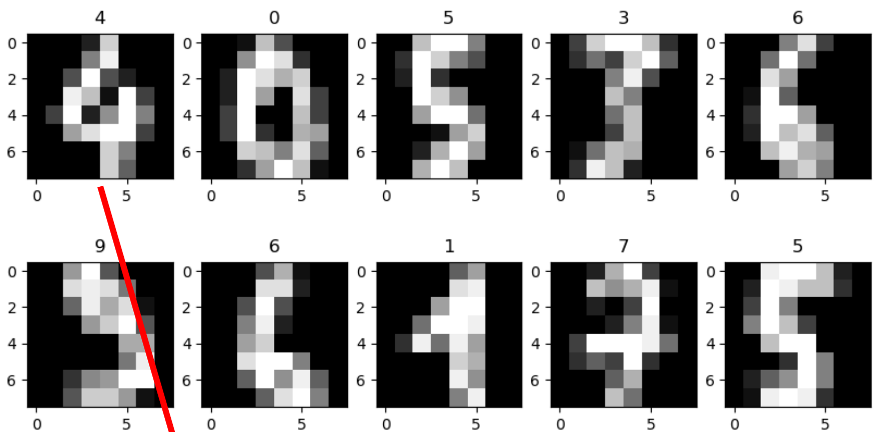
3次元→2次元に変換

total , x_size , y_size = image.shape

配列の大きさ(3次元)
(total, z_size, y_size)

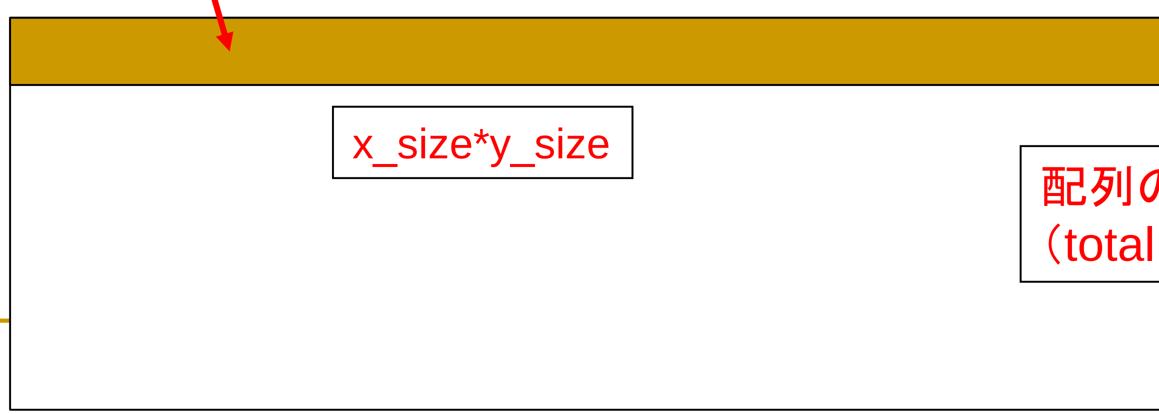
横:x_size

縦:y_size



個数:total

image = np.reshape(image , [total,x_size*y_size])



x_size*y_size

total

配列の大きさ(2次元)
(total, z_size*y_size)

```
# 学習データ，テストデータ
train_data, test_data, train_label, test_label = train_test_split(image, label,
test_size=0.5, random_state=None)

# 階層型ニューラルネットワーク
model =
MLPClassifier(hidden_layer_sizes=(32,32),activation='tanh',max_iter=500)

# 学習
model.fit(train_data, train_label)

# 予測
predict = model.predict(test_data)
proba = model.predict_proba(test_data)
```

四層型（第一中間層，第二中間層のニューロン数とも32個）

```
# 予測，正解値の表示
#for i in range(len(train_data)):
for i in range(20):
    print( "{0:3d}: ".format( i ) , end="" )
    for j in range(10):
        print( "{0:.2f}  ".format( proba[i][j] ) , end="" )
    print( "( {0} )".format( test_label[i] ) )

print( "¥n [ 予測結果 ]" )
print( classification_report(test_label, predict) )

print( "¥n [ 正解率 ]" )
print( accuracy_score(test_label, predict) )

print( "¥n [ 混同行列 ]" )
print( confusion_matrix(test_label, predict) )
```

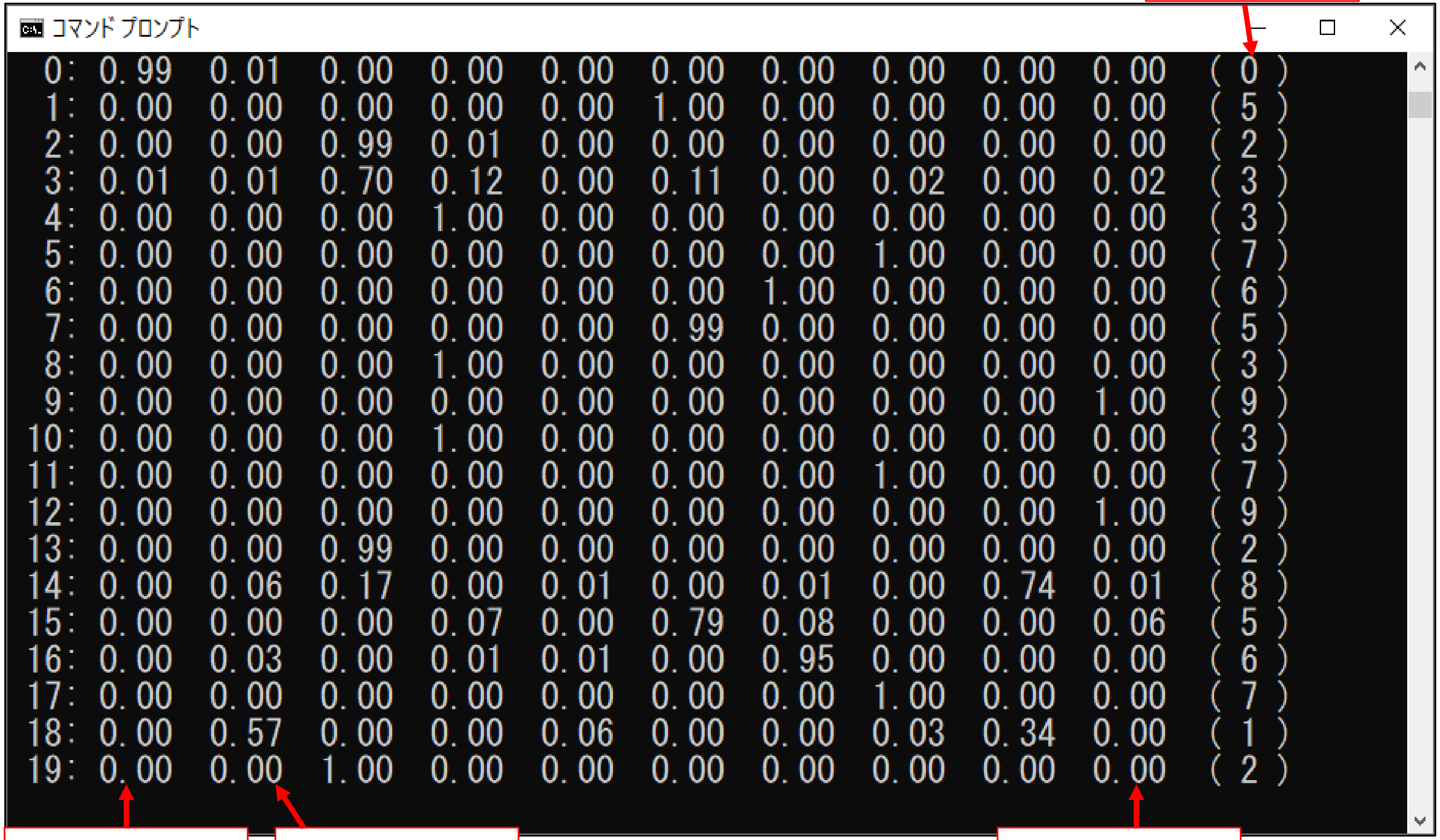
全ての出力結果を表示したい場合

予測確率

正解ラベル

実行結果

正解ラベル



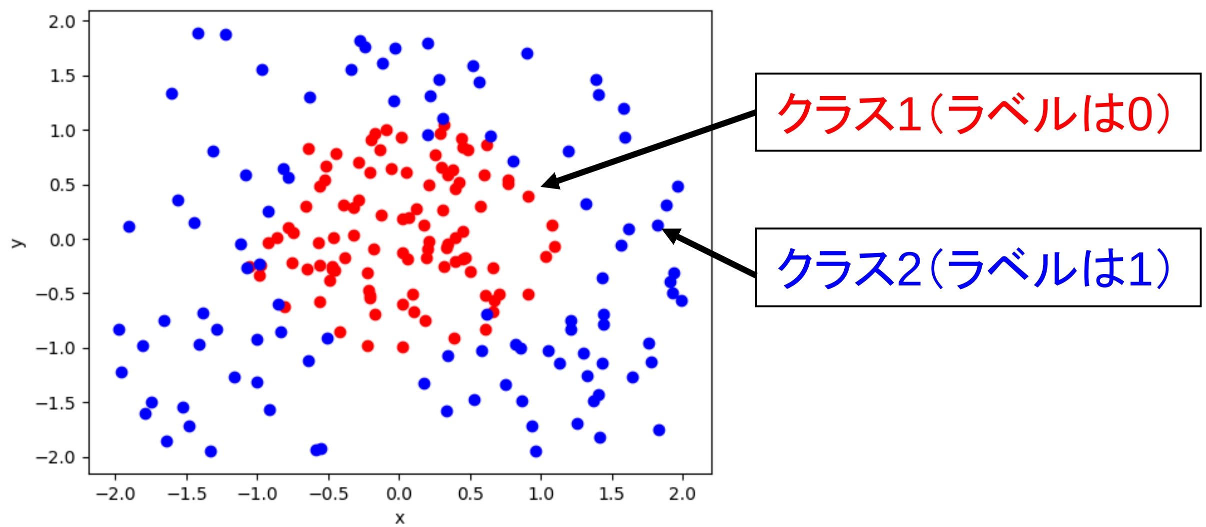
```
コマンド プロンプト
 0:  0.99  0.01  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  ( 0 )
 1:  0.00  0.00  0.00  0.00  0.00  1.00  0.00  0.00  0.00  0.00  ( 5 )
 2:  0.00  0.00  0.99  0.01  0.00  0.00  0.00  0.00  0.00  0.00  ( 2 )
 3:  0.01  0.01  0.70  0.12  0.00  0.11  0.00  0.02  0.00  0.02  ( 3 )
 4:  0.00  0.00  0.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00  ( 3 )
 5:  0.00  0.00  0.00  0.00  0.00  0.00  0.00  1.00  0.00  0.00  ( 7 )
 6:  0.00  0.00  0.00  0.00  0.00  0.00  1.00  0.00  0.00  0.00  ( 6 )
 7:  0.00  0.00  0.00  0.00  0.00  0.99  0.00  0.00  0.00  0.00  ( 5 )
 8:  0.00  0.00  0.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00  ( 3 )
 9:  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  1.00  ( 9 )
10:  0.00  0.00  0.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00  ( 3 )
11:  0.00  0.00  0.00  0.00  0.00  0.00  0.00  1.00  0.00  0.00  ( 7 )
12:  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  1.00  ( 9 )
13:  0.00  0.00  0.99  0.00  0.00  0.00  0.00  0.00  0.00  0.00  ( 2 )
14:  0.00  0.06  0.17  0.00  0.01  0.00  0.01  0.00  0.74  0.01  ( 8 )
15:  0.00  0.00  0.00  0.07  0.00  0.79  0.08  0.00  0.00  0.06  ( 5 )
16:  0.00  0.03  0.00  0.01  0.01  0.00  0.95  0.00  0.00  0.00  ( 6 )
17:  0.00  0.00  0.00  0.00  0.00  0.00  0.00  1.00  0.00  0.00  ( 7 )
18:  0.00  0.57  0.00  0.00  0.06  0.00  0.00  0.03  0.34  0.00  ( 1 )
19:  0.00  0.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  ( 2 )
```

0の予測確率　1の予測確率　9の予測確率

```
[ 予測結果 ]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        95
           1       0.90      0.99      0.94        78
           2       0.96      0.92      0.94        89
           3       0.96      0.88      0.92       105
           4       0.96      0.96      0.96        84
           5       0.94      0.95      0.94        97
           6       0.96      0.98      0.97        89
           7       0.96      0.98      0.97        92
           8       0.88      0.91      0.89        75
           9       0.95      0.93      0.94        95

    accuracy                           0.95       899
   macro avg       0.95      0.95      0.95       899
weighted avg       0.95      0.95      0.95       899


[ 正解率 ]
0.946607341490545

[ 混同行列 ]
[[94  0  0  0  0  1  0  0  0  0]
 [ 0 77  0  0  0  0  1  0  0  0]
 [ 0  3 82  2  0  0  0  0  2  0]
 [ 0  0  2 92  0  3  1  2  4  1]
 [ 0  1  0  0 81  0  0  1  1  0]
 [ 0  1  1  1  1 92  1  0  0  0]
 [ 1  0  0  0  1  0 87  0  0  0]
 [ 0  0  0  0  1  0  0 90  0  1]
 [ 0  3  0  0  0  0  1  0 68  3]
 [ 0  1  0  1  0  2  0  1  2 88]]
```

練習問題

- 二値分類（SVCの練習問題）
  - NN-exercise.py を実行して下さい
  - 下図（data.png）のように，クラス1（赤点），クラス2（青点）のデータ（二次元データ）を100個ずつ生成します．



練習問題

- SVCの練習問題と同様に，階層型ニューラルネットワークを用いてクラス分類して下さい．
  - 中間層の層数，各層のニューロン数，活性化関数をいろいろと変えてみて下さい

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）
- Richard O. Duda他：パターン識別，アドコム・メディア（2009）
- 平井有三：はじめてのパターン認識，森北出版（2013）
- 岡谷貴之：深層学習，講談社（2015）

参考文献

- MLPClassifier
  - https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html