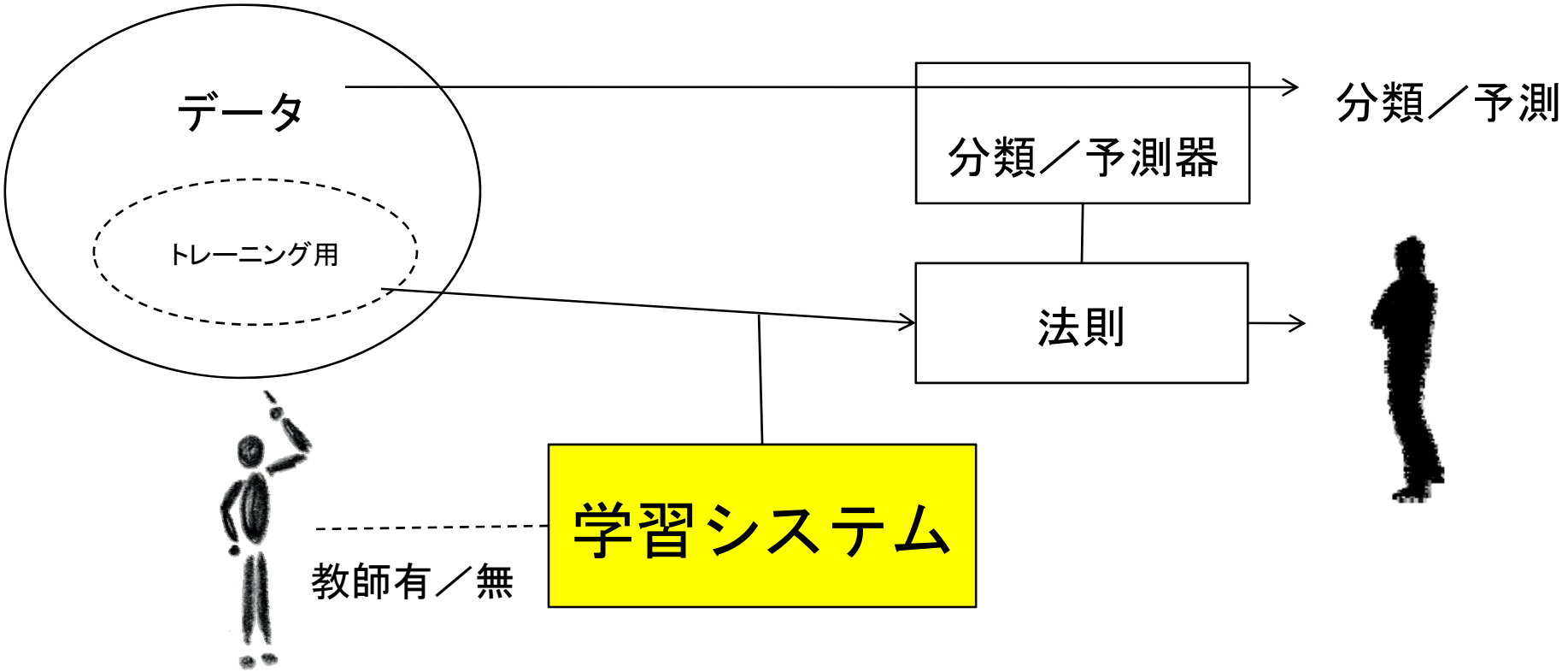
2013年6月14日

データマイニングと機械学習

西田豊明

機械学習とデータマイニング

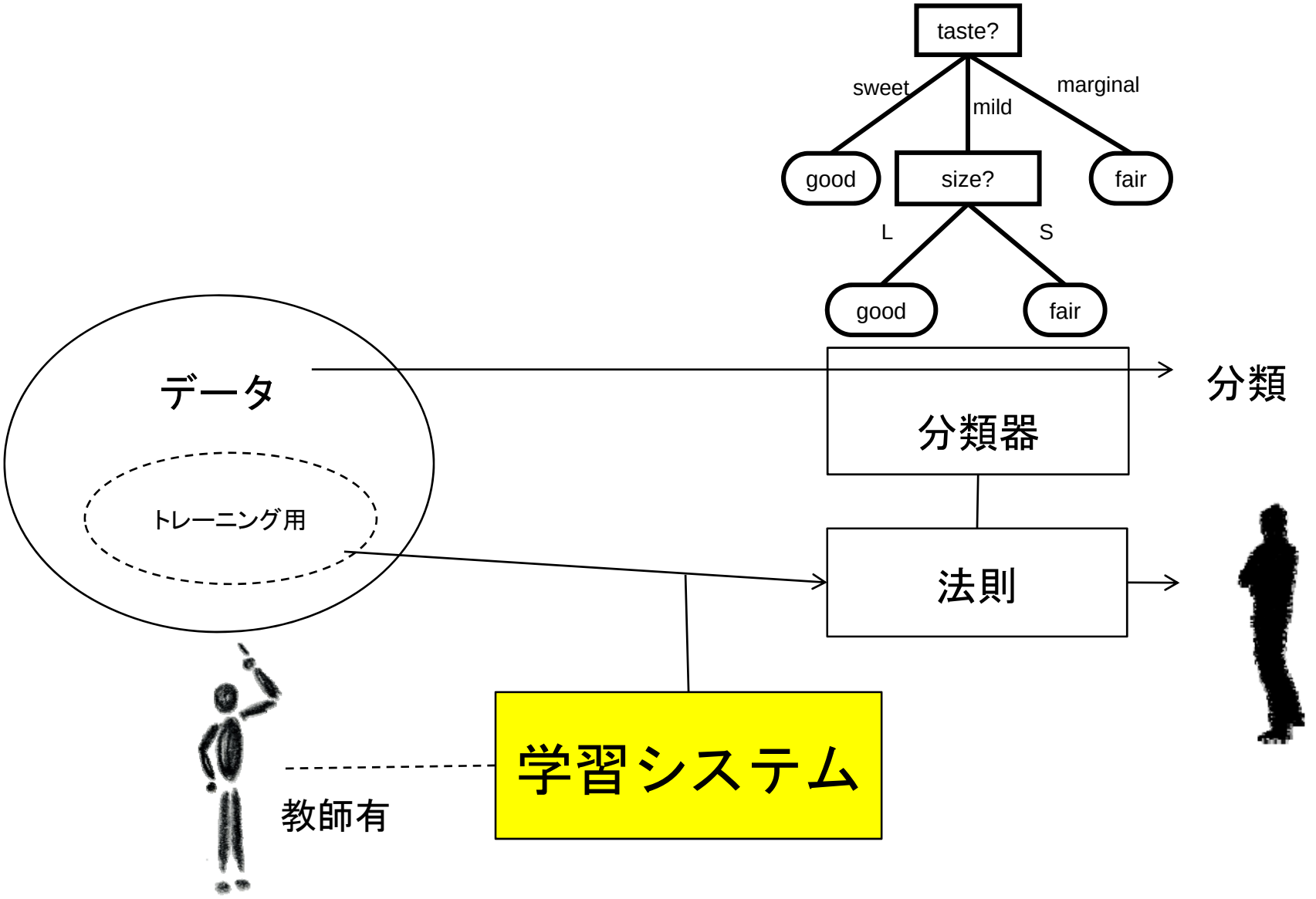
- 機械学習
  AIシステムが経験を通して自らの振舞いや知識を改良するための手法
- データマイニング:
  データの中から法則性を推定する技術.



[西田 1999]

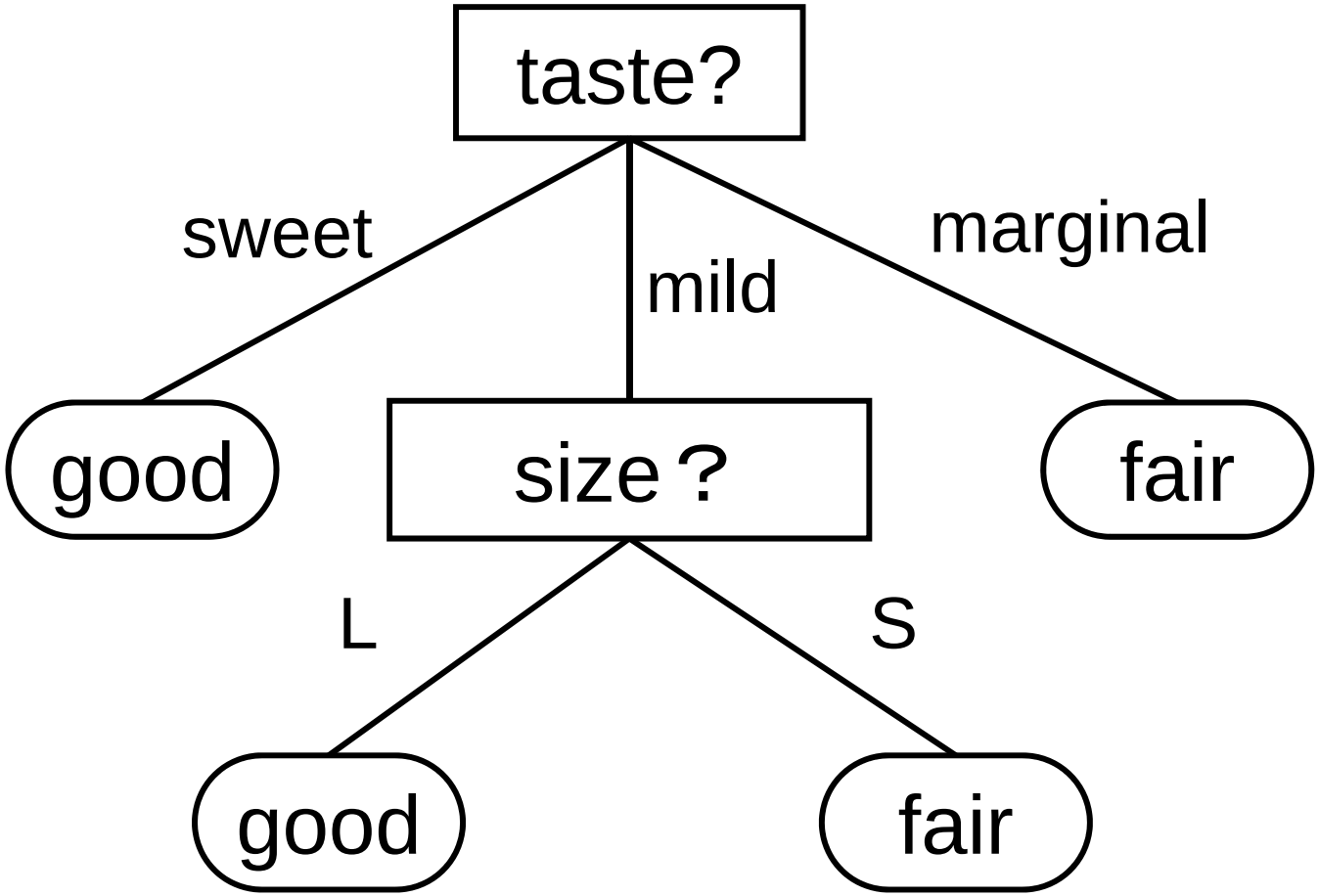
決定木学習

決定木


taste?
sweet
mild
marginal
good
size?
fair
L
S
good
fair
データ
トレーニング用
分類器
分類
法則
学習システム
教師有

[西田 1999]

決定木



[西田 1999]

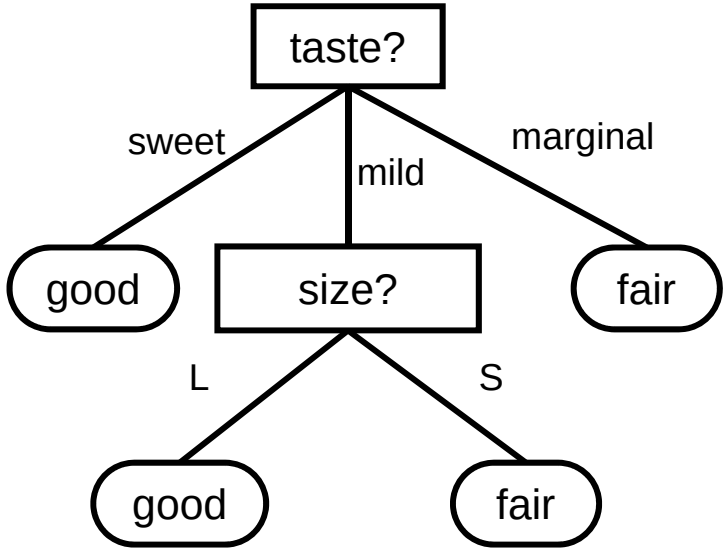
決定木の学習 -- 例題

taste	size	surface	decision
sweet	L	shining	good
sweet	S	shining	good
sweet	S	dull	good
mild	L	shining	good
mild	S	shining	fair
marginal	L	dull	fair
marginal	S	dull	fair
marginal	S	shining	fair

[西田 1999]

決定木の例

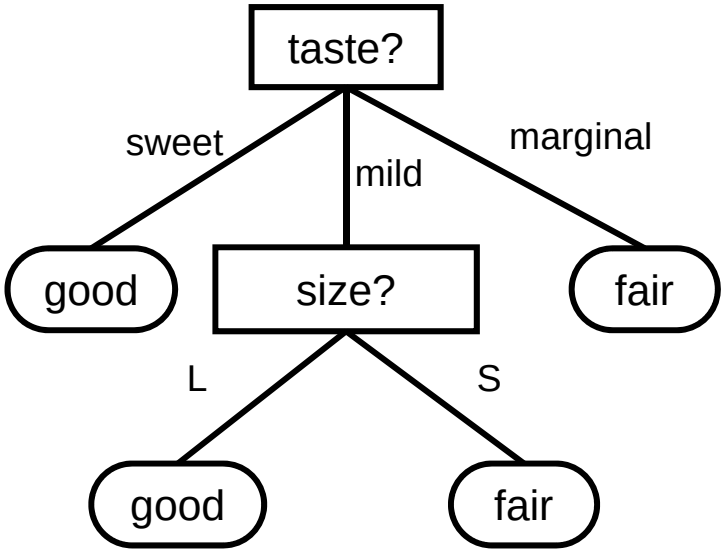
taste	size	surface	decision
sweet	L	shining	good
sweet	S	shining	good
sweet	S	dull	good
mild	L	shining	good
mild	S	shining	fair
marginal	L	dull	fair
marginal	S	dull	fair
marginal	S	shining	fair

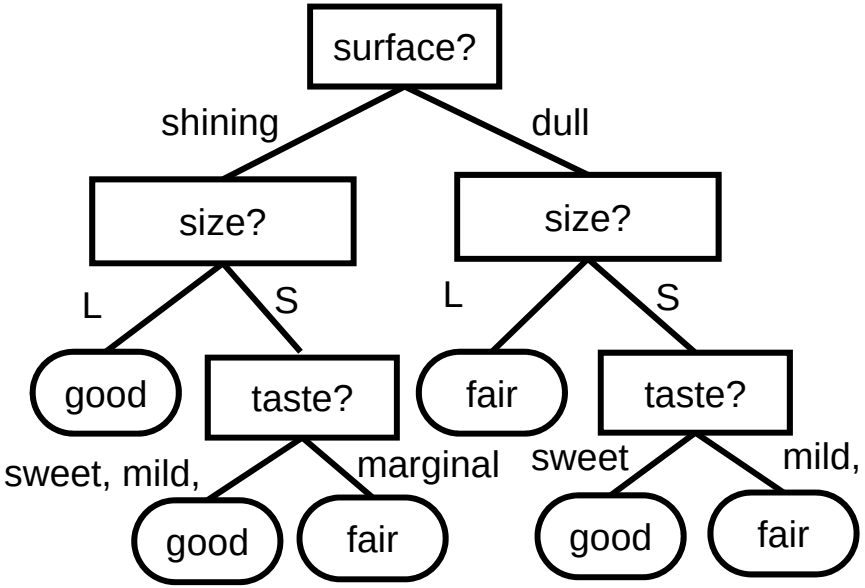


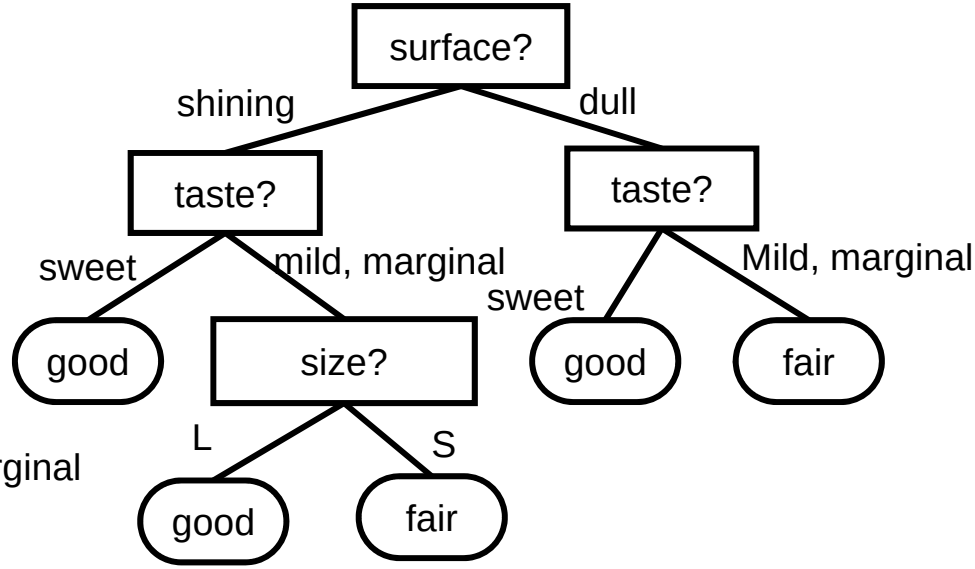
[西田 1999]

決定木の例

taste	size	surface	decision
sweet	L	shining	good
sweet	S	shining	good
sweet	S	dull	good
mild	L	shining	good
mild	S	shining	fair
marginal	L	dull	fair
marginal	S	dull	fair
marginal	S	shining	fair







[西田 1999]

決定木の良さ

構造の簡単な決定木を優先

オッカムのかみそり(Occum‘s razor):
与えられたデータと整合する仮説のなかで最も単純なものを優先する.

データ集合Sのエントロピー$H(S)$:

$$H(S) = -\sum_{c \in C(S)} p(c) \log p(c)$$

$C(S)$: データ集合 S に対する可能なカテゴリの集合
$p(c)$: カテゴリ c に分類されるデータの割合

[西田 1999]

アルゴリズムID3

```
(defun id3 (attributes data)
  (let* ((possible-classes (get-possible-classes data)) children)
     (cond ((= (length possible-classes) 1) (first possible-classes))
           ((null attributes) '?)
           (t (setq children (select-and-split attributes data))
              `(,(first children)
                . ,(mapcar #'(lambda (x)
                               `(,(first x)
                                 ,(id3 (delete (first children) attributes)
                                       (second x))))
                           (second children)))))))

(defun get-possible-classes (data)
   (let* ((result nil))
     (dolist (d data)
        (cond ((not (member (second d) result))
                    (push (second d) result))))
     result))
(defun select-and-split (attributes data)
   (select-minimal-entropy (find-all-partitions attributes data)))

(defun find-all-partitions (attributes data)
   (let* ((result nil) possible-values entropy-attribute-partition)
      (dolist (attribute attributes)
         (setq possible-values (get-possible-values attribute data))
         (setq entropy-attribute-partition (get-partition data attribute possible-values))
         (push entropy-attribute-partition result))
      result))
```

[西田 1999]

アルゴリズムID3

```
(defun get-possible-values (attribute data)
   (let* ((possible-values nil) x)
      (dolist (d data)
         (setq x (second (assoc attribute (cdr (first d)))))
         (cond ((not (member x possible-values)) (push x possible-values))))
       possible-values))

(defun get-partition (data attribute possible-values)
  (let* ((entropy 0) (partition nil) (data-count (length data)) subdata)
     (dolist (possible-value possible-values)
        (setq subdata nil)
        (dolist (d data)
           (cond ((equal possible-value (second (assoc attribute (cdr (first d)))))
                         (push d subdata))))
        (setq entropy (+ entropy (* (/ (length subdata) data-count)
                                    (calculate-entropy subdata))))
        (push `(,possible-value ,subdata) partition))
     `(,entropy ,attribute ,partition)))

(defun calculate-entropy (data)
  (let* ((size (length data)) (entropy 0) (possible-classes (get-possible-classes data)) c)
     (cond ((= (length possible-classes) 1))
           ((= size 1))
           (t (dolist (possible-class possible-classes)
                  (setq c (count-if #'(lambda (x) (equal possible-class (second x))) data))
                  (setq entropy (- entropy (* (/ c size) (log (/ c size) 2)))))))
     entropy))

(defun select-minimal-entropy (partitions)
   (cdr (first (sort partitions #'(lambda (x y) (< (first x) (first y)))))))
```

[西田 1999]

頻出集合発見

頻出集合発見

トランザクションデータベース

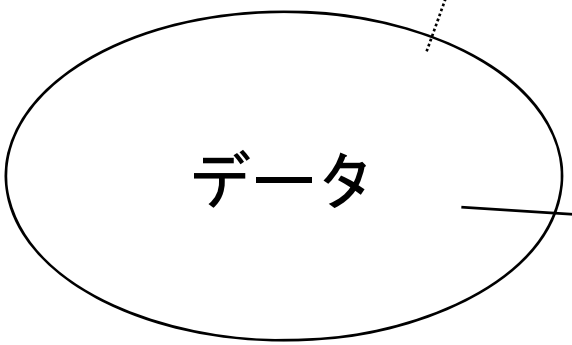
トランザクション名（取引）	レコード：アイテムの集合
A	1,2,5,6,7,9
B	2,3,4,5
C	1,2,7,8,9
D	1,7,9
E	2,3,7,9
F	2,7,9

トランザクション

アイテム

3つ以上のトランザクションに含まれるアイテム集合

{1} {2} {7} {9}
{1, 7} {1, 9}
{2, 7} {2, 9} {7, 9}
{1, 7, 9} {2, 7, 9}



頻出集合

頻出集合発見

[西田 1999]

頻出集合発見問題 (frequent itemset mining)

Given：トランザクションデータベース D

例： t1: {1, 2, 5, 8, 11, 12, 15, 17, 18, 19, 23, 25, 27}
t2: {2, 4, 5, 8, 11, 17, 19, 23, 27, 29}
t3: {1, 3, 5, 6, 7, 8, 9, 10, 14, 15, 21, 25, 26, 30}
t4: {7, 8, 10, 11, 12, 13, 14, 15, 17, 23, 25, 29}
t5: {2, 5, 6, 8, 11, 13, 15, 17, 18, 20, 23, 25, 30}
t6: {2, 3, 7, 8, 12, 13, 15, 17, 18, 23, 26, 29}

Find：頻出度 θ 以上の頻出集合を全て見つける

例（θ=5の場合）： 大きさ1: {{8}, {15}, {17}, {23}}
大きさ2: {{17, 23}, {8 ,23}, {8, 17}, {8 ,15}}
大きさ3: {{17, 8, 23}}

θ：閾値（最小サポート）
θ が大：解となる頻出集合の数は小さい
θ が小：有用な知識をもらさず見つけ出すため，モデルの立場からは望ましい．

[西田 1999]

アプリオリ法

基本アイデア

頻出集合の族の**単調性**：トランザクション t が頻出集合 X を含むならば，t は X の任意の部分集合を含む.

$\Rightarrow$　X の部分集合の出現集合は X の出現集合を含み，頻出度は X より同じか大きくなる.

[西田 1999]

アルゴリズム

アプリオリ法のアルゴリズム
空集合から出発し，幅優先的にアイテムを1つずつ追加していく.

アプリオリ

D_1=大きさ1の頻出集合の集合; k:=1
while $D_k \neq \varphi$
(1) 大きさがk+1であり，D_kの2つのアイテム集合の和集合となっているものを全てD_{k+1} に挿入する
(2) frq(S)<θ となるSを全てD_{k+1}から取り除く
(3) k := k+1
end while

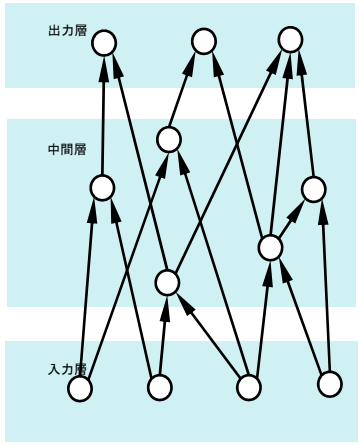
特長：データベースへのアクセス数の少なさ.
計算時間:
〈候補の総数〉×〈データベースの大きさ〉に比例
〈候補の総数〉は頻出アイテム集合の数のn倍(nはアイテムの数)を超えない.
⇒ 頻出集合1つあたり，最悪でも〈データベースの大きさ〉×nに比例

[西田 1999]

ニューラルネットワーク

― 多層フィードフォーワードネットワーク ―

多層フィードフォーワードネットワーク



データ

トレーニング用

分類器

分類

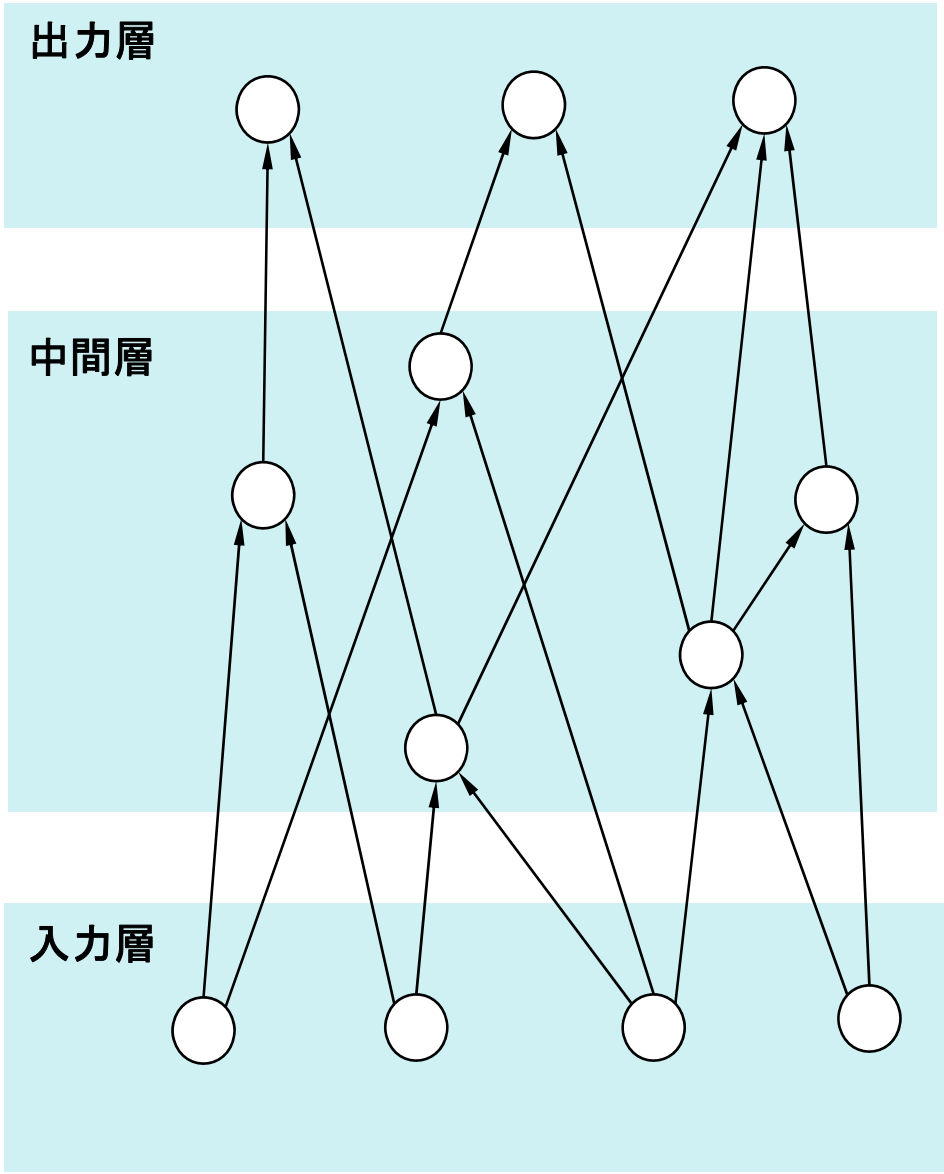
法則

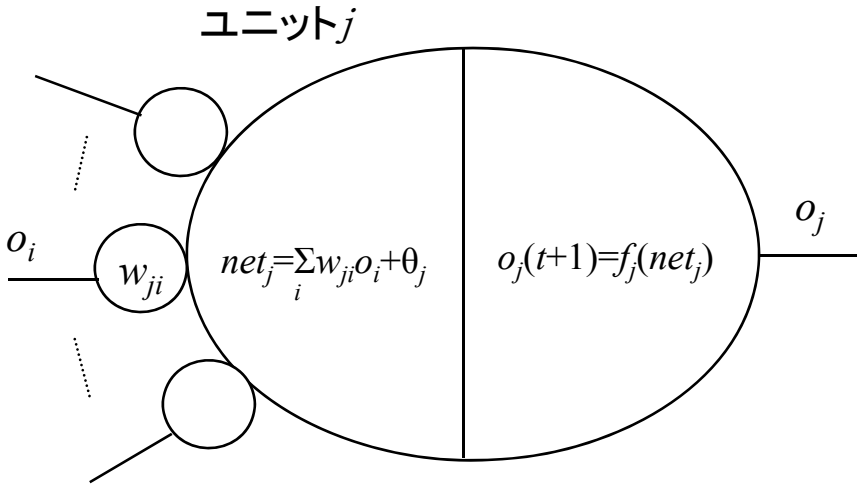
学習システム

教師有

[西田 1999]

フィードフォーワード型ニューラルネットワーク



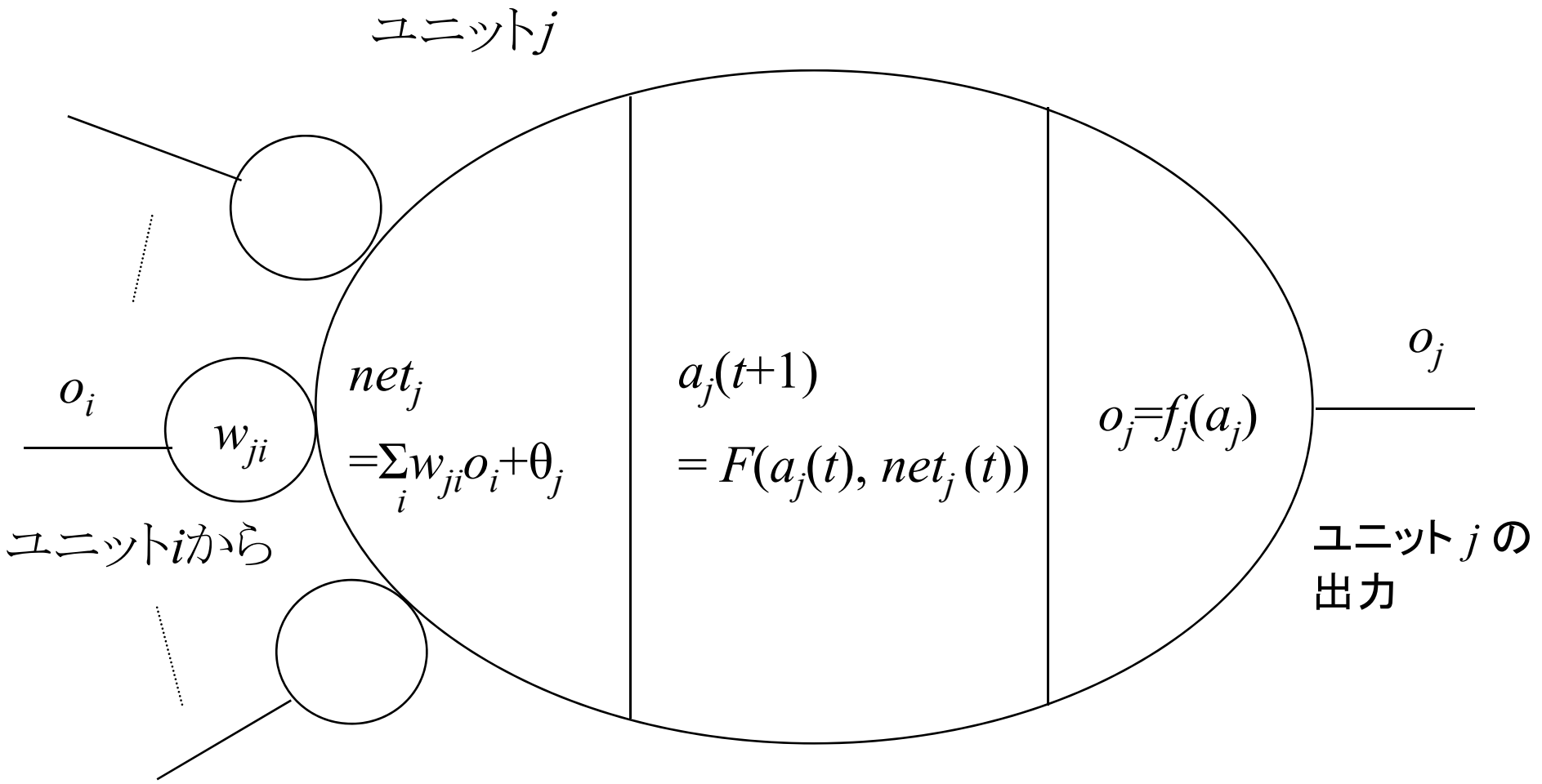


f_j ：単調非減少，微分可能

θ_j：バイアス

[西田 1999]

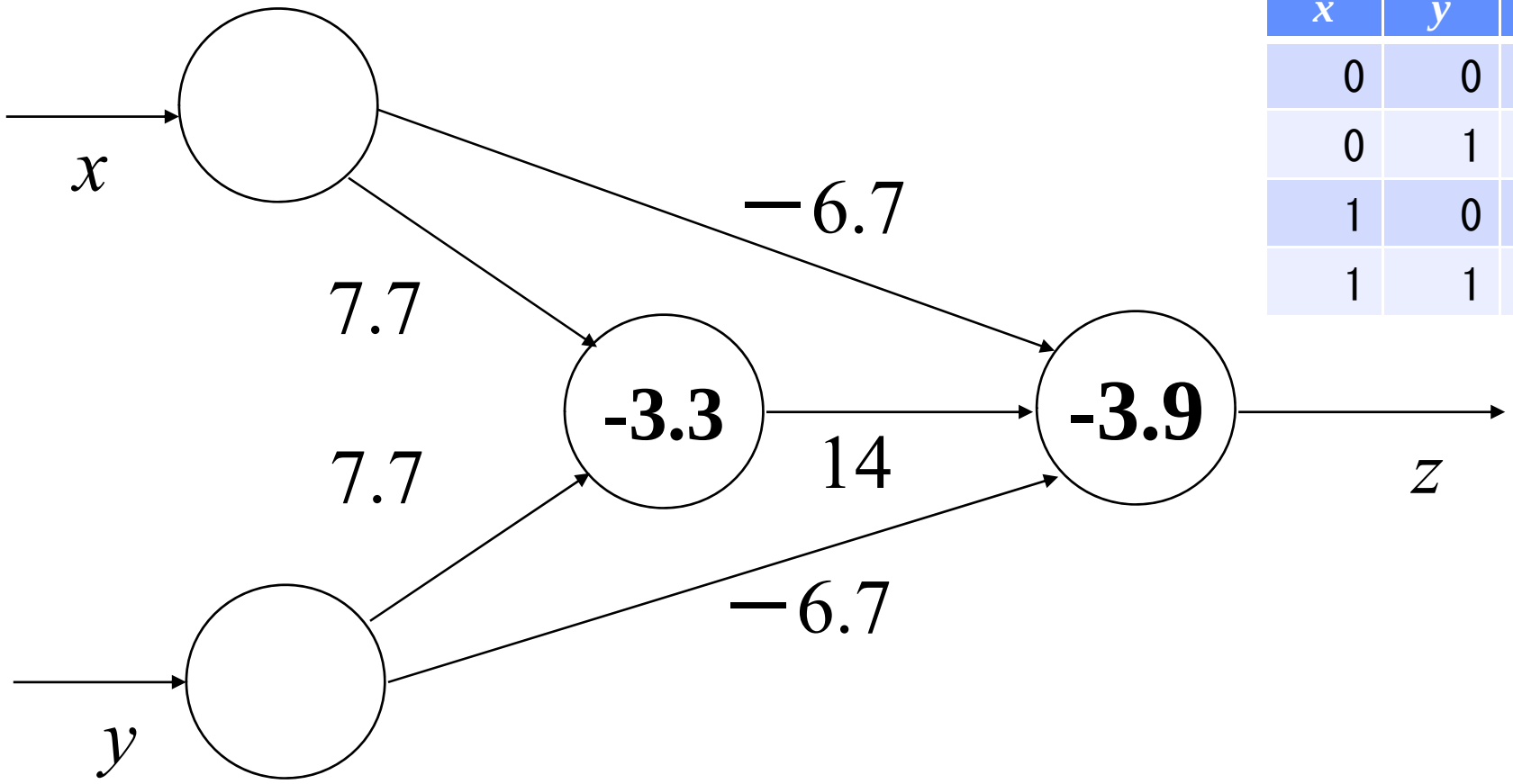
ニューラルネットワークを構成するユニット（より一般的なもの）



a_j：ユニット j の活性度

[西田 1999]

排他的選言(exclusive OR)を計算するためのフィードフォーワード型ネットワーク



x	y	z
0	0	0.0
0	1	1.0
1	0	1.0
1	1	0.0

入力ユニット　　中間ユニット　　出力ユニット

(c) は，そのユニットへのバイアスが c であることを示す

[西田 1999]

誤差逆伝播

入力パターン p に対する出力層での誤差

$$E_p = \sum_j E_{pj} = \sum_j \frac{1}{2}(t_{pj} - o_{pj})^2$$

が最小になるように，ユニットmからユニットnへの結合に対応づけられている重みw_{nm}各ユニットnのバイアス θ_nを調節する．E_{pj}は入力パターンpに対する出力ユニットjにおける誤差(希望出力t_{pj}と実際の出力o_{pj}との差の2乗の半分)である．

pに対するw_{nm}への修正： $\Delta_p w_{nm} = -\eta \frac{\partial E_p}{\partial w_{nm}}$

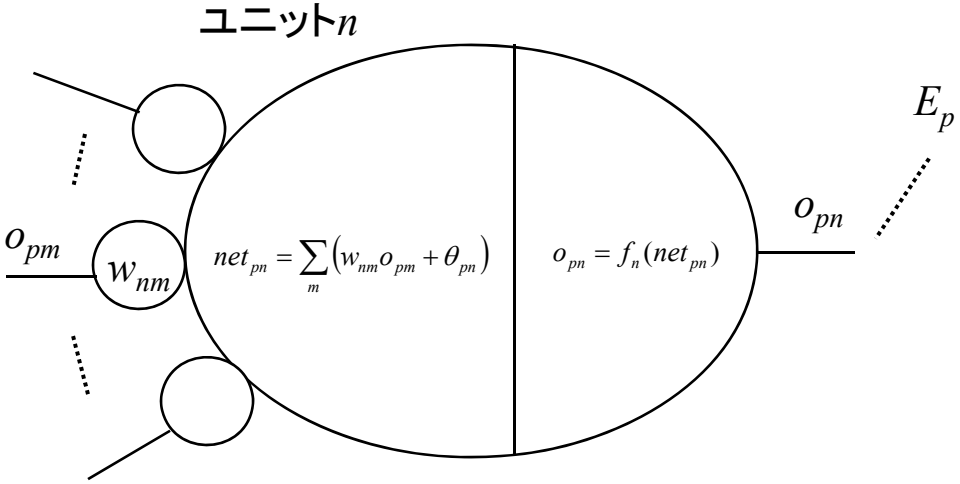
pに対するθ_nへの修正： $\Delta_p \theta_n = -\eta \frac{\partial E_p}{\partial \theta_n}$

[西田 1999]

誤差逆伝播の基本的な考え方

$$net_{pn} = \sum_m \left(w_{nm} o_{pm} + \theta_{pn} \right)$$

$$o_{pn} = f_n(net_{pn})$$



に注意すると

$$\Delta_p w_{nm} = -\eta \frac{\partial E_p}{\partial w_{nm}}$$

$$= -\eta \frac{\partial E_p}{\partial net_{pn}} \frac{\partial net_{pn}}{\partial w_{nm}} = -\eta \frac{\partial E_p}{\partial net_{pn}} o_{pm} = \eta \delta_{pn} o_{pm}$$

$$\Delta_p \theta_n = -\eta \frac{\partial E_p}{\partial \theta_{pn}}$$

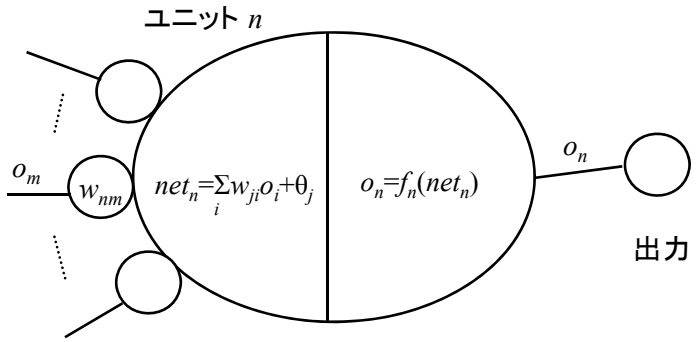
$$= -\eta \frac{\partial E_p}{\partial net_{pn}} \frac{\partial net_{pn}}{\partial \theta_{pn}} = -\eta \frac{\partial E_p}{\partial net_{pn}} = \eta \delta_{pn}$$

ただし, $\frac{\partial E_p}{\partial net_{pn}} = -\delta_{pn}$

[西田 1999]

誤差逆伝播の基本的な考え方

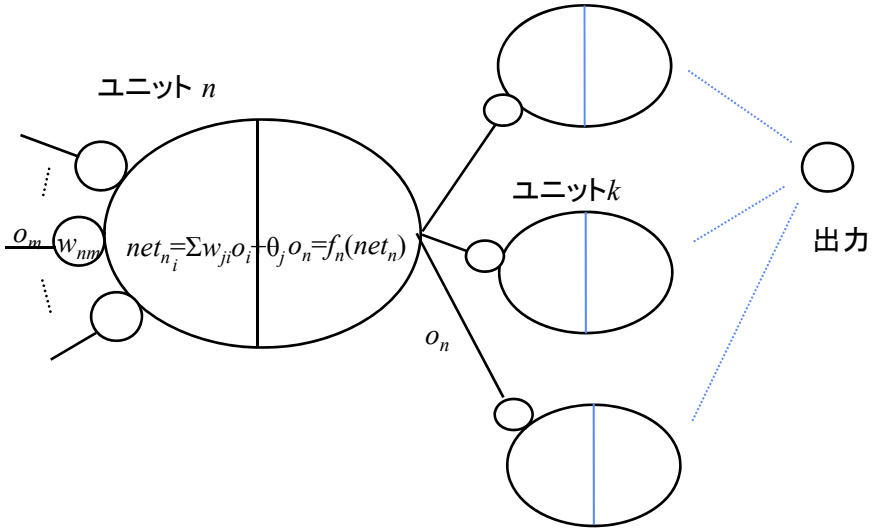
ユニット n が出力ユニットのとき



$$\delta_{pn} = -\frac{\partial E_p}{\partial net_{pn}} = -\frac{\partial}{\partial net_{pn}}\left(\sum_j \frac{1}{2}\left(t_{pj}-o_{pj}\right)^2\right)$$

$$= \left(t_{pn}-o_{pn}\right)\frac{\partial o_{pn}}{\partial net_{pn}} = \left(t_{pn}-o_{pn}\right)f_n'(net_{pn})$$

ユニット n が隠れユニットのとき



$$\delta_{pn} = -\frac{\partial E_p}{\partial net_{pn}}$$

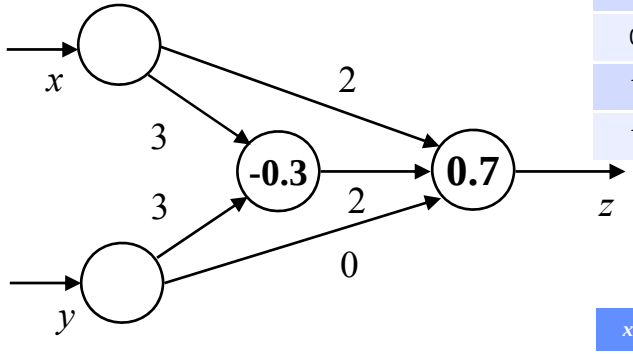
$$= -\frac{\partial E_p}{\partial o_{pn}}\frac{\partial o_{pn}}{\partial net_{pn}}$$

$$= -\sum_k \frac{\partial E_p}{\partial net_{pk}}\frac{\partial net_{pk}}{\partial o_{pn}} f_n'(net_{pn})$$

$$= \sum_k \delta_{pk} w_{kn} f_n'(net_{pn})$$

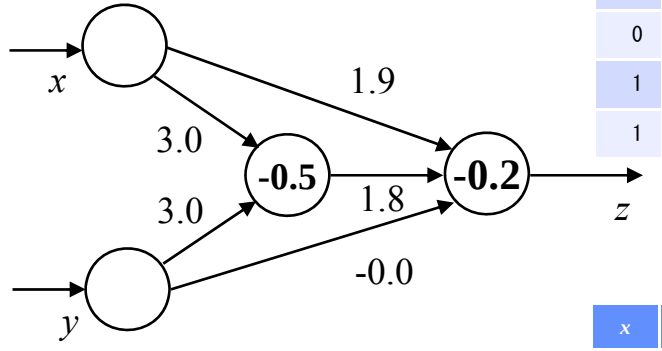
[西田 1999]

実行例



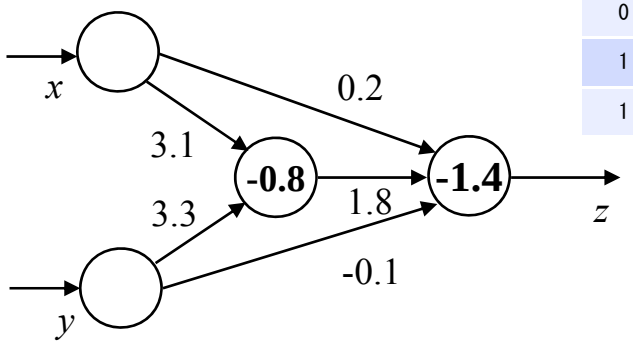
x	y	z
0	0	0.74
0	1	0.89
1	0	0.98
1	1	0.99

10サイクル後



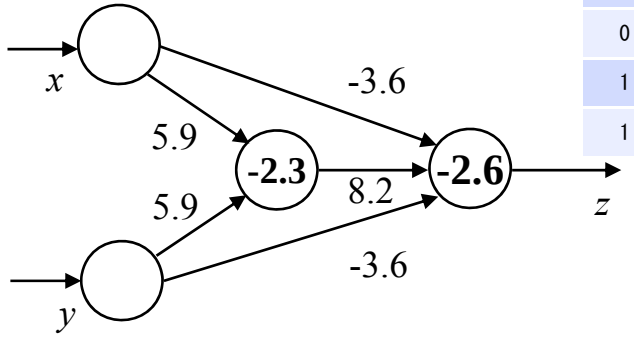
x	y	z
0	0	0.61
0	1	0.81
1	0	0.97
1	1	0.97

100サイクル後



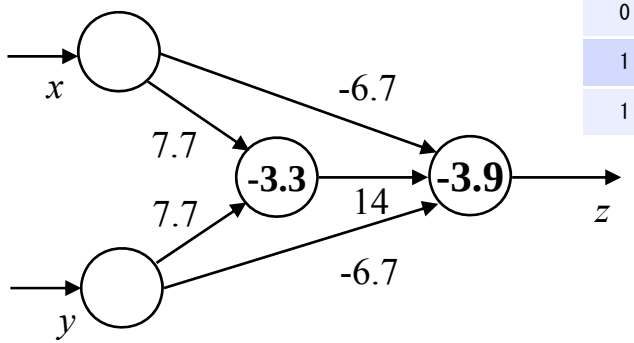
x	y	z
0	0	0.29
0	1	0.52
1	0	0.60
1	1	0.61

1000サイクル後



x	y	z
0	0	0.13
0	1	0.85
1	0	0.85
1	1	0.17

10000サイクル後



x	y	z
0	0	0.00
0	1	0.97
1	0	0.97
1	1	0.00

[西田 1999]