パターン認識と学習
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)
  - これまでの研究
  - ニューロンのモデル化
  - 階層型ニューラルネットワーク
    - パーセプトロン
    - 線形ニューラルネットワーク

  - 実習(線形ニューラルネットワーク)

これまでのニューラルネットワークの研究

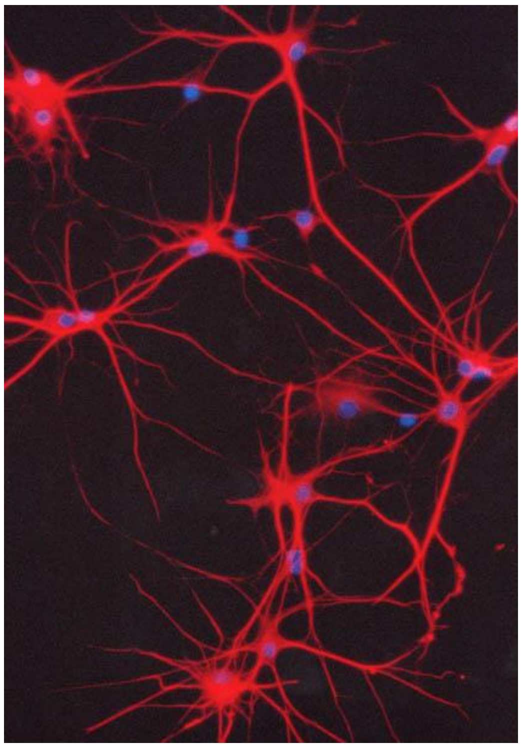
現在のブームのきっかけ(の一つ)

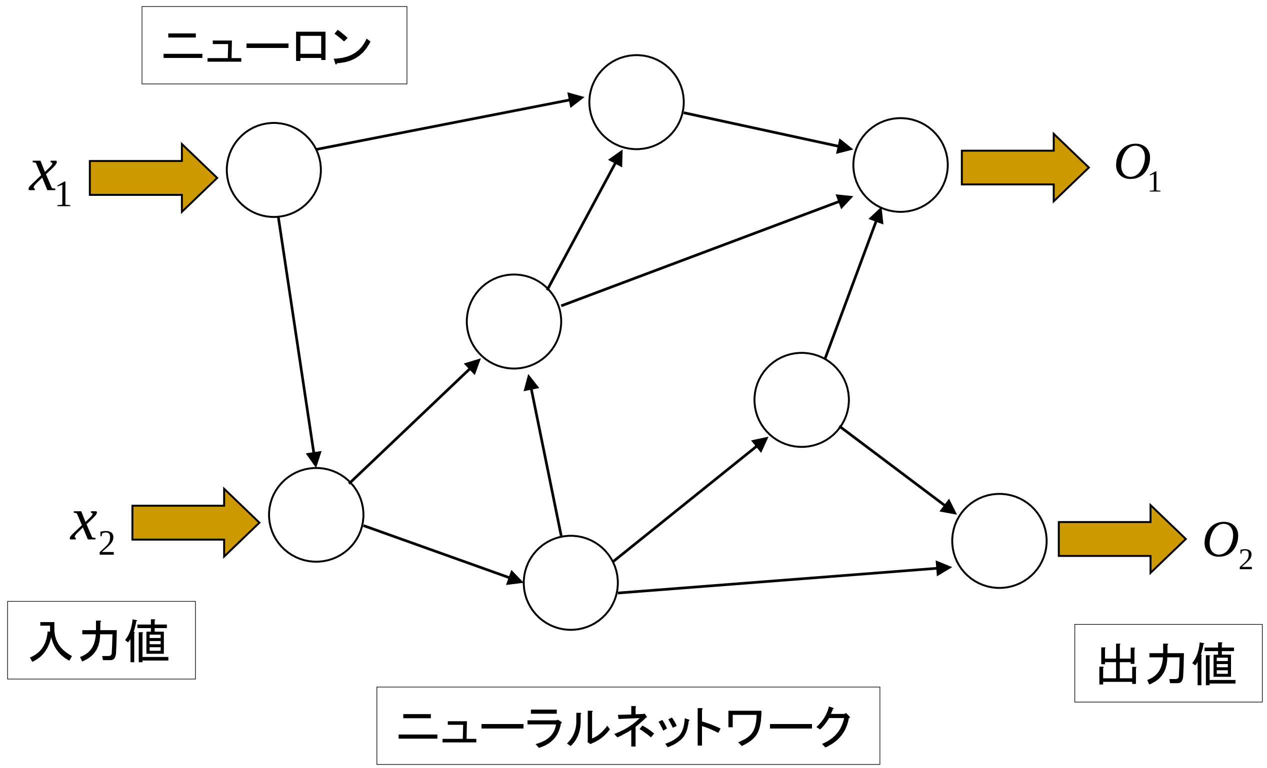
Googleの猫(2012)

図:Q.V.Le, Building high-level features using large scale unsupervised learning, 2012

ニューラルネットワーク

- 神経細胞（ニューロン）のネットワークモデル





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

ニューラルネットワークをとりまく研究環境

ニューロサイエンス

人間の脳，神経系の機能の解明

人間の脳，神経系の機能のモデル化

人間の脳，神経系の機能のモデルを応用化

ニューロコンピューティング

ニューラルネットワークの注目されている機能①

学習データ（入力値）

海

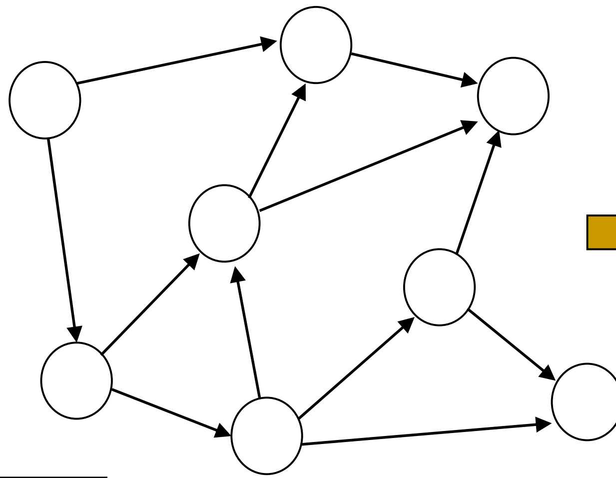
山

●
●
●

川

正解ラベル（教師信号）

ニューラルネットワーク



学習

「出力値＝正解ラベル」となるように
ネットワークのパラメータを修正

出力値

海

山

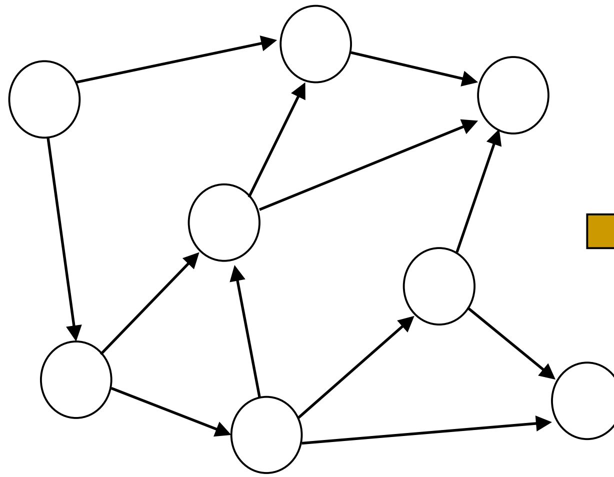
●
●
●

川

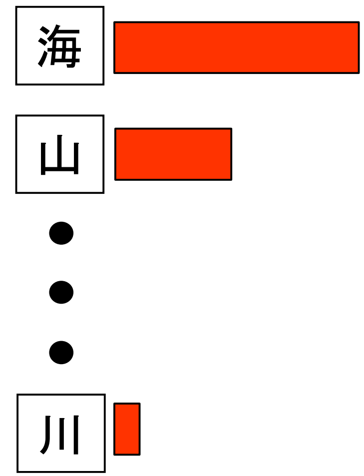
ニューラルネットワークの注目されている機能②

未知データ

学習後のニューラルネットワーク



予測



識別モデル

$f(\omega,x|\theta)$
$=p(\omega|\mathbf{x})$

ニューラルネットワークの注目されている機能③

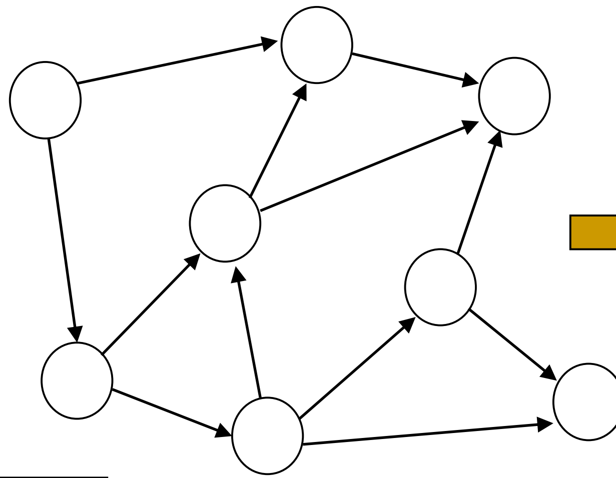
学習データ(入力値)

山

p(**x**,山)

解析

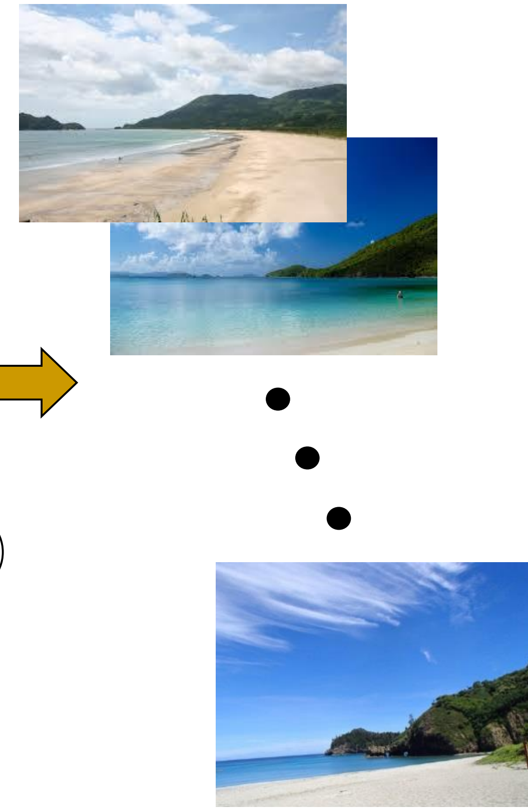
ニューラルネットワーク



学習

p(**x**,山|θ)

生成モデル



現在のニューラルネットワークへの期待

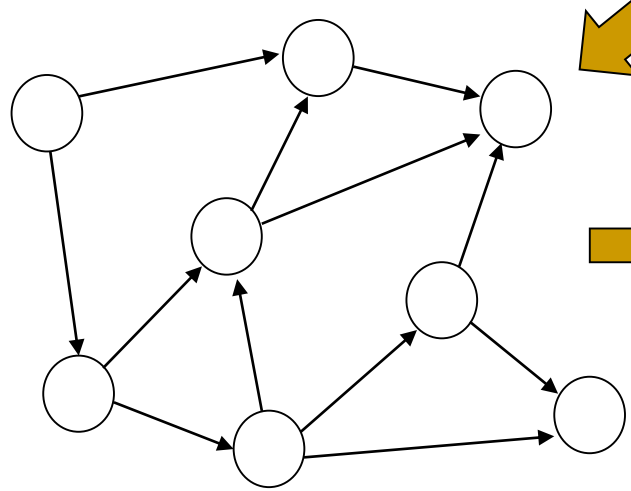
インターネット

学習データ（入力値）

・
・
・

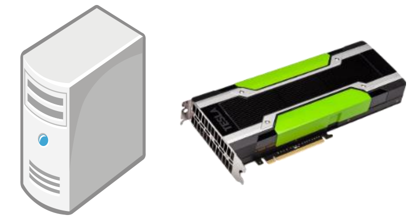
データの大規模化

ニューラルネットワーク



ニューラルネットワークの構造の大規模化

コンピュータの急速な進歩



予測値

精度の飛躍的な向上

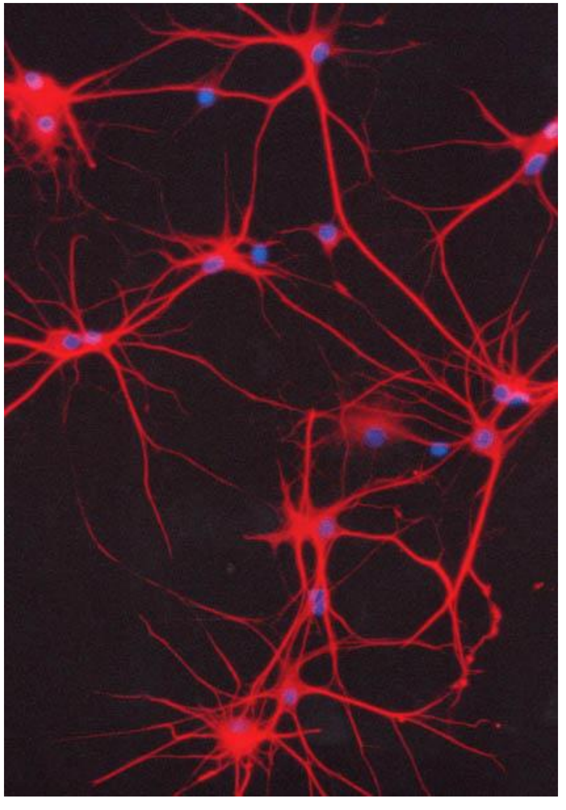
深層学習（deep learning）

ニューラルネットワークの研究①

- 1943 McCulloch-Pittsモデル(W.S.McCulloch, W.Pitts)
- 1949 Hebbの学習則(D. Hebb)
- 1952 Hodgkin-Huxleyモデル(A.L.Hodgkin, A.F.Huxley)
- 1958 パーセプトロン(F.Rosenblatt)
- 1960 デルタールール(B.Widrow, M.E. Hoff)
- 1969 M.Minskyらによるパーセプトロンの限界の指摘
- 1979 ネオコグニトロン(福島邦彦)
- 1982 ホップフィールドネットワーク(J.J.Hopfield)
- 1982 自己組織化マップ(T.Kohonen)
- 1985 ボルツマンマシン(D.H.Ackley)

神経細胞

- 神経細胞（ニューロン）



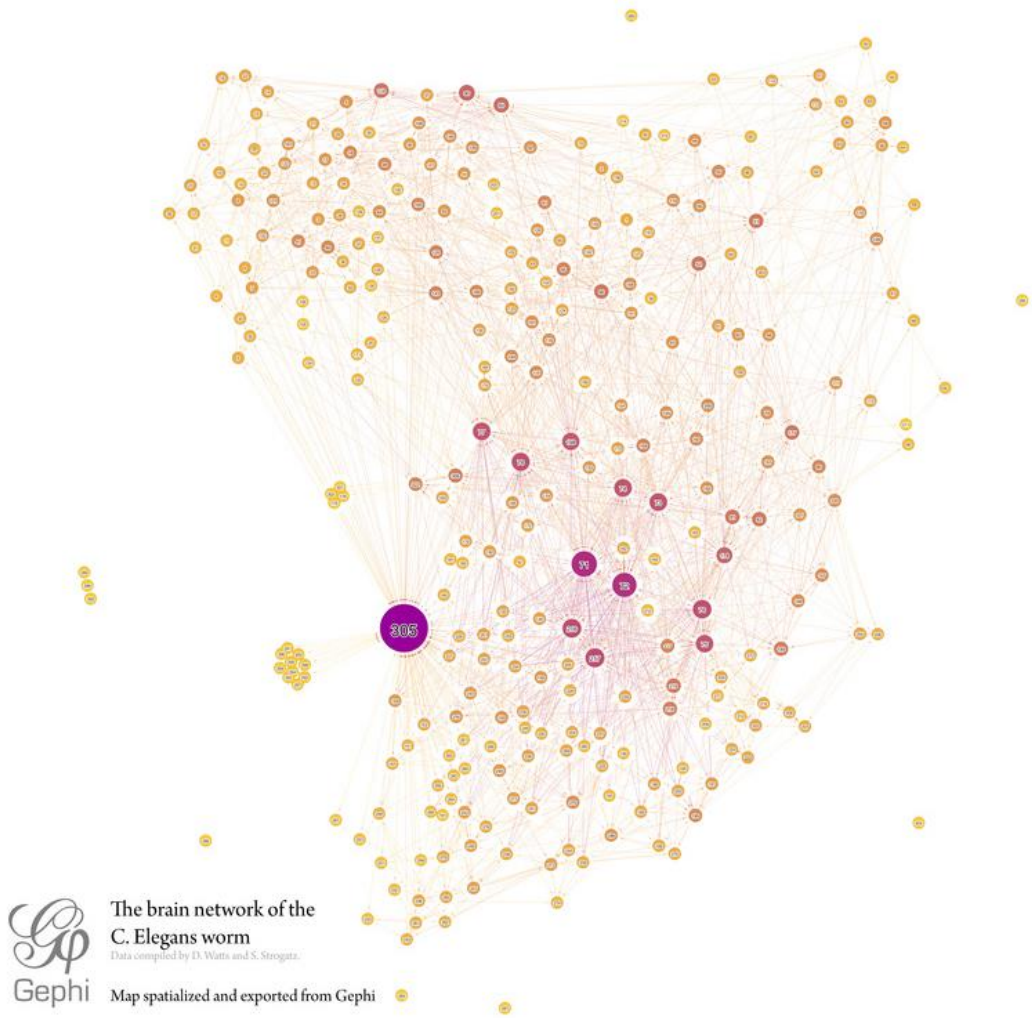
細胞体の大きさ
10マイクロメートル程度

人間
ニューロン　800億個以上
シナプス　　1.5×10^{14}個

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

神経細胞（イメージ図）

コネクトーム（C.elegans）

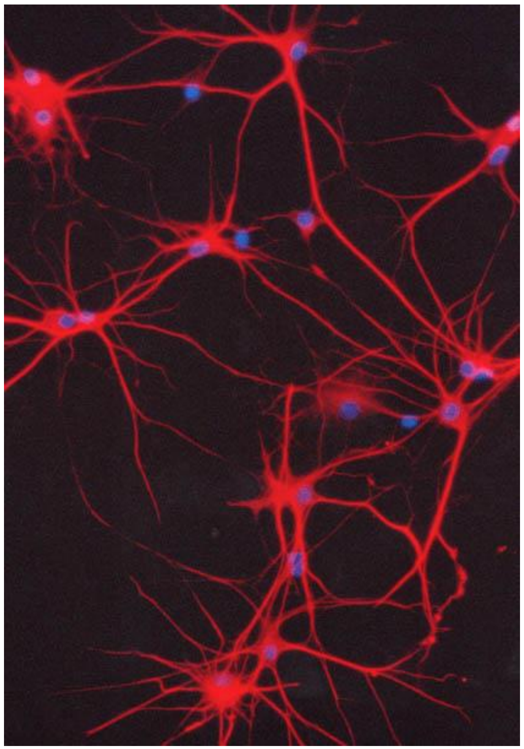


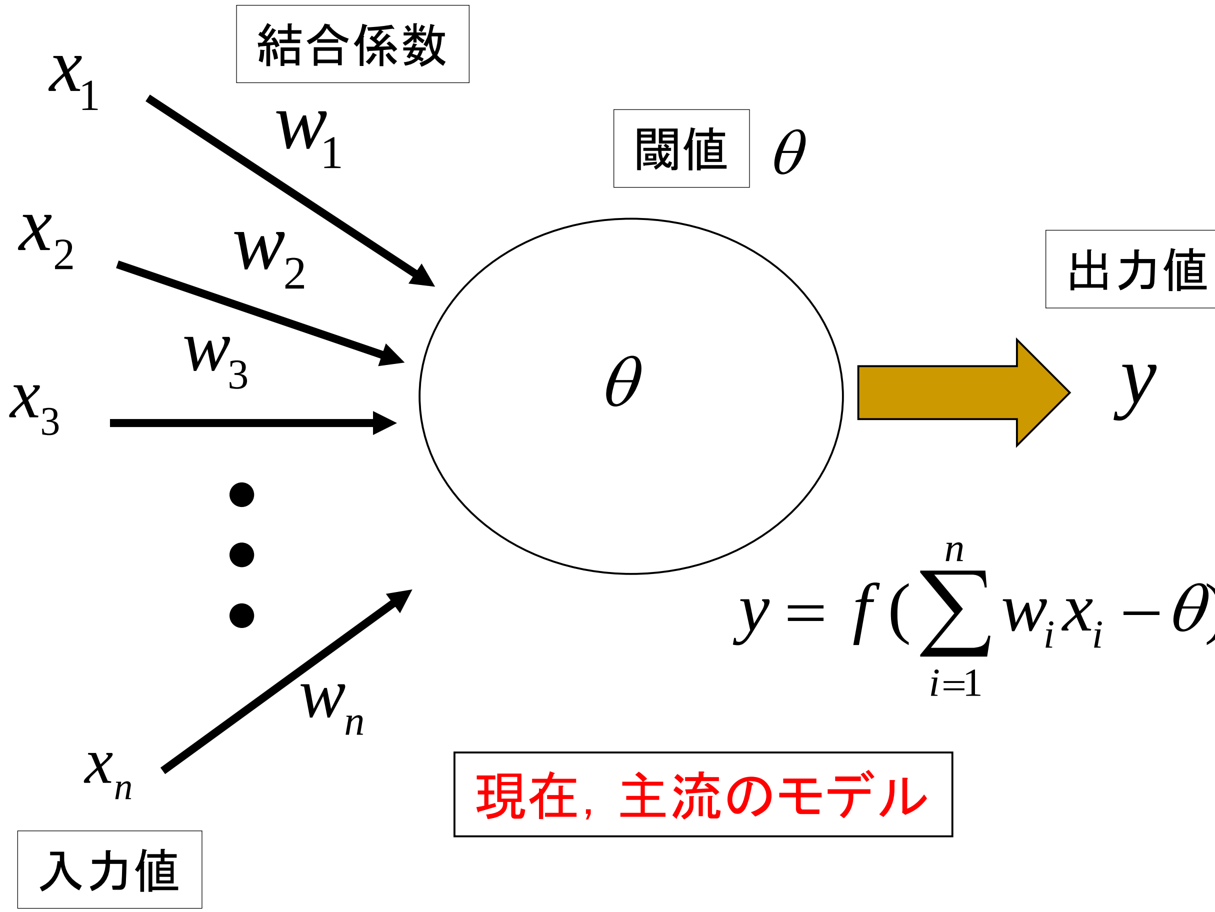
C.elegans

センチュウ
長さ:1mm
ニューロン数:302個

McCulloch-Pittsモデル（1943）

- 神経細胞（ニューロン）のモデル化



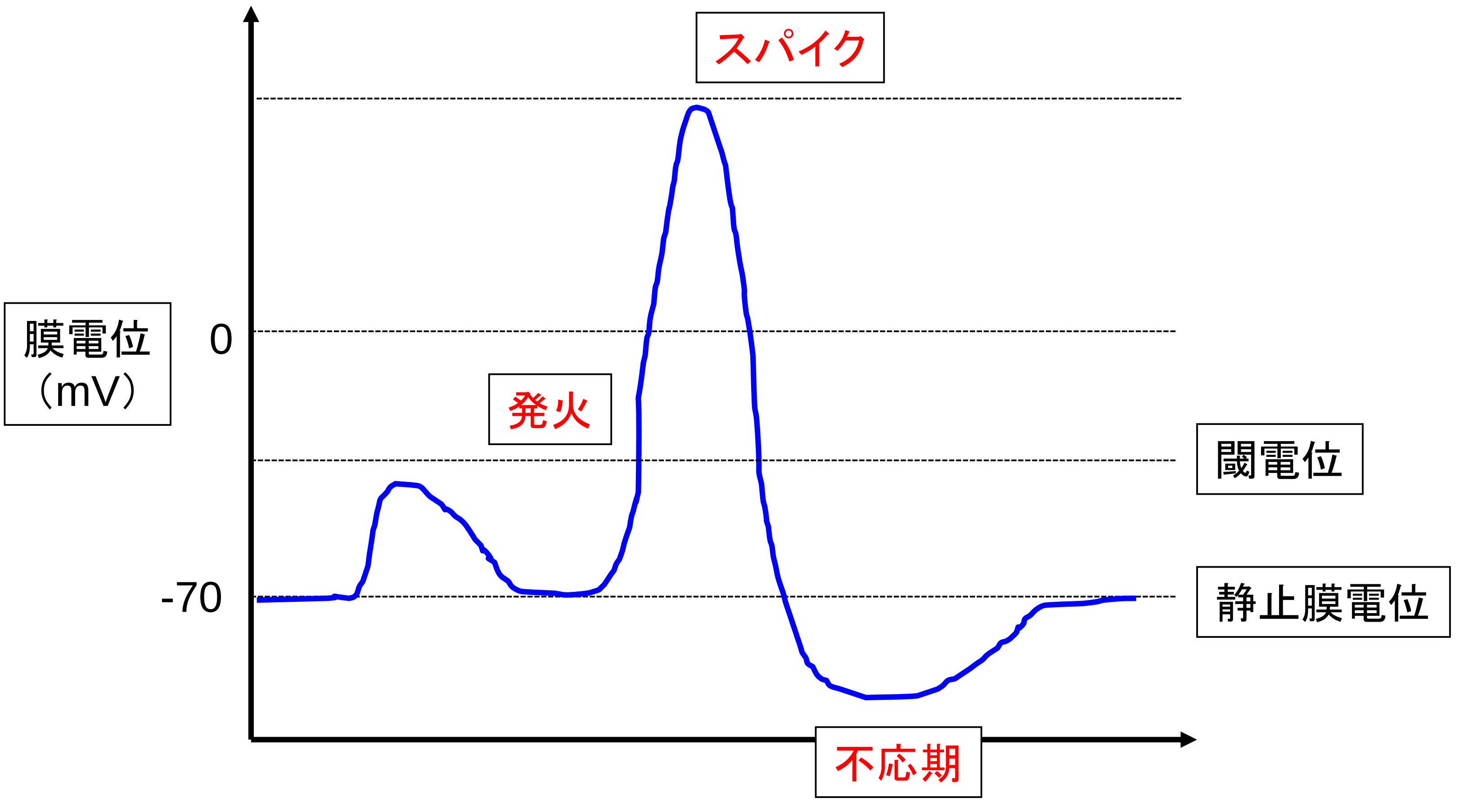


$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

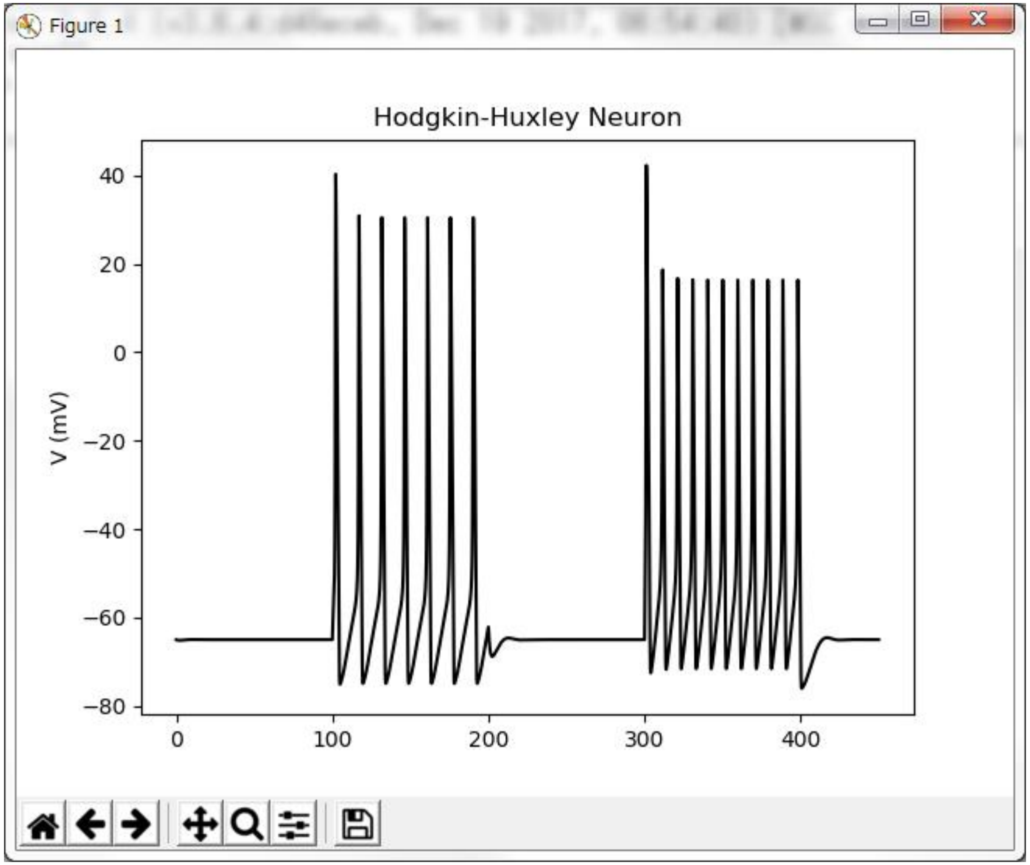
現在，主流のモデル

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

神経細胞の活動



Hodgkin-Huxleyモデルによるシミュレーション



ニューラルネットワークの研究①

- 1943　McCulloch-Pittsモデル（W.S.McCulloch，W.Pitts）
- 1949　Hebbの学習則（D. Hebb）
- 1950　チューリングテスト（A.M.Turing）
- 1952　Hodgkin-Huxleyモデル（A.L.Hodgkin, A.F.Huxley）
- 1956　ダートマス会議
- 1958　パーセプトロン（F.Rosenblatt）
- 1960　デルタールール（B.Widrow，M.E. Hoff）
- 1969　M.Minskyらによるパーセプトロンの欠点
- 1979　ネオコグニトロン（福島邦彦）
- 1982　ホップフィールドネットワーク（J.J.Hopfield）
- 1982　自己組織化マップ（T.Kohonen）
- 1985　ボルツマンマシン（D.H.Ackley）

ニューラルネットワークの研究①

- 1943　McCulloch-Pittsモデル（W.S.McCulloch, W.Pitts）
- 1949　Hebbの学習則（D. Hebb）
- 1952　Hodgkin-Huxleyモデル（A.L.Hodgkin, A.F.Huxley）
- 1958　パーセプトロン（F.Rosenblatt）
- 第一次ニューラルネットワークブーム
- 1960　デルタールール（B.Widrow, M.E. Hoff）
- 1969　M.Minskyらによるパーセプトロンの限界の指摘
- 1979　ネオコグニトロン（福島邦彦）
- 1982　ホップフィールドネットワーク（J.J.Hopfield）
- 1982　自己組織化マップ（T.Kohonen）
- 1985　ボルツマンマシン（D.H.Ackley）

パーセプトロン（F.Rosenblatt,1958）

- 階層型ニューラルネットワーク
  - 線形識別関数の学習アルゴリズム

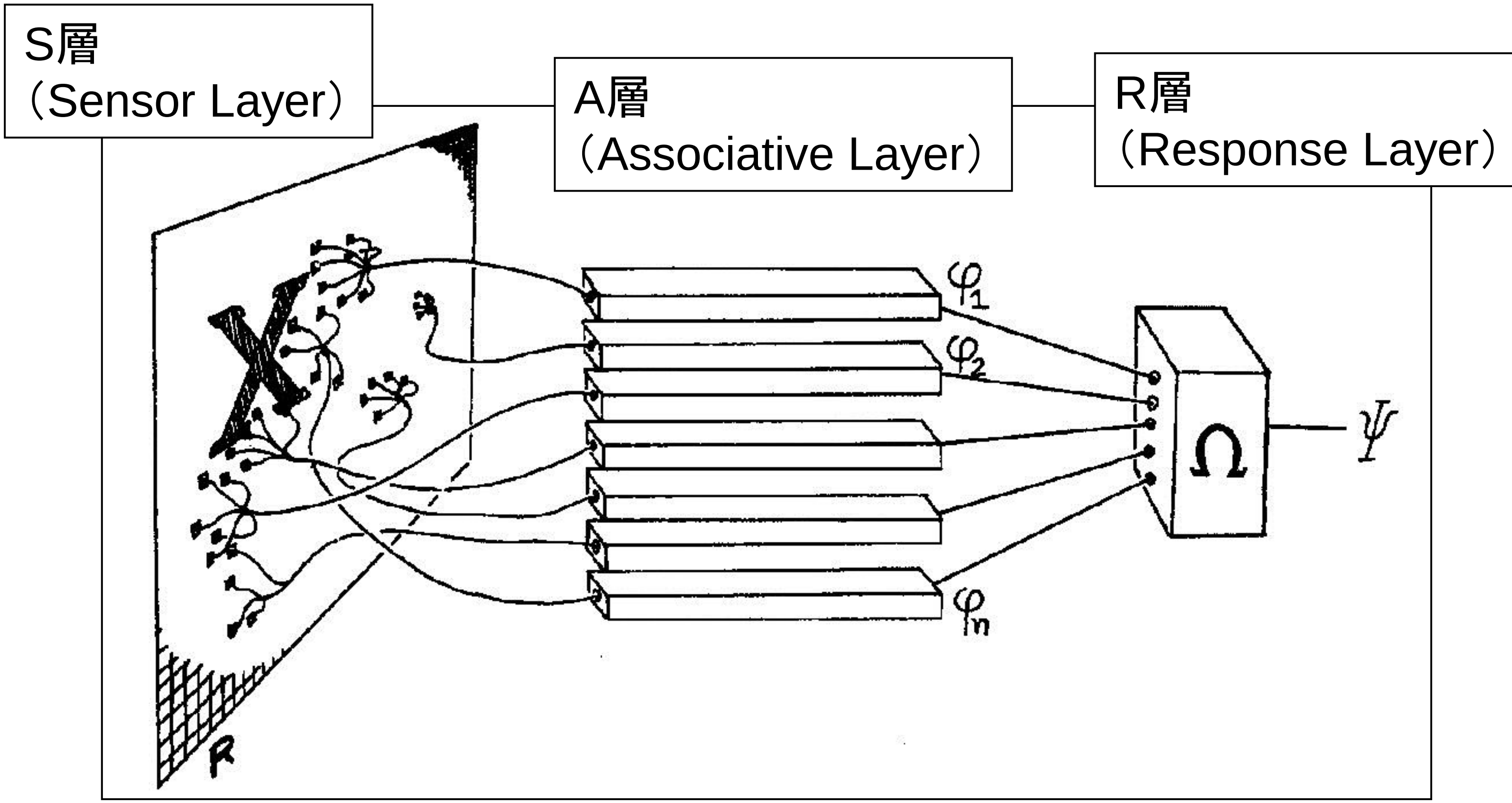


図:（中野馨，阪口豊訳）M.Minsky，S.A.Papert, Perceptrons，パーソナルメディア, 1993

ニューラルネットワークの研究①

- 1943 McCulloch-Pittsモデル（W.S.McCulloch, W.Pitts）
- 1949 Hebbの学習則（D. Hebb）
- 1952 Hodgkin-Huxleyモデル（A.L.Hodgkin, A.F.Huxley）
- 1958 パーセプトロン（F.Rosenblatt）
- 1960 デルタールール（B.Widrow, M.E. Hoff）
- 1969 M.Minskyらによるパーセプトロンの限界の指摘
- 第一次ニューラルネットワークブームが終わる
- 1979 ネオコグニトロン（福島邦彦）
- 1982 ホップフィールドネットワーク（J.J.Hopfield）
- 1982 自己組織化マップ（T.Kohonen）
- 1985 ボルツマンマシン（D.H.Ackley）

ネオコグニトロン(福島邦彦,1979)

- 単純型細胞(S細胞), 複雑型細胞(C細胞)による階層型ニューラルネットワーク

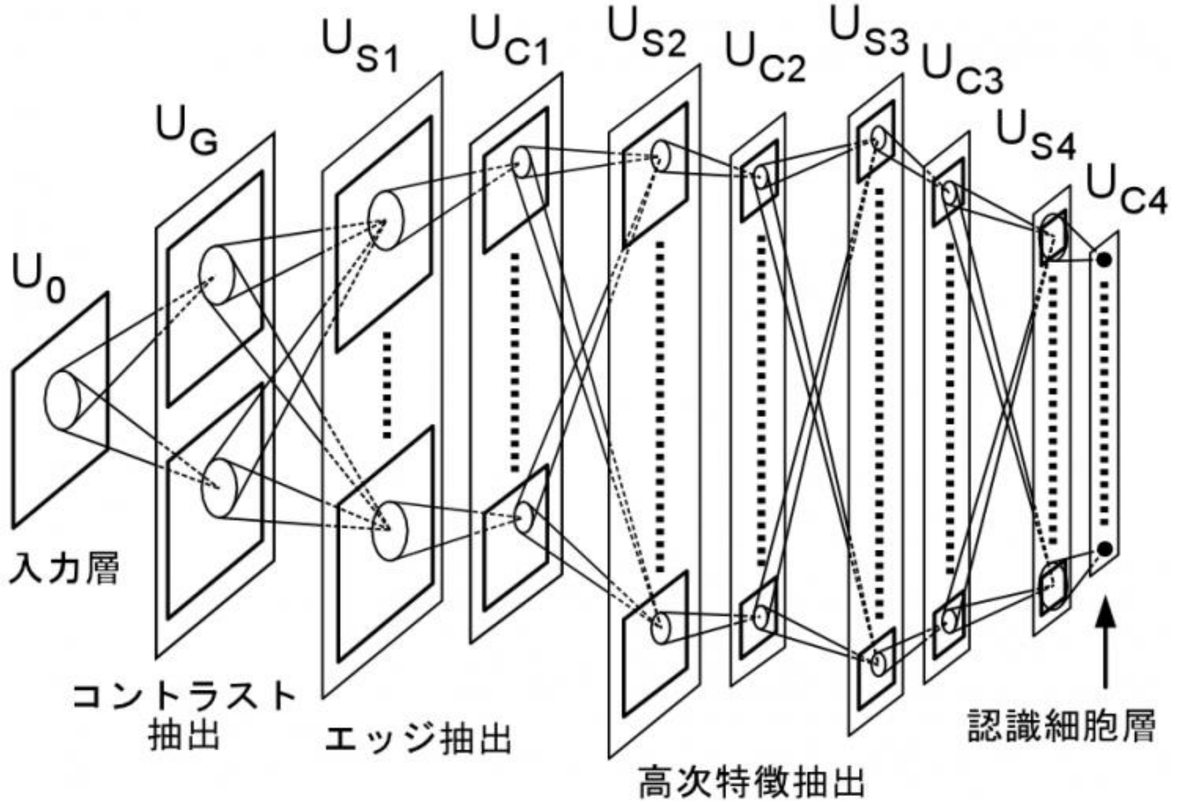
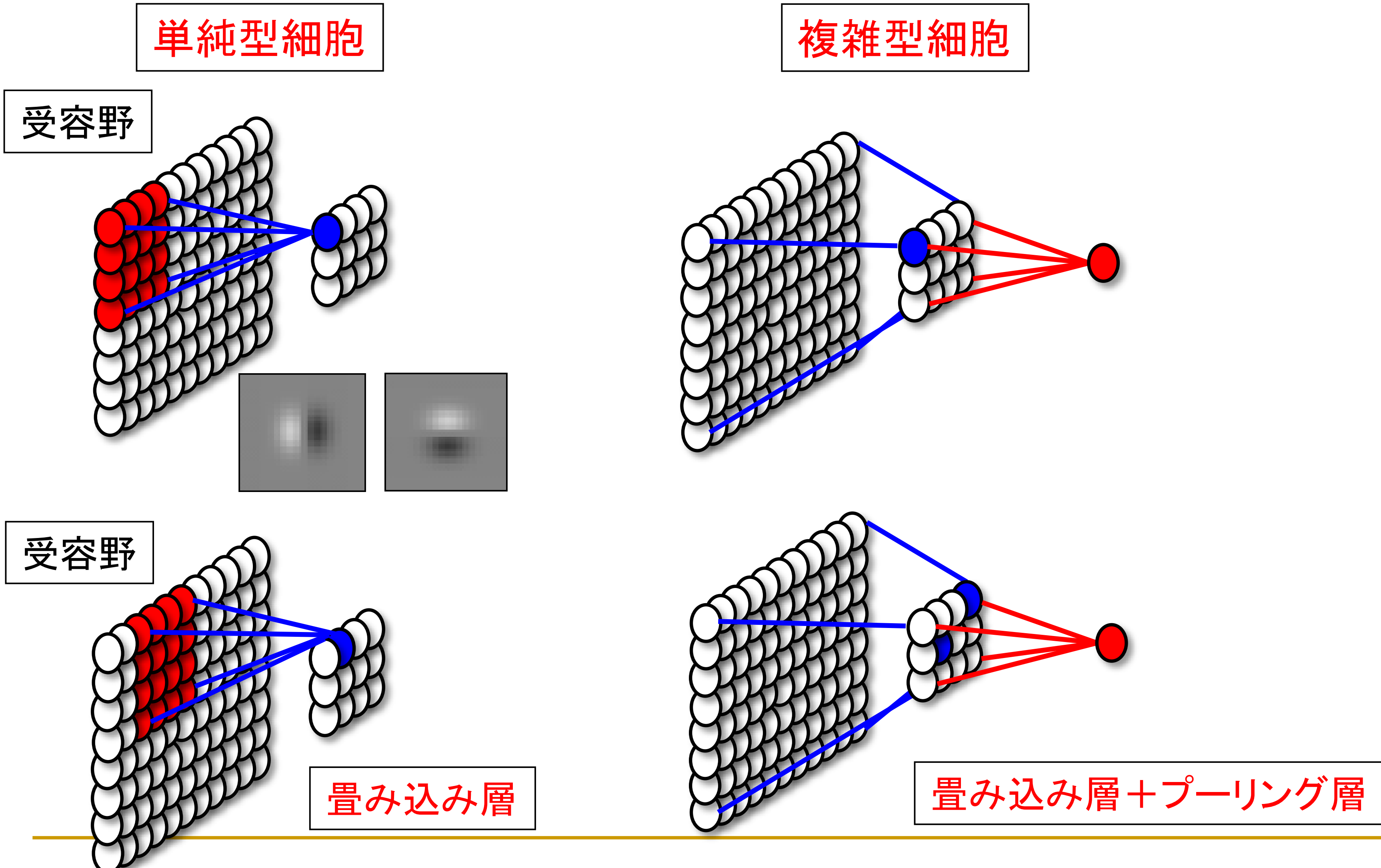


図:福島邦彦, 位置ずれに影響されないパターン認識機構の神経回路のモデルーネオコグニトロンー, 電子通信学会論文誌A, Vol.J62-A, No.10, pp.658-665, 1979

単純型細胞，複雑型細胞



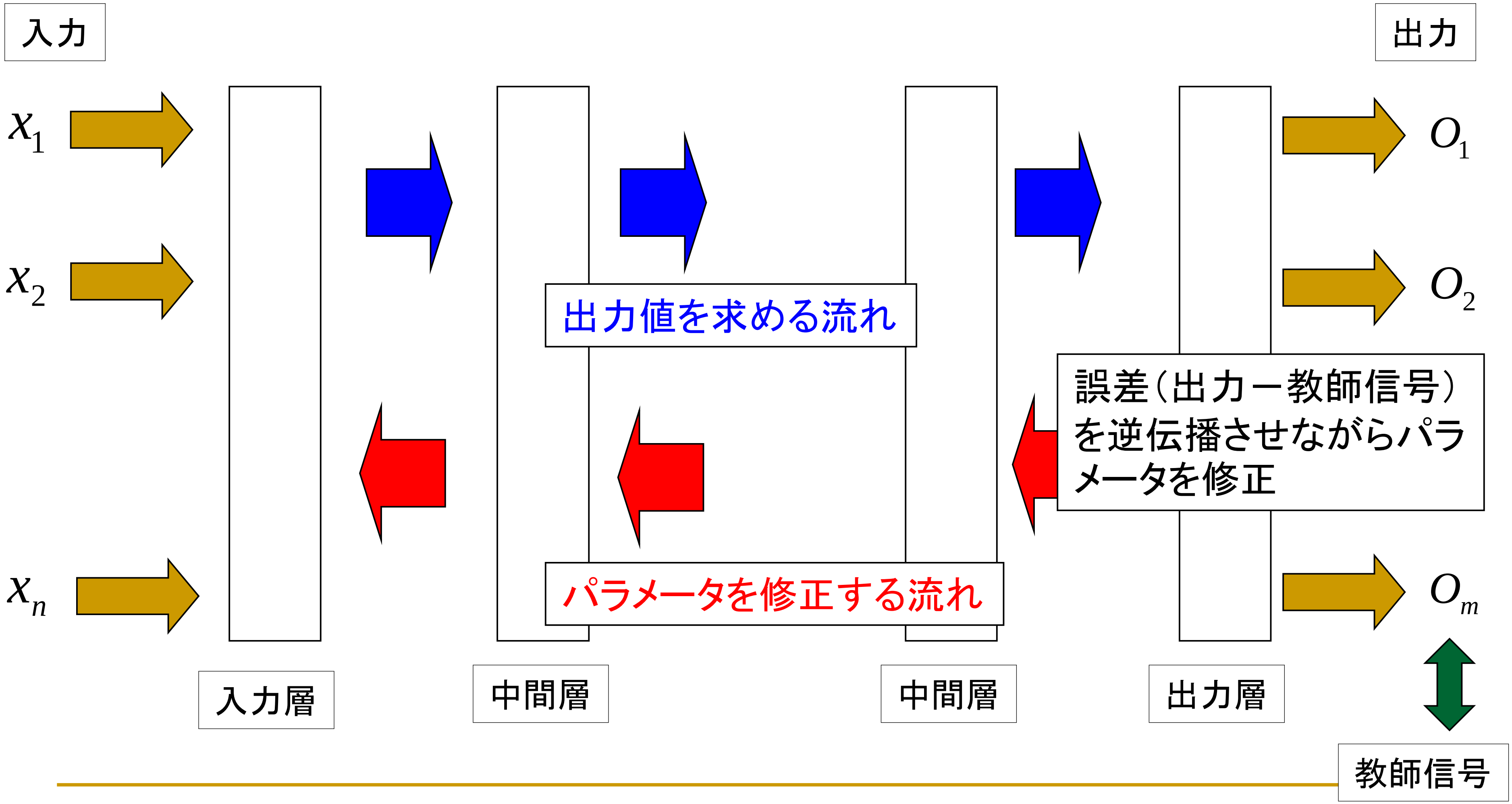
ニューラルネットワークの研究②

- 1986 誤差逆伝播則*（D.E.Rumelhart）
- 第二次ニューラルネットワークブーム
- 1989 Universal Approximation Theorem（G. Cybenko）
- 1989 LeNet（畳み込みニューラルネットワーク）（Y. LeCun）
- 1989 時間遅れニューラルネットワーク（A. Waibel）
- 1990 単純再帰結合型ネットワーク（J.L.Elman）
- 1997 Long Short-Term Memory（S.Hochreiter）
- 2002 コントラスティブ・ダイバージェンス法（G.E. Hinton）
- 2004 Echo State Network（H.Jaeger）

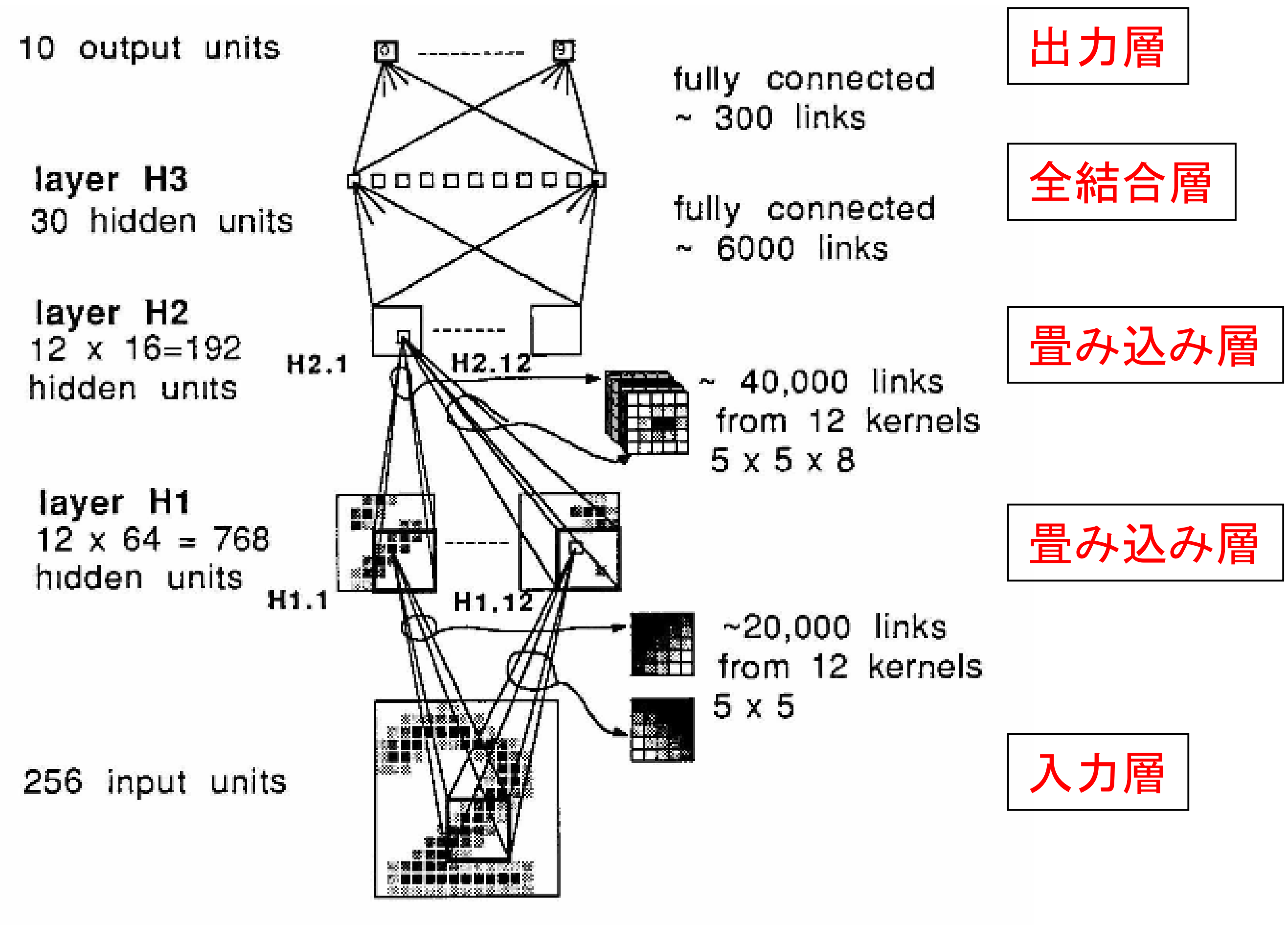
*ほぼ等価な解法は以前にも考案されていた

誤差逆伝播則（D.E.Rumelhart,1986）

- 階層型ニューラルネットワークの学習アルゴリズム



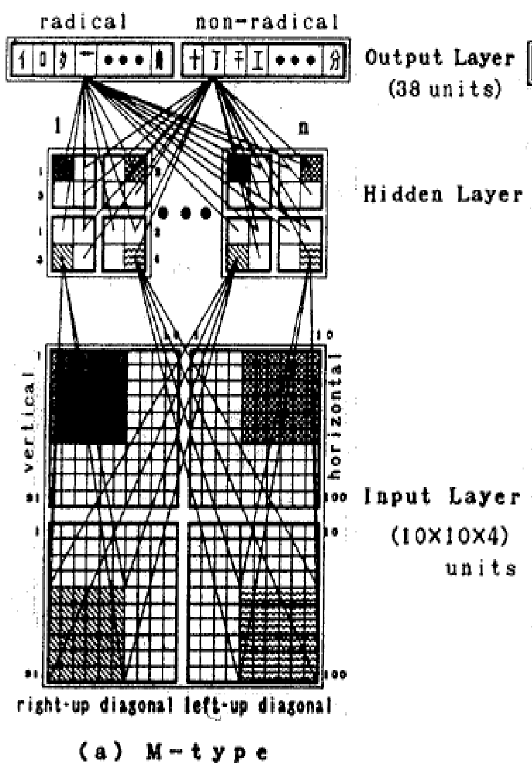
LeNet（畳み込みニューラルネットワーク）（Y.LeCun,1989）



図：Y. LeCun,Backpropagation Applied to Handwritten Zip Code Recognition, 1989

局所結合型ネットワーク(1994)

- 当時，畳み込みネットワークは計算量的に困難



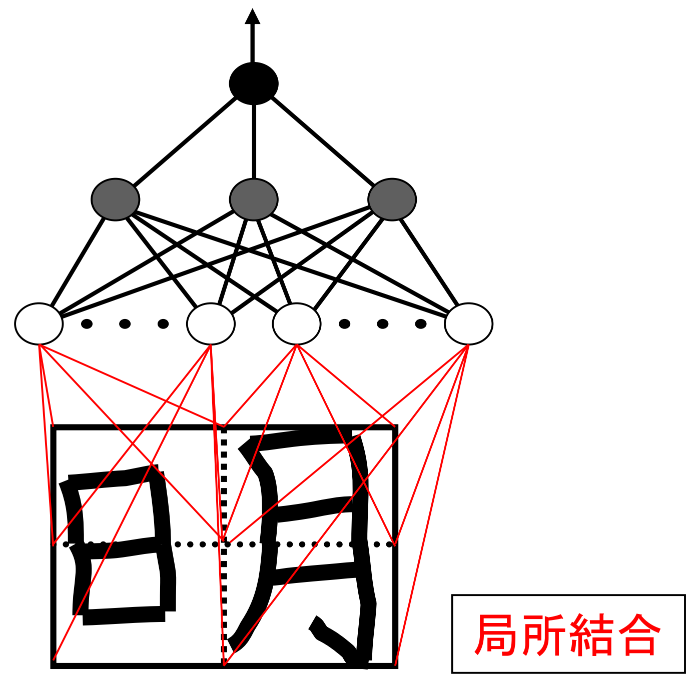
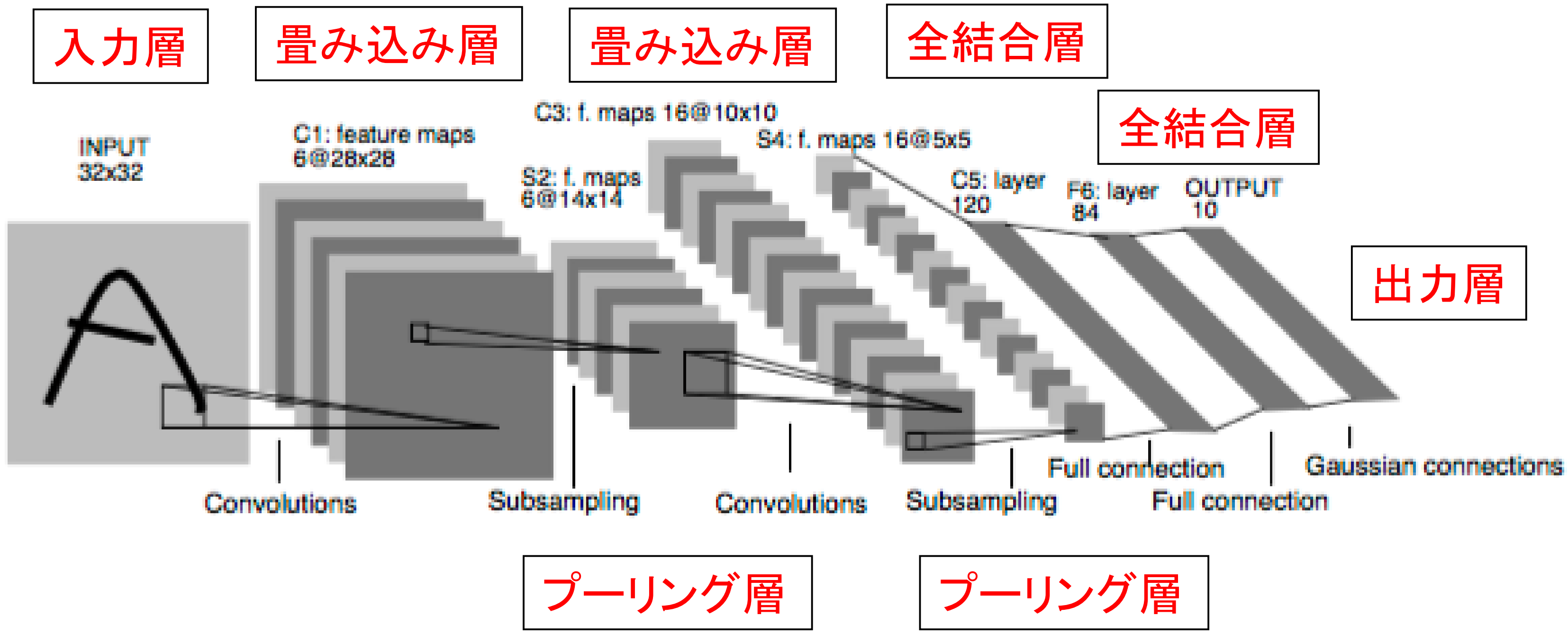


図:大友,局所結合型神経回路網モデルによる手書き漢字の効率的認識法,1994

LeNet-5（畳み込みニューラルネットワーク）（Y.LeCun,1998）

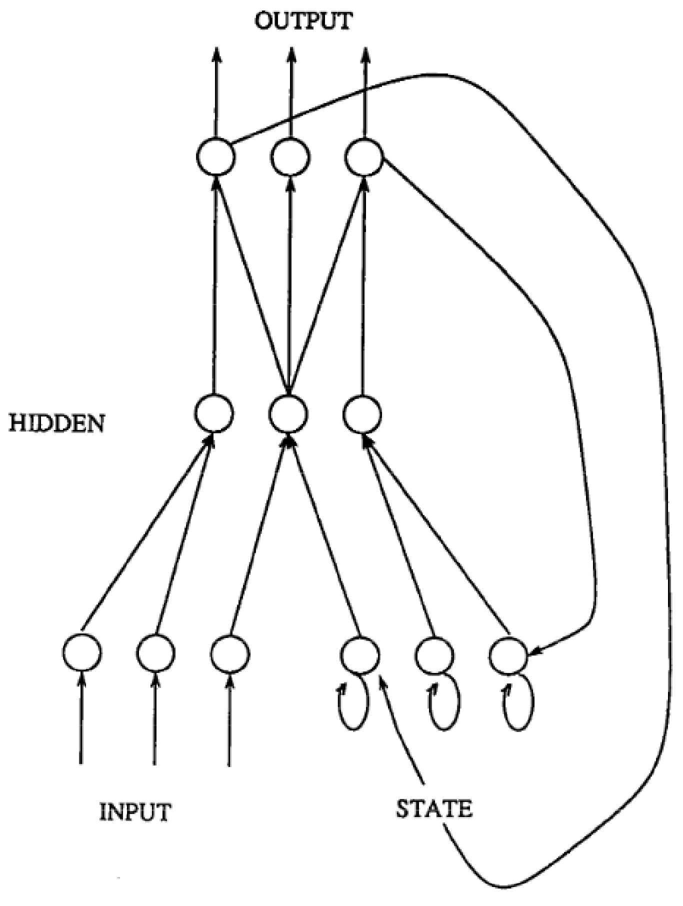
- 畳み込み層，プーリング層から構成されるニューラルネットワーク
  - 畳み込み層･･･単純型細胞
  - 畳み込み層，プーリング層･･･複雑型細胞



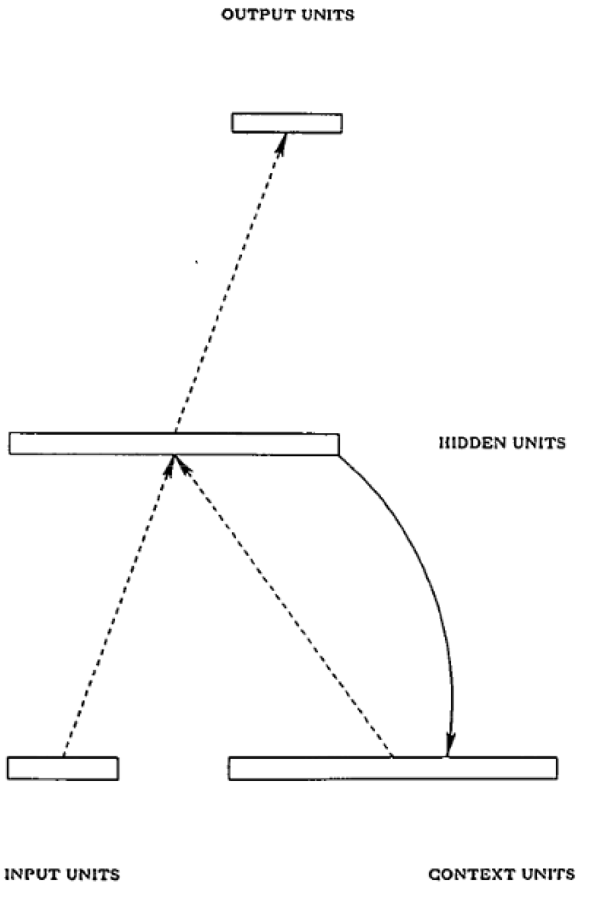
Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, Vol.86, No.11, pp.2278-2324, 1998

再帰結合型ニューラルネットワーク

ジョルダンネット



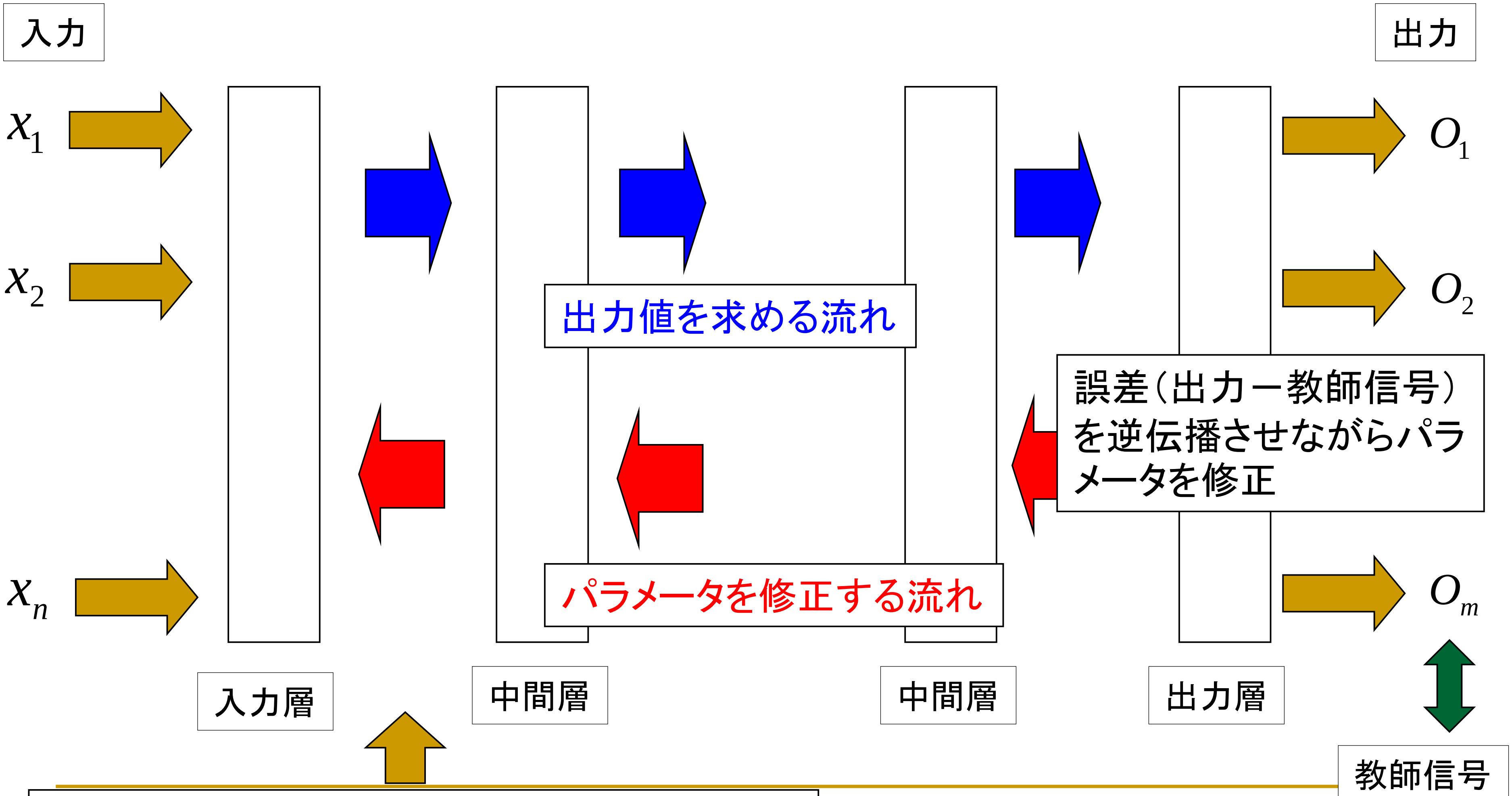
エルマンネット



Elman, J. L., Finding structure in time, Cognitive Science, Vol.14, pp.179–211, 1990

誤差逆伝播則の問題点

- 勾配消失問題，ローカルミニマム問題，etc



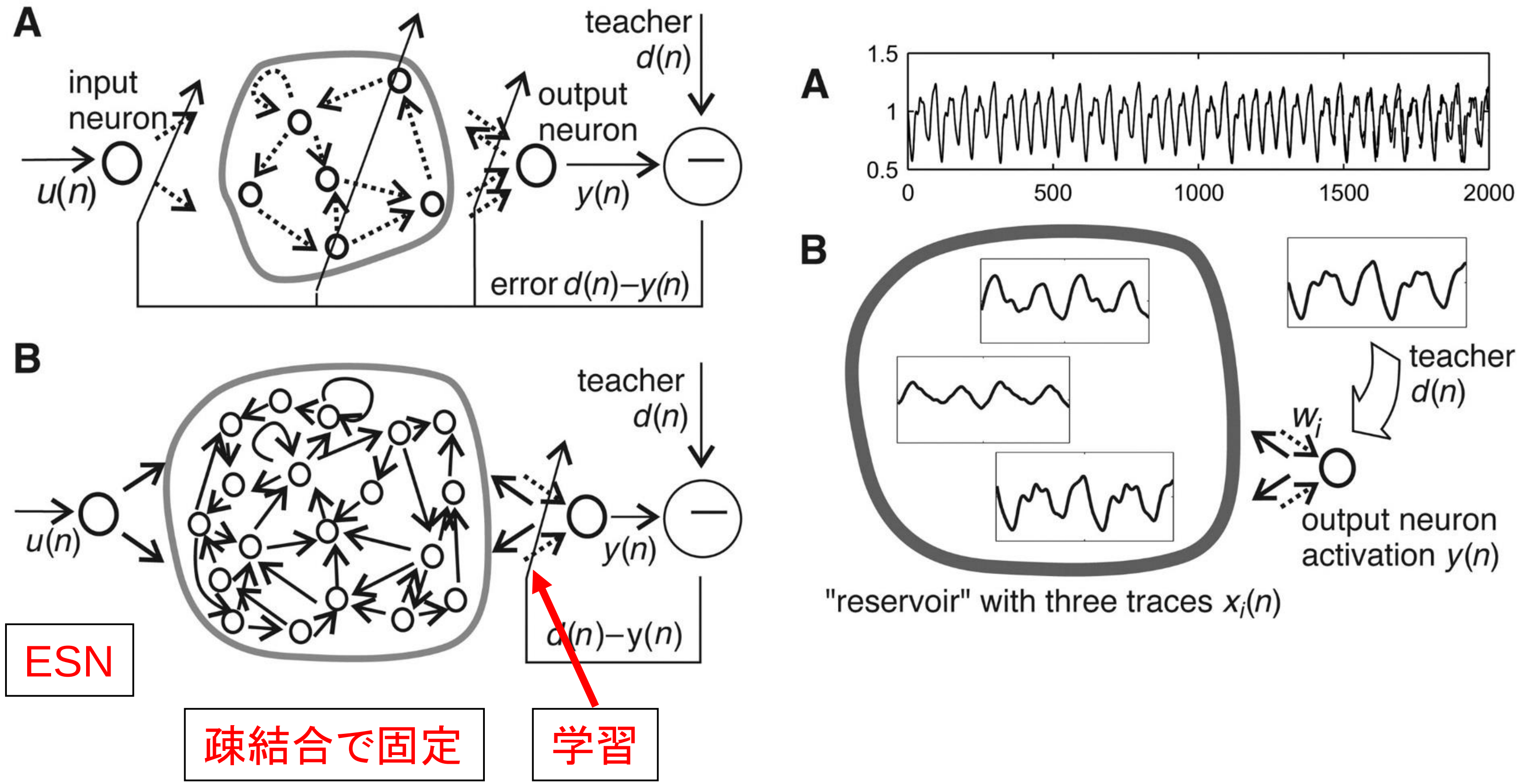
下層ほどパラメータの修正が困難

ニューラルネットワークの研究②

- 1986 誤差逆伝播則（D.E.Rumelhart）
- 1989 Universal Approximation Theorem（G. Cybenko）
- 1989 LeNet（畳み込みニューラルネットワーク）（Y. LeCun）
- 1989 時間遅れニューラルネットワーク（ A. Waibel）
- 1990 単純再帰結合型ネットワーク（J.L.Elman）
- 第二次ニューラルネットワークブームが終わる
- 1995 Support Vector Machine（V.Vapnik）
- 1997 Long Short-Term Memory（S.Hochreiter）
- 2002 コントラスティブ・ダイバージェンス法（G.E. Hinton）
- 2004 Echo State Network（H.Jaeger）

Echo State Network（H.Jaeger）

Recurrent Neural Network



図：Jaeger, H., Haas, H.: Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication, Science, 304, pp.78-80, 2004

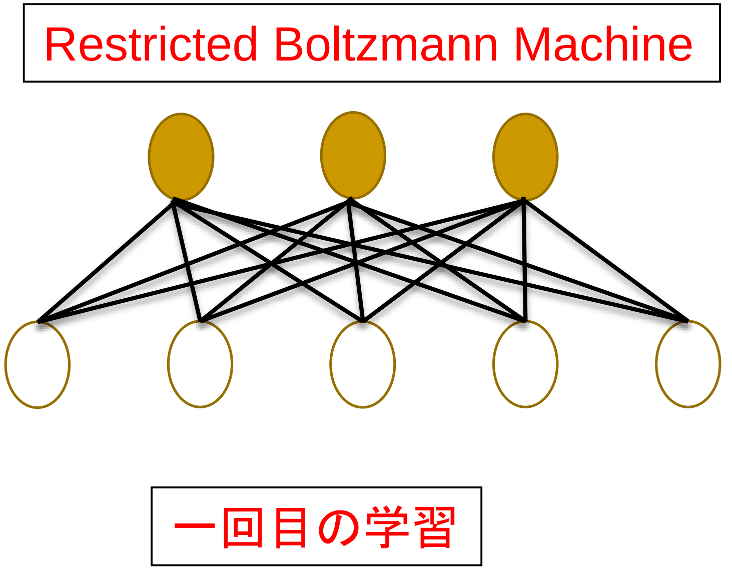
ニューラルネットワークの研究③

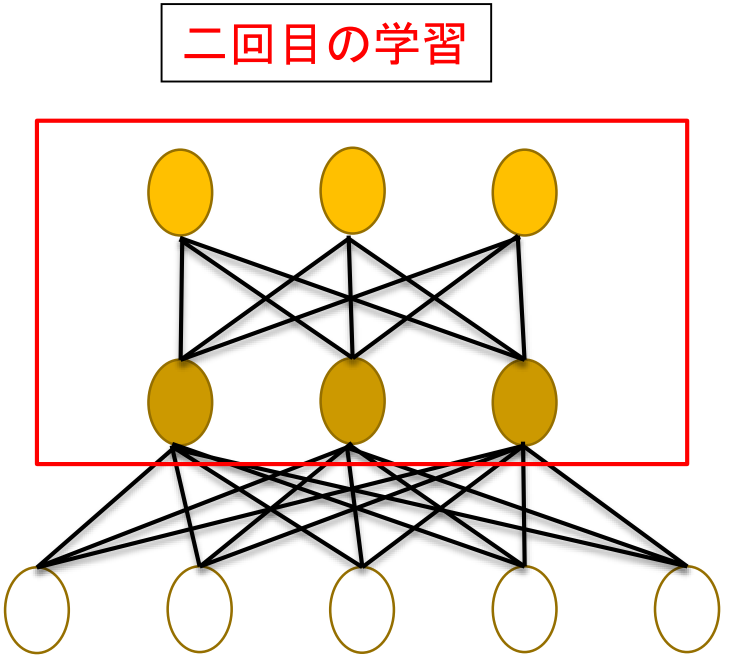
- 2006 Deep Belief Network（G.E.Hinton）
- 2012 AlexNet（ILSVRC2012）
- 第三次ニューラルネットワークブーム
- 2012 Googleの猫（Q.V.Le）
- 2013 word2vec（T. Mikolov）
- 2014 VGG, GoogLeNet（ILSVRC2014）
- 2015 ResNet（ILSVRC2015）
- 2015 Deep Q-Learning（V.Mnih）
- 2016 Generative Adversarial Network（I.J.Goodfellow）

ILSVRC: The ImageNet Large Scale Visual Recognition Challenge

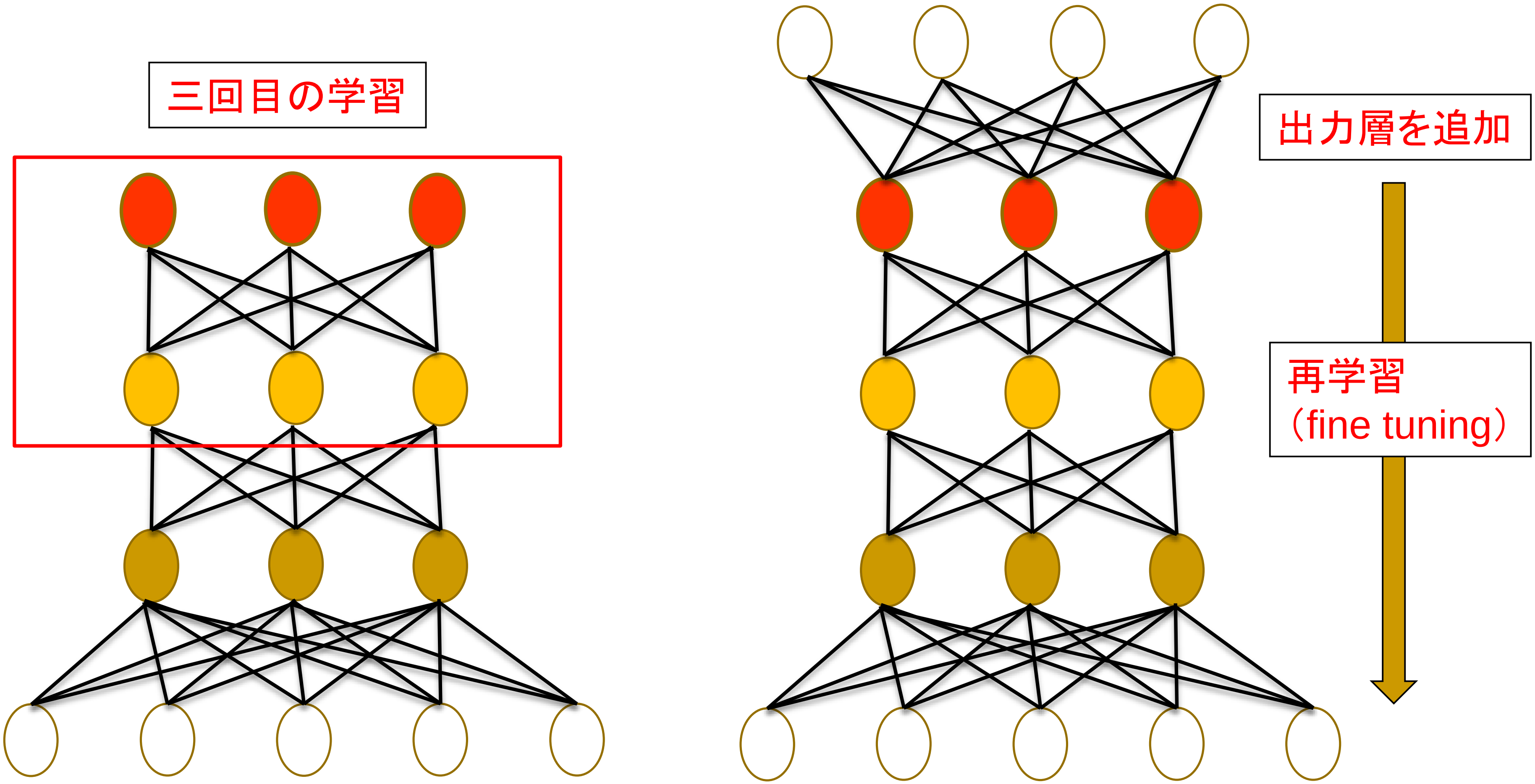
Deep Belief Network（G.E.Hinton）

- 多層の学習方法（深層学習）





Deep Belief Network（G.E.Hinton）



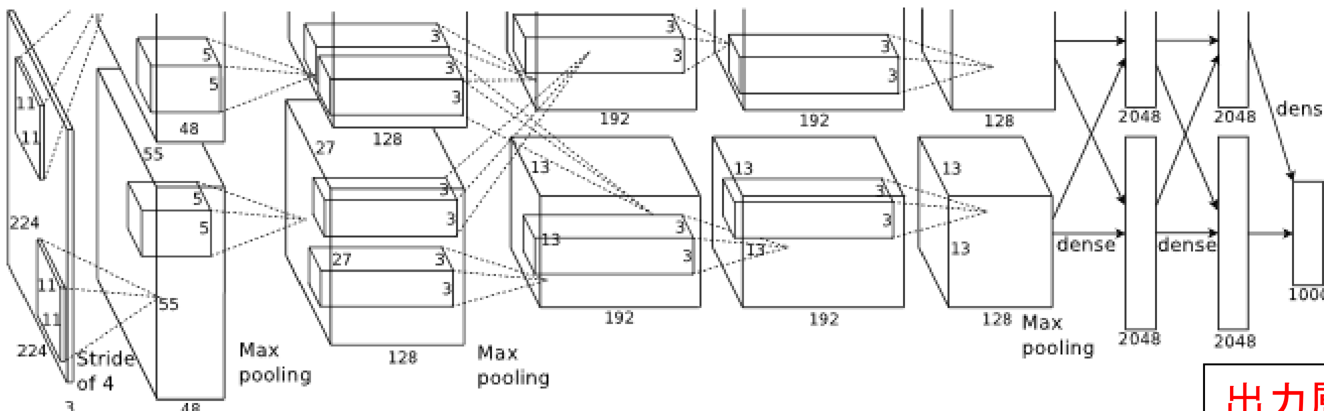
AlexNet（A. Krizhevsky,2012）

- 8層の畳み込みニューラルネットワーク
  - ILSVRC2012において判定エラー率を25.8%から16.4%に改善

入力層

全結合層

ImageNet



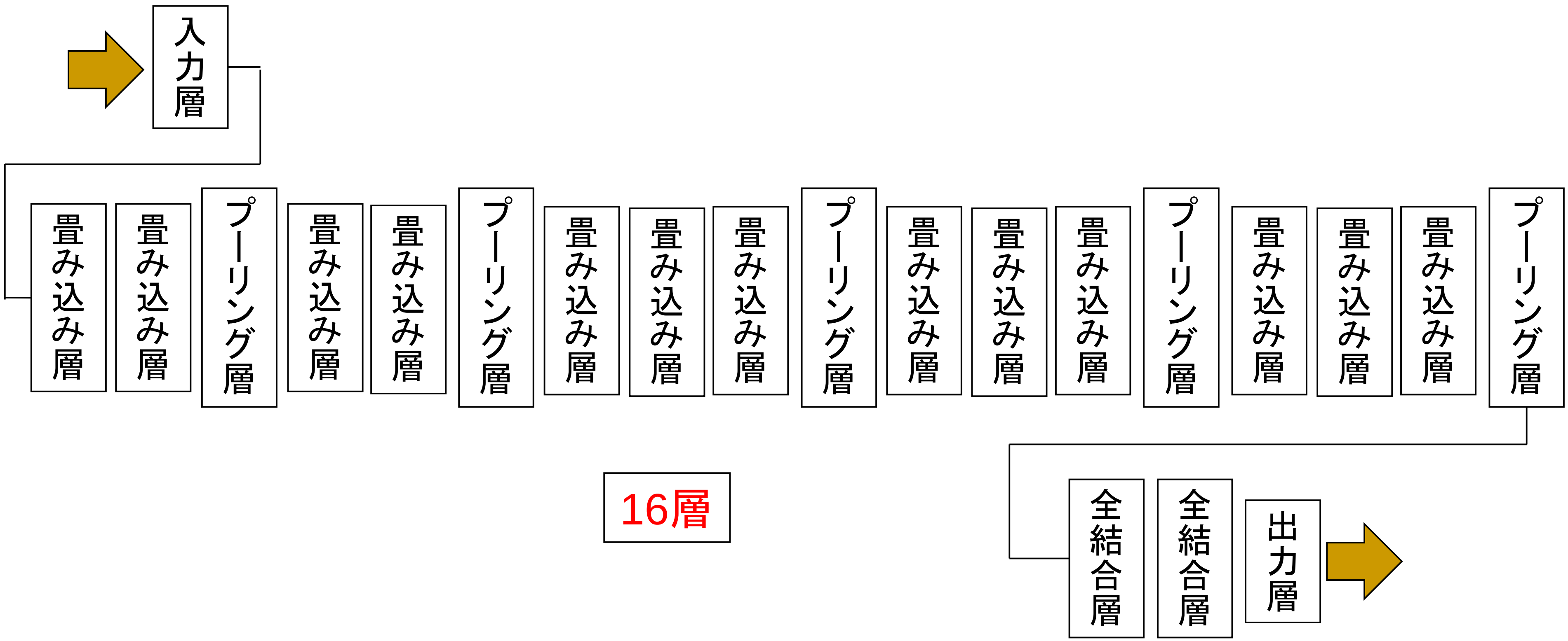
出力層

畳み込み層＋プーリング層

図:A. Krizhevsky, ImageNet Classification with Deep Convolutional Neural Networks, Advances in neural information processing systems, 2012

VGG（Visual Geometry Group）

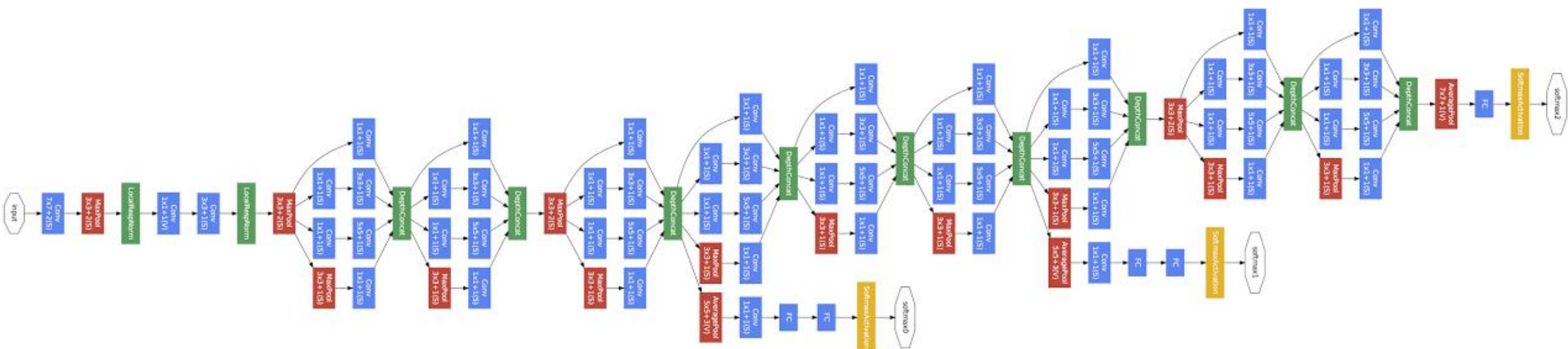
- VGG-16（ILSVRC2014）
  - 判定エラー率を7.3%に改善



K.Simonyan, A.Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, 2015

GoogLeNet（C. Szegedy,2014）

- GooLeNet（Inception-v3）（ILSVRC2014）
  - 判定エラー率を6.7%に改善



22層

畳み込み

プーリング

Softmax

結合

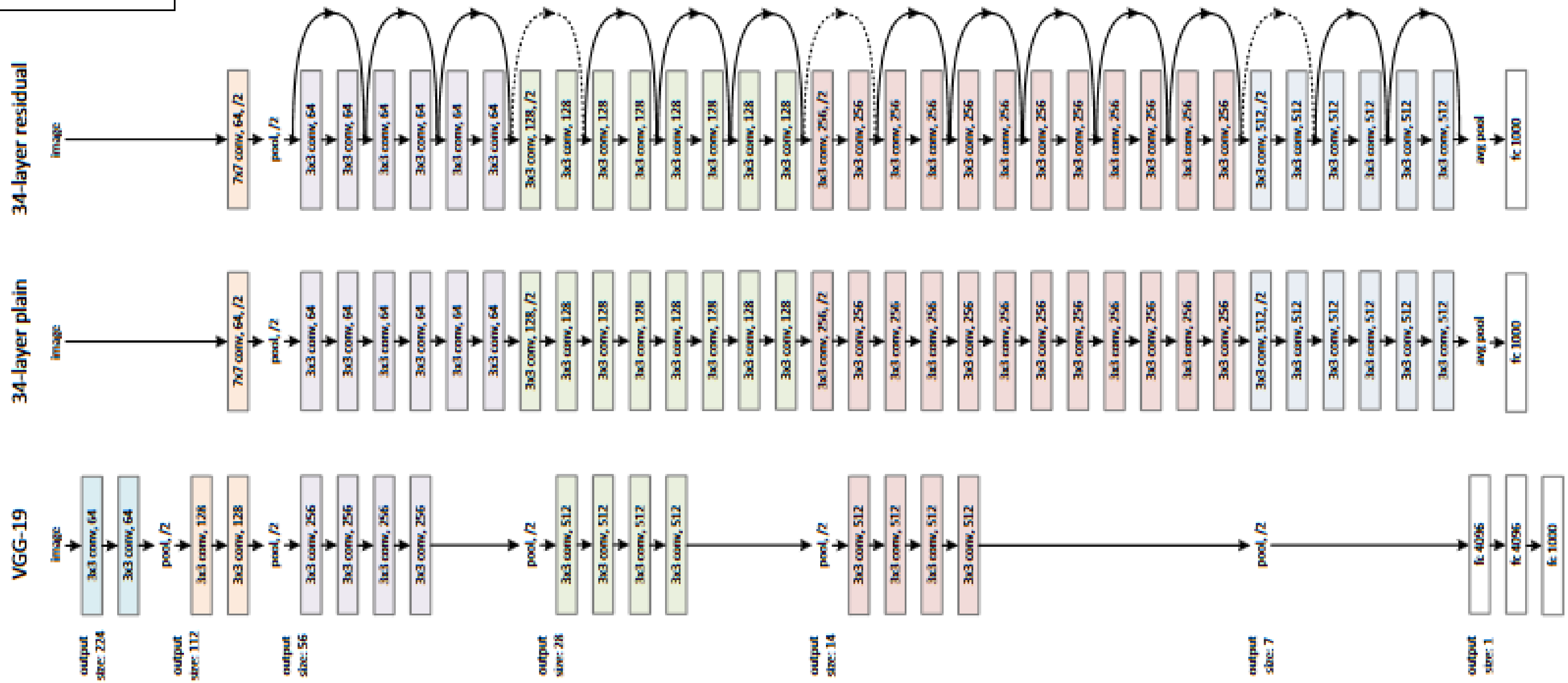
図：http://www.image-net.org/challenges/LSVRC/2014/slides/GoogLeNet.pptx

ResNet（Microsoft Research Asia,2015）

- ResNet（Residual Network）
  - 判定エラー率を3.57%に改善

ResNet

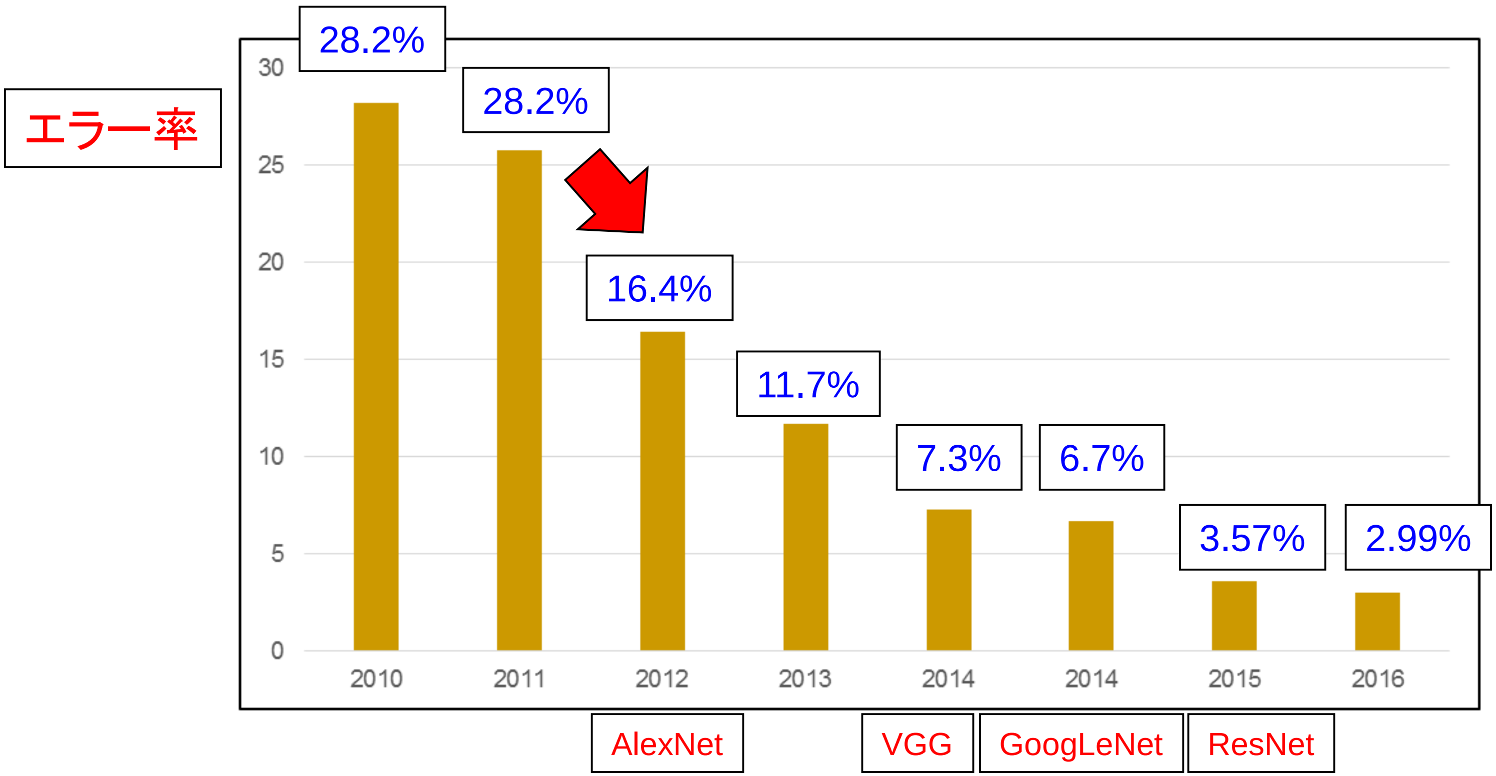
152層（ILSVRC2015）



図：K.He, Deep Residual Learning for Image Recognition, 2015

大規模化と精度の向上

- ILSVRCにおけるエラー率の向上

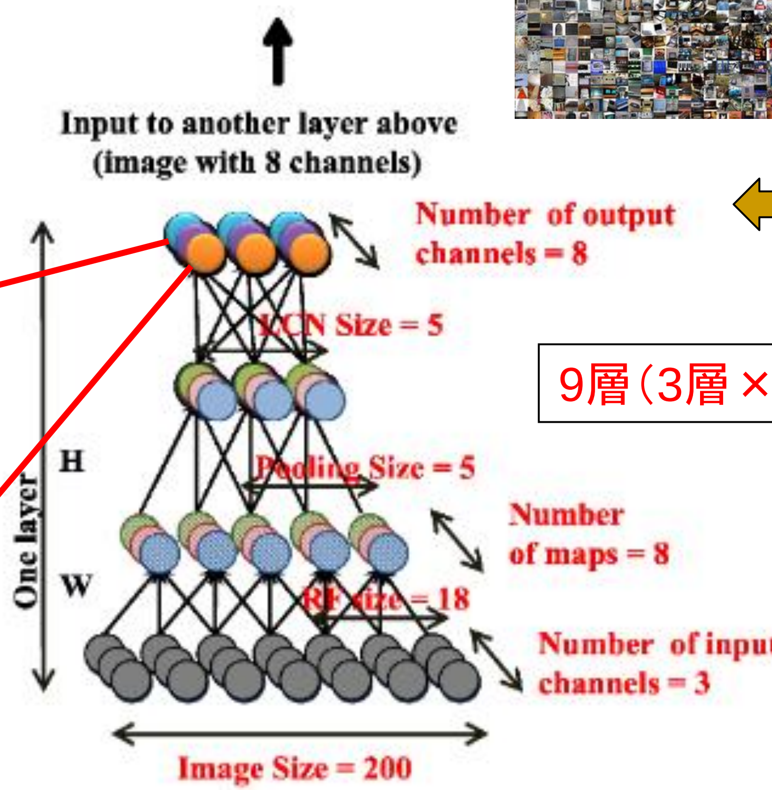


ILSVRC2013：ZFNET
ILSVRC2016：CUImage

Googleの猫（Q.V.Le, 2012）

- 1,000万の画像の教師なし学習

Googleの猫



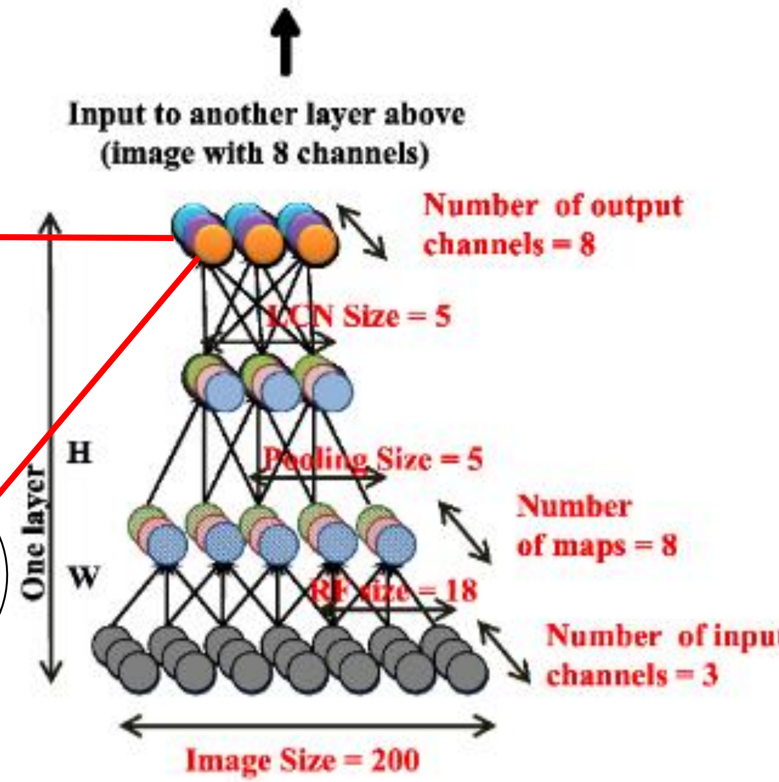
9層（3層×3個）

図：Q.V.Le, Building high-level features using large scale unsupervised learning, 2012

おばあさん細胞説（J.Y.Lettvin）

- 仮想的な細胞

Googleの猫



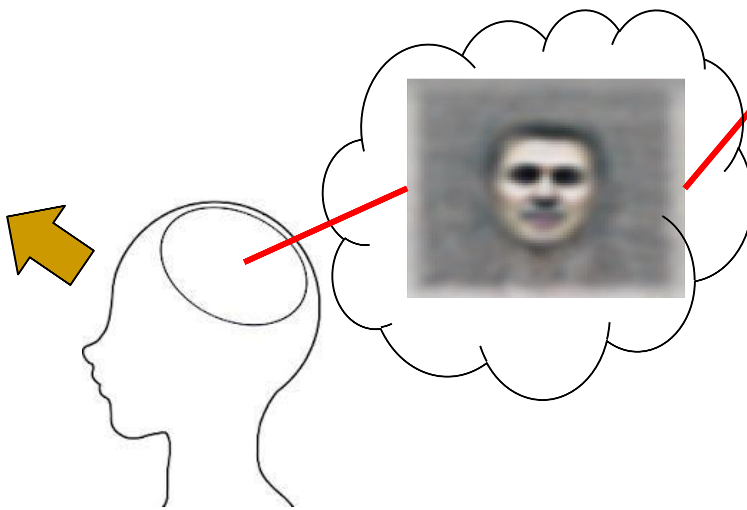
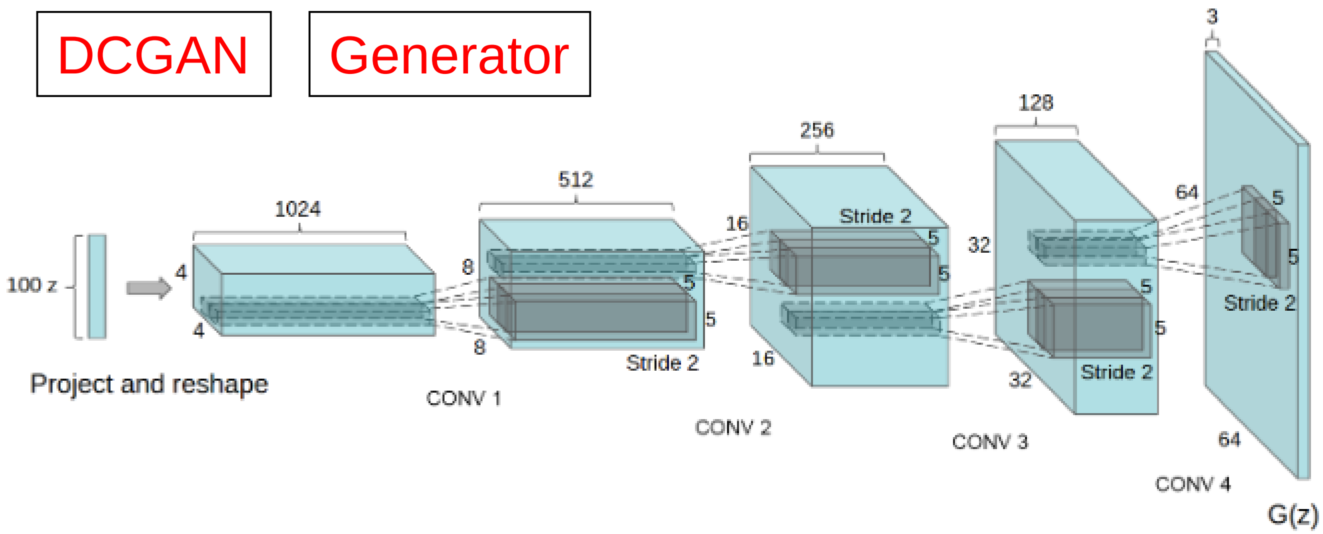


図:M.Moscovitch, What is Face Recognition ? Nineteen Experiments on a Person with Visual Object Agnosia and Dyslexia but Normal Face Recognition, 1997

Generative Adversarial Network

DCGAN

Generator



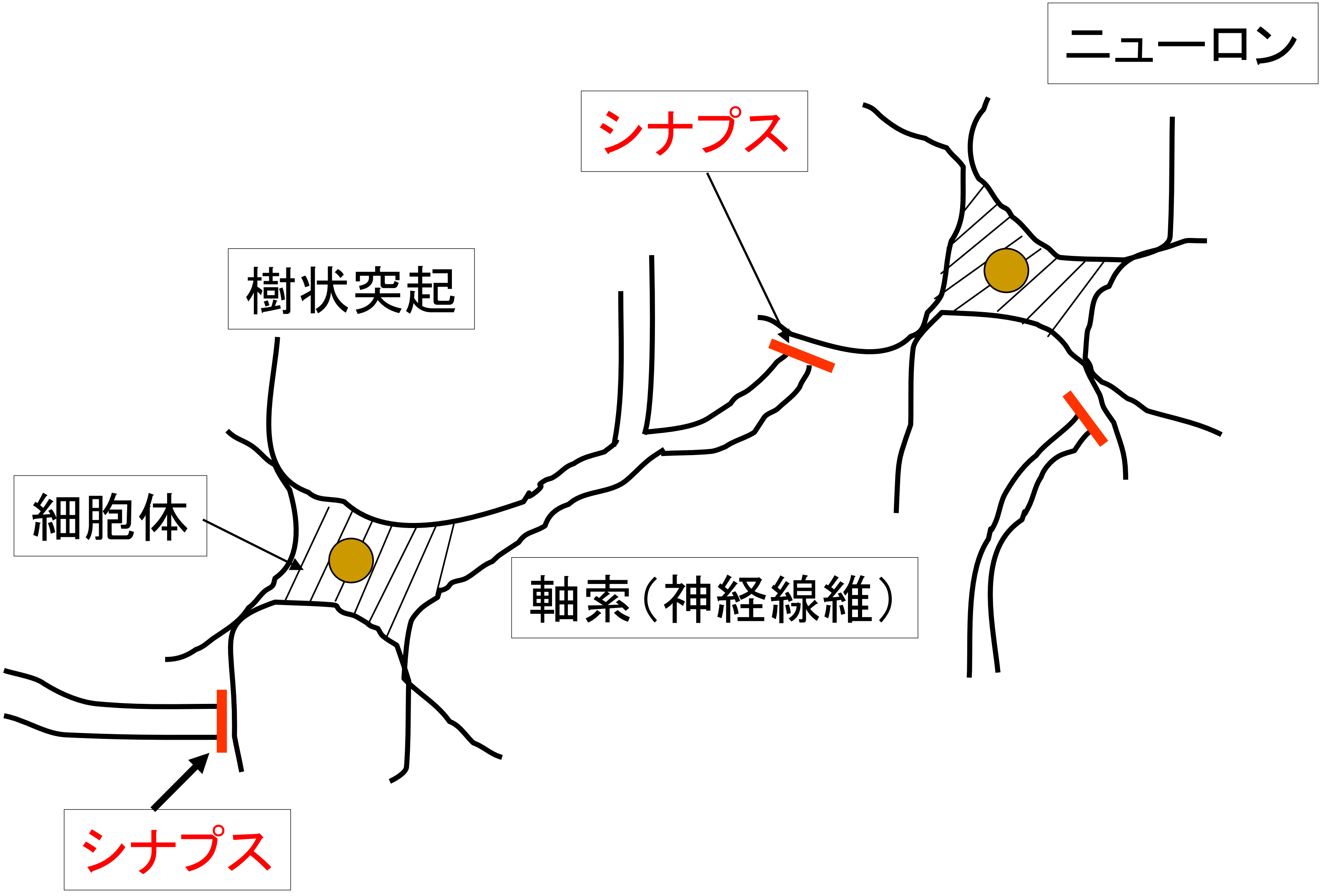
生成された画像

図：A.Radford，Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks，2016

ニューロンのモデル化

McCulloch-Pittsモデル

神経細胞（イメージ図）



ニューロンのモデル化（概念）

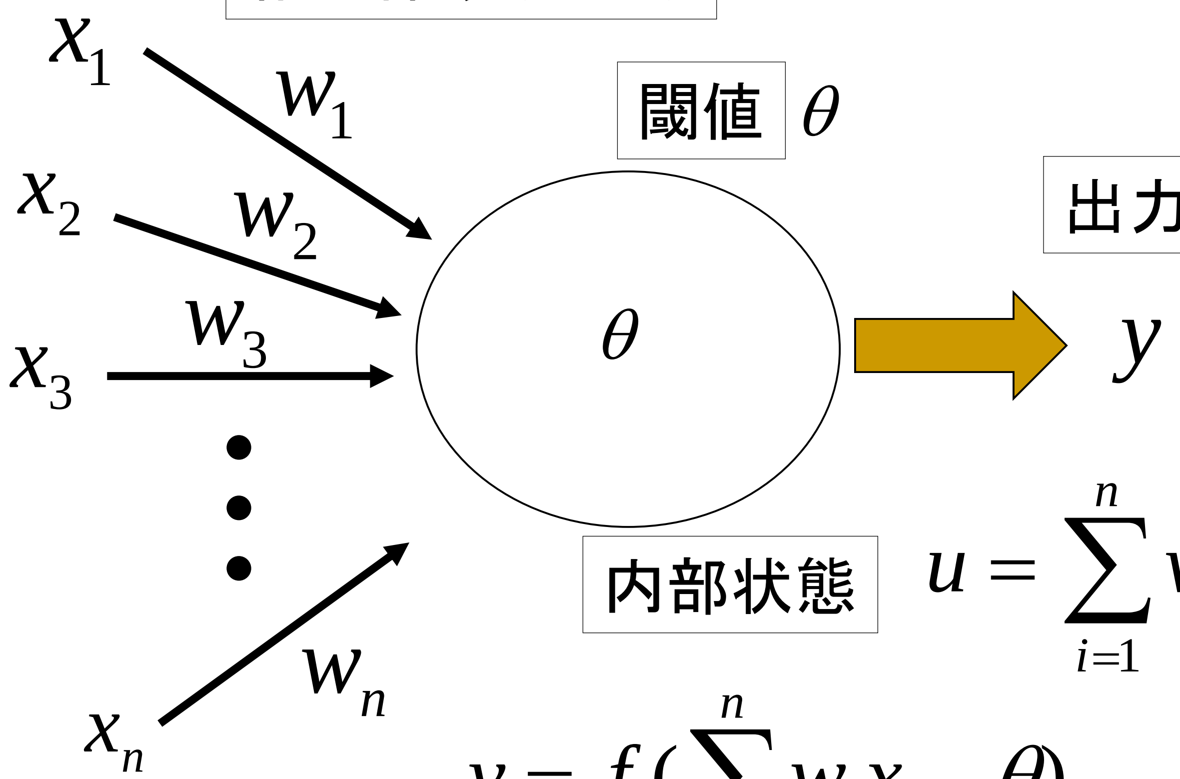
- 神経線維を通して（電気）信号を送る（一出力）
- 樹状突起部のシナプスを介して信号が伝わる
- 複数のシナプスからの信号により内部状態が定まる（多入力）
- 内部状態がしきい値を超えると信号を送る

ニューロンのモデル化①

多入力一出力

結合係数(重み)

入力値



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

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

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

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

⇕

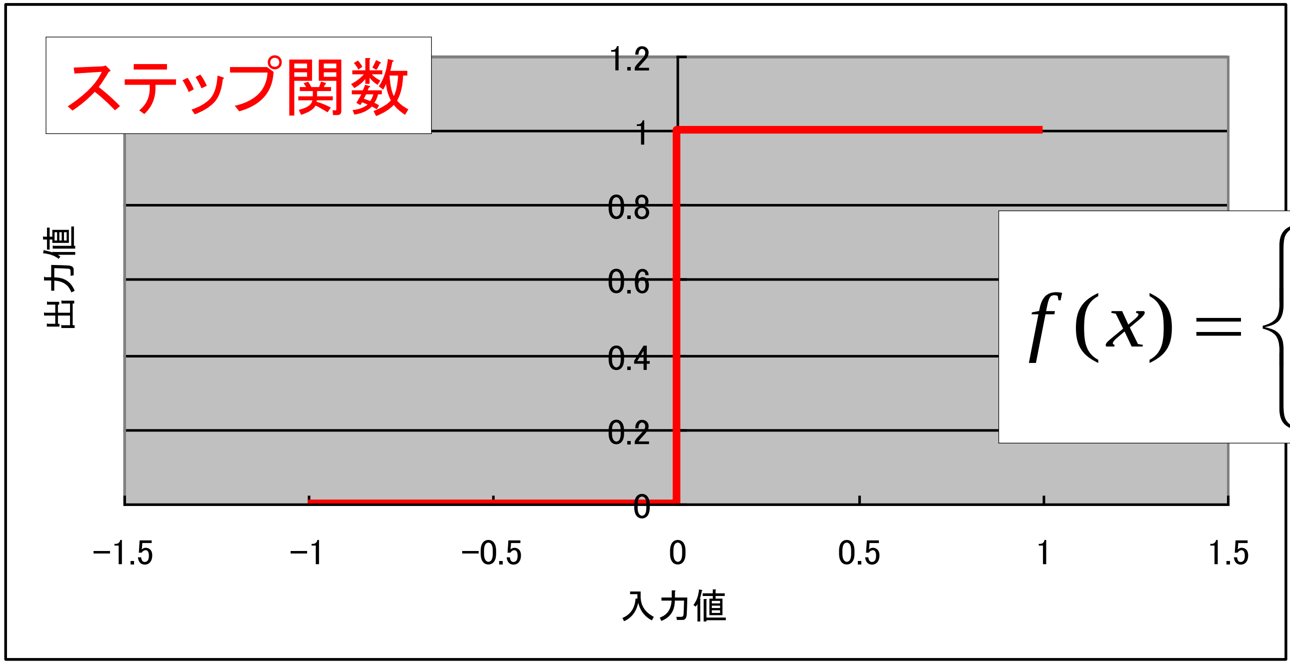
$x_0 = -1, w_0 = \theta$　とすると

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

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）



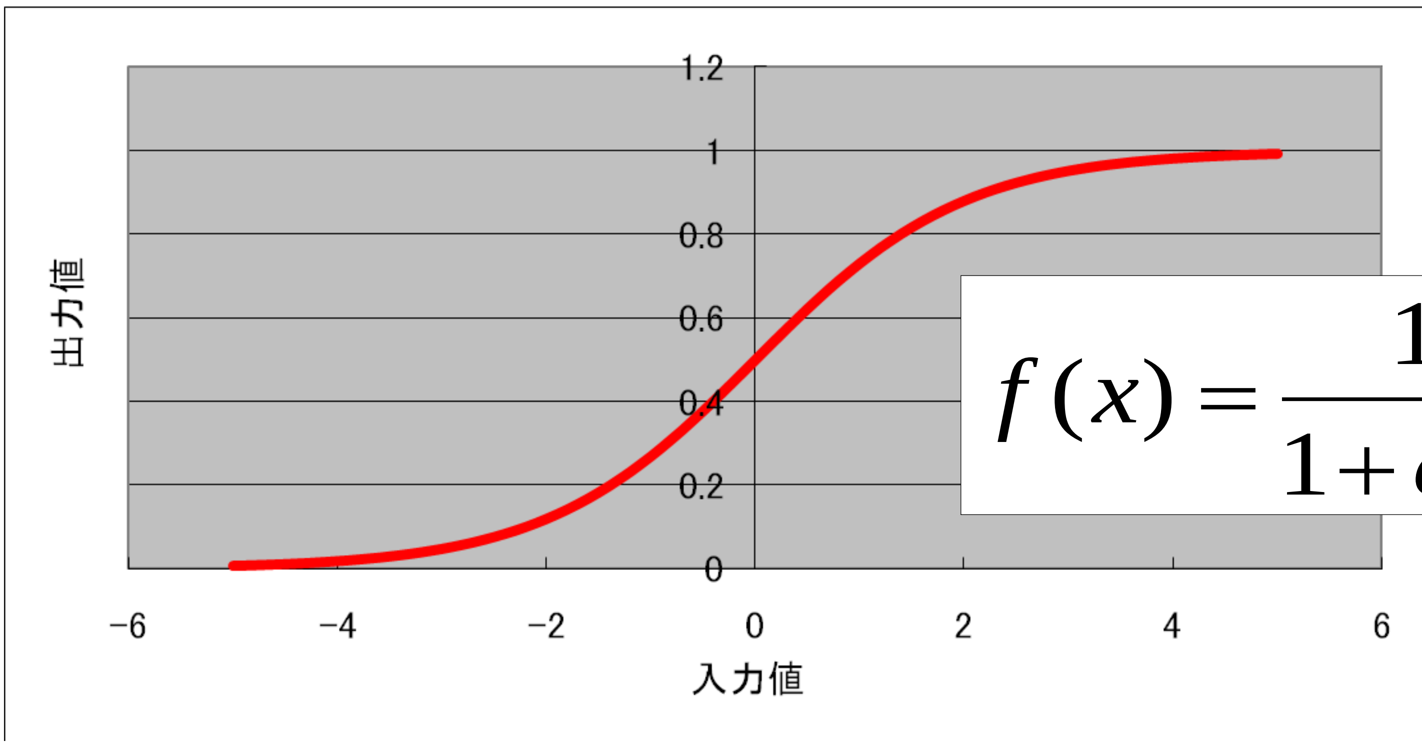
$$f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

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

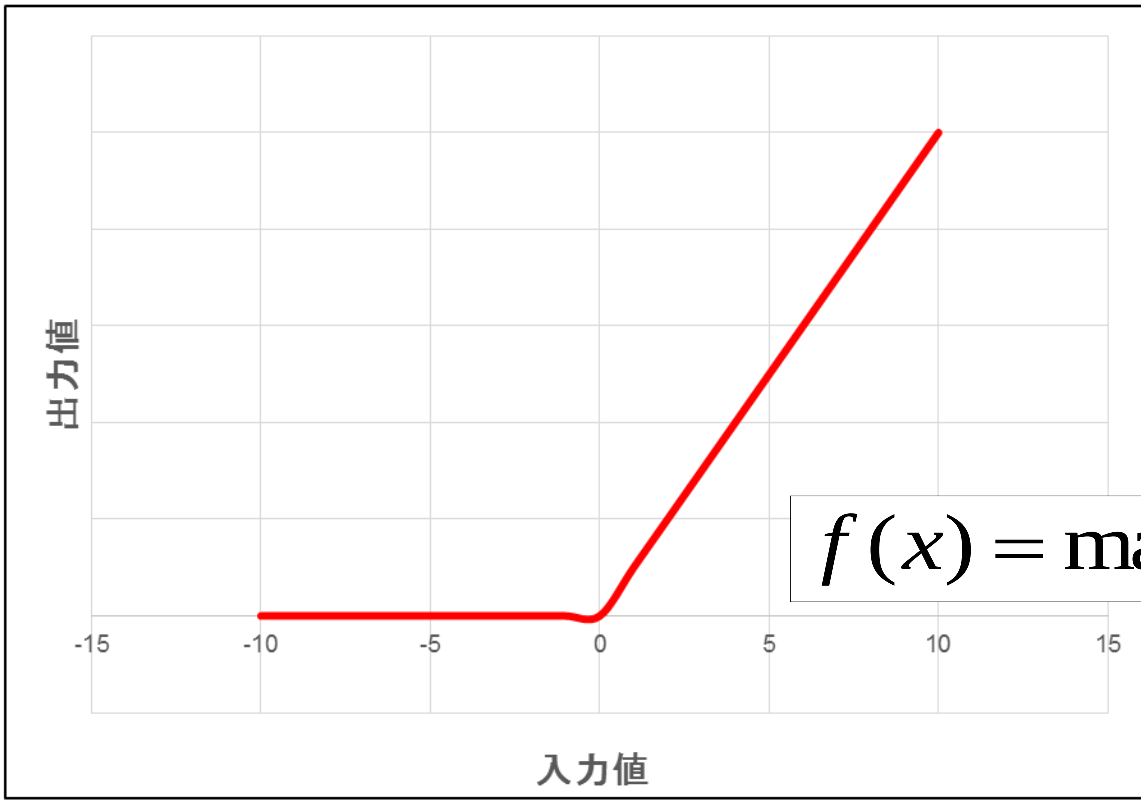
- シグモイド関数の特徴

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

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

活性化関数③

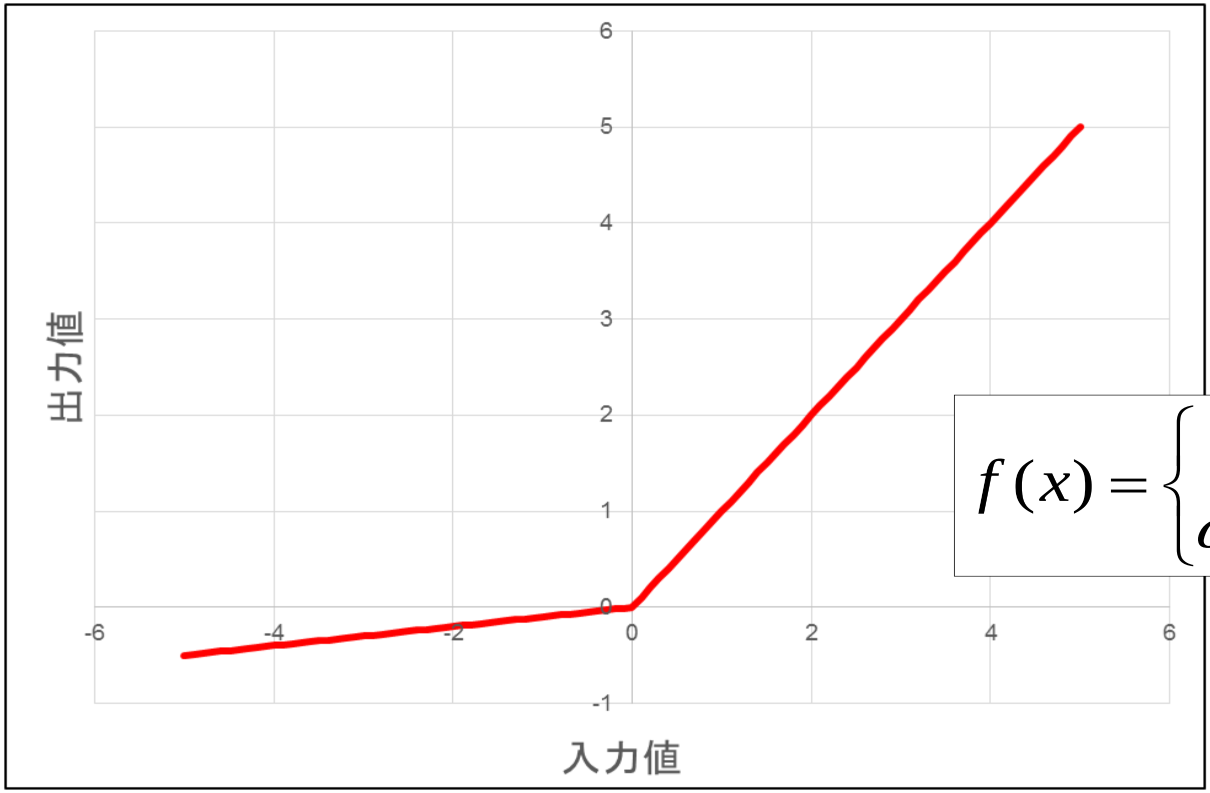
- 正規化線形関数（Rectified Linear Unit）



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

活性化関数④

- Leakly ReLU



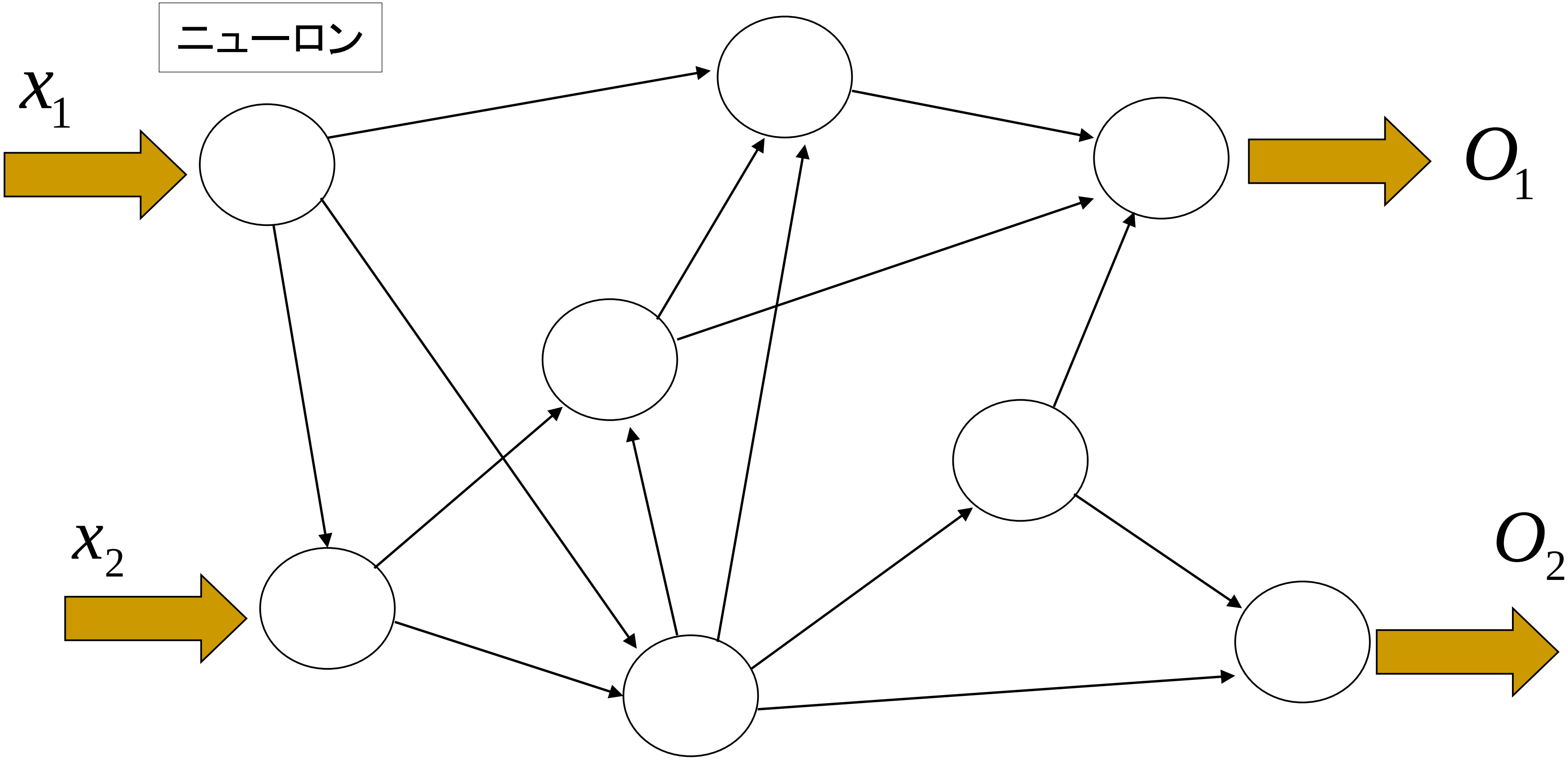
$$f(x) = \begin{cases} x & x > 0 \\ \alpha x & x \leq 0 \end{cases}$$

階層型ニューラルネットワーク

パーセプトロン

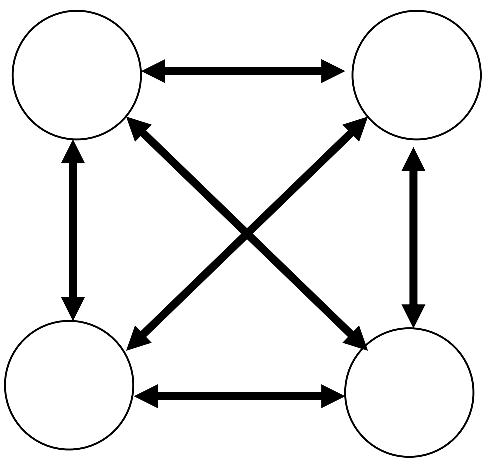
ニューラルネットワーク

ニューロンを互いに結合（ネットワーク化）

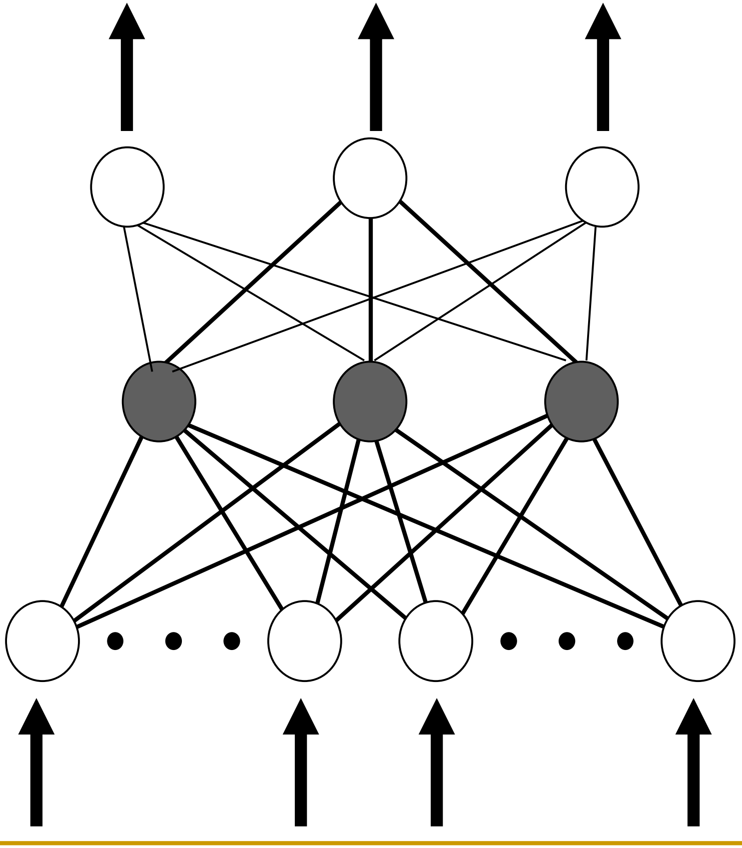


ネットワークの形態

- 相互結合型

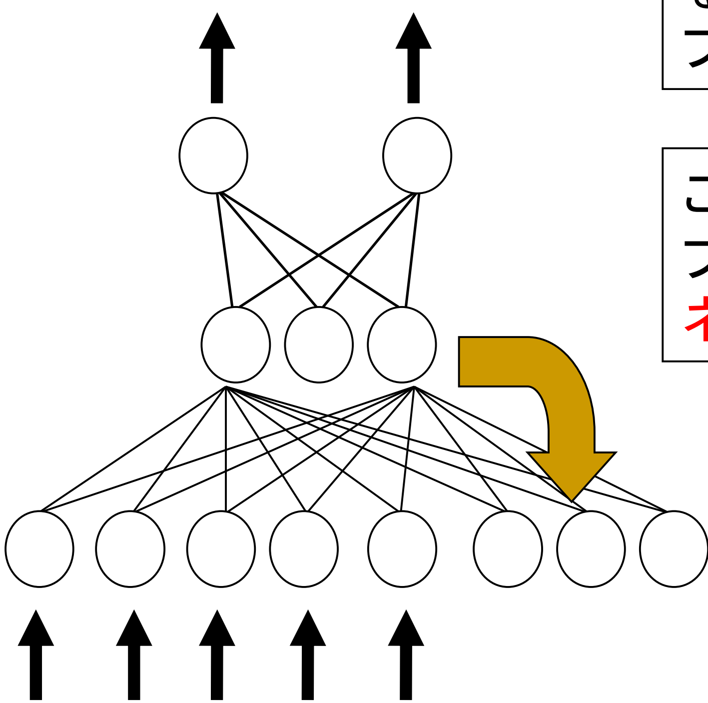


- 階層（フィードフォワード）型



ネットワークの形態

- リカレント型

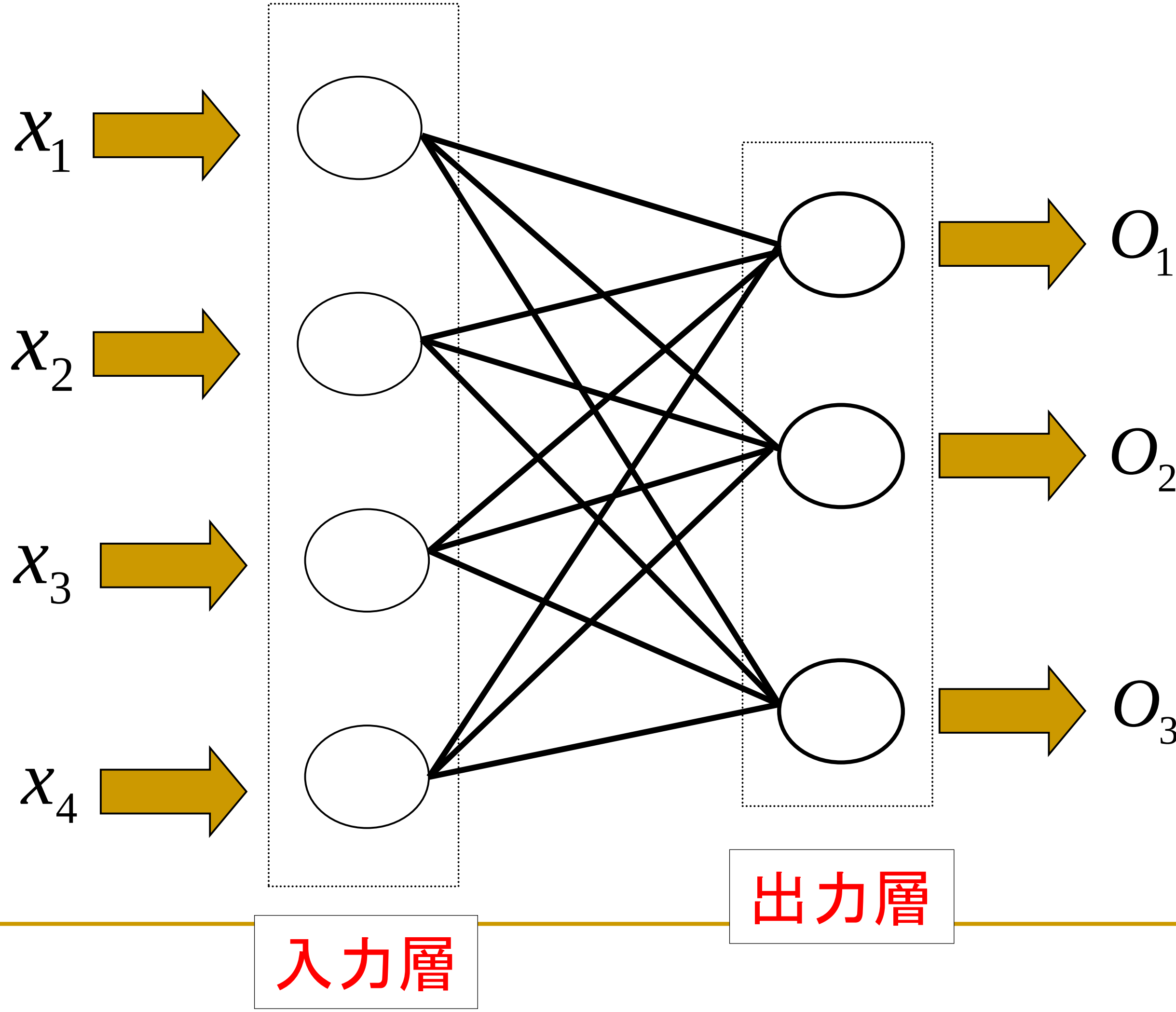


構造的には，フィードフォワード型であるが前の層，もしくは自分自身に対してフィードバックを持つ

この場合は，中間層の値を入力層に戻すフィードバックを持ち，単純再帰結合型ネットワーク（エルマンネット）と呼ばれる

階層型ネットワーク①

二層のフィードフォワード構造



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

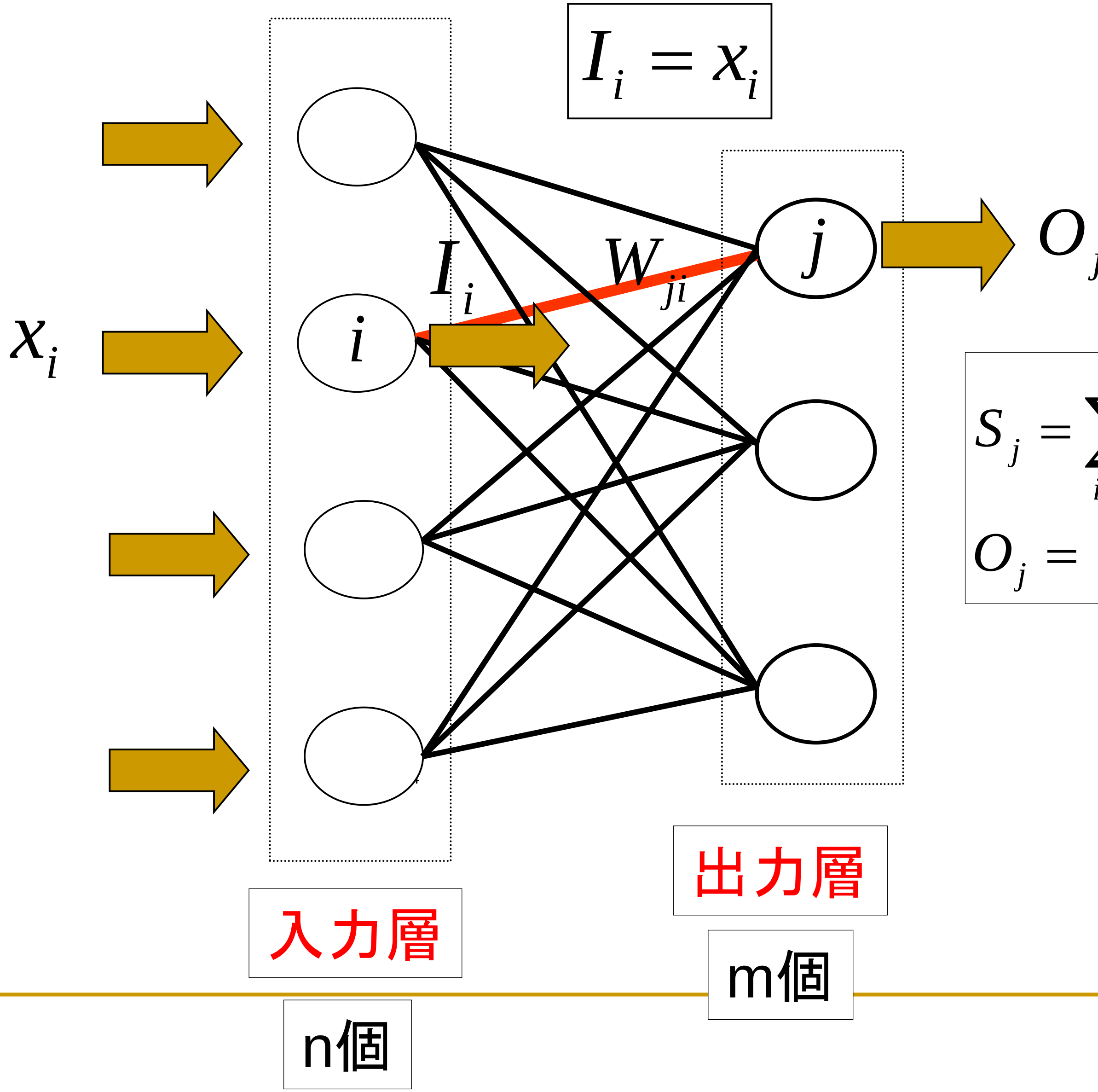
- i 番目の入力層の値（入力信号x_iと同じ値） I_i
- j 番目の出力層の出力値　O_j　内部状態　S_j

- 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

$$I_i = x_i$$



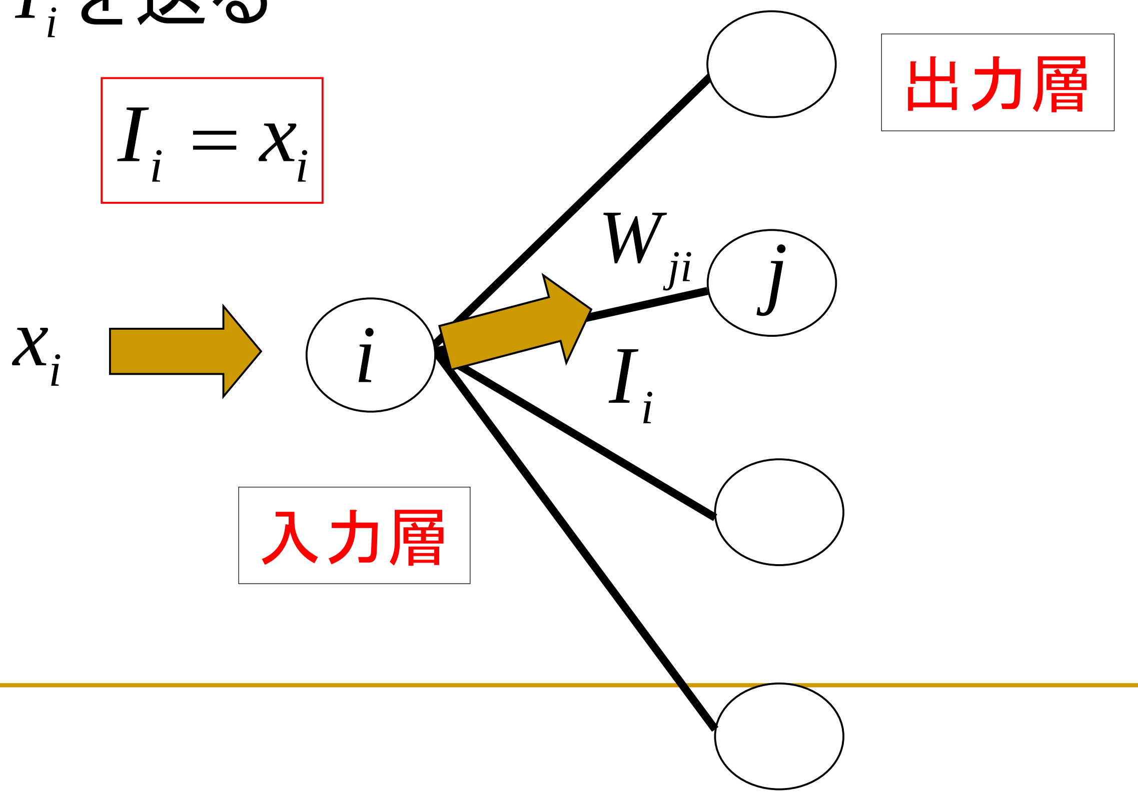
$$S_j = \sum_{i=i}^{n} W_{ji} I_i - \theta_j$$

$$O_j = f(S_j)$$

ネットワークの動作①

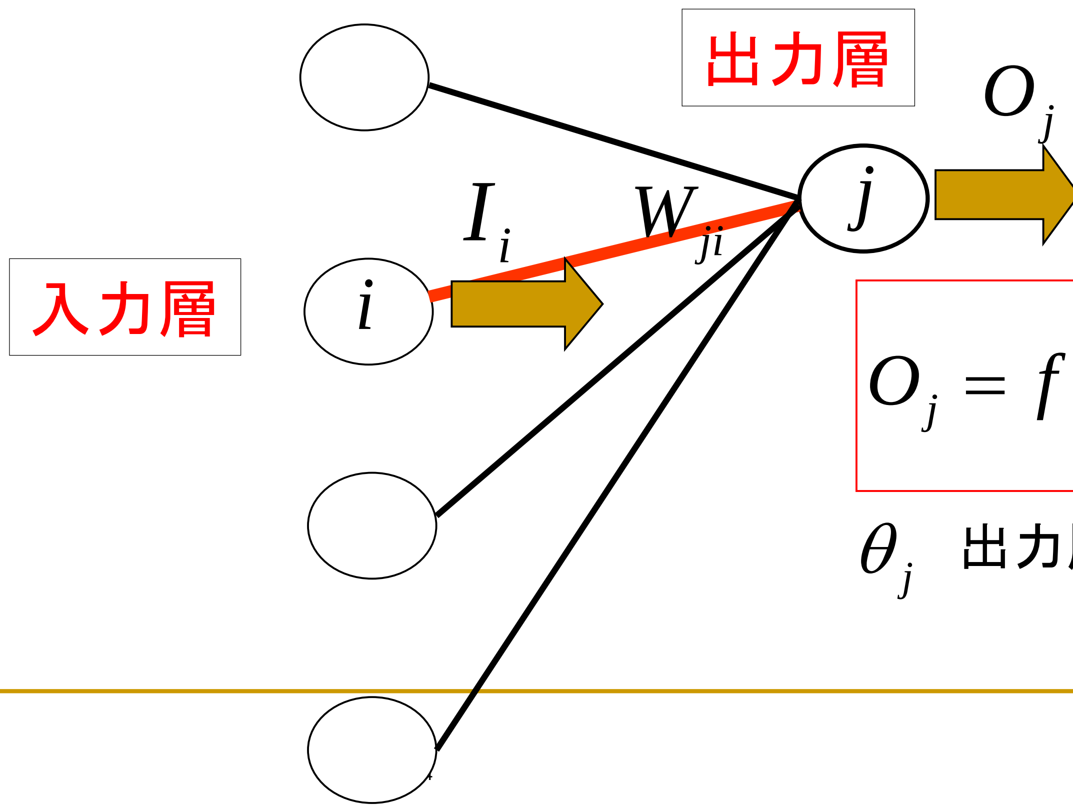
- 入力層の i 番目のニューロン
  - 入力信号 x_i を受け取り，全ての出力層へ出力値 I_i を送る

$$\boxed{I_i = x_i}$$



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

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
- 活性化関数はステップ関数とした場合

↓

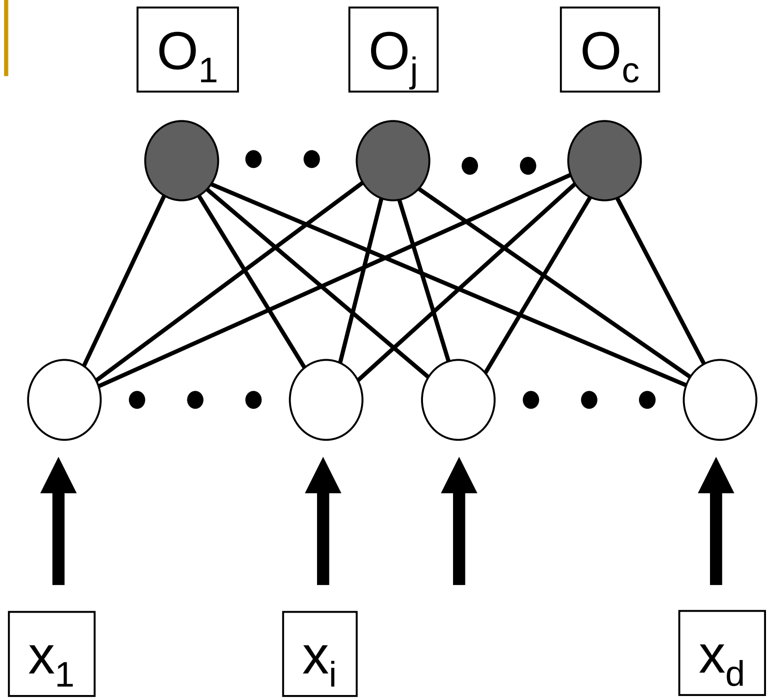
- パーセプトロンと等価
- 二層のフィードフォワード型ネットワークをパーセプトロン*と呼ぶ

*Rosenblattの考案したパーセプトロンは三層構造(後程説明)であり，区別して「単純パーセプトロン」とも呼ばれる

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$\mathbf{x}$（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

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

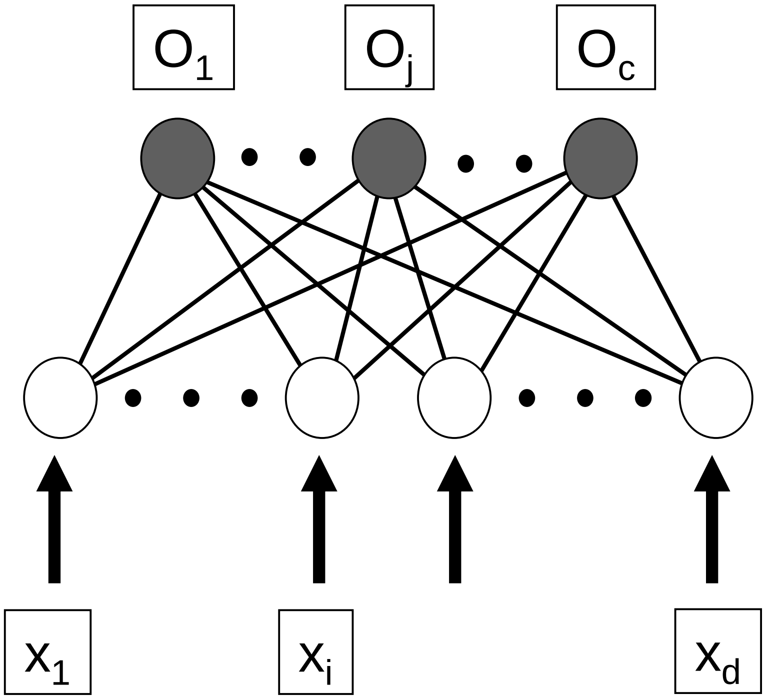
$$\mathbf{x} \in \omega_j$$

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

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

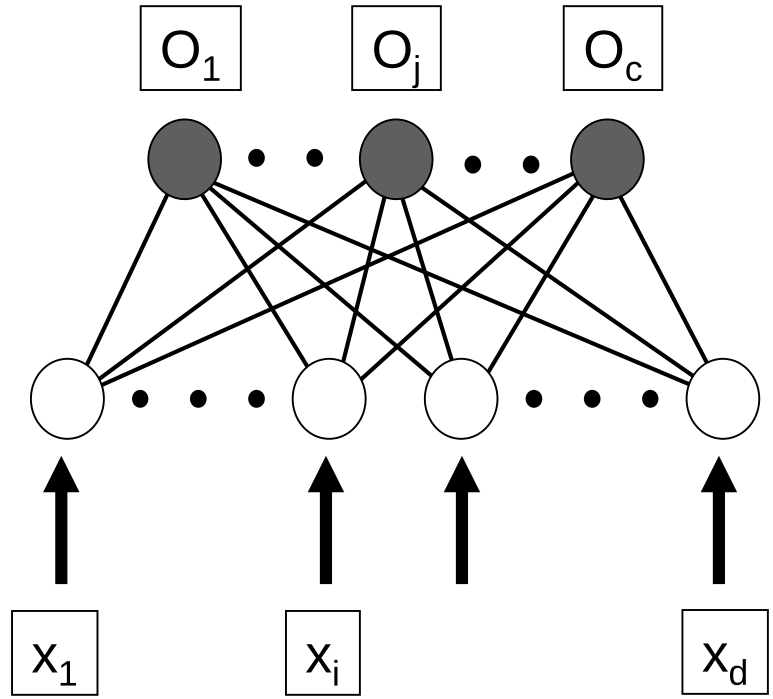


$$\mathbf{x}=(x_1,x_2,\cdots,x_d)^t$$

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

j番目の出力のみ1，他は0

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

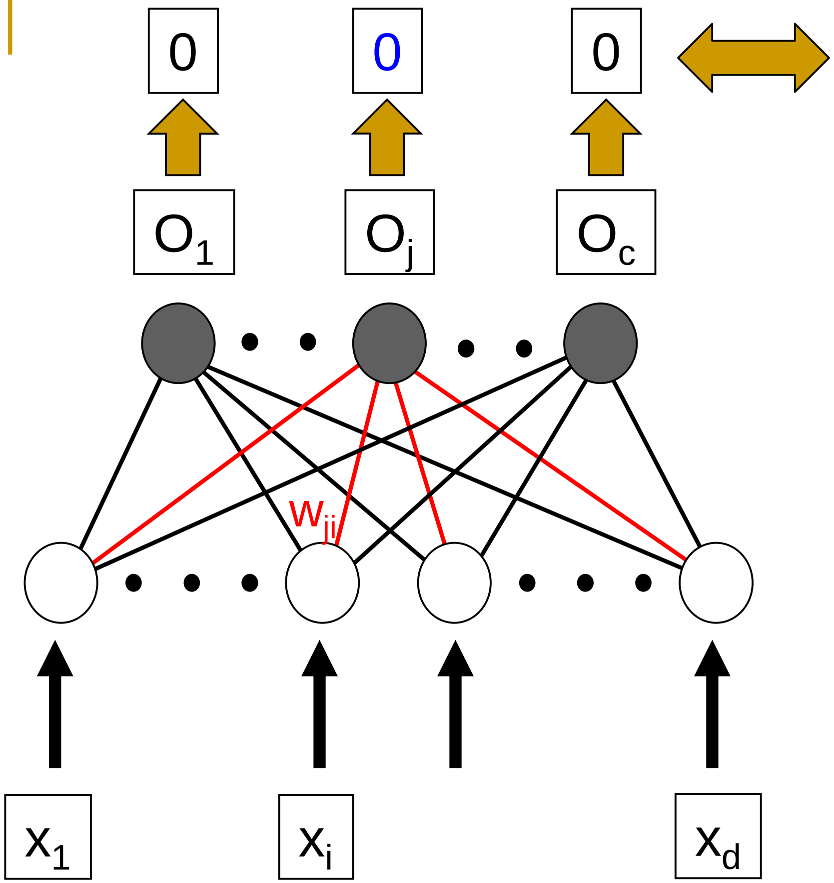


$$\mathbf{x}=(x_1,x_2,\cdots,x_d)^t$$

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

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**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

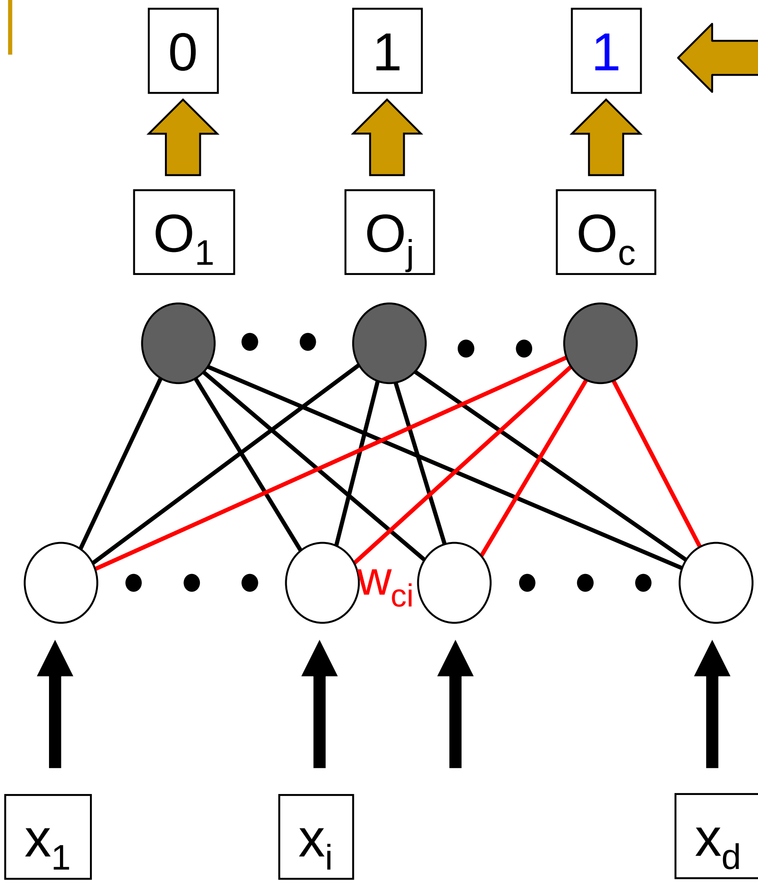
$$\mathbf{w}'_j = \mathbf{w}_j + \rho\mathbf{x}$$

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

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

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

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



j番目の出力のみ1，他は0を出力しなければならない

w_cを更新

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

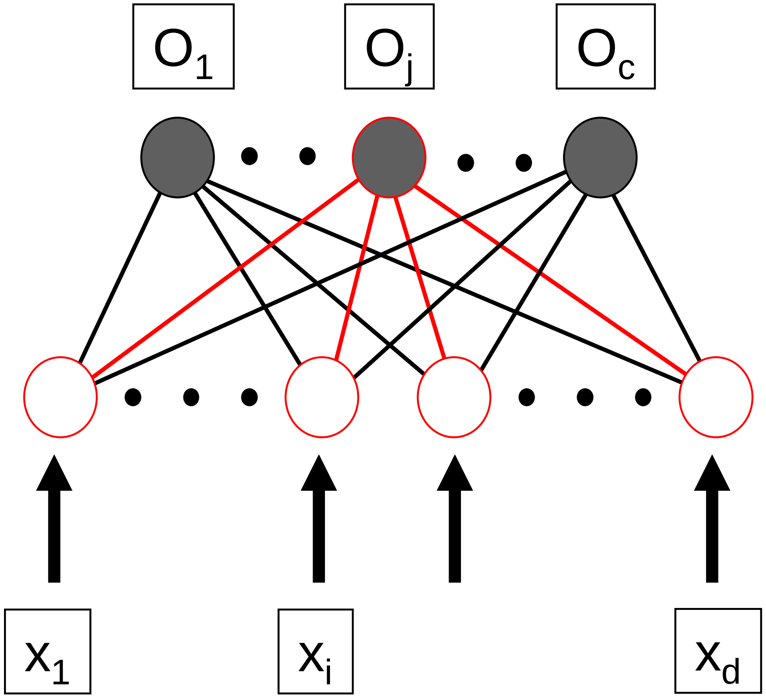
$$w_{ci}^{'} = w_{ci} - \rho x_i$$

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

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

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

学習後のパーセプトロン

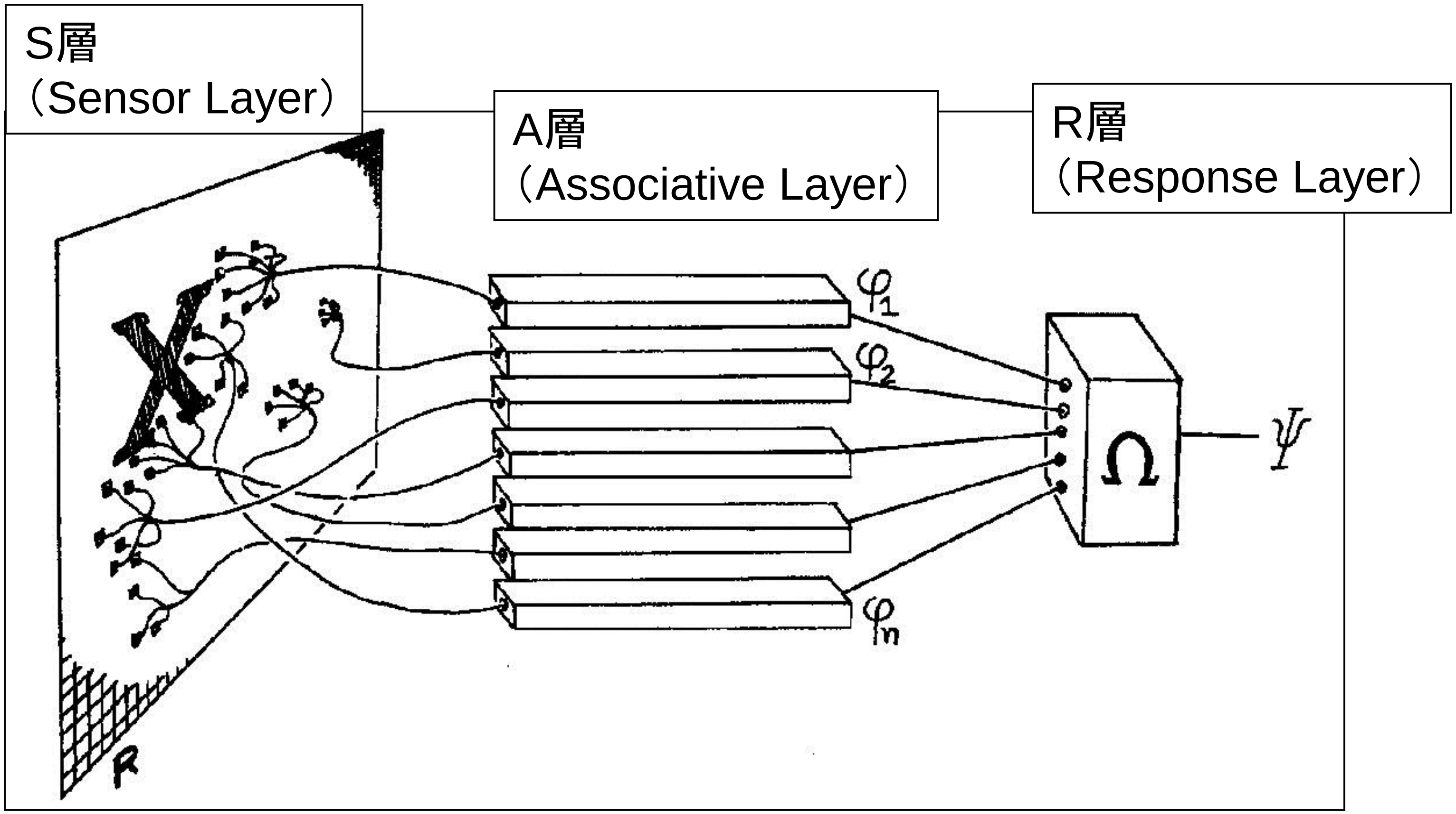


結合係数w_jは線形判別関数g_jの重みベクトルと等価

↓

線形分離可能な問題のみに対応

（オリジナル？）パーセプトロン

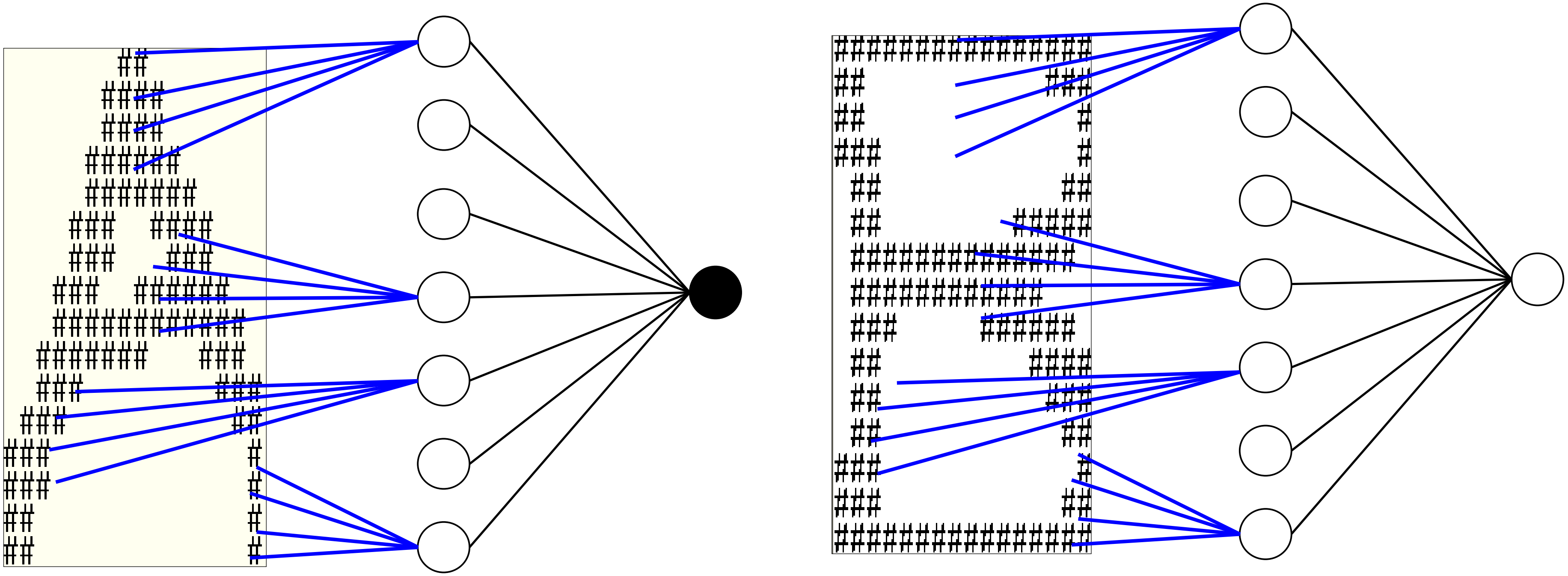


M.ミンスキー，S.パパート:「パーセプトロン」, 中野馨，阪口豊訳，パーソナルメディア（1993）

（オリジナル？）パーセプトロンの構造

- S層，A層，R層の三層構造
- A層とR層が単純パーセプトロンに相当
  - A層→入力層　R層→出力層
- S層には刺激が入力される
- S層とA層はランダムに-1もしくは1の重みによって結合（固定）
- A層とR層間の結合係数を学習する

「A」と「B」を認識するパーセプトロン①

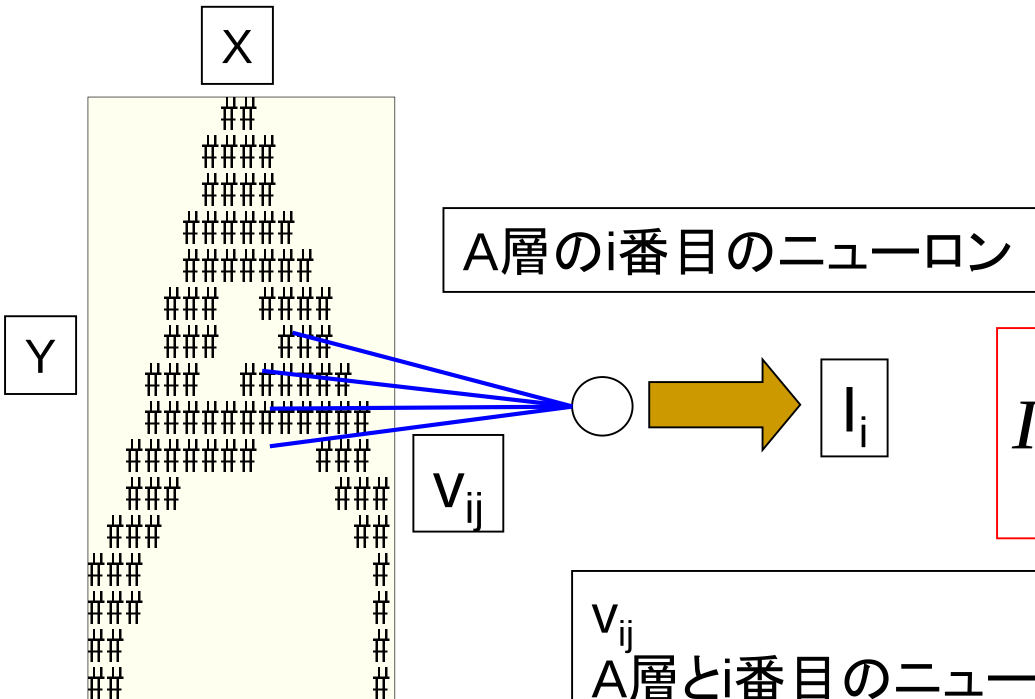


S層　A層　R層

S層とA層との結合係数はランダムで1もしくは-1（固定）

「A」を入力した場合→R層からの出力値は1
「B」を入力した場合→R層からの出力値は0
となるようにA層とR層間の結合係数を学習

「A」と「B」を認識するパーセプトロン②



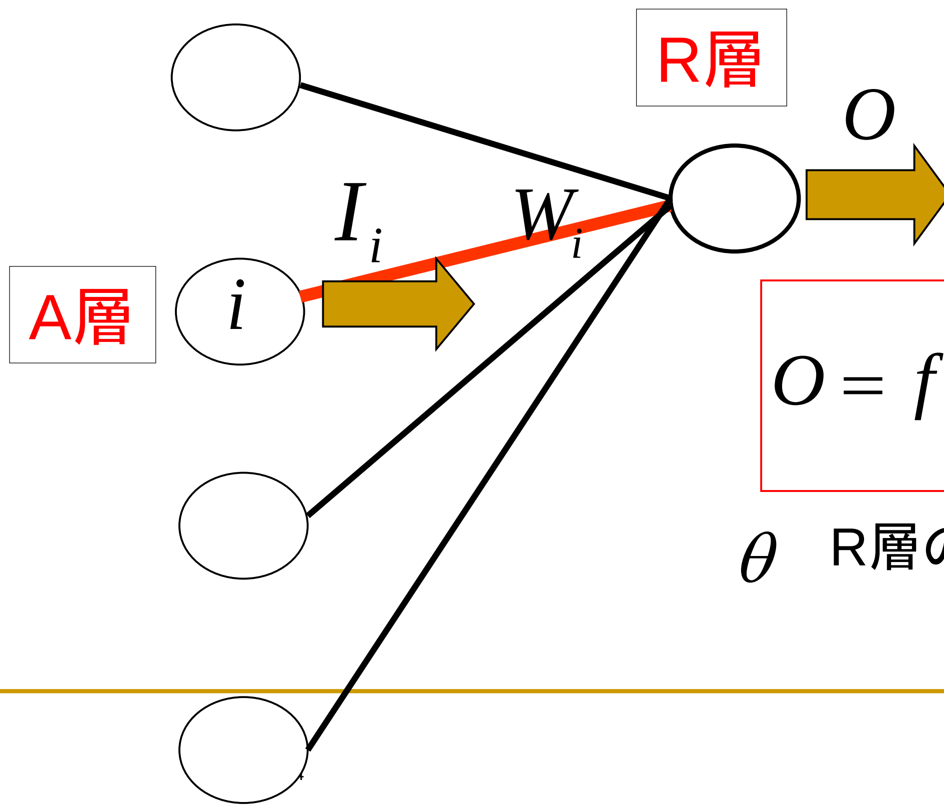
$$I_i = \sum_{j=1}^{XY} v_{ij} s_j$$

S層での入力信号
→ **s**（X×Y次元）

v_{ij}
A層とi番目のニューロンとS層のj番目のニューロンの結合係数
ランダムに1, -1（固定）

「A」と「B」を認識するパーセプトロン③

- R層のニューロン
  - 全てのA層のニューロンから値を受け取り，出力値を計算



$$O = f\left(\sum_{i=1}^{n} W_i I_i - \theta\right)$$

θ　R層のニューロンの閾値

「A」と「B」を認識するパーセプトロン③

- R層の出力値O
  - 1の場合 → 「A」と認識
  - 0の場合 → 「B」と認識
- R層とA層の結合係数w_iをパーセプトロンで学習

線形ニューラルネットワーク

アダライン

デルタルール

その他のモデル

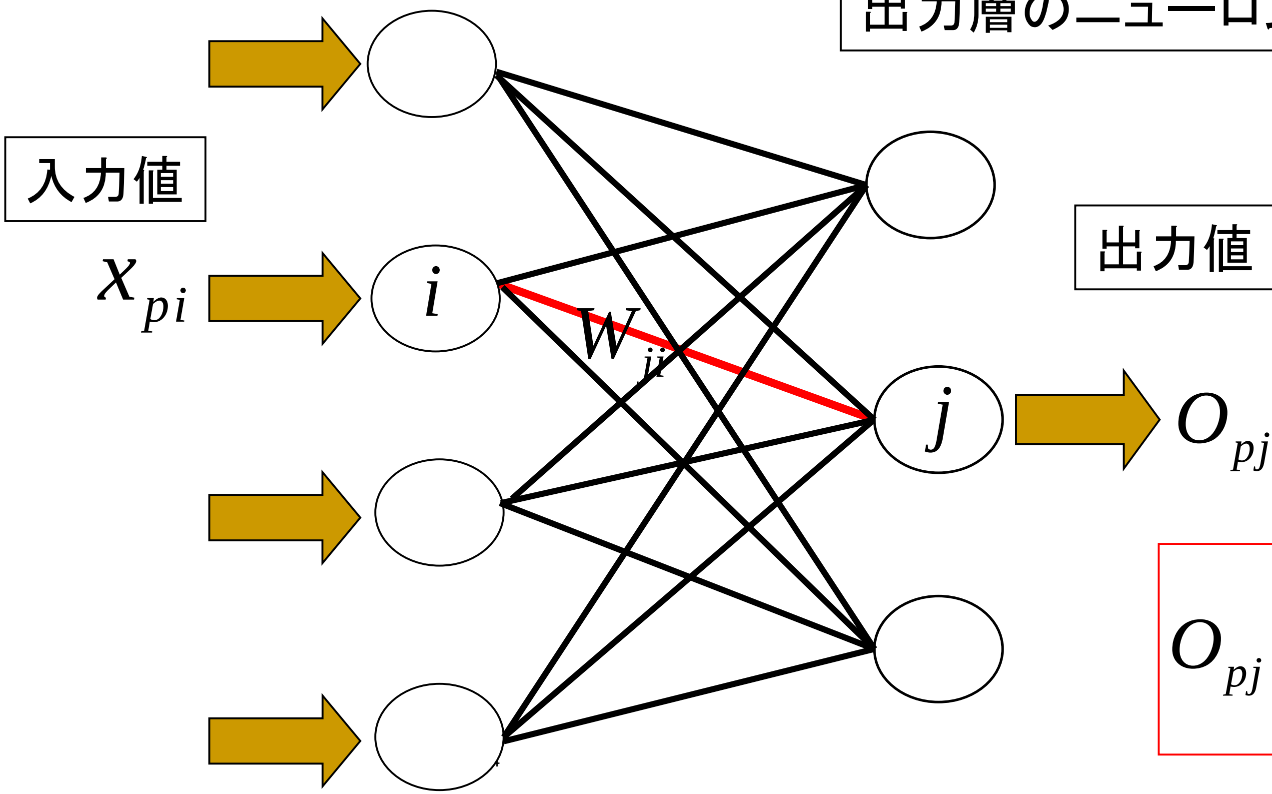
- Adaline
  - adaptive linear neuron
  - B.Widrow and M.E. Hoff（1960）
  - 線形ニューラルネットワーク
- Madaline
  - multilayer adaline
  - 階層型パーセプトロン（次に説明）とほぼ等価

アダラインの構造

線形ニューラルネットワーク*

入力層のニューロン n個（i=1,2,・・・,n）
出力層のニューロン c個（j=1,2,・・・,c）

入力値 m個 x_{pi}
p=1,2,・・・,m



$$O_{pj} = \sum_{i=1}^{n} W_{ji} x_{pi}$$

*出力層の活性化関数を恒等関数とした場合と考えた場合と同じです

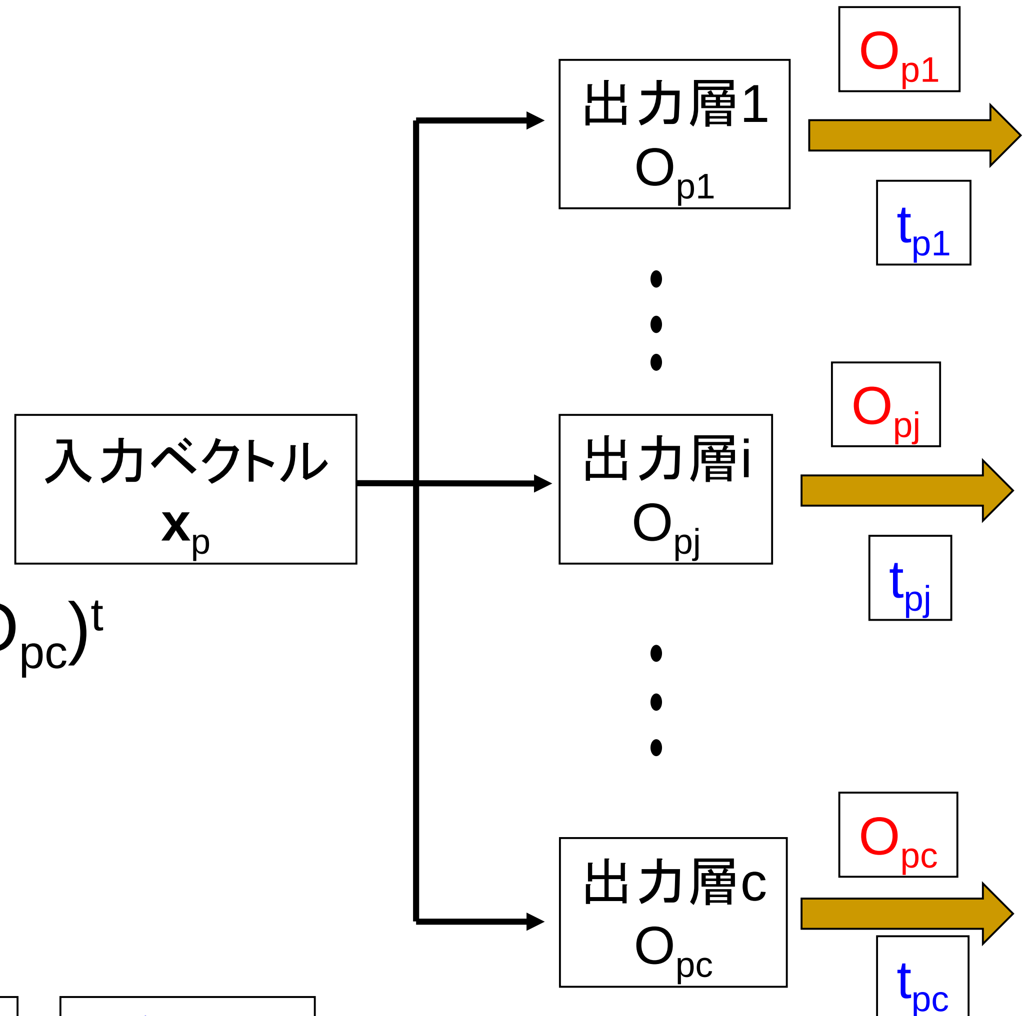
アダラインの学習則①

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

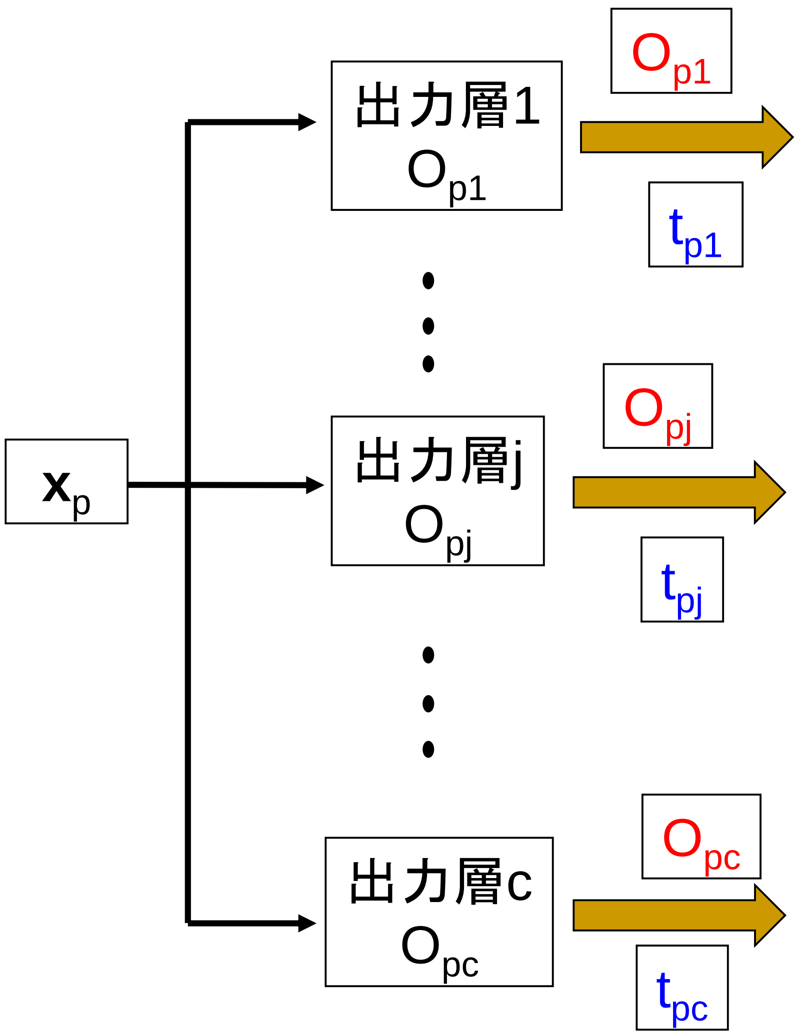
出力

  - $(O_{p1}, O_{p2}, \cdots, O_{pc})^t$
  - $(t_{p1}, t_{p2}, \cdots, t_{pc})^t$

望ましい出力結果　教師信号



アダラインの学習則②



学習パターンx_pに対する出力層jの出力値と教師信号 t_{pj}との誤差

$$\varepsilon_{pj} = O_{pj} - t_{pj}$$

全ての出力層での誤差の自乗和

$$J_p = \sum_{j=1}^{c} \varepsilon_{pj}^2$$

全ての学習パターンの誤差の自乗和

$$J = \sum_{p=1}^{m} J_p = \sum_{p=1}^{m} \sum_{j=1}^{c} \varepsilon_{pj}^2$$

アダラインの学習則③

- 全ての学習パターンの誤差の自乗和が最小となるように結合係数を決定

$$J=\sum_{p=1}^{m}\sum_{j=1}^{c}\varepsilon_{pj}^{2}=\sum_{p=1}^{m}\sum_{j=1}^{c}(O_{pj}-t_{pj})^{2}=\sum_{p=1}^{m}\sum_{j=1}^{c}(\sum_{i=1}^{n}W_{ji}x_{pi}-t_{pj})^{2}$$

最小

各パターンごとの誤差の自乗和

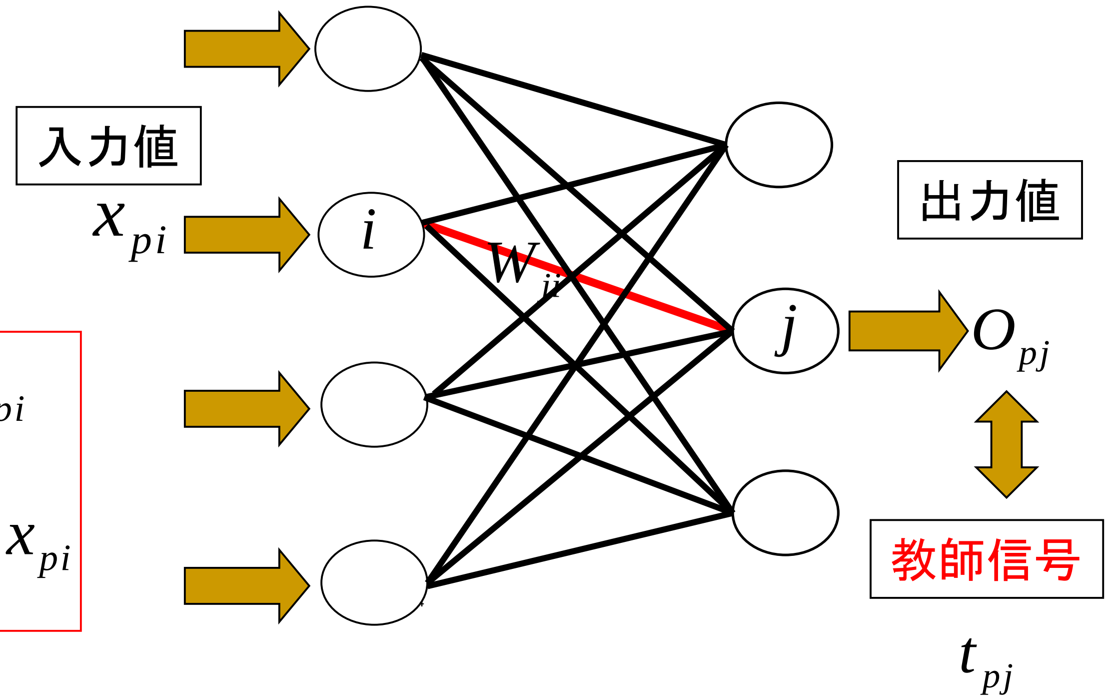
$$J_{p}=\sum_{j=1}^{c}\varepsilon_{pj}^{2}=\sum_{j=1}^{c}(O_{pj}-t_{pj})^{2}$$

最小

デルタルールの導出方法と等価

アダラインの学習則④

- 重みの更新方法



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

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

デルタルール（Widrow-Hoffの学習規則）
線形ニューラルネットワークの学習則

線形識別関数の構築

- 二層のニューラルネットワークによって線形識別関数の重み係数を求めることが可能
  - パーセプトロン
  - アダライン

実習（アダラインの学習）

ニューラルネットワークのプログラム

- デルタルールのプログラムは既に線形識別関数の学習の回に話しました.
- 今後，ニューラルネットワークのプログラミングにおいては，オブジェクト指向でのプログラミングの方が，便利です.
- アダライン（デルタルール）について，オブジェクト指向（完全なカプセル化はしていない）で作成したプログラムについて再度，説明します.

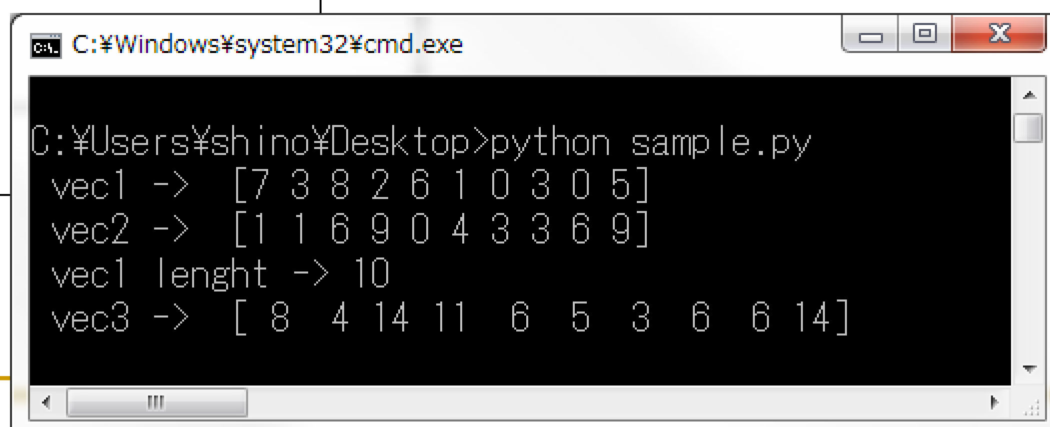
オブジェクト指向(python)

```
class Vec:

    def __init__(self, m):
        self.m = m
        self.v = np.random.randint(0,10,m)

    def len(self):
        return self.m

    def add(self, a, b ):
        self.v = a.v + b.v
```

コンストラクター

メソッドの定義

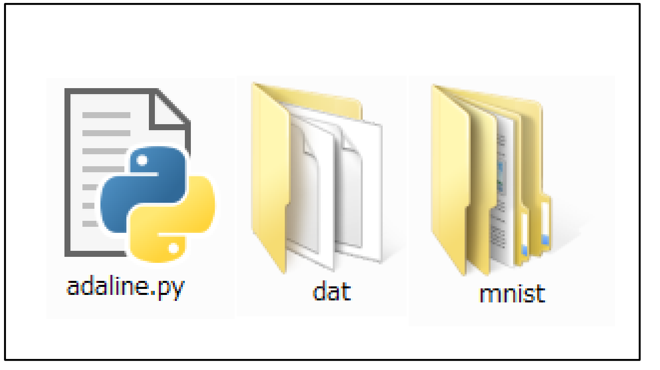
```
vec1 = Vec(10)
vec2 = Vec(10)
print( " vec1 -> " , vec1.v )
print( " vec2 -> " , vec2.v )
print( " vec1 lenght ->" , vec1.len() )

vec3 = Vec(10)
vec3.add( vec1 , vec2 )
print( " vec3 -> " , vec3.v )
```



アダライン(adaline.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 adaline.py t
  - 認識
  - > python adaline.py p

引数をつけて下さい

変数の定義

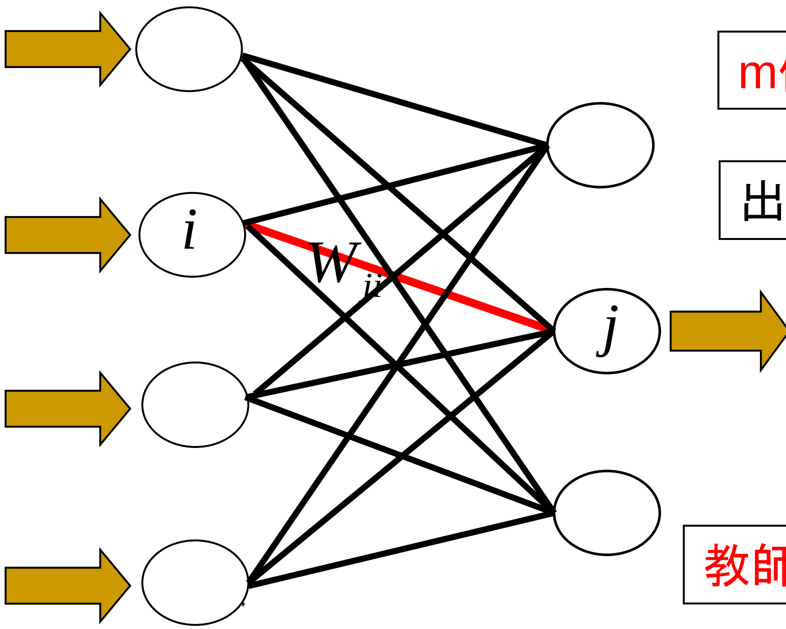
```
# クラス数
class_num = 10

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

# 学習データ数（テストデータ数も同じ）
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）

n個

入力値 x_{pi}



m個

出力値

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

教師信号

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

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

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

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

$$\boxed{\mathbf{o}_p = \mathbf{u}_p}$$

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（活性化関数なし）
        self.out = self.u
```

m：出力層の個数
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

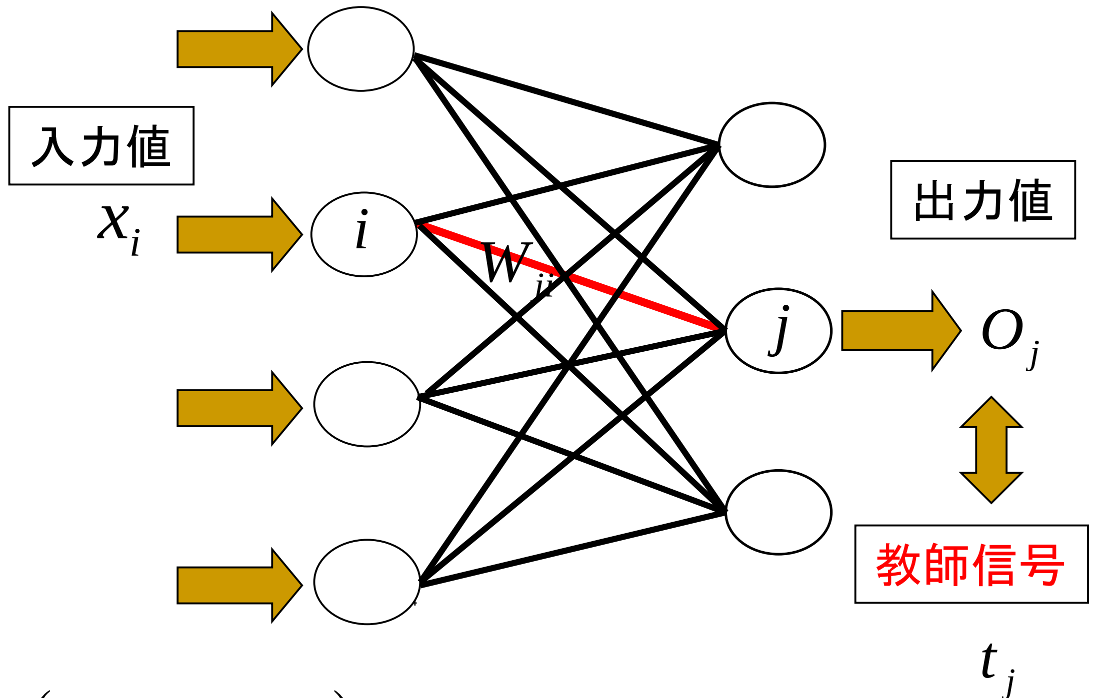
閾値：m次元のベクトル
→ 0.5から0.5の乱数で初期化

x：入力ベクトル（n次元）

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

$$\mathbf{o}_p = \mathbf{u}_p$$

デルタールール（行列計算）①



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

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

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

誤差 $\delta_j = (O_j - t_j)$

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

デルタールール(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 $\boxed{W' = W - \partial W = W - \mathbf{x}^t \mathbf{\Delta}}$

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

*閾値をn+1番目の重みと考えた場合，入力値は1

出力層のクラス②

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

    # 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)

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

t：教師信号（m次元）

$\boldsymbol{\Delta} = (\mathbf{o} - \mathbf{t})$

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

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

$W' = W - \partial W$

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

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

    # 重みの画像化
    for i in range(class_num):
        a = np.reshape( self.w[:,i] , (size,size) )
        plt.imshow(a , interpolation='nearest')
        file = "dat/weight-" + str(i) + ".png"
        plt.savefig(file)
        plt.close()

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」

「dat」の下に重みベクトルを画像化し，保存

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①

```
if __name__ == '__main__':

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

    argvs = sys.argv

    # 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()
```

出力層の個数：class_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
outunit.w：feature×class_num
outunit.b：class_num

flag=0
→学習データの読み込み

メインメソッド②

```
# 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=1
→テストデータの読み込み

学習

```
def Train():

    # エポック数
    epoch = 100
    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
→0,1,2,･･･と順番に入力し，学習した場合，どうなるのか

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

Propagationメソッド
入力値（input_data）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 )
```

Errorメソッド
教師信号（teach）を渡す

```
        # 重みの修正
        outunit.Update_weight()
```

Update_Weightメソッド
重みの更新

```
        # 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/adaline.npz" )
```

Saveメソッド
保存ファイル名（"dat/Adaline.npz"）を渡す

予測

```
# 予測
def Predict():

    # 重みのロード
    outunit.Load( "dat/adaline.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)
```

Loadメソッド
ロードしたいファイル名（"dat/Adaline.npz"）を渡す

入力データ
（1×feature）に変形

```
    # 伝播
    outunit.Propagation( input_data )

    # 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) )
```

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

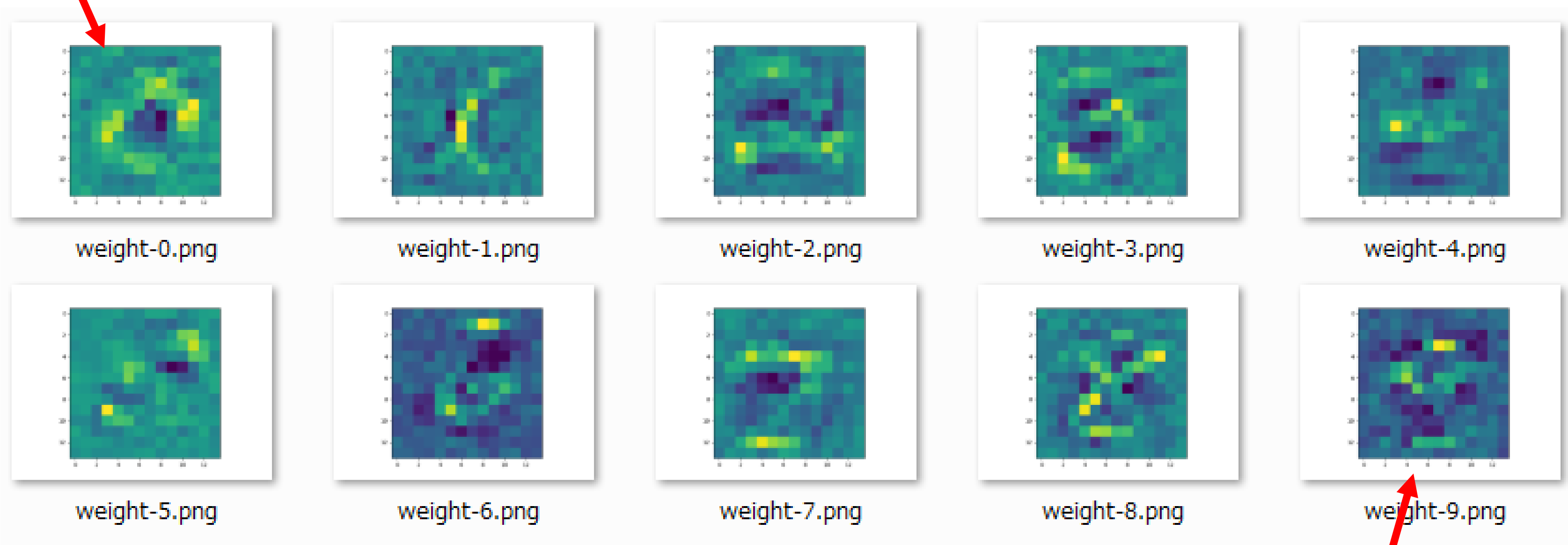
> python adaline.py t

誤差二乗和が下がっていくことを確認

実行(学習)②

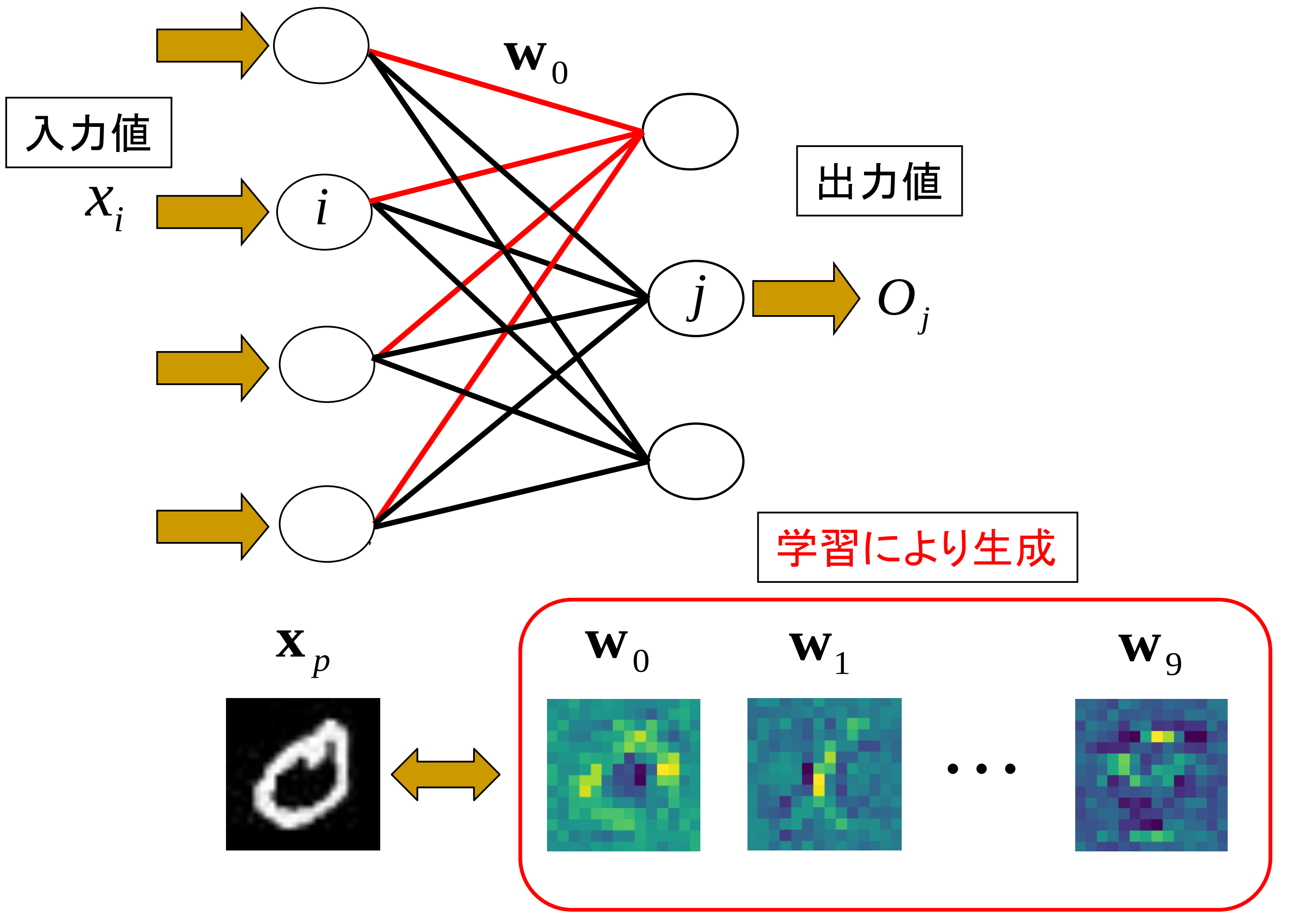
- 重みベクトルの画像化

0番目の出力層と入力層との結合係数



9番目の出力層と入力層との結合係数

ニューラルネットワークの動作



テンプレートマッチングと畳み込み処理（復習）

- 類似度

$$R_p = \cos\theta = \frac{\boxed{\mathbf{t}^t}\boxed{\mathbf{x}_p}}{\|\mathbf{t}\| \cdot \|\mathbf{x}_p\|}$$

- 畳み込み処理

$$g(x,y) = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1} \boxed{h(k,l)}\boxed{f(x+k, y+l)}$$

実行(予測)

> python adaline.py p

宿題⑧

- adaline.pyをオリジナル(?)パーセプトロンに改良しなさい．

- adaline.pyは，A層，R層を対象とした二層のネットワークです．
- S層とA層の結合を追加し，三層のネットワークとして下さい．
  - 結合係数は，ランダムに-1，1と固定して下さい
  - この二つの層間の結合係数は学習しなくてよい．
- 活性化関数にステップ関数を導入して下さい．
- 学習は収束しませんので，適当な回数で停止して下さい．

（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）