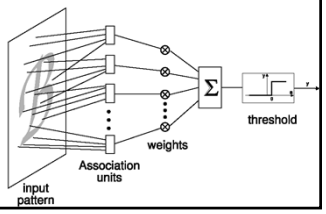
# 情報意味論
# パーセプトロンと多層パーセプトロン

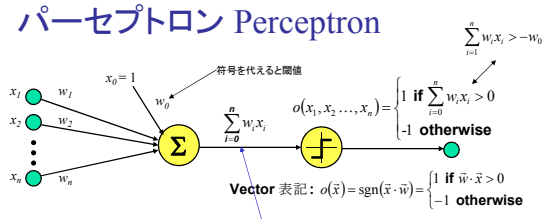
櫻井彰人
慶應義塾大学理工学部

## 多義語: パーセプトロン

- パーセプトロン: 同じ言葉で別のものを指している
  - 線型閾値素子: 次のスライド
  - 元祖パーセプトロン: 下記. これが本当！
  - シグモイド素子: 今回の後半
  - シグモイド素子のネットワーク: 多層パーセプトロンと呼ばれる: 今回の後半
  - 線型閾値素子のネットワーク: 多層パーセプトロン. 稀
- 本講義では, 習慣に従い「間違った」用法に従う
- 元祖パーセプトロン
  - Rosenblatt 1962
  - Minsky and Papert 1969

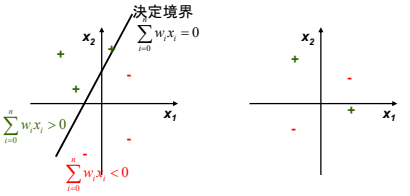



## パーセプトロン Perceptron




- パーセプトロン Perceptron: 単一ニューロンのモデル
  - 別名 線型閾値素子 Linear Threshold Unit (LTU) or Linear Threshold Gate (LTG)
  - 素子への純入力 net input: 線型関数 $\boldsymbol{net} = \sum_{i=0}^{n} w_i x_i$
  - 素子の出力: 純入力に閾値関数 threshold function を施したもの (閾値 threshold $\theta = -w_0$)
    - 純入力に施して出力を得る関数を 活性化関数 activation function と呼ぶ
- パーセプトロンネットワーク Perceptron Networks
  - パーセプトロン同士が 荷重つき結合 weighted links $w_i$ によって繋がっている
  - Multi-Layer Perceptron (MLP): 稀

## パーセプトロンで表現できるもの

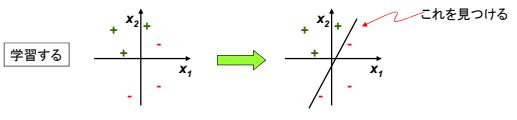



- パーセプトロン: 重要な関数がいくつも簡単に表現できる
  - 論理関数 (McCulloch and Pitts, 1943) の多くのもの
  - e.g., 簡単な荷重で $AND(x_1, x_2)$, $OR(x_1, x_2)$, $NOT(x)$
- 表現できない関数もある
  - 線型分離可能でないもの – 同語反復ですが

## パーセプトロンで何をするか？

他の学習器と同じです

- 学習する
  - 学習データを $\langle \boldsymbol{x}_i, t_i \rangle$ とする ($1 \le i \le$個数)
    - $\boldsymbol{x}_i$ は(高次元)ユークリッド空間の点
    - $t_i$ は +1 or –1
  - パラメータを調節して、$\boldsymbol{x}_i$ が入力されたとき $t_i$ を出力するようにする
- 使用する(予測する)
  - $\boldsymbol{x}_i$ と同じ次元の $\boldsymbol{x}$ に対してそのあるべき出力 $t$ を出力する




## パーセプトロン学習アルゴリズム

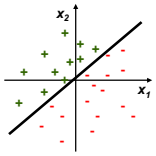
- 比較: Hebbの学習則 Hebbian Learning Rule (Hebb, 1949)
  - アイデア: もし2個の素子が両方とも active ("firing")であれば, 結合荷重は増加する
  - $w_{ij} = w_{ij} + r\, o_i\, o_j$, 但し $r$ は学習係数 learning rate で、定数である
  - 神経生理学的に, ほぼ, 支持されている
  - しかし、収束しない
- パーセプトロン学習アルゴリズム Perceptron Learning Rule (Rosenblatt, 1959)
  - アイデア: 各入力ベクトルに対して出力値が与えられているなら, 荷重を漸進的に更新することにより, 当該出力値が出力できるようになる
  - 2値出力 (Bool値, Boolean-valued) を仮定; 単一パーセプトロン素子
  - $w_i \leftarrow w_i + \Delta w_i$
    $\Delta w_i = r(t - o) x_i$
    但し $t = c(x)$ は目標出力値, $o$ はパーセプトロンの現在の出力値, $r$ は学習係数, 正定数であれば何でも良い. 1でよいので、実は、パーセプトロン学習アルゴリズムでは, $r$ は不要
  - $D$ が線型分離可能 linearly separable であれば, 収束する. $r$ が十分小さいことを条件とする説明もあるがそれは誤り

# パーセプトロン学習アルゴリズム

- 単純な勾配降下 gradient descent アルゴリズムである
  - このアイデアは, 適当な表現を用いれば, 概念学習にも記号学習にも適用可能
- アルゴリズム *Train-Perceptron* ($D \equiv \{<x, t(x) \equiv c(x)>\}$)
  - 荷重 $w_i$ をランダム値に初期化する　　// パーセプトロン時は0に初期化してもよい
  - WHILE 正しい出力をしない事例がある DO
    FOR それぞれの事例 $x \in D$
    **現在の出力 $o(x)$ を計算**
    **FOR $i$ = 1 to $n$**
    **$w_i \leftarrow w_i + r(t - o)x_i$　// perceptron learning rule. r is an any positive number**
- パーセプトロン学習可能性
  - $h \in H$ のときのみ学習可能 - i.e., 線型分離可能 linearly separable (LS) functions
  - Minsky and Papert (1969) *Perceptrons* : 元祖パーセプトロンの表現・学習の限界を示した
    - 注: 素子一個では parity ($n$-変数 XOR: $x_1 \oplus x_2 \oplus \ldots \oplus x_n$) 関数が表現できない, というのは既知
    - e.g., 画像の symmetry, connectedness は(元祖パーセプトロンで)表現できない
    - "*Perceptrons*" のせいで ANN 研究が10年近く遅れたといわれもするが、それはなかろう。

# 線型分離可能

- 定義
  - $f(x) = 1$ if $w_1x_1 + w_2x_2 + \ldots + w_nx_n \geq \theta$, 0 otherwise
  - $\theta$: 閾値
- 線型分離可能か？
  - 注: $D$ が線型分離可能だからといって,
    真の概念 $c(x)$ が線型分離可能とは限らない
  - 選言 disjunction: $c(x) = x_1' \vee x_2' \vee \ldots \vee x_m'$
  - $m$ of $n$: $c(x)$ = at least 3 of ($x_1', x_2', \ldots, x_m'$)
  - 排他的 exclusive *OR* (*XOR*): $c(x) = x_1 \oplus x_2$
  - 一般の DNF: $c(x) = T_1 \vee T_2 \vee \ldots \vee T_m$; $T_i = l_1 \wedge l_1 \wedge \ldots \wedge l_k$
- 表現の変換
  - 線型分離可能でない問題を線型分離可能な問題に変換できるか?
  - それは意味のあることなのか? 現実的なのか?
  - 現実問題の重要な部分を占めるのか?




○ Linearly Separable (LS) Data Set
○
× 
×

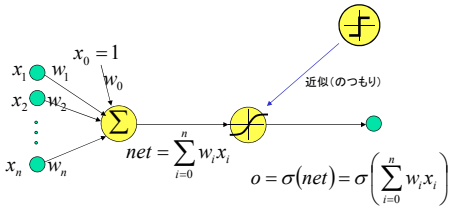
# パーセプトロン学習の収束

- パーセプトロン学習の収束定理
  - 主張: もし訓練データと consistent な荷重集合があれば (i.e., データが線型分離可能なら), パーセプトロン学習アルゴリズムは収束する
  - 証明: 探索空間が限界のある順序をなしている ("楔の幅" が厳密に減少していく) – 参照 Minsky and Papert, 11.2-11.3
  - 注意 1: 収束までの平均時間は?
  - 注意 2: もし線型分離可能でなければどうなるのか?
- パーセプトロン循環定理
  - 主張: 訓練データが線型分離可能でなければ パーセプトロン学習アルゴリズムにより得られる荷重ベクトルは、ある有界集合内に留まる. 荷重が整数ベクトルなら, 有限集合内に留まる.
  - 証明: もし十分に絶対値が大きい荷重ベクトルから始めると, 絶対値は殆ど大きくなれないことが示せる; 訓練事例の次元 $n$ の数学的帰納法による – Minsky and Papert, 11.10
- よりロバストに, またより表現力を上げるには?
  - 目的 1: もっとも良い近似を発見するアルゴリズムの開発
  - 目的 2: 表現の制約を超える新しいアーキテクチャの開発

# ネットワークの形

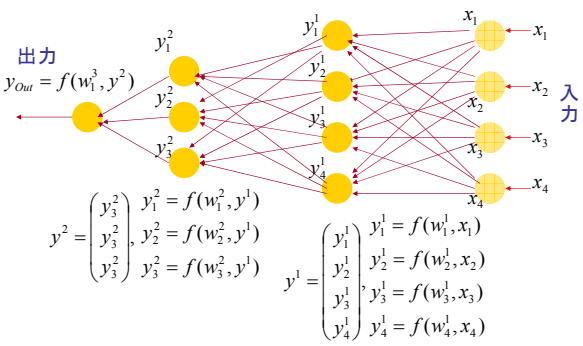
- 階層型（フィードフォワード型）
  - 多層ネットワーク
- 相互結合
  - フィードバック型（ディレイ付き、クロック付き）
  - 純相互結合
- モジュール＆その結合

# シグモイド素子



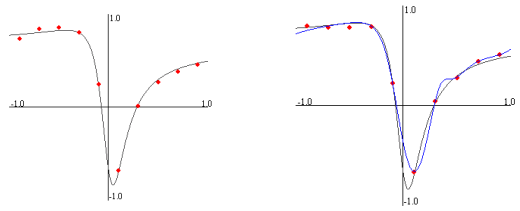

極めて頻繁に使われる σ：　$\sigma(x) \equiv \dfrac{1}{1+e^{-x}}$

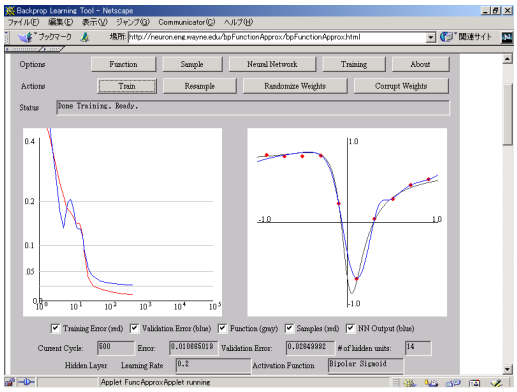
# 基本的な計算



# ニューラルネットの原理

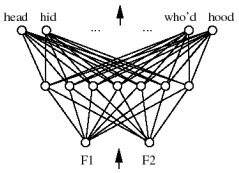
- 一側面：関数近似・回帰である
  - 例：1入力・1出力とする

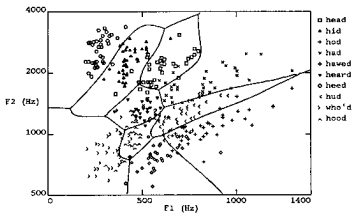





http://neuron.eng.wayne.edu/bpFunctionApprox/bpFunctionApprox.html

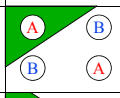
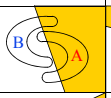
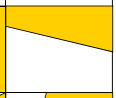
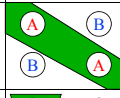
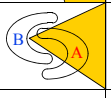
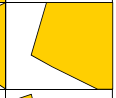
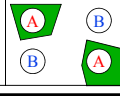
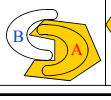
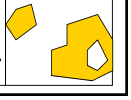
# 実際の決定境界例: 音声認識



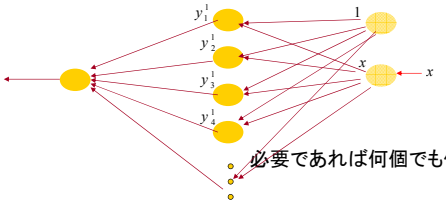

http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/theo-3/www/ml.html

# 領域と境界面

| 構造 | 境界面の形 | 例1<br>対XOR問題 | 例2 | 例3 |
|---|---|---|---|---|
| 中間層なし | 超平面 |  |  |  |
| 中間層2素子 | 2超平面、それらを滑らかにしたもの |  |  |  |
| 中間層多素子 | 任意（但し、素子数に依存） |  |  |  |

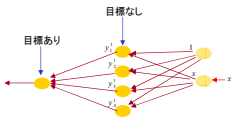
# 近似定理

- ニューラルネットワークの中間素子数を必要なだけ用意できるなら、任意の滑らかな関数を任意の精度で近似することができる




必要であれば何個でも使う

# 学習方法




- パーセプトロン学習法の拡張を考えた
  - 素子は、閾値素子で

パーセプトロン学習のためには

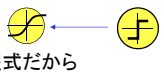
- 各素子で目標との差である「誤差」が必要
- しかし、多層パーセプトロンでは、出力素子以外には、「目標」がない
  - 一つの誤差（失敗）に多くの素子が関与している。
  - どの素子がどれだけ関与しているかを定め、責任（パラメータの修正量）を定める必要がある。
  - ご褒美で考えてもよい。ある利益を得た。貢献者に報酬を配りたい。当然貢献度合いに応じて配りたい。では、貢献度合いをどう定めればよいか？
  - こうした問題は、学習課題では頻繁に発生する。
  - 一般に、credit assignment problem といわれる
- つまり、パーセプトロン学習法は適用できない。
- さて、、、、

## 誤差の最小化 -- シグモイド関数へ

- 基本的な考え方
  - 出力素子における、(実出力 – 目標出力値)$^2$ を最小化しよう
    - この値、実際には、この値の全データ、全出力素子に関する総和、を E(w) と書こう
  - パラメータは、素子間の結合荷重 w
  - E(w) を w で微分(偏微分です)して0と置こう
    - 得られる w に関する方程式を解けばよい
    - パーセプトロンでは、微分できない

$$E(W)=\frac{1}{N}\sum_{i=1}^{N}(F(W,x_i)-t_i)^2$$

- では、微分できるようにしよう
  - 閾値関数をシグモイド関数に



- でも、解けない！ 複雑な非線形方程式だから
  - 数値計算にしよう。それでも、解けない。
  - 反復解法を考えよう

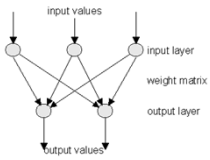
## 実は1980年代まで待った なぜか？

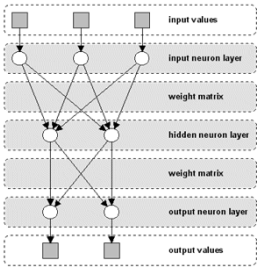
- PDP (Parallel Distributed Computing) という書籍の出版を機に有名になった。しかし、
- 二乗誤差を求め、パラメータで微分し、0と置くのは、もっと前から常識であった。
  - 実は, 類似の技法は, この間, 発見されていた (Amari 1967; Werbos, 1974, "dynamic feedback"; Parker, 1982, "learning logic") ので、再発見だとも言える。貢献は「実際にうまくいく」ということを示したこと
- では、なぜ、こんなに遅くなったのか？ 考えられる理由:
  - 反復法で収束するとは思えなかった。
  - 反復法そのものが大変な計算と思われていた。
  - 考え方が余りにも易しすぎて、極めて簡単な場合以外うまく行くとは思えなかった
    - 甘利研でも修論のテーマぐらいであった。
  - 少し素子数が増えると、当時のコンピュータでは収束しなかった
  - そんなことが脳内で行われているとは思わなかった。

## 適用するネットワークの形状

- アイデアは基本的なので、いろいろな場合に利用できる。
- 基本的には、
  - 階層型(feedforward)
  - 再帰型(recurrent)

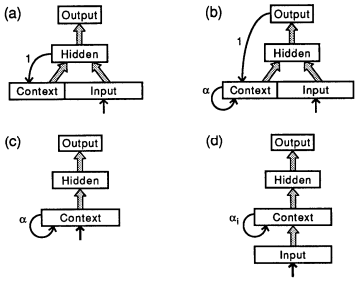
## 再掲: 階層型(多層パーセプトロン)




## 再掲: 相互結合(recurrent)

- 階層型＋フィードバック(離散時間遅れ)



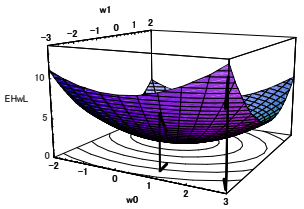

## 再び: 誤差最小化の方法

- 微分して0とおいた方程式を解けばよい！
  本当か？
  - もっとも、まずは、微分できないことには話しにならん
  - つまり、パーセプトロンではだめ。
  - そこで、シグモイド関数にした。

  そうであっても
- 非線形連立方程式になり、到底、解けない
- 反復解法(少しずつ、解を改善していく方法)を考える。すなわち、 $E(W_1)>E(W_2)>E(W_3)>\cdots$
  となる $W_1,W_2,W_3\cdots$ を求める方法を考える

## 反復最小化法

- 様々な方法が提案されている。
- 中でも最も単純なものが、最急降下法
  - 最大値を求めるなら、最急上昇法（あまり使わない）。




最急降下方向と等高線とのなす角度に注目！

## 最急降下法の計算式

- 実際の計算はどうすればよいか？
- 微係数は、最急上昇方向であった！
- そこで、

$$\Delta W = \alpha\left(-\frac{\partial E}{\partial W}\right)$$

$$W \leftarrow W + \Delta W$$

とする。αは学習係数（上手に決めないといけない定数）

## 学部生に戻って

$$E = (t-y)^2 \qquad y = \sum_{i=1}^{n} w_i x_i$$

$$\frac{\partial E}{\partial W} = 2(t-y)\frac{\partial (t-y)}{\partial W} = -2(t-y)x$$

くどいですが $\frac{d}{dx}[f(x)]^2 = 2f(x)\frac{d}{dx}f(x)$

$$\frac{\partial (t-y)}{\partial W} = -\frac{\partial y}{\partial W} = -\frac{\partial\left(\sum_{i=1}^{n} w_i x_i\right)}{\partial W} = -x$$

(t は目標値であり、W に依存しない)

（x はベクトルです）

## もう少し

$$W \leftarrow W + \alpha\left(-\frac{\partial E}{\partial W}\right) \quad \text{かつ} \quad \frac{\partial E}{\partial W} = -2(t-y)x$$

であるから、2α を α と書き換えれば

$$W \leftarrow W + \Delta W \quad \text{かつ} \quad \Delta W = \alpha(t-y)x$$

この形の学習則は、デルタ則とも呼ばれる（Widrow-Hoff 則とも呼ばれる）

## Perceptron 学習則との関係

- しかし、学習則だけみてみると, perceptron 学習則

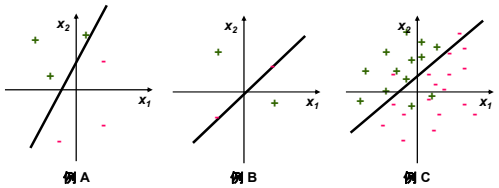
$$W \leftarrow W + \Delta W \quad \text{かつ} \quad \Delta W = \begin{cases} x & \text{if } t \neq y \text{ and } t = 1 \\ -x & \text{if } t \neq y \text{ and } t = -1 \\ 0 & \text{if } t = y \end{cases}$$

は, 下記において, $t = \pm 1, y = \pm 1$ という状況で、α=1 /2 とおいたもの

$$W \leftarrow W + \Delta W \quad \text{かつ} \quad \Delta W = \alpha(t-y)x$$

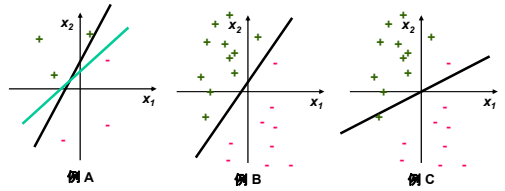
- しかし閾値関数は微分不能なため、perceptron 学習則の妥当性は、二乗誤差最小化では説明できない。

## デルタ規則とperceptron学習則
### シグモイド素子と閾値素子




- 線型分離可能: 完全な分類ができる
  - 例 A: パーセプトロン学習アルゴリズムが収束
- 線型分離不能: 近似できるのみ
  - 例 B: 線型分離不能; デルタ規則は収束, しかし3個正解よりはよくならない
  - 例 C: 線型分離不能; デルタ規則でよい結果

## デルタ規則とperceptron学習則
### 線形素子と閾値素子




- 線型分離可能: 同じ分類をする
  - 例 A: パーセプトロン学習則もデルタ規則（線形素子）も同じ分類をする
    ただし、結果は異なるので、未知入力に対する動作は異なる可能性あり
- 線型分離可能: 異なる分類をする。線形素子の場合、全体のバランスをとる
  - 例 B: 線型分離可能; Perceptron学習則は正解通り分類
  - 例 C: 線型分離不能; デルタ規則は正解通りは分類しない。バランスはよい？

## 活性化関数が恒等関数以外のとき

$$\Delta W_s \leftarrow \alpha(t-y)f'(y_{in})x_s$$
$$W \leftarrow W + \Delta W_s$$

$\Delta W_s$ サンプル s に対する荷重の更新量

$\alpha$ 学習率 (スカラー, 多くの場合定数)

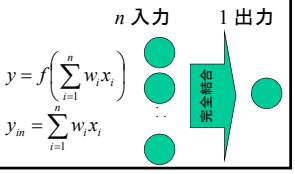
$t$ 目標値

$x_s$ サンプル s

$y_{in} = \sum_{i=1}^{n} w_i x_i$

$f$ 微分可能な関数

$y = f(y_{in})$ 実際の出力値




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

$y_{in} = \sum_{i=1}^{n} w_i x_i$

## もう一度

$$E = (t-y)^2 \qquad y = f(y_{in}),\ y_{in} = \sum_{i=1}^{n} w_i x_i$$

$$\frac{\partial E}{\partial W} = 2(t-y)\frac{\partial (t-y)}{\partial W}$$

$$= -2(t-y)f'(y_{in})x$$

$$\frac{\partial (t-y)}{\partial W} = -\frac{\partial y}{\partial W}$$
$$= -\frac{\partial f(y_{in})}{\partial W}$$
$$= -\frac{\partial f(y_{in})}{\partial y_{in}}\frac{\partial y_{in}}{\partial W}$$
$$= -f'(y_{in})\frac{\partial \left(\sum_{i=1}^{n} w_i x_i\right)}{\partial W}$$
$$= -f'(y_{in})x$$ (x はベクトルです)

## 前と全く同様に

$$W \leftarrow W + \alpha\left(-\frac{\partial E}{\partial W}\right) \quad \text{かつ} \quad \frac{\partial E}{\partial W} = -2(t-y)f'(y_{in})x$$

であるから、2α を α と書き換えれば

$$W \leftarrow W + \Delta W \quad \text{かつ} \quad \Delta W = \alpha(t-y)f'(y_{in})x$$

## ところで

- これまでは、サンプルが一個与えられたときの、荷重更新を考えてきた。
- しかし、これは、おかしい。
- なぜなら、誤差というものは、与えられたサンプル全体に対する誤差を考えないと意味がない
- なぜなら、あるサンプル $x_1$ に関する誤差を減少させた結果、他のサンプル $x_2$ に対する誤差が増加してしまい、結果として、全体誤差は増加してしまう可能性があるからである
- であるから、

  $E = (t-y)^2$ ではなく $E = \sum_s (t_s - y_s)^2$

  そして、

  $W \leftarrow W + \Delta W$ かつ $\Delta W = \alpha \sum_s (t_s - y_s) f'(y_{in,s}) x_s$

  であるべき（これを batch mode という（vs. online mode））

## 正しい議論か？

- 完全に正しい。そしてこの時、αが定数でなく、学習ステップを繰り返す間に、適度な速度で α→0 となるなら、

  $W \leftarrow W + \Delta W$ かつ $\Delta W = \alpha \sum_s (t_s - y_s) f'(y_{in,s}) x_s$

  によって、$E = \sum_s (t_s - y_s)^2$ は（局所）最小値となることが示される
- では、batch mode の方がいいのか？
- 話はそう簡単ではない。
- 実は、online mode の方が, 一般に, より小さい E を得ることが経験上知られている。

# 出力素子が複数個のとき

$$\Delta W_{s,j} \leftarrow \alpha(t_j - y_j) f'(y_{in,j}) x_s$$
$$W_j \leftarrow W_j + \Delta W_{s,j}$$

$\Delta W_{s,j}$ サンプル s に対する第j出力素子への荷重の更新量

$\alpha$ 学習率 (スカラー, 多くの場合定数)

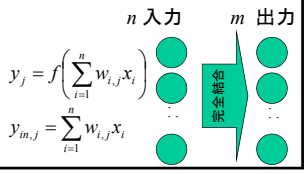
$t_j$ 第j出力素子の目標値

$x_s$ サンプル s

$y_{in,j} = \sum_{i=1}^{n} w_{i,j} x_i$

$f$ 微分可能な関数

$y_j = f(y_{in,j})$ 実際の出力値




# 前と全く同様に

$$W_j \leftarrow W_j + \alpha\left(-\frac{\partial E}{\partial W_j}\right) \quad \text{かつ} \quad \frac{\partial E}{\partial W_j} = -2(t_j - y_j) f'(y_{in,j}) x$$

であるから、2α を α と書き換えれば

$$W_j \leftarrow W_j + \Delta W_j \quad \text{かつ} \quad \Delta W_j = \alpha(t_j - y_j) f'(y_{in,j}) x$$

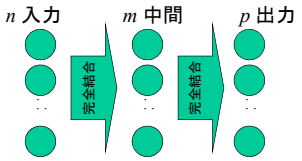
# 多層のネットワークでは?

$\Delta W_j = \alpha(t_j - y_j) f'(y_{in,j}) x$

- 各中間素子での荷重更新量をどうやって求めればよいか
- perceptron 学習則やデルタ規則を見れば分かるが、荷重の更新量を求めるためには、誤差が必要である、そのためには、目標出力値が必要である。
- ところが、中間素子には、目標出力値がない（与えられていない！）
  - 実出力値は、勿論、ある
- つまり、ネットワーク全体では（従って、出力素子では）誤差が定義できるが、個々の中間素子における誤差は定義できない、すなわち、全体誤差の中間素子への割り振り方が分からない。
- このように、ある系全体での利得や損失が分かったとき、その貢献・負担を各構成員にどのように分配すべきかという問題は、いろいろな場面で発生する。
- この問題は、credit assignment の問題といわれる。
- 中間層があるNNで、credit assignment 問題が発生したのである

貢献に対する賞賛・評価、credit titleのcreditと同じ

# Credit Assignment 問題



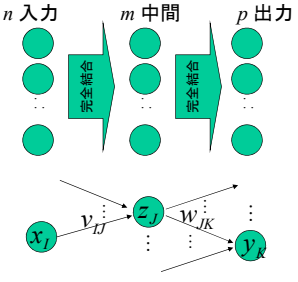

- このように、ある系全体での利得や損失が分かったとき、その貢献・負担を各構成員にどのように分配すべきかという問題は、いろいろな場面で発生する。
- この問題は、credit assignment の問題といわれる。
- 中間層があるNNで、credit assignment 問題が発生したのである

# シグモイド素子を使えば

- しかし、シグモイド素子のように、微分可能な活性化関数を持つ場合には、別の考え方があり、それに従えば自動的に credit assignment の問題は解消する
- すなわち、デルタ規則のときと全く同様に、

$$E = \sum_s (t_s - y_s)^2$$

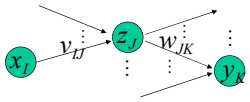
とおき、例えば、最急降下法を適用すればよい




以前は, $E = (t - y)^2$ ただし: $t$ 目標出力, $y$ 実出力

今回は、$p$ 出力を考える。ただし、サンプル数は1個としておく:

$$E = \sum_{k=1}^{p} [t_k - y_k]^2$$




$$E=\sum_{k=1}^{p}[t_k-y_k]^2$$

$$\frac{\partial E}{\partial v_{IJ}}=-2\sum_{k=1}^{p}\left\{[t_k-y_k]\frac{\partial y_k}{\partial v_{IJ}}\right\}$$

$$\frac{\partial y_K}{\partial v_{IJ}}=\frac{\partial f(y_{in,K})}{\partial v_{IJ}}=\frac{\partial f(y_{in,K})}{\partial y_{in,K}}\frac{\partial y_{in,K}}{\partial v_{IJ}}=f'(y_{in,K})\frac{\partial y_{in,K}}{\partial v_{IJ}}$$

$$=-2\sum_{k=1}^{p}\left\{[t_k-y_k]f'(y_{in,k})\frac{\partial y_{in,k}}{\partial v_{IJ}}\right\}$$

表記を簡潔にするため, $\delta_k=[t_k-y_k]f'(y_{in,k})$

$$=-2\sum_{k=1}^{p}\left\{\delta_k\frac{\partial y_{in,k}}{\partial v_{IJ}}\right\}$$

(これは直接計算する)

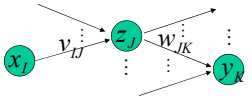
$$\frac{\partial E}{\partial v_{IJ}}=-2\sum_{k=1}^{p}\left\{\delta_k\frac{\partial y_{in,k}}{\partial v_{IJ}}\right\}$$

$$E=\sum_{k=1}^{p}[t_k-y_k]^2$$

$$\delta_k=[t_k-y_k]f'(y_{in,k})$$

$$\frac{\partial y_{in,K}}{\partial v_{IJ}}=\frac{\partial\sum_{j=1}^{m}z_j w_{jK}}{\partial v_{IJ}}=w_{JK}\frac{\partial z_J}{\partial v_{IJ}}$$

$$\frac{\partial z_J}{\partial v_{IJ}}=\frac{\partial f(z_{in,j})}{\partial v_{IJ}}=f'(z_{in,j})\frac{\partial z_{in,J}}{\partial v_{IJ}}=f'(z_{in,j})\frac{\partial\sum_{i=1}^{n}x_i v_{iJ}}{\partial v_{IJ}}=f'(z_{in,j})x_I$$




$$\frac{\partial E}{\partial v_{IJ}}=-2\sum_{k=1}^{p}\left\{\delta_k\frac{\partial y_{in,k}}{\partial v_{IJ}}\right\}$$

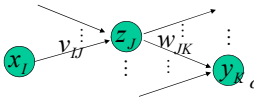
$$=-2\sum_{k=1}^{p}\left\{\delta_k w_{Jk}\frac{\partial z_J}{\partial v_{IJ}}\right\}$$

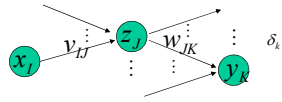
$$=-2\sum_{k=1}^{p}\left\{\delta_k w_{Jk}f'(z_{in,J})x_I\right\}$$

$$=-2\sum_{k=1}^{p}\left\{\delta_k w_{Jk}\right\}f'(z_{in,J})x_I$$

$$=(-2)(x_I)(f'(z_{in,J}))\left(\sum_{k=1}^{p}w_{Jk}\delta_k\right)$$

$E=\sum_{k=1}^{p}[t_k-y_k]^2$



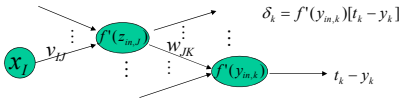

$$\frac{\partial E}{\partial v_{IJ}}=(-2)(x_I)(f'(z_{in,J}))\left(\sum_{k=1}^{p}w_{Jk}\delta_k\right)$$

$$\Delta v_{IJ}=-\alpha\frac{\partial E}{\partial v_{IJ}}$$

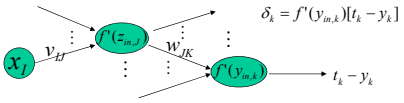
最急降下: $v_{IJ}\leftarrow v_{IJ}+\Delta v_{IJ}$

2αをαと書き換えると

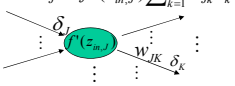
$$\Delta v_{IJ}=\alpha(x_I)(f'(z_{in,J}))\left(\sum_{k=1}^{p}w_{Jk}\delta_k\right)$$



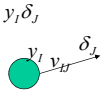

$$\Delta v_{IJ}=\alpha(x_I)(f'(z_{in,J}))\left(\sum_{k=1}^{p}w_{Jk}\delta_k\right)$$



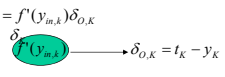

$$\delta_J=f'(z_{in,J})\sum_{k=1}^{p}w_{Jk}\delta_k$$




$$\Delta v_{IJ}=\alpha\, y_I\delta_J$$

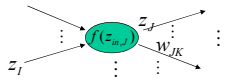



$$\delta_k=f'(y_{in,k})\delta_{O,K}$$




$$\sum_{i=1}^{n}z_i\rightarrow z_{in,J},\ f(z_{in,J})\rightarrow z_J$$

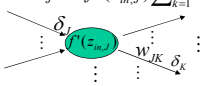
forward propagation




z が当該素子の出力である

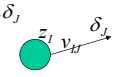
$$\delta_J\leftarrow f'(z_{in,J})\sum_{k=1}^{p}w_{Jk}\delta_k$$

backword propagation

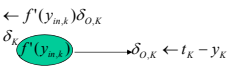



δ が誤差への貢献(?)分である

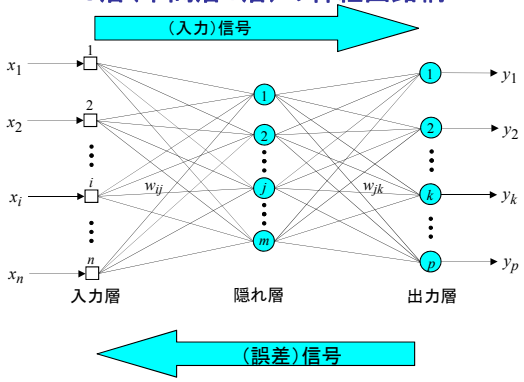
$$\Delta v_{IJ}=\alpha\, z_I\delta_J$$

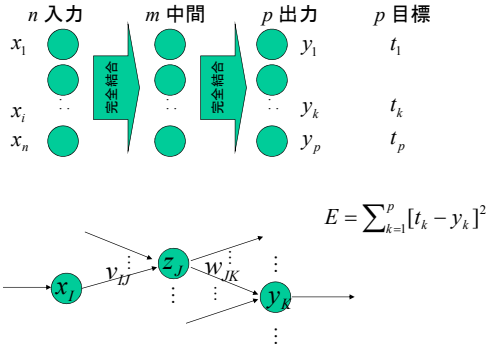



$$\delta_k\leftarrow f'(y_{in,k})\delta_{O,K}$$



## 3層（中間層1層）の神経回路網

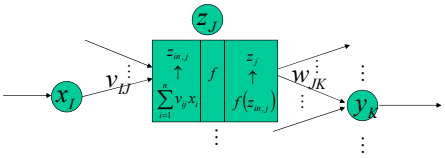



## 前進：出力値の計算

$$z_{in,j} = \sum_{i=1}^{n} v_{ij} x_i$$

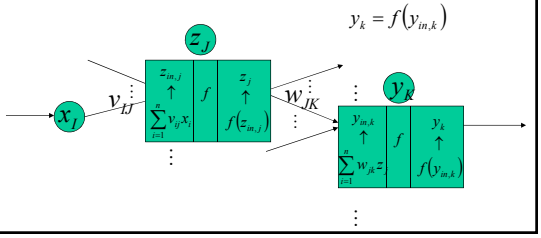
$$z_j = f(z_{in,j})$$



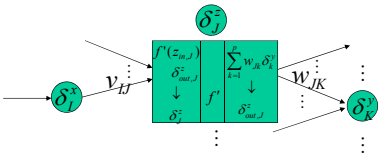

## 前進：出力値の計算

$$y_{in,k} = \sum_{j=1}^{m} w_{jk} z_j$$

$$y_k = f(y_{in,k})$$




## 後退：誤差値の計算

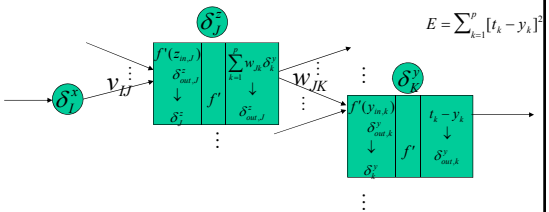



$$\delta^z_{out,J} = \sum_{k=1}^{p} w_{Jk} \delta^y_k$$

$$\delta^z_J = f'(z_{in,J}) \delta^z_{out,J}$$

わざわざ記憶する必要はない

## 後退：誤差値の計算

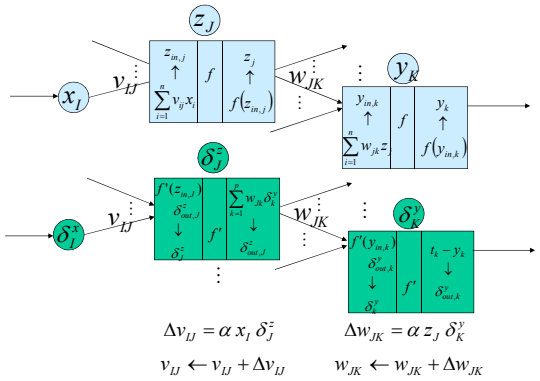



$$\delta^y_{out,k} = t_k - y_k$$

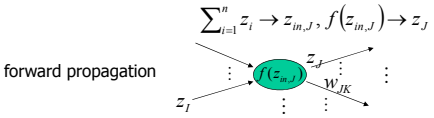
$$\delta^y_k = f'(y_{in,k}) \delta^y_{out,k}$$

わざわざ記憶する必要はない

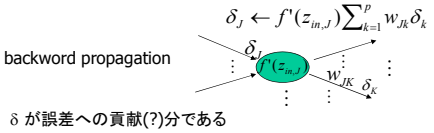
## 修正量の計算



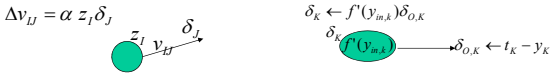

まとめ： 講義スライドから




z が当該素子の出力である




δ が誤差への貢献(?)分である




## まとめ1: 教師付き学習

- 結合荷重の求め方
  - 学習データを $\{(x_t, y_t) | 1 \le t \le N\}$ とする
  - またネットワークの入出力関係を $y = F(W, x)$
  - 誤差関数を設定する。通常は、

$$E(W) = \frac{1}{N} \sum_{t=1}^{N} (F(W, x_t) - y_t)^2$$

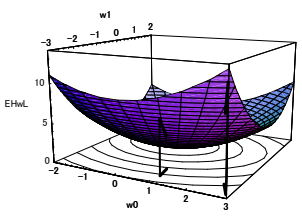
  - この*E*を最小化する*W*を求めればよい

## まとめ2: 誤差最小化の方法

- 微分して0とおいた方程式を解けばよい！本当か？
  - まずは、微分できないことには話しにならん
  - パーセプトロンではだめ。
- 非線形連立方程式になり、到底、解けない
- 反復解法（少しずつ、解を改善していく方法）を考える。すなわち、 $E(W_1) > E(W_2) > E(W_3) > \cdots$ となる $W_1, W_2, W_3 \cdots$ を求める方法を考える

## まとめ3: 反復最小化法

- 様々な方法が提案されている。
- 中でも最も単純なものが、最急降下法
  - 最大値を求めるなら、最急上昇法（あまり使わない）。




最急降下方向と等高線とのなす角度に注目！

## まとめ4: 反復最小化方法の計算式

- 実際の計算はどうすればよいか？
- 微係数は、最急上昇方向であった！
- そこで、

$$\Delta w_i^j = -\eta \cdot \frac{\partial E}{\partial w_i^j}(W)$$

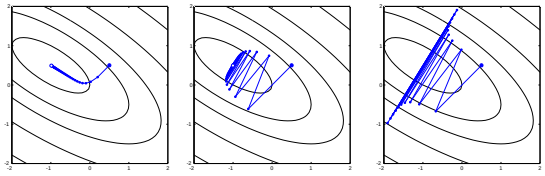
$$w_i^{j,new} = w_i^j + \Delta w_i^j$$

とする。ηは学習係数（上手に決めないといけない定数）

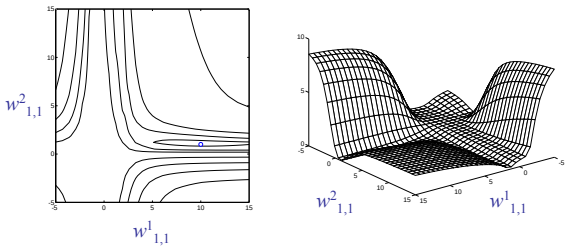
## 誤差逆伝播法の収束

- 大域的な最適解への収束は保証されない
  - 比較: パーセプトロンの収束 (最適な $h \in H$ に, 但し $h \in H$ なる条件下; i.e., 線型分離可能)
  - ある局所最適解 (まあ大域的最適解ではなかろう) へ近づいて行く
  - backprop (BP) に対する改善(かもしれない)
    - 慣性項 ( 荷重更新規則を多少変更): 浅い局所最小解はスキップするかも
      - $\Delta W^{new} \leftarrow \Delta W + \alpha \Delta W^{new}$ ; 加速係数αは1より小さい定数
    - 確率的最急降下 stochastic gradient descent : 局所解に捕まる確率が低下
    - 複数個のネットを異なる荷重値で初期化; うまく混合する
  - フィードフォワードネットワークの改善
    - ANNs のベイズ学習 Bayesian learning (e.g., simulated annealing)
- 収束過程
  - 0に近い初期値, i.e., 線型に近いネットワークから開始, 徐々に非線形ネットワークへ: 未解明
- プラトーと収束速度
  - プラトーの解消: 自然勾配法 natural gradient [Amari, 1998]

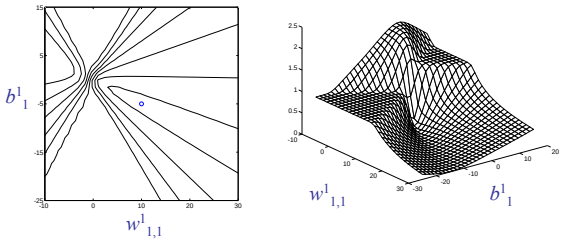
## 学習パラメータによる違い



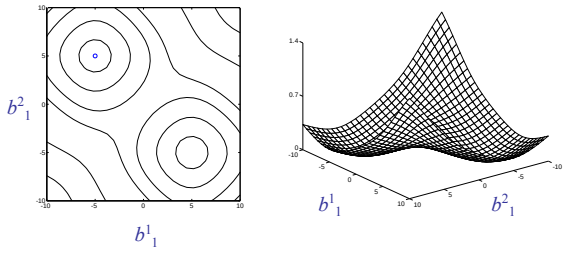
## 二乗誤差 vs. $w^1_{1,1}$ と $w^2_{1,1}$



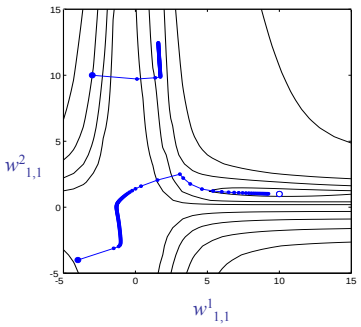

## 二乗誤差 vs. $w^1_{1,1}$ と $b^1_1$



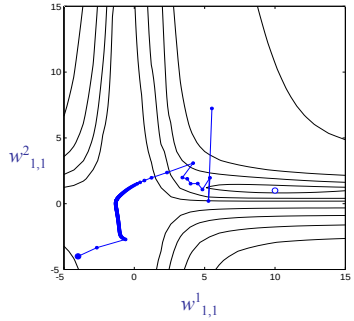

## 二乗誤差 vs. $b^1_1$ と $b^1_2$




## 収束過程の例



## 学習係数が大きすぎる場合

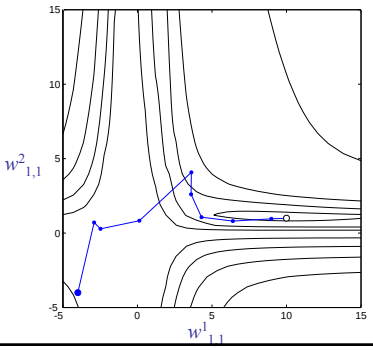



## BP法で満足か？

- とんでもない！
- "最適化法" でできることがわかってしまえば、最急降下法よりよい（よさそうな）ものは、いくらでもある。
- 様々な方法が試みられた
- それなりにうまくはいくのだが、目を見張るほどではない
  - 特に問題なのは計算時間
    - 高速な手法は、|W|*|W|（|W|は荷重の個数）の行列の逆行列の計算が必要とするから
    - 逆行列を、荷重の逐次更新とともに、逐次近似する方法が用いられる
  - それに見合うだけの、成功率と局所解回避率が得られない
  - Neural Networks は単純な形のようだが、結構性質が悪い。特に「特異点」があって、収束を遅くしたり、行列が特異になったりする
- 私が調べ・試みた中で最良のものは、Levenberg-Marquardt 法

Timothy Masters, Advanced Algorithms for Neural Networks: A C++ Sourcebook, John Wiley & Sons (1995).

## Levenberg-Marquardt 法

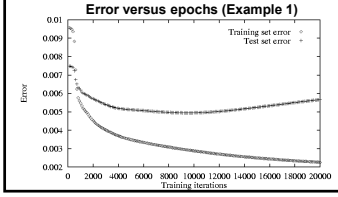


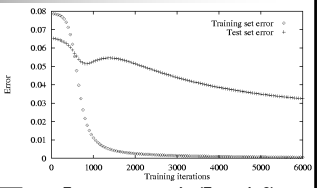

## Feedforward ANNs: 表現力とバイアス

- 表現力
  - 隠れ層1のfeedforward ANN
    - 任意の Boolean function が実現できる(“できる” ことは自明. AND-OR ネットワークを真似)
    - 任意の 有界連続関数 bounded continuous function (任意精度で近似) [Funahashi, 1989; Cybenko, 1989; Hornik *et al*, 1989]
  - シグモイド関数（でなくともよい）: 基底関数 basis functions; （ほぼ）局所的な和で関数近似
  - ANNs が近似容易な関数: Network Efficiently Representable Functions (NERFs) - 特徴づけはできていない [Russell and Norvig, 1995]
- ANNs の帰納バイアス
  - *n*-次元ユークリッド空間 (結合荷重の空間 weight space)
  - 連続関数 (荷重パラメータに関して連続)
  - 選択バイアス: 訓練事例の “滑らかな内挿”
  - よくは分かっていない

## ANNs の過学習

- 復習: 過学習の定義
  - $h'$ は $h$ と比較して, worse on $D_{train}$ better
- 過学習: ある型
  - 繰り返し過ぎ
  - 回避: 停止条件
    (cross-validation: holdout, *k*-fold)
  - 回避策: weight decay




Error versus epochs (Example 2)

## ANNs の過学習

- 過学習の考えられる他の原因
  - 予め設定する隠れ素子数の個数
  - 少なすぎると, 十分に学習できない (“underfitting”)
    - 成長不足
    - 連想: 連立方程式で、式（NNモデル）の個数（自由度）より変数（真の概念）の個数が多い
  - 多すぎると過学習
    - 枝刈りされていない
    - 連想: 2次多項式をより高次の多項式で近似する
- 解
  - 予防: 属性部分集合選択 attribute subset selection (pre-filter または wrapper)
  - 回避
    - cross-validation (CV)
    - Weight decay: エポックごとに荷重を一定値で（絶対値を）減少させる
  - 発見/回復: random restarts: 初期値をランダムにかえて, 荷重や素子の addition and deletion
- 過学習は存在しない（非常に小さい確率でのみ存在する）という議論がある

S. Amari, N. Murata, K.-R. Muller, M Finke and H. H. Yang, Asymptotic Statistical Theory of Overtraining and Cross-Validation, IEEE Transactions on Neural Networks, Vol. 8, No. 5, pp. 985-996, 1997.

## 他の誤差関数

- 大きな荷重にペナルティを

$$E(\vec{w}) \equiv \sum_{d \in D} \sum_{k \in outputs} \left(t_{k,d} - o_{k,d}\right)^2 + \gamma \sum_{i,j} w_{j,i}^{\ 2}$$

- 関数の傾きも学習対象

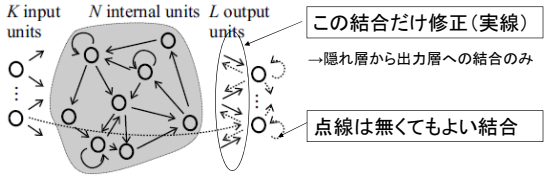
$$E(\vec{w}) \equiv \sum_{d \in D} \sum_{k \in outputs} \left[ \left(t_{k,d} - o_{k,d}\right)^2 + \mu \sum_{j \in inputs} \left( \frac{\partial t_{k,d}}{\partial x_d^j} - \frac{\partial o_{k,d}}{\partial x_d^j} \right)^2 \right]$$

## そのほかのニューロンモデル

- 状態をもったニューロン
  - Neuroids [Valiant, 1994]
  - それぞれの基本素子が状態をもつ
  - それぞれの更新規則は異なってもよい (または 状態に基づく異なった計算)
  - 適応的なネットワークモデル
    - ランダムグラフの構造
    - 基本素子は学習過程の一部として意味も受取る
- パルス・コーディング
  - スパイク・ニューロン spiking neurons [Maass and Schmitt, 1997]
  - 活動度が出力の表現ではない
  - 発火の列間の相のずれが意味をもつ
    - 古い時間コーディング temporal coding では rate coding が用いられ、それは活動度で表現可能
- 新しい更新規則
  - 非加算的更新 [Stein and Meredith, 1993; Seguin, 1998]
  - スパイク・ニューロン・モデル spiking neuron model

## ESN: 少し変わったNN

ESN: Echo State Network
Jaeger H. and Haas, H. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science, 304:78-80, 2004.



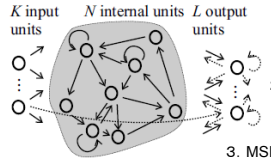

**結合加重**
**隠れ層と出力層**: 可変
**その他**:ランダムに決定し、以降固定

**内部ユニットの出力**

$$x(n+1) = f(W^{in}u(n+1) + Wx(n) + W^{back}y(n))$$

## ESNの学習例

**入力**: $u(n) = \sin(n/5)$ →10πの間隔で入力・教師データを取得
**出力**: $y(n)_{teach} = 1/2\sin^7(n/5)$




1. データ入力
2. 教師データとのMSEを計算
3. MSEを最小化する結合加重を計算

**結合加重**
**隠れ層内部**: 0 +0.4 -0.4に0.95 0.025 0.025の確率で決定
**入力層と隠れ層**: 1 -1に等確率で決定
**フィードバック結合**: 1 -1に等確率で決定

## ESNの学習例:
## The Mackey Glass system

- Chaotic attractorの学習・・・dynamical system の学習のためのテスト.ポピュラーだが難しい

$$\dot{y}(t) = \alpha y(t-\tau)/(1 + y(t-\tau)^{\beta}) - \gamma y(t)$$

・パラメータは以下のように設定

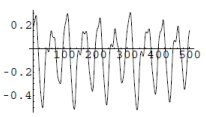
$$\alpha = 0.2, \beta = 10, \gamma = 0.1$$

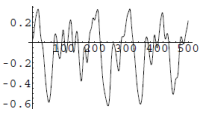
・Mildな動き $\tau = 17$　・Wildな動き $\tau = 30$

→**離散化**

$$y(n+1) = y(n) + \delta\left(\frac{0.2y(n-\tau/\delta)}{1 + y(n-\tau/\delta)^{10}} - 0.1y(n)\right)$$

## 学習するデータ例



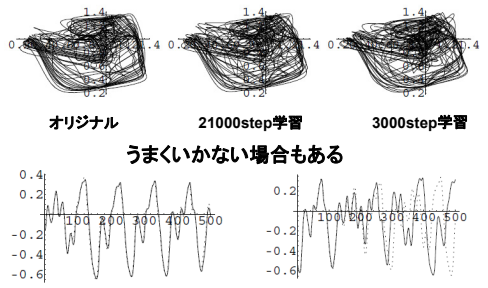




・Mildな動き $\tau = 17$　・Wildな動き $\tau = 30$

**内部ユニットの出力**:ノイズ入り

$$x(n+1) =$$
$$(1 - \delta C a)x(n) + \delta C(f(W^{in}u(n+1) + Wx(n) + W^{back}y(n) + v(n)))$$

## 学習結果(Wildな動き) $\tau = 30$

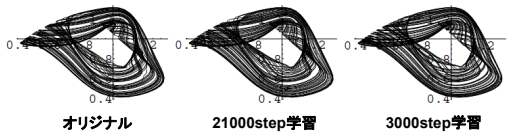


オリジナル　21000step学習　3000step学習

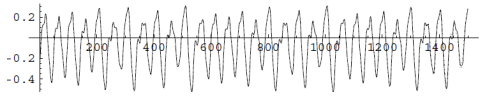
うまくいかない場合もある



## 学習結果(Mildな動き) $\tau = 17$



オリジナル　21000step学習　3000step学習



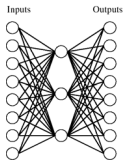
→かなりうまくいっている

## 補足：中間層に発現する表現

- 中間層には、プログラマが意図しなかった内部表現が発生する（ことがある）
- よくよく見ると「意味深い」表現であったりする
- 実は、オンライン逐次学習法を用いるとBayes学習的なことがおこり、「情報の圧縮」すなわち、「意味抽出」が行われうることが示せる

## 中間層での表現

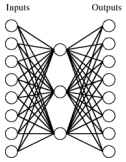
- これは学習できるか？




| Input | | Output |
|---|---|---|
| 10000000 | → | 10000000 |
| 01000000 | → | 01000000 |
| 00100000 | → | 00100000 |
| 00010000 | → | 00010000 |
| 00001000 | → | 00001000 |
| 00000100 | → | 00000100 |
| 00000010 | → | 00000010 |
| 00000001 | → | 00000001 |

## 中間層での表現（2）

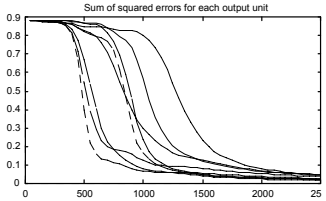
- 学習結果




| Input | | | | | | Output |
|---|---|---|---|---|---|---|
| 10000000 | → | .89 | .04 | .08 | → | 10000000 |
| 01000000 | → | .01 | .11 | .88 | → | 01000000 |
| 00100000 | → | .01 | .97 | .27 | → | 00100000 |
| 00010000 | → | .99 | .97 | .71 | → | 00010000 |
| 00001000 | → | .03 | .05 | .02 | → | 00001000 |
| 00000100 | → | .22 | .99 | .99 | → | 00000100 |
| 00000010 | → | .80 | .01 | .98 | → | 00000010 |
| 00000001 | → | .60 | .94 | .01 | → | 00000001 |

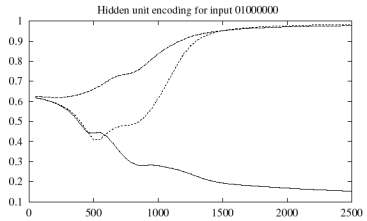
## 隠れ層での表現（3）

- 学習の進行の様子



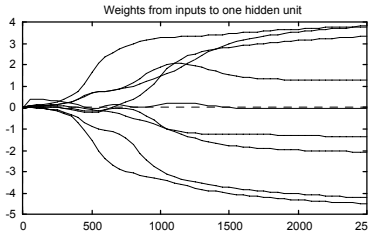
## 隠れ層での表現(4)

- 学習の進行の様子(2)

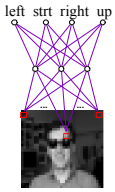



## 隠れ層での表現(5)

- 学習の進行の様子(3)




## 古い例: 顔画像の学習

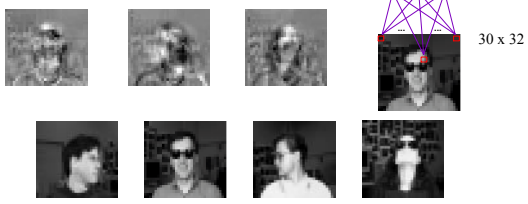



30 x 32

- 顔画像の例

## 顔画像の学習

- 学習後の荷重






30 x 32