情報意味論
（多層）ニューラルネットワーク

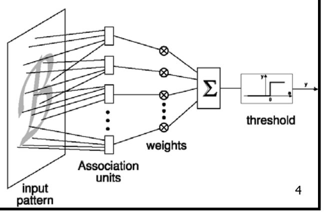
櫻井彰人
慶應義塾大学理工学部

目次

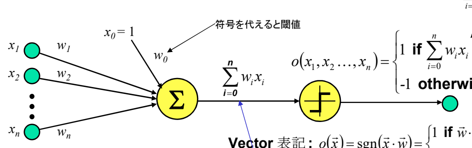
目次

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



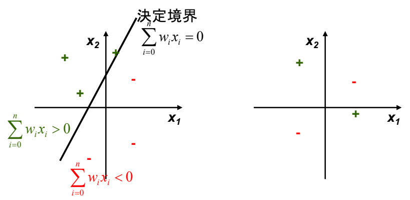
$$\sum_{i=1}^{n} w_i x_i > -w_0$$

$$o(x_1, x_2 \ldots, x_n) = \begin{cases} 1 & \textbf{if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \textbf{otherwise} \end{cases}$$

Vector 表記 : $o(\vec{x}) = \mathrm{sgn}(\vec{x} \cdot \vec{w}) = \begin{cases} 1 & \textbf{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \textbf{otherwise} \end{cases}$

- 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 $\textbf{net} = \sum_{i=0}^{n} w_i x_i$
  - 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



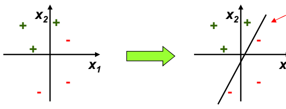
- パーセプトロン: 重要な関数がいくつも簡単に表現できる
  - 論理関数 (McCulloch and Pitts, 1943) の多くのもの
  - e.g., 簡単な荷重で $AND(x_1, x_2)$, $OR(x_1, x_2)$, $NOT(x)$
- 表現できない関数もある
  - 線型分離可能でないもの – 同語反復ですが

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 $<\boldsymbol{x}_i, t_i>$ 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

学習する



7

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* ($D \equiv \{<x, t(x) \equiv c(x)>\}$) // 実は少し嘘が入っている
  - 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 $x \in D$

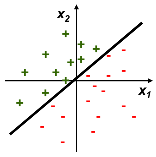
    現在の出力 o=*o*(*x*) を計算

    FOR *i* = 1 to *n*

    $\boldsymbol{w_i \leftarrow w_i + r(t - o)x_i}$ // r は正数ならなんでもよい
- 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
    - 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
  - 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。

8

線型分離可能



Linearly Separable (LS) Data Set

- 定義
  - $f(x) = 1$ if $w_1x_1 + w_2x_2 + \ldots + w_nx_n \ge \theta$, 0 otherwise
  - θ: 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?

9

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: 探索空間が有界な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

10

目次

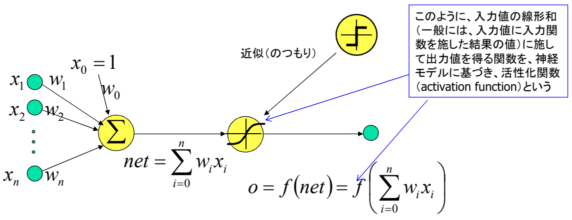
11

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

12

基本的素子: シグモイド素子



$$x_0 = 1$$

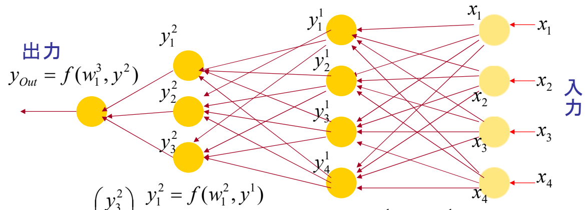
$$net = \sum_{i=0}^{n} w_i x_i \qquad o = f(net) = f\left(\sum_{i=0}^{n} w_i x_i\right)$$

極めて頻繁に使われる活性化関数: $\tanh(x) = \dfrac{1-e^{-x}}{1+e^{-x}} = 2\sigma(x) - 1$

$$\sigma(x) \equiv \frac{1}{1+e^{-x}}$$

13

基本的な計算



$$y_{Out} = f(w_1^3, y^2)$$

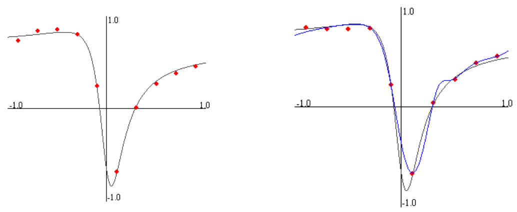
$$y^2 = \begin{pmatrix} y_3^2 \\ y_3^2 \\ y_3^2 \end{pmatrix}, \quad \begin{aligned} y_1^2 &= f(w_1^2, y^1) \\ y_2^2 &= f(w_2^2, y^1) \\ y_3^2 &= f(w_3^2, y^1) \end{aligned}$$

$$y^1 = \begin{pmatrix} y_1^1 \\ y_2^1 \\ y_3^1 \\ y_4^1 \end{pmatrix}, \quad \begin{aligned} y_1^1 &= f(w_1^1, x_1) \\ y_2^1 &= f(w_2^1, x_2) \\ y_3^1 &= f(w_3^1, x_3) \\ y_4^1 &= f(w_4^1, x_4) \end{aligned}$$

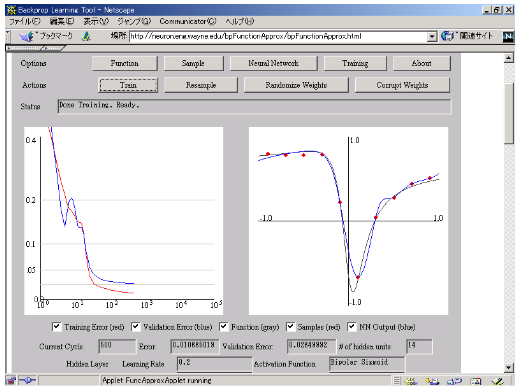
14

基本的な原理1: 関数近似・回帰

- 例:1入力・1出力



15

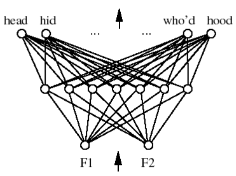


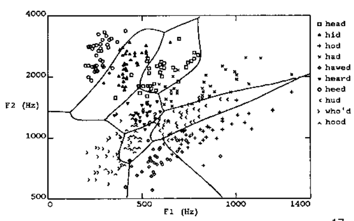
http://neuron.eng.wayne.edu/bpFunctionApprox/bpFunctionApprox.html

16

基本的な原理2: 分類

- 例:音声認識
  - 2入力・多出力
  - 各入力値ごと、出力値の内、最大値を与えるクラスを分類結果とする



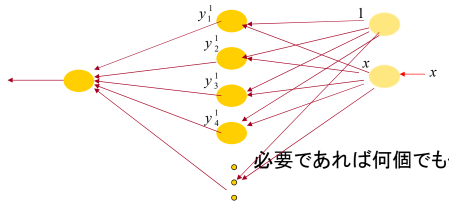


http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/theo-3/www/ml.html

17

表現能力: 関数近似の場合

- 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

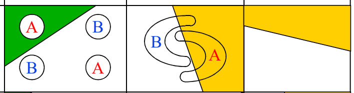
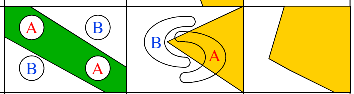
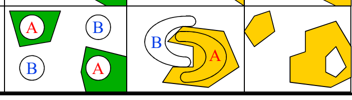


K.Funahashi: "On the Approximate Realization of Continuous Mappings by Neural Networks", Neural Networks,2,pp.183-192(1989).

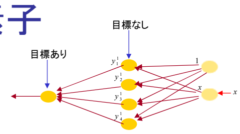
G. Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of Control Signals Systems, 2:303-314, 1989

表現能力: 分類の場合

- 例：２入力・１出力（図は閾値素子の場合）
  - 出力値を閾値以上か未満かで2クラスに分ける

構造	境界面の形	例１ 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）			

学習方法: 閾値素子



- 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、、、、

20

学習＝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

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

  - 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
- 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

21

補足: 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、（再）発見が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。

コンピュータの能力は、
当時は現在の 10^{-4}～10^{-6} 倍ぐらい

  - 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。

22

目次

- パーセプトロン
  - 実は、１素子
- 多層パーセプトロン
- 学習アルゴリズム
  - ネットワークの種類
  - 誤差最小化
  - 誤差逆伝播
  - 収束の例図
  - LM法
- 表現力と過学習
- その他のニューラルネットワーク
  - リカレント、DLN、補足：中間層表現
  - ESN

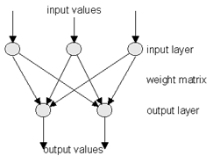
23

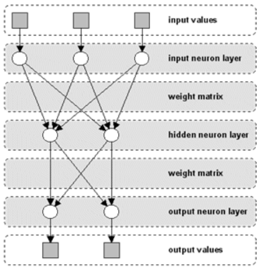
適用するネットワークの形状

- アイデア「学習＝誤差最小化」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）

24

再掲: 階層型（多層パーセプトロン）

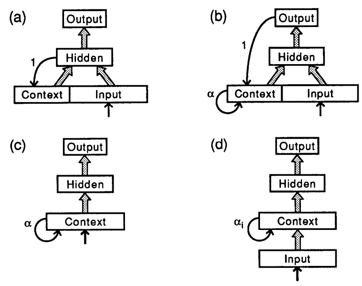




25

再掲: 相互結合（recurrent）

- 階層型＋フィードバック（離散時間遅れ）



26

目次

- パーセプトロン
  - 実は、1素子
- 多層パーセプトロン
- 学習アルゴリズム
  - ネットワークの種類
  - 誤差最小化
  - 誤差逆伝播
  - 収束の例図
  - LM法
- 表現力と過学習
- その他のニューラルネットワーク
  - リカレント、DLN、補足：中間層表現
  - ESN

27

戻って：誤差最小化の方法

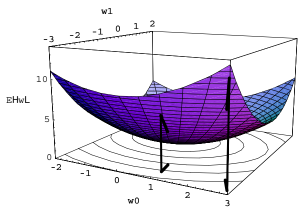
- 微分して0とおいた方程式を解けばよい！
本当か？
  - もっとも、まずは、微分できないことには話しにならん
  - つまり、パーセプトロンではだめ。
  - そこで、シグモイド関数にした。

  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

28

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！

29

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）

30

学部生に戻って

$$E = (t-y)^2$$

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

$$= -2(t-y)x$$

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

$$\frac{\partial (t-y)}{\partial W} = -\frac{\partial y}{\partial W}$$ (t は目標値であり、W に依存しない)

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

$$= -x$$ （x はベクトルです）

活性化関数が恒等関数の場合、i.e. $y = \sum_{i=1}^{n} w_i x_i$

もう少し

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

であるから、2α を α と書き換えれば

$$W \leftarrow W + \Delta W$$ かつ $$\Delta W = \alpha(t-y)x$$

この形の学習則を1個のニューロンに適用した規則は、デルタ則とも呼ばれる（Widrow-Hoff 則とも呼ばれる）

$y = \sum_{i=1}^{n} w_i x_i$ を仮定している

32

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

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

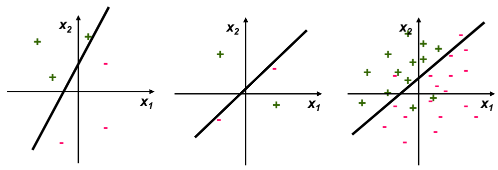
- つまり、perceptron 学習則の妥当性を示しているように見える。
- しかし閾値関数は微分不能なため、perceptron 学習則の妥当性は、二乗誤差最小化では説明できない。

33

デルタ規則とperceptron学習則

シグモイド素子と閾値素子

活性化関数が、シグモイド関数の場合、閾値関数の場合



例 A　例 B　例 C

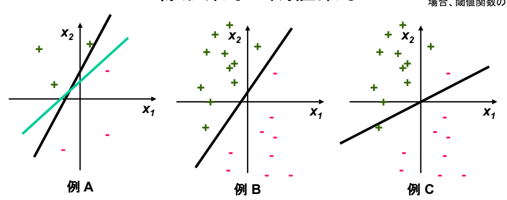
- 線型分離可能: 完全な分類ができる
  - 例 A: パーセプトロン学習アルゴリズムが収束
- 線型分離不能: 近似できるのみ
  - 例 B: 線型分離不能; デルタ規則は収束, しかし3個正解よりはよくならない
  - 例 C: 線型分離不能; デルタ規則でよい結果

34

デルタ規則とperceptron学習則

線形素子と閾値素子

活性化関数が、恒等関数の場合、閾値関数の場合



例 A　例 B　例 C

- 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
- 線型分離不能: 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には分類しない。バランスはよい？

35

活性化関数が恒等関数以外のとき

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

ΔW_s　サンプル s に対する荷重の更新量

α　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

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

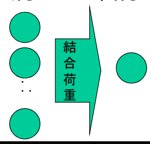
f　活性化関数。微分可能

$y = f(y_{in})$　実際の出力値

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

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

n 入力　1 出力



もう一度

$$E = (t - y)^2$$

$$y = f(y_{in}),\ y_{in} = \sum_{i=1}^{n} w_i x_i$$

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

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

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

(x はベクトルです)

前と全く同様に

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

であるから、2α を α と書き換えれば

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

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。
  - なぜだろう？
- 実は、学習サンプル数が多いとき、mini-batch がよいと言われている

出力素子が複数個のとき

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

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

α　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

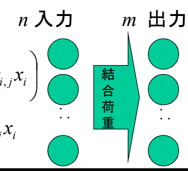
$y_{in,j} = \sum_{i=1}^{n} w_{i,j} x_i$

f　活性化関数: 微分可能

$y_j = f(y_{in,j})$　実際の出力値

$$y_j = f\left(\sum_{i=1}^{n} w_{i,j} x_i\right)$$

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



前と全く同様に

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

であるから、2α を α と書き換えれば

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

目次

43

多層のネットワークでは?

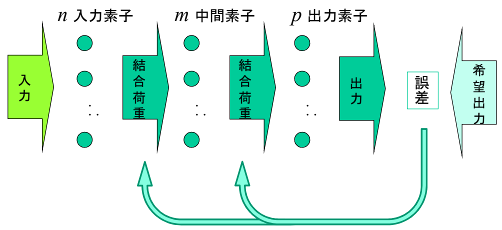
$\Delta W_j = \alpha(t_j - y_j) f'(y_{in,j}) x$

- 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

44

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

45

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、

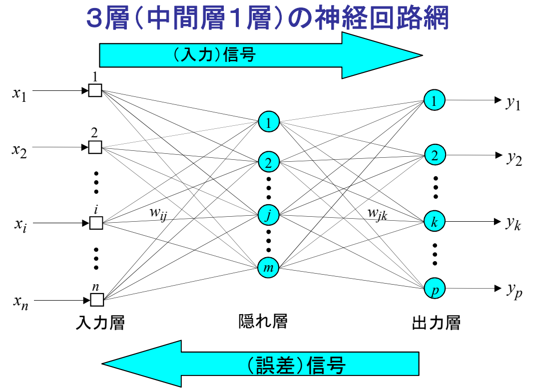
$$E = \sum_s (t_s - y_s)^2$$

とおき、例えば、最急降下法を適用すればよい

E の選び方は、勿論、他にもいろいろある

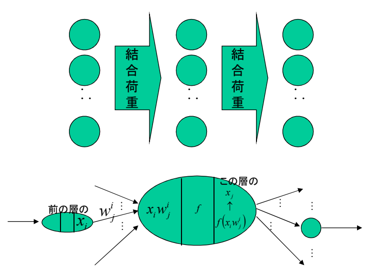
46

3層(中間層1層)の神経回路網



誤差逆伝播法(error back-propagation or BP)

47



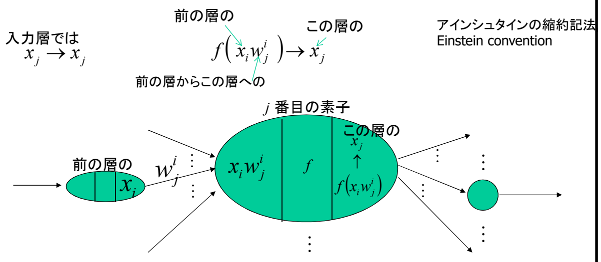
簡単には　$E = (t - x)^2$　　ただし: t 目標出力, x 出力層の出力値

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:

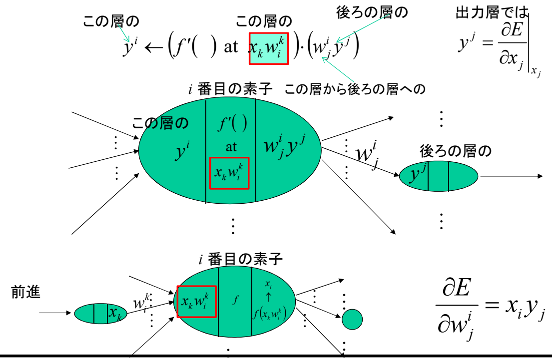
$E = \sum_{k=1}^{p} [t_k - x_k]^2$　　クロスエントロピーも良く使われる

48

出力値の計算（前方伝播）



勾配の計算（後方伝播）



まとめ1: 教師付き学習

- 結合荷重の求め方
  - 学習データを $\{(x_t, y_t) \mid 1 \le t \le N\}$ 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、

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

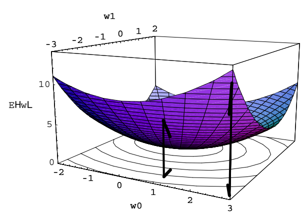
  - このEを最小化するWを求めればよい

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！
  それはそうだが、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）

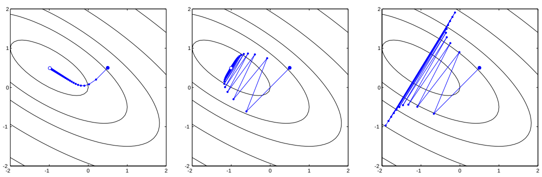
目次

- パーセプトロン
  - 実は、1素子
- 多層パーセプトロン
- 学習アルゴリズム
  - ネットワークの種類
  - 誤差最小化
  - 誤差逆伝播
  - 収束の例図
  - LM法
- 表現力と過学習
- その他のニューラルネットワーク
  - リカレント、DLN、補足：中間層表現
  - ESN

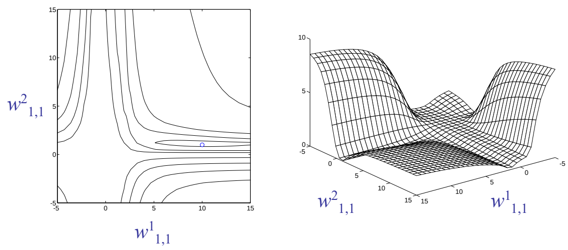
誤差逆伝播法の収束

- 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]

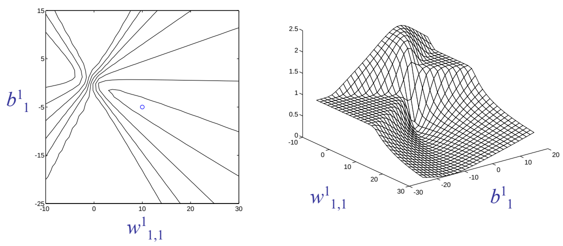
学習パラメータによる違い



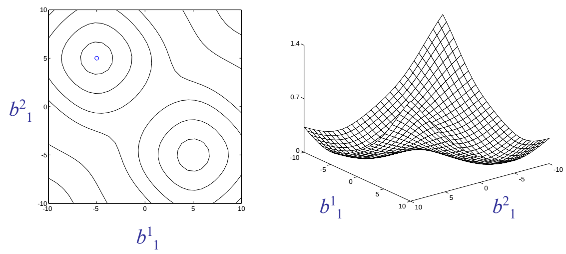
二乗誤差 vs. $w^1_{1,1}$ と $w^2_{1,1}$



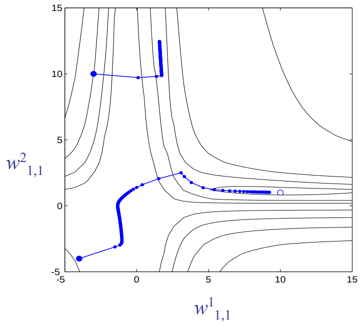
二乗誤差 vs. $w^1_{1,1}$ と b^1_1



二乗誤差 vs. b^1_1 と b^1_2

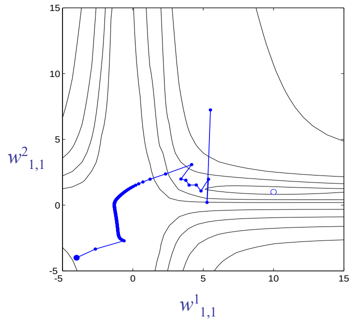


収束過程の例



61

学習係数が大きすぎる場合



62

目次

- パーセプトロン
  - 実は、1素子
- 多層パーセプトロン
- 学習アルゴリズム
  - ネットワークの種類
  - 誤差最小化
  - 誤差逆伝播
  - 収束の例図
  - LM法
- 表現力と過学習
- その他のニューラルネットワーク
  - リカレント、DLN、補足：中間層表現
  - ESN

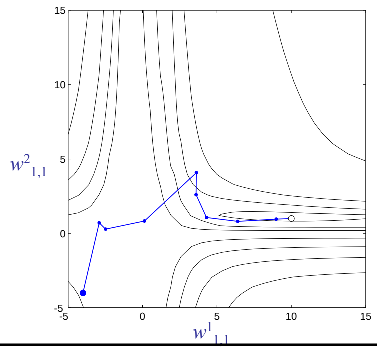
63

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



65

LMの説明のために: BP

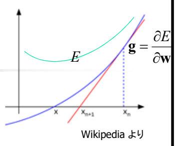
- 最急降下法

$$\Delta \mathbf{w}^{(k)} = -\alpha\, \mathbf{g}^{(k)} \qquad \mathbf{g} = \frac{\partial E}{\partial \mathbf{w}} \qquad E = \frac{1}{2}\sum_{p=1}^{P}\sum_{m=1}^{M} e_{pm}^2$$

- 利点:
  - 簡単
  - （比較的）安定
- 欠点:
  - 学習能力が低い（失敗することが多い）
  - 収束が遅い

66

LMの説明のために: ニュートン法



Wikipedia より

- ニュートン法: 非線形方程式の解の近似解法. 極小点(停留点)の座標を求めるのに使用

$$\Delta \mathbf{w}^{(k)} = -\left(\mathbf{H}^{(k)}\right)^{-1} \mathbf{g}^{(k)}$$

$$\mathbf{g} = \frac{\partial E}{\partial \mathbf{w}} \qquad g_i = \frac{\partial E}{\partial w_i}$$

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{m=1}^{M} e_{pm}^2$$

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_N} \\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial w_2^2} & \cdots & \frac{\partial^2 E}{\partial w_2 \partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 E}{\partial w_N \partial w_1} & \frac{\partial^2 E}{\partial w_N \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_N^2} \end{bmatrix}$$

- 利点:
  - 収束が早い
- 欠点:
  - 安定性に乏しい
  - (誤差関数の)2階微分を計算する必要がある

67

LMの説明のために: ガウス・ニュートン法

- ガウス・ニュートン法: ニュートン法における2階微分の計算を省略する。ヤコビアンを用いる。なお、誤差関数が2乗和であることを利用している。

$$\Delta \mathbf{w}^{(k)} = -\left(\left(\mathbf{J}^{(k)}\right)^T \mathbf{J}^{(k)}\right)^{-1} \left(\mathbf{J}^{(k)}\right)^T \mathbf{e}^{(k)} \qquad E = \frac{1}{2}\sum_{p=1}^{P}\sum_{m=1}^{M} e_{pm}^2$$

$$\mathbf{g} = \mathbf{J}^T \mathbf{e}$$

$$\mathbf{H} = \mathbf{J}^T \mathbf{J} + \text{2nd derivative}$$

$$\approx \mathbf{J}^T \mathbf{J}$$

cf. ニュートン法

$$\Delta \mathbf{w}^{(k)} = -\alpha\left(\mathbf{H}^{(k)}\right)^{-1} \mathbf{g}^{(k)}$$

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w_1} & \frac{\partial e_{1,1}}{\partial w_2} & \cdots & \frac{\partial e_{1,1}}{\partial w_N} \\ \frac{\partial e_{1,2}}{\partial w_1} & \frac{\partial e_{1,2}}{\partial w_2} & \cdots & \frac{\partial e_{1,2}}{\partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{1,M}}{\partial w_1} & \frac{\partial e_{1,M}}{\partial w_2} & \cdots & \frac{\partial e_{1,M}}{\partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{P,1}}{\partial w_1} & \frac{\partial e_{P,1}}{\partial w_2} & \cdots & \frac{\partial e_{P,1}}{\partial w_N} \\ \frac{\partial e_{P,2}}{\partial w_1} & \frac{\partial e_{P,2}}{\partial w_2} & \cdots & \frac{\partial e_{P,2}}{\partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{P,M}}{\partial w_1} & \frac{\partial e_{P,M}}{\partial w_2} & \cdots & \frac{\partial e_{P,M}}{\partial w_N} \end{bmatrix} \qquad \boldsymbol{e} = \begin{bmatrix} e_{1,1} \\ e_{1,2} \\ \cdots \\ e_{1,M} \\ \cdots \\ e_{P,1} \\ e_{P,2} \\ \cdots \\ e_{P,M} \end{bmatrix}$$

- 利点:
  - 収束が高速
- 欠点:
  - 不安定

Levenberg Marquardt 法

- LM法: 誤差逆伝播法とガウスニュートン法の混合

$$\Delta \mathbf{w}^{(k)} = -\left(\left(\mathbf{J}^{(k)}\right)^T \mathbf{J}^{(k)} + \mu^{(k)}\mathbf{I}\right)^{-1} \left(\mathbf{J}^{(k)}\right)^T \mathbf{e}_k$$

  - 誤差が増大するとき, μ を増大させる, LM はBP に近くなる
  - 誤差が減少するとき, μ を減少させる, LM はガウス・ニュートンに近くなる

- 利点
  - 収束が早い
  - 安定している
- 欠点:
  - 計算量が多い

K. Levenberg, "A method for the solution of certain problems in least squares," *Quarterly of Applied Machematics*, 5, pp. 164-168, 1944.
D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *SIAM J. Appl. Math.*, vol. 11, no. 2, pp. 431-441, Jun. 1963.

SGDにおける加速方法

- BP のオンライン版を考える
  - ミニバッチと呼ばれるものも含まれる
  - SGD (Stochastic Gradient Descent) のNN版である
  - 一個または複数個(全部に比べればごく少数)のサンプルのみで、荷重の更新量を決める
  - サンプルはランダムに選ぶ
    - 通常は、ランダムに並べて順番に取る。一巡したらランダムに並べなおす。
  - NNの時には、なぜか、非常に遅くなることがある。
    - なぜか？

$$\vec{w} \leftarrow \vec{w} - \alpha g_{\vec{w}} \qquad g_{\vec{w}} = \frac{\partial E}{\partial \vec{w}}$$

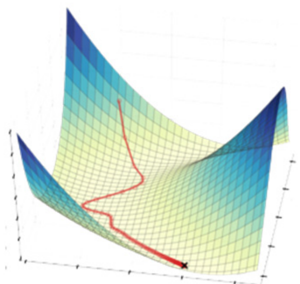
70

NNにおけるSGD

- 誤差関数の減少過程を追っていくと、現象が停滞する時間がある。それが長い。
  - プラトーと呼ばれる
  - その原因は？
- 荷重ベクトルが作る空間での、誤差関数の縮退(特異点の存在)が考えられる
  - 同じ「形」の結合ばかり
  - SGDの計算過程で、その付近を通過する
- そこで、高速化手法が試みられた

71

プラトーを進むということ



http://librimind.com/2016/03/optimizations-of-gradient-descent/

72

脱線: プラトーとは

momentum の考え方

- 基本的なアイデア
  - プラトーでは、歩幅は小さいが、似た方向へ進む
  - 加速するには、過去の歩みを加算すればよい
    - もっとも、大昔の歩みは忘れた方がよさそう。
  - 指数減衰させる

$$\vec{v} \leftarrow \mu\vec{v} - \alpha g_{\vec{w}}$$
$$\vec{w} \leftarrow \vec{w} + \vec{v}$$

$\Leftarrow$

$$\vec{w} \leftarrow \vec{w} - \alpha g_{\vec{w}}$$

様々な加速方法 の内の少々

- AdaGrad

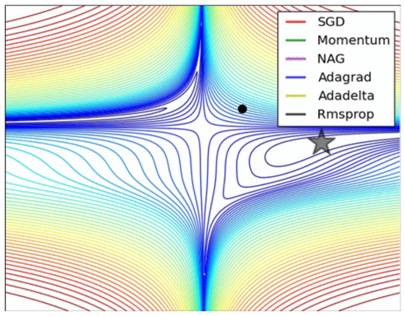
$$\vec{r} \leftarrow \vec{r} + g_{\vec{w}}^2$$
$$\vec{w} \leftarrow \vec{w} - \frac{\alpha}{\sqrt{\vec{r}} + \varepsilon} g_{\vec{w}}$$

- AdaDelta

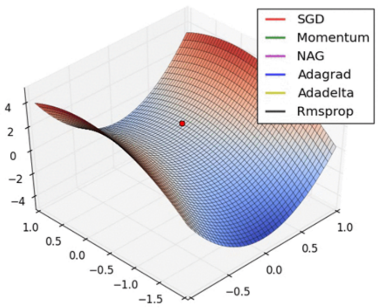
$$\vec{r} \leftarrow \beta\vec{r} + (1-\beta) g_{\vec{w}}^2$$
$$\vec{v} \leftarrow \frac{\sqrt{\vec{s}} + \varepsilon}{\sqrt{\vec{r}} + \varepsilon} g_{\vec{w}}$$
$$\vec{s} \leftarrow \beta\vec{s} + (1-\beta)\vec{v}^2$$
$$\vec{w} \leftarrow \vec{w} - \vec{v}$$

- Adam

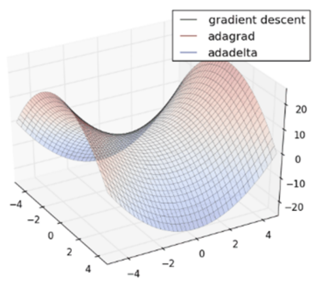
$$\vec{v} \leftarrow \beta\vec{v} + (1-\beta) g_{\vec{w}}$$
$$\vec{r} \leftarrow \gamma\vec{r} + (1-\gamma) g_{\vec{w}}^2$$
$$\vec{w} \leftarrow \vec{w} - \frac{\alpha}{\sqrt{\frac{\vec{r}}{1-\gamma^t}} + \varepsilon} \frac{\vec{v}}{1-\beta^t} g_{\vec{w}}$$



http://cs231n.github.io/neural-networks-3/



http://cs231n.github.io/neural-networks-3/



http://cpmarkchang.logdown.com/posts/467674-optimization-method-adadelta

目次

- パーセプトロン
  - 実は、1素子
- 多層パーセプトロン
- 学習アルゴリズム
  - ネットワークの種類
  - 誤差最小化
  - 誤差逆伝播
  - 収束の例図
  - LM法
- 表現力と過学習
- その他のニューラルネットワーク
  - リカレント、DLN、補足：中間層表現
  - ESN

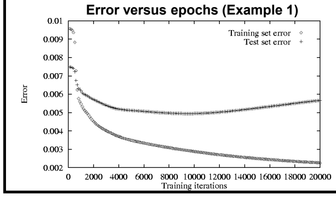
Feedforward ANNs: 表現力とバイアス

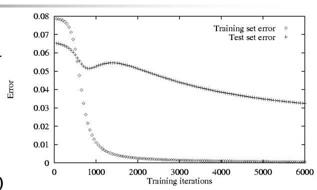
- 表現力
  - 隠れ層1のfeedforward ANN
    - 任意の Boolean function が実現できる(“できる” ことは自明. AND-OR ネットワークを真似)
    - 任意の 有界連続関数 bounded continuous function (任意精度で近似) [Funahashi, 1989; Cybenko, 1989; Hornik *et al*, 1989]
  - シグモイド関数（でなくともよい）: 基底関数 basis functions; （ほぼ）局所的な和で関数近似
  - ANNs が近似容易な関数: Network Efficiently 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 1)





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.

正則化項の例

- 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

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

- 最近は drop-out が有効だと考えられている

目次

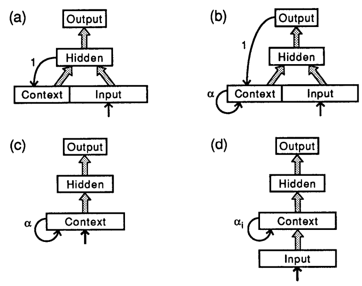
- パーセプトロン
  - 実は、1素子
- 多層パーセプトロン
- 学習アルゴリズム
  - ネットワークの種類
  - 誤差最小化
  - 誤差逆伝播
  - 収束の例図
  - LM法
- 表現力と過学習
- その他のネットワーク
  - リカレント、DLN、補足：中間層表現
  - ESN

その他のニューロンモデル

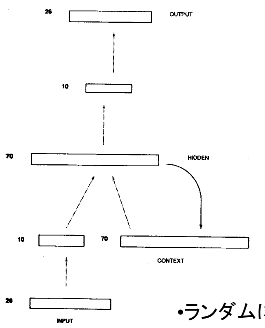
- リカレント（再帰型）ニューラルネットワーク
  - 時系列の学習に利用
- Deep learning neural network
  - 中間層数 ≥ 2
    - 非常に誤解されている（2〜3ぐらいは意見が分かれるが）

相互結合（recurrent）

- 階層型＋フィードバック（離散時間遅れ）



リカレントニューラルネットワーク
文法の学習（Elman）

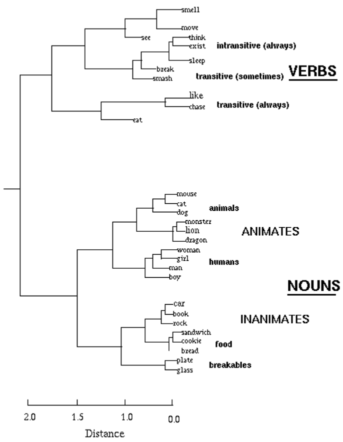


S → NP VP “.”
NP → PropN | N | N RC
VP → V (NP)
RC → *who* NP VP | *who* VP (NP)
N → *boy* | *girl* | *cat* | *dog* | *boys* | *girls* | *cats* | *dogs*
PropN → *John* | *Mary*
V → *chase* | *feed* | *see* | *hear* | *walk* | *live* | *chases* | *feeds* | *sees* | *hears* | *walks* | *lives*
Additional restrictions:
- number agreement between N & V within clause, and (where appropriate) between head N & subordinate V
- verb arguments:
  - *chase*, *feed* → require a direct object
  - *see*, *hear* → optionally allow a direct object
  - *walk*, *live* → preclude a direct object

•ランダムに10,000文生成し学習させる

J.L.Elman「Distributed Representations, Simple Recurrent Networks, and Grammatical Structure」

単語間の近接性

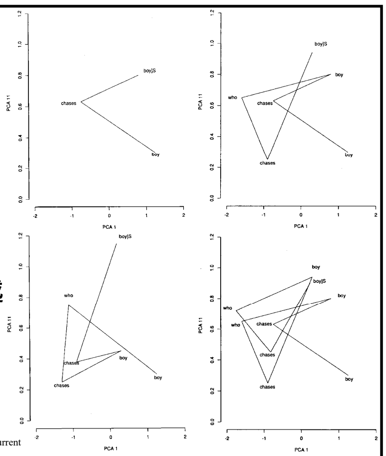


中間層（文脈層）の活動

J.L.Elman「Language as a dynamical system」

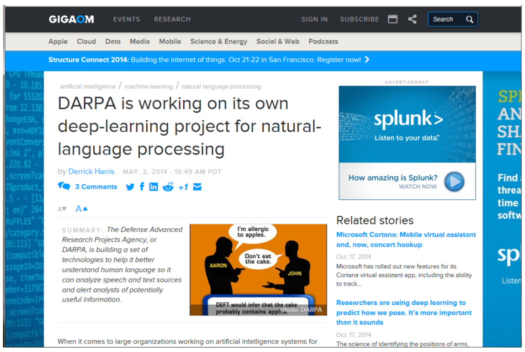
埋め込み文表現

- “who chases boy” の表現がどのような文脈でも似ている
- 埋め込み文を越えてagreementがとれている



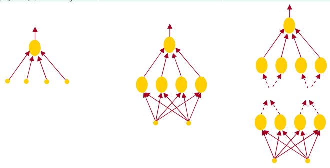
J.L.Elman「Distributed Representations, Simple Recurrent Networks, and Grammatical Structure」

Deep Learning



DLN: 歴史における位置づけ

	第一次	第二次	第三次？
年代	1950後半～1960年代	1986～1990年代	2012～
キーワード	perceptron	back-propagation	deep learning
何ができるようになったか	閾値素子1個の学習	シグモイド素子の隠れ層1層NNの学習	シグモイド素子の隠れ層多層NNの学習
\$1,000, 1秒当りの計算回数	1～10	10^5～10^6	10^9～
	IBM709～7090 (真空管～TR)	intel 486	core i7

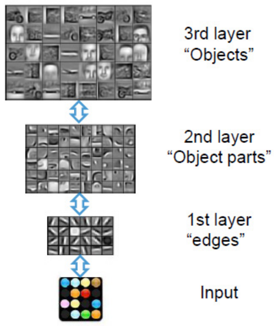


91

DLNの大枠

- 特徴量を学習する
  - Hand-craftはしない
- 抽象度が低い特徴から抽象度の高い特徴までを階層的に学習する
  - 抽象度の低い特徴は、類似タスクで利用可能
- 主な手法
  - Deep belief networks (Hinton)
  - Deep autoencoder (Bengio)
  - Deep neural networks

  etc.



もう少し細かい話は、後程

93

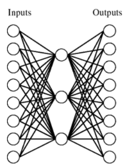
補足：中間層に発現する表現

- 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

94

中間層での表現

- これは学習できるか？

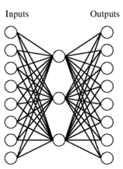


Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

95

中間層での表現(2)

- 学習結果

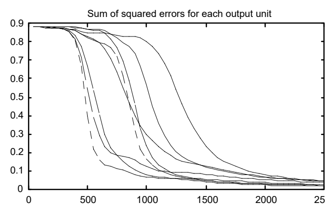


Input						Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

96

隠れ層での表現(3)

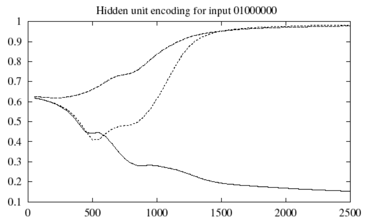
- 学習の進行の様子



97

隠れ層での表現（4）

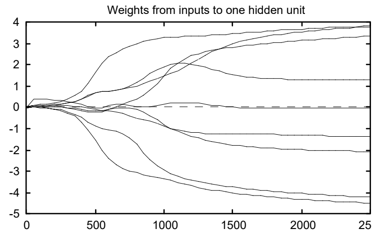
- 学習の進行の様子（2）



98

隠れ層での表現（5）

- 学習の進行の様子（3）



99

そのほかのニューロンモデル2

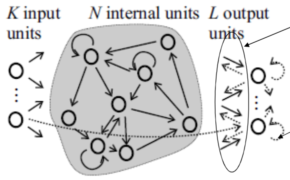
- 状態をもったニューロン
  - Neuroids [Valiant, 1994]
  - それぞれの基本素子が状態をもつ
  - それぞれの更新規則は異なってもよい（または 状態に基づく異なった計算）
  - 適応的なネットワークモデル
    - ランダムグラフの構造
    - 基本素子は学習過程の一部として意味も受取る
- パルス・コーディング
  - スパイク・ニューロン spiking neurons [Maass and Schmitt, 1997]
  - 活動度が出力の表現ではない
  - 発火の列間の相のずれが意味をもつ
    - 古い時間コーディング temporal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

102

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.



この結合だけ修正（実線）

→隠れ層から出力層への結合のみ

点線は無くてもよい結合

内部ユニットの出力

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

結合加重

隠れ層と出力層：可変

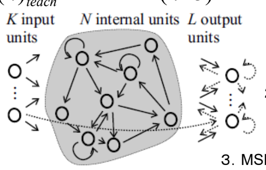
その他：ランダムに決定し、以降固定

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

104

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

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

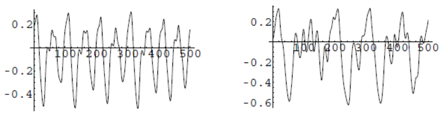
・Mildな動き $\tau=17$　・Wildな動き $\tau=30$

→**離散化**

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

105

学習するデータ例

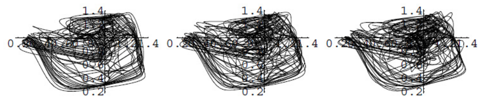


・Mildな動き $\tau=17$　・Wildな動き $\tau=30$

内部ユニットの出力:ノイズ入り

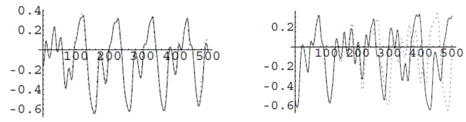
$$x(n+1)=$$
$$(1-\delta Ca)x(n)+\delta C(f(W^{in}u(n+1)+Wx(n)+W^{back}y(n)+\nu(n)))$$

学習結果(Wildな動き) $\tau=30$

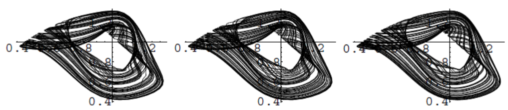


オリジナル　**21000step学習**　**3000step学習**

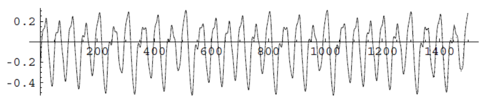
うまくいかない場合もある



学習結果(Mildな動き) $\tau=17$

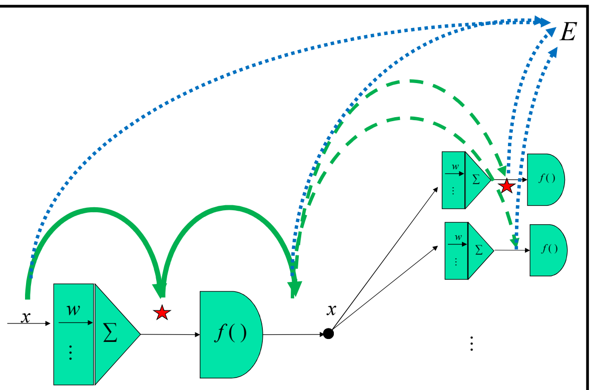


オリジナル　**21000step学習**　**3000step学習**

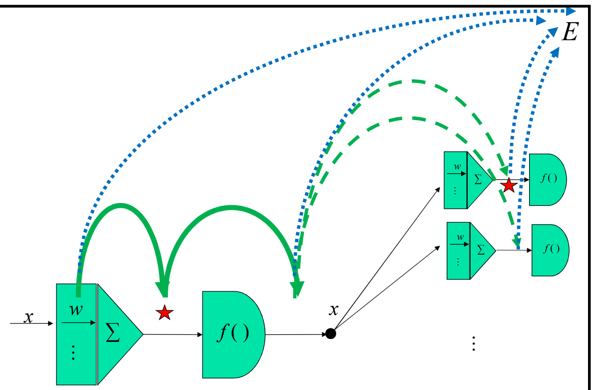


→かなりうまくいっている

付録



微分を「分数」と考える
合成関数の微分公式は、分数を分数の積で書いたもの



微分を「分数」と考える
合成関数の微分公式は、分数を分数の積で書いたもの