Pattern Recognition and Machine Learning

第5章　ニューラルネットワーク

博士課程１年
原　祐輔

BehaviorInNetworks
都市生活学・ネットワーク行動学研究室

５章の内容

- ５１ フィードフォワードネットワーク関数
- 5.2 ネットワーク訓練
- 5.3 誤差逆伝播（でんぱ）
- 5.4 ヘッセ行列
- 5.5 ニューラルネットワークの正則化
- 5.6 混合密度ネットワーク
- ５７ ベイズニューラルネットワーク

本当の目次

- フィードフォワードネットワーク(階層型NN)
- 活性化関数
- 誤差逆伝播法
- 誤差逆伝播法の評価
- NNの正規化と不変性のための方策
- RでNNを実装
- まとめ

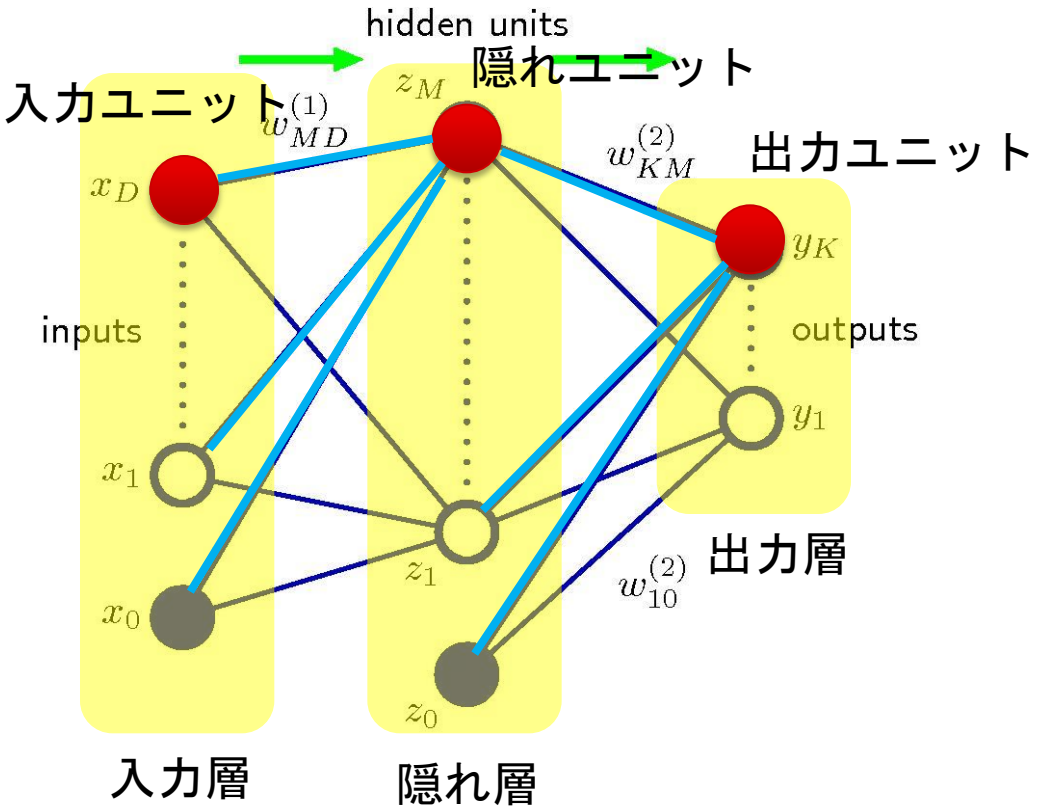
ニューラルネットワークとは (Wikipediaより）

- ニューラルネットワーク(Neural network)は、脳機能に見られるいくつかの特性を計算機上のシミュレーションによって表現することを目指した数学モデルである。生物学や神経科学との区別のため、人工ニューラルネットワークとも呼ばれる。
- ニューラルネットワークは、教師信号（正解）の入力によって問題に最適化されていく教師あり学習と、教師信号を必要としない教師なし学習に分けられる。明確な解答が用意される場合には教師あり学習が、データ・クラスタリングには教師なし学習が用いられる。結果としていずれも次元削減されるため、画像や統計など多次元量のデータでかつ線形分離不可能な問題に対して、比較的小さい計算量で良好な解を得られることが多い。このことから、パターン認識やデータマイニングをはじめ、さまざまな分野において応用されている。

簡単に言うと
「すごく非線形なロジスティック回帰」

5.1フィードフォワードネットワーク関数

- ニューラルネットワークの基本モデル



$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

活性　重みパラメータ　入力変数　バイアスパラメータ

活性は非線形活性化関数h(・)で変換され

$$z_j = h(a_j)$$

出力ユニットでも同様に線形和

$$a_k = \sum_{i=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

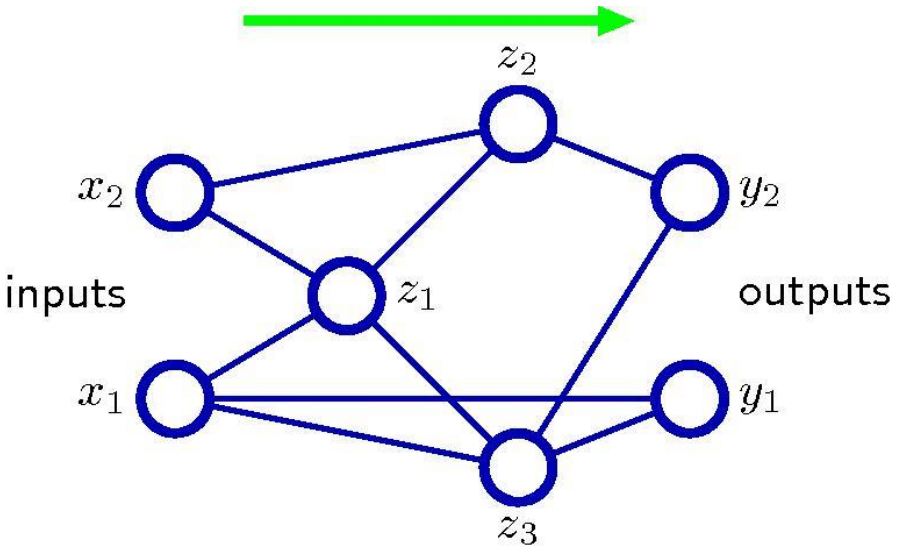
出力ユニット活性は適当な活性化関数で変換されてネットワークの出力となる

$$y_k = \sigma(a_k)$$

5.1フィードフォワードネットワーク関数

- これらをまとめるとネットワーク全体の関数は

$$y_k(\mathbf{x},\mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

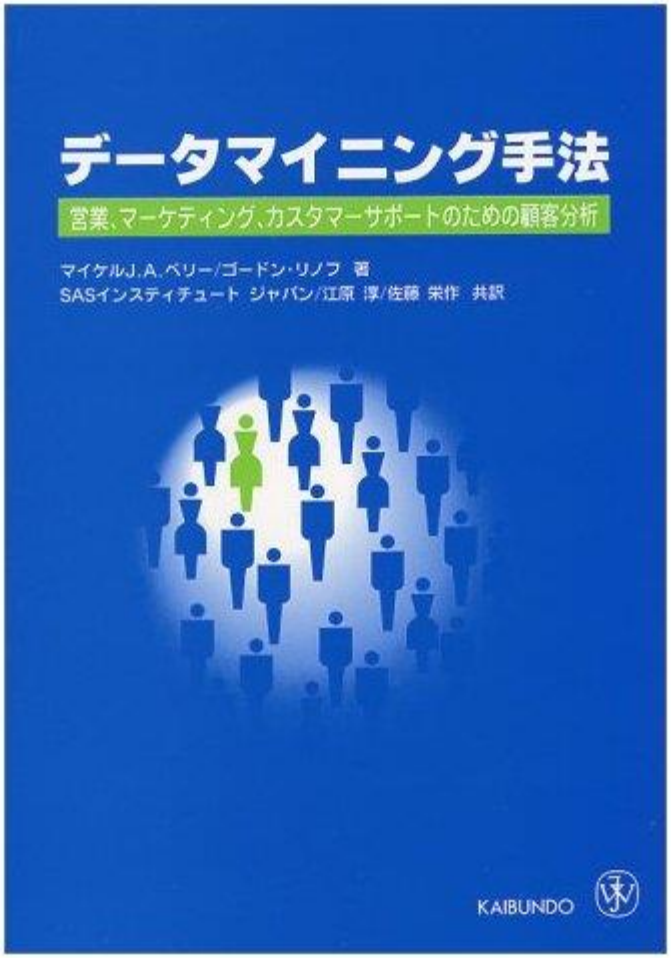


ネットワーク上の情報は順伝播(forward propagation)すると解釈する.

ネットワーク図は数学的な関数と直接対応しているので，複雑なネットワークはより一般的な写像に発展させることができる.

しかし，フィードフォワード(feed-forward)構造でなければならない.
つまり，閉じた有向回路があってはダメ

別の本に脱線して説明を試みる



ニューラルネットワークは予測や見積もりを計算する問題に適している．適用が適切な問題には次の3つの特徴がある．

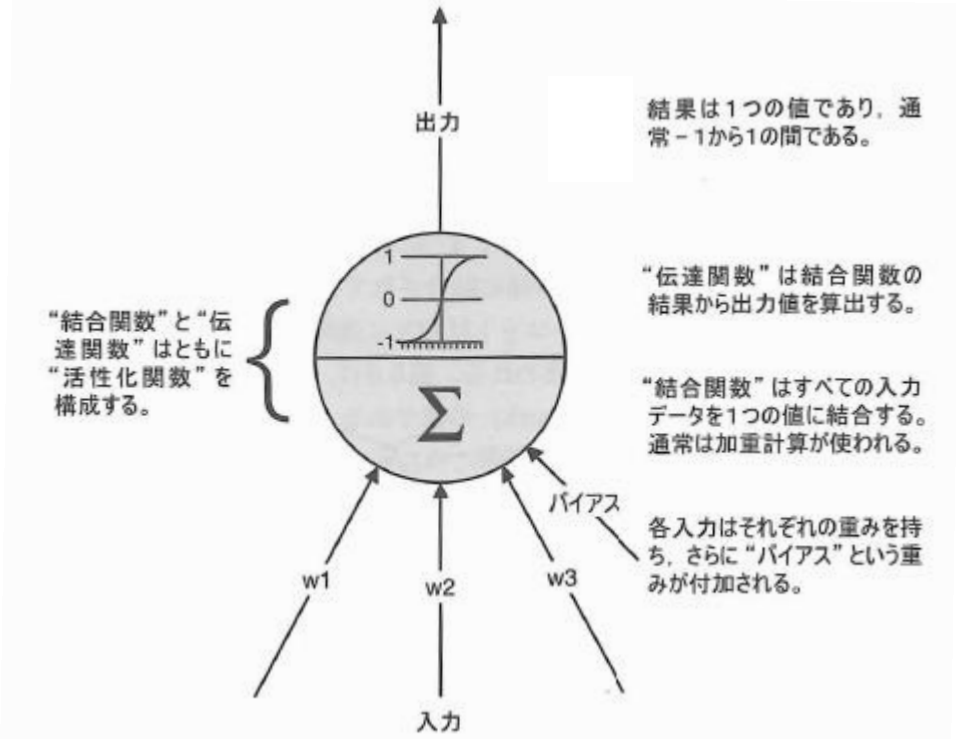
（1）入力データを十分に理解していること
（2）出力データを十分理解していること．つまり何を予測したいのかはっきりしていること
（3）経験を利用できること．別の分析で結果が知られている場合，学習に利用しやすい．

階層型ニューラルネットワーク

- さきほどのfeed-forward neural networkは階層型ニューラルネットワークとも呼ばれる.
- これを理解するには次の３つの問いが重要

- どんなユニットがどう動いているのか？
  つまり，活性化関数とは何か？
- どのようにしてユニットは連結されるのか？
  つまり，ネットワークの位相とは何か？
- どのようにしてネットワークは認知パターンを学習するのか？つまり，誤差逆伝播法とは何か？

活性化関数（１）



- ユニットは複数の入力を１つの出力値に結合する
- この結合を活性化関数(activation function)と呼ぶ
- これは非線形であり，これこそNNの強力かつ複雑さの由縁
- 活性化関数は「結合関数(combination function)」と「伝達関数(transfer function)」に分けて考えることができる

結合関数

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

活性　重みパラメータ　入力変数　バイアスパラメータ

-1から1までの入力値を重み付けて結合する（-1から1の間に）

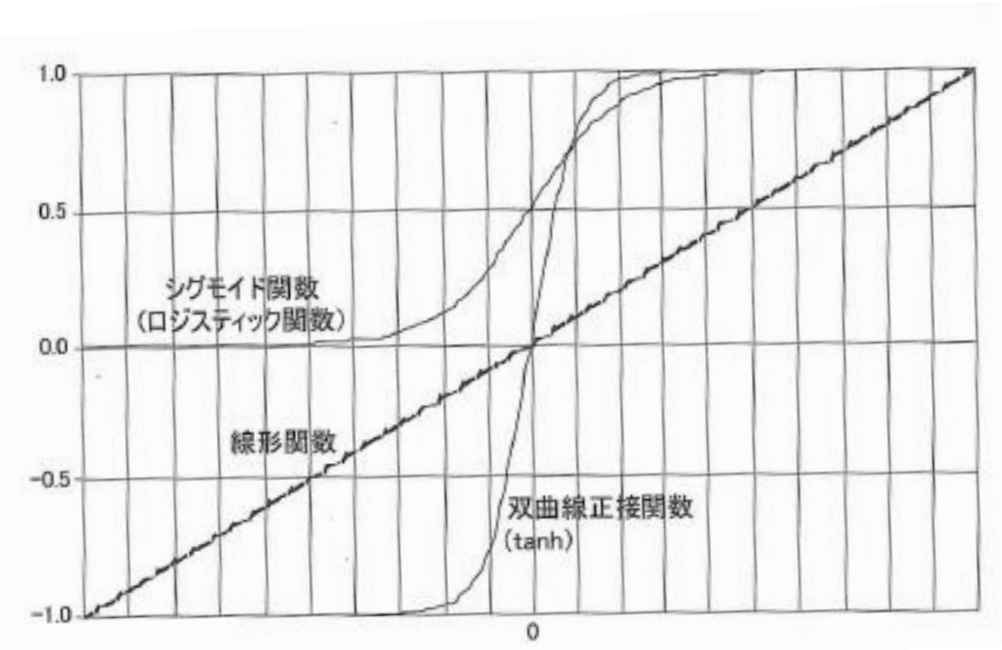
伝達関数

$$z_j = h(a_j)$$

-1から1までの値を別の-1から1の間の値に変換して出力する

活性化関数（２）～伝達関数～

- 伝達関数は伝統的にシグモイド関数，線形関数，双曲線正接関数(hyperbolic tangent)が用いられることが多い．が，別に何を用いても良いらしい．



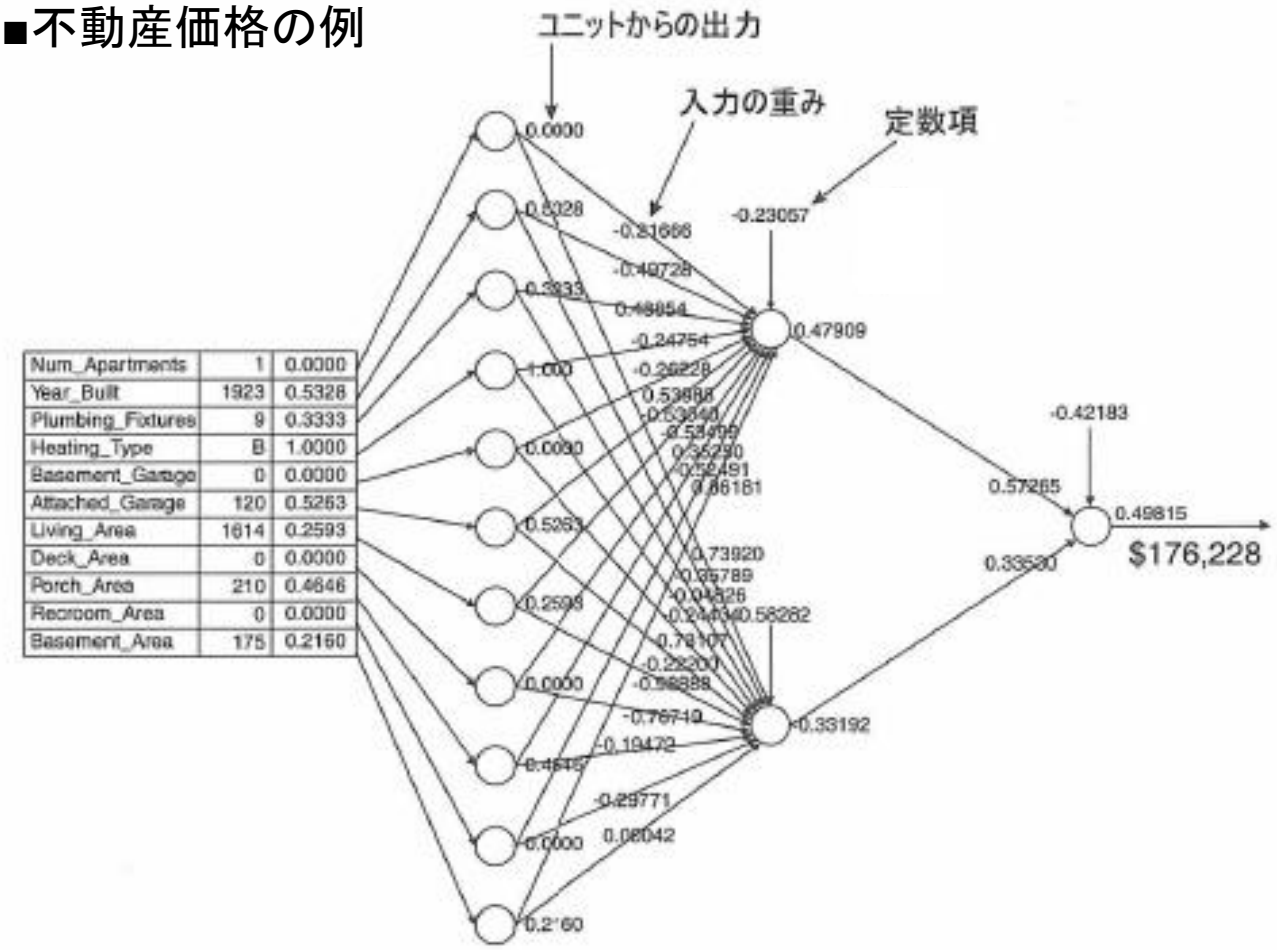
$$h(x) = \mathrm{logistic}(x) = \frac{1}{1+e^{-x}}$$

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

Pattern Recognition and Machine Learning

ネットワークの位相

■不動産価格の例



- 各層は入力層，隠れ層，出力層となっている
- 最初に入力値を-1から1の間に加工する必要がある
- 各層の繋がりが回帰（線形回帰やロジスティック回帰）のようになっている
- 隠れ層の存在が統計モデルをうまく調整している
- -1から1の間の出力値を解釈しやすい値に変換し直す

誤差逆伝播法（１）

- 誤差逆伝播法(back propagation)は求めたい出力に近い値を得るための重みを調整する優れた手法

- ネットワークは学習用データを入手し，ネットワークの既存の重みを使って，出力値を計算する
- 算出した結果と期待した結果（実際の値）との差から誤差を算出する.
- ネットワークを通じて誤差をフィードバックし，誤差が最小になるように重みを調整する（誤差がネットワークを通じて送り返されるがゆえに誤差逆伝播法と呼ばれる）

誤差逆伝播法（２）

- ユニットは重みをどのように調整するのか？
- まず，各入力値に対する重みが変化することによってどれくらい誤差が増減するかが測られる
- 各ユニットにおいて誤差が減少するよう，それぞれの重みを調整する
- 学習用データの各サンプルに対して重みを調整することにより，ゆっくりと重みは最適値に近づいていく
- この手法は一般化デルタルールと呼ばれる

5.3 誤差逆伝播

- PRMLに復帰
- この節の目的はNNにおける誤差関数E(w)の勾配を効率よく評価するテクニックを見つけること
- 多くの誤差関数は訓練集合の各データに対応する誤差項の和，すなわち

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

- １つの項の勾配 $\nabla E_n(\mathbf{w})$ を評価する問題を考える

5.3.1 誤差関数微分の評価（１）

- 出力y_kが入力変数x_iの線形和 $y_k = \sum_i w_{ki} x_i$
- で，誤差関数が次の形のときを考える

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2$$

- この誤差関数の重みw_{ji}における勾配は

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni}$$

出力側の「誤差信号」と入力側の「変数」の積という
局所的な計算と解釈できる

- この単純な結果をより複雑なNNに拡張しよう

5.3.1 誤差関数微分の評価（２）

- 一般のフィードフォワードネットワークではそれぞれのユニットは
  $a_j = \sum_i w_{ji} z_i$ (5.48) の形の入力の重み付き和を計算する
- この和は非線形活性化関数h(・)によって変換され，出力は
  $z_j = h(a_j)$ (5.49) の形で与えられる
- 訓練集合によって上記を適用することですべてのユニットの出力が計算されているとしよう．これを順伝播と呼ぶ.
- ここで，誤差関数の重みに対する微分の評価を考えよう
  $\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}$ (5.50) であり，$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$ (5.51) を導入し，δを誤差と呼ぶ
- (5.48)を用いると $\frac{\partial a_j}{\partial w_{ji}} = z_i$ (5.52) であり，(5.51)(5.52)を(5.50)に代入すると
  $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$ (5.53) が得られる

5.3.1 誤差関数微分の評価（３）

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i \qquad (5.53)$$

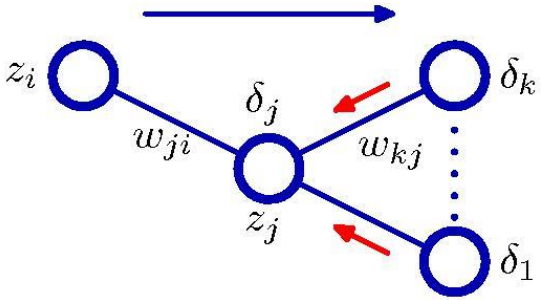
- 式(5.53)よりある重みの出力側のユニットのδと重みの入力側のユニットのzの値を掛け合わせるだけで，必要な微分が得られる.
- これは最初に考えた線型モデルと同じ形をしている！
- したがって，微分を評価するにはネットワークの各隠れユニットおよび出力ユニットのδjの値を計算し，(5.53)を適用するだけでよい！
- 隠れユニットのδを評価するには再度，適用すればよく

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \qquad (5.55)$$

となる．(5.48)(5.49)を用いると

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k \qquad (5.56)$$

という逆伝播の公式が得られる



5.3.1 誤差関数微分の評価（４）

1. 入力ベクトルx_nをネットワークにいれ，(5.48)(5,49)を用いてネットワーク上を順伝播させ，すべての隠れユニットと出力ユニットの出力を求める
$$a_j = \sum_i w_{ji} z_i \quad (5.48) \qquad z_j = h(a_j) \quad (5.49)$$
2. (5.54)を用いてすべての出力ユニットのδ_kを評価する
$$\delta_k = y_k - t_k \quad (5.54)$$
3. (5.56)を用いてδを逆伝播させ，ネットワークの全ての隠れユニットのδ_jを得る
$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \quad (5.56)$$
4. (5.53)を用いて必要な微分を評価する
$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i \quad (5.53)$$

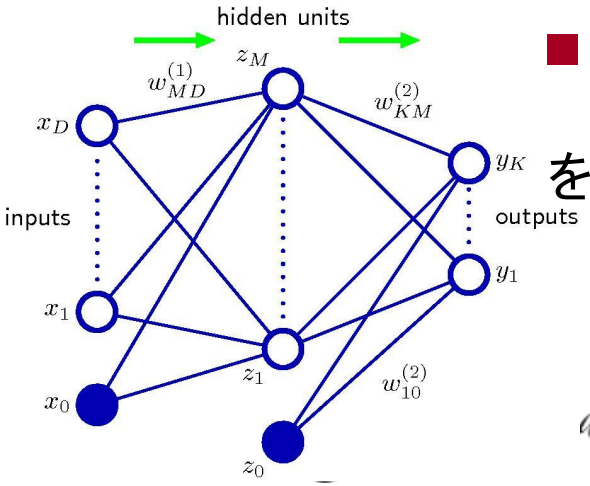
5.3.2 単純な例

- 単純な例：２層ネットワークで二乗和の誤差関数を持ち，出力ユニットは線形活性化関数，隠れユニットはシグモイド活性化関数を持つもので誤差逆伝播法の有効性を示す

$$y_k = a_k \quad h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \quad (5.59) \qquad h'(a) = 1 - h(a)^2 \quad (5.60) \qquad E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2 \quad (5.61)$$

- ここで，訓練集合の各パターンに対し，順伝播を実施する

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \quad (5.62) \qquad z_j = \tanh(a_j) \quad (5.63) \qquad y_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j \quad (5.64) \qquad \delta_k = y_k - t_k \quad (5.65)$$



- 次に誤差δkを逆伝播させ $\delta_j = \left(1 - z_j^2\right)\sum_{k=1}^{K} w_{kj}\delta_k$ (5.66)

を用いて隠れユニットのδを得る．最後に重みの微分は

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j \quad (5.67)$$

5.4 ヤコビ行列・ヘッセ行列

- 逆伝播法のテクニックは他の微分計算にも応用可能

$$J_f = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}$$

$$H_f = \begin{pmatrix} \frac{\partial^2 f}{\partial^2 x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial^2 x_n} \end{pmatrix}$$

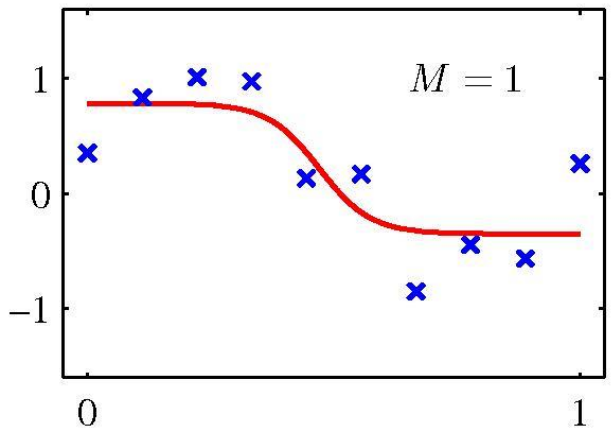
- などを計算するときにも役立つ…ようである

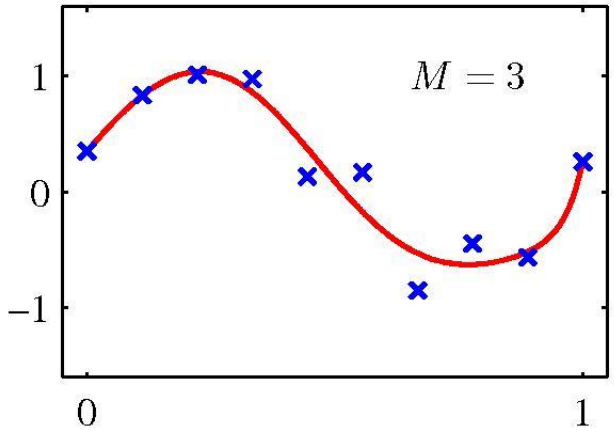
１回目のまとめ

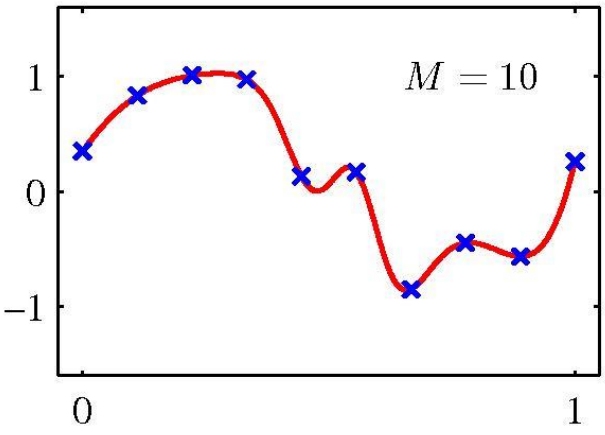
- ニューラルネットワークは簡単に言えば「すごく非線形なロジスティック回帰分析」
- 入力値→入力値の重み付け線形和→ロジスティック関数にぶちこむ→出力値
- 重み付け線形和が「結合関数」
- ロジスティック関数が「伝達関数」
- 各重み付け（パラメータ）を決定するのは最終的な出力値の誤差から逆に流していくことで各誤差を小さくしていく
- これが誤差逆伝播法
- ということなので，伝達関数を線形にして，隠れ層をなくせばただの線形回帰分析だったりする

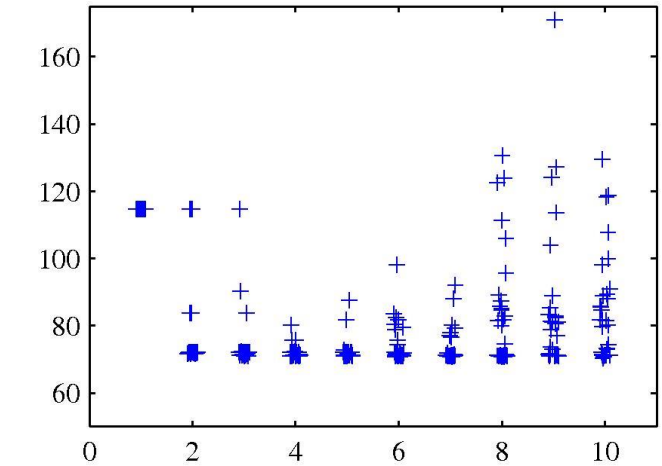
5.5 NNの正則化

- 入力ユニット数と出力ユニット数はデータから決まる
  →隠れユニット数Mはどう決めればいいのか？









誤差関数は局所的極小点があるので，汎化誤差はMの単純な関数にはならない．
Mを選ぶアプローチとしては実際に左図のようなグラフをプロットした上で検証用集合に対して誤差を最小とする特定の解を選ぶのが良い方法

5.5 NNの正則化（２）

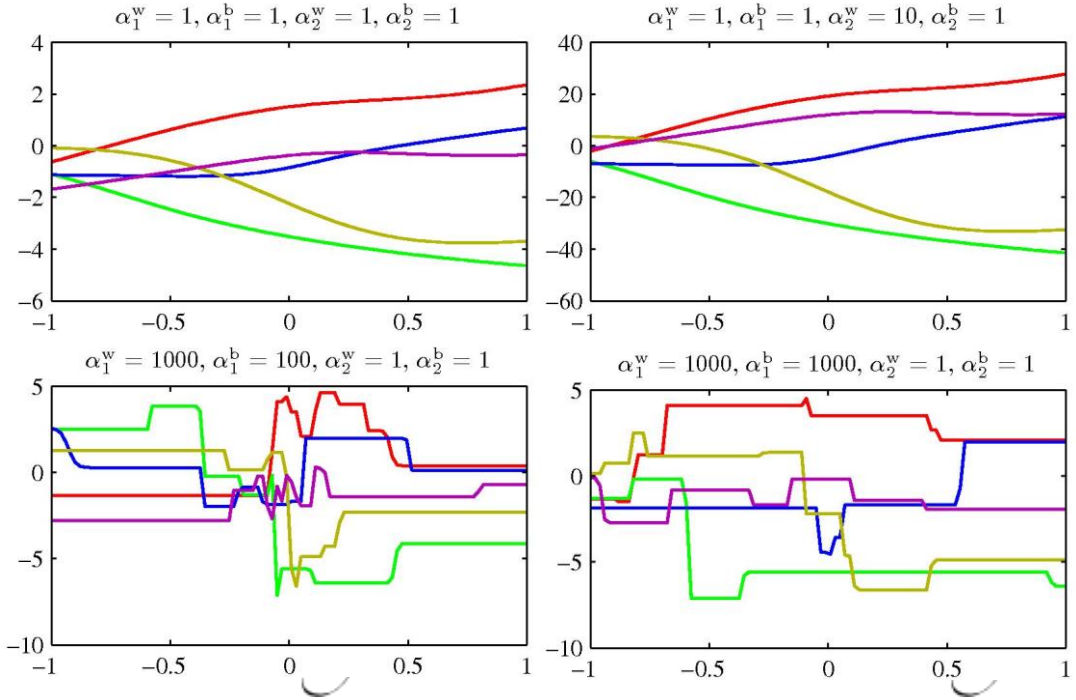
- 過学習を避ける別の方法として１章であったように，正則化項を誤差関数に追加することで複雑さを制御するアプローチも存在

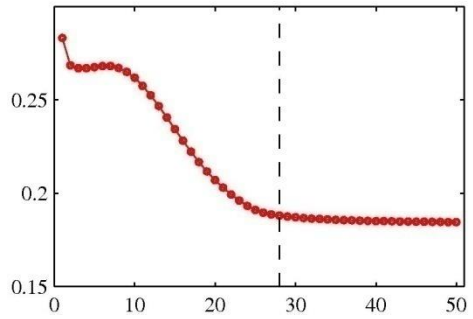
$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

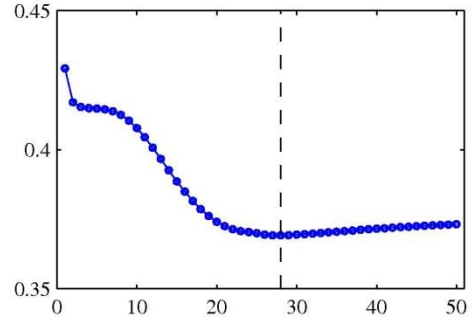
- この正則化項は荷重減衰(weight decay)と呼ばれ，3章で詳細に議論した．この場合モデルの複雑さは正則化係数λの選択で定まる．

5.5.2 早期終了

- ネットワークの複雑さを制御する方法として早期終了
- 訓練集合における誤差は反復回数の非増加関数
- しかし，一般に検証集合の誤差は過学習していると増加
- 訓練集合の誤差が最小値に達する前に訓練を止めることは実効的なネットワークの複雑さを制限する方法の１つ







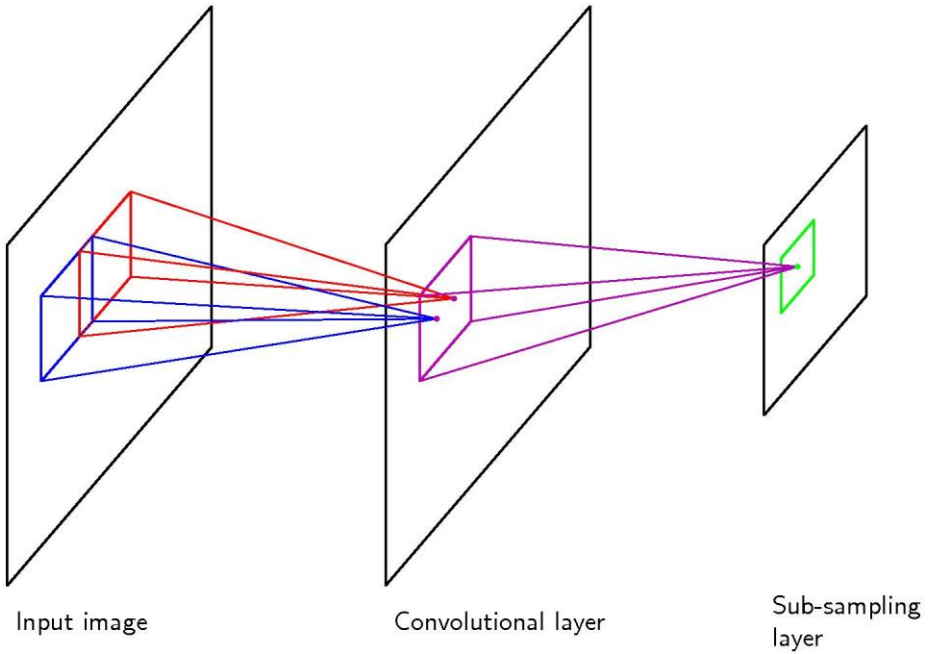
5.5.3 不変性

- 予測は入力変数がある変換を受けても変化しない不変性が求められる(ex. 平行移動不変性，尺度不変性)
- 十分な訓練集合があればOK
- 訓練集合の数が限られている場合，不変性がいくつも含まれている場合，実用的でない

1. 不変性ごとに訓練パターンを変換して複製を作成して訓練集合を増加させる→計算量が増大
2. 正則化項を誤差関数に加えて入力の変換に対して出力が変化した場合にペナルティ→接線伝播法
3. 前処理段階で特徴抽出→職人芸的
4. NN構造に不変性を構築する→たたみ込みNN

5.5.6 たたみ込みNN

- convolutional neural network
- 画像の近くの画素は遠くの画素よりも強い相関を持つ
- 局所的な特徴を抽出→高次の特徴検出→画像の情報
- たたみ込みNN
- (1)局所的受容野
- (2)重み共有
- (3)部分サンプリング

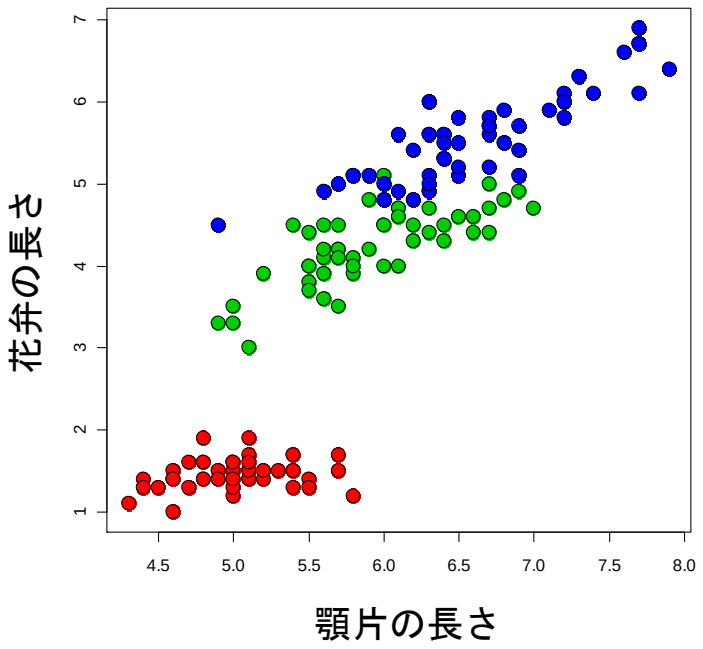


２回目のまとめ

- 隠れユニット数の決め方や過学習の避け方
- NNにとって不変性を持つということが重要な性質
- そのための方法がいくつか提案されている
- 接線伝播法
- たたみ込みニューラルネットワーク
- 混合密度ネットワーク，ベイズニューラルネットワークは各自で

RでNNを実装（１）

- 細かい話はわからんが，大まかにNNとは何かがわかったはず…
- ということでRで実装しよう



- データはRに用意されているirisデータ
- setosa, versicolaor, virginicaというアヤメの花におけるそれぞれ50標本データ
- 顎片の長さ(Sepal.Length)
- 顎片の幅(Sepal.Width)
- 花弁の長さ(Petal.Length)
- 花弁の幅(Petal.Width)
- を計測した全150標本データ

RでNNを実装（２）

- 150サンプルのうち奇数番目を訓練集合に，偶数番目を検証集合にしてみる
- 各説明変数からクラス分けを行う
- 隠れユニット数を3つ，反復計算100回に設定
- nnetパッケージを利用

１回目

```
# weights:  27
initial  value 93.994176
iter  10 value 80.280294
iter  20 value 41.496061
iter  30 value 24.513165
iter  40 value 19.263561
iter  50 value 18.710379
iter  60 value 18.448083
iter  70 value 17.977434
iter  80 value 17.738539
final  value 17.732462
converged
```

２回目

```
# weights:  27
initial  value 88.146736
iter  10 value 44.410065
iter  20 value 19.912025
iter  30 value 18.134131
iter  40 value 17.690056
iter  50 value 17.557055
iter  60 value 17.485894
iter  70 value 17.432931
final  value 17.432930
converged
```

３回目

```
# weights:  27
initial  value 89.594150
iter  10 value 47.731864
iter  20 value 29.490810
iter  30 value 20.122875
iter  40 value 19.211706
iter  50 value 19.146616
iter  60 value 19.116675
iter  70 value 18.060390
iter  80 value 17.732479
final  value 17.732461
converged
```

４回目

```
# weights:  27
initial  value 80.767635
iter  10 value 30.069632
iter  20 value 18.860712
iter  30 value 17.736227
iter  40 value 17.732475
final  value 17.732462
converged
```

非常に複雑な関数形なので微妙に最終誤差二乗和が異なっているが，全て収束しているので，実用上はおそらく問題にしない…に違いない

RでNNを実装（３）

- こういうものらしい…

１回目

```
a 4-3-3 network with 27 weights
options were - softmax modelling  decay=0.1
 b->h1 i1->h1 i2->h1 i3->h1 i4->h1
  1.99   1.60   1.69  -2.60  -2.77
 b->h2 i1->h2 i2->h2 i3->h2 i4->h2
  0.22   0.37   1.11  -1.91  -0.77
 b->h3 i1->h3 i2->h3 i3->h3 i4->h3
 -0.22  -0.34  -0.89   1.40   0.65
 b->o1 h1->o1 h2->o1 h3->o1
 -0.09   1.01   2.92  -2.74
 b->o2 h1->o2 h2->o2 h3->o2
 -1.11   3.28  -2.62   0.84
 b->o3 h1->o3 h2->o3 h3->o3
  1.20  -4.29  -0.31   1.90
```

２回目

```
a 4-3-3 network with 27 weights
options were - softmax modelling  decay=0.1
 b->h1 i1->h1 i2->h1 i3->h1 i4->h1
 -0.20  -0.34  -1.05   1.83   0.73
 b->h2 i1->h2 i2->h2 i3->h2 i4->h2
 -2.09  -1.57  -1.69   2.55   2.74
 b->h3 i1->h3 i2->h3 i3->h3 i4->h3
  0.20   0.36   1.01  -1.69  -0.69
 b->o1 h1->o1 h2->o1 h3->o1
  0.51  -2.37  -0.94   2.87
 b->o2 h1->o2 h2->o2 h3->o2
  0.54   2.51  -3.47  -1.80
 b->o3 h1->o3 h2->o3 h3->o3
 -1.05  -0.13   4.41  -1.06
```

３回目

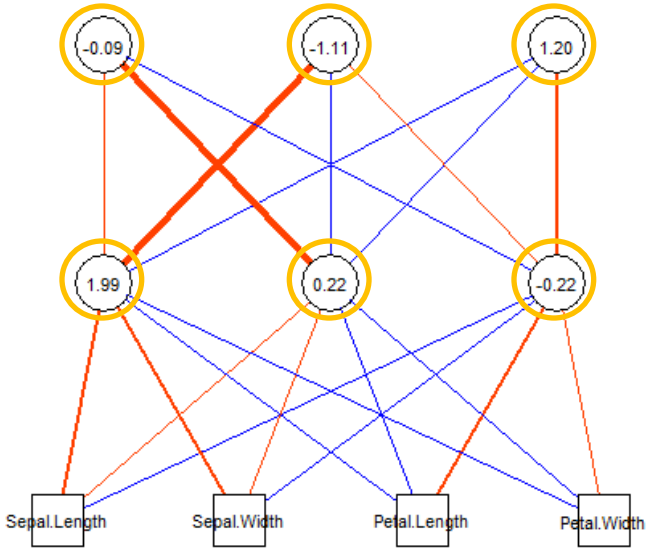
```
a 4-3-3 network with 27 weights
options were - softmax modelling  decay=0.1
 b->h1 i1->h1 i2->h1 i3->h1 i4->h1
 -0.22  -0.34  -0.88   1.40   0.65
 b->h2 i1->h2 i2->h2 i3->h2 i4->h2
  1.99   1.60   1.69  -2.59  -2.77
 b->h3 i1->h3 i2->h3 i3->h3 i4->h3
  0.22   0.37   1.11  -1.91  -0.77
 b->o1 h1->o1 h2->o1 h3->o1
 -0.09  -2.74   1.01   2.92
 b->o2 h1->o2 h2->o2 h3->o2
 -1.11   0.84   3.28  -2.62
 b->o3 h1->o3 h2->o3 h3->o3
  1.20   1.90  -4.29  -0.31
```

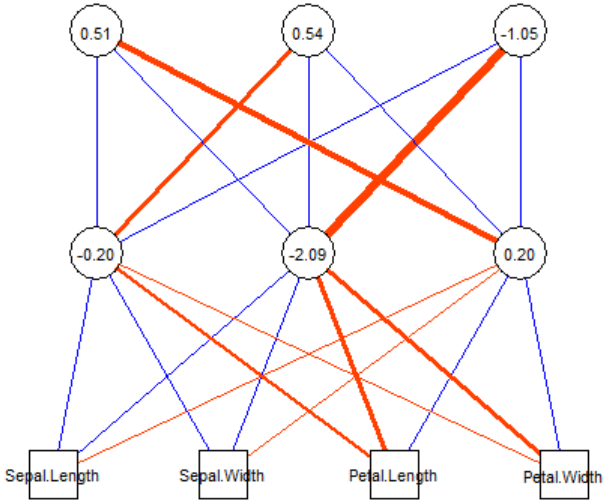
４回目

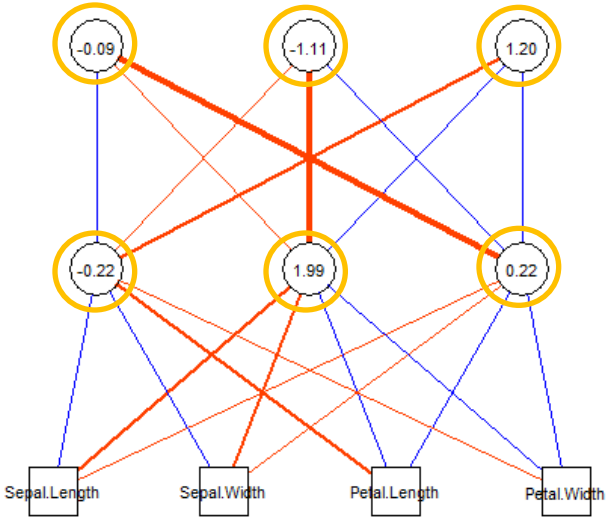
```
a 4-3-3 network with 27 weights
options were - softmax modelling  decay=0.1
 b->h1 i1->h1 i2->h1 i3->h1 i4->h1
  0.22   0.37   1.11  -1.91  -0.77
 b->h2 i1->h2 i2->h2 i3->h2 i4->h2
 -0.22  -0.34  -0.89   1.40   0.65
 b->h3 i1->h3 i2->h3 i3->h3 i4->h3
  1.99   1.60   1.69  -2.60  -2.77
 b->o1 h1->o1 h2->o1 h3->o1
 -0.09   2.92  -2.74   1.01
 b->o2 h1->o2 h2->o2 h3->o2
 -1.11  -2.62   0.84   3.28
 b->o3 h1->o3 h2->o3 h3->o3
  1.20  -0.31   1.90  -4.29
```

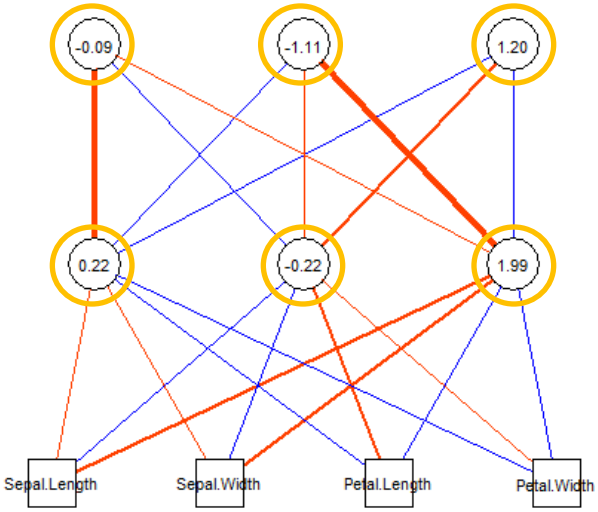
RでNNを実装（４）

- NN構造を表示（北大の久保先生のコード利用）









RでNNを実装（５）

- 検証集合による識別結果を見る限り，どれも誤識別が2つであり，さきほどの違いは出力結果に対してそれほど影響を与えないようである

iris.nnetp1

	setosa	versicolor	virginica
setosa	25	0	0
versicolor	0	23	2
virginica	0	0	25

iris.nnetp2

	setosa	versicolor	virginica
setosa	25	0	0
versicolor	0	23	2
virginica	0	0	25

iris.nnetp3

	setosa	versicolor	virginica
setosa	25	0	0
versicolor	0	23	2
virginica	0	0	25

iris.nnetp4

	setosa	versicolor	virginica
setosa	25	0	0
versicolor	0	23	2
virginica	0	0	25

RでNNの実装と解釈とまとめ

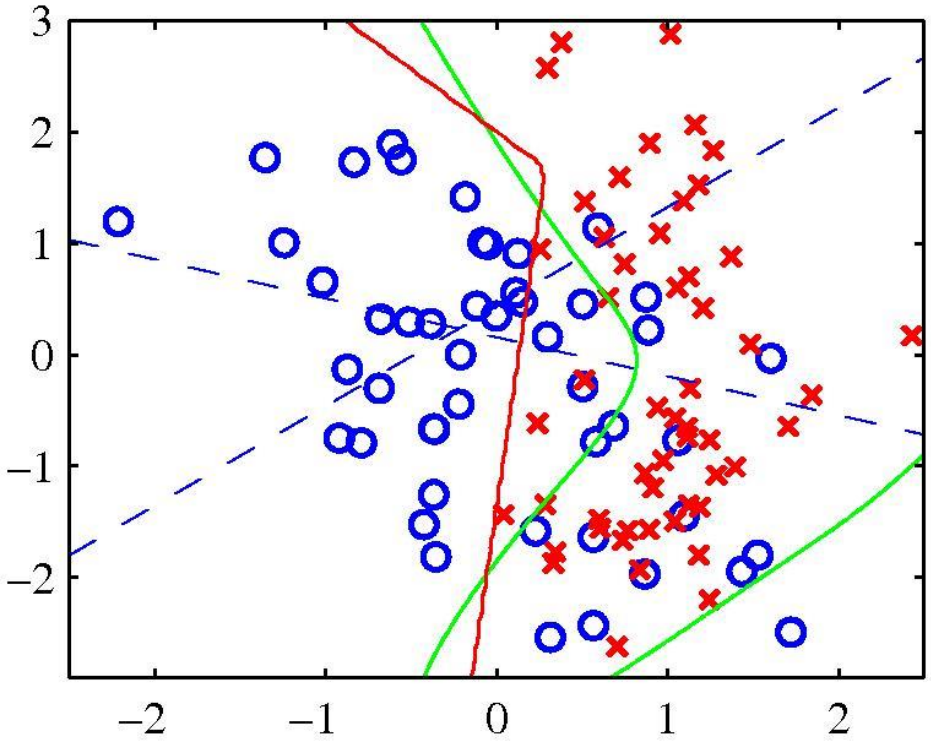
- パッケージのおかげで簡単に実装可能
- NNは機械学習らしい情緒のなさ
- 目的こそ先にありき
- 理解よりも予測の世界
- それこそ画像認識や音声認識など各入力変数（こっちの分野でいう説明変数）が解釈しにくく，でもなにか予測・分類したい場合に用いるが吉
- こっちの分野だと説明変数が解釈するし，推定したパラメータも解釈しようとするし．なんか出ました…じゃ許されない
- けれども，機械学習的な問題もあるはず
- 頭を使った意思決定は説明が必要だが，微視的な意思決定は予測でいいのかも？歩行速度モデル，歩数モデル，1秒後の存在地予測モデル，行動パターンクラスタリング，BCALs…

ここより後ろは残骸

NNとSVM

- SVM：訓練自体が非線形最適化を含んでいるが目的関数が常に凸となるため，最適化が比較的簡単
- フィードフォワードニューラルネットワーク（多層パーセプトロン）：事前に基底関数の数を固定してパラメトリック形を用いて，パラメータ値を訓練流に適応させる
- →SVMに比べてコンパクトで評価も迅速であるが尤度関数が凸関数にならない

２クラス分類問題にNNを利用した例



- 細い点線が各隠れユニットのz=0.5の等高線を示し，赤線はネットワークのy=0.5の決定面を示す.
- 比較のためにデータを生成した分布から計算される最適な決定境界が緑線で示す

２．ネットワーク訓練

- ネットワークパラメータの決定問題
- 二乗和誤差関数を最小化すること
- 入力ベクトル{xn}，目標ベクトル{tn}とすると誤差関数

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\| y(\mathbf{x_n}, \mathbf{w}) - t_n \right\|^2$$ を最小化すればよい

たとえば回帰問題であれば $$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{ y(\mathbf{x_n}, \mathbf{w}) - t_n \right\}^2$$

２クラス分類問題であれば4.3.6節の正準連結関数の議論に従い，ロジスティックシグモイド関数を活性化関数として，

$$y = \sigma(a) = \frac{1}{1+\exp(-a)} \qquad p(t \mid \mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \left\{1 - y(\mathbf{x}, \mathbf{w})\right\}^{1-t}$$

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\left\{ t_n \ln y_n + (1-t_n)\ln(1-y_n) \right\}$$ という交差エントロピー誤差関数を最小化すればよい

２．ネットワーク訓練

- 標準的な多クラス分類問題

誤差関数 $$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_k(\mathbf{x_n}, \mathbf{w})$$

出力ユニットの活性化関数はソフトマックス関数

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))}$$