第17回 順伝播型ニューラルネットワーク

Code

1. 概要

順伝播型ニューラルネットワーク（feedfoward neural network）は、ある変数を他の変数から予測する式を作る手法の一つです。目的変数が量的変数の場合に用いるニューラルネットワーク回帰と目的変数が質的変数のときに用いるニューラルネットワーク分類があります。今回は、特に2種類のラベルを目的変数としたニューラルネットワーク分類に絞って解説します。

2. ニューラルネットワーク分類のデモ

2.1 パッケージのインストール

今回は、 `neuralnet` パッケージを用います。このパッケージはCRANには公開されていない（つまり `install.packages` 関数ではインストールできない）最新版のパッケージを用いる必要があるため、次のような方法でインストールします。

Hide

```
# パッケージのインストールと読み込み
install.packages("devtools")
devtools::install_github("bips-hb/neuralnet")    # 最新版のneuralnetパッケージ
```

インストールが完了したら、 `neuralnet` パッケージを読み込みましょう。

Hide

```
library(neuralnet)
```

2.2 デモデータ

`data` ディレクトリの `xor_demo_train.csv` をデモデータに用います。このデータは、標本サイズが300で3変数のデータです。 `x1` と `x2` は量的変数、 `y` は `1` と `2` からなる2値の質的変数です。今回は、 `x1` , `x2` を説明変数、 `y` を目的変数としたニューラルネットワーク分類を行ってみます。

Hide

```
train <- read.csv("./data/xor_demo_train.csv",
                  fileEncoding = "utf-8")
train$y <- as.factor(train$y)
head(x = train, n = 5)
```

	x1 <dbl>	x2 <dbl>	y <fctr>
1	1.619380	1.5255072	1
2	0.368291	1.6351876	1

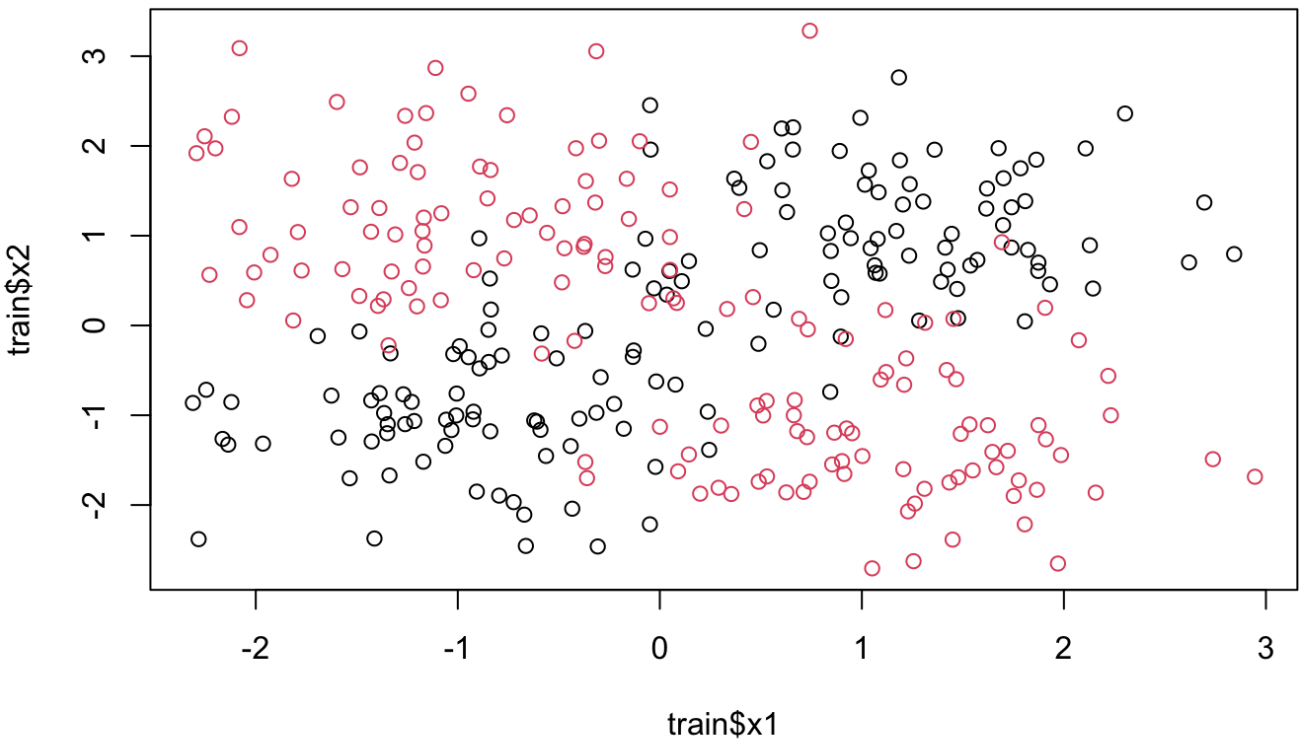
	x1 <dbl>	x2 <dbl>	y <fctr>
3	1.087715	0.5777269	1
4	1.571684	0.7303400	1
5	0.603115	2.1941157	1

5 rows

y の値に応じて色分けした散布図をかいて、分布の様子を確認しておきましょう。

Hide

```
plot(train$x1, train$x2, col = train$y)
```



問題：2値の質的変数からなる目的変数 y を説明変数 x を用いて予測するとき、予測が切り替わる境界線を**決定境界**（decision boundary）といいます。今回のデータに対して適切な決定境界を事前に想定しておきましょう。

2.3 ニューラルネットワーク分類の計算

ニューラルネットワーク分類は neuralnet パッケージの neuralnet 関数で計算することが出来ます。指定する引数は以下のとおりです。

- formula ：目的変数と説明変数を指定する。
- data ：サポートベクトル分類を作るために用いるデータ。
- hidden ：隠れ層とそのユニットの数
- linear.output ：特殊な引数のため、解説を省略します。今回は FALSE に指定しておいてください。
- act.fct ：隠れ層の活性化関数
- output.act.fct ：出力層の活性化関数
- algorithm ：パラメータを求めるためのアルゴリズムの指定

- `err.fct` ：損失関数
- `stepmax` ：パラメータの更新回数の最大値
- `threshold` ：パラメータを更新したとき、損失関数の値の差がこの値を下回った場合、更新をやめる。
- `learningrate` ：更新の大きさ（学習率という）

わからない単語もあると思いますが、3節で詳しく解説します。

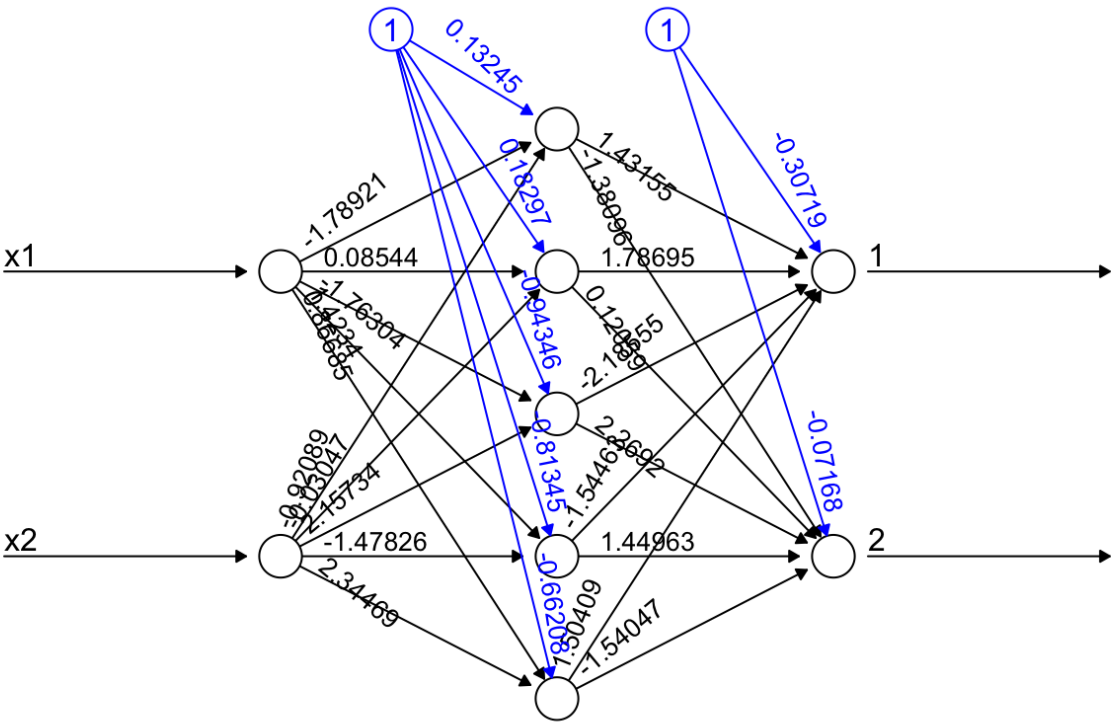
Hide

```
# 隠れ層1層のニューラルネットワークの学習
nnet <- neuralnet(formula = y ~ x1 + x2,
                  data = train,
                  # アーキテクチャの設計
                  hidden = 5,
                  linear.output = FALSE,    # act.fct, output.act.fctを使用するため
                  act.fct = "relu",
                  output.act.fct = "logistic",
                  # 学習アルゴリズムの設計
                  algorithm = "backprop",
                  err.fct = "ce",    # 2乗誤差ならsse
                  stepmax = 1e+04,
                  threshold = 0.5,
                  learningrate = 0.0005)
```

`plot` 関数を用いて得られたモデルの式を確認できます。

Hide

```
# 結果の可視化
plot(nnet)
```



2.4 決定境界の確認

ニューラルネットワーク分類で得られる決定境界の式は複雑なので、数式を確認する代わりに散布図上に図示しましょう。

Hide

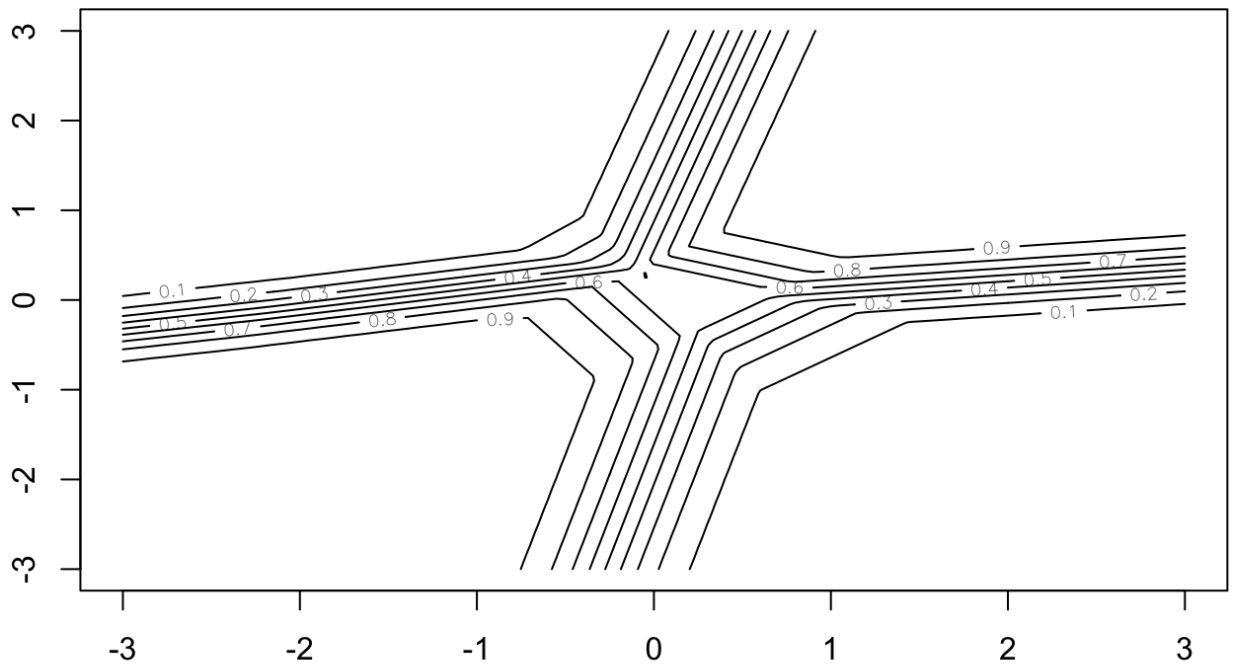
```
# メッシュグリッドの作成
xx1 <- seq(-3, 3, 0.05)
xx2 <- seq(-3, 3, 0.05)
meshgrid <- expand.grid(xx1, xx2)
names(meshgrid) <- c("x1", "x2")
head(meshgrid, n = 5)
```

	x1 <dbl>	x2 <dbl>
1	-3.00	-3
2	-2.95	-3
3	-2.90	-3
4	-2.85	-3
5	-2.80	-3

5 rows

Hide

```
# 決定関数の計算
pred <- predict(nnet, meshgrid)
contour(xx1, xx2,
        array(pred, dim = c(length(xx1), length(xx2))))
```



2.5 予測精度の確認と過剰適合・過少適合

ニューラルネットワーク分類は、隠れ層とそのユニットの数 `hidden` を調整することで、とても複雑な決定関数を作ることができます。このような場合、決定関数を求めるために用いたデータに対しては精度を確認しても、未知のデータに対する予測精度とは大きく結果が乖離してしまうことがあります。このような現象を**過剰適合**（overfit）といいます。逆に単純すぎる決定関数を計算してしまった場合を**過少適合**（underfit）といいます。

このため、一般に予測の精度を確認するためには、モデルの作成には用いないデータを別に準備しておき、このデータで予測精度を確認することが一般的です。これを**ホールドアウト法**（hold-out method）といいます。モデルの作成に用いるデータを**訓練データ**、モデルの予測精度の検討に使うデータを**テストデータ**といいます。

今回はホールドアウト法のために、 `xor_demo_test.csv` を準備しました。

Hide

```
# テストデータの読み込み
test <- read.csv("./data/xor_demo_test.csv",
                 fileEncoding = "utf-8")
test$y <- as.factor(test$y)
head(x = test, n = 5)
```

	x1 <dbl>	x2 <dbl>	y <fctr>
1	-0.03780117	2.0447945	1
2	-0.17821851	2.1147932	1
3	1.33747359	-0.5855833	1
4	1.19782619	2.3928090	1
5	0.88209690	0.8831835	1

5 rows

では、テストデータに対する予測を行い、精度を評価してみましょう。ラベルの予測の精度を評価するうえでは、正解と予測のクロス集計表をかくのが便利でしょう。これを**混同行列**（confusion matrix）といいます。

Hide

```
# テストデータでの混同行列
pred_test_proba <- predict(nnet, test)
pred_test <- apply(pred_test_proba, 1, which.max)
table(test$y, pred_test)
```

```
   pred_test
     1  2
  1 36 14
  2  7 43
```

参考のため訓練データでの混同行列も計算しておきましょう。

Hide

```
# 訓練データでの混同行列
pred_train_proba <- predict(nnet, train)
pred_train <- apply(pred_train_proba, 1, which.max)
table(train$y, pred_train)
```

```
   pred_train
      1   2
  1 134  16
  2  19 131
```

3. 順伝播型ニューラルネットワークの仕組み

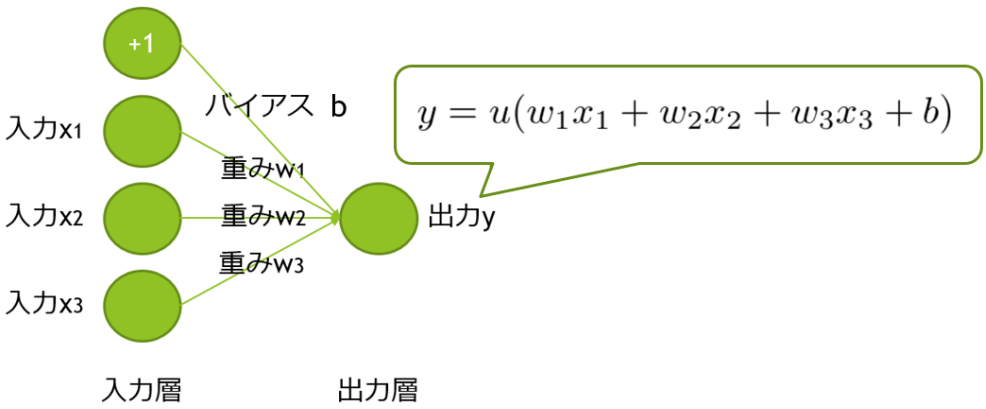
3.1 決定関数の表現の仕方

ユニット（unit）

順伝播型ニューラルネットワークの最小構成単位はユニットとよばれるものです。ユニットは、複数の値 $x_1, \cdots, x_d$ を入力されたら、次のような式を計算します。

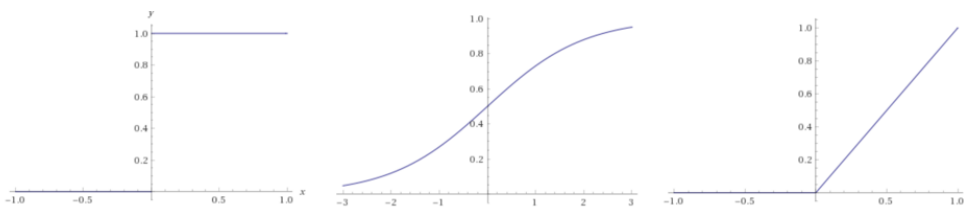
$$u(b + w_1 x_1 + \cdots + w_d x_d)$$

ここで、u は活性化関数とよばれる非線形な関数で、様々な種類のものから予め使うものを決めておきます。一方、$w_1, \cdots, w_d$ および b は重み・バイアスとよばれ、データから求めるパラメータになります。



活性化関数

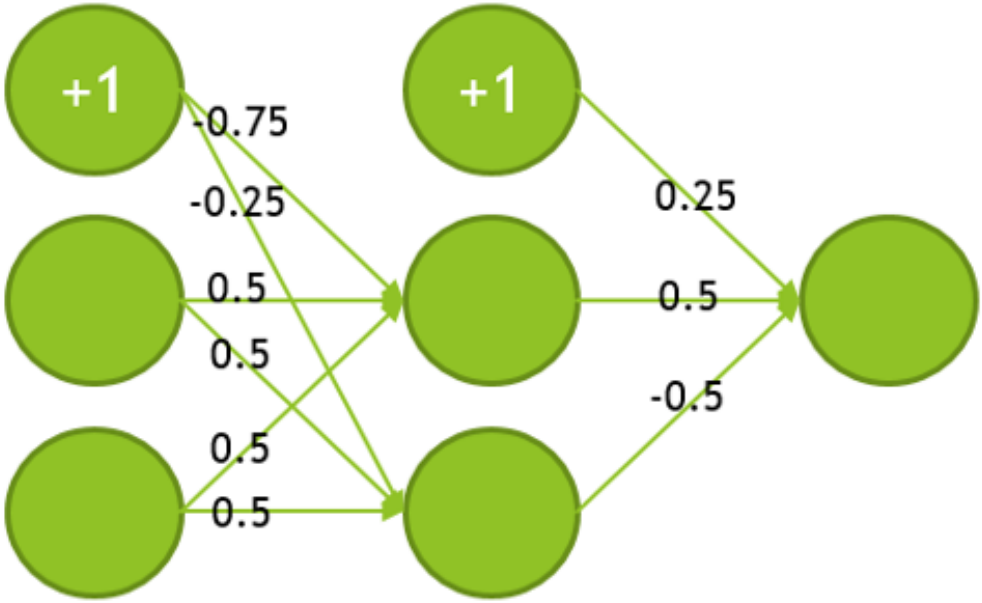
活性化関数の代表例には、ReLU（rectified linear unit）やlogistic関数（sigmoid関数ともいう）があります。それぞれのグラフを示します。



階段関数　sigmoid関数　ReLU

層（layer）

順伝播型ニューラルネットワークでは、複数のユニットをまとめた層とよばれるものを複数積むことで決定関数を定義します。数式で表すと複雑になりがちなので、次のようなグラフで表現することが一般的です。



最初の層は説明変数の値を入力する入力層、最後の層は予測値を出力する出力層といいます。入力層と出力層の間にある層を隠れ層といいます。 `neuralnet` 関数では `hidden` に値を渡すことで隠れ層のユニットの数を指定できます。複数の隠れ層を積む場合は、 `c(10, 5)` のようにかくことで、10個の隠れ層と5個の隠れ層を準備することが出来ます。

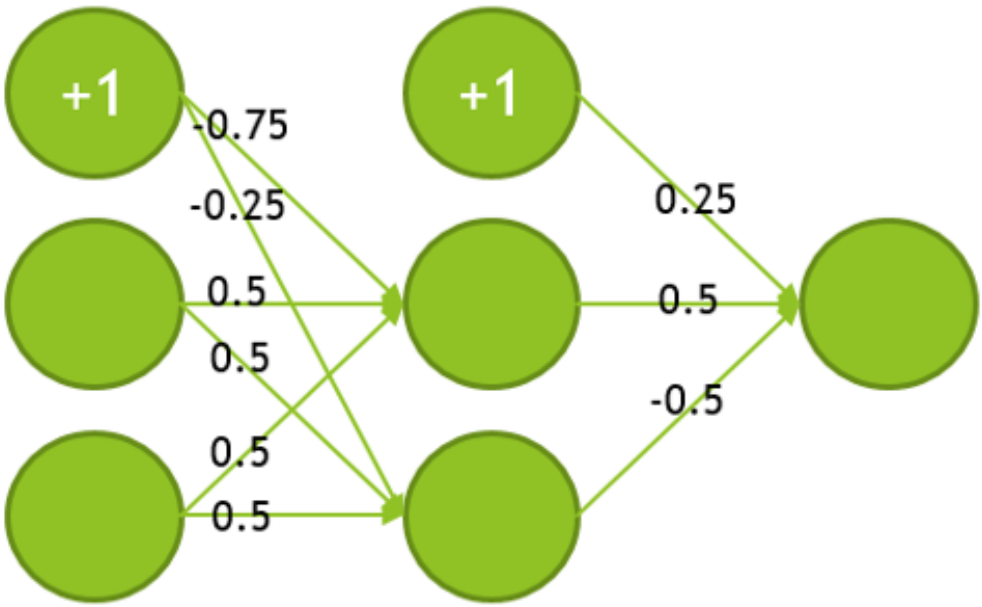
隠れ層のユニットの活性化関数にはReLUが用いられることが多いです。 `neuralnet` 関数では `act.fct` に relu` を渡すことで隠れ層のユニットの活性化関数をReLUに指定できます。

隠れ層の役割

隠れ層のユニットは、複雑な決定境界をつくるために役に立ちます。実際、ある程度の複雑な決定境界であれば、隠れ層に十分にたくさんのユニットを準備することで、所望の精度で近似できることが知られています。これを**万能近似定理**（universal approximation theorem）といいます。

次の問題を解いて、隠れ層のユニットが決定境界の複雑度に対応していることを確認してみましょう。

問：以下の順伝播型ニューラルネットワークが定める決定関数の決定境界を示してください。



Remark：同様に万能近似定理が成り立つものに、RBFカーネルのサポートベクトルマシンがあります。ただし、RBFカーネルのサポートベクトルマシンでは近似できない決定境界のなかに、順伝播型ニューラルネットワークでは近似できる関数が存在することが知られています。

3.2 パラメータの求め方

交差エントロピー損失

2値のラベル $0/1$ のどちらかの値をとる変数 y を予測するモデル $p(x; w, b)$ をニューラルネットワークで作りたいとします。このとき、予測モデル $p(x; w, b)$ の出力層の活性化関数にはlogistic関数を用いることで、$y = 1$ の確信度を出力できるようなモデルを作ります。 `neuralnet` 関数では `output.act.fct` に `logistic` を渡すことで出力層の活性化関数をlogistic関数に指定できます。

このとき、モデルの予測と目的変数 y の食い違いを測る方法として、**交差エントロピー損失**（crossentropy loss）があります。

$$l(y, p) = \begin{cases} -\log p & \text{if } y = 1 \\ -\log(1-p) & \text{if } y = 0 \end{cases}$$

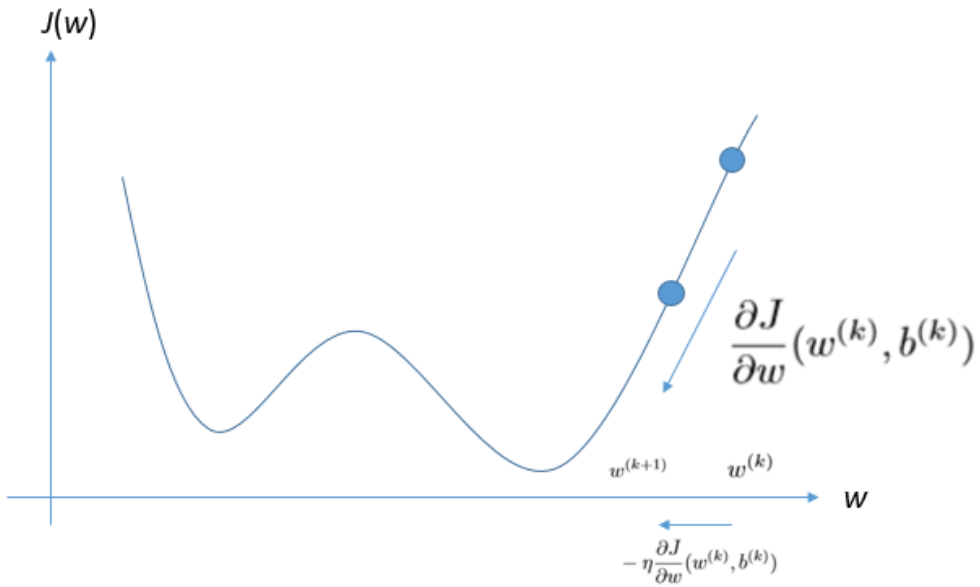
予測モデルは重み w とバイアス b によって決まります。これらの値は、与えられたデータに対して交差エントロピー損失の平均値が最小になるようなものを求めます。 `neuralnet` 関数では `err.fct` に `ce` を渡すことで損失関数を交差エントロピーに指定できます。

勾配降下法

重みとバイアスは、最初は初期値をランダムに振っておき、良い予測を返せるような値に少しずつ更新していくことで求めます。 `neuralnet` 関数では `stepmax` がこの更新の最大回数を決める引数、 `theshold` が更新前後の交差エントロピーの値の差をみて、更新を停止するための条件を与える引数になっています。

パラメータ（重みとバイアス）を求める代表的な手法は**勾配降下法**（gradient descent）です。

$$w^{(k+1)} = w^{(k)} - \eta \frac{\partial l}{\partial w}(w^{(k)})$$



`neuralnet` 関数では `algorithm` に `backprop` を渡すことで勾配降下法によるパラメータの計算が実行できます。 勾配降下法の η は**学習率**（learning rate）といい、更新の大きさを決める値です。この値は事前に自分で決めておきます。 `neuralnet` 関数では `learningrate` がこの更新の最大回数を決める引数になっています。