Logistic regression & Neural networks

内容

1. 5分钟复习
2. GLM求解
3. 从Logistic regression到Neural networks
4. Back propagation
5. 应用：手写字识别

5分钟复习

GLMs模型

y 的均值是 $X\beta$的函数

$$\mathbb{E}(y|X) = g^{-1}(X\beta)$$

g为*link function*, 是确定和已知的函数。

Logistic regression

采用logit function 为g，模型形式为

$$\begin{aligned}\mathbb{E}(y|X) &= \text{logit}^{-1}\left(X\beta\right) \\ &= \frac{1}{1+e^{-X\beta}}\end{aligned}$$

y_i服从Bernoilli分布，其未知参数只有$p_i = \mathbb{P}(y = 1)$，因此对于样本点i，似然函数为

$$L(y_i|p_i) = p_i^{y_i} \cdot (1 - p_i)^{1-y_i}$$

将其写成指数分布族的形式

$$\begin{aligned} L(y_i|p_i) &= (1 - p_i) \cdot \left(\frac{p_i}{1 - p_i} \right)^{y_i} \\ &= (1 - p_i) \cdot \exp \left(y_i \cdot \log \left(\frac{p_i}{1 - p_i} \right) \right) \end{aligned}$$

$$\begin{aligned}\eta_i &= \log\left(\frac{p_i}{1-p_i}\right) \\ &= \text{logit}(p_i)\end{aligned}$$

因此, 不妨令 η_i 是 x_i 的线性函数:

$$\eta_i = x_i^t \beta$$

这时，g是logit function.

于是有

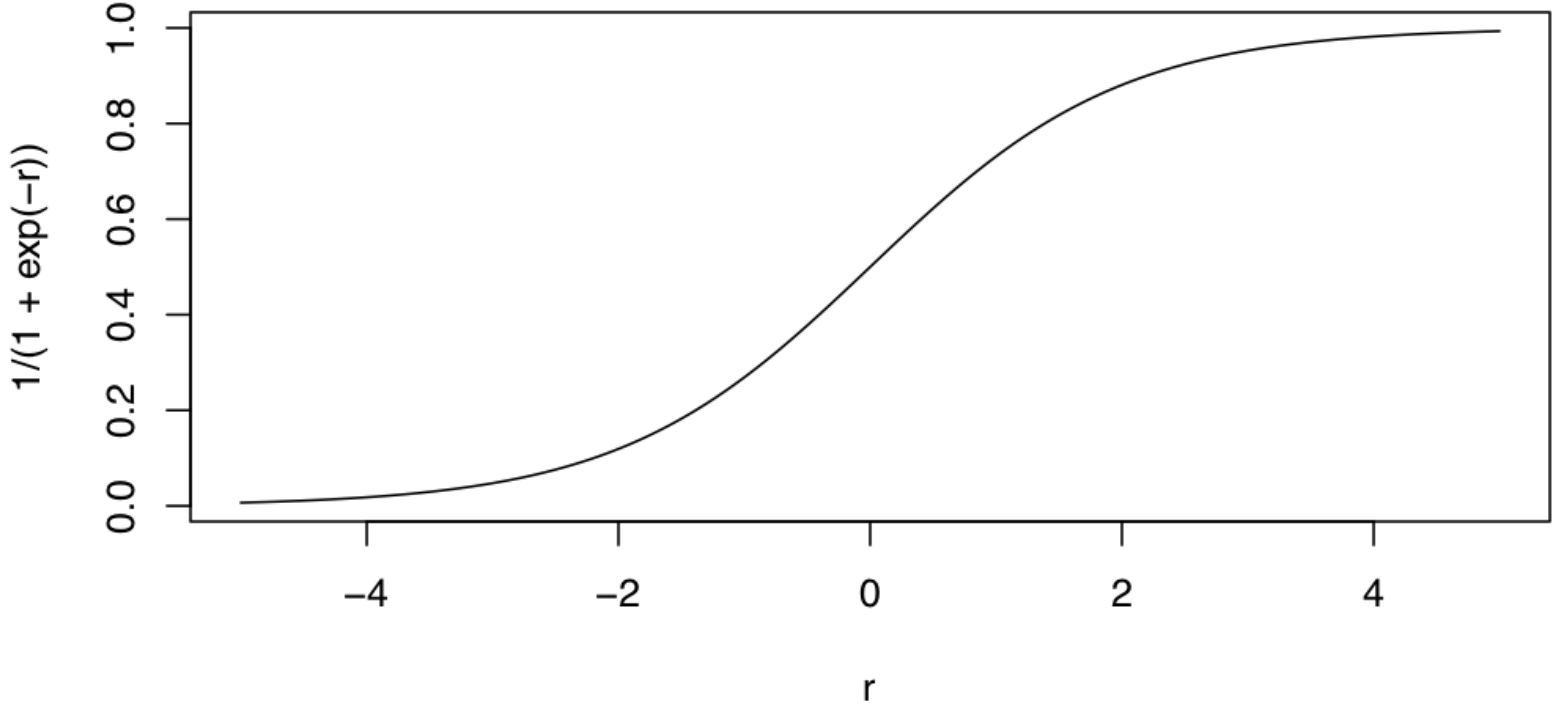
$$
\begin{aligned}
\log\left(\frac{p_i}{1-p_i}\right) &= x_i^t\beta \\
\frac{p_i}{1-p_i} &= e^{x_i^t\beta} \\
p_i &= (1-p_i)\cdot e^{x_i^t\beta} \\
\left(1+e^{x_i^t\beta}\right)p_i &= e^{x_i^t\beta} \\
p_i &= \frac{e^{x_i^t\beta}}{1+e^{x_i^t\beta}} \\
&= \frac{1}{1+e^{-x_i^t\beta}}
\end{aligned}
$$

GLM模型:$\mathbb{E}(y|X) = g^{-1}(X\beta)$

若y_i是独立伯努利实验，则模型为

$$\mathbb{E}(y|X) = \frac{1}{1 + e^{-X\beta}}$$

$x^t \beta$ 和 p_i 之间的关系?



如何通过加权最小二乘求解GLM

对于样本点i，似然函数中有两项与p_i有关：

$$L_i(y_i|p_i) = \exp\left\{ y_i \cdot \log\left(\frac{p_i}{1-p_i} \right) + \log(1-p_i) \right\}$$

- $\log\left(\frac{p_i}{1-p_i} \right)$: the projection of the regression vector $x_i^t \beta$.
- $\log(1-p_i)$:

$$\begin{aligned} p_i &= \frac{1}{1+e^{-x_i^t\beta}} \\ 1-p_i &= \frac{e^{-x_i^t\beta}}{1+e^{-x_i^t\beta}} \\ &= \frac{1}{1+e^{x_i^t\beta}} \\ \log(1-p_i) &= -1 \cdot \log(1+e^{x_i^t\beta}) \end{aligned}$$

对数似然函数为:

$$
\begin{aligned}
l(y_i|x_i,\beta) &= \sum_i \left(y_i \cdot x_i^t\beta - \log(1+e^{x_i^t\beta})\right) \\
&= y^t X\beta - \log(1+e^{X\beta})
\end{aligned}
$$

下面计算极大似然估计量，即令对数似然函数导数=0。

导数是

$$\begin{aligned}\nabla_\beta l(y_i|x_i,\beta) &= \nabla y^t X\beta - \nabla \log(1+e^{X\beta}) \\ &= X^t y - \nabla \log(1+e^{X\beta})\end{aligned}$$

第二项可以通过每一个β_k的梯度来计算。

对于第二项

根据链式法则

$$\frac{\partial}{\partial \beta_k} \log(1 + e^{X\beta}) = \frac{1}{1 + e^{X\beta}} \cdot x_k^t \cdot e^{X\beta}$$

将其简化成与p有关的形式:

$$\frac{1}{1 + e^{X\beta}} \cdot x_k^t \cdot e^{X\beta} = x_k^t p$$

写成向量形式:

$$\nabla \log(1 + e^{X\beta}) = X^t p$$

因此，对数似然梯度的显式形式：

$$\begin{aligned}\nabla_\beta l(y_i|x_i,\beta) &= X^t y - \nabla \log(1+e^{X\beta}) \\ &= X^t y - X^t p \\ &= X^t(y-p)\end{aligned}$$

- 不能简单地令其为零，p是X的非线性函数
- intuition:逻辑回归中的残差应该与X矩阵不相关（类似线性回归）。
- 要找到函数$\nabla_\beta l(y_i|x_i,\beta)$零值的数值解，可使用Newton-Raphson方法。

使用Newton-Raphson确定f的最优值的标量形式为

$$\beta^{(k+1)} = \beta^{(k)} - \frac{f'(\beta^{(k)})}{f''(\beta^{(k)})}$$

相应地，向量形式为

$$\beta^{(k+1)} = \beta^{(k)} - H^{-1}(\beta^{(k)})\nabla f(\beta^{(k)})$$

其中H是所有二阶偏导数的Hessian矩阵，f是原始的对数似然函数。

对数似然的Hessian是什么?

梯度为$X^t(y-p)$，因此Hessian仅取决于$X^t p$。

Hessian中每个元素：

$$\begin{aligned}\frac{\partial^2 l(\beta)}{\partial\beta_j\partial\beta_k} &= \frac{\partial}{\partial\beta_j}\sum_i x_{i,k}\cdot p_i \\ &= \sum_i x_{i,k}\cdot\frac{\partial p_i}{\partial\beta_j}\end{aligned}$$

p_i 相对于 β_j 的偏导数由下式给出：

$$\begin{aligned}
\frac{\partial p_i}{\partial \beta_j} &= \frac{\partial}{\partial \beta_j} \frac{1}{1+e^{-x_i^t \beta}} \\
&= \frac{-1}{(1+e^{-x_i^t \beta})^2} \cdot -1 \cdot x_{i,j} \cdot e^{-x_i^t \beta} \\
&= x_{i,j} \cdot \left(\frac{1}{1+e^{-x_i^t \beta}} \right) \cdot \left(\frac{e^{-x_i^t \beta}}{1+e^{-x_i^t \beta}} \right) \\
&= x_{i,j} \cdot p_i \cdot (1-p_i) \\
&= x_{i,j} \cdot \mathrm{Var}(y_i)
\end{aligned}$$

- （实际不需要最后一行）

于是，Hessian矩阵为

$$Hl(y|x_i, \beta) = -X^t DX$$

其中，D是$n \times n$对角矩阵，每个元素是$p_i \cdot (1 - p_i)$。

因此，Newton-Raphson法实际上是

$$\begin{aligned}\beta^{(k+1)} &= \beta^{(k)} - \boldsymbol{H}^{-1}(\beta^{(k)})\nabla f\,(\beta^{(k)}) \\ &= \beta^{(k)} + (X^t \boldsymbol{D} X)^{-1} X^t (y - p^{(k)})\end{aligned}$$

采用合适的z将上式写为

$$\begin{aligned}\beta^{(k+1)} &= (X^t DX)^{-1}(X^t DX)\beta^{(k)} + (X^t DX)^{-1}X^t(y - p^{(k)}) \\ &= (X^t DX)^{-1}X^t D(X\beta^{(k)} + D^{-1}(y - p)) \\ &= (X^t DX)^{-1}X^t Dz\end{aligned}$$

- 因此，解决逻辑回归等于解决一系列加权最小二乘模型

$$\beta^{(k+1)} = \underset{b}{\operatorname{argmin}} \left\{ ||D^{1/2}(z - Xb)||_2^2 \right\}$$

- 若预测概率接近0或1，则会对观测进行加更大的权重。 这有意义吗?
- 将最高的权重放在具有最小方差的观测上。

In [1]:

```
n <- 1000
p <- 10
X <- matrix(rnorm(n*p),ncol=p)
beta <- runif(p)
y <- as.numeric(X%*%beta + rnorm(n) > 0)

# Fit a model; what do you think the iter is?
out <- glm(y ~ X - 1, family=binomial())
out$iter
```

6

In [2]:

```
# Show that the residuals are orthogonal to col space of X
t(X) %*% (y - predict(out, type="response"))
```

A matrix: 10 × 1
of type dbl

6.517880e-10
4.703924e-10
4.595624e-10
6.891673e-11
3.829013e-10
3.548814e-10
3.784546e-10
1.510716e-10
2.657839e-12
3.663790e-11

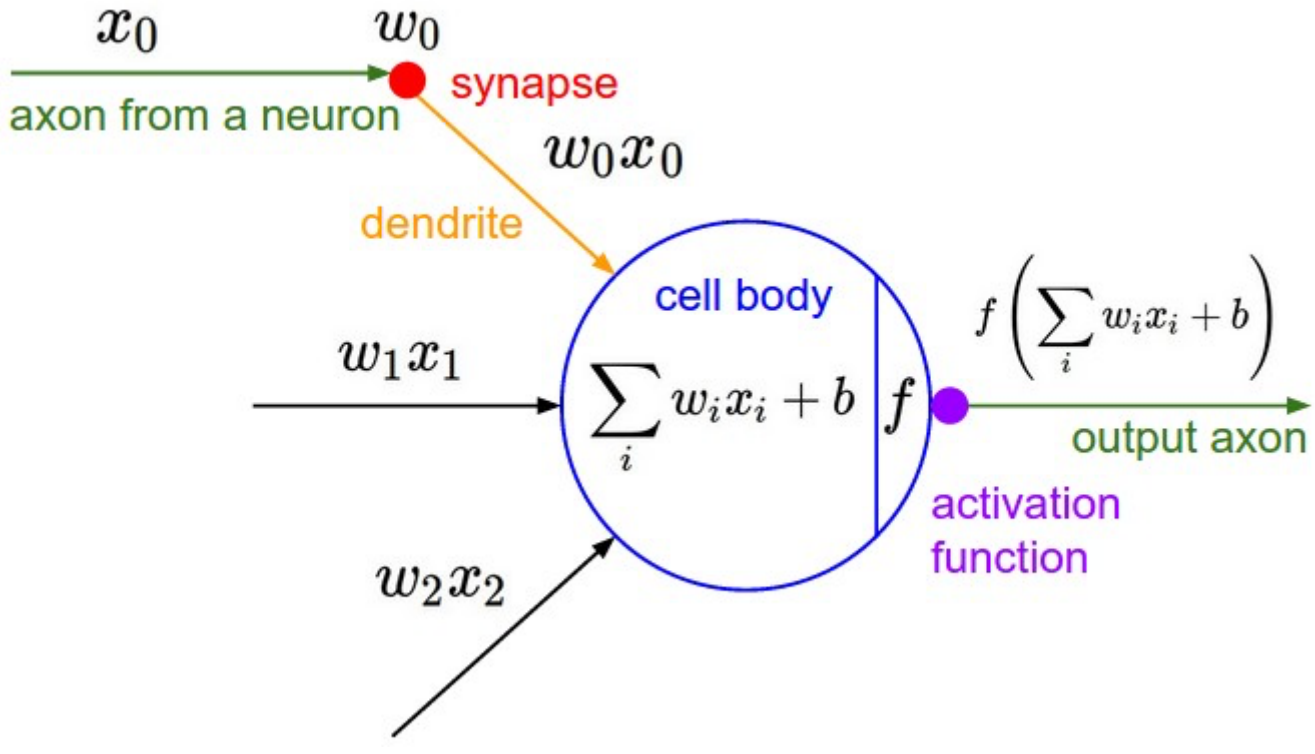
In [3]:

```r
# Fit GLM IRWLS
beta <- rep(0,p)
for (k in 1:10) {
  prob <- 1 / (1 + exp(-X %*% beta))
  D <- diag(as.numeric(prob*(1-prob)))
  z <- X %*% beta + (y - prob) / diag(D)
  beta <- qr.solve(crossprod(X, D %*% X), crossprod(X, D %*% z))
  print(sum((beta - coef(out))^2))
}
```

```
[1] 3.381727
[1] 0.802604
[1] 0.059604
[1] 0.0003847109
[1] 1.687253e-08
[1] 3.209456e-17
[1] 2.503075e-21
[1] 2.502413e-21
[1] 2.502499e-21
[1] 2.502213e-21
```

Logistic regression & Neural networks

Logistic regression：仅含有一个神经元的单层的神经网络



Activation functions

- Logistic regression

$$\text{logit}(p_i) = \log\left(\frac{p_i}{1-p_i}\right)$$

$$\text{logit}^{-1}\,(X\beta) = \frac{1}{1+e^{-X\beta}}$$

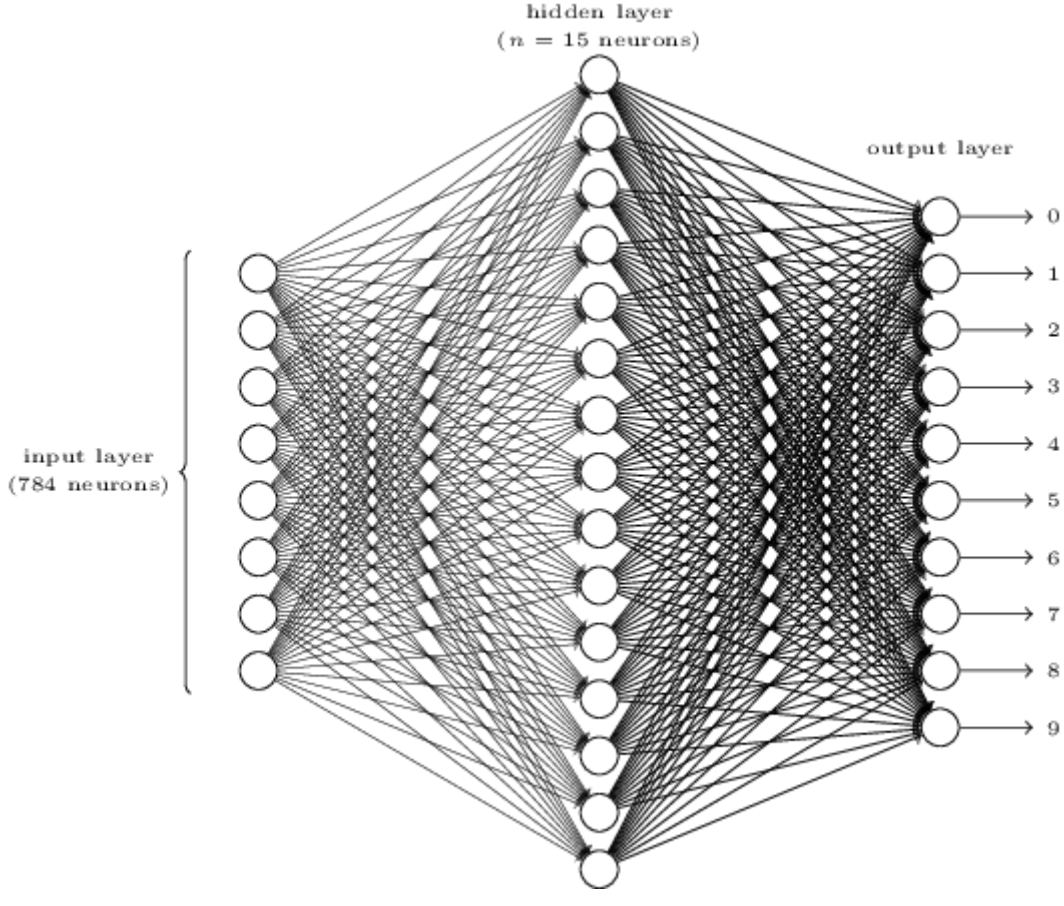
- Neural networks

$$\sigma(x \cdot w + b) = \frac{1}{1+e^{-(x\cdot w+b)}}$$

The inverse of the logit function is the sigmoid function .

Feed-forward network (前馈网络)

- 节点连接只有一个方向，朝一个方向传送信息
- 每个隐藏层内部，不允许相互连接（RNN允许）



查看哪个输出神经元具有最大输出，就分为哪一类。

如何求解weights和bias?

如果是单层、单个神经元，则可以用logistic regression 或SVM等。

cost function (loss function)

squared error loss:

$$C(w, b) = \frac{1}{2n} \sum_i (y_i - \hat{y}(x_i))^2$$

for classification, the hinge loss:

$$C(w, b) = \sum_i \left[1 - y_i \cdot \hat{y}(x_i)\right]_+$$

Optimization problem & gradient descent

$$w_{k+1} = w_k - \eta \cdot \nabla_w C$$
$$b_{k+1} = b_k - \eta \cdot \nabla_b C$$

- learning rate, $\eta > 0$

Decomposable cost function

神经网络中的成本函数能够在样本上分解：

$$C = \frac{1}{n} \sum_i C_i$$

For the individual costs C_i of the i'th sample.

Decomposable cost function

Consider now taking a subset $M \subseteq \{1, 2, \ldots n\}$ with size m of the training set. It would seem that we can approximate the cost function using only this subsample of the data:

$$\frac{\sum_{i \in M} \nabla C_i}{m} \approx \frac{\sum_{i=1}^{n} \nabla C_i}{n} \approx \nabla C$$

Stochastic gradient descent (SGD)

- Stochastic gradient descent uses this idea to speed up the process of doing gradient descent.
- the input data are randomly partitioned into disjoint groups $M_1, M_2, \ldots, M_{n/m}$.
- update:

$$
\begin{aligned}
w_{k+1} &= w_k - \frac{\eta}{m} \sum_{i \in M_1} \nabla C_i \\
w_{k+2} &= w_{k+1} - \frac{\eta}{m} \sum_{i \in M_2} \nabla C_i \\
&\vdots \\
w_{k+n/m+1} &= w_{k+n/m} - \frac{\eta}{m} \sum_{i \in M_{n/m}} \nabla C_i
\end{aligned}
$$

Each set M_j is called a mini-batch and going through the entire dataset as above is called an epoch.

Stochastic gradient descent (SGD)

main tuning parameters:

- size of themini-batch} (m)
- the learning rate (η)
- number of epochs (E)

为了便于理解，下面以OLS为例，来看SGD是怎么做的。

SGD Examples: OLS

$$
\begin{aligned}
f(\beta) &= \frac{1}{n} \cdot \sum_i f_i(\beta) \\
&= \frac{1}{n} \cdot \sum_i (y_i - x_i \beta)^2 \\
&= \frac{1}{n} \cdot \sum_i (y_i^2 + \beta^t x_i^t x_i \beta - 2 y_i x_i \beta)
\end{aligned}
$$

SGD Examples: OLS

The gradient of f:

$$\begin{aligned}\nabla f &= \frac{1}{n} \cdot \sum_i f_i(\beta) \\ &= \frac{1}{n} \cdot \sum_i \nabla f_i(\beta) \\ &= \frac{2}{n} \cdot \sum_i (x_i^t x_i \beta - x_i^t y_i)\end{aligned}$$

使用大小为m的mini-batch M来近似

$$\nabla f \approx \frac{2}{m} \cdot \sum_{i \in M} (x_i^t x_i \beta - x_i^t y_i)$$

In [3]:

```
set.seed(42)
n <- 1000
p <- 10
X <- matrix(rnorm(n*p),ncol=p)
beta <- 1 / 1:p
y <- X %*% beta + rnorm(n, sd=0.5)

betaOls <- coef(lm(y ~ X - 1))
round(betaOls,2)
```

X1: 0.99 **X2:** 0.49 **X3:** 0.34 **X4:** 0.26 **X5:** 0.21 **X6:** 0.16
X7: 0.14 **X8:** 0.1 **X9:** 0.1 **X10:** 0.09

In [4]:

```
m <- 5
miniBatchIndex <- matrix(sample(1:n),ncol=m)
head(miniBatchIndex)
```

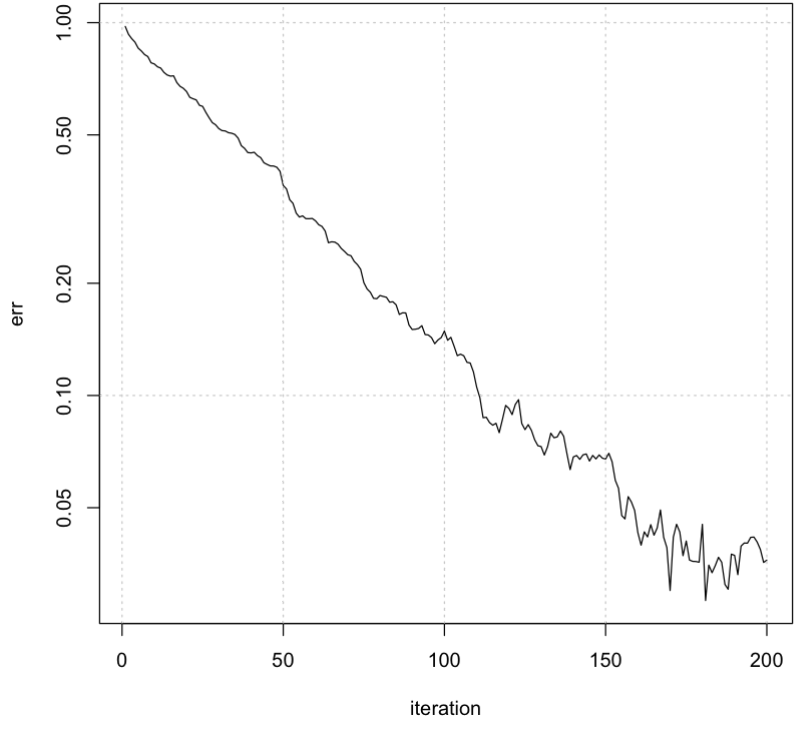
A matrix: 6 × 5 of type int

603	123	460	104	935
400	515	693	42	381
447	717	774	720	949
389	474	463	740	155
423	610	656	623	966
646	211	569	911	768

In [5]:

```
eta <- 0.01
b <- rep(0, p)
err <- rep(NA, nrow(miniBatchIndex))
for (j in 1:nrow(miniBatchIndex)) {
  ind <- miniBatchIndex[j,]
  gradMini <- (2/m) * ((t(X[ind,]) %*% X[ind,]) %*% b - t(X[ind,]) %*% y[ind])
  b <- b - eta * gradMini
  err[j] <- max(abs(b - betaOls))
}

plot(err, log="y", type="l", xlab="iteration")
grid()
```



In [6]:

```
eta <- 0.01
b <- rep(0, p)
E <- 10
err <- rep(NA, nrow(miniBatchIndex)*E)
for (k in 1:E) {
  miniBatchIndex <- matrix(sample(1:n),ncol=m)
  for (j in 1:nrow(miniBatchIndex)) {
    ind <- miniBatchIndex[j,]
    gradMini <- (1/m) * (2 * (t(X[ind,]) %*% X[ind,]) %*% b - 2 * t(X[ind,]) %*% y
[ind])
    b <- b - eta * gradMini
    err[j+(k-1)*nrow(miniBatchIndex)] <- max(abs(b - betaOls))
  }
}

plot(err, log="y", type="l", axes=FALSE, xlab="epoch")
box()
axis(2)
axis(1, at=(0:E)*nrow(miniBatchIndex), 0:E)
abline(v=(1:E)*nrow(miniBatchIndex))
```

Stochastic gradient descent (SGD)

- Stochastic gradient descent uses this idea to speed up the process of doing gradient descent.
- the input data are randomly partitioned into disjoint groups $M_1, M_2, \ldots, M_{n/m}$.
- update：

$$
\begin{aligned}
w_{k+1} &= w_k - \frac{\eta}{m} \sum_{i \in M_1} \nabla C_i \\
w_{k+2} &= w_{k+1} - \frac{\eta}{m} \sum_{i \in M_2} \nabla C_i \\
&\vdots \\
w_{k+n/m+1} &= w_{k+n/m} - \frac{\eta}{m} \sum_{i \in M_{n/m}} \nabla C_i
\end{aligned}
$$

- 如何计算∇C_i？

back-propagation

一些数学符号

- $w^l_{j,k}$： the weight of node k in layer $l-1$ as applied by node j in layer l.
- b^l_j： the bias of node j of layer l
- z^l_j： The weighted input emitted from node j in layer l；类似logsitic regression 中的$X\beta$
- a^l_j： $a^l_j = \sigma(z^l_j)$

这些符号之间的关系

$$\begin{aligned} a_j^l &= \sigma(z_j^l) \\ &= \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l) \end{aligned}$$

（最后一个数学符号）error functions:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

假设神经网络共有L层，那么cost function为：

$$C(w,b) = \sum_k (y_k - a_k^L)^2$$

k是输出的维度标号，y可以是多元的

Feed-forward step

对于最靠近输出的一层:

$$
\begin{aligned}
\delta_j^L &= \frac{\partial C}{\partial z_j^L} \\
&= \frac{\partial C}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L} \\
&= \frac{\partial C}{\partial a_j^L} \cdot \sigma'(z_j^L)
\end{aligned}
$$

Feed-forward step

第一项:

$$\begin{aligned}\frac{\partial C}{\partial a_j^L} &= \frac{\partial}{\partial a_j^L} \sum_k (y_k - a_k^L)^2 \\ &= 2 \cdot (a_j^L - y_j)\end{aligned}$$

第二项:

$$\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z))$$

Back-propagation step

将δ_j^l和δ_j^{l+1}联系起来：

$$\begin{aligned}\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\ &= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \sum_k \delta_k^{l+1} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l}\end{aligned}$$

对于导数这一项，有

$$\begin{aligned}z_k^{l+1} &= \sum_i w_{ki}^{l+1} a_i^l + b_k^{l+1} \\ &= \sum_i w_{ki}^{l+1} \sigma(z_i^l) + b_k^{l+1}\end{aligned}$$

Back-propagation step

于是，有

$$\begin{aligned}\frac{\partial z_k^{l+1}}{\partial z_j^l} &= \frac{\partial}{\partial z_j^l}\left(\sum_i w_{ki}^{l+1}\sigma(z_j^l) + b_k^{l+1}\right) \\ &= w_{kj}^{l+1} \cdot \sigma'(z_j^l)\end{aligned}$$

带回到最初的式子中，有

$$\delta_j^l = \sum_k \delta_k^{l+1} \cdot w_{kj}^{l+1} \cdot \sigma'(z_j^l)$$

在每个节点上，将errors和bias联系起来

$$
\begin{aligned}
\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\
&= \frac{\partial C}{\partial b_j^l} \cdot \frac{\partial b_j^l}{\partial z_j^l} \\
&= \frac{\partial C}{\partial b_j^l}
\end{aligned}
$$

在每个节点上，将errors和weights联系起来

$$\begin{aligned}\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\ &= \frac{\partial C}{\partial w_{jk}^l} \cdot \frac{\partial w_{jk}^l}{\partial z_j^l} \\ &= \frac{\partial C}{\partial w_{jk}^l} \cdot \frac{1}{a_k^{l-1}}\end{aligned}$$

也即

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$$

小结

$$
\begin{aligned}
\delta_j^L &= \nabla_{a^L} C \circ \sigma'(z^L) \\
\delta_j^l &= ((w^{l+1})^T \delta^{l+1}) \circ \sigma'(z^l) \\
\frac{\partial C}{\partial b_j^l} &= \delta_j^l \\
\frac{\partial C}{\partial w_{jk}^l} &= a_k^{l-1} \delta_j^l
\end{aligned}
$$

符号∘是Hadamard product

The back-propagation algorithm

- Select an element i from the current minibatch and calculate the weighted inputs z and activations a for every layer using a forward pass through the network
- Now, use these values to calculate the errors δ for each layer, starting at the last hidden layer and working backwards, using back-propagation
- Calculate $\nabla_w C_i$ and $\nabla_b C_i$ using the last two equations
- Repeat for all samples in the minibatch, to get:

$$\frac{\sum_{i \in M} \nabla_w C_i}{|M|} \approx \nabla_w C \qquad \frac{\sum_{i \in M} \nabla_b C_i}{|M|} \approx \nabla_b C$$

- Update the weights and biases by (the estimates of) $-\eta \cdot \nabla_w C$ and $-\eta \cdot \nabla_b C$, respectively
- Repeat over every mini-batch, and for the desired number of epochs

In [25]:
```
train <- read.csv("mnist_train.psv", as.is=TRUE, header=FALSE, sep = "|", stringsA
sFactors = FALSE)
test <- read.csv("mnist_test.psv", as.is=TRUE, header=FALSE, sep = "|", stringsAsF
actors = FALSE)
dim(train)
dim(test)
```

7291 · 257

2007 · 257

In [26]:

```
X <- as.matrix(train[,-1]) / 256
Xtest <- as.matrix(test[,-1]) / 256

# convert the categorical response into 10 indicator functions:
Y <- matrix(0, nrow=nrow(X), ncol=10)
Ytest <- matrix(0, nrow=nrow(Xtest), ncol=10)
for (i in 0:9) {
  Y[train[,1] == i,i+1] <- 1
  Ytest[test[,1] == i,i+1] <- 1
}
head(Y)
```

A matrix: 6 × 10 of type dbl

0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0

In [39]:

```
nn <- nnetObj(sizes <- c(256,9,10))
nn2 <- sgd(nn, X, Y, Xtest, Ytest, m=1, epochs=15, eta=0.2, FALSE)

par(mfrow=c(3,3))
par(mar=c(0,0,0,0))
for (j in 1:9) {
  z <- nn2$w[[1]][j,]
  z <- (z - min(z)) / (max(z) - min(z))
  z <- matrix(as.matrix(z),16,16,byrow=TRUE)
  plot(0,0,axes=FALSE,xlab="",ylab="",main="")
  rasterImage((1 - z),-1,-1,1,1)
  box()
}
```

Thank you!

shan.lu@cufe.edu.cn