hw4_514

Homework 4 Problem Statement

Implement gradient descent for softmax regression using the cross-entropy (CE) cost function. Train models using synthetic data as well as the MNIST handwritten digit data.

You will complete the following functions as part of this homework and some additional helper functions:

- one.hot : Takes a Y vector of integers and converts it to one hot encoded matrix. Our implementation stores one-hot vectors as columns
- bk.prop: Computes the gradient of the cost function wrt to adjustable parameters b and w. Your solution should be "vectorized" and use vector/matrix operations to compute the gradient for the entire dataset on each call.
- fwd.prop: Computes model outputs (softmax). fwd.prop should be vectorized
- softmax.fit: Implements a simple gradient descent loop given a fixed learning rate and the number of iterations
- cost: Compute Cross-Entropy cost for entire dataset.

Step 1 Generate data

Use the code chunk below to to create a dataset. This chunk creates a data set that is a mixture of 3 classes of 2d normals with different means and variances. Use the plot provided to visualize the data.

```
source('hw4_514.R')
```

```
library(MASS)
library(zeallot)
require(Matrix)
```

```
## Loading required package: Matrix
```

```
require("beepr")
```

```
## Loading required package: beepr
```

```
n1=50; mu1=c(.5,.5);  cov1=diag(.2,2)
n2=40; mu2=c(1.5,1.5);cov2=diag(.1,2)
n3=30; mu3=c(1.5,0);  cov3=diag(.1,2)
mean.cov.list=list()
mean.cov.list[[1]]=list(n=50, mu=c(.5,.5),  cov=diag(.2,2))
mean.cov.list[[2]]=list(n=40, mu=c(1.5,1.5),cov=diag(.1,2))
mean.cov.list[[3]]=list(n=30, mu=c(1.5,0),  cov=diag(.1,2))

#c(X,y) %<-% generate.gaussian.data.class3(n1,mu1,cov1,n2,mu2,cov2,n3,mu3,cov3)
c(x,y) %<-% gen.gaussian.data.2d(mean.cov.list)
```

```
## [1] 0.5 0.5
## [1] 1.5 1.5
## [1] 1.5 0.0
```

```
plot(x[1,],x[2,],pch=1,col=y+1,lwd=2,cex=1)
```

Step 2 Implement one hot encoding function

Implement a function called one.hot that takes a vector of integers and returns the corresponding one-hot encode matrix

```
one.hot <- function(Y){
    return(unname( as.matrix( as.data.frame( t( model.matrix(~ as.factor(y) + 0) ) ) ) )) }
```

Step 3 Cost Function and Model Output Function

- Use latex to document your cross-entropy cost function

$$C^{(i)} = -\sum_{k=1}^{K} t_k^i \log\left(p_k^{(i)}\right)$$

$$C = \frac{1}{m}\sum_{i=1}^{m} C^{(i)} = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} t_k^i \log\left(p_k^{(i)}\right)$$

- Implement (complete function) the cost function code. The cost function takes as input the entire X and Y datasets and b,w parameters and should be vectorized. Make sure to divide your total cost by the number of samples.

```
cost <- function(X,Y,b,w){
    H <- fwd.prop(X,b,w)
    (-1/dim(X)[2]) * sum(  colSums( one.hot(Y) * log( H)) )}
```

- Use latex to document the model output probabilites (softmax)

$$S(\mathbf{x}) = \frac{1}{\sum_{i=1}^{k} \exp(x_i)} \begin{pmatrix} \exp(x_1) \\ \exp(x_2) \\ \vdots \\ \exp(x_k) \end{pmatrix} = \begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_k \end{pmatrix}$$

- Implement the model output function, called fwd.prop. fwd.prop takes the full X dataset, along with model parameters and computes output probabilities for every sample. Your code should be vectorized.

```
fwd.prop <- function( X, b ,w ){
    Z <- ( b %*% matrix( rep(1, dim(X)[2] ) , nrow = 1) + w %*% X)
    Z <-  exp( sweep(Z,2,apply(Z,2,max) ))
    H <- Z %*% (1/colSums( Z ) * Diagonal(dim(Z)[2])     )
    return( as.matrix(H))  }
```

```
print.fun('fwd.prop');
```

```
## [1] "fwd.prop (X, b, w) "
## {
##     Z <- (b %*% matrix(rep(1, dim(X)[2]), nrow = 1) + w %*% X)
##     Z <- exp(sweep(Z, 2, apply(Z, 2, max)))
##     H <- Z %*% (1/colSums(Z) * Diagonal(dim(Z)[2]))
##     return(as.matrix(H))
## }
```

```
print.fun('cost');
```

```
## [1] "cost (X, Y, b, w) "
## {
##     H <- fwd.prop(X, b, w)
##     (-1/dim(X)[2]) * sum(colSums(one.hot(Y) * log(H)))
## }
```

Step 4 Gradient Calculations with Numerical Check

Use latex to document the cross-entropy cost calculation. Implement (complete function) the code to return the gradients in a function called bk.prop. Your gradient code should be vectorized. Make sure to divide your total gradient by the number of samples.

$$\begin{aligned}
\sum_{i=1}^{m} \frac{\partial C^{(i)}}{\partial \mathbf{w}} &= \sum_{i=1}^{m} \frac{\partial C^{(i)}}{\partial \mathbf{z^{(i)}}} \left(\mathbf{x}^{(i)}\right)^T \\
&= \frac{\partial C}{\partial Z} X^T \\
&= \Big(S(Z) - T\Big) X^T
\end{aligned}$$

$$\begin{aligned}
\langle \frac{\partial C}{\partial b} \rangle &= \frac{1}{m}\Big(S(Z) - T\Big)\begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \\
\langle \frac{\partial C}{\partial W} \rangle &= \frac{1}{m}\Big(S(Z) - T\Big)\ X^T
\end{aligned}$$

Check your gradient functions by comparing with a numerical gradient calculation.

```r
print.fun('num.gradient');
```

```
## [1] "num.gradient (cost, X, Y, b, w, g = 1e-08) "
## {
##     db <- b
##     for (i in 1:length(b)) {
##         bP <- bM <- b
##         bP[i] <- bP[i] + g
##         bM[i] <- bM[i] - g
##         db[i] <- (cost(X, Y, bP, w) - cost(X, Y, bM, w))/(2 *
##             g)
##     }
##     dw <- w * 0
##     for (i in 1:nrow(w)) {
##         for (j in 1:ncol(w)) {
##             wP <- wM <- w
##             wP[i, j] <- wP[i, j] + g
##             wM[i, j] <- wM[i, j] - g
##             dw[i, j] <- (cost(X, Y, b, wP) - cost(X, Y, b, wM))/(2 *
##                 g)
##         }
##     }
##     return(list(db = db, dw = dw))
## }
```

```
print.fun('bk.prop');
```

```
## [1] "bk.prop (X, Y, fprop) "
## {
##     M <- dim(X)[2]
##     E <- fprop - one.hot(Y)
##     db <- (1/M) * E %*% rep(1, M)
##     dw <- (1/M) * E %*% t(X)
##     return(list(db = db, dw = dw))
## }
```

```
dim(x)
```

```
## [1]   2 120
```

```
b0 <- rnorm(3)
w0 <- matrix(rnorm(3*dim(x)[1]),3,dim(x)[1])

bk.prop(x,y ,fwd.prop(x,b0,w0) )
```

```
## $db
##            [,1]
## [1,] -0.3908722
## [2,] -0.3116055
## [3,]  0.7024777
##
## $dw
##            [,1]       [,2]
## [1,] -0.1779078 -0.1986268
## [2,] -0.4937894 -0.4703083
## [3,]  0.6716972  0.6689351
```

```
num.gradient(cost,x,y,b0,w0)
```

```
## $db
## [1] -0.3908722 -0.3116055  0.7024777
##
## $dw
##            [,1]       [,2]
## [1,] -0.1779078 -0.1986268
## [2,] -0.4937894 -0.4703083
## [3,]  0.6716972  0.6689351
```

Step 4 Optimizer/Gradient Descent

Next code the softmax.fit function. Arguments include a learning rate parameter and the number of iterations. This function should return a complete list of generated data: - history of (b,w) - history of cost - history of gradient norm - history of timestamps (Sys.time()) - Use print fun to display code in softmax.fit

```
## [1] "softmax.fit (X, Y, lr = 0.01, max.its = 100) "
## {
##     K <- length(unique(as.vector(Y)))
##     N <- dim(X)[1]
##     M <- dim(X)[2]
##     b <- rnorm(K) * 0.1
##     W <- matrix(rnorm(K * N) * 0.1, K, N)
##     blist <- wlist <- list()
##     Costs <- gradlist <- peformance <- rep(NA, max.its)
##     Class <- sort(unique(as.vector(Y)))
##     st <- system.time(for (i in 1:max.its) {
```

```
##          FP <- fwd.prop(X, b, W)
##          peformance[i] <- sum((Class[apply(FP, 2, function(M) which(M ==
##              max(M)))] == Y) * 1)/length(Y)
##          Costs[i] <- (-1/M) * sum(colSums(one.hot(Y) * log(FP)))
##          BP <- bk.prop(X, Y, FP)
##          b <- blist[[i]] <- b - lr * BP$db
##          W <- wlist[[i]] <- W - lr * BP$dw
##          gradlist[i] <- norm(cbind(b, W), "2")
##      })
##      return(list(b = b, W = W, st = st, wlist = wlist, peformance = peformance,
##          blist = blist, costs = Costs, gradlist = gradlist))
## }
```

Implement a function called predict that will take as input b,w, and X and return predictions. It should return the class value with the highest predicted probability as the prediction.

```
Predict <- function(X,Y,b,w){
    Classes  <-  sort(unique(as.vector(Y)))
    Classes[apply( fwd.prop( X , b , w ), 2 , function( M )
        which( M == max(M) ) ) ]    }
```

Step 5 Use softmax regression on synthetic data

- Use the softmax.fit to train a model on synthetic data from step 1. Use a learning rate of .2 and 2000 iterations.
- Use the predict function to test model accuracy on out-of-sample data
- Plot the decision boundary of the model. Do this by creating a grid of points covering the synthetic and coloring the grid points using the model predicted class. Overlay the data points, colored by their predict values, on top of the grid.

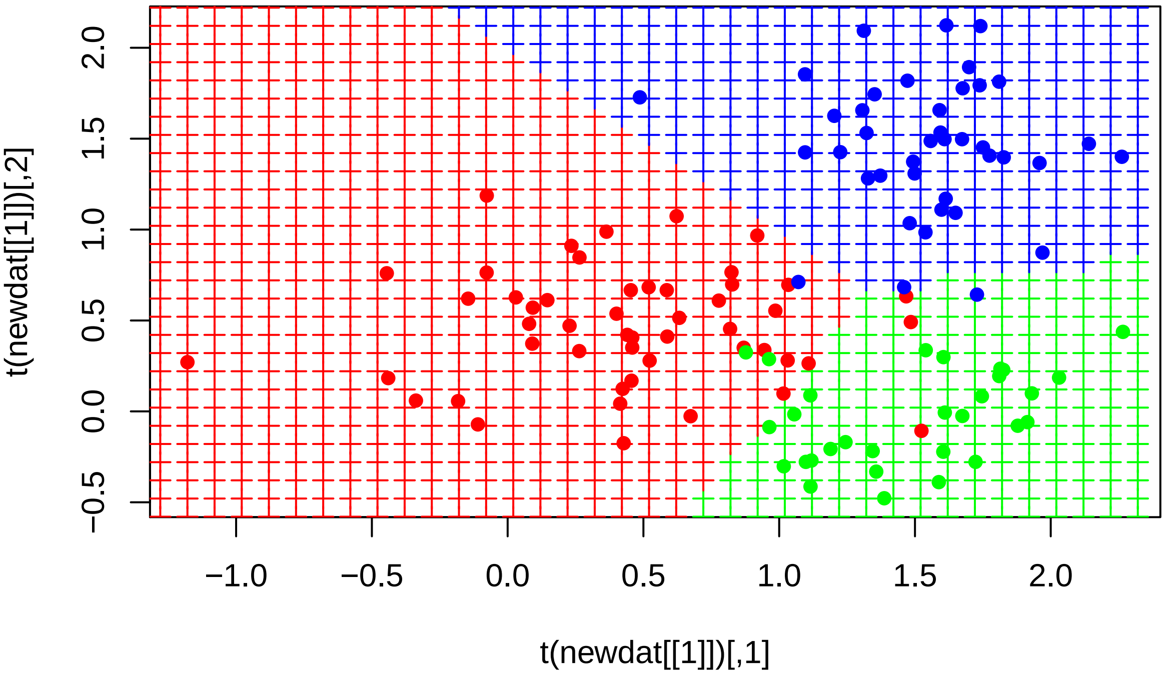
```
fit <- softmax.fit(x,y,lr=.2,max.its = 2000)

newdat <- gen.gaussian.data.2d(mean.cov.list)
```

```
## [1] 0.5 0.5
## [1] 1.5 1.5
## [1] 1.5 0.0
```

```
sum(( Predict(newdat[[1]], newdat[[2]], fit$b, fit$W) == newdat[[2]] )*1 ) / length(newdat[[2]])
```

```
## [1] 0.9166667
```

```
px <-  seq(  min( newdat[[1]] )-.1 , max( newdat[[1]] )+.1   , .1 )
Colors <- c("red", "blue", "green")

plot(t(newdat[[1]]), pch = 16 , col = "white")
for( i in 1:length(px) ){
for( j in 1:length(px) ){
    points(  px[i], px[j]  , pch = 3, col = Colors[
    Predict(  ( as.matrix( c( px[i], px[j] )  ) ), y , fit$b , fit$W)] )
}}

plotdat <- cbind( t(newdat[[1]]),   as.vector( newdat[[2]] ) )

apply( plotdat, 1 , function(A) points( A[1], A[2], col = Colors[A[3]], pch = 16 ) )
```



```
## NULL
```

Step 6 change synthetic data to consider a 5 class problem

- Re-do step 5 using a synthetic data set with 5 classes

```
n1=50; mu1=c(.5,.5);  cov1=diag(.2,2)
n2=40; mu2=c(1.5,2.); cov2=diag(.1,2)
n3=30; mu3=c(1.5,-1); cov3=diag(.1,2)
n4=30; mu4=c(2,0);    cov4=diag(.1,2)
n5=30; mu5=c(2,1);    cov5=diag(.1,2)

mean.cov.list=list()
mean.cov.list[[1]]=list(n=n1, mu=mu1,  cov=cov1)
mean.cov.list[[2]]=list(n=n2, mu=mu2,  cov=cov2)
mean.cov.list[[3]]=list(n=n3, mu=mu3,  cov=cov3)
mean.cov.list[[4]]=list(n=n4, mu=mu4,  cov=cov4)
mean.cov.list[[5]]=list(n=n5, mu=mu5,  cov=cov5)

#c(X,y) %<-% generate.gaussian.data.class3(n1,mu1,cov1,n2,mu2,cov2,n3,mu3,cov3)
c(x,y) %<-% gen.gaussian.data.2d(mean.cov.list)
```

```
## [1] 0.5 0.5
## [1] 1.5 2.0
## [1]  1.5 -1.0
## [1] 2 0
## [1] 2 1
```

```
#Y=one.hot(t(y))

fit <- softmax.fit(x,y,lr=.2,max.its = 2000)
```

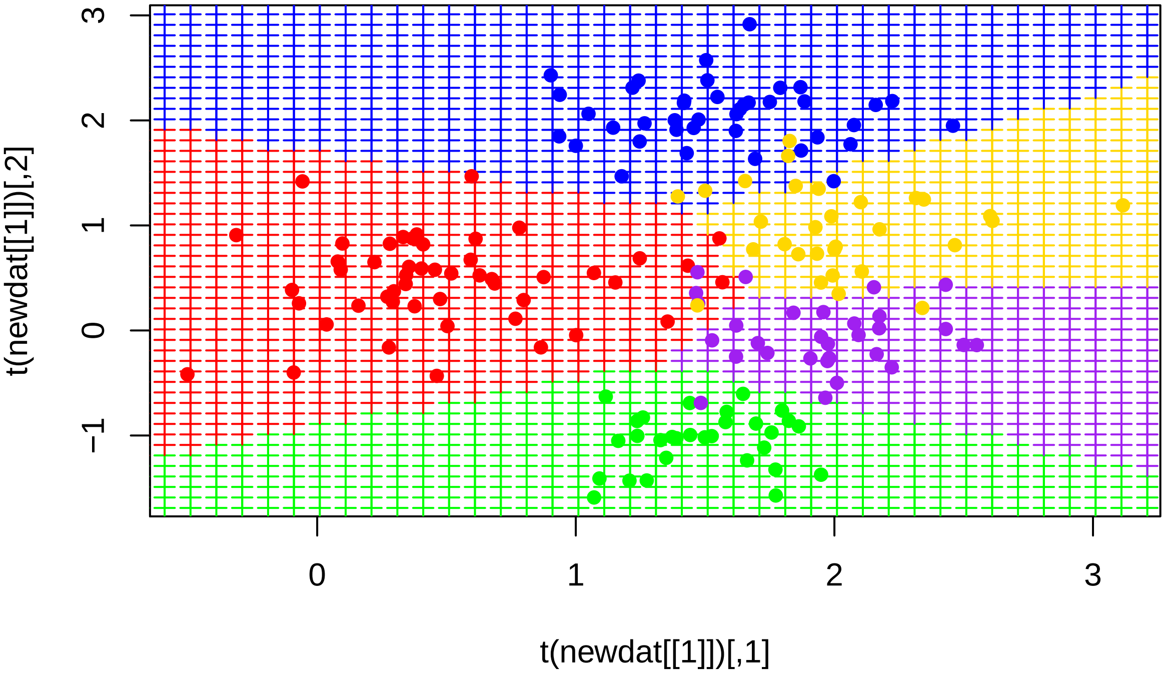
```
newdat <- gen.gaussian.data.2d(mean.cov.list)
```

```
## [1] 0.5 0.5
## [1] 1.5 2.0
## [1]  1.5 -1.0
## [1] 2 0
## [1] 2 1
```

```
sum(( Predict(newdat[[1]], newdat[[2]], fit$b, fit$W) == newdat[[2]] )*1 ) / length(newdat[[2]])
```

```
## [1] 0.9
```

```
px <-  seq(  min( newdat[[1]] )-.1 , max( newdat[[1]] )+.1  , .1 )
Colors <- c("red", "blue", "green", "purple", "gold")

plot(t(newdat[[1]]), pch = 16 , col = "white")
for( i in 1:length(px) ){
for( j in 1:length(px) ){
    points(  px[i], px[j] , pch = 3, col = Colors[
    Predict(  ( as.matrix( c( px[i], px[j] )  ) ), y , fit$b , fit$W)] )
}}

plotdat <- cbind( t(newdat[[1]]),   as.vector( newdat[[2]] ) )

apply( plotdat, 1 , function(A) points( A[1], A[2], col = Colors[A[3]], pch = 16 ) )
```



```
## NULL
```

Step 7 Download and load MNIST data

Using the following URLs download the MNIST data (manually, or you could use the download code included in the hw4_514.R file). The dataset contains input X data containing image of numbers between 0 and 9 and actual y label of the number. Image pixels range between 0 and 255 and have to be scaled by dividing by matrix by 255 before you use the data.

- http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
- http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
- http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
- http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz

LeCun's MNIST data has 60k train samples and 10k test samples

Step 8 Train a model to predict MNIST data

Use your softmax.fit to train a model on MNIST training data. Use the load_image_file and load_label_file provided in the hw4_514.R file to read the binary ubyte files

- build a model using softmax.fit using a random sample of 1000 train images using a learning rate of .1 and 100 iterations
- test model accuracy both in sample and out of sample. Use the provided test data for out-of-sample images and tags
- plot the cost history of the solution
- plot weight history
- plot gradient history
- plot performance history of the model

```r
weighthist <- function(BList, WList){
  WH <- matrix( NA , length(BList)  , 2 )
  for( i in 1:length(BList)){
    WH[i,1] <-  norm(BList[[i]],"2")
    WH[i,2] <-  norm(WList[[i]],"2") }
  return(WH) }

gradhist <- function(BList, WList){
  GH <- rep(NA, length(BList))
  for( i in 1:length(BList)){
  GH[i] <- ( norm( cbind( BList[[i]],WList[[i]] ) , "2")) }
  return(GH)
 }
```

```r
mtrain <- read.csv("mnist_train.csv" , header = F)
#mtrain <- read.csv("C:\\Users\\Administrator\\Downloads\\mnist_train.csv", header = F)
test <- read.csv("mnist_test.csv" , header = F)
colnames(test )[1] <- colnames(mtrain )[1] <- "Y"

set.seed(1)
mtrains <- mtrain[ sample( 1:nrow(mtrain) , 1000 ) , ]
mtrains <- as.matrix(mtrains)

nx <- unname( test[,-1] )
nx <- t(nx)
nx <- nx / 255
ny <-  unname( test[,1] )

x <- unname( mtrains[,-1] )
x <- t(x)
x <- x / 255
y <-  unname( mtrains[,1] )

fits <- fit <- softmax.fit( x , y, lr = .1 ,max.its= 100 );#beep("coin")
```

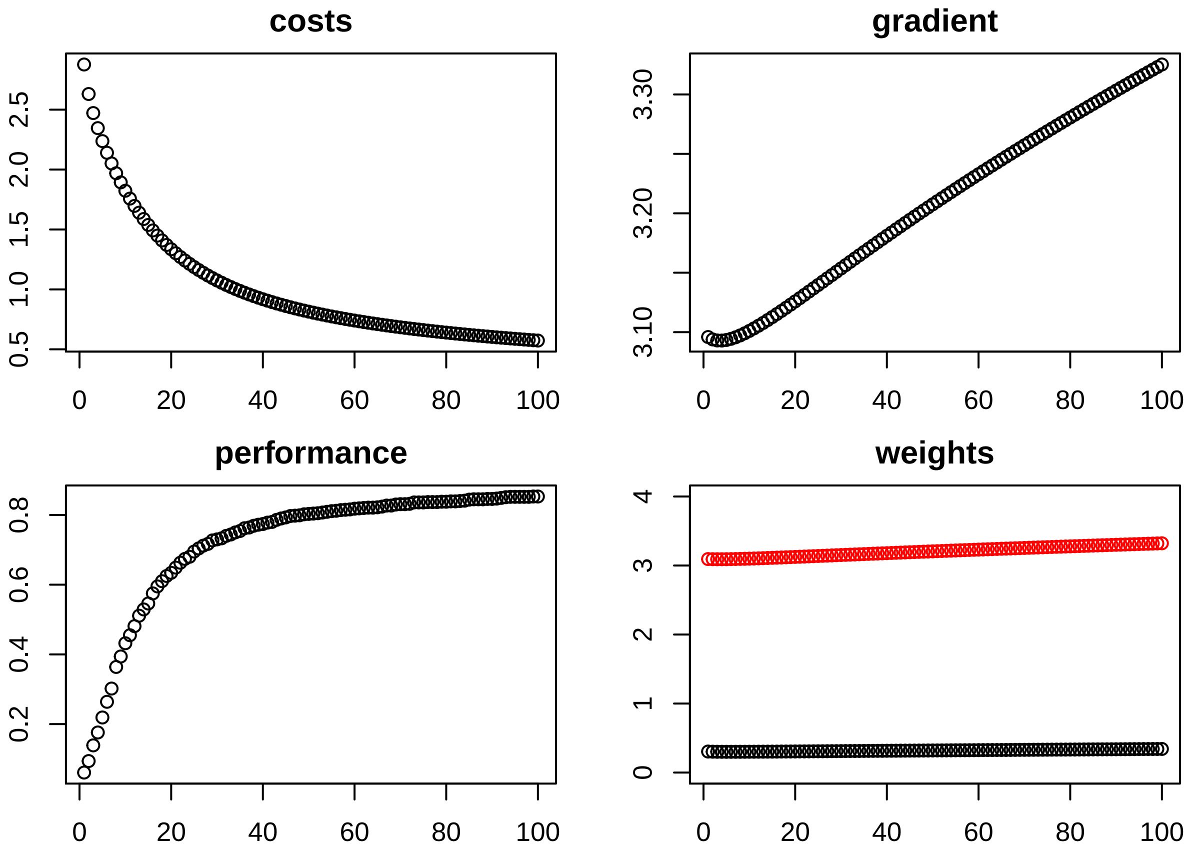
```
sum(( Predict(x,  y, fit$b, fit$W) == y )*1 ) / length(y)
```

```
## [1] 0.854
```

```
sum(( Predict(nx,  ny, fit$b, fit$W) == ny )*1 ) / length(ny)
```

```
## [1] 0.8287
```

```
par(mfrow=c(2,2) , mar=c(2.1,2.1,2.1,2.1) )
plot(fit$costs, main = "costs", ylab=" ")
plot(gradhist(fit$blist,fit$wlist) , main = "gradient", ylab=" ")
plot(fit$peformance, main = "performance", ylab=" ")
plot( sapply(fit$blist, function(BB) norm(BB,"2")) , main = "weights", ylab=" ", ylim = c(0,4))
points( sapply(fit$wlist, function(BB) norm(BB,"2")) , main = "weights", ylab=" ", col = "red")
```



Step 9 Train model on full data

Re-run the model now using the full 60k train data and check performance using the MNIST test data

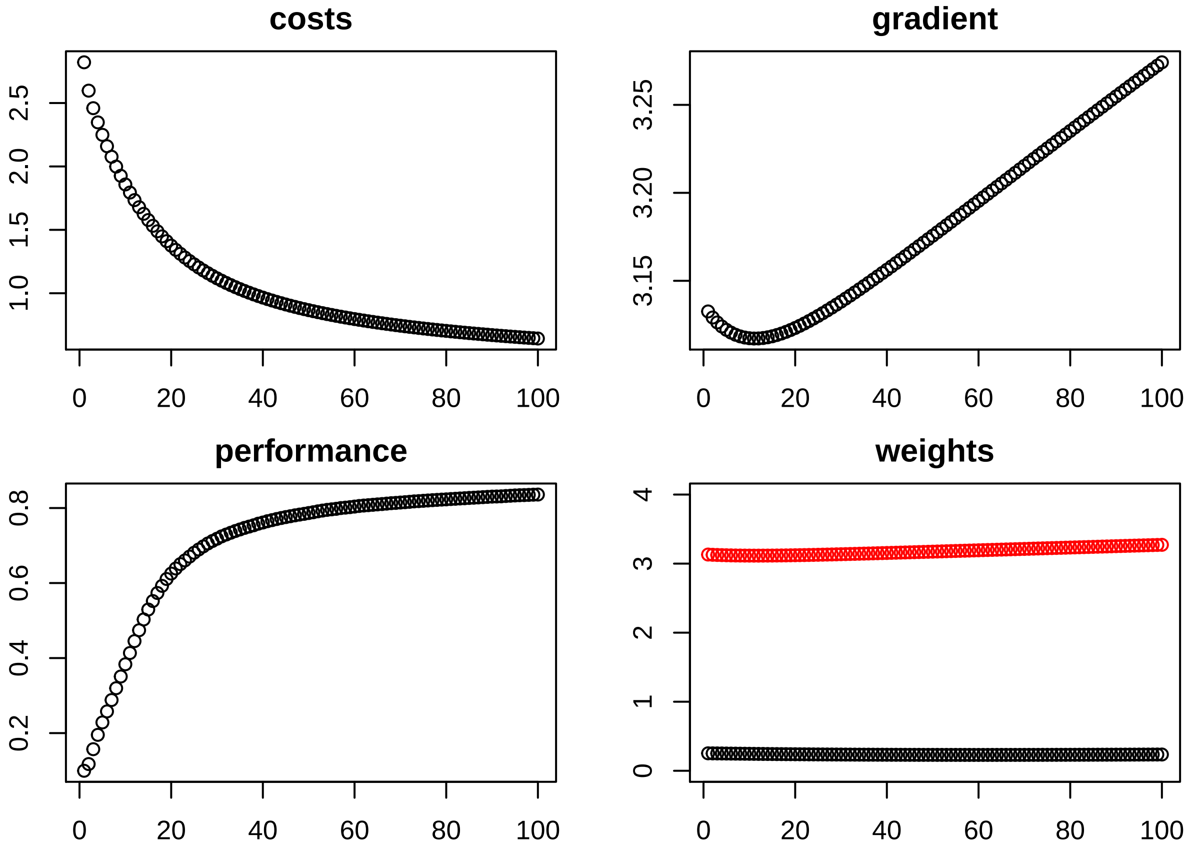
```
x <- unname( mtrain[,-1] )
x <- t(x)
x <- x / 255
y <-  as.matrix( unname(  mtrain[,1] ))

fita <- fit <- softmax.fit( x , y, lr = .1 ,max.its= 100 ); beep("coin")

sum(( Predict(x,  y, fit$b, fit$W) == y )*1 ) / length(y)
```

```
## [1] 0.8365833
```

```
sum(( Predict(nx,  ny, fit$b, fit$W) == ny )*1 ) / length(ny)
```

```
## [1] 0.8474
```

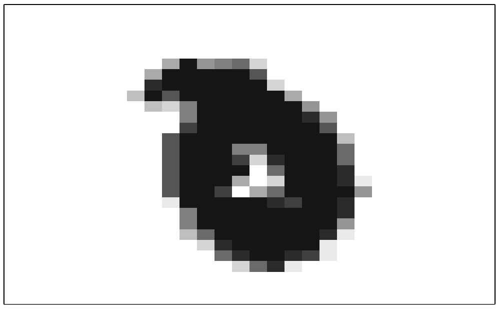
```
par(mfrow=c(2,2) , mar=c(2.1,2.1,2.1,2.1) )
plot(fit$costs, main = "costs", ylab=" ")
plot(gradhist(fit$blist,fit$wlist) , main = "gradient", ylab=" ")
plot(fit$peformance, main = "performance", ylab=" ")
plot( sapply(fit$blist, function(BB) norm(BB,"2")) , main = "weights", ylab=" ", ylim = c(0,4))
points( sapply(fit$wlist, function(BB) norm(BB,"2")) , main = "weights", ylab=" ", col = "red")
```



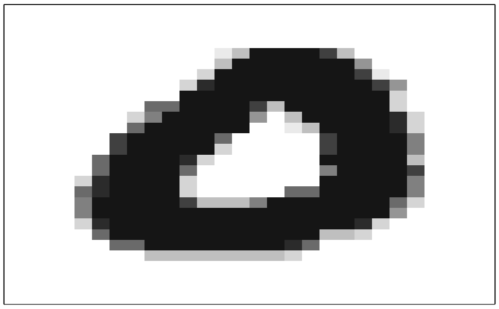
Step 10 Examine Errors

Score the train data using the model from step 10. For each digit, find the image that scored the highest and the lowest. For example, for all of the ‘7’ images, find the one that scored the highest to be a ‘7’ and the one that scored the lowest to be a ‘7’. Plot the best/worst side by side. Annotate your images with their prediction. For us, the ‘worst’ 7 was predicted to be a 2. Use the provided function show_digit2 to display images.

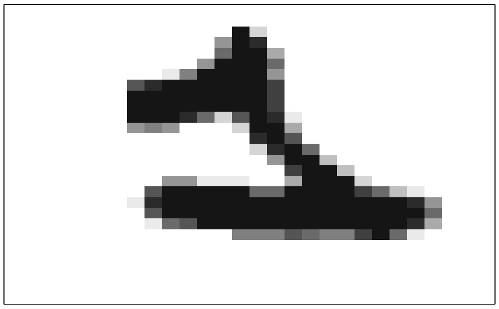
```
pix <- cbind( t(fwd.prop(x , fit$b, fit$W )) , as.matrix(y) , 1:length(y))

for( j in seq(0,9, by = 2)){
par(mfrow=c(2,2) , mar=c(2.1,2.1,2.1,2.1) )
for( k in j:(j+1)){
no = k
pix0 <- pix[which(pix[,11] == no),]
fP <- fwd.prop( x , fit$b , fit$W )
minpix <- round( pix0[which( pix0[,no + 1 ] == min(pix0[,no + 1 ] ) ),,drop=F ], 4 )
maxpix <-  round(pix0[which( pix0[,no + 1 ] == max(pix0[,no + 1 ] ) ), ,drop=F], 4 )

Minpt <- paste0( "Prediction: ",   (which( minpix == max(minpix[,1:10] ) )-1), ", " , max(minpix[,1:10] ),
show_digit2(x[,minpix[,ncol(minpix)]], main = Minpt )
Maxpt <- paste0( "Prediction: ",   (which( maxpix == max(maxpix[,1:10] ) )-1), ", " , max(maxpix[,1:10] ),
show_digit2(x[,maxpix[,ncol(maxpix)]], main = Maxpt  )
}}
```



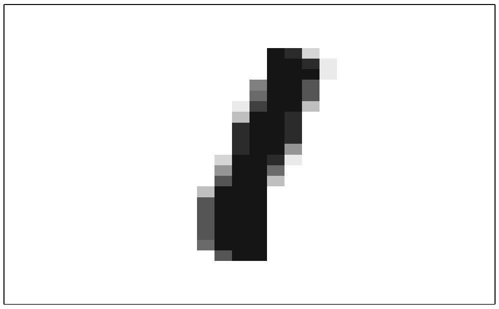
Prediction: 4, 0.2073%



Prediction: 0, 0.9979%

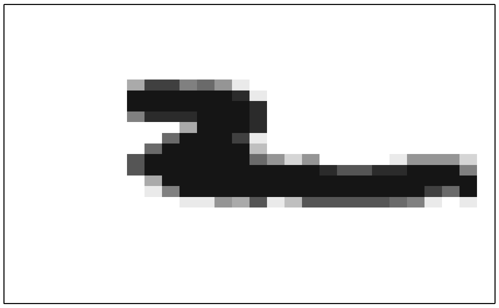


Prediction: 2, 0.714%

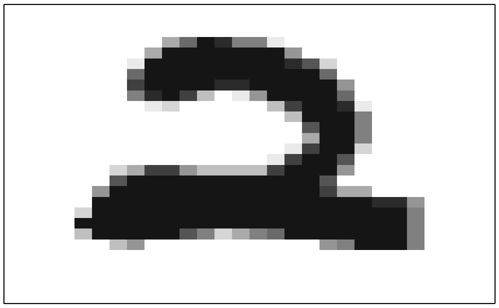


Prediction: 1, 0.9726%

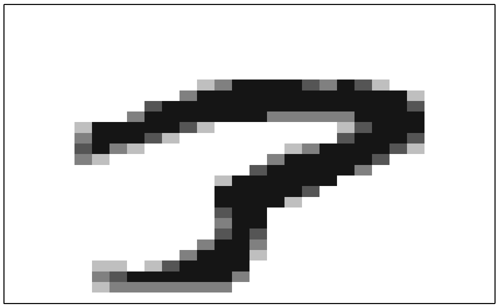
Prediction: 4, 0.3664%



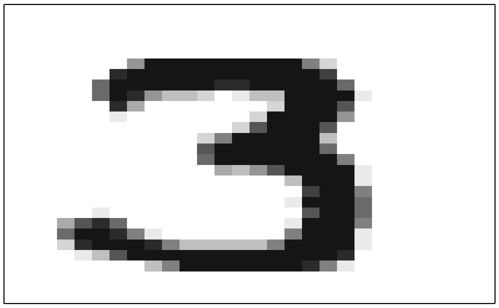
Prediction: 2, 0.9991%



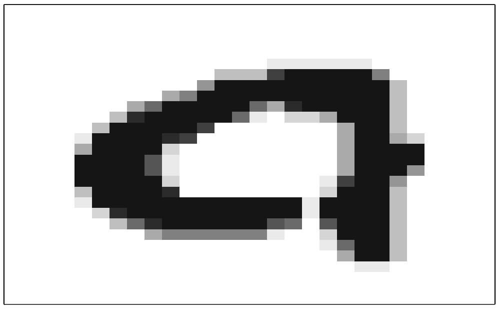
Prediction: 7, 0.8879%



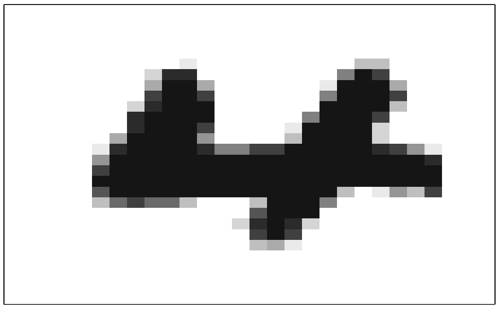
Prediction: 3, 0.9944%



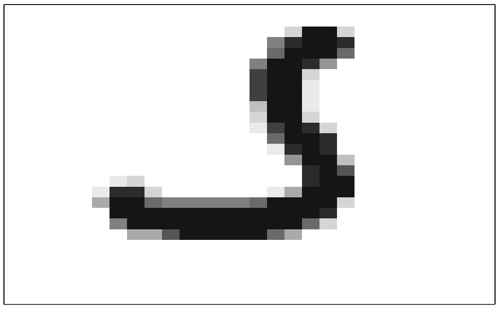
Prediction: 0, 0.976%



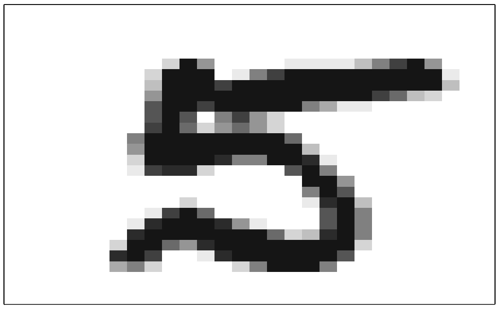
Prediction: 4, 0.9865%

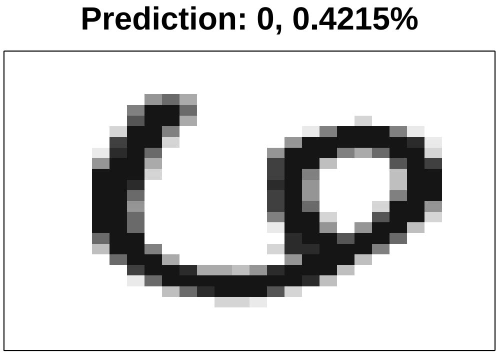


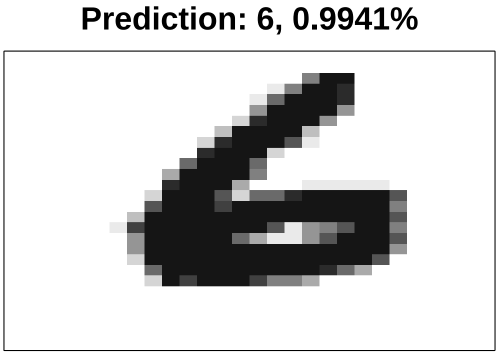
Prediction: 2, 0.6992%

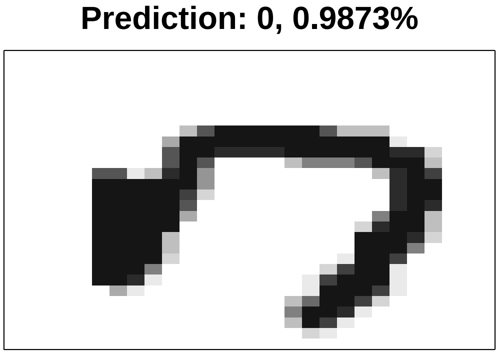


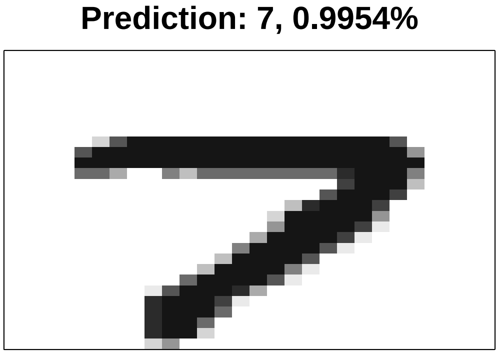
Prediction: 5, 0.9713%

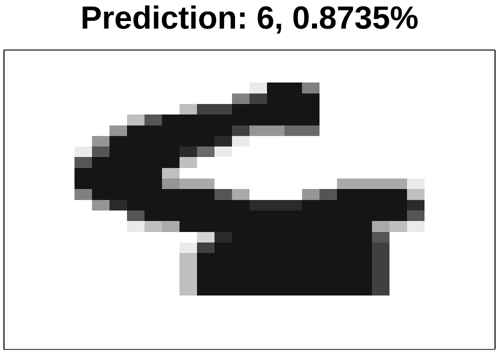


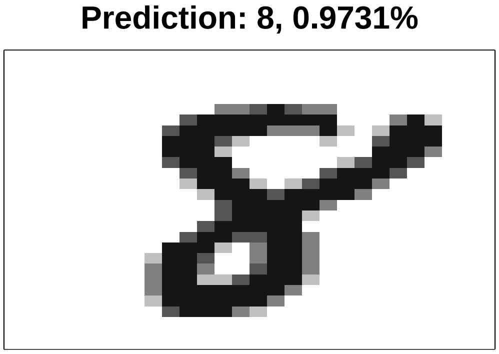


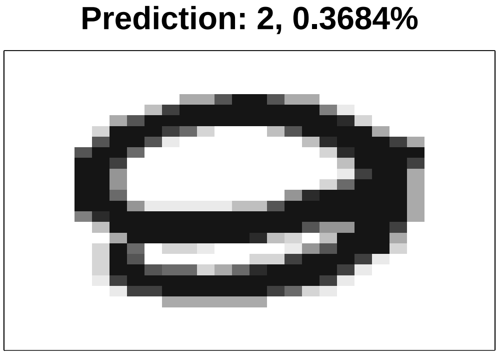


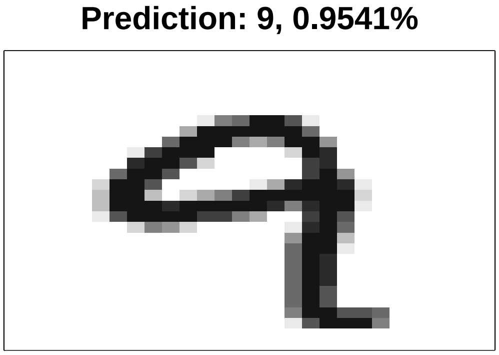












Step 11 Explore learning rate parameters to try to get better performance

Various online reference achieve 92% accuracy on MNIST when using a learning rate of .5 and 1000 iterations. Explore the impact of learning rate and iterations on your MNIST model.

Setting the learning rate is one of the most important parameter in models. Run an experiment to evaluate a range of learning rates and plot the resulting cost histories.

- What combination of leaning rate and iterations gave you the best out-of-sample performance?

```
fit1 <- softmax.fit( x , y, lr = .5 ,max.its= 1000 );#
sum(( Predict(x,  y, fit1$b, fit1$W) == y )*1 ) / length(y)
```

```
## [1] 0.9200667
```

```
sum(( Predict(nx,  ny, fit1$b, fit1$W) == ny )*1 ) / length(ny)
```

```
## [1] 0.9206
```

```
fit2 <- softmax.fit( x , y, lr = 1 ,max.its= 50 );#beep("coin")
sum(( Predict(x,  y, fit2$b, fit2$W) == y )*1 ) / length(y)
```

```
## [1] 0.8896167
```

```
sum(( Predict(nx,  ny, fit2$b, fit2$W) == ny )*1 ) / length(ny)
```

```
## [1] 0.8964
```

```
fit3 <- softmax.fit( x , y, lr = 2 ,max.its= 50 );#beep("coin")
sum(( Predict(x,  y, fit3$b, fit3$W) == y )*1 ) / length(y)
```

```
## [1] 0.8907
```

```
sum(( Predict(nx,  ny, fit3$b, fit3$W) == ny )*1 ) / length(ny)
```

```
## [1] 0.9027
```

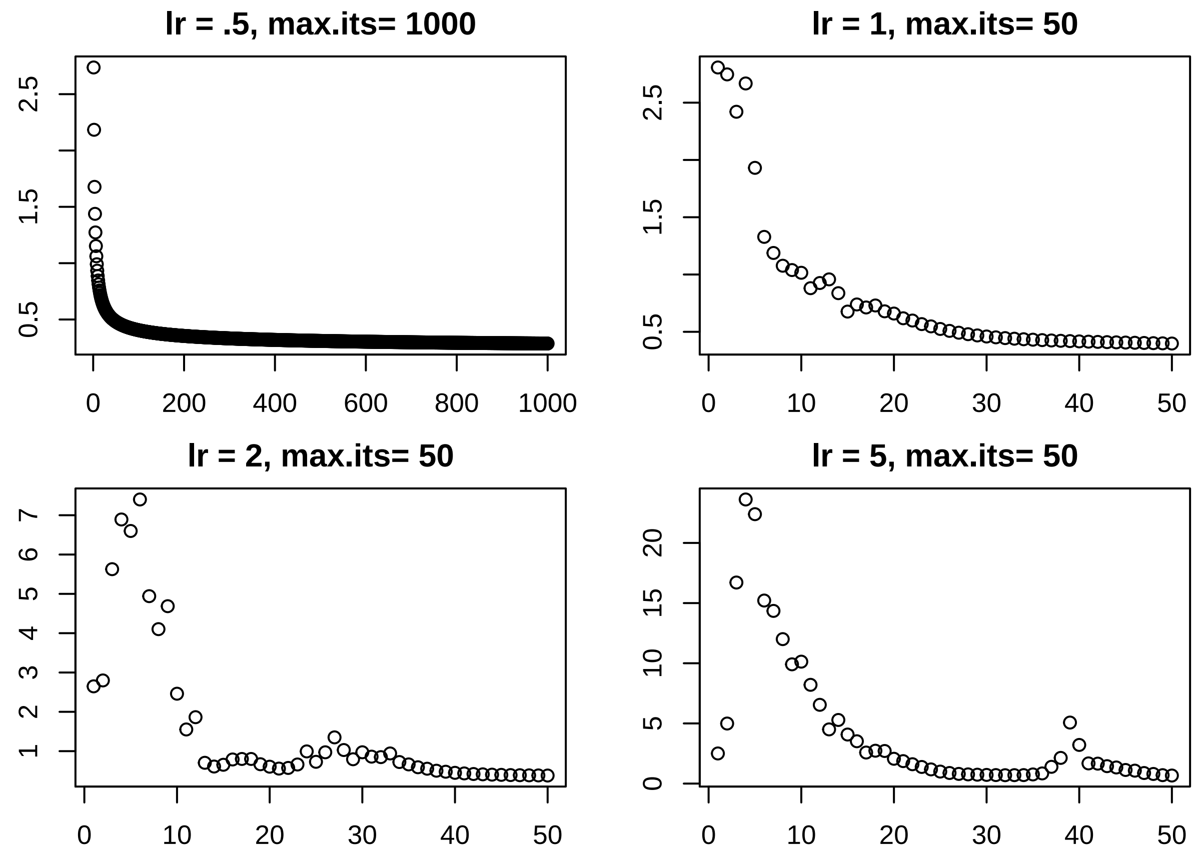
```
fit4 <- softmax.fit( x , y, lr = 5 ,max.its= 50 );#beep("coin")
sum(( Predict(x,  y, fit4$b, fit4$W) == y )*1 ) / length(y)
```

```
## [1] 0.8821833
```

```
sum(( Predict(nx,  ny, fit4$b, fit4$W) == ny )*1 ) / length(ny)
```

```
## [1] 0.8886
```

```
par(mfrow=c(2,2) , mar=c(2.1,2.1,2.1,2.1) )
plot(fit1$costs, main = "lr = .5, max.its= 1000", ylab=" ")
plot(fit2$costs, main = "lr = 1, max.its= 50", ylab=" ")
plot(fit3$costs, main = "lr = 2, max.its= 50", ylab=" ")
plot(fit4$costs, main = "lr = 5, max.its= 50", ylab=" ")
```



```
fita$st
```

```
##    user  system elapsed
##  279.73   13.48  295.88
```