Stochastic gradient descent

Makayla Tang

4/29/2022

Stochastic gradient descent

```
## lm_gd code in class
for(it in 1:niter)
        {
                beta_gd = beta_gd - learn_rate*(-2/n)*(t(x)%*%(y-x%*%beta_gd))

                MSE_new = mean((y - x%*%beta_gd)^2)

                if(verbose)
                {
                        print(MSE_new)

                        x1_grid = seq(0,1,length.out = 100)
                        y_grid_hat =  cbind(rep(1,100),x1_grid)%*%beta_gd

                        plot(x[,2],y)
                        lines(x1_grid,y_grid_hat)
                        Sys.sleep(0.1)
                }
                loss_gd[it] <- MSE_new
        }
```

Task1.

Implement and test a function `lm_sgd` that performs stochastic gradient descent as described above. Describe (in words and small code sections) the main changes you have made to the `lm_gd` code, introduced in class, to implement `lm_sgd`.

Main changes:

- Split the n samples into multiple mini-batches
- Check if the number of mini-batches > the number of observations (the execution will stop if #of mini-batches > # of obs)
- Write a for loop for batches

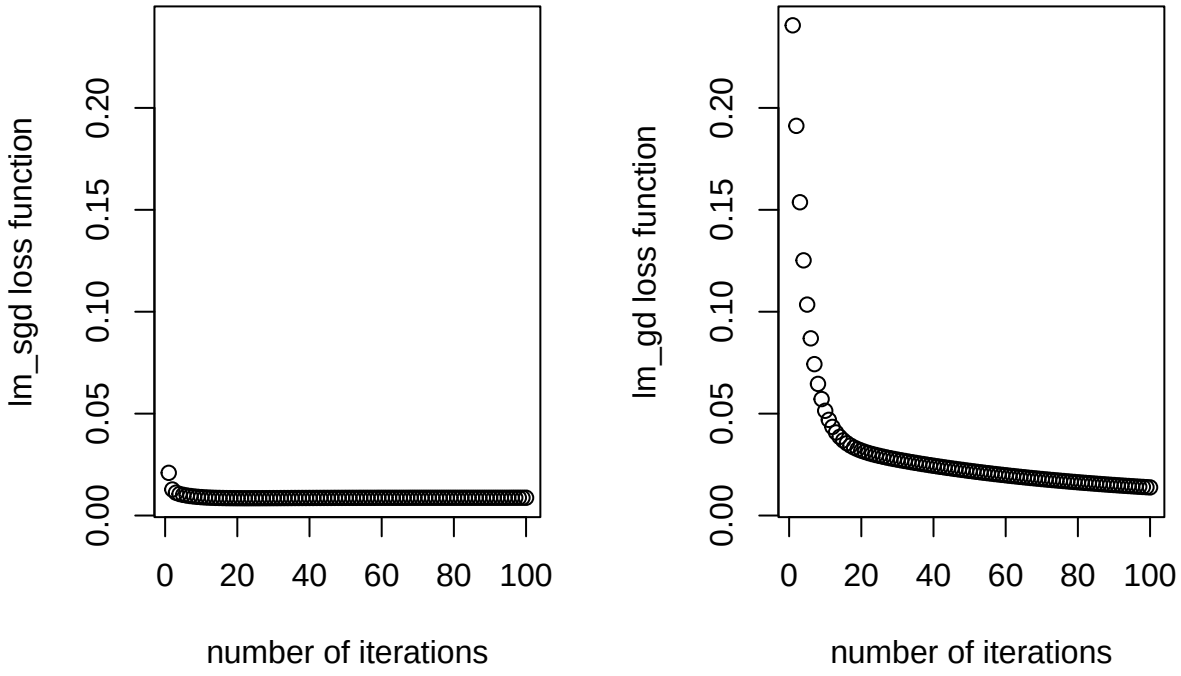
```r
## -------------------------------------
## stochastic gradient descent
## install.packages("caret")
library(caret)
rm(list=ls())
lm_sgd = function(x, y, beta_init = NULL, learn_rate = 0.05,
                  niter = 100, batch=10, verbose = F)
{
        n = nrow(x)
        p = ncol(x)-1

        if(nrow(x) != length(y)) stop("Check x,y dimensions")
        if(nrow(x) < batch) stop("Mini-batches exceed number of observations")
        if(verbose && p>2) stop("p>2, -- Plotting not implemented")
        if(is.null(beta_init)) beta_init = runif(p+1)

        beta_sgd = beta_init

        f <- createFolds(y, k = batch, list = T, returnTrain = F)
        loss_sgd <- rep(0, niter)

        for(it in 1:niter)
        {
                for(mini_batch in 1:batch){
                        x_b <- x[f[[mini_batch]], ]
                        y_b <- y[f[[mini_batch]]]

                        beta_sgd = beta_sgd - learn_rate*
                          (-2/length(y_b))*(t(x_b)%*%(y_b - x_b%*%beta_sgd))

                        MSE_new = mean((y_b - x_b%*%beta_sgd)^2)

                        if(verbose)
                        {
                                print(MSE_new)

                                x1_grid = seq(0,1,length.out = 100)
                                y_grid_hat = cbind(rep(1,100),x1_grid)%*%beta_gd

                                plot(x[,2],y)
                                lines(x1_grid,y_grid_hat)
                                Sys.sleep(0.1)
                        }
                }
                loss_sgd[it] <- MSE_new
        }
        return(list(beta_sgd, loss_sgd))
}
## Test
n = 30
p = 1
beta = rep(1,p+1)
x = cbind(rep(1,n),matrix(runif(n*p,0,1),n,p))
```

```
epsilon = rnorm(n,0,.1)
y = x%*%beta + epsilon
lm_sgd(x, y)
```

Task2.

Display the values of the loss function at every iteration in a scatter plot `# iteration` vs `loss function` for both `lm_gd` and `lm_sgd`. In light of this plot, why do you think the technique is called stochastic gradient descent?

According to the plot, it suggests that stochastic gradient descent converges much faster than gradient descent. The reason for this is that stochastic gradient descent swiftly reaches the optimal values and then keeps oscillating around them. In contrast, using gradient descent takes more time since it needs to run through the complete training set in every iteration to update the parameter values.



Task3.

Generate a list of 20 random vectors `beta_init`. For every element in the list run stochastic gradient descent with that initialization value. Use `purrr:map` for both the generation of the random vectors and the application of `lm_sgd` (see Lecture 4). Display the 20 estimation errors $\|\beta - \beta_0\|^2$, where β_0 is the true beta used to generate the data and β is the estimated one from `lm_sgd`.

```
## [[1]]
## [1] 1.047874
##
## [[2]]
## [1] 1.075157
##
## [[3]]
## [1] 1.054368
```

```
##
## [[4]]
## [1] 1.034218
##
## [[5]]
## [1] 1.036963
##
## [[6]]
## [1] 1.046792
##
## [[7]]
## [1] 1.048145
##
## [[8]]
## [1] 1.041438
##
## [[9]]
## [1] 1.026658
##
## [[10]]
## [1] 1.045807
##
## [[11]]
## [1] 1.036909
##
## [[12]]
## [1] 1.050725
##
## [[13]]
## [1] 1.060734
##
## [[14]]
## [1] 1.036757
##
## [[15]]
## [1] 1.04642
##
## [[16]]
## [1] 1.037651
##
## [[17]]
## [1] 1.027209
##
## [[18]]
## [1] 1.034535
##
## [[19]]
## [1] 1.027078
##
## [[20]]
## [1] 1.042705
```

Code Appendix

```
library(knitr)
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(warning = FALSE, message = FALSE)
## lm_gd code in class
for(it in 1:niter)
        {
                beta_gd = beta_gd - learn_rate*(-2/n)*(t(x)%*%(y-x%*%beta_gd))

                MSE_new = mean((y - x%*%beta_gd)^2)

                if(verbose)
                {
                        print(MSE_new)

                        x1_grid = seq(0,1,length.out = 100)
                        y_grid_hat =  cbind(rep(1,100),x1_grid)%*%beta_gd

                        plot(x[,2],y)
                        lines(x1_grid,y_grid_hat)
                        Sys.sleep(0.1)
                }
                loss_gd[it] <- MSE_new
        }
## ------------------------------------
## stochastic gradient descent
## install.packages("caret")
library(caret)
rm(list=ls())
lm_sgd = function(x, y, beta_init = NULL, learn_rate = 0.05,
                  niter = 100, batch=10, verbose = F)
{
        n = nrow(x)
        p = ncol(x)-1

        if(nrow(x) != length(y)) stop("Check x,y dimensions")
        if(nrow(x) < batch) stop("Mini-batches exceed number of observations")
        if(verbose && p>2) stop("p>2, -- Plotting not implemented")
        if(is.null(beta_init)) beta_init = runif(p+1)

        beta_sgd = beta_init

        f <- createFolds(y, k = batch, list = T, returnTrain = F)
        loss_sgd <- rep(0, niter)

        for(it in 1:niter)
        {
                for(mini_batch in 1:batch){
                        x_b <- x[f[[mini_batch]], ]
                        y_b <- y[f[[mini_batch]]]

                        beta_sgd = beta_sgd - learn_rate*
```

```
                        (-2/length(y_b))*(t(x_b)%*%(y_b - x_b%*%beta_sgd))

                    MSE_new = mean((y_b - x_b%*%beta_sgd)^2)

                    if(verbose)
                    {
                            print(MSE_new)

                            x1_grid = seq(0,1,length.out = 100)
                            y_grid_hat = cbind(rep(1,100),x1_grid)%*%beta_gd

                            plot(x[,2],y)
                            lines(x1_grid,y_grid_hat)
                            Sys.sleep(0.1)
                    }
            }
            loss_sgd[it] <- MSE_new
    }
    return(list(beta_sgd, loss_sgd))
}
## Test
n = 30
p = 1
beta = rep(1,p+1)
x = cbind(rep(1,n),matrix(runif(n*p,0,1),n,p))
epsilon = rnorm(n,0,.1)
y = x%*%beta + epsilon
lm_sgd(x, y)
## -------------------------------------
## linear regression
lm_gd = function(x, y, beta_init = NULL, learn_rate = 0.05,
                 niter = 100, verbose = F)
{
    n = nrow(x)
    p = ncol(x)-1

    if(nrow(x) != length(y)) stop("Check x,y dimensions")
    if(verbose && p>2) stop("p>2, -- Plotting not implemented")
    if(is.null(beta_init)) beta_init = runif(p+1)

    beta_gd = beta_init

    loss_gd <- rep(0, niter)

    for(it in 1:niter)
    {
            beta_gd = beta_gd - learn_rate*(-2/n)*(t(x)%*%(y-x%*%beta_gd))

            MSE_new = mean((y - x%*%beta_gd)^2)

            if(verbose)
            {
                    print(MSE_new)
```

```
                        x1_grid = seq(0,1,length.out = 100)
                        y_grid_hat =  cbind(rep(1,100),x1_grid)%*%beta_gd

                        plot(x[,2],y)
                        lines(x1_grid,y_grid_hat)
                        Sys.sleep(0.1)
                }
                loss_gd[it] <- MSE_new
        }
        return(list(beta_gd, loss_gd))
}
## Test
n = 30
p = 1
beta = rep(1,p+1)
x = cbind(rep(1,n),matrix(runif(n*p,0,1),n,p))
epsilon = rnorm(n,0,.1)
y = x%*%beta + epsilon
## ---------------------------------------
## Store value of loss function
gd_loss <- lm_gd(x, y)[[2]]
sgd_loss <- lm_sgd(x, y)[[2]]
## Plot scatterplots of # iteration vs loss function for both gradient descent and stochastic gradient
par(mfrow=c(1,2))
yrange <- range(gd_loss, sgd_loss)
plot(x=1:100, y=sgd_loss, ylim = yrange,
     xlab = "number of iterations", ylab = "lm_sgd loss function")
plot(x=1:100, y=gd_loss, ylim = yrange,
     xlab = "number of iterations", ylab = "lm_gd loss function")
## ---------------------------------------
library(purrr)
beta_init <- map(1:20, ~ sample(1:100, size=2))
map(beta_init, ~ lm_sgd(x,y)[[1]][[2]])
```