Homework 3 Code

Peng Shao

March 1, 2016

```
setwd("~/Documents/git/DL")

# 1

# minst0_test <- read.table(file = './minst0_test.dat')

# minst0_train <- read.table(file = './minst0_train.dat')

# minst9_test <- read.table(file = './minst9_test.dat')

# minst9_train <- read.table(file = './minst9_train.dat')

# save(list = c('minst0_test', 'minst0_train', 'minst9_test',
# 'minst9_train'), file = './hw3q1_data.RData')


load(file = "./hw3q1_data.RData")
library(EBImage)
draw <- function(mat, main = "") {
    image(t(mat)[, ncol(mat):1], axes = FALSE, col = grey(seq(0, 1, length = 256)),
        main = main)
}
par(mfrow = c(3, 3), oma = c(0, 0, 0, 0), mar = c(1, 1, 1, 1))


m <- matrix(data = unlist(minst9_train[1, ]), nrow = 28, byrow = TRUE)
draw(m)
# a, rotate, clockwise, 90 degree
ya <- matrix(c(0, 1, 0, 0, 0, 0, 0, 0, 0), nrow = 3)
draw(filter2(m, ya))
yb <- matrix(c(0, 0, 0, 0, 0, 0, 0, 1, 0), nrow = 3)
draw(filter2(m, yb))
yc <- matrix(c(0, 0, 0, 1, 0, 0, 0, 0, 0), nrow = 3)
draw(filter2(m, yc))
yd <- matrix(c(0, 0, 0, 0, 0, 1, 0, 0, 0), nrow = 3)
draw(filter2(m, yd))
ye1 <- matrix(c(-1, 0, 1, -2, 0, 2, -1, 0, 1), nrow = 3)
draw(filter2(m, ye1))
ye2 <- matrix(c(1, 2, 1, 0, 0, 0, -1, -2, -1), nrow = 3)
draw(filter2(m, ye2))
ye3 <- matrix(c(0, -1, 0, -1, 4, -1, 0, -1, 0), nrow = 3)
draw(filter2(m, ye3))
ye4 <- matrix(c(-1, -1, -1, -1, 8, -1, -1, -1, -1), nrow = 3)
draw(filter2(m, ye4))

# b
par(mfrow = c(2, 3))
```

```r
yb_3 <- matrix(rep(1/3^2, 3^2), nrow = 3)
yb_5 <- matrix(rep(1/5^2, 5^2), nrow = 5)
yb_7 <- matrix(rep(1/7^2, 7^2), nrow = 7)
draw(filter2(m, yb_3))
draw(filter2(m, yb_5))
draw(filter2(m, yb_7))
ys_3 <- matrix(numeric(3^2), nrow = 3)
ys_3[(3 + 1)/2, (3 + 1)/2] <- 2
ys_3 <- ys_3 - yb_3
ys_5 <- matrix(numeric(5^2), nrow = 5)
ys_5[(5 + 1)/2, (5 + 1)/2] <- 2
ys_5 <- ys_5 - yb_5
ys_7 <- matrix(numeric(7^2), nrow = 7)
ys_7[(7 + 1)/2, (7 + 1)/2] <- 2
ys_7 <- ys_7 - yb_7
draw(filter2(m, ys_3))
draw(filter2(m, ys_5))
draw(filter2(m, ys_7))
# yu_3 <- matrix(runif(3^2), nrow = 3) yu_5 <- matrix(runif(5^2), nrow = 5)
# yu_7 <- matrix(runif(7^2), nrow = 7) image(filter2(m, yu_3), axes = FALSE,
# col = grey(seq(0, 1, length = 256))) image(filter2(m, yu_5), axes = FALSE,
# col = grey(seq(0, 1, length = 256))) image(filter2(m, yu_7), axes = FALSE,
# col = grey(seq(0, 1, length = 256)))

# c
pooling <- function(x, pool, method) {
    m <- nrow(x)/pool
    n <- ncol(x)/pool
    z <- matrix(nrow = m, ncol = n)
    func <- ifelse(method == "mean", mean, max)
    for (i in 1:m) {
        for (j in 1:n) {
            z[i, j] <- func(x[(pool * (i - 1) + 1):(pool * i), (pool * (j -
                1) + 1):(pool * j)])
        }
    }
    z
}

par(mfrow = c(2, 3))
draw(m)
draw(pooling(m, 4, "mean"))
draw(pooling(m, 4, "max"))
draw(m)
draw(pooling(m, 7, "mean"))
draw(pooling(m, 7, "max"))


## d
cov_laplacian <- filter2(m, ye3)
detect_ReLU <- ifelse(cov_laplacian > 0, cov_laplacian, 0)
pool_max <- pooling(detect_ReLU, 4, "max")
par(mfrow = c(2, 2))
```

```r
draw(m)
draw(cov_laplacian)
draw(detect_ReLU)
draw(pool_max)


## e
train_data <- rbind(cbind(minst9_train, label = 1)[1:100, ], cbind(minst0_train,
    label = 0)[1:100, ])
sigmoid <- function(x) 1/(1 + exp(-x))
forward_prop <- function(pars) {
    kern_1 <- matrix(pars[1:9], nrow = 3, ncol = 3)
    kern_2 <- matrix(pars[10:18], nrow = 3, ncol = 3)
    kern_3 <- matrix(pars[19:27], nrow = 3, ncol = 3)
    kern_4 <- matrix(pars[28:36], nrow = 3, ncol = 3)
    kern_5 <- matrix(pars[37:45], nrow = 3, ncol = 3)
    alpha_0 <- matrix(pars[46:55], ncol = 10)
    alpha_1 <- matrix(pars[56:255], ncol = 10)
    beta_0 <- matrix(pars[256:256], ncol = 1)
    beta_1 <- matrix(pars[257:266], ncol = 1)
    outs <- numeric(nrow(train_data))
    for (i in 1:nrow(train_data)) {
        temp_m <- matrix(data = unlist(train_data[i, -785]), nrow = 28, byrow = TRUE)
        cov_laplacian_1 <- filter2(temp_m, kern_1)
        detect_ReLU_1 <- ifelse(cov_laplacian_1 > 0, cov_laplacian_1, 0)
        pool_max_1 <- pooling(detect_ReLU_1, 14, "max")
        cov_laplacian_2 <- filter2(temp_m, kern_2)
        detect_ReLU_2 <- ifelse(cov_laplacian_2 > 0, cov_laplacian_2, 0)
        pool_max_2 <- pooling(detect_ReLU_2, 14, "max")
        cov_laplacian_3 <- filter2(temp_m, kern_3)
        detect_ReLU_3 <- ifelse(cov_laplacian_3 > 0, cov_laplacian_3, 0)
        pool_max_3 <- pooling(detect_ReLU_3, 14, "max")
        cov_laplacian_4 <- filter2(temp_m, kern_4)
        detect_ReLU_4 <- ifelse(cov_laplacian_4 > 0, cov_laplacian_4, 0)
        pool_max_4 <- pooling(detect_ReLU_4, 14, "max")
        cov_laplacian_5 <- filter2(temp_m, kern_5)
        detect_ReLU_5 <- ifelse(cov_laplacian_5 > 0, cov_laplacian_5, 0)
        pool_max_5 <- pooling(detect_ReLU_5, 14, "max")
        input <- t(as.numeric(c(pool_max_1, pool_max_2, pool_max_3, pool_max_4,
            pool_max_5)))
        output <- (input %*% alpha_1 + alpha_0) %*% beta_1 + beta_0
        outs[i] <- sigmoid(output)
    }
    outs
}

cost_function <- function(y, yhat) {
    -mean(y * log(yhat) + (1 - y) * log(1 - yhat))
}


# start_time <- Sys.time() forward_prop(pars) dur <- Sys.time() - start_time
```

```r
INITIAL_PARTICLE <- function() {
    cur_loc <- runif(266, -0.1, 0.1)
    cur_velocity <- runif(266, -0.1, 0.1)
    cur_pred <- forward_prop(cur_loc)
    cur_cost <- cost_function(train_data$label, cur_pred)
    best_loc <- cur_loc
    best_cost <- cur_cost
    return(list(cur_loc = cur_loc, cur_velocity = cur_velocity, cur_cost = cur_cost,
        best_loc = best_loc, best_cost = best_cost))
}
# start_time <- Sys.time() p1 <- INITIAL_PARTICLE() dur <- Sys.time() -
# start_time Time difference of 2.691472 secs

INITIAL_SWARM <- function(n) {
    swarm <- list()
    for (i in 1:n) {
        swarm[[i]] <- INITIAL_PARTICLE()
    }
    return(swarm)
}
# start_time <- Sys.time() swarm <- INITIAL_SWARM(5) dur <- Sys.time() -
# start_time Time difference of 13.50203 secs

GET_GLOBAL_BEST <- function(swarm) {
    g_best <- swarm[[1]]
    for (i in 1:length(swarm)) {
        if (swarm[[i]]$cur_cost < g_best$cur_cost) {
            g_best <- swarm[[i]]
        }
    }
    return(g_best)
}

UPDATE_VELOCITY <- function(particle, g_best, w = 0.729, c1 = 1.49445, c2 = 1.49445,
    max_v) {
    v1 <- c1 * runif(266) * (particle$best_loc - particle$cur_loc)
    v2 <- c2 * runif(266) * (g_best$best_loc - particle$cur_loc)
    particle$cur_velocity <- ifelse(w * particle$cur_velocity + v1 + v2 > max_v,
        max_v, ifelse(w * particle$cur_velocity + v1 + v2 < -max_v, -max_v,
            w * particle$cur_velocity + v1 + v2))
    return(particle)
}

UPDATE_LOCATION <- function(particle, bond) {
    particle$cur_loc <- ifelse(particle$cur_loc + particle$cur_velocity > bond[2],
        bond[2], ifelse(particle$cur_loc + particle$cur_velocity < bond[1]),
        bond[1], particle$cur_loc + particle$cur_velocity)
    cur_pred <- forward_prop(particle$cur_loc)
    particle$cur_cost <- cost_function(train_data$label, cur_pred)
    if (particle$cur_cost < particle$best_cost) {
        particle$best_cost <- particle$cur_cost
        particle$best_loc <- particle$cur_loc
    }
```

```r
    return(particle)
}



SEARCH <- function(iter = 1000, size = 20, w = 0.729, c1 = 1.49445, c2 = 1.49445,
    max_v = 2, bond = c(-10, 10)) {
    swarm <- INITIAL_SWARM(size)
    g_best <- GET_GLOBAL_BEST(swarm)
    for (i in 1:iter) {
        for (j in length(swarm)) {
            swarm[[j]] <- UPDATE_VELOCITY(swarm[[j]], g_best, w, c1, c2, max_v)
            swarm[[j]] <- UPDATE_LOCATION(swarm[[j]], bond)
        }
        g_best <- GET_GLOBAL_BEST(swarm)
        return(g_best)
    }
}

# start_time <- Sys.time() swarm <- SEARCH(10) dur <- Sys.time() -
# start_time Time difference of 13.50203 secs
```

```r
## 2

# a
k <- 32
ising <- matrix(sample(c(0, 1), k * k, replace = TRUE), ncol = k)
par(mfrow = c(3, 2))
draw(ising, main = "initial status")

betas <- c(0.2, 0.5, 0.75, 0.9, 1.25)
for (n in 1:5) {
    beta <- betas[n]
    ising_vec <- as.numeric(ising)
    for (t in 1:2000) {
        for (i in 1:length(ising_vec)) {
            neighbor <- c()
            if (i + 1 <= length(ising_vec))
```

```r
                neighbor <- c(neighbor, ising_vec[i + 1])
            if (i + k <= length(ising_vec))
                neighbor <- c(neighbor, ising_vec[i + k])
            if (i - 1 > 0)
                neighbor <- c(neighbor, ising_vec[i - 1])
            if (i - k > 0)
                neighbor <- c(neighbor, ising_vec[i - k])
            p <- exp(beta * sum(neighbor == 1))/(exp(beta * sum(neighbor ==
                1)) + exp(beta * sum(neighbor == 0)))
            U <- runif(1)
            if (U < p)
                ising_vec[i] <- 1 else ising_vec[i] <- 0
        }
    }
    ising_mat <- matrix(ising_vec, nrow = k)
    draw(ising_mat, main = paste("beta=", beta, sep = ""))
}


# b
library(pscl)
x <- runif(10)
q <- 1/2
r <- 1/2  #rate, instead of scale, to be same definition in rigamma of package pscl
Sigma <- diag(c(100, 100))
beta0 <- rnorm(1, 0, Sigma[1, 1])
beta1 <- rnorm(1, 0, Sigma[2, 2])
sigma_2 <- rigamma(1, q, r)
betas <- matrix(NA, nrow = 100, ncol = 2)
sigma_2s <- numeric(100)
y <- beta0 + beta1 * x + sigma_2
X <- cbind(1, x)
for (t in 1:10) {
    repeat {
        mu <- solve(t(X) %*% X + sigma_2 * Sigma) %*% t(X) %*% y
        S <- sigma_2 * solve(t(X) %*% X + sigma_2 * Sigma)
        beta0_star <- rnorm(1, mu[1], S[1, 1])
        beta1_star <- rnorm(1, mu[2], S[2, 2])
        sigma_2_star <- rigamma(1, q + 10/2, r + 1/2 * sum((y - beta0_star -
            beta1_star * x)^2))
        y_star <- beta0_star + beta1_star * x + sigma_2_star
        if (ks.test(y, y_star)$p.value > 0.5) {
            beta0 <- beta0_star1
            beta1 <- beta1_star2
            sigma_2 <- sigma_2_star
            betas[t, 1] <- beta0_star
            betas[t, 2] <- beta1_star
            sigma_2s[t] <- sigma_2_star
            break
        }
    }

}
```

```r
# c
myst_im <- as.matrix(read.table(file = "./myst_im.dat"))
draw(myst_im)
size <- nrow(myst_im)
x <- as.numeric(matrix(sample(c(0, 1), size * size, replace = TRUE), ncol = size))
beta <- 0.5
sigma2 <- 1
q <- 1/2
r <- 1/2
tau <- 100
B <- 1
y <- as.numeric(myst_im)
start_q3_c <- Sys.time()
for (t in 1:10) {
    hx <- c()
    hz <- c()
    for (s in 1:20) {
        x_prime <- 1 - x
        for (i in 1:length(x)) {
            neighbor <- c()
            if (i + 1 <= length(x))
                neighbor <- c(neighbor, x[i + 1])
            if (i + k <= length(x))
                neighbor <- c(neighbor, x[i + k])
            if (i - 1 > 0)
                neighbor <- c(neighbor, x[i - 1])
            if (i - k > 0)
                neighbor <- c(neighbor, x[i - k])
            d <- beta * sum(neighbor == x[i]) + (-1/(2 * sigma2) * (y[i] - x[i])^2)
            d_prime <- beta * sum(neighbor == x_prime[i]) + (-1/(2 * sigma2) *
                (y[i] - x_prime[i])^2)
            p <- exp(min(c(d_prime - d), 0))
            U <- runif(1)
            if (U < p)
                x[i] <- x_prime[i]
        }
    }
    sigma2 <- rigamma(1, q + size * size/2, (r + 1/2 * sum((y - x)^2)))
    beta_star <- rnorm(1, beta, B)
    z <- x
    for (M in 1:10) {
        for (i in 1:length(z)) {
            neighbor <- c()
            if (i + 1 <= length(z))
                neighbor <- c(neighbor, z[i + 1])
            if (i + k <= length(z))
                neighbor <- c(neighbor, z[i + k])
            if (i - 1 > 0)
                neighbor <- c(neighbor, z[i - 1])
            if (i - k > 0)
                neighbor <- c(neighbor, z[i - k])
```

```r
            p <- exp(beta_star * sum(neighbor == 1))/(exp(beta_star * sum(neighbor ==
                1)) + exp(beta_star * sum(neighbor == 0)))
            U <- runif(1)
            if (U < p)
                z[i] <- 1 else z[i] <- 0
        }
    }
    for (i in 1:length(z)) {
        neighbor <- c()
        if (i + 1 <= length(z))
            neighbor <- c(neighbor, z[i + 1])
        if (i + k <= length(z))
            neighbor <- c(neighbor, z[i + k])
        if (i - 1 > 0)
            neighbor <- c(neighbor, z[i - 1])
        if (i - k > 0)
            neighbor <- c(neighbor, z[i - k])
        hz[i] <- sum(neighbor == z[i])
    }
    hz <- sum(hz)
    for (i in 1:length(x)) {
        neighbor <- c()
        if (i + 1 <= length(x))
            neighbor <- c(neighbor, x[i + 1])
        if (i + k <= length(z))
            neighbor <- c(neighbor, x[i + k])
        if (i - 1 > 0)
            neighbor <- c(neighbor, x[i - 1])
        if (i - k > 0)
            neighbor <- c(neighbor, x[i - k])
        hx[i] <- sum(neighbor == x[i])
    }
    hx <- sum(hx)
    if (abs(hx - hz) < 0.001 * hx) {
        ratio <- ((dnorm(beta_star, 0, tau))/(dnorm(beta, 0, tau)))/((dnorm(beta,
            beta_star, B))/(dnorm(beta_star, beta, B)))
        u <- runif(1)
        if (u < ratio)
            beta <- beta_star
    }
    print(t)
}
dur_q3_c <- Sys.time() - start_q3_c
```