STAT 9100 Homework 2

Peng Shao

February 9, 2016

Problem 1

For this homework, these two problems have 5 hyperparameters which can be tuned for SGD: the range of initial parameters, learning rate γ, decay coefficient λ and number of hidden variables, batch size m.

	BP(train)	BP(test)	nnet(train)	nnet(test)
range=0.005	4.698	16.873	2.742	22.580
range=0.01	4.051	14.044	2.933	24.355
range=0.07	3.007	17.631	2.544	24.767
range=0.1	2.658	13.315	2.524	15.944
range=0.7	2.199	13.240	2.678	22.991

	BP(train)	BP(test)	nnet(train)	nnet(test)
decay=0.001	2.998	17.835	1.827	33.398
decay=0.01	3.181	19.565	2.894	21.481
decay=0.1	2.867	15.174	2.374	18.503
decay=1	4.782	15.594	3.296	14.139
decay=10	12.282	21.337	7.477	17.329

	train	test
nnet	7.572	16.038
gamma=0.1	12.330	21.103
gamma=0.2	14.598	22.158
gamma=0.3	14.368	24.676
gamma=0.4	15.265	27.010
gamma=0.5	20.350	27.030
gamma=0.6	22.553	30.000
gamma=0.7	27.601	37.458
gamma=0.8	34.347	46.889
gamma=0.9	46.583	47.748
gamma=1	86.020	78.867

	BP(train)	BP(test)	nnet(train)	nnet(test)
n_hidden=3	18.220	26.091	10.889	20.088
n_hidden=5	15.217	22.394	10.174	17.952
n_hidden=10	12.384	21.506	7.519	17.115
n_hidden=20	11.632	21.594	7.157	16.719
n_hidden=50	11.844	22.683	7.625	17.490

	train	test
nnet	7.513	16.705
m=100	13.941	23.016
m=200	10.799	18.467
m=300	8.703	17.085
m=400	7.788	16.612

All the comparison results are listed as above. We can see that

1. the range of initial values of parameters do not significantly affect the `nnet` package, and slightly affect SGD algorithm. The larger the range is, the better the algorithm performs. Furthermore, the result of SGD is better than that of `nnet`, since `nnet` is overfitted.
2. when the coefficients of decay between 0.01 and 1, there are almost not differences among the results.
3. the smaller the learning rate is, the better the result is. But this time, `nnet` is better than SGD.
4. when the number of hidden variables is no less than 10, and then batch size is no less than 300, the result almost becomes the best.

Problem 2

	BP(train)	BP(test)	nnet(train)	nnet(test)
range=0.005	0.224	0.224	0.016	0.037
range=0.01	0.179	0.178	0.016	0.038
range=0.07	0.163	0.164	0.016	0.037
range=0.1	0.227	0.226	0.016	0.037
range=0.7	0.158	0.158	0.016	0.041

	BP(train)	BP(test)	nnet(train)	nnet(test)
decay=0.001	0.071	0.081	0.000	0.015
decay=0.01	0.134	0.124	0.002	0.026
decay=0.1	0.844	0.850	0.015	0.038
decay=1	0.231	0.224	0.112	0.121
decay=10	0.987	1.008	0.715	0.726

	train	test
nnet	0.112	0.121
gamma=0.005	0.210	0.199
gamma=0.006	0.155	0.155
gamma=0.007	0.156	0.157
gamma=0.008	0.164	0.165
gamma=0.009	0.183	0.183
gamma=0.01	0.191	0.186

	BP(train)	BP(test)	nnet(train)	nnet(test)
n_hidden=3	0.835	0.863	0.155	0.161
n_hidden=5	0.182	0.186	0.133	0.141
n_hidden=10	0.156	0.157	0.113	0.121
n_hidden=20	0.140	0.141	0.101	0.109
n_hidden=50	0.133	0.130	0.094	0.100

	train	test
nnet	0.094	0.100
m=50	0.123	0.123
m=100	0.073	0.081
m=142	0.056	0.064

The pattern of results for this problem is not so obvious like that for problem 1, since when I change the tuning parameters, the results do not significantly change. But the results themselves are not stable when the tuning parameters are unchanged.

Appendix

```
house_train_input <- read.table(file = "./House_inputs_train.dat")
house_train_output <- read.table(file = "./House_output_train.dat")
house_test_input <- read.table(file = "./House_inputs_test.dat")
house_test_output <- read.table(file = "./House_output_test.dat")
house_train_input <- sapply(house_train_input, scale)
house_test_input <- sapply(house_test_input, scale)
sigm <- function(x) 1/(1 + exp(-x))
nn_bp_sgd <- function(input, output, gamma = 0.1, lambda = 0.1,
                      maxiter = 100000, size = 10, tol = 1e-4,
                      m = round(nrow(input)/3), rang = 0.01){
        x <- t(as.matrix(cbind(int = rep(1, nrow(input)), input)))
        y <- t(as.matrix(output))
        I <- nrow(x)
        J <- nrow(y)
        alpha <- matrix(data = runif(size*I, -rang, rang), ncol = nrow(x))
        beta <- matrix(data = runif(J*(size+1), -rang, rang), ncol = size+1)
        mse_past <- var(as.numeric(y))
        for(i in 1:maxiter){
                # Sampling
                ind <- sample(1:ncol(x), m)

                # Forward
                z <- sigm(rbind(int = rep(1, ncol(x[, ind])), alpha %*% x[, ind]))

                y_hat <- beta %*% z

                mse <- mean(as.numeric((y[, ind]-y_hat)^2))
                if (abs(mse_past - mse) <= tol) break
                mse_past <- mse
                # Backward
```

```
                beta_update <- -(y[, ind]-y_hat) %*% t(z)/length(y[, ind]) * gamma +
                        lambda * gamma * c(0, beta[-1])/length(y[, ind])
                alpha_update <- -(do.call(rbind, replicate(size, y[, ind]-y_hat,
                                                           simplify=FALSE)) *
                                          z[-1, ] * (1 - z[-1, ]) * beta[, -1]) %*%
                        t(x[, ind])/length(y[, ind]) * gamma + lambda * gamma *
                        cbind(numeric(size), alpha[, -1])/length(y[, ind])

                beta <- beta - beta_update
                alpha <- alpha - alpha_update
        }
        z <- sigm(rbind(int = rep(1, ncol(x)), alpha %*% x))
        y_hat <- beta %*% z
        mse <- mean(as.numeric((y-y_hat)^2))
        list(alpha, beta, mse, i)

}

predict_bp <- function(input, output, nn_bp_out){
        x <- t(as.matrix(cbind(int = rep(1, nrow(input)), input)))
        y <- t(as.matrix(output))
        alpha <- nn_bp_out[[1]]
        beta <- nn_bp_out[[2]]
        z <- sigm(rbind(int = rep(1, ncol(x)), alpha %*% x))
        y_hat <- beta %*% z
        mse <- mean(as.numeric((y-y_hat)^2))
        list(y_hat, mse)

}


nn_bp_1 <- nn_bp_sgd(input = house_train_input, output = house_train_output)
nnet_1 <- nnet(x = house_train_input, y = house_train_output, size = 10,
               linout = TRUE, rang = 0.01, decay = 0.1, maxit = 10000)
pred_bp_1 <- predict_bp(house_test_input, house_test_output, nn_bp_1)
pred_nnet_1 <- predict(nnet_1, house_test_input)
pred_nnet_1_mse <- mean((house_test_output - pred_nnet_1)^2)

var_train <- var(house_train_output[, 1])
var_test <- var(house_test_output[, 1])




wine_input <- read.table(file = "./Wine_input.dat")
wine_output <- read.table(file = "./Wine_output.dat")
ind <- sample(1:nrow(wine_input), round(nrow(wine_input)/5))
wine_train_input <- wine_input[-ind, ]
wine_train_output <- wine_output[-ind, ]
wine_test_input <- wine_input[ind, ]
wine_test_output <- wine_output[ind, ]
wine_train_input <- sapply(wine_train_input, scale)
wine_test_input <- sapply(wine_test_input, scale)
```

```
softmax <- function(y) exp(y)/sum(exp(y))
crossentropy <- function(y, yhat){
        -mean(apply(y * (log(yhat)), 1, sum))
}
nn_bp_sgd_2 <- function(input, output, gamma = 0.005, lambda = 1,
                        maxiter = 100000, size = 10, tol = 1e-6,
                        m = round(nrow(input)/3), rang = 0.01){
        x <- t(as.matrix(cbind(int = rep(1, nrow(input)), input)))
        y <- t(as.matrix(output))
        I <- nrow(x)
        J <- nrow(y)
        alpha <- matrix(data = runif(size*I, -rang, rang), ncol = nrow(x))
        beta <- matrix(data = runif(J*(size+1), -rang, rang), ncol = size+1)
        y_hat <- matrix(rep(1/3, nrow(y)*ncol(y)),
                        nrow = nrow(y), ncol = ncol(y))
        ce_past <- crossentropy(t(y), t(y_hat))
        for(i in 1:maxiter){
                # Sampling
                ind <- sample(1:ncol(x), m)

                # Forward
                z <- sigm(rbind(int = rep(1, ncol(x[, ind])), alpha %*% x[, ind]))

                y_hat <- apply(beta %*% z, 2, softmax)

                ce <- crossentropy(t(y[, ind]), t(y_hat))
                print(ce)
                print(i)
                if (abs(ce_past - ce) <= tol) break
                ce_past <- ce
                # Backward
                beta_update <- -(y[, ind]-y_hat) %*% t(z)/length(y[, ind]) * gamma +
                        lambda * gamma * c(0, beta[-1])/length(y[, ind])
                alpha_update <- -(t(beta[, -1]) %*% (y[, ind]-y_hat) *
                        z[-1, ] * (1 - z[-1, ]))  %*%
                        t(x[, ind])/length(y[, ind]) * gamma + lambda * gamma *
                        cbind(numeric(size), alpha[, -1])/length(y[, ind])

                beta <- beta - beta_update
                alpha <- alpha - alpha_update
        }
        z <- sigm(rbind(int = rep(1, ncol(x)), alpha %*% x))
        y_hat <- apply(beta %*% z, 2, softmax)
        ce <- crossentropy(t(y), t(y_hat))
        list(alpha, beta, y_hat, ce, i)
}

predict_bp_2 <- function(input, output, nn_bp_out){
        x <- t(as.matrix(cbind(int = rep(1, nrow(input)), input)))
        y <- t(as.matrix(output))
        alpha <- nn_bp_out[[1]]
        beta <- nn_bp_out[[2]]
        z <- sigm(rbind(int = rep(1, ncol(x)), alpha %*% x))
```

```
        y_hat <- apply(beta %*% z, 2, softmax)
        ce <- crossentropy(t(y), t(y_hat))
        list(y_hat, ce)
}


nn_bp_2 <- nn_bp_sgd_2(input = wine_train_input, output = wine_train_output)
nnet_2 <- nnet(x = wine_train_input, y = wine_train_output, size = 10,
               softmax = TRUE, rang = 0.01, decay = 10, maxit = 10000)
nnet_2_ce <- crossentropy(wine_train_output, nnet_2$fitted.values)
pred_bp_2 <- predict_bp_2(wine_test_input, wine_test_output, nn_bp_2)
pred_nnet_2 <- predict(nnet_2, wine_test_input)
pred_nnet_2_ce <- crossentropy(wine_test_output, pred_nnet_2)


y_hat <- matrix(rep(1/3, nrow(wine_test_output)*ncol(wine_test_output)),
                nrow = nrow(wine_test_output), ncol = ncol(wine_test_output))
ce_test_base <- crossentropy(wine_test_output, y_hat)
y_hat <- matrix(rep(1/3, nrow(wine_train_output)*ncol(wine_train_output)),
                nrow = nrow(wine_train_output), ncol = ncol(wine_train_output))
ce_train_base <- crossentropy(wine_train_output, y_hat)
save(list = ls(), file = "./hw2output.RData")
```