HW06

Elliot Smith, Eugen Hruska, Varun Suriyanaranana

3/9/2018

Problem 2.1

Eigenvectors

	PC1	PC2	PC3	PC4
Sep_L	0.5042407	0.4142865	-0.7250504	-0.2200226
Sep_W	-0.3463521	0.8995347	0.2353428	0.1244639
Pet_L	0.5693906	0.0596525	0.1878865	0.7980818
Pet_W	0.5491592	0.1250821	0.6193661	-0.5469591

Eigenvalues

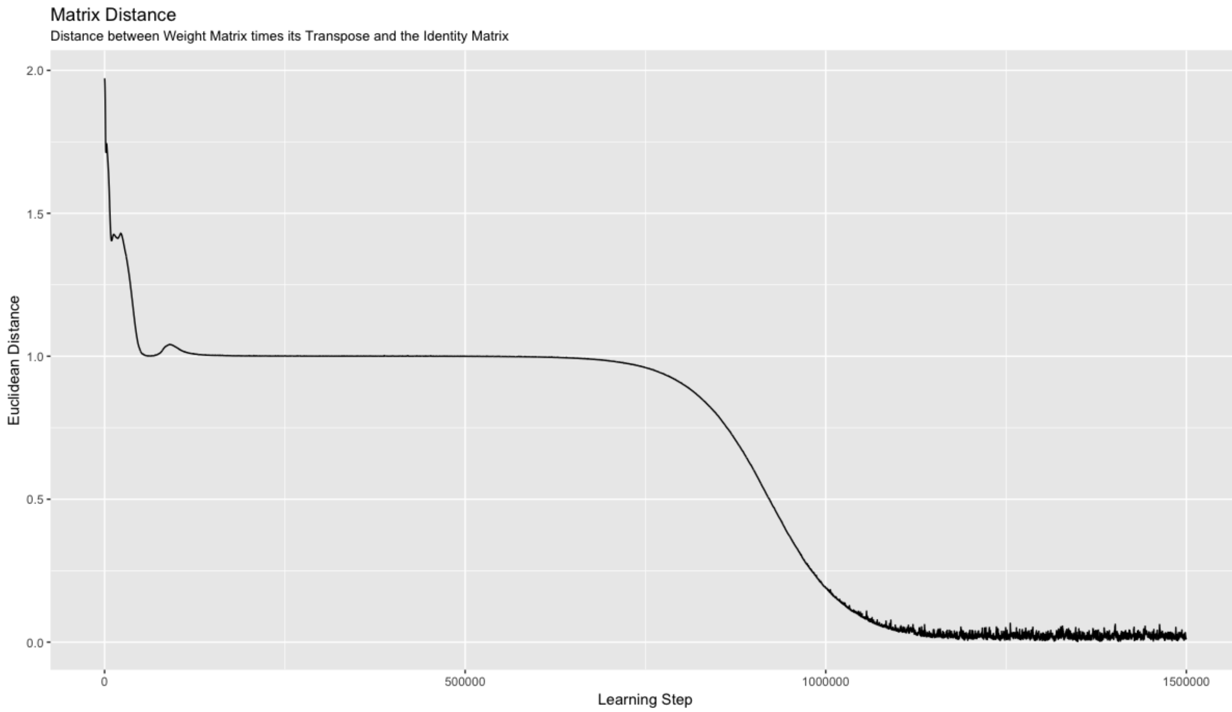
1.736645	0.8805899	0.4341965	0.1417718

Eigenvector Matrix x Eigenvector Matrix Transposed

	Sep_L	Sep_W	Pet_L	Pet_W
Sep_L	1	0	0	0
Sep_W	0	1	0	0
Pet_L	0	0	1	0
Pet_W	0	0	0	1

Problem 2.2

Learning History



Comparison 2.1 to 2.2

Initial Weight Matrix

-0.0260170	-0.0419635	-0.0189653	0.0706779
0.0263623	-0.0742642	-0.0608601	0.0486890
0.0594472	-0.0948662	0.0796796	-0.0946747
-0.0370218	-0.0388697	-0.0578086	0.0351973

Learned Weight Matrix

0.4898328	-0.4187306	0.7248350	-0.2196442
-0.3609935	-0.8972447	-0.2335556	0.1240414
0.5705937	-0.0607485	-0.1885708	0.7982313
0.5517073	-0.1257724	-0.6202459	-0.5470671

Network Information

- Number of Learning Steps: 1500000
- Learning Rate: 0.0005

- Learning History: See graph above (defined by Euclidean distance between weight matrix times its transpose and the Identity matrix)
- Preprocessing: All data is centered (zero mean)
- Error Measure: Euclidean distance

Code Appendix

```
##### Workspace Preparation

## Load in packages

library(ggplot2)

## Load in euclidean distance function

euclidean_distance <- function(p, q){
    sqrt(sum((p - q)^2))
}

##### Problem 2.1

## Load in the iris training data

train_data <- read.table("~/Documents/Rice_University/Spring_2018/NML502/HW04_Part2/iris-train.txt",
                         skip = 8)
train_attr <- train_data[c(TRUE, FALSE), ]
train_attr$V1 <- as.numeric(as.character(train_attr$V1))
train_cats <- train_data[c(FALSE, TRUE), ][ ,2:4]

data <- cbind(train_attr, train_cats)
names(data) <- c("Sep_L", "Sep_W", "Pet_L", "Pet_W", "Setosa", "Versacolor", "Virginica")

## Scale the data

scaled_data <- apply(data[, 1:4], 2, function(x) {(x - mean(x)) / sd(x)})

## Perform the PCA

iris_pca <- prcomp(scaled_data)

eigen_vals <- iris_pca$sdev
eigen_vecs <- iris_pca$rotation

## Confirm we get the identity matrix

iris_pca$rotation %*% t(iris_pca$rotation)



##### Problem 2.2

## Construct the learning function
```

```r
learn_gha <- function(x, n, num_outputs, num_layers, ler_rate, weights) {

    learn_matrices <- list()
    learn_step <- numeric()
    euc_dist <- numeric()

    for (i in 1:n) {

        rand_ind <- sample(1:75, 1)

        y <- weights %*% matrix(x[rand_ind, ], ncol = 1)

        temp_mat <- matrix(0, nrow = 4, ncol = 4)

        for (j in 1:length(y)) {

            if (j == 1) {

                temp_mat[j, ] <- (y[j] * y[j] * weights[j, ])

            } else if (j == 2) {

                temp_mat[j, ] <- (y[j] * y[j - 1] * weights[j - 1, ]) +
                    (y[j] * y[j] * weights[j, ])

            } else if (j == 3) {

                temp_mat[j, ] <- (y[j] * y[j - 2] * weights[j - 2, ]) +
                    (y[j] * y[j - 1] * weights[j - 1, ]) +
                    (y[j] * y[j] * weights[j, ])

            } else if (j == 4) {

                temp_mat[j, ] <- (y[j] * y[j - 3] * weights[j - 3, ]) +
                    (y[j] * y[j - 2] * weights[j - 2, ]) +
                    (y[j] * y[j - 1] * weights[j - 1, ]) +
                    (y[j] * y[j] * weights[j, ])

            }

        }

        weights <- weights +
            ler_rate * (matrix(y, ncol = 1) %*% matrix(x[rand_ind, ], nrow = 1)) -
            ler_rate * temp_mat

        if (i %% 100 == 0) {

            learn_step[length(learn_step) + 1] <- i
            learn_matrices[[length(learn_matrices) + 1]] <- weights %*% t(weights)
            euc_dist[length(euc_dist) + 1] <- euclidean_distance(id_matrix, weights %*% t(weights))

        }
```

```
    }

    ggplot() +
        geom_line(aes(x = learn_step, y = euc_dist)) +
        labs(x = "Learning Step", y = "Euclidean Distance", title = "Matrix Distance",
             subtitle = "Distance between Weight Matrix times its Transpose and the Identity Matrix")

}

## Set the network parameters

x <- scaled_data
n <- 1500000
num_outputs <- c(4, 4)
num_layers <- 1
ler_rate <- 0.0005

## Build the weight matrix

weights <- list()

weights <- matrix(runif(num_outputs[num_layers] * num_outputs[num_layers + 1], min = -0.1, max = 0.1),
                  nrow = num_outputs[num_layers + 1],
                  ncol = num_outputs[num_layers])

## Set the identity matrix

id_matrix <- diag(x = 4)

## Learn the network

learn_gha(x, n, num_outputs, num_layers, ler_rate, weights)
```