S3 method for Shallow neural network and C++ implementation

Eliza Chai

05/25/2022

Problem 1

In this problem, you will refactor the shallow neural network code for binary classification provided in class and use the S3 object-oriented system.

- 1.1

To this end, you will define a new S3 class named `shallow_net` that contains the parameters of your model. More in detail, define the following:

- A constructor that given `p` and `q` returns an object `shallow_net` with randomly initialized parameters `theta` and `beta`.
- An S3 method `predict` that for a given object of the type `shallow_net` and a n_2Öp data matrix `X` returns a n_2-vector with the predicted probabilities
- An S3 method `train`, that for a given a nÖp data matrix `X`, a n-vector `y` of categorical outputs, the learning rate and number of iterations, uses gradient descent to learn the parameters of the neural network.

- 1.3

Re-run Examples 1 and 2 provided in class (see `script_NN_class.R` on canvas) by using the redesigned code

```
## ---------------------------------------------------------------------------
########## Example 1 ##########

n = 100
p = 1
q = 4

set.seed(1)
X = as.matrix(runif(n, -2, 2))
y_prob = sigmoid(2 - 3*X^2)
y = rbinom(n,1,y_prob)

object.shallow_net = shallow_net(p, q)

## Forward pass
```

```r
f_hat = predict(object.shallow_net, X)

## Plot prediction
X_grid = as.matrix(seq(-2,2,length.out = 100))
y_prob_grid = sigmoid(2 - 3*X_grid^2)
f_hat_grid = predict(object.shallow_net, X_grid)

plot(X,y, pch = 20)
lines(X_grid,y_prob_grid, col = 'red')
lines(X_grid,f_hat_grid, col = 'blue')


## Animation

for(s_it in 1:60)
{
        out_nn = train(X, y, learn_rate = .3, n_iter = 100, object.shallow_net)

        ## Update the shallow_net parameters
        object.shallow_net = out_nn

        X_grid = as.matrix(seq(-2,2,length.out = 100))
        y_prob_grid = sigmoid(2 - 3*X_grid^2)
        f_hat_grid = predict(out_nn, X_grid)

        #plot(X,y, pch = 20)
        #lines(X_grid,y_prob_grid, col = 'red')
        #lines(X_grid,f_hat_grid, col = 'blue')
        #Sys.sleep(.8)
}


########## Example 2 ##########

n = 200
p = 1
q = 8

set.seed(1)
X = as.matrix(runif(n, -2, 2))
y_prob = sigmoid(3 + X - 3*X^2 + 3*cos(4*X))
y = rbinom(n,1,y_prob)

object.shallow_net = shallow_net(p, q)

## Forward pass

f_hat = predict(object.shallow_net, X)

## Plot prediction
X_grid = as.matrix(seq(-2,2,length.out = 100))
y_prob_grid = sigmoid(3 + X_grid - 3*X_grid^2 + 3*cos(4*X_grid))
f_hat_grid = predict(object.shallow_net, X_grid)
```

```r
plot(X,y, pch = 20)
lines(X_grid,y_prob_grid, col = 'red')
lines(X_grid,f_hat_grid, col = 'blue')

## Animation

for(s_it in 1:1000)
{
        out_nn = train(X, y, learn_rate = .5, n_iter = 500, object.shallow_net)

        ## Update the shallow_net parameters
        object.shallow_net = out_nn

        X_grid = as.matrix(seq(-2,2,length.out = 100))
        y_prob_grid = sigmoid(3 + X_grid - 3*X_grid^2 + 3*cos(4*X_grid))
        f_hat_grid = predict(object.shallow_net, X_grid)

        #plot(X,y, pch = 20)
        #lines(X_grid,y_prob_grid, col = 'red')
        #lines(X_grid,f_hat_grid, col = 'blue')
        #Sys.sleep(.8)
}
```

Problem 2

- 2.1

Profile the S3 method `train`. Comment on the results.

```r
## ----------------------------------------------------------------------------
library(microbenchmark)
time_comp <- microbenchmark(
        Rcode = train(X, y, learn_rate = .5, n_iter = 500, object.shallow_net),
        times = 100L
)
summary(time_comp)
```

The S3 method `train` is very slow.

- 2.2

Re-implement in C++ the computation of the *gradient of the parameter* `theta` of the neural network. Use the skeleton code below. You will only need to fill `%%%` in the code below with the missing variable(s) (See the expression of the gradient of `theta` in Lecture_NN.Rmd)

```r
## ----------------------------------------------------------------------------
## The matrix X passed has an intercept column (i.e. this is X_aug)
## The matrix A passed does NOT have an intercept column
Rcpp::cppFunction(
 " NumericVector compute_gradient_theta(NumericMatrix X,
```

```
        NumericVector f_hat,
        NumericVector y,
        NumericVector beta,
        NumericMatrix A) {
   int q = beta.size() - 1, p = X.ncol(), n = X.nrow(); // Compute q,p, and n

   NumericMatrix dL_dtheta(p, q); // Matrix with gradient of theta
   double sum_theta;

    for(int l = 0; l < q; l++){
      for(int j = 0; j < p; j++){
        sum_theta = 0;

        for(int i = 0; i < n; i++){
         sum_theta = sum_theta + (f_hat(i) - y(i))*A(i,l)*(1-A(i,l))*beta(l+1)*X(i,j);
        }
        dL_dtheta(j,l) = sum_theta/n;
     }
   }

   return dL_dtheta;
  }
"
)
```

- 2.3

Define a new S3 method `train_fast` that is a variation of `train` that replaces the code for the computation of the gradient of theta with the Cpp function implemented by calling `dL_dtheta <- compute_gradient_theta(X_aug, f_hat, y, beta, A)`

```
## ---------------------------------------------------------------------------
## Create a train_fast function with C++ code
train_fast <- function(X, y, learn_rate = 0.001, n_iter = 200, object)
{
        if ( !is( object, "shallow_net") )
                stop( "predict.shallow_net requires an object of class 'shallow_net'" )
        if ( !is( X, "matrix") )
                stop( "X requires an object of class 'matrix'" )

        beta = object$beta
        theta = object$theta

        q = ncol(theta)
        n = nrow(X)
        p = ncol(X)

        for (it in 1:n_iter)
        {
                if(it %% 1000 == 0) cat("Iter: ", it, "\n")

                # Forward pass
                X_aug = cbind(rep(1,n),X)
```

```
                A = sigmoid(X_aug %*% theta)
                A_aug = cbind(rep(1,n),A)
                f_hat = sigmoid(A_aug %*% beta)

                # Backward pass
                dL_dbeta = (1/n)*t(A_aug)%*%(f_hat - y)
                dL_dtheta = matrix(rep(NA, (p+1)*q), ncol = q)

                dL_dtheta <- compute_gradient_theta(X_aug, f_hat, y, beta, A)

                beta  = beta - learn_rate*dL_dbeta
                theta = theta - learn_rate*dL_dtheta

                # Check objective function value
                # print(loss_func(y,f_hat_func(X,theta,beta)))

        }

        out = list(theta = theta, beta = beta)
        class(out) <- 'shallow_net'

        return(out)
}
```

- 2.4

Check that `train` and `train_fast` give the same results by re-running the Example 1 and 2 with `train_fast`. Compare their performance.

```
## ----------------------------------------------------------------------------
########## Example 1 ##########

n = 100
p = 1
q = 4

set.seed(1)
X = as.matrix(runif(n, -2, 2))
y_prob = sigmoid(2 - 3*X^2)
y = rbinom(n,1,y_prob)

object.shallow_net = shallow_net(p, q)

## Forward pass

f_hat = predict(object.shallow_net, X)

## Plot prediction
X_grid = as.matrix(seq(-2,2,length.out = 100))
y_prob_grid = sigmoid(2 - 3*X_grid^2)
f_hat_grid = predict(object.shallow_net, X_grid)

plot(X,y, pch = 20)
```

```r
lines(X_grid,y_prob_grid, col = 'red')
lines(X_grid,f_hat_grid, col = 'blue')


## Animation

for(s_it in 1:60)
{
        out_nn = train_fast(X, y, learn_rate = .3, n_iter = 100, object.shallow_net)

        ## Update the shallow_net parameters
        object.shallow_net = out_nn

        X_grid = as.matrix(seq(-2,2,length.out = 100))
        y_prob_grid = sigmoid(2 - 3*X_grid^2)
        f_hat_grid = predict(out_nn, X_grid)

        #plot(X,y, pch = 20)
        #lines(X_grid,y_prob_grid, col = 'red')
        #lines(X_grid,f_hat_grid, col = 'blue')
        #Sys.sleep(.8)
}

########## Example 2 ##########

n = 200
p = 1
q = 8

set.seed(1)
X = as.matrix(runif(n, -2, 2))
y_prob = sigmoid(3 + X - 3*X^2 + 3*cos(4*X))
y = rbinom(n,1,y_prob)

object.shallow_net = shallow_net(p, q)

## Forward pass

f_hat = predict(object.shallow_net, X)

## Plot prediction
X_grid = as.matrix(seq(-2,2,length.out = 100))
y_prob_grid = sigmoid(3 + X_grid - 3*X_grid^2 + 3*cos(4*X_grid))
f_hat_grid = predict(object.shallow_net, X_grid)

plot(X,y, pch = 20)
lines(X_grid,y_prob_grid, col = 'red')
lines(X_grid,f_hat_grid, col = 'blue')

## Animation

for(s_it in 1:1000)
{
```

```r
        out_nn = train_fast(X, y, learn_rate = .5, n_iter = 500, object.shallow_net)

        ## Update the shallow_net parameters
        object.shallow_net = out_nn

        X_grid = as.matrix(seq(-2,2,length.out = 100))
        y_prob_grid = sigmoid(3 + X_grid - 3*X_grid^2 + 3*cos(4*X_grid))
        f_hat_grid = predict(object.shallow_net, X_grid)

        #plot(X,y, pch = 20)
        #lines(X_grid,y_prob_grid, col = 'red')
        #lines(X_grid,f_hat_grid, col = 'blue')
        #Sys.sleep(.8)
}

time_com <- microbenchmark(
        Rcode = train(X, y, learn_rate = .5, n_iter = 500, object.shallow_net),
        Rcppcode = train_fast(X, y, learn_rate = .5, n_iter = 500, object.shallow_net),
        times = 100L
)
summary(time_com)
## ----------------------------------------------------------------------------
```

The `train_fast` function is a lot faster than `train` function. Implementing in C++ the computation of the gradient of the parameter `theta` of the neural network helps speed up the computation and increase performance.

Code Appendix

```
## ----------------------------------------------------------------------------
## Setting up the packages, options we'll need:
library(knitr)
knitr::opts_chunk$set(echo = T)
rm(list=ls())

## Create a constructor function for the "shallow_net" class
## given p & q returns initialized parameters theta & beta
shallow_net <- function(p,q) {
        if (p<1) stop("'p' needs to be >= 1" )
        if (q<1) stop("'q' needs to be >= 1" )

        parameter <- list(theta = replicate(q, 0.1*runif(p+1, -.5, .5)),
                          beta = 0.1*runif(q+1, -.5, .5))
        attr(parameter, "class") <- "shallow_net"
        parameter
}

## Create a sigmoid function
sigmoid =  function(x)
{
        1/(1+exp(-x))
}

## Create generic predict function
predict <- function(object, ...) UseMethod("predict")

## Create a method for shallow_net
predict.shallow_net <- function(object, X){
        if ( !is( object, "shallow_net") )
                stop( "predict.shallow_net requires an object of class 'shallow_net'" )
        if ( !is( X, "matrix") )
                stop( "X requires an object of class 'matrix'" )

        sigmoid =  function(x){ 1/(1+exp(-x))}

        theta = object$theta
        beta = object$beta

        n = nrow(X)
        X_aug = cbind(rep(1,n),X)
        A = sigmoid(X_aug %*% theta)
        A_aug = cbind(rep(1,n),A)
        f_pred = sigmoid(A_aug %*% beta)

        return(f_pred)
}

## Test
n = 100
set.seed(1)
```

```r
X = as.matrix(runif(n, -2, 2))
y_prob = sigmoid(2 - 3*X^2)
y = rbinom(n,1,y_prob)

s <- shallow_net(1,4)
predict.shallow_net(s, X)

## Create a train function for shallow_net
train <- function(X, y, learn_rate = 0.001, n_iter = 200, object)
{
        if ( !is( object, "shallow_net") )
                stop( "predict.shallow_net requires an object of class 'shallow_net'" )
        if ( !is( X, "matrix") )
                stop( "X requires an object of class 'matrix'" )

        beta = object$beta
        theta = object$theta

        q = ncol(theta)
        n = nrow(X)
        p = ncol(X)

        for (it in 1:n_iter)
        {
                if(it %% 1000 == 0) cat("Iter: ", it, "\n")

                # Forward pass
                X_aug = cbind(rep(1,n),X)
                A = sigmoid(X_aug %*% theta)
                A_aug = cbind(rep(1,n),A)
                f_hat = sigmoid(A_aug %*% beta)

                # Backward pass
                dloss_beta = (1/n)*t(A_aug)%*%(f_hat - y)
                dloss_theta = matrix(rep(NA, (p+1)*q), ncol = q)

                sum_theta = matrix(rep(0, (p+1)*q), ncol = q)
                for(i in 1:n)
                {
                        sum_theta = sum_theta +
                          X_aug[i,]%*%t((f_hat[i] - y[i])*(A[i,]*(1-A[i,]))*beta[-1])
                }

                dloss_theta = sum_theta/n

                beta  = beta - learn_rate*dloss_beta
                theta = theta - learn_rate*dloss_theta

                # Check objective function value
                # print(loss_func(y,f_hat_func(X,theta,beta)))

        }
```

```r
        out = list(theta = theta, beta = beta)
        class(out) <- 'shallow_net'

        return(out)
}

## -----------------------------------------------------------------------------
########## Example 1 ##########

n = 100
p = 1
q = 4

set.seed(1)
X = as.matrix(runif(n, -2, 2))
y_prob = sigmoid(2 - 3*X^2)
y = rbinom(n,1,y_prob)

object.shallow_net = shallow_net(p, q)

## Forward pass

f_hat = predict(object.shallow_net, X)

## Plot prediction
X_grid = as.matrix(seq(-2,2,length.out = 100))
y_prob_grid = sigmoid(2 - 3*X_grid^2)
f_hat_grid = predict(object.shallow_net, X_grid)

plot(X,y, pch = 20)
lines(X_grid,y_prob_grid, col = 'red')
lines(X_grid,f_hat_grid, col = 'blue')


## Animation

for(s_it in 1:60)
{
        out_nn = train(X, y, learn_rate = .3, n_iter = 100, object.shallow_net)

        ## Update the shallow_net parameters
        object.shallow_net = out_nn

        X_grid = as.matrix(seq(-2,2,length.out = 100))
        y_prob_grid = sigmoid(2 - 3*X_grid^2)
        f_hat_grid = predict(out_nn, X_grid)

        #plot(X,y, pch = 20)
        #lines(X_grid,y_prob_grid, col = 'red')
        #lines(X_grid,f_hat_grid, col = 'blue')
        #Sys.sleep(.8)
}
```

```r
########## Example 2 ##########

n = 200
p = 1
q = 8

set.seed(1)
X = as.matrix(runif(n, -2, 2))
y_prob = sigmoid(3 + X - 3*X^2 + 3*cos(4*X))
y = rbinom(n,1,y_prob)

object.shallow_net = shallow_net(p, q)

## Forward pass

f_hat = predict(object.shallow_net, X)

## Plot prediction
X_grid = as.matrix(seq(-2,2,length.out = 100))
y_prob_grid = sigmoid(3 + X_grid - 3*X_grid^2 + 3*cos(4*X_grid))
f_hat_grid = predict(object.shallow_net, X_grid)

plot(X,y, pch = 20)
lines(X_grid,y_prob_grid, col = 'red')
lines(X_grid,f_hat_grid, col = 'blue')

## Animation

for(s_it in 1:1000)
{
        out_nn = train(X, y, learn_rate = .5, n_iter = 500, object.shallow_net)

        ## Update the shallow_net parameters
        object.shallow_net = out_nn

        X_grid = as.matrix(seq(-2,2,length.out = 100))
        y_prob_grid = sigmoid(3 + X_grid - 3*X_grid^2 + 3*cos(4*X_grid))
        f_hat_grid = predict(object.shallow_net, X_grid)

        #plot(X,y, pch = 20)
        #lines(X_grid,y_prob_grid, col = 'red')
        #lines(X_grid,f_hat_grid, col = 'blue')
        #Sys.sleep(.8)
}

## ---------------------------------------------------------------------------
library(microbenchmark)
time_comp <- microbenchmark(
        Rcode = train(X, y, learn_rate = .5, n_iter = 500, object.shallow_net),
        times = 100L
)
summary(time_comp)
```

```r
## ----------------------------------------------------------------------------
## The matrix X passed has an intercept column (i.e. this is X_aug)
## The matrix A passed does NOT have an intercept column
Rcpp::cppFunction(
 " NumericVector compute_gradient_theta(NumericMatrix X,
        NumericVector f_hat,
        NumericVector y,
        NumericVector beta,
        NumericMatrix A) {
   int q = beta.size() - 1, p = X.ncol(), n = X.nrow(); // Compute q,p, and n

   NumericMatrix dL_dtheta(p, q); // Matrix with gradient of theta
   double sum_theta;

    for(int l = 0; l < q; l++){
      for(int j = 0; j < p; j++){
        sum_theta = 0;

        for(int i = 0; i < n; i++){
         sum_theta = sum_theta + (f_hat(i) - y(i))*A(i,l)*(1-A(i,l))*beta(l+1)*X(i,j);
        }
        dL_dtheta(j,l) = sum_theta/n;
     }
   }

   return dL_dtheta;
  }
"
)

## ----------------------------------------------------------------------------
## Create a train_fast function with C++ code
train_fast <- function(X, y, learn_rate = 0.001, n_iter = 200, object)
{
        if ( !is( object, "shallow_net") )
                stop( "predict.shallow_net requires an object of class 'shallow_net'" )
        if ( !is( X, "matrix") )
                stop( "X requires an object of class 'matrix'" )

        beta = object$beta
        theta = object$theta

        q = ncol(theta)
        n = nrow(X)
        p = ncol(X)

        for (it in 1:n_iter)
        {
                if(it %% 1000 == 0) cat("Iter: ", it, "\n")

                # Forward pass
                X_aug = cbind(rep(1,n),X)
                A = sigmoid(X_aug %*% theta)
```

```r
                A_aug = cbind(rep(1,n),A)
                f_hat = sigmoid(A_aug %*% beta)

                # Backward pass
                dL_dbeta = (1/n)*t(A_aug)%*%(f_hat - y)
                dL_dtheta = matrix(rep(NA, (p+1)*q), ncol = q)

                dL_dtheta <- compute_gradient_theta(X_aug, f_hat, y, beta, A)

                beta  = beta - learn_rate*dL_dbeta
                theta = theta - learn_rate*dL_dtheta

                # Check objective function value
                # print(loss_func(y,f_hat_func(X,theta,beta)))

        }

        out = list(theta = theta, beta = beta)
        class(out) <- 'shallow_net'

        return(out)
}

## --------------------------------------------------------------------------------
########## Example 1 ##########

n = 100
p = 1
q = 4

set.seed(1)
X = as.matrix(runif(n, -2, 2))
y_prob = sigmoid(2 - 3*X^2)
y = rbinom(n,1,y_prob)

object.shallow_net = shallow_net(p, q)

## Forward pass

f_hat = predict(object.shallow_net, X)

## Plot prediction
X_grid = as.matrix(seq(-2,2,length.out = 100))
y_prob_grid = sigmoid(2 - 3*X_grid^2)
f_hat_grid = predict(object.shallow_net, X_grid)

plot(X,y, pch = 20)
lines(X_grid,y_prob_grid, col = 'red')
lines(X_grid,f_hat_grid, col = 'blue')


## Animation
```

```
for(s_it in 1:60)
{
        out_nn = train_fast(X, y, learn_rate = .3, n_iter = 100, object.shallow_net)

        ## Update the shallow_net parameters
        object.shallow_net = out_nn

        X_grid = as.matrix(seq(-2,2,length.out = 100))
        y_prob_grid = sigmoid(2 - 3*X_grid^2)
        f_hat_grid = predict(out_nn, X_grid)

        #plot(X,y, pch = 20)
        #lines(X_grid,y_prob_grid, col = 'red')
        #lines(X_grid,f_hat_grid, col = 'blue')
        #Sys.sleep(.8)
}

########## Example 2 ##########

n = 200
p = 1
q = 8

set.seed(1)
X = as.matrix(runif(n, -2, 2))
y_prob = sigmoid(3 + X - 3*X^2 + 3*cos(4*X))
y = rbinom(n,1,y_prob)

object.shallow_net = shallow_net(p, q)

## Forward pass

f_hat = predict(object.shallow_net, X)

## Plot prediction
X_grid = as.matrix(seq(-2,2,length.out = 100))
y_prob_grid = sigmoid(3 + X_grid - 3*X_grid^2 + 3*cos(4*X_grid))
f_hat_grid = predict(object.shallow_net, X_grid)

plot(X,y, pch = 20)
lines(X_grid,y_prob_grid, col = 'red')
lines(X_grid,f_hat_grid, col = 'blue')

## Animation

for(s_it in 1:1000)
{
        out_nn = train_fast(X, y, learn_rate = .5, n_iter = 500, object.shallow_net)

        ## Update the shallow_net parameters
        object.shallow_net = out_nn

        X_grid = as.matrix(seq(-2,2,length.out = 100))
```

```
        y_prob_grid = sigmoid(3 + X_grid - 3*X_grid^2 + 3*cos(4*X_grid))
        f_hat_grid = predict(object.shallow_net, X_grid)

        #plot(X,y, pch = 20)
        #lines(X_grid,y_prob_grid, col = 'red')
        #lines(X_grid,f_hat_grid, col = 'blue')
        #Sys.sleep(.8)
}

time_com <- microbenchmark(
        Rcode = train(X, y, learn_rate = .5, n_iter = 500, object.shallow_net),
        Rcppcode = train_fast(X, y, learn_rate = .5, n_iter = 500, object.shallow_net),
        times = 100L
)
summary(time_com)
## ----------------------------------------------------------------------------
```