# Stats 315B: Homework 3

*Joe Higgins, Austin Wang, Jessica Wetstone*

*Due 5/20/2018*

## Question 1

**Consider a multi-hidden layer neural network trained by sequential steepest-descent using the weight updating formula $w_t = w_{t-1} - \eta G(w_{t-1})$ Here $t$ labels the observations presented in sequence (time) and $G(w)$ is the gradient of the squared-error criterion evaluated at $w$. Derive a recursive "back-propagation" algorithm for updating all of the network weights at each step. With this algorithm the update for an input weight to a particular hidden node is computed using only the value of its corresponding input (that it weights), the value of the output of the hidden node to which it is input, and an "error signal" from each of the nodes in the next higher layer to which this node is connected. Thus, each node in the network can update its input weights using information provided only by the nodes to which it is connected.**

## Question 2

**Consider a radial basis function network with spherical Gaussian basis of the form $B(x|\mu_m, \sigma_m) = \left(-\frac{1}{2\sigma_m^2}\sum_{j=1}^{n}(x_j - \mu_{jm})^2\right)$, with the function approximation given by $\hat{F}(x) = \sum_{m=1}^{M} a_m B(x|\mu_m, \sigma_m)$ and sum-of-squares error criterion. Derive expressions for the gradient $G(x)$ with respect to all (types of) parameters in the network.**

## Question 3

## Question 4

## Question 5

**Describe $K$—fold cross-validation. What is it used for. What are the advan- tages/disadvantages of using more folds (increasing $K$). When does cross—validation estimate the performance of the actual predicting function being used.**

## Question 6

**Suppose there are several outcome variables $\{y_1, y_2, ..., y_M\}$ associated with a common set of predictor variables $x = \{x_1, x_2, ..., x_n\}$. One could train separate single output neural networks for each outcome y_m or train a single network with multiple outputs, one for each y_m. What are the relative advantages/disadvantages of these two respective approaches. In what situations would one expect each to be better than the other.**

## Question 7

**Spam Email. The data sets spam_stats315B_train.csv, spam_stats315B_test.csv and documentation for this problem are the same as in Homework 2 and can be found in the class web page. You need first to standardize predictors and choose all the weights starting values at random in the interval [-0.5, 0.5].**

see: https://piazza.com/class/jfehm8n4ied2w9?cid=256

```r
rm(list = ls())
```

```r
#data labels
rflabs<-c("make", "address", "all", "3d", "our", "over", "remove",
  "internet","order", "mail", "receive", "will",
  "people", "report", "addresses","free", "business",
  "email", "you", "credit", "your", "font","000","money",
  "hp", "hpl", "george", "650", "lab", "labs",
  "telnet", "857", "data", "415", "85", "technology", "1999",
  "parts","pm", "direct", "cs", "meeting", "original", "project",
  "re","edu", "table", "conference", ";", "(", "[", "!", "$", "#",
  "CAPAVE", "CAPMAX", "CAPTOT","type")

#load the data
data_path <- paste(getwd(),'/data',sep='')
setwd(data_path)
train <- read.csv(file="spam_stats315B_train.csv", header=FALSE, sep=",")
test <- read.csv(file="spam_stats315B_test.csv", header=FALSE, sep=",")
colnames(train)<-rflabs
colnames(test)<-rflabs

#scale the predictors (X) in train and test by the train data
num_cols <- dim(train)[2]
train_X <- train[,c(1:(num_cols-1))]
train_X_means <- apply(train_X,2,mean)
train_X_sds <- apply(train_X,2,sd)

train_scaled_X <- train_X
train_scaled_X <- sweep(train_scaled_X, 2, train_X_means, "-")
train_scaled_X <- sweep(train_scaled_X, 2, train_X_sds, "/")

test_X <- test[,c(1:(num_cols-1))]
test_scaled_X <- test_X
test_scaled_X <- sweep(test_scaled_X, 2, train_X_means, "-")
test_scaled_X <- sweep(test_scaled_X, 2, train_X_sds, "/")

#create y vectors
train_y <- data.frame(train[,'type'])
test_y  <- data.frame(test[,'type'])
```

(a) Fit on the training set one hidden layer neural networks with 1, 2,. . . , 10 hidden units and different sets of starting values for the predictors (obtain in this way one model for each number of units). Which structural model performs best at classifying on the test set?

reference: https://piazza.com/class/jfehm8n4ied2w9?cid=235

```r
get_accuracy <- function(y_hat, y, threshold){
  y_hat[y_hat >  threshold] <- 1
  y_hat[y_hat <= threshold] <- 0
  correct <- y_hat == y
  pct_correct = sum(correct)/length(correct)
  return(pct_correct)
}

wt_rang = 0.5
num_neurons <- seq(1:10)
num_reps <- 10
```

```r
threshold <- .50
accuracies <- list()

#for each structural model
for(size in num_neurons){

  sum_accuracy <- 0

  #average over several random initializations
  for(i in c(1:num_reps)){
    model <- nnet(
        train_scaled_X, train_y, size=num_neurons[size],
        linout = FALSE, entropy = FALSE, softmax = FALSE,
        censored = FALSE, skip = FALSE, rang = wt_rang, decay = 0,
        maxit = 100, Hess = FALSE, trace = FALSE
    )

    y_hat <- predict(model, test_scaled_X)
    sum_accuracy <- sum_accuracy + get_accuracy(y_hat, test_y, threshold)
  }

  accuracies <- c(accuracies, sum_accuracy/num_reps)
}


best_performing_idx <- which.max(accuracies)
best_performing_num_neurons <- num_neurons[best_performing_idx]

cat("Best performing number of hidden layer neurons: ",
    best_performing_num_neurons, "\n")
```

```
## Best performing number of hidden layer neurons:  8
```

(b) Choose the optimal regularization (weight decay for parameters 0,0.1,. . . ,1) for the structural model found above by averaging your estimators of the misclassification error on the test set. The average should be over 10 runs with different starting values. Describe your final best model obtained from the tuning process: number of hidden units and the corresponding value of the regularization parameter. What is an estimation of the misclassification error of your model?

see: https://piazza.com/class/jfehm8n4ied2w9?cid=223

```r
weight_decays <- seq(0,1,.1)
num_reps <- 10
accuracies <- list()

#for each structural model
for(weight_decay in weight_decays){

  sum_accuracy <- 0

  #average over several random initializations
  for(i in c(1:num_reps)){
    model <- nnet(
        train_scaled_X, train_y, size= best_performing_num_neurons,
        linout = FALSE, entropy = FALSE, softmax = FALSE,
        censored = FALSE, skip = FALSE, rang = wt_rang, decay = weight_decay,
        maxit = 100, Hess = FALSE, trace = FALSE
    )

    y_hat <- predict(model, test_scaled_X)
    sum_accuracy <- sum_accuracy + get_accuracy(y_hat, test_y, threshold)
  }

  accuracies <- c(accuracies, sum_accuracy/num_reps)
}

best_performing_idx <- which.max(accuracies)
best_performing_weight_decay <- weight_decays[best_performing_idx]

cat("Best performing weight decay for best structural model: ",
    best_performing_weight_decay, "\n")
```

```
## Best performing weight decay for best structural model:  0.2
```

(c) As in the previous homework the goal now is to obtain a spam filter. Repeat the previous point requiring this time the proportion of misclassified good emails to be less than 1%.

See: https://piazza.com/class/jfehm8n4ied2w9?cid=257

```r
percent <- function(x, digits = 2, format = "f", ...) {
  paste0(formatC(100 * x, format = format, digits = digits, ...), "%")
}

wt_rang = 0.5
num_neurons <- seq(1:10)
weight_decays <- seq(0,1,.1)
num_reps <- 10

find_threshold <- function(model){
  thresholds <- seq(0,1,0.01)
  y_hat <- predict(model, test_scaled_X)

  for(thresh in thresholds){
    y_hat_nonspam <- y_hat[test_y == 0]
    y_hat_nonspam[y_hat_nonspam >  thresh] <- 1
    y_hat_nonspam[y_hat_nonspam <= thresh] <- 0
    nonspam_misclassification_rate <- sum(y_hat_nonspam)/length(y_hat_nonspam)

    if(nonspam_misclassification_rate <= 0.01) {break}
  }
  return(thresh)
}

get_misclassification_rates <- function(model, threshold){
  y_hat <- predict(model, test_scaled_X)
  y_hat[y_hat >  threshold] <- 1
  y_hat[y_hat <= threshold] <- 0

  correct <- y_hat == test_y
  correct_spam    <- correct[test_y == 1]
  correct_nonspam <- correct[test_y == 0]

  misclassification_rate <- 1 - sum(correct)/length(correct)
  spam_misclassification_rate <- 1 - sum(correct_spam)/length(correct_spam)
  nonspam_misclassification_rate <- 1 - sum(correct_nonspam)/length(correct_nonspam)


  return(c(misclassification_rate, spam_misclassification_rate, nonspam_misclassification_rate))
}


models <- list()
for(i in c(1:length(num_neurons))){

  for(j in c(1:length(weight_decays))){
    model <- nnet(
        train_scaled_X, train_y, size=num_neurons[i],
        linout = FALSE, entropy = FALSE, softmax = FALSE,
        censored = FALSE, skip = FALSE, rang = wt_rang, decay = weight_decays[j],
        maxit = 100, Hess = FALSE, trace = FALSE
    )
    models[[paste(i,j,sep="_")]] <- model
  }
}

model_thresholds <- lapply(models, function(x) {find_threshold(x)})

misclassification_rates <- mapply(get_misclassification_rates, models, model_thresholds)
best_model_idx <- which.min(misclassification_rates[1,])

best_model <- models[[best_model_idx]]
best_misclassification_rates <- misclassification_rates[,best_model_idx]

cat("Best model: ","\n")
cat("Hidden layer size: ", best_model$n[2], "\n")
cat("Decay: ", best_model$decay, "\n\n")

cat("Misclassifiction rates: ","\n")
cat(percent(best_misclassification_rates[1]), ": Overall","\n")
cat(percent(best_misclassification_rates[2]), ": Spam","\n")
cat(percent(best_misclassification_rates[3]), ": Nonspam","\n")
```

```
## Best model:
## Hidden layer size:  10
## Decay:  0.1
##
## Misclassifiction rates:
## 5.80% : Overall
## 12.94% : Spam
## 0.98% : Nonspam
```