Statistical_Learning_Exercise_4

2022-05-23

Exercise 1

We will develop a predictive model for the South African heart disease data available as data object `SAheart` in package **ElemStatLearn**. First, we divide into training (80%) and test (20%) dataset to later compare the predictive performance of the different models. As a next step, we fit a logistic regression model with stepwise variable selection using only linear effects for the covariates (`glm` and `step`).

```
data(SAheart)

train_index <- sample(1:nrow(SAheart), nrow(SAheart)*0.8, replace = FALSE)
train <- SAheart[train_index,]
test <- SAheart[-train_index,]

y_values <- as.data.frame(matrix(NA, 93, 4)) #matrix for predicted and original y values (test dataset)
colnames(y_values) <- c("chd", "Stepwise_selection", "Lasso", "Gamboost")
y_values$chd <- test$chd

misclassification <- matrix(NA,1,3) #matrix for misclassification rates
colnames(misclassification) <- c("Stepwise_selection", "Lasso", "Gamboost")


model_step <- glm(chd ~ ., data = train, family = binomial)
model_step_null <- glm(chd ~ 1, data = train, family = binomial)
model_step <- step(model_step, scope = list(upper=model_step, lower=model_step_null), direction = "both
```

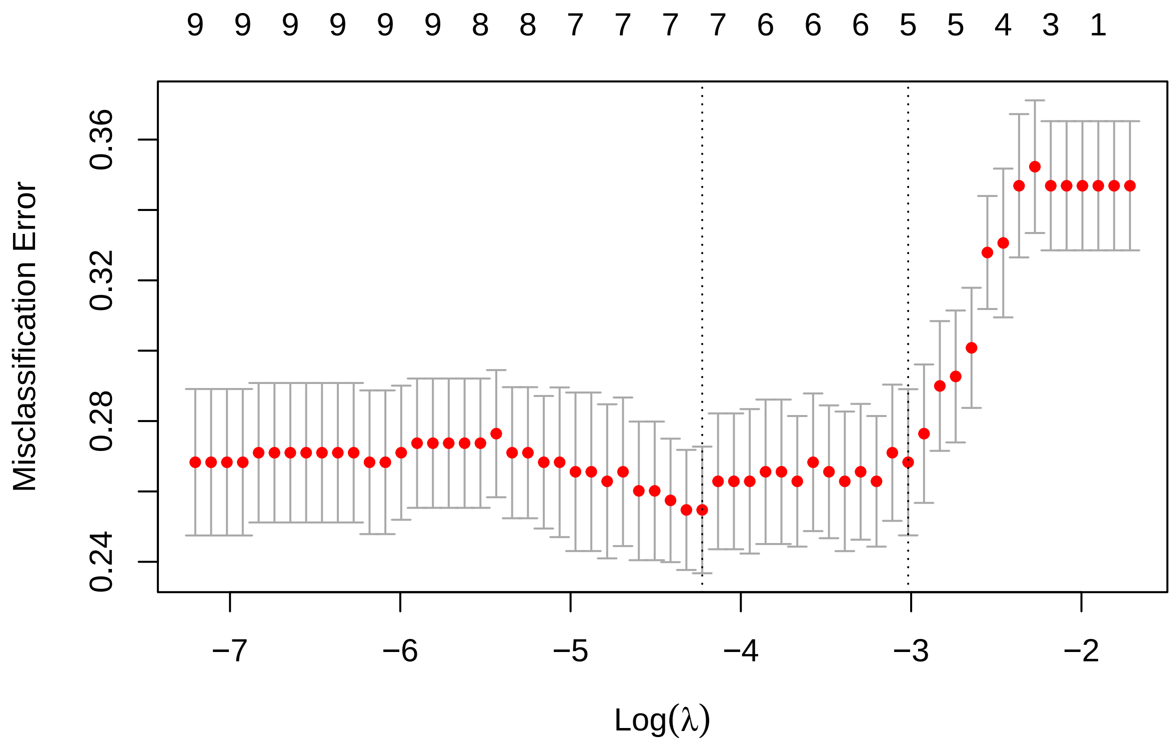
```
## Start:  AIC=391.41
## chd ~ sbp + tobacco + ldl + adiposity + famhist + typea + obesity +
##     alcohol + age
##
##             Df Deviance    AIC
## - alcohol    1   371.52 389.52
## - adiposity  1   371.56 389.56
## - obesity    1   373.03 391.03
## <none>           371.41 391.41
## - sbp        1   373.92 391.92
## - ldl        1   379.09 397.09
## - tobacco    1   379.63 397.63
## - famhist    1   381.66 399.66
## - age        1   383.83 401.83
## - typea      1   383.85 401.85
##
## Step:  AIC=389.52
## chd ~ sbp + tobacco + ldl + adiposity + famhist + typea + obesity +
##     age
```

```
##
##              Df Deviance    AIC
## - adiposity  1   371.65 387.65
## - obesity    1   373.11 389.11
## <none>           371.52 389.52
## - sbp        1   373.96 389.96
## + alcohol    1   371.41 391.41
## - ldl        1   379.73 395.73
## - tobacco    1   379.74 395.74
## - famhist    1   381.66 397.66
## - typea      1   383.85 399.85
## - age        1   384.23 400.23
##
## Step:  AIC=387.65
## chd ~ sbp + tobacco + ldl + famhist + typea + obesity + age
##
##              Df Deviance    AIC
## <none>           371.65 387.65
## - obesity    1   373.87 387.87
## - sbp        1   374.13 388.13
## + adiposity  1   371.52 389.52
## + alcohol    1   371.56 389.56
## - tobacco    1   379.90 393.90
## - ldl        1   380.89 394.89
## - famhist    1   381.75 395.75
## - typea      1   383.86 397.86
## - age        1   390.40 404.40
```

```
#we also tried out the command step(model_step) without further specificatins -->
#this yields the same results per default

step_predict <- predict(model_step, newdata = test, type = "response")
y_values$Stepwise_selection <- ifelse(step_predict > 0.5, 1, 0)
misclassification[,"Stepwise_selection"] <- nrow(y_values[y_values$chd != y_values$Stepwise_selection,])
```

Then, we fit a logistic regression model with Lasso penalty using only linear effects for the covariates. The Lasso penalty is selected with cross-validation (1-SE rule). Friedman, Hastie, and Tibshirani (2010) describe this rule as choosing “the most par- simonious model whose error is no more than one standard error above the error of the best model”. This should be a hedge against overfitting.

```
train$famhist <- ifelse(train$famhist == "Absent", 0, 1) #need to convert to dummy variable
test$famhist <- ifelse(test$famhist == "Absent", 0, 1) #need to convert to dummy variable
#s.t. cv.glmnet can be used

X_train <- as.matrix(train[, 1:9])
y_train <- as.vector(train[, 10])
X_test <- as.matrix(test[, 1:9])
y_test <- as.vector(test[, 10])

cv_lasso <- cv.glmnet(X_train, y_train, family = "binomial", type.measure = "class")
#type.measure = "class" takes misclassification error as loss
plot(cv_lasso)
```




```
cv_lasso$lambda.1se
```

```
## [1] 0.04892526
```

```
cv_lasso$lambda.min
```

```
## [1] 0.01459757
```

This plot shows us the number of nonzero coefficient estimates at the top. λ_{min} is equal to 0.015, whereas λ_{1se} equals to 0.049.

```
#model_lasso <- glmnet(X, y, family = "binomial", alpha = 1, lambda = cv_lasso$lambda.1se)
#do not use the above command --> does not give the exact same results as coef(cv_lasso, s = "lambda.1s
#glmnet should not be used with a single value of lambda (see R Documentation)
coef(cv_lasso, s = "lambda.1se") #our betas
```

```
## 10 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept) -3.40539920
## sbp          .
## tobacco      0.04271947
## ldl          0.08790170
## adiposity    .
## famhist      0.40141295
```

```
## typea        0.01053813
## obesity      .
## alcohol      .
## age          0.03269543
```

```
model_lasso <- glmnet(X_train, y_train, family = "binomial", alpha = 1)
lasso_pred <- predict(model_lasso, s = cv_lasso$lambda.1se, newx = X_test, type = "response")
#default option of type would give us probabilities on logit scale

#misclassification rate
y_values$Lasso <- ifelse(lasso_pred > 0.5, 1, 0)
misclassification[,"Lasso"] <- nrow(y_values[y_values$chd != y_values$Lasso,]) / nrow(y_values)
```

Next, we fot a boosted regression model with generalized additive effects (`gamboost` from package **mboost**.

```
train$chd <- as.factor(train$chd) #needs to be converted before estimating gamboost
model_boost <- gamboost(chd ~ sbp  + ldl + adiposity +
                          bols(famhist) + typea + obesity + alcohol +
                          age, data = train, family = Binomial())

boost_predict <- predict(model_boost, newdata = test, type = "response")
y_values$Gamboost <- ifelse(boost_predict > 0.5, 1, 0)
misclassification[,"Gamboost"] <- nrow(y_values[y_values$chd != y_values$Gamboost,]) / nrow(y_values)

misclassification
```

```
##      Stepwise_selection     Lasso  Gamboost
## [1,]          0.2688172 0.2795699 0.3010753
```

The misclassification rates are not too different, but the stepwise selection produces the best results, followed by the Lasso approach and the boosted logistic regression model.

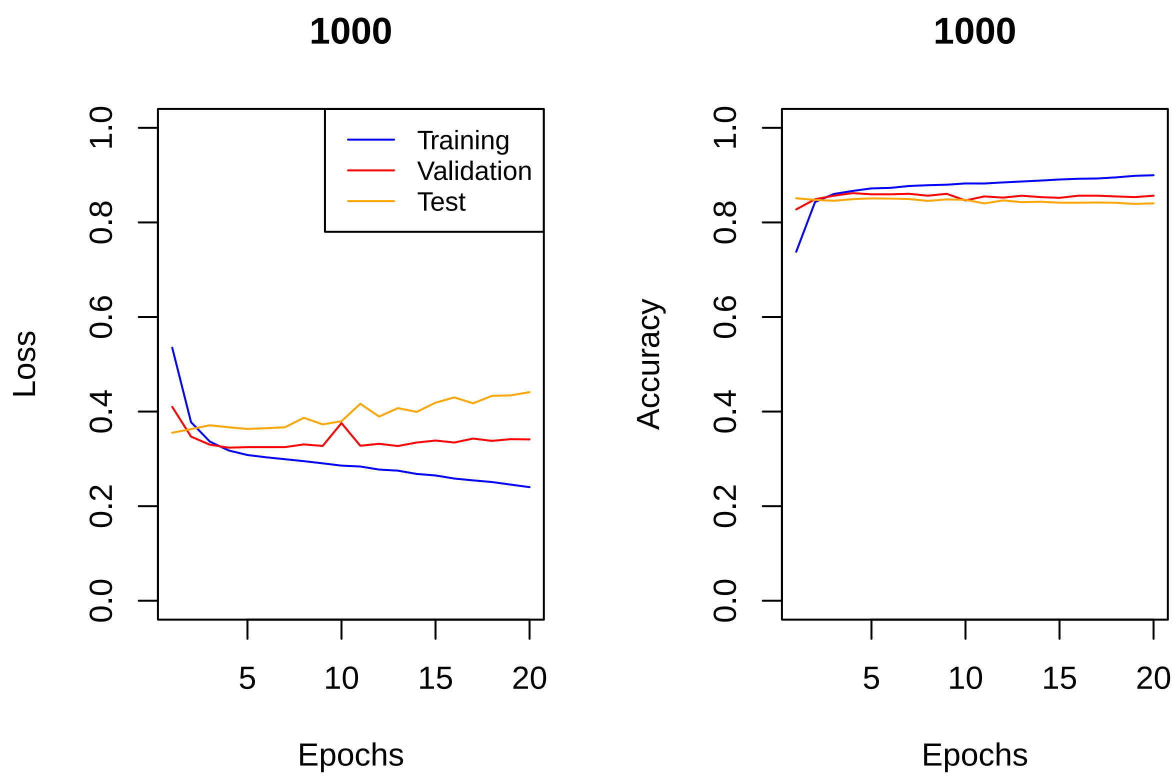
Exercise 4

We take the `IMDb` dataset from the **keras** package to perform document classification. We restrict the vocabulary to the most frequently-used words and tokens, starting with the 1,000 most frequent words. We stick to the code provided by James et al. (2021).

```
set.seed(123)
neural_networks <- function(max_features){
    imdb <- dataset_imdb(num_words = max_features)#limit to 1,000 most frequent words
#binary feature vector will score 1 for a match with the dictionary (dictionary is
#limited to 1,000 words), and 0 otherwise
    c(c(x_train, y_train), c(x_test, y_test)) %<-% imdb
#training and test data set are balanced with regard to sentiment and contain each
#25,000 observations
#each element i of x_train[[i]] is a vector of numbers between 0 and 999
#--> the locations of the nonzero entries (matches with dictionaries) are stored


#for estimating neural network:
#convert data from list to matrix
```

```r
    list_to_matrix <- function(list, dimension = max_features) {
  # create empty matrix with dimension needed
    results <- matrix(0, nrow = length(list), ncol = dimension)
  # Replace 0 with a 1 for each column of the matrix given in the list
    for (i in 1:length(list))
    results[i, list[[i]]] <- 1
    results
    }

    x_train <- list_to_matrix(x_train)
    x_test <- list_to_matrix(x_test)
#convert also y from integer to numeric
    y_train <- as.numeric(y_train)
    y_test <- as.numeric(y_test)


# In addition to the test data set, we use 2,000 observations of the training data set for the validati
# Then, we fit a fully-connected neural network with two hidden layers, each with 16 units and ReLu act

    ival <- sample(1:length(y_train), 2000, replace = FALSE)

    model <- keras_model_sequential() %>%
    layer_dense(units = 16, activation = "relu", input_shape = c(max_features)) %>%
    layer_dense(units = 16, activation = "relu") %>%
    layer_dense(units = 1, activation = "sigmoid")

    model %>% compile(
    optimizer = "rmsprop",
    loss = "binary_crossentropy", #because we have a binary classification problem
  #we use binary_crossentropy loss
    metrics = c("accuracy")
    )

    history <- model %>% fit(x_train[-ival,], y_train[-ival],
                         epochs = 20, batch_size = 512,
                         validation_data = list(x_train[ival,], y_train[ival]))

#at each epoch (=number of times the fitting algorithm passes through training set)
#training and validation accuracy and loss is computed

    loss_train <- history$metrics$loss
    loss_val <- history$metrics$val_loss
    accuracy_train <- history$metrics$accuracy
    accuracy_val <- history$metrics$val_accuracy

#compute also test accuracy:

    history <- model %>% fit(x_train[-ival,], y_train[-ival],
                         epochs = 20, batch_size = 512,
                         validation_data = list(x_test, y_test))

    loss_test <- history$metrics$val_loss
    accuracy_test <- history$metrics$val_accuracy
```

```
loss_plot <- plot(loss_train, type = "l", col = "blue", xlab = "Epochs", ylab = "Loss",
                  main = as.character(max_features), ylim=c(0,1))
lines(loss_val, type = "l", col = "red")
lines(loss_test, type = "l", col = "orange")
legend("topright", legend=c("Training", "Validation", "Test"),
       col=c("blue", "red", "orange"), lty = 1, cex=0.8)

accuracy_plot <- plot(accuracy_train, type = "l", col = "blue", xlab = "Epochs",
                      ylab = "Accuracy", main = as.character(max_features), ylim=c(0,1))
lines(accuracy_val, type = "l", col = "red")
lines(accuracy_test, type = "l", col = "orange")
# legend("topleft", legend=c("Training", "Validation", "Test"),
#         col=c("blue", "red", "orange"), lty = 1, cex=0.8)

return(list(loss_plot, accuracy_plot))
}
par(mfrow=c(1,2))
neural_networks(1000)
```



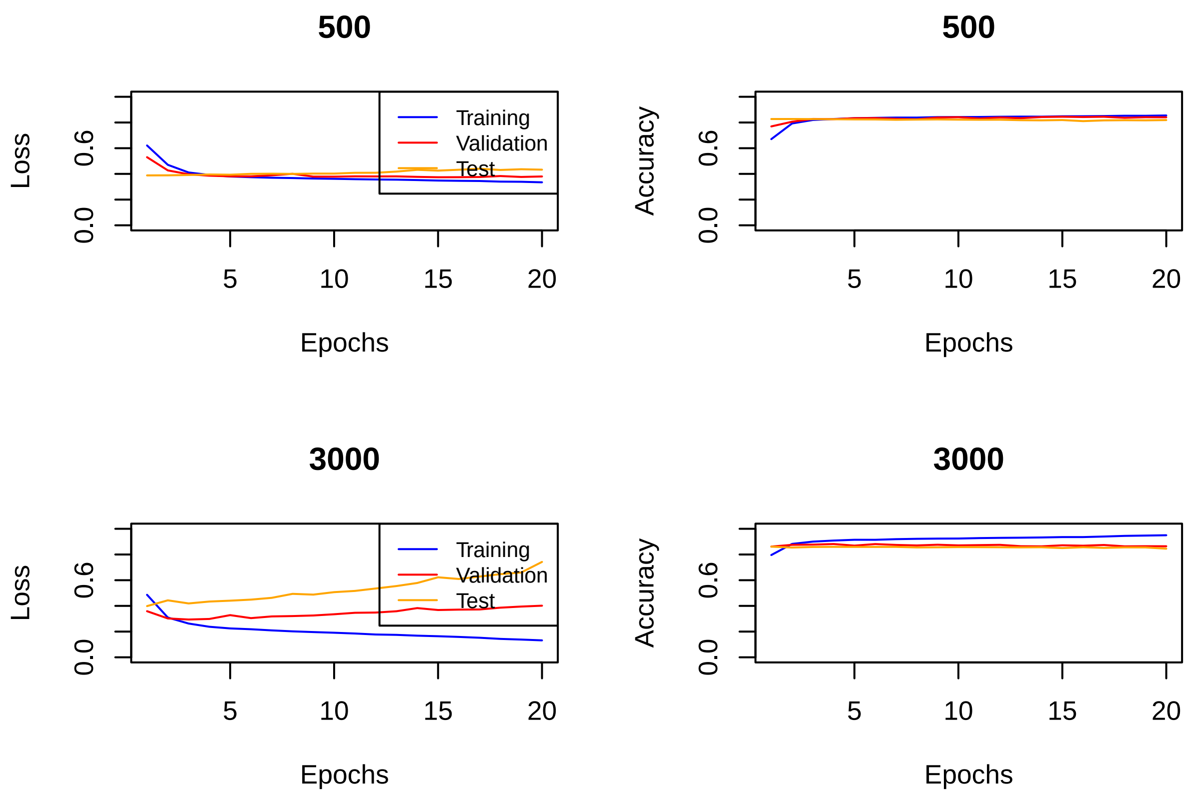

```
## [[1]]
## NULL
##
## [[2]]
## NULL
```

We then vary the dictionary size and try out values 500, 1000, 3000, 5000, 10,000.

```
par(mfrow=c(2,2))
neural_networks(500)
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
```

```
neural_networks(3000)
```



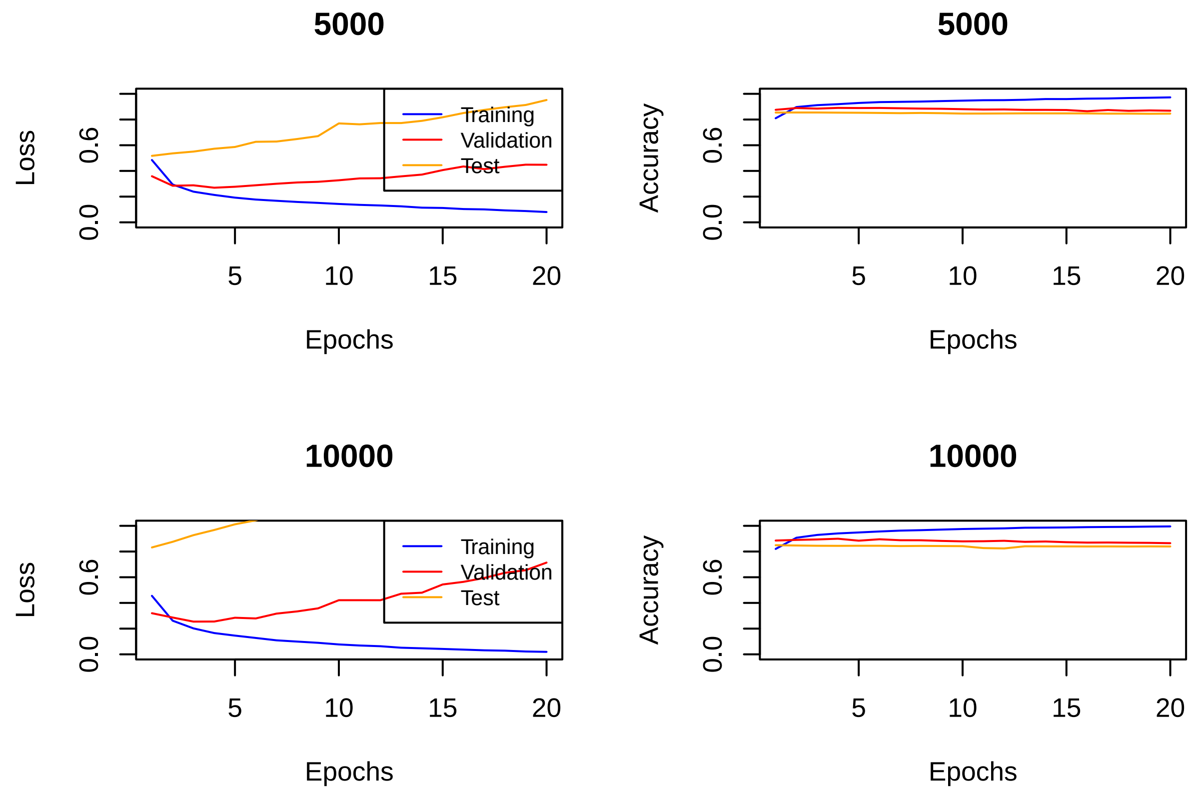

```
## [[1]]
## NULL
##
## [[2]]
## NULL
```

```
par(mfrow=c(2,2))
neural_networks(5000)
```

```
## [[1]]
## NULL
```

```
##
## [[2]]
## NULL
```

```
neural_networks(10000)
```




```
## [[1]]
## NULL
##
## [[2]]
## NULL
```

From looking at the plots, we see that the higher the dictionary size, 1) the more the loss varies between training, test and validation set; also, the loss for the test set gets higher (with more epochs); the loss for the training data set gets lower (with more epochs). 2) the higher the accuracy of the training set; the (slightly) higher the accuracy of the validation and test sets.

HW5_Theresa

Theresa Traxler

10 6 2022

Exercise 2

We generate data from the additive error model

$$Y = f(X_1, X_2) + \epsilon = \sigma(a_1^T X_1) + \sigma(a_2^T X_2) + \epsilon$$

with

$$a_1 = (3, 3), a_2 = (3, -3).$$

- Each $X_j, j = 1, 2$ is a standard Gaussian variate with $p = 2$.
- ϵ is an independent Gaussian error with variance chosen such that the signal-to-noise ratio as measured by the respective variances equals four.

The training set is of size 100 and the test sample of size 1000.

```
library(mvtnorm)
set.seed(123)

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

a1 <- c(3,3)
a2 <- c(3,-3)
generate <- function(a1,a2,size){
  data <- numeric()
  for (i in 1:size) {
    X <- rmvnorm(2, mean = c(0,0), sigma = matrix(c(1,0,0,1),ncol=2))
    epsilon <- rnorm(1, mean = 0, sd = 1/2)
    Y <- sigmoid(t(a1)%*%X[1,]) + sigmoid(t(a2)%*%X[2,]) + epsilon
    data <- (rbind(data,c(Y,X[1,],X[2,])))
    colnames(data) <- c("Y","X1_1","X1_2","X2_1","X2_2")
  }
  data
}
train <- generate(a1,a2,100)
test <- generate(a1,a2,10000)

library(torch)
library(luz) # high-level interface for torch


x_train <- torch_tensor(train[,-1])
y_train <- torch_tensor(train[,1])
```

```
x_test <- torch_tensor(test[,-1])
y_test <- torch_tensor(test[,1])
```

Now we fit neural networks with weight decay of 0.0005, vary the number of hidden units from 0 to 10 and record the average test error

$$E_{Test}(Y - \hat{f}(X_1, X_2))^2$$

for each of 10 random starting weights.

```
error_test <- list()

for (i in 1:10) {

#0 hidden layers
model <- nn_sequential(
  nn_linear(4, 1),
)
parameters <- model$parameters

optimizer <- optim_adam(parameters, weight_decay = 0.0005)

for (t in 1:100) {

  ### -------- Forward pass --------

  y_train_hat <- model(x_train)

  ### -------- compute loss --------
  loss <- nnf_mse_loss(y_train_hat, y_train, reduction = "sum")
  #if (t %% 10 == 0)
    #cat("Epoch: ", t, "   Loss: ", loss$item(), "\n")

  ### -------- Backpropagation --------

  # Still need to zero out the gradients before the backward pass, only this time,
  # on the optimizer object
  optimizer$zero_grad()

  # gradients are still computed on the loss tensor (no change here)
  loss$backward()

  ### -------- Update weights --------

  # use the optimizer to update model parameters
  optimizer$step()
}
y_test_hat <- model(x_test)
error_test[[i]] <- mean(as.numeric(y_test_hat-y_test))^2



#repeat everything with new models
```

```r
#1 hidden layer
model <- nn_sequential(
  nn_linear(4, 50),
  nn_relu(),
  nn_linear(50, 1)
)
parameters <- model$parameters

optimizer <- optim_adam(parameters, weight_decay = 0.0005)

for (t in 1:100) {
  y_train_hat <- model(x_train)
  loss <- nnf_mse_loss(y_train_hat, y_train, reduction = "sum")
  #if (t %% 10 == 0)
    #cat("Epoch: ", t, "   Loss: ", loss$item(), "\n")
  optimizer$zero_grad()
  loss$backward()
  optimizer$step()
}
y_test_hat <- model(x_test)
#collect MSEs
error_test[[i]] <- c(error_test[[i]],mean(as.numeric(y_test_hat-y_test))^2)


#2 hidden layers
model <- nn_sequential(
  nn_linear(4, 50),
  nn_relu(),
  nn_linear(50, 40),
  nn_relu(),
  nn_linear(40, 1)
)
parameters <- model$parameters

optimizer <- optim_adam(parameters, weight_decay = 0.0005)

for (t in 1:100) {
  y_train_hat <- model(x_train)
  loss <- nnf_mse_loss(y_train_hat, y_train, reduction = "sum")
  #if (t %% 10 == 0)
    #cat("Epoch: ", t, "   Loss: ", loss$item(), "\n")
  optimizer$zero_grad()
  loss$backward()
  optimizer$step()
}
y_test_hat <- model(x_test)
#collect MSEs
error_test[[i]] <- c(error_test[[i]],mean(as.numeric(y_test_hat-y_test))^2)


#3 hidden layers
model <- nn_sequential(
  nn_linear(4, 50),
```

```r
  nn_relu(),
  nn_linear(50, 40),
  nn_relu(),
  nn_linear(40, 30),
  nn_relu(),
  nn_linear(30, 1)
)
parameters <- model$parameters

optimizer <- optim_adam(parameters, weight_decay = 0.0005)

for (t in 1:100) {
  y_train_hat <- model(x_train)
  loss <- nnf_mse_loss(y_train_hat, y_train, reduction = "sum")
  #if (t %% 10 == 0)
    #cat("Epoch: ", t, "   Loss: ", loss$item(), "\n")
  optimizer$zero_grad()
  loss$backward()
  optimizer$step()
}
y_test_hat <- model(x_test)
error_test[[i]] <- c(error_test[[i]],mean(as.numeric(y_test_hat-y_test))^2)


#4 hidden layers
model <- nn_sequential(
  nn_linear(4, 50),
  nn_relu(),
  nn_linear(50, 40),
  nn_relu(),
  nn_linear(40, 30),
  nn_relu(),
  nn_linear(30, 20),
  nn_relu(),
  nn_linear(20, 1)
)
parameters <- model$parameters

optimizer <- optim_adam(parameters, weight_decay = 0.0005)

for (t in 1:100) {
  y_train_hat <- model(x_train)
  loss <- nnf_mse_loss(y_train_hat, y_train, reduction = "sum")
  #if (t %% 10 == 0)
    #cat("Epoch: ", t, "   Loss: ", loss$item(), "\n")
  optimizer$zero_grad()
  loss$backward()
  optimizer$step()
}
y_test_hat <- model(x_test)
error_test[[i]] <- c(error_test[[i]],mean(as.numeric(y_test_hat-y_test))^2)
```

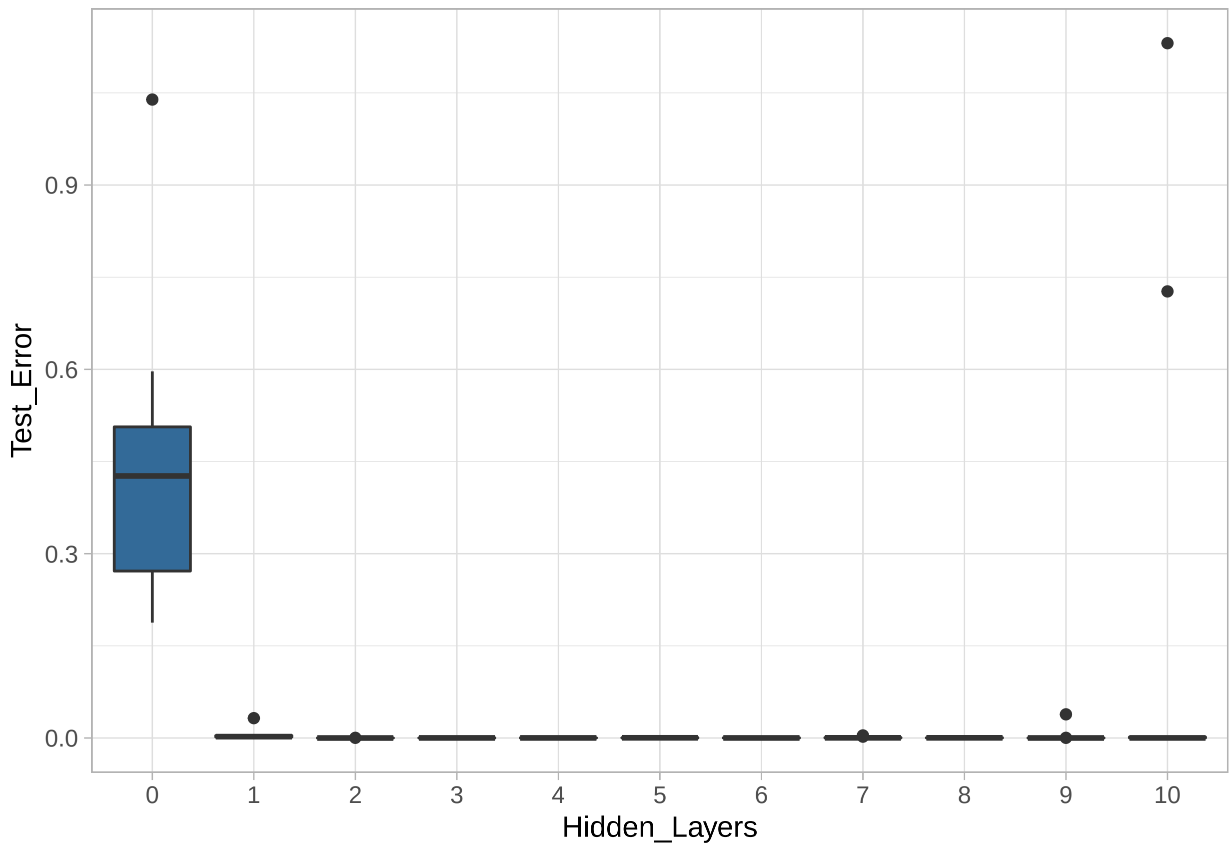
```r
#5 hidden layers
model <- nn_sequential(
  nn_linear(4, 50),
  nn_relu(),
  nn_linear(50, 40),
  nn_relu(),
  nn_linear(40, 30),
  nn_relu(),
  nn_linear(30, 20),
  nn_relu(),
  nn_linear(20, 10),
  nn_relu(),
  nn_linear(10, 1)
)
parameters <- model$parameters

optimizer <- optim_adam(parameters, weight_decay = 0.0005)

for (t in 1:100) {
  y_train_hat <- model(x_train)
  loss <- nnf_mse_loss(y_train_hat, y_train, reduction = "sum")
  #if (t %% 10 == 0)
    #cat("Epoch: ", t, "   Loss: ", loss$item(), "\n")
  optimizer$zero_grad()
  loss$backward()
  optimizer$step()
}
y_test_hat <- model(x_test)
error_test[[i]] <- c(error_test[[i]],mean(as.numeric(y_test_hat-y_test))^2)


#6 hidden layers
model <- nn_sequential(
  nn_linear(4, 50),
  nn_relu(),
  nn_linear(50, 45),
  nn_relu(),
  nn_linear(45, 40),
  nn_relu(),
  nn_linear(40, 30),
  nn_relu(),
  nn_linear(30, 20),
  nn_relu(),
  nn_linear(20, 10),
  nn_relu(),
  nn_linear(10, 1)
)
parameters <- model$parameters

optimizer <- optim_adam(parameters, weight_decay = 0.0005)

for (t in 1:100) {
  y_train_hat <- model(x_train)
```

```r
  loss <- nnf_mse_loss(y_train_hat, y_train, reduction = "sum")
  #if (t %% 10 == 0)
    #cat("Epoch: ", t, "   Loss: ", loss$item(), "\n")
  optimizer$zero_grad()
  loss$backward()
  optimizer$step()
}
y_test_hat <- model(x_test)
error_test[[i]] <- c(error_test[[i]],mean(as.numeric(y_test_hat-y_test))^2)


#7 hidden layers
model <- nn_sequential(
  nn_linear(4, 50),
  nn_relu(),
  nn_linear(50, 45),
  nn_relu(),
  nn_linear(45, 40),
  nn_relu(),
  nn_linear(40, 35),
  nn_relu(),
  nn_linear(35, 30),
  nn_relu(),
  nn_linear(30, 20),
  nn_relu(),
  nn_linear(20, 10),
  nn_relu(),
  nn_linear(10, 1)
)
parameters <- model$parameters

optimizer <- optim_adam(parameters, weight_decay = 0.0005)

for (t in 1:100) {
  y_train_hat <- model(x_train)
  loss <- nnf_mse_loss(y_train_hat, y_train, reduction = "sum")
  #if (t %% 10 == 0)
    #cat("Epoch: ", t, "   Loss: ", loss$item(), "\n")
  optimizer$zero_grad()
  loss$backward()
  optimizer$step()
}
y_test_hat <- model(x_test)
error_test[[i]] <- c(error_test[[i]],mean(as.numeric(y_test_hat-y_test))^2)


#8 hidden layers
model <- nn_sequential(
  nn_linear(4, 50),
  nn_relu(),
  nn_linear(50, 45),
  nn_relu(),
  nn_linear(45, 40),
```

```
  nn_relu(),
  nn_linear(40, 35),
  nn_relu(),
  nn_linear(35, 30),
  nn_relu(),
  nn_linear(30, 25),
  nn_relu(),
  nn_linear(25, 20),
  nn_relu(),
  nn_linear(20, 10),
  nn_relu(),
  nn_linear(10, 1)
)
parameters <- model$parameters

optimizer <- optim_adam(parameters, weight_decay = 0.0005)

for (t in 1:100) {
  y_train_hat <- model(x_train)
  loss <- nnf_mse_loss(y_train_hat, y_train, reduction = "sum")
  #if (t %% 10 == 0)
    #cat("Epoch: ", t, "   Loss: ", loss$item(), "\n")
  optimizer$zero_grad()
  loss$backward()
  optimizer$step()
}
y_test_hat <- model(x_test)
error_test[[i]] <- c(error_test[[i]],mean(as.numeric(y_test_hat-y_test))^2)


#9 hidden layers
model <- nn_sequential(
  nn_linear(4, 50),
  nn_relu(),
  nn_linear(50, 45),
  nn_relu(),
  nn_linear(45, 40),
  nn_relu(),
  nn_linear(40, 35),
  nn_relu(),
  nn_linear(35, 30),
  nn_relu(),
  nn_linear(30, 25),
  nn_relu(),
  nn_linear(25, 20),
  nn_relu(),
  nn_linear(20, 15),
  nn_relu(),
  nn_linear(15, 10),
  nn_relu(),
  nn_linear(10, 1)
)
parameters <- model$parameters
```

```r
optimizer <- optim_adam(parameters, weight_decay = 0.0005)

for (t in 1:100) {
  y_train_hat <- model(x_train)
  loss <- nnf_mse_loss(y_train_hat, y_train, reduction = "sum")
  #if (t %% 10 == 0)
    #cat("Epoch: ", t, "   Loss: ", loss$item(), "\n")
  optimizer$zero_grad()
  loss$backward()
  optimizer$step()
}
y_test_hat <- model(x_test)
error_test[[i]] <- c(error_test[[i]],mean(as.numeric(y_test_hat-y_test))^2)


#10 hidden layers
model <- nn_sequential(
  nn_linear(4, 50),
  nn_relu(),
  nn_linear(50, 45),
  nn_relu(),
  nn_linear(45, 40),
  nn_relu(),
  nn_linear(40, 35),
  nn_relu(),
  nn_linear(35, 30),
  nn_relu(),
  nn_linear(30, 25),
  nn_relu(),
  nn_linear(25, 20),
  nn_relu(),
  nn_linear(20, 15),
  nn_relu(),
  nn_linear(15, 10),
  nn_relu(),
  nn_linear(10, 5),
  nn_relu(),
  nn_linear(5, 1)
)
parameters <- model$parameters

optimizer <- optim_adam(parameters, weight_decay = 0.0005)

for (t in 1:100) {
  y_train_hat <- model(x_train)
  loss <- nnf_mse_loss(y_train_hat, y_train, reduction = "sum")
  #if (t %% 10 == 0)
    #cat("Epoch: ", t, "   Loss: ", loss$item(), "\n")
  optimizer$zero_grad()
  loss$backward()
  optimizer$step()
}
y_test_hat <- model(x_test)
```

```
error_test[[i]] <- c(error_test[[i]],mean(as.numeric(y_test_hat-y_test))^2)

}
test_error <- data.frame(matrix(unlist(error_test),nrow = 10, byrow = T))
test_error <- cbind(1:10,test_error)
colnames(test_error) <- c("ID",0:10)

#plot result
library(reshape2)
test_error_plot <- melt(test_error,id.vars="ID")
colnames(test_error_plot) <- c("ID","Hidden_Layers","Test_Error")

library(ggplot2)
ggplot(test_error_plot, aes(x = Hidden_Layers, y = Test_Error)) +
  geom_boxplot(aes(fill =2), show.legend = F) + theme_light()
```



We can see that there is a strong decrease in error already for one hidden layer, then the error stays rather constant on a low level. We can conclude that few hidden layers are sufficient in this context.

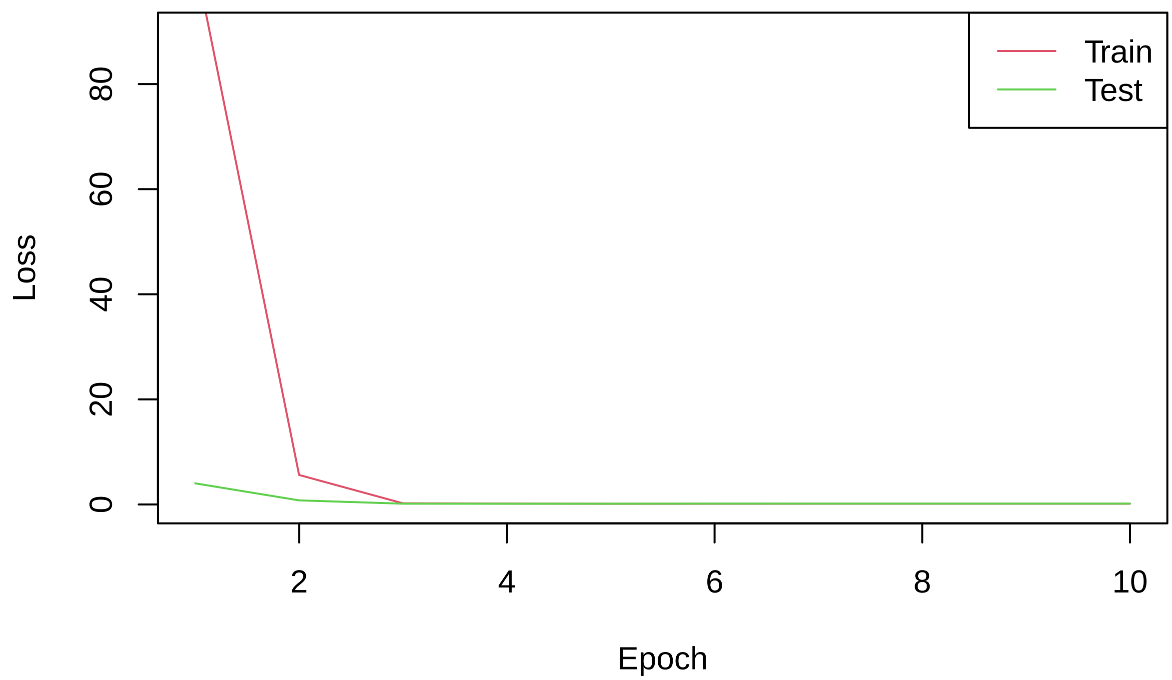
Exercise 3

In the following we will estimate a predictive model for the Default data from the `ISLR` package. \ We fit a neural network using a single hidden layer with 10 units and dropout regularization and compare the classification performance of this model with that of linear logistic regression.

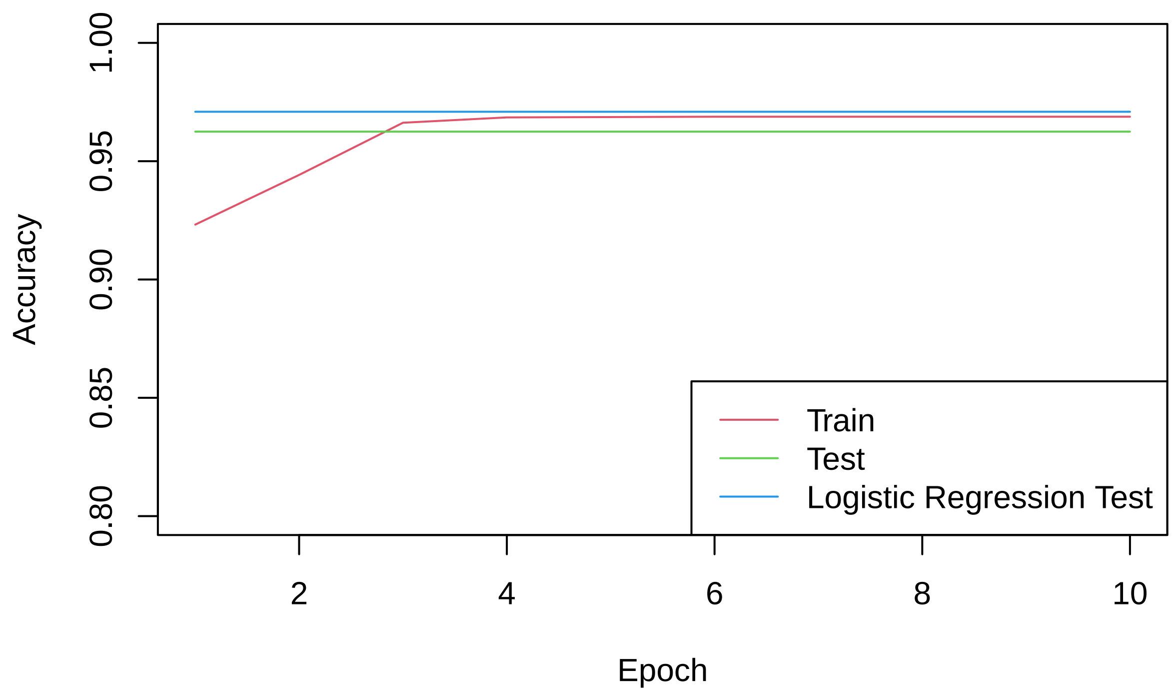
```
library(ISLR2)
n <- nrow(Default)
```

```r
set.seed(13)
ntest <- trunc(n/3)
testid <- sample(1:n, ntest)

logistic_fit <- glm(default ~ ., data = Default[-testid, ], family = binomial)
step_pred <- predict(logistic_fit, Default[testid, ], type = "response")
y_pred <- ifelse(step_pred > 0.5, "Yes", "No")
y_test <- Default[testid,]$default
accuracy_logistic <- mean(y_test == y_pred)


library(torch)
library(luz) # high-level interface for torch
torch_manual_seed(13)


Default_ <- as.matrix(cbind(as.numeric(factor(Default$default))-1,
                            as.numeric(factor(Default$student))-1,
                            Default[,c(3,4)]))
colnames(Default_) <- colnames(Default)

x <- Default_[,-1]
y <- Default_[,1]

modnn <- nn_module(
  initialize = function() {
    self$hidden <- nn_linear(ncol(x), 10)
    self$activation <- nn_relu()
    self$dropout <- nn_dropout(0.4)
    self$output <- nn_linear(10, 1)
  },
  forward = function(x) {
    x %>%
      self$hidden() %>%
      self$activation() %>%
      self$dropout() %>%
      self$output()
  }
)



modnn <- modnn %>%
  setup(
    loss = nn_bce_with_logits_loss(),
    optimizer = optim_rmsprop,
    metrics = list(luz_metric_binary_accuracy_with_logits())
  )



fitted <- modnn %>%
  fit(
```

```
    data = list(torch_tensor(x[-testid, ]), torch_tensor(matrix(y[-testid]))),
    valid_data = list(torch_tensor(x[testid, ]), torch_tensor(matrix(y[testid]))),
    epochs = 10
  )

metrics_train <- matrix(unlist(fitted$records$metrics$train),ncol=2, byrow=T)
metrics_test <- matrix(unlist(fitted$records$metrics$valid),ncol=2, byrow=T)


plot(metrics_train[,1],col=2, type="l", ylim = c(0,90), ylab = "Loss", xlab = "Epoch")
lines(metrics_test[,1],col=3, type = "l")
legend("topright",legend = c("Train","Test"),col=2:3,lty = 1)
```




```
plot(metrics_train[,2],col=2, type="l",ylim = c(0.8,1), ylab = "Accuracy", xlab = "Epoch")
lines(metrics_test[,2],col=3, type = "l")
lines(rep(accuracy_logistic,nrow(metrics_test)),col=4,type="l")
legend("bottomright",legend = c("Train","Test","Logistic Regression Test"),col=2:4,lty = 1)
```



We can see that after a certain training period the accuracy on the training data set is higher for the neural network than on the test data set. Intuitively, the pattern for the loss is the other way round. \ When comparing the accuracy of both model, we observe that it is actually higher for the logistic regression model (on test data).

Statistical_Learning_HW5_5_6

Exercise 5

The data sets `zip.train` and `zip.test` from package `ElemStatLearn` contain the information on the gray color values of the pixels on a 16×16 pixel image of hand-written digits.

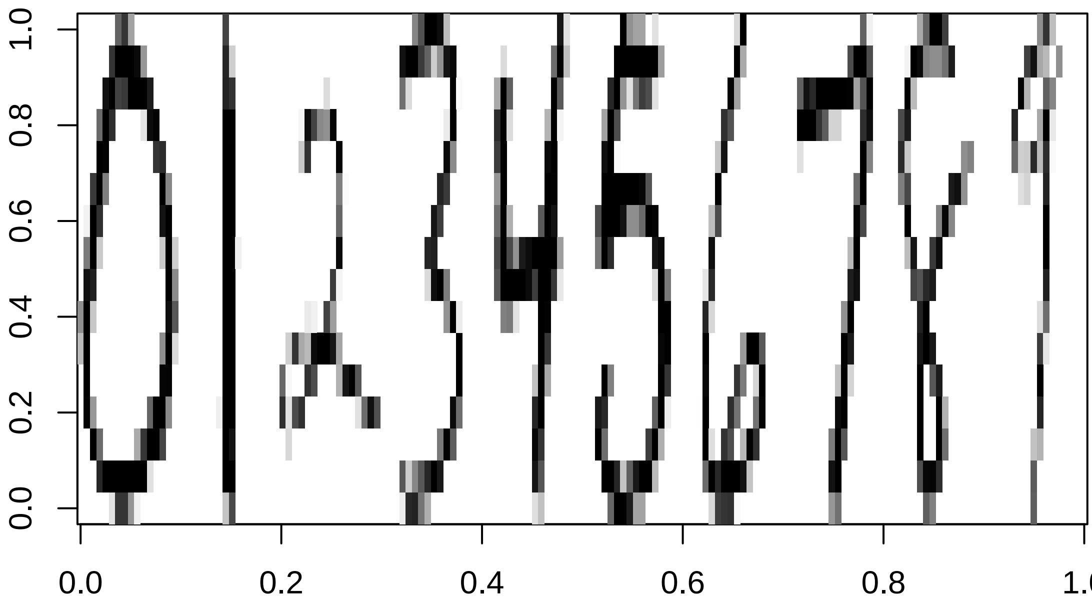
- Visualize for each digit one randomly selected observation (see `?zip.train`).
- Fit a multinomial logistic regression model to the training data and evaluate it on the training and the test data. Determine the overall missclassification rate on the training and the test data and the digit-specific missclassification rates on the test data. Which digits are the most difficult and the easiest to classify?
- Add a positive weight decay of 0.05 when fitting the multinomial logistic regression model to the training data and evaluate the model on the training and the test data. Determine the overall missclassification rate on the training and the test data. Explain why it makes sense to also include weight decay when fitting this multinomial logistic regression model.

```
library(nnet)
library(neuralnet)
#load train and test data
load(file='zip.train.RData')
load(file='zip.test.RData')

zip2image <-
function( zip, line ) {
     im <- zip[line, ]
     print(paste("digit ", im[1], " taken"))
     im <- im[-1]
     im <- t(matrix(im, 16, 16, byrow=TRUE))
     im <- im[, 16:1]
     return(im) }

findRows <- function(zip, n) {
 # Find  n (random) rows with zip representing 0,1,2,...,9
 res <- vector(length=10, mode="list")
 names(res) <- 0:9
 ind <- zip[,1]
 for (j in 0:9) {
    res[[j+1]] <- sample( which(ind==j), n ) }
 return(res) }

# Making the plot for exercise 1:

digits <- vector(length=10, mode="list")
names(digits) <- 0:9
rows <- findRows(zip.train, 1)
for (j in 0:9) {
    digits[[j+1]] <- do.call("cbind", lapply(as.list(rows[[j+1]]),
```

```
                          function(x) zip2image(zip.train, x)) )
}
```

```
## [1] "digit  0  taken"
## [1] "digit  1  taken"
## [1] "digit  2  taken"
## [1] "digit  3  taken"
## [1] "digit  4  taken"
## [1] "digit  5  taken"
## [1] "digit  6  taken"
## [1] "digit  7  taken"
## [1] "digit  8  taken"
## [1] "digit  9  taken"
```

```
im <- do.call("rbind", digits)
image(im, col=gray(256:0/256), zlim=c(0,1),xlab="", ylab="" )
```




```
#fit multinomial logistic regression model


#multinom documentation says that :"The variables on the rhs of the formula should be roughly scaled to
zip.train[,2:257] <- scale(zip.train[,2:257])
zip.train_df <- as.data.frame(zip.train)
zip.train_df$V1 <- as.factor(zip.train_df$V1)
zip.train_df$V1 <- relevel(zip.train_df$V1, ref = "0") #set reference category
zip.test[,2:257] <- scale(zip.test[,2:257])
zip.test_df <- as.data.frame(zip.test)
zip.test_df$V1 <- as.factor(zip.test_df$V1)
zip.test_df$V1 <- relevel(zip.test_df$V1, ref = "0") #set reference category
#the multinom function calls nnet with a shallow neural network (one layer) and a softmax readout map
#since for multinomial logistic regression we need a linear predictor with an additional softmax transf
multinom_model <- multinom(zip.train_df$V1 ~ .,data=zip.train_df,MaxNWts =10000)
```

```
## # weights:  2580 (2313 variable)
## initial  value 16788.147913
## iter  10 value 2184.178620
```

```
## iter  20 value 1111.258774
## iter  30 value 674.263153
## iter  40 value 506.070002
## iter  50 value 427.293596
## iter  60 value 314.210991
## iter  70 value 194.415629
## iter  80 value 150.625146
## iter  90 value 129.328969
## iter 100 value 106.622665
## final  value 106.622665
## stopped after 100 iterations
```

```r
#We compute the overall misclassification rate on the training and the test data and the digit-specific
sum(predict(multinom_model)!=zip.train[,1])/length(zip.train[,1])
```

```
## [1] 0.0001371554
```

```r
sum(predict(multinom_model,newdata=zip.test_df)!=zip.test[,1])/length(zip.test[,1])
```

```
## [1] 0.1270553
```

```r
#digit specific
misscl_index <- which(predict(multinom_model,newdata=zip.test_df)!=zip.test[,1])
misscl_per_digit <- zip.test[,1][misscl_index]
table(misscl_per_digit)/table(zip.test[,1])
```

```
## misscl_per_digit
##          0          1          2          3          4          5          6
## 0.08635097 0.04924242 0.20202020 0.13855422 0.18500000 0.21250000 0.10000000
##          7          8          9
## 0.08843537 0.21084337 0.06779661
```

```r
#it seems that 5,8 and 2 are quite hard to classify

#we add weight decay of 0.05
multinom_model2 <- multinom(zip.train_df$V1 ~ .,data=zip.train_df,MaxNWts =10000,decay=0.05)
```

```
## # weights:  2580 (2313 variable)
## initial  value 16788.147913
## iter  10 value 2188.720013
## iter  20 value 1124.235108
## iter  30 value 698.821327
## iter  40 value 537.608583
## iter  50 value 474.430515
## iter  60 value 353.614953
## iter  70 value 250.105214
## iter  80 value 214.805733
## iter  90 value 195.236834
## iter 100 value 176.693446
## final  value 176.693446
## stopped after 100 iterations
```

```r
#We compute the overall misclassification rate on the training and the test data and the digit-specific
sum(predict(multinom_model2)!=zip.train[,1])/length(zip.train[,1])
```

```
## [1] 0.0004114662
```

```
sum(predict(multinom_model2,newdata=zip.test_df)!=zip.test[,1])/length(zip.test[,1])
```

```
## [1] 0.1200797
```

```
#digit specific
misscl_index <- which(predict(multinom_model2,newdata=zip.test_df)!=zip.test[,1])
misscl_per_digit <- zip.test[,1][misscl_index]
table(misscl_per_digit)/table(zip.test[,1])
```

```
## misscl_per_digit
##          0          1          2          3          4          5          6
## 0.07520891 0.04924242 0.17676768 0.13855422 0.18000000 0.18750000 0.10000000
##          7          8          9
## 0.08843537 0.19879518 0.07909605
```

```
#the missclassification rate on the test data set was reduced
#weight decay acts as a form of regularization which makes a lot of sense since these kind of models te
```

Exercise 6

- Use only a subset from `zip.train` of size 320 observations with an equal number of observations for each digit to fit a multinomial logistic regression model and a neural network.
- Use all remaining training observations and the test data set to evaluate the fitted models.
- For this small training data overfitting is an issue. Visualize the performance on the test data in dependence of the training epochs when fitting the models.

```
digits <- 0:9
index_list <- c()
for(i in digits){
  index_list <- append(index_list,sample(which(zip.train[,1]==i),32))
}

train_new <- zip.train[index_list,]
train_new[,2:257] <- scale(train_new[,2:257])
train_new_df <- as.data.frame(train_new)
train_new_df$V1 <- as.factor(train_new_df$V1)
train_new_df$V1 <- relevel(train_new_df$V1, ref = "0") #set reference category
zip.test_df <- rbind(zip.test_df,zip.train_df[-index_list,])
zip.test <- rbind(zip.test,zip.train[-index_list,])

plot_list_multinom <- list()
plot_list_nn <- list()
k <- 1
for(j in c(100,250,500,1000,10000)){
multinom_model3 <- multinom(train_new_df$V1 ~ .,data=train_new_df,MaxNWts =10000,maxit=j)
#it is assumed that with 'neural network' a deep neural network is meant
nn <- neuralnet(train_new_df$V1 ~ .,data=train_new_df,hidden=10,linear.output=FALSE,lifesign="full",err
#We compute the digit-specific missclassification rates on the test data

misscl_index <- which(predict(multinom_model3,newdata=zip.test_df)!=zip.test[,1])
misscl_per_digit <- zip.test[,1][misscl_index]
pred_nn <- apply(predict(nn,newdata=zip.test_df),1,which.max)-1
misscl_index_nn <- which(pred_nn!=zip.test[,1])
misscl_per_digit_nn <- zip.test[,1][misscl_index_nn]
```

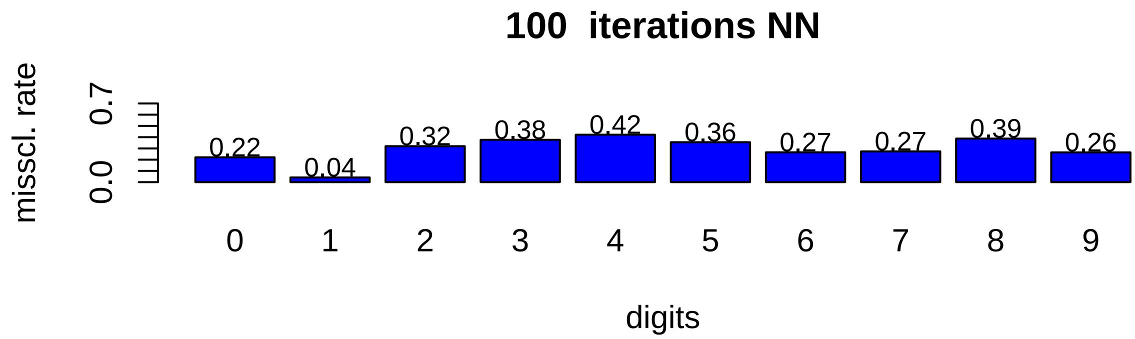
```
par(mfrow=c(2,1))

x <- barplot(table(misscl_per_digit_nn)/table(zip.test[,1]),ylab="misscl. rate",xlab="digits",main=paste
text(x,table(misscl_per_digit_nn)/table(zip.test[,1])+0.09,labels=as.character(round(table(misscl_per_di

x <- barplot(table(misscl_per_digit)/table(zip.test[,1]),ylab="misscl. rate",xlab="digits",main=paste(j
text(x,table(misscl_per_digit)/table(zip.test[,1])+0.09,labels=as.character(round(table(misscl_per_digi
plot_list_multinom[[k]]<-table(misscl_per_digit)/table(zip.test[,1])
plot_list_nn[[k]]<-table(misscl_per_digit_nn)/table(zip.test[,1])

k <- k+1
}
```

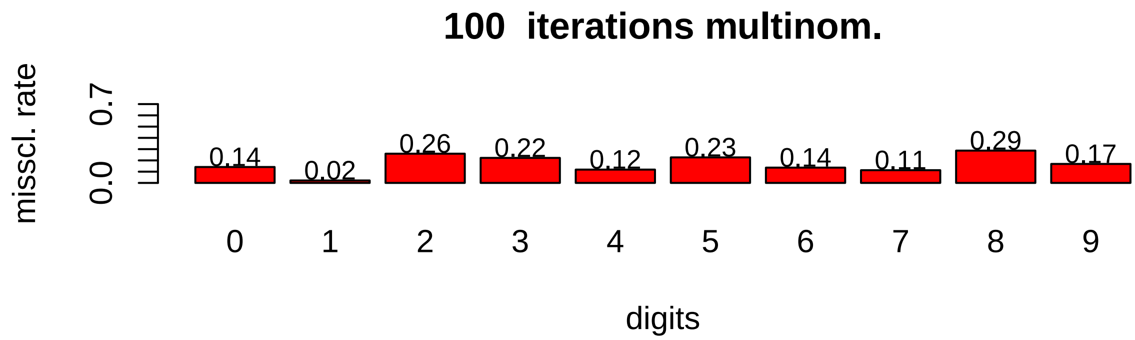
```
## # weights:  2580 (2313 variable)
## initial  value 736.827230
## iter  10 value 5.713331
## iter  20 value 0.590314
## iter  30 value 0.031703
## iter  40 value 0.009937
## iter  50 value 0.005003
## iter  60 value 0.003052
## iter  70 value 0.001980
## iter  80 value 0.001423
## iter  90 value 0.001011
## iter 100 value 0.000712
## final  value 0.000712
## stopped after 100 iterations
```

```
## hidden: 10    thresh: 0.01    rep: 1/1    steps:     546 error: 0.21177  time: 0.66 secs
```

```
## # weights:  2580 (2313 variable)
## initial  value 736.827230
## iter  10 value 5.713331
## iter  20 value 0.590314
## iter  30 value 0.031703
## iter  40 value 0.009937
## iter  50 value 0.005003
## iter  60 value 0.003052
## iter  70 value 0.001980
## iter  80 value 0.001423
## iter  90 value 0.001011
## iter 100 value 0.000712
## iter 110 value 0.000556
## iter 120 value 0.000368
## iter 130 value 0.000336
## iter 140 value 0.000259
## iter 150 value 0.000232
## iter 160 value 0.000194
## iter 170 value 0.000148
## iter 180 value 0.000142
## final  value 0.000100
## converged
```

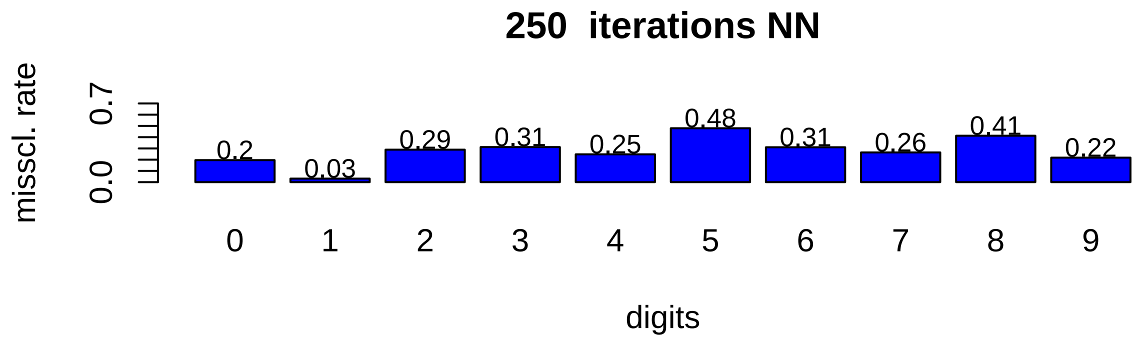
```
## hidden: 10    thresh: 0.01    rep: 1/1    steps:     239 error: 0.2599   time: 0.27 secs
```

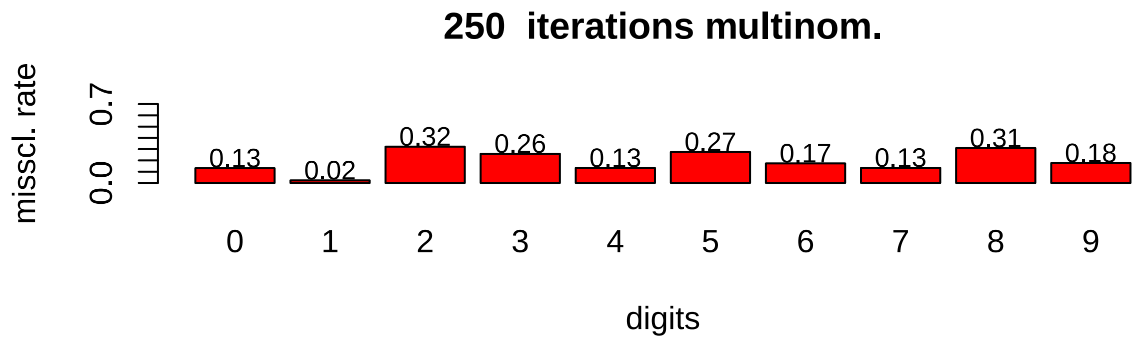





```
## # weights:  2580 (2313 variable)
## initial  value 736.827230
## iter  10 value 5.713331
## iter  20 value 0.590314
## iter  30 value 0.031703
## iter  40 value 0.009937
## iter  50 value 0.005003
## iter  60 value 0.003052
## iter  70 value 0.001980
## iter  80 value 0.001423
## iter  90 value 0.001011
## iter 100 value 0.000712
## iter 110 value 0.000556
## iter 120 value 0.000368
## iter 130 value 0.000336
## iter 140 value 0.000259
## iter 150 value 0.000232
## iter 160 value 0.000194
## iter 170 value 0.000148
## iter 180 value 0.000142
## final  value 0.000100
## converged
```

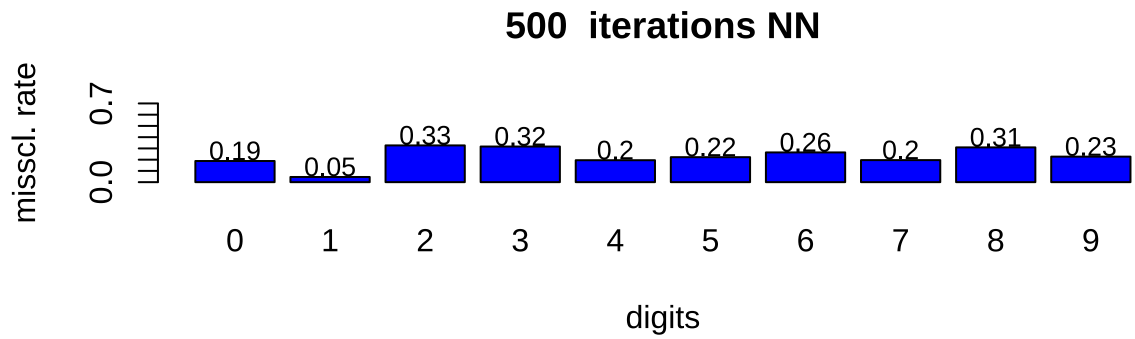
```
## hidden: 10    thresh: 0.01    rep: 1/1    steps:     242 error: 0.12254  time: 0.28 secs
```

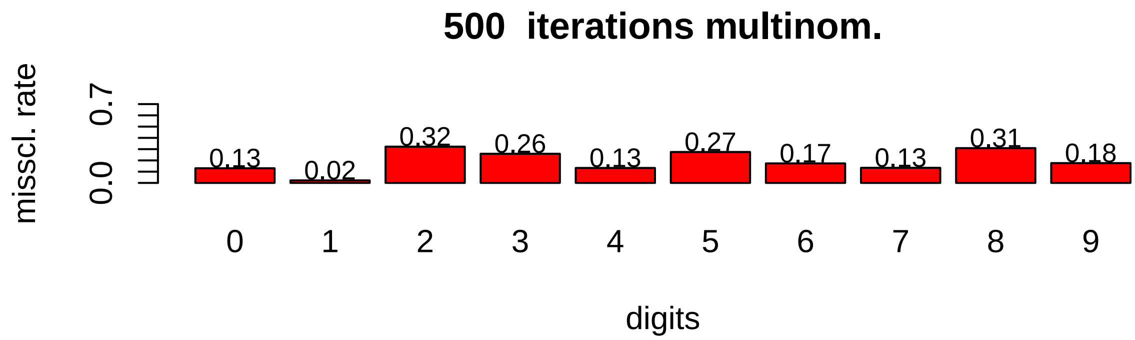





```
## # weights:  2580 (2313 variable)
## initial  value 736.827230
## iter  10 value 5.713331
## iter  20 value 0.590314
## iter  30 value 0.031703
## iter  40 value 0.009937
## iter  50 value 0.005003
## iter  60 value 0.003052
## iter  70 value 0.001980
## iter  80 value 0.001423
## iter  90 value 0.001011
## iter 100 value 0.000712
## iter 110 value 0.000556
## iter 120 value 0.000368
## iter 130 value 0.000336
## iter 140 value 0.000259
## iter 150 value 0.000232
## iter 160 value 0.000194
## iter 170 value 0.000148
## iter 180 value 0.000142
## final  value 0.000100
## converged
```

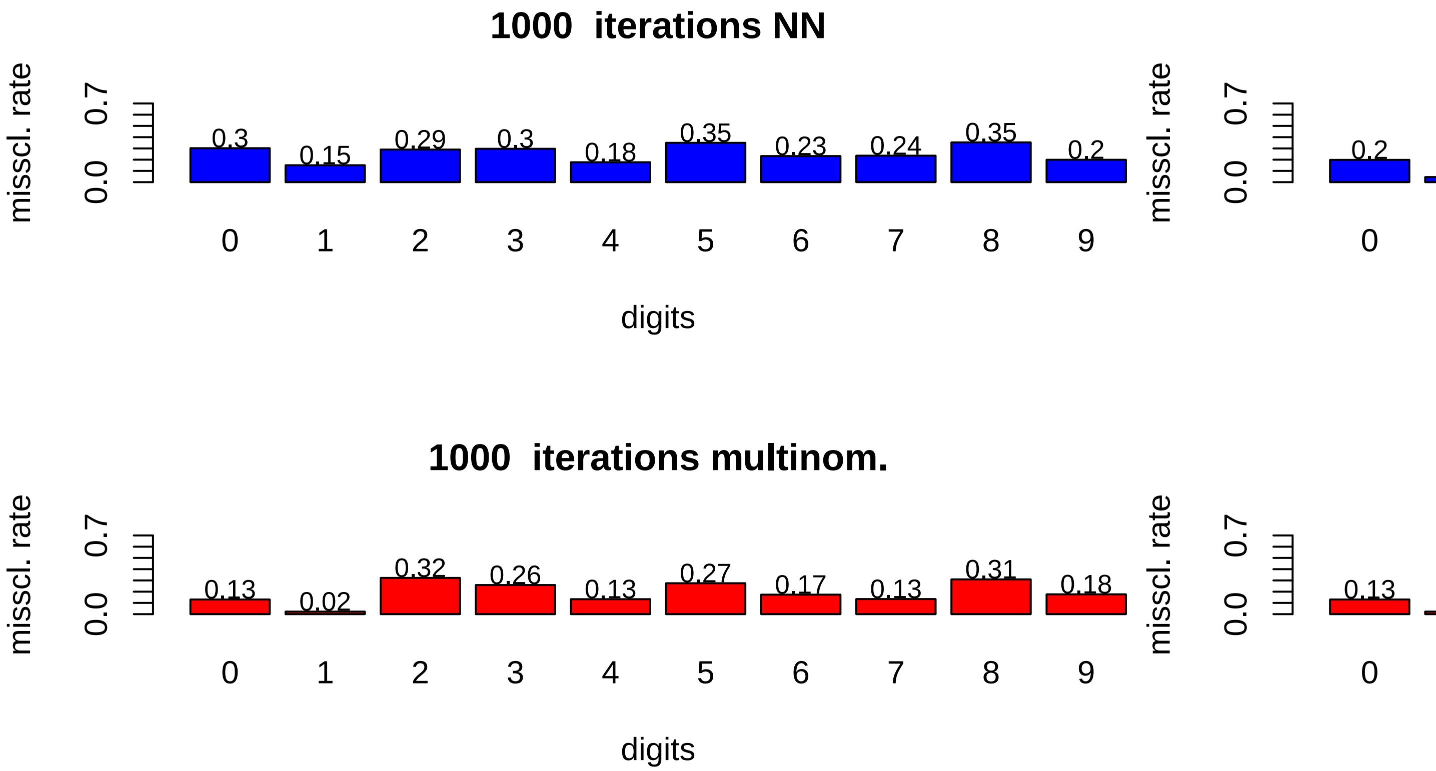
```
## hidden: 10    thresh: 0.01    rep: 1/1    steps:     376 error: 0.06056  time: 0.44 secs
```






```
## # weights:  2580 (2313 variable)
## initial  value 736.827230
## iter  10 value 5.713331
## iter  20 value 0.590314
## iter  30 value 0.031703
## iter  40 value 0.009937
## iter  50 value 0.005003
## iter  60 value 0.003052
## iter  70 value 0.001980
## iter  80 value 0.001423
## iter  90 value 0.001011
## iter 100 value 0.000712
## iter 110 value 0.000556
## iter 120 value 0.000368
## iter 130 value 0.000336
## iter 140 value 0.000259
## iter 150 value 0.000232
## iter 160 value 0.000194
## iter 170 value 0.000148
## iter 180 value 0.000142
## final  value 0.000100
## converged
```

```
## hidden: 10    thresh: 0.01    rep: 1/1    steps:     265 error: 0.09646  time: 0.31 secs
```


1000 iterations NN
missclassification rate
0.0
0.7
0.3
0.15
0.29
0.3
0.18
0.35
0.23
0.24
0.35
0.2
0 1 2 3 4 5 6 7 8 9
digits
missclassification rate
0.0
0.7
0.2
0
1000 iterations multinom.
missclassification rate
0.0
0.7
0.13
0.02
0.32
0.26
0.13
0.27
0.17
0.13
0.31
0.18
0 1 2 3 4 5 6 7 8 9
digits
missclassification rate
0.0
0.7
0.13
0