Deep Learning

Indrajith Wasala Mudiyanselage

Section 1

Question 01

a) Since the dataset is large, fitting multinational logistic regression (MLR) using the "multinom" function is difficult. Therefore I used shallow learning to fit MLR. For this model, the training error is 0.2502114 and the validation error is 0.2799808.

b) The model with two hidden layers is the best of the models considered.

Model	Training error	Validation error
NNM with 1 hidden layer	0.05511252	0.1085238
NNM with 2 hidden layers	0.0254432	0.09725994
NNM with 3 hidden layers	0.03340233	0.1080917

Table 1: Tabular summary of errors

c) Among the used grid of parameters the best regularization parameter is 0.0001. The corresponding validation error for this parameter is 0.1265614. Therefore, the regularization does not help ($0.1265614 > 0.09725994$).

d) Among the used parameters the best dropout rate is 0.25. The corresponding validation error for this parameter is 0.1117960. Therefore, the dropout rate does not help.

e) Since the validation errors in parts c and d do not improve, I recommend the best model in part b (the basic model with two hidden layers). The test error is 0.1406495 (tanning error= 0.0254432 and validation error= 0.09725994).

f) To achieve the best results, increasing the power of a neural network requires a combination of techniques, experimentation, and tuning. For example, increase the network size, use a larger training dataset, use better optimization algorithms, and so on.

Question 02

a) The training error is 3.308168 and the validation error is 3.517034.

b) The model with one hidden layers is the best of the models considered.

Model	Training error	Validation error
NNM with 1 hidden layer	1.720874	2.360861
NNM with 2 hidden layers	1.551988	2.415809

Table 2: Tabular summary of errors

c) Among the used grid of parameters the best regularization parameter is 0.0005. The corresponding validation error for this parameter is 2.118346. Therefore, the regularization improves the model ($2.118346 < 2.360861$).

d) Among the used parameters the best dropout rate is 0.25. The corresponding validation error for this parameter is 2.221878. Therefore, the dropout rate improves the model (But not like regularization).

e) I recommend the model in part c (the model with one hidden layers and regularization parameter 0.0005). The test error is 2.815951 (tanning error= 1.924338 and validation error= 2.118346).

f) To achieve the best results, increasing the power of a neural network requires a combination of techniques, experimentation, and tuning. For example, increase the network size, use a larger training dataset, use better optimization algorithms, and so on.

Section 2 (R codes)

```
knitr::opts_chunk$set(echo = TRUE}

## ----include=FALSE-------------------------------------------------------
## Question 01

library(keras)

mnist <- dataset_mnist()
train_images <- mnist$train$x
train_labels <- mnist$train$y
test_images <- mnist$test$x
test_labels <- mnist$test$y


### Preprocess so that the training and test data for features are
### in matrix form (as needed for model fitting)

train_images <- array_reshape(train_images, c(60000, 28*28)) # matrix
train_images <- train_images/255 # ensures all values are in [0, 1]
test_images <- array_reshape(test_images, c(10000, 28*28))
test_images <- test_images/255

### Make the response categorical

train_labels <- to_categorical(train_labels)
test_labels <- to_categorical(test_labels)


# Validation data split (80-20)
set.seed(6390)
rand<-sample(1:60000,size = 12000) #  60000*0.2=12000

validation_images<-train_images[rand,]
validation_lables<-train_labels[rand,]

new_train_images<-train_images[-rand,]
new_train_labels<-train_labels[-rand,]



## ----include=FALSE--------------------------------------------
## # a)
### set up the network architecture

network <- keras_model_sequential() %>%
  layer_dense(units = 10, activation = "softmax", input_shape = c(28*28))


### Compile the model

network %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",  # loss function to minimize
  metrics = c("accuracy") # monitor classification accuracy
)

network %>% fit(new_train_images, new_train_labels, epochs = 15,
                batch_size = 128,validation_data = list(validation_images,validation_lables))

### Examine traning and validation performance
metrics_train <- network %>% evaluate(new_train_images, new_train_labels)
```

```r
metrics_val <- network %>% evaluate(validation_images,validation_lables)



## ----include=FALSE-------------------------------------------
## # b)
##
### set up the network architectures

# Model with one hidden layer
network_b1 <-keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu", input_shape = c(28*28)) %>%
  layer_dense(units = 10, activation = "softmax")

# Model with two hidden layers
network_b2 <-keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu", input_shape = c(28*28)) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")

# Model with three hidden layers
network_b3 <-keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu", input_shape = c(28*28)) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dense(units = 256, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")

### Compile the models
network_b1 %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",  # loss function to minimize
  metrics = c("accuracy") # monitor classification accuracy
)

network_b2 %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",  # loss function to minimize
  metrics = c("accuracy") # monitor classification accuracy
)

network_b3 %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",  # loss function to minimize
  metrics = c("accuracy") # monitor classification accuracy
)

### Fit the models

network_b1 %>% fit(new_train_images, new_train_labels, epochs = 10,
                   batch_size = 128,validation_data = list(validation_images,validation_lables))
network_b2 %>% fit(new_train_images, new_train_labels, epochs = 10,
                   batch_size = 128,validation_data = list(validation_images,validation_lables))
network_b3 %>% fit(new_train_images, new_train_labels, epochs = 10,
                   batch_size = 128,validation_data = list(validation_images,validation_lables))
### Examine training and validation performance
metrics_train_b1 <- network_b1 %>% evaluate(new_train_images, new_train_labels)
metrics_val_b1 <- network_b1 %>% evaluate(validation_images, validation_lables)


metrics_train_b2 <- network_b2 %>% evaluate(new_train_images, new_train_labels)
metrics_val_b2 <- network_b2 %>% evaluate(validation_images, validation_lables)
```

```r
metrics_train_b3 <- network_b3 %>% evaluate(train_images, train_labels)
metrics_val_b3 <- network_b3 %>% evaluate(validation_images, validation_lables)



## ----include=FALSE------------------------------------------
## # c)
##
# Add L2 weight regularization to the best model

para_grid<-c(0.0001,0.0005,0.001)
loss_c<-c()

for (i in 1:length(para_grid)) {

  best_model_reg <- keras_model_sequential() %>%
    layer_dense(units = 64, activation = "relu", input_shape = c(28*28),
                kernel_regularizer = regularizer_l2(para_grid[i])) %>%
    layer_dense(units = 128, activation = "relu",
                kernel_regularizer = regularizer_l2(para_grid[i])) %>%
    layer_dense(units = 10, activation = "softmax")

  best_model_reg %>% compile(
    optimizer = "rmsprop",
    loss = "categorical_crossentropy",  # loss function to minimize
    metrics = c("accuracy") # monitor classification accuracy
  )

  best_model_reg %>% fit(new_train_images, new_train_labels, epochs = 10,
                       batch_size = 128,
                       validation_data=list(validation_images,validation_lables))
  metrics_val_best <- best_model_reg %>% evaluate(validation_images,validation_lables)
  loss_c[i]<- metrics_val_best[1]

}


## ----include=FALSE------------------------------------------
## # d)
drop_grid<-c(0.25,0.50)
loss_d<-c()

for (i in 1:length(drop_grid)) {

  best_model_reg <- keras_model_sequential() %>%
    layer_dense(units = 64, activation = "relu", input_shape = c(28*28)) %>%
    layer_dropout(rate = drop_grid[i]) %>%
    layer_dense(units = 128, activation = "relu") %>%
    layer_dropout(rate = drop_grid[i]) %>%
    layer_dense(units = 10, activation = "softmax")


  best_model_reg %>% compile(
    optimizer = "rmsprop",
    loss = "categorical_crossentropy",  # loss function to minimize
    metrics = c("accuracy") # monitor classification accuracy
  )

  best_model_reg %>% fit(new_train_images, new_train_labels, epochs = 15,
                          batch_size = 128,validation_data = list(validation_images,validation_lables))

  metrics_val_best <- best_model_reg %>% evaluate(validation_images,validation_lables)
```

```r
  loss_d[i]<- metrics_val_best[1]

}


## ----include=FALSE-----------------------------------------------
## # e)
## # Calculate test error

network_b2 %>% fit(train_images, train_labels, epochs = 10, batch_size = 128)

metrics_test_b2 <- network_b2 %>% evaluate(test_images, test_labels)

##############################################################################
## ----include=FALSE------------------------------------------------------------
## Question 02

library(keras)
boston <- dataset_boston_housing()
c(c(train_data, train_targets), c(test_data, test_targets)) %<-% boston

### Preprocess the data

### Standardize the training and test features

mean <- apply(train_data, 2, mean)
std <- apply(train_data, 2, sd)
train_data <- scale(train_data, center = mean, scale = std)
test_data <- scale(test_data, center = mean, scale = std)



## ----include=FALSE------------------------------------------------------------
## # a)

library(glmnet)
new_train <- unlist(train_data)
new_train <- matrix(new_train, nrow = 404, ncol = 13)
new_target <- as.vector(unlist(train_targets))
cvfit <- cv.glmnet(new_train, new_target,
    type.measure = "mae", nfolds = 4)

trn_pred <- predict(cvfit, newx = new_train, s = "lambda.min")

# Calculate training and cross-validation errors
train_error <- mean(abs(new_target - trn_pred))
cv_error <- cvfit$cvm[which.min(cvfit$cvm)]



## ----include=FALSE-----------------------------------------------
## # b)

### set up the network architectures

# Model with one hidden layer
Q2network_b1 <-keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu", input_shape = c(13)) %>%
  layer_dense(units = 1, activation = "linear")

# Model with two hidden layers
Q2network_b2 <-keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu", input_shape = c(13)) %>%
```

```r
  layer_dense(units = 128, activation = "relu") %>%
  layer_dense(units = 1, activation = "linear")


### Compile the models
Q2network_b1 %>% compile(
  optimizer = "rmsprop",
  loss = "mse",    # loss function to minimize
  metrics = c("mae")
)

Q2network_b2 %>% compile(
  optimizer = "rmsprop",
  loss = "mse",    # loss function to minimize
  metrics = c("mae")
)

  k<-4
  set.seed(1)
  indices <- sample(1:nrow(train_data),replace = FALSE)
  folds <- cut(indices, breaks = k, labels = FALSE)

  Q2b1_train<-c()
  Q2b1_val<-c()

  Q2b2_train<-c()
  Q2b2_val<-c()

  for (i in 1:k) {

    val_ind<- which(folds == i, arr.ind = TRUE)
    val_x<-train_data[val_ind,]
    train_x<-train_data[-val_ind,]

    train_y<-train_targets[-val_ind]
    val_y<-train_targets[val_ind]

    Q2network_b1 %>% fit(train_x, train_y, epochs = 100,
                         batch_size = 16)
    Q2network_b2 %>% fit(train_x, train_y, epochs = 100,
                         batch_size = 16)

    ### Examine training and validation performance
    metrics_train_Q2b1 <- Q2network_b1 %>% evaluate(train_x, train_y)
    metrics_val_Q2b1 <- Q2network_b1 %>% evaluate(val_x, val_y)
    Q2b1_train[i]<-metrics_train_Q2b1[2]
    Q2b1_val[i]<-metrics_val_Q2b1[2]

    metrics_train_Q2b2 <- Q2network_b2 %>% evaluate(train_x, train_y)
    metrics_val_Q2b2 <- Q2network_b2 %>% evaluate(val_x, val_y)
    Q2b2_train[i]<-metrics_train_Q2b2[2]
    Q2b2_val[i]<-metrics_val_Q2b2[2]

  }

# Model 1 training error and validation error
error_Q2b1_train<- mean(Q2b1_train)
error_Q2b1_val<-mean(Q2b1_val)

# Model 2 training error and validation error
error_Q2b2_train<- mean(Q2b2_train)
error_Q2b2_val<-mean(Q2b2_val)
```

```r
## ----include=FALSE------------------------------------------
## # c)

# Add L2 weight regularization to the best model

para_grid<-c(0.0001,0.0005,0.001)
k<-4
  set.seed(1)
  indices <- sample(1:nrow(train_data),replace = FALSE)
  folds <- cut(indices, breaks = k, labels = FALSE)

  Q3_train<-c()
  Q3_val<-c()

  l1_erorr<-c()



for (j in 1:length(para_grid)) {
  # Model with one hidden layer
Q3network <-keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu",
              input_shape = c(13),kernel_regularizer = regularizer_l2(para_grid[j])) %>%
  layer_dense(units = 1, activation = "linear")

Q3network %>% compile(
  optimizer = "rmsprop",
  loss = "mse",    # loss function to minimize
  metrics = c("mae")
)

 for (i in 1:k) {

    val_ind<- which(folds == i, arr.ind = TRUE)
    val_x<-train_data[val_ind,]
    train_x<-train_data[-val_ind,]

    train_y<-train_targets[-val_ind]
    val_y<-train_targets[val_ind]

    Q3network %>% fit(train_x, train_y, epochs = 100,
                      batch_size = 16)

    ### Examine training and validation performance
    metrics_train_Q3 <- Q3network %>% evaluate(train_x, train_y)
    metrics_val_Q3 <- Q3network %>% evaluate(val_x, val_y)
    Q3_train[i]<-metrics_train_Q3[2]
    Q3_val[i]<-metrics_val_Q3[2]


  }
l1_erorr[j]<-mean(Q3_val)

}



## ----include=FALSE------------------------------------------
## # d)

drop_grid<-c(0.25,0.50)
k<-4
```

```r
  set.seed(1)
  indices <- sample(1:nrow(train_data),replace = FALSE)
  folds <- cut(indices, breaks = k, labels = FALSE)

  Q2d_train<-c()
  Q2d_val<-c()

  DO_erorr<-c()



for (j in 1:length(drop_grid)) {
  # Model with one hidden layer
  Q2d_network <-keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu", input_shape = c(13)) %>%
  layer_dropout(rate = drop_grid[j]) %>%
  layer_dense(units = 1, activation = "linear")

  Q2d_network %>% compile(
  optimizer = "rmsprop",
  loss = "mse",    # loss function to minimize
  metrics = c("mae")
)

 for (i in 1:k) {

    val_ind<- which(folds == i, arr.ind = TRUE)
    val_x<-train_data[val_ind,]
    train_x<-train_data[-val_ind,]

    train_y<-train_targets[-val_ind]
    val_y<-train_targets[val_ind]

    Q2d_network %>% fit(train_x, train_y, epochs = 100,
                       batch_size = 16)

    ### Examine training and validation performance
    metrics_train_Q2d <- Q2d_network %>% evaluate(train_x, train_y)
    metrics_val_Q2d <- Q2d_network %>% evaluate(val_x, val_y)
    Q2d_train[i]<-metrics_train_Q2d[2]
    Q2d_val[i]<-metrics_val_Q2d[2]


  }
DO_erorr[j]<-mean(Q2d_val)

}



## ----include=FALSE-------------------------------------------
## # e)

# Model with one hidden layer
Q2e_network <-keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu",
              input_shape = c(13),kernel_regularizer = regularizer_l2(0.0005)) %>%
  layer_dense(units = 1, activation = "linear")

Q2e_network %>% compile(
  optimizer = "rmsprop",
  loss = "mse",    # loss function to minimize
  metrics = c("mae")
```

```
)


Q2e_network %>% fit(train_data, train_targets, epochs = 100,
                       batch_size = 16)

### Examine training and validation performance
metrics_train_Q2e <- Q2e_network %>% evaluate(train_data, train_targets)
metrics_val_Q2e <- Q2e_network %>% evaluate(test_data, test_targets)
```