STAT 6390 - Deep Learning - Mini Project 4

Nisansala Wickramasinghe

Question 1 - MNIST dataset

a) R code for the implementation of multinomial logistic regression can be found in section 2 part a). Multinomial Logistic Regression model using nnet package doesn't work because there are too many weights to calculate. Therefore multinomial logistic regression was implemented using deep learning model with no hidden layers. After the 10th epoch model does not show big improvement. Training error and validation error for the model are 0.2649926 and 0.2796236 respectively.

b) R code for the implementation of neural network models with different number of hidden layers can be found in section 2 part b). **Table** 1 represent the training and validation error for each combination. Neural network model with 2 hidden layers perform better than other 2 models as it has smaller validation error.

Model	units per hidden layers	activations	Training error	Validation error
1 hidden layer	64	softmax	0.06443295	0.10657630
2 hidden layers	64, 128	relu, softmax	0.03759047	0.09324037
3 hidden layers	64, 128, 256	relu, relu, softmax	0.03059510	0.10161899

Table 1: Summary for the training and validation errors for different neural network models

c) R code for the implementation of neural network models with L2 weight regularization can be found in section 2 part c). L2 weight regularization was used to the model with 2 hidden layers. Optimal value for the tuning parameter is 0.0001. **Table** 2 represent the training and validation error for different parameters. Regularization appears to have limited or negligible impact on improving the performance of this dataset.

Tuning parameter	Training error	Validation error
0.01	0.32123530	0.3087860
0.001	0.14998582	0.1594464
0.0001	0.07291593	0.1127549

Table 2: Summary for the training and validation errors for the model with L2 weight regilarization

d) R code for the implementation of neural network models with drop out can be found in section 2 part d). Drop out was added to the model with 2 hidden layers. Optimal value for the drop out rate is 0.25. **Table** 3 represent the training and validation error for different rates. Adding dropout appears to have limited or negligible impact on improving the performance of this dataset.

Drop out rate	Training error	Validation error
0.25	0.1118707	0.1055799
0.5	0.2060379	0.1557025

Table 3: Summary for the training and validation errors for the model with diffrent drop out rates

e) I would recommend neural network model with 2 hidden layers obtained in part b) since it has the lowest validation error among all other models. This model has 2 hidden layers with 64 and 128 number of units per hidden layer respectively and does not incorporate any regularization or dropouts.

- Training error: 0.03759047
- Validation error: 0.09324037
- Test error: 0.09325194

f) Training multiple models and combining their predictions can improve the generalization performance of the model. We can consider training an ensemble of models with different architectures or different random initializations.

Question 2 - Boston Hosing Price dataset

a) R code for the implementation of linear regression can be found in section 2 part a). Training error and validation error for the model are 3.3183 and 3.463781 respectively.

b) R code for the implementation of neural network models with different number of hidden layers can be found in section 2 part b). **Table** 4 represent the training and validation error for each combination. Neural network model with 1 hidden layers perform better than the other models as it has smaller validation error.

Model	units per hidden layers	activations	Training error	Validation error
1 hidden layer	64	linear	2.008515	2.442444
2 hidden layers	64, 128	relu, linear	1.515102	2.500980

Table 4: Summary for the training and validation errors for different neural network models

c) R code for the implementation of neural network models with L1 weight regularization can be found in section 2 part c). L1 weight regularization was used to the model with 1 hidden layers. Optimal value for the tuning parameter is 0.001. **Table** 5 represent the training and validation error for different parameters. Regularization appears to have impact on improving the performance of this dataset.

Tuning parameter	Training error	Validation error
0.1	2.057065	2.362022
0.01	2.027539	2.367469
0.001	2.019204	2.357646
0.0001	2.017663	2.371652

Table 5: Summary for the training and validation errors for the model with L1 weight regilarization

d) R code for the implementation of neural network models with drop out can be found in section 2 part d). Drop out was added to the model with 1 hidden layers. Optimal value for the drop out rate is 0.25. **Table** 6 represent the training and validation error for different rates. Adding dropout appears to have limited or negligible impact on improving the performance of this dataset.

Drop out rate	Training error	Validation error
0.25	2.623615	2.471557
0.5	2.916082	3.055397

Table 6: Summary for the training and validation errors for the model with diffrent drop out rates

e) I would recommend the model best model obtained in part c) since it has the lowest validation error among all other models. The model is neural network model with 1 hidden layers and L1 regularization with 0.001 parameter.

- Training error: 2.01920
- Validation error: 2.357646
- Test error: 2.8780

f) Training multiple models and combining their predictions can improve the generalization performance of the model. We can consider training an ensemble of models with different architectures or different random initializations.

```
## ----setup, include=FALSE--------------------------------
knitr::opts_chunk$set(echo = TRUE)
options(xtable.comment = FALSE)
knitr::opts_chunk$set(dev = 'pdf')


## ----echo=FALSE------------------------------------------
library(knitr)
opts_chunk$set(comment="",warning = FALSE, message=FALSE,tidy.opts=list(keep.blank.line=TRUE, width.cutoff=120),


## ----eval=FALSE, include=FALSE---------------------------
## library(keras)
## library(nnet)
## library(glmnet)

#########################################################################
#                              Question 1
#########################################################################


## ----eval=FALSE, include=FALSE---------------------------
## ######################## Data Pre processing ############################
##
## # extract the data and get the training and test sets
## mnist <- dataset_mnist()
## ox_train <- mnist$train$x
## g_train <- mnist$train$y
## x_test <- mnist$test$x
## g_test <- mnist$test$y
##
## # flatten the 28x28 arrays into 784-vectors
## ox_train <- array_reshape(ox_train, c(nrow(ox_train), 784))
## x_test <- array_reshape(x_test, c(nrow(x_test), 784))
##
## # scale the pixels to lie between 0 and 1
## ox_train <- ox_train / 255
## x_test <- x_test / 255
##
## # make the response a 10-class vector (one-hot encoding)
## oy_train <- to_categorical(g_train, 10)
## y_test <- to_categorical(g_test, 10)


## ----eval=FALSE, include=FALSE---------------------------
## # 80-20 Training-validation split
## set.seed(123)
## train_indices <- sample(1:nrow(ox_train), round(0.8 * nrow(ox_train)), replace = FALSE)
## x_train <- ox_train[train_indices, ]
## y_train <- oy_train[train_indices, ]
## x_val <- ox_train[-train_indices, ]
## y_val <- oy_train[-train_indices, ]


## ----eval=FALSE, include=FALSE---------------------------
## ######################## Part a) ############################
##
## #Multinomial Logistic Regression model using nnet
##
```

```
## #Doesn't work because too many (7860) weights
## #MLR_model <- multinom(y_train ~ x_train)
## #summary(MLR_model)
##
## #Multinomial Logistic Regression model using neural network
##
## # Define the model architecture
## ML_model <- keras_model_sequential()
## ML_model %>%
##   layer_dense(units = 10, input_shape = c(784), activation = "softmax")
##
## # Compile the model
## ML_model %>% compile(
##   optimizer = "rmsprop",
##   loss = "categorical_crossentropy", # loss function to minimize
##   metrics = c("accuracy") # monitor classification accuracy
## )
##
## # Train the model and obtain the training history
## history <- ML_model %>% fit(
##   x_train, y_train,
##   epochs = 10,
##   batch_size = 128,
##   validation_data = list(x_val, y_val)
## )
##
## # Obtain the training and validation loss from the history object
## train_loss <- history$metrics$loss
## val_loss <- history$metrics$val_loss


## ----eval=FALSE, include=FALSE-----------------------------
## ######################## Part b) ############################
## #Neural network models with different hidden layers
##
## # Define the model architecture with 1 hidden layer
## model1 <- keras_model_sequential()
## model1 %>%
##   layer_dense(units = 64, input_shape = c(784), activation = "relu") %>%
##   layer_dense(units = 10, activation = "softmax")
##
## # Define the model architecture with 2 hidden layers
## model2 <- keras_model_sequential()
## model2 %>%
##   layer_dense(units = 64, input_shape = c(784), activation = "relu") %>%
##   layer_dense(units = 128, activation = "relu") %>%
##   layer_dense(units = 10, activation = "softmax")
##
## # Define the model architecture with 3 hidden layer
## model3 <- keras_model_sequential()
## model3 %>%
##   layer_dense(units = 64, input_shape = c(784), activation = "relu") %>%
##   layer_dense(units = 128, activation = "relu") %>%
##   layer_dense(units = 256, activation = "relu") %>%
##   layer_dense(units = 10, activation = "softmax")
##
## # Compile the models
## model1 %>% compile(
##   optimizer = "rmsprop",
```

```
##   loss = "categorical_crossentropy", # loss function to minimize
##   metrics = c("accuracy") # monitor classification accuracy
## )
##
## model2 %>% compile(
##   optimizer = "rmsprop",
##   loss = "categorical_crossentropy", # loss function to minimize
##   metrics = c("accuracy") # monitor classification accuracy
## )
##
## model3 %>% compile(
##   optimizer = "rmsprop",
##   loss = "categorical_crossentropy", # loss function to minimize
##   metrics = c("accuracy") # monitor classification accuracy
## )
##
## # Train the model and obtain the training history
## history1 <- model1 %>% fit(
##   x_train, y_train,
##   epochs = 10,
##   batch_size = 128,
##   validation_data = list(x_val, y_val)
## )
##
## history2 <- model2 %>% fit(
##   x_train, y_train,
##   epochs = 10,
##   batch_size = 128,
##   validation_data = list(x_val, y_val)
## )
##
## history3 <- model3 %>% fit(
##   x_train, y_train,
##   epochs = 10,
##   batch_size = 128,
##   validation_data = list(x_val, y_val)
## )
##
## # Obtain the training and validation loss from the history object
## train_loss1 <- history1$metrics$loss
## val_loss1 <- history1$metrics$val_loss
##
## train_loss2 <- history2$metrics$loss
## val_loss2 <- history2$metrics$val_loss
##
## train_loss3 <- history3$metrics$loss
## val_loss3 <- history3$metrics$val_loss


## ----eval=FALSE, include=FALSE-----------------------------
## ######################## Part c) ############################
## #Neural network models with 2 hidden layers with L2 regularization
##
## # Define the model architecture with 2 hidden layers
## l_parm <- c(0.01, 0.001, 0.0001)
## i = 1
## train_loss_L2 <- c()
## val_loss_L2 <- c()
##
```

```
## for(l in l_parm)
## {
##   model_L2 <- keras_model_sequential() %>%
##     layer_dense(units = 64, input_shape = c(784), activation = "relu",
##       kernel_regularizer = regularizer_l2(l)) %>%
##     layer_dense(units = 128, activation = "relu",
##       kernel_regularizer = regularizer_l2(l)) %>%
##     layer_dense(units = 10, activation = "softmax")
##
##   # Compile the model
##   model_L2 %>% compile(
##     optimizer = "rmsprop",
##     loss = "categorical_crossentropy", # loss function to minimize
##     metrics = c("accuracy") # monitor classification accuracy
##   )
##
## # Train the model and obtain the training history
##   history_L2 <- model_L2 %>% fit(
##     x_train, y_train,
##     epochs = 10,
##     batch_size = 128,
##     validation_data = list(x_val, y_val)
##   )
##
## # Obtain the training and validation loss from the history object
##   train_loss_L2[i] <- history_L2$metrics$loss[[10]]
##   cat("L2 regularization =", l, "Training loss =", train_loss_L2[i], "\n")
##
##   val_loss_L2[i] <- history_L2$metrics$val_loss[[10]]
##   cat("L2 regularization =", l, "validation loss =", val_loss_L2[i], "\n")
##
##   i = i+ 1
## }


## ----eval=FALSE, include=FALSE-----------------------------
## ######################## Part d) ############################
## #Neural network models with 2 hidden layers with different drop out rate
##
## # Define the model architecture with 2 hidden layers
## drop_parm <- c(0.25, 0.50)
## i = 1
## train_loss_drop <- c()
## val_loss_drop <- c()
##
## for(d in drop_parm)
## {
##   model_drop <- keras_model_sequential() %>%
##     layer_dense(units = 64, input_shape = c(784), activation = "relu") %>%
##       layer_dropout(rate = d) %>%
##     layer_dense(units = 128, activation = "relu") %>%
##       layer_dropout(rate = d) %>%
##     layer_dense(units = 10, activation = "softmax")
##
##   # Compile the model
##   model_drop %>% compile(
##     optimizer = "rmsprop",
##     loss = "categorical_crossentropy", # loss function to minimize
##     metrics = c("accuracy") # monitor classification accuracy
```

```
##   )
##
## # Train the model and obtain the training history
##   history_drop <- model_drop %>% fit(
##     x_train, y_train,
##     epochs = 10,
##     batch_size = 128,
##     validation_data = list(x_val, y_val)
##   )
##
## # Obtain the training and validation loss from the history object
##   train_loss_drop[i] <- history_drop$metrics$loss[[10]]
##   cat("L2 regularization =", d, "Training loss =", train_loss_drop[i], "\n")
##
##   val_loss_drop[i] <- history_drop$metrics$val_loss[[10]]
##   cat("L2 regularization =", d, "validation loss =", val_loss_drop[i], "\n")
##
##   i = i+ 1
## }


## ----eval=FALSE, include=FALSE-----------------------------
## ######################## Part e) ##########################
## # Define the model architecture with 2 hidden layers
## model_best <- keras_model_sequential()
## model_best %>%
##   layer_dense(units = 64, input_shape = c(784), activation = "relu") %>%
##   layer_dense(units = 128, activation = "relu") %>%
##   layer_dense(units = 10, activation = "softmax")
##
## # Compile the models
## model_best %>% compile(
##   optimizer = "rmsprop",
##   loss = "categorical_crossentropy", # loss function to minimize
##   metrics = c("accuracy") # monitor classification accuracy
## )
##
## # Train the model using all training data to calculate test error
## history_best <- model_best %>% fit(
##   ox_train, oy_train,
##   epochs = 10,
##   batch_size = 128,
## )
##
## #evaluate model to get the test error
## test_loss <- model_best %>% evaluate(x_test, y_test)


#############################################################################
#                             Question 2
#############################################################################


## ----eval=FALSE, include=FALSE-----------------------------
## ######################## Data Pre processing ##########################
##
## boston <- dataset_boston_housing()
## c(c(train_data, train_targets), c(test_data, test_targets)) %<-% boston
##
```

```
## ### 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)


## ----eval=FALSE, include=FALSE-----------------------------
## ######################## Part a) ##########################
## nw_train <- unlist(train_data)
## nw_train <- matrix(nw_train, nrow = 404, ncol = 13)
## nw_target <- as.vector(unlist(train_targets))
## cvfit <- cv.glmnet(nw_train, nw_target,
##     type.measure = "mae", nfolds = 4)
##
## train_pred <- predict(cvfit, newx = nw_train, s = "lambda.min")
##
## # Calculate training and cross-validation errors
## train_err <- mean(abs(nw_target - train_pred))
## cv_err <- cvfit$cvm[which.min(cvfit$cvm)]
## train_err
## cv_err


## ----eval=FALSE, include=FALSE-----------------------------
######################## Part b) ##########################
## # Define the model architecture with 1 hidden layers and compile model
## build_model1 <- function(){
##  model <- keras_model_sequential() %>%
##   layer_dense(units = 64, activation = "relu", input_shape = dim(train_data)[2]) %>%
##   layer_dense(units = 1)
##
##  model %>% compile(
##    optimizer = "rmsprop",
##    loss = "mse",
##    metrics = c("mae")
##  )
## }
##
## # Define the model architecture with 2 hidden layers and compile model
## build_model2 <- function(){
##  model <- keras_model_sequential() %>%
##   layer_dense(units = 64, activation = "relu", input_shape = dim(train_data)[2]) %>%
##   layer_dense(units = 128, activation = "relu") %>%
##   layer_dense(units = 1)
##
##  model %>% compile(
##    optimizer = "rmsprop",
##    loss = "mse",
##    metrics = c("mae")
##  )
## }
##
## # K-fold CV
##
## k <- 4
## indices <- sample(1:nrow(train_data))
## folds <- cut(indices, breaks = k, labels = FALSE)
## num_epochs <- 100 # number of epochs
```

```
## all_scores_model1 <- all_scores_model2 <- c()
## loss_m1 <- loss_m2 <-c()
##
## for (i in 1:k){
##   cat("Processing fold #", i, "\n")
##
##   # prepares the validation data
##   val_indices <- which(folds == i, arr.ind = TRUE)
##   val_data <- train_data[val_indices,]
##   val_targets <- train_targets[val_indices]
##
##   # prepares the training data
##   partial_train_data <- train_data[-val_indices,]
##   partial_train_targets <- train_targets[-val_indices]
##
##   # use precompiled model function
##   lm_model1 <- build_model1()
##   lm_model2 <- build_model2()
##
##   # trains the model in silent mode (verbose = 0)
##   history_m1 <- lm_model1 %>% fit(partial_train_data, partial_train_targets,
##                 epochs = num_epochs, batch_size = 16,
##                 verbose = 0)
##
##   history_m2 <- lm_model2 %>% fit(partial_train_data, partial_train_targets,
##                 epochs = num_epochs, batch_size = 16,
##                 verbose = 0)
##
##   loss_m1[i] <- history_m1$metrics$mae[[num_epochs]]
##   loss_m2[i] <- history_m2$metrics$mae[[num_epochs]]
##
##   # evaluate model on the validation data
##   results_model1 <- lm_model1 %>% evaluate(val_data, val_targets, verbose = 0)
##   all_scores_model1 <- c(all_scores_model1, results_model1["mae"])
##
##   results_model2 <- lm_model2 %>% evaluate(val_data, val_targets, verbose = 0)
##   all_scores_model2 <- c(all_scores_model2, results_model2["mae"])
## }
##
## all_scores_model1
## mean(all_scores_model1)
##
## all_scores_model2
## mean(all_scores_model2)
##


## ----eval=FALSE, include=FALSE-----------------------------
###################### Part c) ########################
## # Define the model architecture with L1 regularization and compile model
## build_model_L1 <- function(l1){
##  model <- keras_model_sequential() %>%
##   layer_dense(units = 64, activation = "relu",
##               input_shape = dim(train_data)[2],
##               kernel_regularizer = regularizer_l2(l1)) %>%
##   layer_dense(units = 1)
##
##  model %>% compile(
##    optimizer = "rmsprop",
```

```
##    loss = "mse",
##    metrics = c("mae")
##  )
## }
##
## l1_param <- c(0.1,0.01,0.001,0.0001)
## L1_error <- L1_training <-  c()
## r = 1
##
## # K-fold CV
## for(j in l1_param)
## {
##   k <- 4
##   indices <- sample(1:nrow(train_data))
##   folds <- cut(indices, breaks = k, labels = FALSE)
##   num_epochs <- 100 # number of epochs
##   all_scores_L1 <- loss_training_L1 <- c()
##
##   for (i in 1:k){
##     cat("Processing fold #", i, "\n")
##
##     # prepares the validation data
##     val_indices <- which(folds == i, arr.ind = TRUE)
##     val_data <- train_data[val_indices,]
##     val_targets <- train_targets[val_indices]
##
##     # prepares the training data
##     partial_train_data <- train_data[-val_indices,]
##     partial_train_targets <- train_targets[-val_indices]
##
##     # use precompiled model function
##     L1_model <- build_model_L1(j)
##
##     # trains the model in silent mode (verbose = 0)
##     history_L1 <- L1_model %>% fit(partial_train_data, partial_train_targets,
##                 epochs = num_epochs, batch_size = 16,
##                 verbose = 0)
##
##     loss_training_L1[i] <- history_L1$metrics$mae[[num_epochs]]
##
##     # evaluate model on the validation data
##     results_L1 <- L1_model %>% evaluate(val_data, val_targets, verbose = 0)
##     all_scores_L1 <- c(all_scores_L1, results_L1["mae"])
##   }
##
##   L1_training[r] <- mean(loss_training_L1)
##   L1_error[r] <- mean(all_scores_L1)
##   r = r + 1
## }


## ----eval=FALSE, include=FALSE-----------------------------
###################### Part d) ##########################
## # Define the model architecture with dropout and compile model
## build_model_drop <- function(d){
##  model <- keras_model_sequential() %>%
##   layer_dense(units = 64, activation = "relu", input_shape = dim(train_data)[2]) %>%
##     layer_dropout(rate = d) %>%
##   layer_dense(units = 1)
```

```
##
##  model %>% compile(
##    optimizer = "rmsprop",
##    loss = "mse",
##    metrics = c("mae")
##  )
## }
##
## #Defining parameter grid
## drop_param <- c(0.25, 0.5)
## drop_error <-  drop_training_err <- c()
## r = 1
##
## # K-fold CV
## for(j in drop_param)
## {
##   k <- 4
##   indices <- sample(1:nrow(train_data))
##   folds <- cut(indices, breaks = k, labels = FALSE)
##   num_epochs <- 100 # number of epochs
##   all_scores_drop <- loss_training_drop <- c()
##
##   for (i in 1:k){
##     cat("Processing fold #", i, "\n")
##
##     # prepares the validation data
##     val_indices <- which(folds == i, arr.ind = TRUE)
##     val_data <- train_data[val_indices,]
##     val_targets <- train_targets[val_indices]
##
##     # prepares the training data
##     partial_train_data <- train_data[-val_indices,]
##     partial_train_targets <- train_targets[-val_indices]
##
##     # use precompiled model function
##     drop_model <- build_model_drop(j)
##
##     # trains the model in silent mode (verbose = 0)
##     history_drop_lm <- drop_model %>% fit(partial_train_data, partial_train_targets,
##                 epochs = num_epochs, batch_size = 16,
##                 verbose = 0)
##
##     loss_training_drop[i] <- history_drop_lm$metrics$mae[[num_epochs]]
##
##     # evaluate model on the validation data
##     results_drop <- drop_model %>% evaluate(val_data, val_targets, verbose = 0)
##     all_scores_drop <- c(all_scores_drop, results_drop["mae"])
##   }
##   drop_training_err[r] <- mean(loss_training_drop)
##   drop_error[r] <- mean(all_scores_drop)
##   r = r + 1
## }


## ----eval=FALSE, include=FALSE-----------------------------
###################### Part e) ########################
## # Define the model architecture with 2 hidden layers
## lm_model_best <- keras_model_sequential()
## lm_model_best %>%
```

```
##   layer_dense(units = 64, activation = "relu",
##               input_shape = dim(train_data)[2],
##               kernel_regularizer = regularizer_l2(l)) %>%
##   layer_dense(units = 1)
##
## # Compile the models
## lm_model_best %>% compile(
##    optimizer = "rmsprop",
##    loss = "mse",
##    metrics = c("mae")
## )
##
## # Train the model using all training data to calculate test error
## history_best_lm <- lm_model_best %>% fit(
##   train_data , train_targets,
##   epochs = 100,
##   batch_size = 16,
## )
##
## #evaluate model to get the test error
## test_loss <- lm_model_best %>% evaluate(test_data, test_targets)
```