R code for Section 3 of "DL 101: Basic introduction to deep learning with its application in biomedical related fields''

Tianyu Zhan

In this R Markdown document, we provide the R code to replicate results in Section 3 of the simulation study, including training data simulation, hyperparameter tuning, final FNN training.

1. Preparation

In this section, we load required R packages and functions, simulate data, and conduct data processing for FNN training.

```
## Install and load R packages
library(ggplot2)
library(ggforce)
library(gghighlight)
library(keras)
library(reticulate)
library(tensorflow)
library(tibble)
library(kernlab)
library(ddpcr)
```

```
## The function to train FNN
FNN.fit.func = function(data.train.scale.in, data.train.label.in,
                   drop.rate.in, active.name.in, n.node.in,
                   n.layer.in, max.epoch.in, validation.prop.in,
                   batch.size.in, learning.rate.in){
  k_clear_session()
  set_random_seed(12)
  build_model <- function(drop.rate.in) {
    model <- NULL

    model.text.1 = paste0("model <- keras_model_sequential() %>%
                          layer_dense(units = n.node.in, activation =",
                          shQuote(active.name.in),
                          ",input_shape = dim(data.train.scale.in)[2]) %>%
                          layer_dropout(rate=", drop.rate.in, ")%>%")

    model.text.2 = paste0(rep(paste0(" layer_dense(units = n.node.in, activation = ",
                                shQuote(active.name.in),
                                ") %>% layer_dropout(rate=", drop.rate.in, ")%>%"),
                            (n.layer.in-1)), collapse ="")

    ### model.text.3
    model.text.3 = paste0("layer_dense(units = 1, activation = ",
```

```r
                          shQuote("sigmoid"), ")")

    eval(parse(text=paste0(model.text.1, model.text.2, model.text.3)))

    model %>% compile(
      loss = "binary_crossentropy",
      optimizer = optimizer_rmsprop(learning.rate.in),
      metrics = c('accuracy')
    )

    model
  }

  out.model <- build_model(drop.rate.in)
  out.model %>% summary()

  print_dot_callback <- callback_lambda(
    on_epoch_end = function(epoch, logs) {
      if (epoch %% 100 == 0) cat("\n")
      cat(".")
    }
  )

  history <- out.model %>% fit(
    data.train.scale.in,
    data.train.label.in,
    epochs = max.epoch.in,
    validation_split = validation.prop.in,
    verbose = 0,
    callbacks = list(print_dot_callback),
    batch_size = batch.size.in,
    shuffle = FALSE
  )
  return(list("model" = out.model, "history" = history))
}
```

```r
## The data generation code below is adopted from Lang and Witbrock 1989 to simulate
## two spirals.
n=100
x = y = rep(NA, n)
tol = 0

for (i in 1:n) {
  angle = i * pi / 16;
  radius = 6.5 * (104-i)/104 ;
  x[i] = radius * sin(angle) + runif(1, min=-tol, max = tol);
  y[i] = radius * cos(angle) + runif(1, min=-tol, max = tol);
}

## Data frame for the training data
data.train = data.frame("x1" = c(x, -x),
                        "x2" = c(y, -y),
                        "label" = c(rep(0, n), rep(1, n))
                        )
```

```r
## Add square terms and the interaction term
data.train$x1_2 = data.train$x1^2
data.train$x2_2= data.train$x2^2
data.train$x1_x2 = data.train$x1*data.train$x2
```

```r
## Data pre-processing
## Shuffle data
set.seed(12)
data.train = data.train[sample(1:dim(data.train)[1]), ]

## 20% of data for validation
validation.prop = 0.2
data.train$train_ind = c(rep(0, (1-validation.prop)*dim(data.train)[1]),
                         rep(1, validation.prop*dim(data.train)[1]))
data.train.tibble = as_tibble(data.train[,c("x1", "x2", "x1_2", "x2_2", "x1_x2")])

## Output label
label.train = data.train[, "label"]

## Standardize input data to be mean 0 and SD 1
data.train.scale = scale(data.train.tibble)
col_means_train = attr(data.train.scale, "scaled:center")
col_stddevs_train = attr(data.train.scale, "scaled:scale")
```

2. Hyperparameter Tuning

In this section, we first select the sub-optimal values of activation function, learning rate, number of training epochs and DNN structure. Then we simulate 30 sets of hyperparameters to simultaneously find the optimal set.

```r
## Varying activation functions
results.af = data.frame("af" = c("sigmoid", "relu", "tanh"))
results.af$train_acc = results.af$train_loss =
  results.af$val_acc = results.af$val_loss = NA

for (ind in 1:length(results.af$af)){
  FNN.fit.af = FNN.fit.func(data.train.scale.in = data.train.scale,
                            data.train.label.in = as.numeric(label.train),
                            drop.rate.in = 0.2,
                            active.name.in = results.af$af[ind],
                            n.node.in = 30,
                            n.layer.in = 3,
                            max.epoch.in = (10^4),
                            validation.prop.in = validation.prop,
                            batch.size.in = 32,
                            learning.rate.in = 0.01)

  results.af$train_loss[ind] = tail(FNN.fit.af$history$metrics$loss, 1)
  results.af$train_acc[ind] = tail(FNN.fit.af$history$metrics$acc, 1)
  results.af$val_loss[ind] = tail(FNN.fit.af$history$metrics$val_loss, 1)
  results.af$val_acc[ind] = tail(FNN.fit.af$history$metrics$val_acc, 1)
}
```

```r
write.csv(results.af, "results_af.csv")
```

```r
## Print results
print(results.af)
```

```
##        af  val_loss val_acc train_loss train_acc
## 1 sigmoid 0.1008072   0.950  0.1328502   0.96250
## 2    relu 7.0286422   0.550 13.3057251   0.61875
## 3    tanh 0.4992707   0.825  0.1787582   0.93125
```

```r
## Varying learning rates
results.lr = data.frame("lr" = c(10^(-3), 10^(-2), 0.05, 0.1))
results.lr$train_acc = results.lr$train_loss =
  results.lr$val_acc = results.lr$val_loss = NA

for (ind in 1:length(results.lr$lr)){
  FNN.fit.lr = FNN.fit.func(data.train.scale.in = data.train.scale,
                            data.train.label.in = as.numeric(label.train),
                            drop.rate.in = 0.2,
                            active.name.in = "sigmoid",
                            n.node.in = 30,
                            n.layer.in = 3,
                            max.epoch.in = (10^4),
                            validation.prop.in = validation.prop,
                            batch.size.in = 32,
                            learning.rate.in = results.lr$lr[ind])
  results.lr$train_loss[ind] = tail(FNN.fit.lr$history$metrics$loss, 1)
  results.lr$train_acc[ind] = tail(FNN.fit.lr$history$metrics$acc, 1)
  results.lr$val_loss[ind] = tail(FNN.fit.lr$history$metrics$val_loss, 1)
  results.lr$val_acc[ind] = tail(FNN.fit.lr$history$metrics$val_acc, 1)
}

write.csv(results.lr, "results_lr.csv")
```

```r
## Print results
print(results.lr)
```

```
##      lr  val_loss val_acc train_loss train_acc
## 1 0.001 0.6525650   0.725  0.3463239   0.81250
## 2 0.010 0.1008072   0.950  0.1328502   0.96250
## 3 0.050 0.8311044   0.875  0.1932663   0.93125
## 4 0.100 0.6365913   0.775  0.2392740   0.90625
```

```r
## Varying numbers of training epochs
results.epoch = data.frame("epoch" = c(10^2, 10^3, 10^4, 10^5))
results.epoch$train_acc = results.epoch$train_loss =
  results.epoch$val_acc = results.epoch$val_loss = NA

for (ind in 1:length(results.epoch$epoch)){
  FNN.fit.epoch = FNN.fit.func(data.train.scale.in = data.train.scale,
                         data.train.label.in = label.train,
                         drop.rate.in = 0.2,
                         active.name.in = "sigmoid",
                         n.node.in = 30,
                         n.layer.in = 3,
                         max.epoch.in = (results.epoch$epoch[ind]),
```

```r
                          validation.prop.in = validation.prop,
                          batch.size.in = 32,
                          learning.rate.in = 0.01)
  results.epoch$train_loss[ind] = tail(FNN.fit.epoch$history$metrics$loss, 1)
  results.epoch$train_acc[ind] = tail(FNN.fit.epoch$history$metrics$acc, 1)
  results.epoch$val_loss[ind] = tail(FNN.fit.epoch$history$metrics$val_loss, 1)
  results.epoch$val_acc[ind] = tail(FNN.fit.epoch$history$metrics$val_acc, 1)
}

write.csv(results.epoch, "results_epoch.csv")
```

```r
## Print results
print(results.epoch)
```

```
##   epoch  val_loss val_acc train_loss train_acc
## 1 1e+02 0.7228225   0.475  0.6901244   0.53125
## 2 1e+03 0.3547103   0.750  0.3853561   0.80625
## 3 1e+04 0.1008072   0.950  0.1328502   0.96250
## 4 1e+05 0.3315821   0.925  0.1481519   0.94375
```

```r
## Varying DNN structures
results.structure = data.frame("layer" = c(1, 1, 4, 4, 7, 7),
                               "node" = c(20, 40, 20, 40, 20, 40))
results.structure$train_acc = results.structure$train_loss =
  results.structure$val_acc = results.structure$val_loss = NA

for (ind in 1:length(results.structure$layer)){
  FNN.fit.structure = FNN.fit.func(data.train.scale.in = data.train.scale,
                               data.train.label.in = label.train,
                               drop.rate.in = 0.2,
                               active.name.in = "sigmoid",
                               n.node.in = results.structure$node[ind],
                               n.layer.in = results.structure$layer[ind],
                               max.epoch.in = (10^4),
                               validation.prop.in = validation.prop,
                               batch.size.in = 32,
                               learning.rate.in = 0.01)
  results.structure$train_loss[ind] = tail(FNN.fit.structure$history$metrics$loss, 1)
  results.structure$train_acc[ind] = tail(FNN.fit.structure$history$metrics$acc, 1)
  results.structure$val_loss[ind] = tail(FNN.fit.structure$history$metrics$val_loss, 1)
  results.structure$val_acc[ind] = tail(FNN.fit.structure$history$metrics$val_acc, 1)
}

write.csv(results.structure, "results_structure.csv")
```

```r
## Print results
print(results.structure)
```

```
##   layer node  val_loss val_acc train_loss train_acc
## 1     1   20 0.8521945   0.425  0.4726190   0.75625
## 2     1   40 0.9124352   0.700  0.2631585   0.88125
## 3     4   20 0.2690582   0.850  0.3084063   0.85625
## 4     4   40 0.5676410   0.900  0.0510916   0.98750
## 5     7   20 0.6482310   0.700  0.4771845   0.77500
## 6     7   40 0.6969763   0.450  0.6932508   0.51250
```

```r
## Simulate 30 sets of hyperparameters based on ranges determined by sub-optimal values
set.seed(12)
n.selection = 30
results.para = data.frame("layer" = round(runif(n.selection, min = 3, max = 5)),
                          "node" = round(runif(n.selection, min = 30, max = 50)),
                          "epoch" = round(runif(n.selection, min = 1000, max = 100000)),
                          "dropout" = runif(n.selection, min = 0, max = 0.5),
                          "batchsize" = round(runif(n.selection, min = 10, max = 80)),
                          "rate" = runif(n.selection, min = 0.001, max = 0.05)
                          )
results.para$train_acc = results.para$train_loss =
  results.para$val_acc = results.para$val_loss = NA

for (ind in 1:length(results.para$layer)){
  FNN.fit.para = FNN.fit.func(data.train.scale.in = data.train.scale,
                              data.train.label.in = label.train,
                              drop.rate.in = results.para$dropout[ind],
                              active.name.in = "sigmoid",
                              n.node.in = results.para$node[ind],
                              n.layer.in = results.para$layer[ind],
                              max.epoch.in = (results.para$epoch[ind]),
                              validation.prop.in = validation.prop,
                              batch.size.in = results.para$batchsize[ind],
                              learning.rate.in = results.para$rate[ind])
  results.para$train_loss[ind] = tail(FNN.fit.para$history$metrics$loss, 1)
  results.para$train_acc[ind] = tail(FNN.fit.para$history$metrics$acc, 1)
  results.para$val_loss[ind] = tail(FNN.fit.para$history$metrics$val_loss, 1)
  results.para$val_acc[ind] = tail(FNN.fit.para$history$metrics$val_acc, 1)
}

write.csv(results.para, "results_para.csv")
```

3. Final FNN training

In this section, we train FNN based on this set of hyperparameters.

```r
## Find the set of hyperparameters with the largest validation accuracy
data.para.best = results.para[which.max(results.para$val_acc),]

## Train FNN with this set of hyperparameters
FNN.fit.best = FNN.fit.func(data.train.scale.in = data.train.scale,
                            data.train.label.in = label.train,
                            drop.rate.in = data.para.best$dropout,
                            active.name.in = "sigmoid",
                            n.node.in = data.para.best$node,
                            n.layer.in = data.para.best$layer,
                            max.epoch.in = (data.para.best$epoch),
                            validation.prop.in = 0.2,
                            batch.size.in = data.para.best$batchsize,
                            learning.rate.in = data.para.best$rate)
```

```r
## Print the optimal set of hyperparameters
print(data.para.best)
```

```
##    layer node epoch    dropout batchsize       rate    val_loss val_acc
## 18     4   31 70476 0.09763384        55 0.04769786 0.003796674       1
##    train_loss train_acc
## 18 0.03468056   0.99375
```

```r
## Print training loss, training accuracy, validation loss, and validation accuracy of FNN.
print(FNN.fit.best$history)
```

```
##
## Final epoch (plot to see history):
##         loss: 0.03468
##     accuracy: 0.9937
##     val_loss: 0.003797
## val_accuracy: 1
```