The Case Study of 'Optimizing Graphical Procedures for Multiplicity Control in a Confirmatory Clinical Trial via Deep Learning'

Tianyu Zhan, Alan Hartford, Jian Kang, and Walter Offen

July 10, 2020

In this document, we illustrate how to replicate the case study in the main article. The user can refer to https://keras.rstudio.com/ for the installation of Keras and Tensorflow.

```
setwd("~/Dropbox/Research/AbbVie/graph nnw/R code/code_sharing_Github/R_markdown_help_file/")
source("graph_nn_general_functions_keras.R")
library(gMCP)
library(doParallel)
```

```
## Loading required package: foreach

## Loading required package: iterators

## Loading required package: parallel
```

```
library(MASS)
library(nloptr)
library(stringr)
library(ANN2)
library(CVTuningCov)
library(keras)
library(reticulate)
library(tensorflow)
```

```
##
## Attaching package: 'tensorflow'

## The following object is masked from 'package:ANN2':
##
##     train
```

```
library(keras)
library(tibble)
options(warn=-1)
seed.number = 123
set.seed(seed.number)

corr = c(0.5) # correlation magnitude between endpoints
n.sim = 10^6 # number of simulation per graph

obj.weight = c(0, 0.6, 0.2, 0.1, 0.1) # weights in the objective function
n.hypo = length(obj.weight) # number of endpoints
alpha.const = c(1, rep(0, 4)) # contraints in the initial alpha allocation vector
```

```r
## constraints in the transition weight matrix
w.const = matrix(1, nrow = n.hypo, ncol = n.hypo)
diag(w.const) = 0
w.const[,1] = 0

n.graph = (10^3) # number of graphs in the training dataset
max.epoch = 10^4 # training epochs for FNN

type.1.error = 0.025 # one-sided FWER
pow.vec = c(0.95, 0.9, 0.85, 0.65, 0.6) # marginal power vector
trt.vec = qnorm(1-type.1.error)-qnorm(pow.vec, lower.tail = FALSE) # marginal treatment effect vector b


# compound symmetric correlation structure
sigma.mat = matrix(corr, nrow = n.hypo, ncol = n.hypo)
diag(sigma.mat) = 1
```

Generate training data for FNN

```r
# simulate alpha vector and transition
alpha.fit = draw.alpha.fun(n.hypo, n.graph, alpha.const)
w.fit = draw.w.fun(n.hypo, n.graph, w.const)
obtain.name.fit = obtain.name.func(alpha.const, w.const)
name.free.space = obtain.name.fit$name.free.space
name.free.plus = obtain.name.fit$name.free.plus
name.free.comma = obtain.name.fit$name.free.comma

# generate training data for FNN
sim.data.fit = sim.data.function(n.hypo.in = n.hypo,
                                 n.sim.in = n.sim,
                                 trt.vec.in = trt.vec,
                                 alpha.fit.in = alpha.fit,
                                 w.fit.in = w.fit,
                                 sigma.in = sigma.mat,
                                 corr.in = corr)

  save(sim.data.fit, file = paste0("data pow ", paste(pow.vec, collapse = " "),
                                   " obj ", paste(obj.weight, collapse = " "),
                                     " corr ", corr, " n.sim ", log(n.sim)/log(10), " n.graph ",
                                     n.graph, ".RData"))
```

Cross validation to select FNN structure

```r
cross.time = Sys.time() ## starting time for cross-validation
n.nn = 6 # number of candidate structures
n.k = 5 # 5-fold cross-validation
neu.fit.cross.par = matrix(NA, nrow = n.nn*n.k, ncol = 12 + length(name.free.space))
neu.fit.cross = matrix(NA, nrow = n.nn, ncol = 12 + length(name.free.space))
colnames(neu.fit.cross.par) = colnames(neu.fit.cross) =
  c("TD_MSE", "VD_MSE", "opt_fit_power", "opt_real_power", "opt_rank",
```

```r
    name.free.space,
    "max_power", "hidden", "layer", "drop_rate", "time", "iters", "status")

for (n.fit.itt in 1:(n.nn*n.k)){

    n.nn.itt = (n.fit.itt-1)%/%n.k+1
    n.k.itt = n.fit.itt%%n.k
    if (n.k.itt==0) n.k.itt = n.k

    if (n.nn.itt==1){n.node = 30; n.layer = 2; drop.rate = 0.3}
    if (n.nn.itt==2){n.node = 30; n.layer = 3; drop.rate = 0.3}
    if (n.nn.itt==3){n.node = 30; n.layer = 4; drop.rate = 0.3}

    if (n.nn.itt==4){n.node = 30; n.layer = 2; drop.rate = 0}
    if (n.nn.itt==5){n.node = 30; n.layer = 3; drop.rate = 0}
    if (n.nn.itt==6){n.node = 30; n.layer = 4; drop.rate = 0}

  # print(paste("n.fit.itt", n.fit.itt))
  neu.func.fit = neu.function(n.node.in = n.node,
                              n.layer.in = n.layer,
                              k.indicator = FALSE,
                              k.itt.in = n.k.itt,
                              data.net.in = sim.data.fit$data.matrix,
                              pval.sim.mat.in = sim.data.fit$pval.matrix,
                              parallel = FALSE,
                              obtain.name.fit = obtain.name.fit,
                              drop.rate.in = drop.rate,
                              max.epoch.in = 10^3,
                              df.fit.tol.in = 10^(-3), # tolerance on the fine-tune step
                              df.max.n.in = 1, # 1 iteration for the fine-tune step
                              df.max.t.in = -1)
  neu.fit.cross.par[n.fit.itt, ] = as.numeric(neu.func.fit$n.nodes.output)
}

for (n.nn.itt in 1:n.nn){
  print(paste("n.nn.itt", n.nn.itt))

  neu.fit.temp = neu.fit.cross.par[(1:n.k)+(n.nn.itt-1)*n.k, ]
  neu.fit.cross[n.nn.itt, ] = apply(neu.fit.temp, 2, function(x){mean(x, na.rm = TRUE)})
}

neu.fit.cross = data.frame(neu.fit.cross)
neu.fit.cross$time = difftime(Sys.time(), cross.time, units="secs")
neu.fit.cross = data.frame(neu.fit.cross)

write.csv(neu.fit.cross,
        paste0("cross pow ", paste(pow.vec, collapse = " "),
               " obj ", paste(obj.weight, collapse = " "),
               " corr ", corr, " hypo ", n.hypo,
               ".csv"),  row.names=FALSE)

## select the final DNN structure with the smallest training error
opt.nn.ind = which.min(neu.fit.cross$TD_MSE)
```

```r
opt.n.node = neu.fit.cross$hidden[opt.nn.ind]
opt.n.layer = neu.fit.cross$layer[opt.nn.ind]
opt.drop.rate = neu.fit.cross$drop_rate[opt.nn.ind]

print(neu.fit.cross)
```

FNN-based method

```r
n.fit = 1 # number of fitting

neu.fit.mat = array(NA, dim = c(n.fit, dim(sim.data.fit$data.matrix)[1],
                                dim(sim.data.fit$data.matrix)[2]+4))
neu.fit.opt = matrix(NA, nrow = n.fit, ncol = 12 + length(name.free.space))
colnames(neu.fit.opt) =
  c("TD_MSE", "VD_MSE", "opt_fit_power", "opt_real_power", "opt_rank",
    name.free.space,
    "max_power", "hidden", "layer", "drop_rate", "time", "iters", "status")

for (n.fit.itt in 1:n.fit){

  print(paste("n.fit.itt", n.fit.itt))
  neu.func.fit = neu.function(n.node.in = opt.n.node,
                              n.layer.in = opt.n.layer,
                              k.indicator = FALSE,
                              k.itt.in = 1,
                              data.net.in = sim.data.fit$data.matrix,
                              pval.sim.mat.in = sim.data.fit$pval.matrix,
                              parallel = FALSE,
                              obtain.name.fit = obtain.name.fit,
                              drop.rate.in = opt.drop.rate,
                              max.epoch.in = max.epoch,
                              df.fit.tol.in = 10^(-4), ## tolerance for fine-tune step
                              df.max.n.in = 10^4, ## number of iteration for fine-tunestep
                              df.max.t.in = -1)
  neu.fit.opt[n.fit.itt, ] = as.numeric(neu.func.fit$n.nodes.output)

  data.tv.com = rbind(neu.func.fit$data.train, neu.func.fit$data.val)
  neu.fit.mat[n.fit.itt, , ] = as.matrix(data.tv.com)

}

neu.fit.opt = data.frame(neu.fit.opt)
neu.fit.opt.sum = neu.fit.opt[which.max(neu.fit.opt$opt_real_power), ]
neu.fit.mat.sum = neu.fit.mat[which.max(neu.fit.opt$opt_real_power), , ]
colnames(neu.fit.mat.sum) = colnames(data.tv.com)

## total fitting time for FNN-based method
neu.fit.opt.sum$total_time = neu.fit.opt$time +
  as.numeric(sim.data.fit$sim.data.time.diff)+
  neu.fit.cross$time[1]

neu.fit.opt.sum = data.frame(neu.fit.opt.sum)
```

```
write.csv(neu.fit.opt.sum,
          paste0("fit pow ", paste(pow.vec, collapse = " "),
                 " obj ", paste(obj.weight, collapse = " "),
                 " corr ", corr, " hypo ", n.hypo,
                 ".csv"),  row.names=FALSE)
write.csv(neu.fit.mat.sum,
          paste0("fit mat ", paste(pow.vec, collapse = " "),
                 " obj ", paste(obj.weight, collapse = " "),
                 " corr ", corr, " hypo ", n.hypo,
                 ".csv"),  row.names=FALSE)
```

ISRES and COBYLA

```
GF.name.vec = c("NLOPT_GN_ISRES", "NLOPT_LN_COBYLA")
GF.fit.n = 1 ## number of fitting

naive.fit.mat =  matrix(NA, nrow = length(GF.name.vec),
                        ncol = 4 + length(name.free.space))

for (n.fit.itt in 1:length(GF.name.vec)){
  GF.fit = naive.opt.func(nloptr.func.name = GF.name.vec[n.fit.itt],
                          naive.opt.n = 1,
                          naive.tol = 10^(-4),
                          naive.max.n = -1,
                          naive.max.t = neu.fit.opt.sum$total_time*1.5, ## the maximum wall time
                          pval.sim.mat.in = sim.data.fit$pval.matrix,
                          x0.given = NULL)
  naive.fit.mat[n.fit.itt, ] = c(GF.fit$naive.fit, GF.fit$solution,
                                 GF.fit$status, GF.fit$iters, GF.fit$time)
}
colnames(naive.fit.mat) = c("fit.power", name.free.space, "status", "iters", "time")
naive.fit.mat = data.frame(naive.fit.mat)
naive.fit.mat$name = GF.name.vec

write.csv(naive.fit.mat,
          paste0("GF ", paste(pow.vec, collapse = " "),
                 " obj ", paste(obj.weight, collapse = " "),
                 " corr ", corr, " GF_n ", GF.fit.n, " hypo ", n.hypo,
                 ".csv"),  row.names=FALSE)
```

Final results:

```
## FNN-based method solution:
print(neu.fit.opt.sum$opt_real_power)
```

```
## [1] 0.7802472
```

```
## FNN-based method fitting time in minutes:
print(neu.fit.opt.sum$total_time/60)
```

```
## [1] 17.09146
```

```r
## ISRES solution:
print(naive.fit.mat$fit.power[1])
```

```
## [1] 0.7744362
```

```r
## ISRES fitting time in minutes:
print(naive.fit.mat$time[1]/60)
```

```
## [1] 25.66756
```

```r
## COBYLA solution:
print(naive.fit.mat$fit.power[2])
```

```
## [1] 0.7724017
```

```r
## COBYLA fitting time in minutes:
print(naive.fit.mat$time[2]/60)
```

```
## [1] 4.010412
```

```r
## BF solution:
print(neu.fit.opt.sum$max_power)
```

```
## [1] 0.7662147
```

```r
## BF fitting time in minutes:
print(as.numeric(sim.data.fit$sim.data.time.diff)/60)
```

```
## [1] 9.992836
```