Practice 2: Hyperparameter Tuning of Initial Practice (Parkinson's Disease)

Statistical Learning with Deep Artificial Neural Networks

Alexander J Ohrt

11 mars, 2022

Contents

1 Introduction

We are working on a dataset describing Parkinson's disease. Click here for more information regarding the dataset. In short, the dataset is composed of a range of biomedical voice measurements from 42 people with early-stage Parkinson's disease recruited to a six-month trial of a telemonitoring device for remote symptom progression monitoring.

The main objective is to predict the severity of Parkinson's disease based on the data. More details are given in the following.

2 Load the Parkinsons Data

```
data <- read.csv("../practice_1/parkinsons_updrs.data")
str(data)
```

```
#> 'data.frame':    5875 obs. of  22 variables:
#>  $ subject.     : int  1 1 1 1 1 1 1 1 1 1 ...
#>  $ age          : int  72 72 72 72 72 72 72 72 72 72 ...
#>  $ sex          : int  0 0 0 0 0 0 0 0 0 0 ...
#>  $ test_time    : num  5.64 12.67 19.68 25.65 33.64 ...
#>  $ motor_UPDRS  : num  28.2 28.4 28.7 28.9 29.2 ...
#>  $ total_UPDRS  : num  34.4 34.9 35.4 35.8 36.4 ...
#>  $ Jitter...    : num  0.00662 0.003 0.00481 0.00528 0.00335 0.00353 0.00422 0.00476 0.00432 0.00496
#>  $ Jitter.Abs.  : num  3.38e-05 1.68e-05 2.46e-05 2.66e-05 2.01e-05 ...
#>  $ Jitter.RAP   : num  0.00401 0.00132 0.00205 0.00191 0.00093 0.00119 0.00212 0.00226 0.00156 0.002
#>  $ Jitter.PPQ5  : num  0.00317 0.0015 0.00208 0.00264 0.0013 0.00159 0.00221 0.00259 0.00207 0.00253
#>  $ Jitter.DDP   : num  0.01204 0.00395 0.00616 0.00573 0.00278 ...
#>  $ Shimmer      : num  0.0256 0.0202 0.0168 0.0231 0.017 ...
#>  $ Shimmer.dB.  : num  0.23 0.179 0.181 0.327 0.176 0.214 0.445 0.212 0.371 0.31 ...
#>  $ Shimmer.APQ3 : num  0.01438 0.00994 0.00734 0.01106 0.00679 ...
#>  $ Shimmer.APQ5 : num  0.01309 0.01072 0.00844 0.01265 0.00929 ...
#>  $ Shimmer.APQ11: num  0.0166 0.0169 0.0146 0.0196 0.0182 ...
#>  $ Shimmer.DDA  : num  0.0431 0.0298 0.022 0.0332 0.0204 ...
#>  $ NHR          : num  0.0143 0.0111 0.0202 0.0278 0.0116 ...
#>  $ HNR          : num  21.6 27.2 23 24.4 26.1 ...
#>  $ RPDE         : num  0.419 0.435 0.462 0.487 0.472 ...
#>  $ DFA          : num  0.548 0.565 0.544 0.578 0.561 ...
#>  $ PPE          : num  0.16 0.108 0.21 0.333 0.194 ...
```

```
summary(data)
```

```
#>     subject.          age            sex           test_time
#>  Min.   : 1.00   Min.   :36.0   Min.   :0.0000   Min.   : -4.263
#>  1st Qu.:10.00   1st Qu.:58.0   1st Qu.:0.0000   1st Qu.: 46.847
#>  Median :22.00   Median :65.0   Median :0.0000   Median : 91.523
#>  Mean   :21.49   Mean   :64.8   Mean   :0.3178   Mean   : 92.864
#>  3rd Qu.:33.00   3rd Qu.:72.0   3rd Qu.:1.0000   3rd Qu.:138.445
#>  Max.   :42.00   Max.   :85.0   Max.   :1.0000   Max.   :215.490
#>   motor_UPDRS      total_UPDRS      Jitter...         Jitter.Abs.
#>  Min.   : 5.038   Min.   : 7.00   Min.   :0.000830   Min.   :2.250e-06
#>  1st Qu.:15.000   1st Qu.:21.37   1st Qu.:0.003580   1st Qu.:2.244e-05
#>  Median :20.871   Median :27.58   Median :0.004900   Median :3.453e-05
#>  Mean   :21.296   Mean   :29.02   Mean   :0.006154   Mean   :4.403e-05
#>  3rd Qu.:27.596   3rd Qu.:36.40   3rd Qu.:0.006800   3rd Qu.:5.333e-05
#>  Max.   :39.511   Max.   :54.99   Max.   :0.099990   Max.   :4.456e-04
```

```
#>    Jitter.RAP        Jitter.PPQ5         Jitter.DDP          Shimmer
#>  Min.   :0.000330   Min.   :0.000430   Min.   :0.000980   Min.   :0.00306
#>  1st Qu.:0.001580   1st Qu.:0.001820   1st Qu.:0.004730   1st Qu.:0.01912
#>  Median :0.002250   Median :0.002490   Median :0.006750   Median :0.02751
#>  Mean   :0.002987   Mean   :0.003277   Mean   :0.008962   Mean   :0.03404
#>  3rd Qu.:0.003290   3rd Qu.:0.003460   3rd Qu.:0.009870   3rd Qu.:0.03975
#>  Max.   :0.057540   Max.   :0.069560   Max.   :0.172630   Max.   :0.26863
#>   Shimmer.dB.      Shimmer.APQ3      Shimmer.APQ5     Shimmer.APQ11
#>  Min.   :0.026   Min.   :0.00161   Min.   :0.00194   Min.   :0.00249
#>  1st Qu.:0.175   1st Qu.:0.00928   1st Qu.:0.01079   1st Qu.:0.01566
#>  Median :0.253   Median :0.01370   Median :0.01594   Median :0.02271
#>  Mean   :0.311   Mean   :0.01716   Mean   :0.02014   Mean   :0.02748
#>  3rd Qu.:0.365   3rd Qu.:0.02057   3rd Qu.:0.02375   3rd Qu.:0.03272
#>  Max.   :2.107   Max.   :0.16267   Max.   :0.16702   Max.   :0.27546
#>   Shimmer.DDA           NHR                HNR              RPDE
#>  Min.   :0.00484   Min.   :0.000286   Min.   : 1.659   Min.   :0.1510
#>  1st Qu.:0.02783   1st Qu.:0.010955   1st Qu.:19.406   1st Qu.:0.4698
#>  Median :0.04111   Median :0.018448   Median :21.920   Median :0.5423
#>  Mean   :0.05147   Mean   :0.032120   Mean   :21.680   Mean   :0.5415
#>  3rd Qu.:0.06173   3rd Qu.:0.031463   3rd Qu.:24.444   3rd Qu.:0.6140
#>  Max.   :0.48802   Max.   :0.748260   Max.   :37.875   Max.   :0.9661
#>       DFA              PPE
#>  Min.   :0.5140   Min.   :0.02198
#>  1st Qu.:0.5962   1st Qu.:0.15634
#>  Median :0.6436   Median :0.20550
#>  Mean   :0.6532   Mean   :0.21959
#>  3rd Qu.:0.7113   3rd Qu.:0.26449
#>  Max.   :0.8656   Max.   :0.73173
```

3 Description of the Variables

As we have seen above, the dataset contains 5875 rows, i.e. 5875 measurements. The columns consist of **patient ID**, **age**, **sex**, **time interval since enrollment date**, **motor_UPDRS**, **total_UPDRS** and 16 voice biomedical measurements. The variables are the following

- subject. - The patient ID. Integer that uniquely identifies each subject.
- age - Age of each subject.
- sex - Gender of the subject; '0' = male and '1' = female.
- test_time - Time since recruitment into the trial. The integer part is the number of days since recruitment.
- motor_UPDRS - Clinician's motor UPDRS score, linearly interpolated.
- ctotal_UPDRS - Clinician's total UPDRS score, linearly interpolated.
- Jitter(%), Jitter(Abs), Jitter:RAP, Jitter:PPQ5, Jitter:DDP - Several measures of variation in fundamental frequency.
- Shimmer, Shimmer(dB), Shimmer:APQ3, Shimmer:APQ5, Shimmer:APQ11, Shimmer:DDA - Several measures of variation in amplitude.
- NHR, HNR - Two measures of ratio of noise to tonal components in the voice.
- RPDE - A nonlinear dynamical complexity measure.
- DFA - Signal fractal scaling exponent.
- PPE - A nonlinear measure of fundamental frequency variation.

As noted, the objective is to predict the severity of the disease, where severity is defined based on the variable **total_UPDRS**: The disease is severe if **total_UPDRS $>$ 25**. This variable is created below.

4 Create the Binary Variable of Parkinson's Severity

```
data$severity <- data$total_UPDRS > 25
dim(data)
```

```
#> [1] 5875   23
```

```
summary(data$severity)
```

```
#>    Mode   FALSE    TRUE
#> logical    2188    3687
```

5 Normalization

The variables of the 16 voice measurements are normalized by means of the min-max transformation.

```
(biom_voice_mesures <- names(data)[7:22])
```

```
#>  [1] "Jitter..."     "Jitter.Abs."   "Jitter.RAP"    "Jitter.PPQ5"
#>  [5] "Jitter.DDP"    "Shimmer"       "Shimmer.dB."   "Shimmer.APQ3"
#>  [9] "Shimmer.APQ5"  "Shimmer.APQ11" "Shimmer.DDA"   "NHR"
#> [13] "HNR"           "RPDE"          "DFA"           "PPE"
```

```
mydata <- data[, biom_voice_mesures]

normalize <- function(x) {
    return((x- min(x))/(max(x)-min(x)))
}

mydata <- as.data.frame(lapply(mydata, normalize))
summary(mydata)
```

```
#>    Jitter...         Jitter.Abs.         Jitter.RAP        Jitter.PPQ5
#>  Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
#>  1st Qu.:0.02773   1st Qu.:0.04553   1st Qu.:0.02185   1st Qu.:0.02011
#>  Median :0.04104   Median :0.07281   Median :0.03356   Median :0.02980
#>  Mean   :0.05369   Mean   :0.09423   Mean   :0.04645   Mean   :0.04118
#>  3rd Qu.:0.06021   3rd Qu.:0.11523   3rd Qu.:0.05174   3rd Qu.:0.04383
#>  Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000
#>    Jitter.DDP         Shimmer          Shimmer.dB.       Shimmer.APQ3
#>  Min.   :0.00000   Min.   :0.00000   Min.   :0.0000   Min.   :0.00000
#>  1st Qu.:0.02185   1st Qu.:0.06047   1st Qu.:0.0716   1st Qu.:0.04762
#>  Median :0.03361   Median :0.09207   Median :0.1091   Median :0.07507
#>  Mean   :0.04650   Mean   :0.11664   Mean   :0.1369   Mean   :0.09652
#>  3rd Qu.:0.05179   3rd Qu.:0.13816   3rd Qu.:0.1629   3rd Qu.:0.11775
#>  Max.   :1.00000   Max.   :1.00000   Max.   :1.0000   Max.   :1.00000
#>   Shimmer.APQ5     Shimmer.APQ11       Shimmer.DDA           NHR
#>  Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
#>  1st Qu.:0.05361   1st Qu.:0.04827   1st Qu.:0.04758   1st Qu.:0.01426
#>  Median :0.08481   Median :0.07407   Median :0.07507   Median :0.02428
#>  Mean   :0.11027   Mean   :0.09155   Mean   :0.09650   Mean   :0.04256
#>  3rd Qu.:0.13215   3rd Qu.:0.11073   3rd Qu.:0.11775   3rd Qu.:0.04168
#>  Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000
#>       HNR              RPDE              DFA               PPE
#>  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
#>  1st Qu.:0.4900   1st Qu.:0.3911   1st Qu.:0.2336   1st Qu.:0.1893
```

```
#>  Median :0.5594   Median :0.4800   Median :0.3685   Median :0.2586
#>  Mean   :0.5528   Mean   :0.4790   Mean   :0.3959   Mean   :0.2784
#>  3rd Qu.:0.6291   3rd Qu.:0.5681   3rd Qu.:0.5612   3rd Qu.:0.3417
#>  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
```

6 Separation into Train and Test Data

I will use (pseudo-) random sampling to separate the data into a training and test set.

```
#set.seed(1)
ratio <- 0.7
sample.size <- floor(nrow(data) * ratio)
train.indices <- sample(1:nrow(data), size = sample.size)
train <- mydata[train.indices, ]
test <- mydata[-train.indices, ]

x_train <- data.matrix(train)
y_train <- as.numeric(data[train.indices,]$severity)
x_test <- data.matrix(test)
y_test <- as.numeric(data[-train.indices,]$severity)
```

7 Implementation of a Dense DNN

A dense deep neural network (DNN) for severity prediction is made. Using the `tfruns` package, we will determine which of the following architectures provides the best accuracy:

- Model 1: two hidden layers of 10 nodes each.
- Model 2: three hideen layers of 20, 10 and 5 nodes respectively.
- adding 40% dropout between hidden layers.
- adding l2 regularization on hidden layers.

7.1 Using One Output Node for Classification

Since we have one output node, we should use the *sigmoid* activation function in the output and the *binary_crossentropy* loss function. Moreover, I will be testing model 1 and model 2, both with and without dropout regularization and l2 regularization (in total 6 models).

```
# Using tfruns to test all the alternative we are given in the task description.
for (model in 1:6){
  training_run('neural_net.R', flags = c(model = model))
}
```

The code that I used to build the different neural networks is displayed below.

```
# Set hyperparameter flags.
FLAGS <- flags(
  flag_integer("model", 1)
)

# Defining the model and layers.
if (FLAGS$model == 1){
  model <- keras_model_sequential() %>%
    layer_dense(units = 10, activation = 'relu', input_shape = c(ncol(x_train))) %>%
    layer_dense(units = 10, activation = 'relu') %>%
    layer_dense(units = 1, activation = 'sigmoid')
```

```
} else if (FLAGS$model == 2){
  model <- keras_model_sequential() %>%
    layer_dense(units = 20, activation = 'relu', input_shape = c(ncol(x_train))) %>%
    layer_dense(units = 10, activation = 'relu') %>%
    layer_dense(units = 5, activation = 'relu') %>%
    layer_dense(units = 1, activation = 'sigmoid')
} else if (FLAGS$model == 3){
  # Model 1 with dropout regularization between the layers.
  model <- keras_model_sequential() %>%
    layer_dense(units = 10, activation = 'relu', input_shape = c(ncol(x_train))) %>%
    layer_dropout (rate = 0.4) %>%
    layer_dense(units = 10, activation = 'relu') %>%
    layer_dropout (rate = 0.4) %>%
    layer_dense(units = 1, activation = 'sigmoid')
} else if (FLAGS$model == 4){
  # Model 2 with dropout regularization between the layers.
  model <- keras_model_sequential() %>%
    layer_dense(units = 20, activation = 'relu', input_shape = c(ncol(x_train))) %>%
    layer_dropout (rate = 0.4) %>%
    layer_dense(units = 10, activation = 'relu') %>%
    layer_dropout (rate = 0.4) %>%
    layer_dense(units = 5, activation = 'relu') %>%
    layer_dropout (rate = 0.4) %>%
    layer_dense(units = 1, activation = 'sigmoid')
} else if (FLAGS$model == 5){
  # Model 1 with l2 regularization on the hidden layers.
  model <- keras_model_sequential() %>%
    layer_dense(units = 10, activation = 'relu', input_shape = c(ncol(x_train)), kernel_regularizer = re
    layer_dense(units = 10, activation = 'relu', kernel_regularizer = regularizer_l2(l = 0.01)) %>%
    layer_dense(units = 1, activation = 'sigmoid')
} else (FLAGS$model == 6){
  # Model 2 with l2 regularization on the hidden layers.
  model <- keras_model_sequential() %>%
    layer_dense(units = 20, activation = 'relu', input_shape = c(ncol(x_train)), kernel_regularizer = re
    layer_dense(units = 10, activation = 'relu', kernel_regularizer = regularizer_l2(l = 0.01)) %>%
    layer_dense(units = 5, activation = 'relu', kernel_regularizer = regularizer_l2(l = 0.01)) %>%
    layer_dense(units = 1, activation = 'sigmoid')
}

summary(model)

# compile (define loss and optimizer)
model %>% compile(loss = 'binary_crossentropy',
                  optimizer = optimizer_rmsprop(),
                  metrics = c('accuracy'))

# train (fit)
history <- model %>% fit(x_train, y_train, epochs = 100,
                         batch_size = 256, validation_split = 0.2)

# plot
plot(history)
```

```
# evaluate on training data.
score.train <- model %>% evaluate(x_train, y_train, verbose = 0)

cat('Train loss:', score.train[1], '\n')
cat('Train accuracy:', score.train[2], '\n')

# evaluate on testing data.
score.test <- model %>% evaluate(x_test, y_test, verbose = 0)

cat('Test loss:', score.test[1], '\n')
cat('Test accuracy:', score.test[2], '\n')
```

Comparison using the `tfruns` package can be done with the commands given in the code below

```
latest_run() # Show the latest trained model.
ls_runs() # Show all trained models.
ls_runs(metric_val_accuracy > 0.94, order = metric_val_accuracy) # Show a selection of trained models.
compare_runs() # Compare two runs (can be specified as an argument in the function).
```

```
ls_runs()
```

```
#> Data frame: 6 x 23
#>                     run_dir     eval_ metric_loss metric_accuracy
#> 1 runs/2022-03-11T17-42-13Z 0.6661904      0.6536          0.6397
#> 2 runs/2022-03-11T17-41-53Z 0.6422822      0.6357          0.6598
#> 3 runs/2022-03-11T17-41-32Z 0.6094625      0.6207          0.6397
#> 4 runs/2022-03-11T17-41-12Z 0.6067647      0.6146          0.6683
#> 5 runs/2022-03-11T17-40-51Z 0.5697196      0.5594          0.7188
#> 6 runs/2022-03-11T17-40-30Z 0.5923944      0.5714          0.7112
#>   metric_val_loss metric_val_accuracy
#> 1          0.6762              0.6002
#> 2          0.6519              0.6209
#> 3          0.6217              0.6002
#> 4          0.6125              0.6829
#> 5          0.5818              0.7145
#> 6          0.5824              0.6987
#> # ... with 17 more columns:
#> #   flag_model, epochs, epochs_completed, metrics, model, loss_function,
#> #   optimizer, learning_rate, script, start, end, completed, output,
#> #   source_code, context, type, NA.
```

According to the list of all runs, the two first models (without either of the regularization techniques) have the smallest loss and largest accuracy. Thus, comparing those two models reveals that model 2 has the largest test accuracy (even though all these models are terrible in practice).

```
compare_runs(ls_runs(metric_val_accuracy > 0.69, order = metric_val_accuracy))
```

8 Add Callbacks

In order to improve the selected model (Model 2) further, we add the following callbacks, before re-training the model and making predictions.

```
checkpoint.filepath <- "checkpoint.h5"
callbacks_list<-list(
  callback_early_stopping(
```

```
    monitor = "accuracy",
    patience=30
  ),
  callback_model_checkpoint(
    filepath = checkpoint.filepath,
    monitor = "accuracy",
    save_best_only = T
  ),
  callback_reduce_lr_on_plateau(
    monitor="accuracy",
    factor = 0.001,
    patience = 8
  )
)
```

```
#> Loaded Tensorflow version 2.7.1
```

The first callback is used to stop training when accuracy stops improving. The patience is set to 30, which means that the training is stopped after 45 consecutive epochs where accuracy is not improved.

The second callback is used to save the best model found during training. The weights of this model can then be loaded into the environment afterwards, to ensure that we are using the best model found during training.

The last callback is used to reduce the learning rate when the accuracy has stopped improving. The patience is set to 8, which means that the learning rate will be decreased by a factor (set to 0.1) after 15 consecutive epochs of no improvement in accuracy.

```
model <- keras_model_sequential() %>%
    layer_dense(units = 20, activation = 'relu', input_shape = c(ncol(x_train))) %>%
    layer_dense(units = 10, activation = 'relu') %>%
    layer_dense(units = 5, activation = 'relu') %>%
    layer_dense(units = 1, activation = 'sigmoid')

summary(model)
```

```
#> Model: "sequential"
#> _________________________________________________________________________
#>  Layer (type)                       Output Shape                     Param #
#> =========================================================================
#>  dense_3 (Dense)                    (None, 20)                       340
#>
#>  dense_2 (Dense)                    (None, 10)                       210
#>
#>  dense_1 (Dense)                    (None, 5)                        55
#>
#>  dense (Dense)                      (None, 1)                        6
#>
#> =========================================================================
#> Total params: 611
#> Trainable params: 611
#> Non-trainable params: 0
#> _________________________________________________________________________
```

```
# compile (define loss and optimizer)
model %>% compile(loss = 'binary_crossentropy',
                  optimizer = optimizer_rmsprop(),
```

```
                  metrics = c('accuracy'))

# train (fit)
set.seed(1)
history <- model %>% fit(x_train, y_train, epochs = 200,
                         batch_size = 256, validation_split = 0.2,
                         callbacks = callbacks_list)

final.model <- load_model_hdf5(checkpoint.filepath)
```

9 Predictions

```
# Predictions for one output node
y_pred <- model %>% predict(x_test) %>% `>`(0.5) %>% k_cast("int32")
y_pred <- as.array(y_pred)
y_pred2 <- as.array(final.model %>% predict(x_test) %>% `>`(0.5) %>% k_cast("int32"))
all.equal(y_pred, y_pred2)
```

```
#> [1] "Mean relative difference: 3"
```

```
(tab <- table("Predictions" = y_pred, "Labels" = y_test))
```

```
#>            Labels
#> Predictions   0   1
#>           0 232 100
#>           1 432 999
```

```
# accuracy in predictions (as shown with the "evaluate" above).
(tab[1]+tab[4])/sum(tab)
```

```
#> [1] 0.6982416
```

```
# Better to do this with the package "caret".
confusionMatrix(factor(y_pred), factor(y_test), positive = "1")
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction   0   1
#>          0 232 100
#>          1 432 999
#>
#>                Accuracy : 0.6982
#>                  95% CI : (0.6762, 0.7196)
#>     No Information Rate : 0.6234
#>     P-Value [Acc > NIR] : 2.646e-11
#>
#>                   Kappa : 0.2868
#>
#>  Mcnemar's Test P-Value : < 2.2e-16
#>
#>             Sensitivity : 0.9090
#>             Specificity : 0.3494
#>          Pos Pred Value : 0.6981
#>          Neg Pred Value : 0.6988
```

```
#>              Prevalence : 0.6234
#>          Detection Rate : 0.5666
#>    Detection Prevalence : 0.8117
#>       Balanced Accuracy : 0.6292
#>
#>        'Positive' Class : 1
#>
```

```
confusionMatrix(factor(y_pred2), factor(y_test), positive = "1")
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction   0   1
#>          0 225 102
#>          1 439 997
#>
#>                Accuracy : 0.6931
#>                  95% CI : (0.671, 0.7146)
#>     No Information Rate : 0.6234
#>     P-Value [Acc > NIR] : 5.119e-10
#>
#>                   Kappa : 0.2735
#>
#>  Mcnemar's Test P-Value : < 2.2e-16
#>
#>             Sensitivity : 0.9072
#>             Specificity : 0.3389
#>          Pos Pred Value : 0.6943
#>          Neg Pred Value : 0.6881
#>              Prevalence : 0.6234
#>          Detection Rate : 0.5655
#>    Detection Prevalence : 0.8145
#>       Balanced Accuracy : 0.6230
#>
#>        'Positive' Class : 1
#>
```

The accuracy is still not good for this model, even though it perhaps is the best model among the ones we tested.