PREDICTING HOUSING PRICES IN RSTUDIO USING KERAS AND TENSORFLOW

By Danielle Oberdier

SETUP

Download CRAN R Version 3.5.1 (64 bit)

Download RStudio

Download Anaconda

- Make sure to click yes to the Navigator Cheat Sheet, otherwise Anaconda Navigator will not be installed, and your packages will not work. If for some reason Anaconda Navigator does not download, open Anaconda Prompt and type in the command line: anaconda-navigator. This will load Anaconda Navigator. You can sign into cloud to ensure that Anaconda Navigator saves to your apps.
- Make sure the destination folder does not have a space in a name, because this could prevent some packages from being downloaded.

Once Anaconda Navigator is installed within your apps, open RStudio and execute the following commands:

```
install.packages(“reticulate”)

library(reticulate)

install.packages(“keras”)

library(keras)

install.packages(“tensorflow”)

library(tensorflow)
```

The reticulate package will allow you to import python’s interface into R with the following commands:

```
conda_install(‘r-tensorflow’,’absl-py)

keras::install_keras()
```

PREPARING THE DATA

Once the red stop sign disappears from your RStudio screen and you are given a new command line, you are ready to open your dataset. The Boston Housing dataset is built into keras and you can call it with the following command:

```
boston_housing <- dataset_boston_housing
```

The dataset will appear in your Environment with a list of two variables. We will partition the data for testing and training using the following commands:

```
c(train_data, train_labels) %<-%boston_housing$train
```

```
c(test_data, test_labels) %<-%boston_housing$test
```

For organizational purposes, we want to change our column labels from numbers to names with the following commands:

```
library(tibble)
```

```
column_names <- c('CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
                  'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT')
```

```
train_df <- as_tibble(train_data)
colnames(train_df) <- column_names
```

Before running the regression, we will need to normalize our data into values between 0 and 1 with the following command:

```
train_data <- scale(train_data)
```

```
col_means_train <- attr(train_data, "scaled:center")
col_stddevs_train <- attr(train_data, "scaled:scale")
test_data <- scale(test_data, center = col_means_train, scale = col_stddevs_train)
```

BUILDING AND TRAINING THE MODEL

We are now ready to start building our regression model using the following commands:

```
build_model <- function() {


model <- keras_model_sequential() %>%
    layer_dense(units = 64, activation = "relu",
                input_shape = dim(train_data)[2]) %>%
    layer_dense(units = 64, activation = "relu") %>%
    layer_dense(units = 1)

model %>% compile(
    loss = "mse",
    optimizer = optimizer_rmsprop(),
    metrics = list("mean_absolute_error")
  )

 model
}

model <- build_model()
```

We can use the following commands to visually document our training process:

```
print_dot_callback <- callback_lambda(
  on_epoch_end = function(epoch, logs) {
    if (epoch %% 80 == 0) cat("\n")
    cat(".")
  }
)

epochs <- 500
```

We then will fit the model and reliably store our calculations using the following commands:

```
history <- model %>% fit(
train_data,
 train_labels,
epochs = epochs,
validation_split = 0.2,
verbose = 0,
callbacks = list(print_dot_callback)
)
```

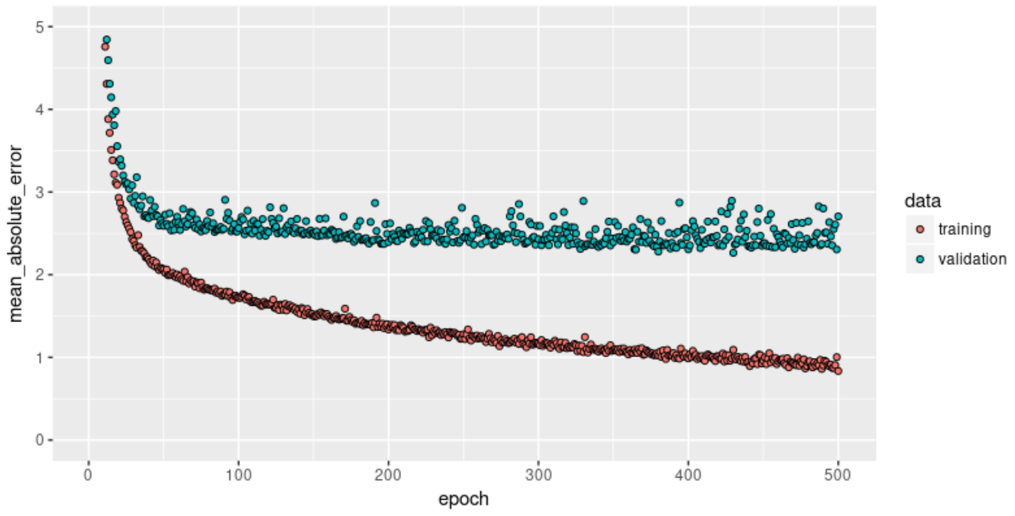
VISUALIZING THE MODEL

The package ggplot2 will allow us to visualize our model using the following commands:

```
install.packages("ggplot2")
```

```
library(ggplot2)plot(history, metrics = "mean_absolute_error", smooth = FALSE) +
coord_cartesian(ylim = c(0, 5))
```

In your plot screen, you should see an image similar to the one below:



For the purpose of accuracy, we can adjust the fit of our model to stop running when no improvements are being made using the following commands:

```
early_stop <- callback_early_stopping(monitor = "val_loss", patience = 20)
model <- build_model()
history <- model %>% fit(train_data,train_labels,epochs = epochs, validation_split =
0.2,verbose = 0, callbacks = list(early_stop, print_dot_callback))
```

Finally, we can use the following commands to identify the mean absolute error of our model:

```
c(loss, mae) %<-% (model %>% evaluate(test_data, test_labels, verbose = 0))
paste0("Mean absolute error on test set: $", sprintf("%.2f", mae * 1000))
```

USING THE MODEL TO MAKE PREDICTONS

We can use our model to make and display predictions for future housing prices using the following commands:

```
test_predictions <- model %>% predict(test_data)
```

```
test_predictions[ , 1]
```

```
[1]  9.159314 17.955666 20.573296 31.487156 25.231384 18.803967 27.139153 [8]
21.052799 18.904579 22.056618 19.140137 17.342262 15.233129 42.001091[15]
19.280727 19.559774 27.276485 20.737257 19.391312 38.450863 12.431134[22]
16.025173 19.910103 14.362184 20.846870 24.595688 31.234753 30.109112[29]
11.271907 21.081585 18.724422 14.542423 33.109241 25.842684 18.071476[36]
9.046785 14.701134 17.113651 21.169674 27.008324 29.676132 28.280304[43]
15.355518 41.252007 29.731274 24.258526 25.582203 16.032135 24.014944[50]
22.071520 35.658638 19.342590 13.662583 15.854269 34.375328 28.051319[57]
13.002036 47.801872 33.513954 23.775620 25.214602 17.864346 14.284246[64]
17.458893 22.757492 22.424841 13.578171 22.530212 15.667303  7.438343[71]
38.318726 29.219141 25.282124 15.476329 24.670732 17.125381 20.079552[78]
23.601147 34.540359 12.151771 19.177418 37.980789 15.576267 14.904464[85]
17.581717 17.851192 20.480953 19.700697 21.921551 31.415789 19.116734[92]
21.192280 24.934101 41.778465 35.113403 19.307007 35.754066 53.983509[99]
26.797831 44.472233 32.520882 19.591730
```