Week 8 - Data Science II

Phileas Dazeley-Gaist

22/02/2022

```r
# Applied Data Science II - Week 8
# # ------------------------------------------------------------------

# Today we are going to talk about NEURAL NETWORKS!
#
#
# # ------------------------------------------------------------------
#
# Load your libraries!
#
# # ------------------------------------------------------------------

library(ISLR2)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(xgboost)
```

```
##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

```r
library(torch)
library(luz) # high-level interface for torch
library(torchvision) # for datasets and image transformation
library(torchdatasets) # for datasets we are going to use
library(zeallot)

# do these not work? Then you'll have to install them!

# install.packages(c("glmnet","torch","luz","torchvision","torchdatasets","zeallot"))
```

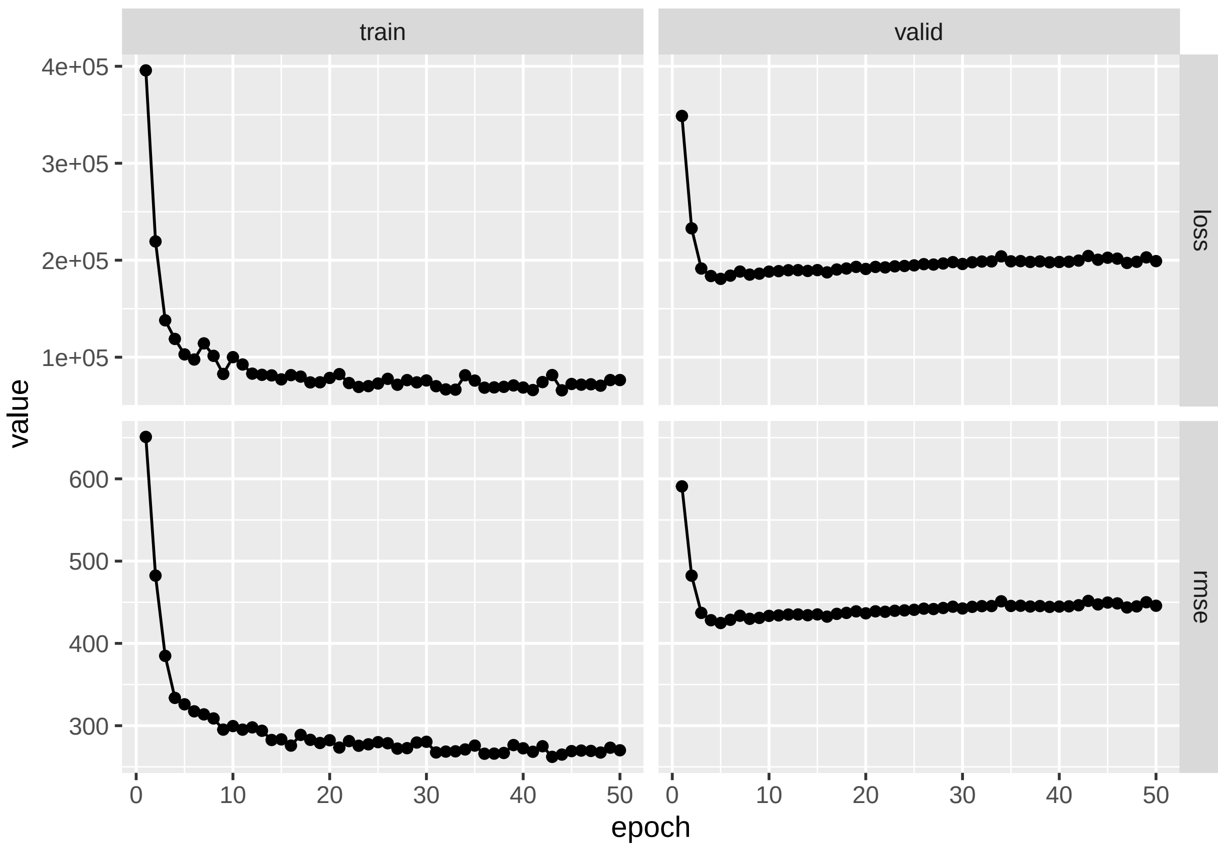
```r
# # ----------------------------------------------------------------
#
# Lets refresh our memory of stuff with the Hitters dataset! This is an example
# of a regression use case
#
# # ----------------------------------------------------------------

# let's setup our hitters data...

attach(Hitters)
hitters <- na.omit(Hitters)

hitters_indx <- createDataPartition(hitters$Salary, p = 0.75, list = FALSE)
hitters_train <- hitters[hitters_indx,]
hitters_test <- hitters[-hitters_indx,]
```

```r
# let's fit a simple linear model ...

lm_model <- lm(Salary ~ ., data = hitters_train)
lm_pred <- predict(lm_model, hitters_test)

# calculate RMSE!
with(hitters_test, sqrt(mean(lm_pred - Salary)^2))
```

```
## [1] 3.498771
```

```
# now let's try a neural net approach!

# what does the code below do:
# We now return to our neural network. The object modnn has a single hidden layer with 200 hid
# and a ReLU activation function. It then has a dropout layer, in which a random 40% of the 20
# from the previous layer are set to zero during each iteration of the stochastic gradient
# descent algorithm. Finally, the output layer has just one unit with no activation function,
# indicating that the model provides a single quantitative output.

modnn <- nn_module(
  initialize = function(input_size) {
    # set ourselves up with 200 hidden nodes
    self$hidden <- nn_linear(input_size, 200)
    # use a RELU function. What's ReLU? It's a linear function that will output
    # a postiive value if it's positive - or 0 otherwise!
    self$activation <- nn_relu()
    # Set our dropout function. What's dropout?
    # This lets us automatically drop nodes from our model to help avoid overfitting
    self$dropout <- nn_dropout(0.4)
    #
    self$output <- nn_linear(200, 1)
  },
  forward = function(x) {
    x %>%
      self$hidden() %>%
      self$activation() %>%
      self$dropout() %>%
      self$output()
  }
)

# next we build a model.matrix object, just like previously!
# this time, though, we use scale() to center the numeric data points

x_train <- model.matrix(Salary ~ . - 1, data = hitters_train) %>% scale()
y_train <- hitters_train$Salary
x_test <- model.matrix(Salary ~ . - 1, data = hitters_test) %>% scale()
y_test <- hitters_test$Salary

# now we go back and adjust that modnn object with further data to describe the fitting proced

modnn <- modnn %>%
  setup(
    loss = nn_mse_loss(),
    optimizer = optim_rmsprop,
    metrics = list(luz_metric_rmse())
  ) %>%
```

```r
  set_hparams(input_size = ncol(x_train))

# now we generate the actual neural network fit
fitted <- modnn %>%
  fit(
    data = list(x_train, matrix(y_train, ncol = 1)),
    valid_data = list(x_test, matrix(y_test, ncol = 1)),
    epochs = 50
  )

# let's look at the fitted plot ...
plot(fitted) # y is the RMSE
```




```r
# and let's calculate the MAPE and RMSE
nn_pred <- predict(fitted, x_test)

# RMSE

sqrt(mean(y_test - nn_pred)^2)
```

```
## torch_tensor
## 43.3095
## [ CPUFloatType{} ]
```