Deep Learning Ch. 10 Exercises

Lucy L.

2024-02-07

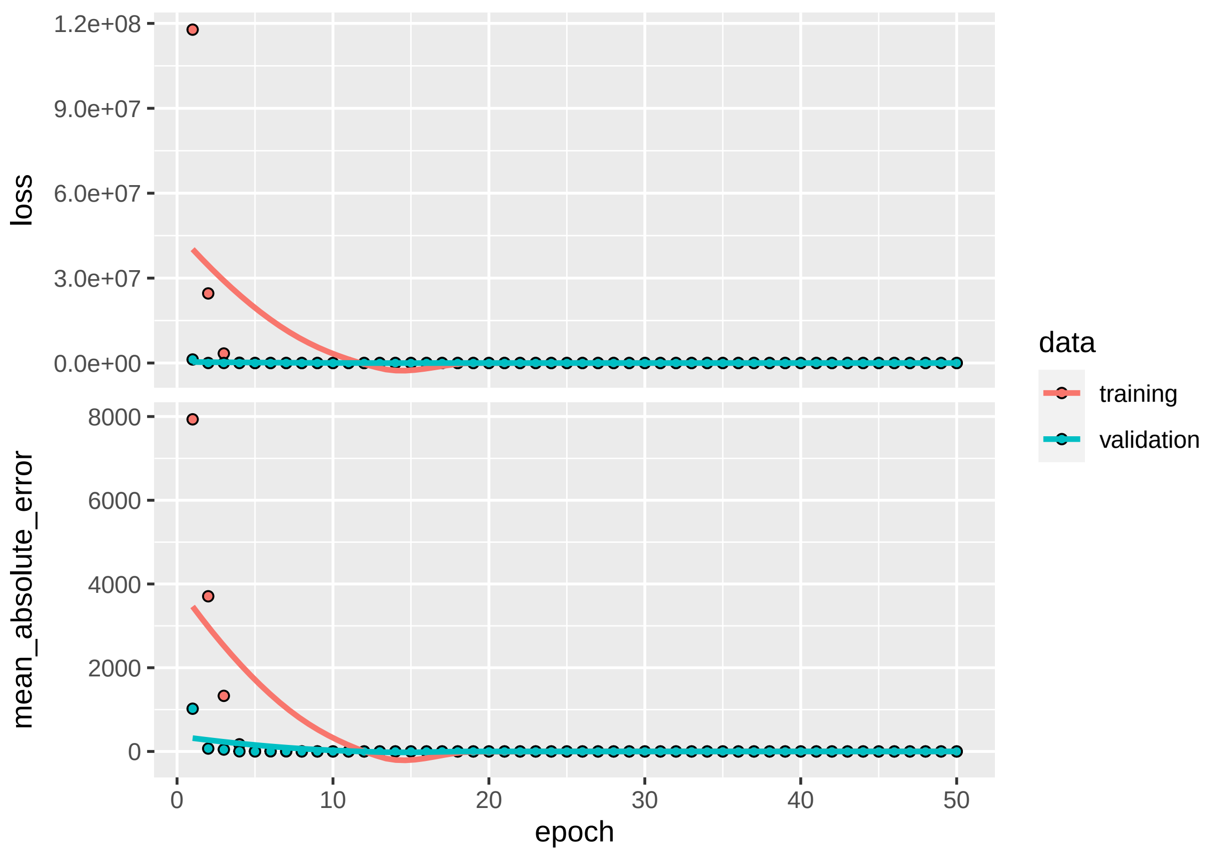
```r
library(ISLR2)
library(keras)
library(tensorflow)
```

Exercise 7

Fit a neural network to the `Default` data. Use a single hidden layer with 10 units, and dropout regularization. Have a look at Labs 10.9.1–10.9.2 for guidance. Compare the classification performance of your model with that of linear logistic regression.

```r
set.seed(1)
default = Default

default$default <- as.integer(default$default == "Yes")
default$student <- as.integer(default$student == "Yes")
x = scale(model.matrix(default ~ . - 1, data = default))
y = default$default

train_index <- sample(1:nrow(default), 0.7 * nrow(default))
train_data <- default[train_index, ]
test_data <- default[-train_index, ]
train_x = default[train_index, 2:4]
train_y = default[train_index, 1]
test_x = default[-train_index, 2:4]
test_y = default[-train_index, 1]

# fitting neural network
modnn = keras_model_sequential() %>%
  layer_dense(units = 10, activation = "relu", input_shape = 3) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 1)

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

```r
# display MAE for train and test data
plot(history)
```



```r
# predicting from the final model
npred = predict(modnn, as.matrix(test_x))
```

```
## 94/94 - 0s - 103ms/epoch - 1ms/step
```

```r
mean(abs(test_y - npred))
```

```
## [1] 0.0886671
```

```r
# comparison to logistic regression model:

# logistic regression model
logistic_model <- glm(default ~ ., data = train_data, family = binomial)

# predictions
predictions <- predict(logistic_model, newdata = test_data, type = "response")
predictions <- ifelse(predictions > 0.5, 1, 0)

# MAE of logistic regression
mean(abs(predictions - test_y))
```

```
## [1] 0.02766667
```

The mean absolute error of the neural network is about 0.102 (for this instance–set.seed will not affect the estimate), while the MAE for logistic regression was about 0.027. Thus, logistic regression outperformed the NN model.

Exercise 9

Fit a lag-5 auto regressive model to the NYSE data, as described in the text and Lab 10.9.6. Refit the model with a 12-level factor representing the month. Does this factor improve the performance of the model?

```r
nyse = NYSE

# generating 12-level factor for month
nyse$month <- as.factor(format(as.Date(NYSE$date), "%m"))

# setting up data and standardizing the variables
xdata <- data.matrix(nyse[, c("DJ_return", "log_volume",
                              "log_volatility", "month")])
istrain <- nyse[, "train"]
xdata <- scale(xdata)

# function that takes as input a data matrix and a lag L, and returns a lagged
# version of the matrix
lagm <- function(x, k = 1) {
  n <- nrow(x)
  pad <- matrix(NA, k, ncol(x))
  rbind(pad, x[1:(n - k), ])
}

# obtaining 5 lags
arframe <- data.frame(log_volume = xdata[, "log_volume"],
                      L1 = lagm(xdata, 1), L2 = lagm(xdata, 2),
                      L3 = lagm(xdata, 3), L4 = lagm(xdata, 4),
                      L5 = lagm(xdata, 5))

# removing missing values
arframe <- arframe[-(1:5), ]
istrain <- istrain[-(1:5)]

# fitting AR model to the training data and obtaining predictions
arfit <- lm(log_volume ~ ., data = arframe[istrain, ])
arpred <- predict(arfit, arframe[!istrain, ])

# calculating R-squared
V0 <- var(arframe[!istrain, "log_volume"])
1- mean((arpred - arframe[!istrain, "log_volume"])^2) / V0
```

```
## [1] 0.4174302
```

The new AR model including the `month` factor has an R-squared of about 0.417. This is slightly better than the R-squared for the AR model without the `date` factor, which was about 0.413, but worse than the R-squared for the AR model including the date factor as `day of the week`, which was about 0.459.

Exercise 10

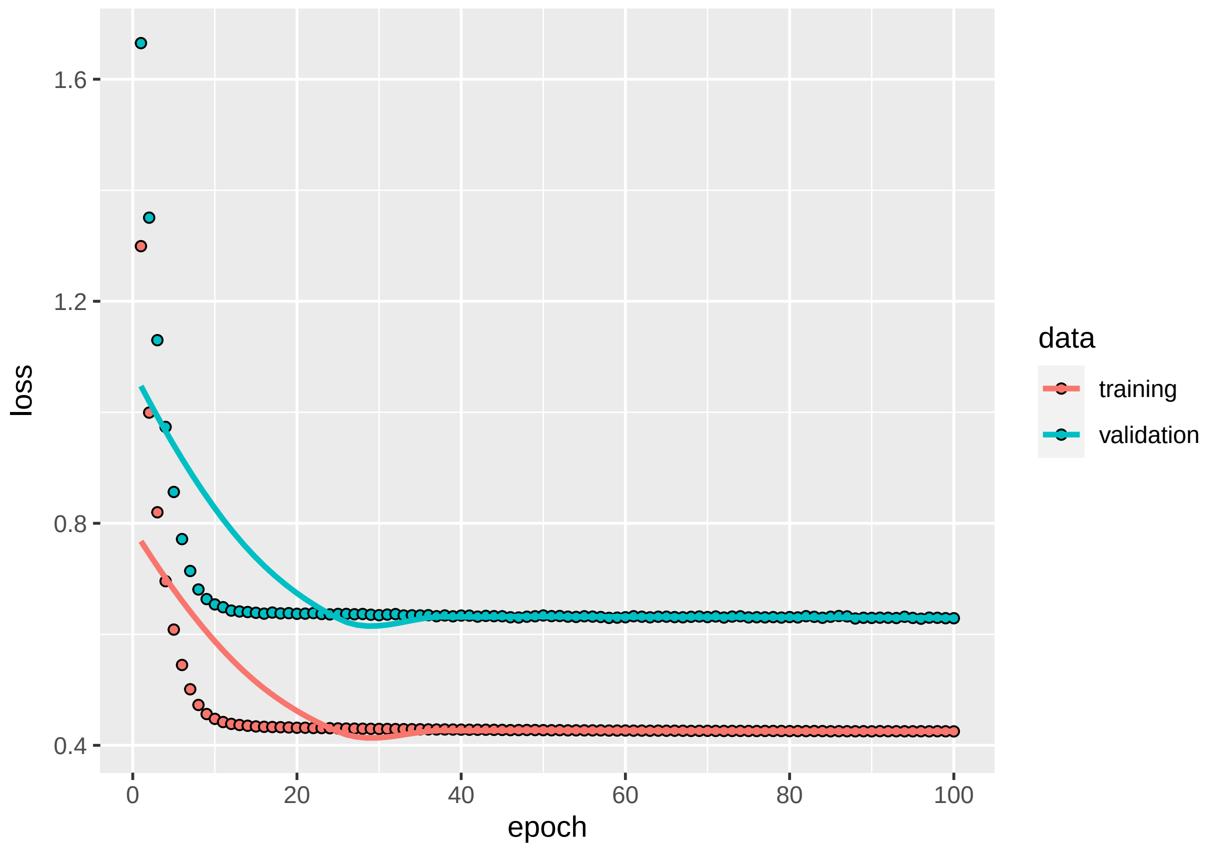
In Section 10.9.6, we showed how to fit a linear AR model to the NYSE data using the `lm()` function. However, we also mentioned that we can "flatten" the short sequences produced for the RNN model in order to fit a linear AR model. Use this latter approach to fit a linear AR model to the `NYSE` data. Compare the test R^2 of this linear AR model to that of the linear AR model that we fit in the lab. What are the advantages/disadvantages of each approach?

```r
# reshape data in order to fit RNN
n <- nrow(arframe)
xrnn <- data.matrix(arframe[, -1])
xrnn <- array(xrnn, c(n, 3, 5))
xrnn <- xrnn[,, 5:1]
xrnn <- aperm(xrnn, c(1, 3, 2))
dim(xrnn)
```

```
## [1] 6046    5    3
```

```r
# performing flattening
model <- keras_model_sequential() %>%
  layer_flatten(input_shape = c(5, 3)) %>%
  layer_dense(units = 1)

model %>% compile(optimizer = optimizer_rmsprop(), loss = "mse")
```

```r
plot(history)
```



```r
# calculating R-squared
kpred <- predict(model, xrnn[!istrain,, ])
```

```
## 56/56 - 0s - 59ms/epoch - 1ms/step
```

```r
1- mean((kpred - arframe[!istrain, "log_volume"])^2) / V0
```

```
## [1] 0.4033225
```

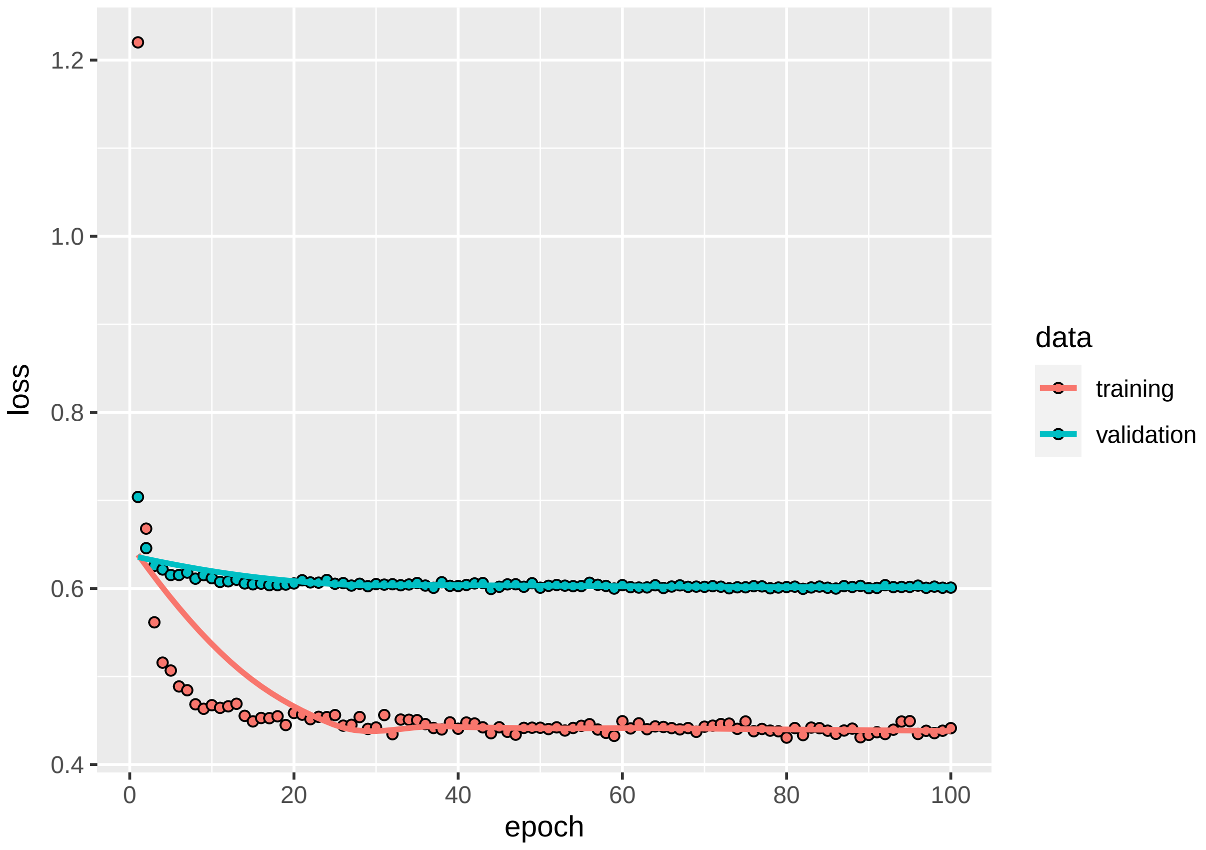
The R-squared for the AR linear model produced using flattening has been calculated to be about 0.4. This performance is worse compared to the previous AR model generated using the same factors but without flattening, which had an R-squared of about 0.417. This is also worse than the two AR models generated in the lab, whose R-squared values are listed in the previous exercise.

Exercise 11

Repeat the previous exercise, but now fit a nonlinear AR model by "flattening" the short sequences produced for the RNN model.

```r
x.10 <- model.matrix(log_volume ~ . - 1, data = arframe)

# generating non-linear model WITH flattening
arnnd <- keras_model_sequential() %>%
  layer_flatten() %>%
  layer_dense(units = 32, activation = 'relu',
              input_shape = ncol(x.10)) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 1)

# fitting model
arnnd %>% compile(loss = "mse", optimizer = optimizer_rmsprop())
```

```r
# plotting
plot(history)
```



```r
# making predictions
npred <- predict(arnnd, x.10[!istrain, ])
```

```
## 56/56 - 0s - 67ms/epoch - 1ms/step
```

```r
1- mean((arframe[!istrain, "log_volume"] - npred)^2) / V0
```

```
## [1] 0.4299462
```

The R-squared of the nonlinear AR model with flattening is about 0.427.

Exercise 12

Consider the RNN fit to the `NYSE` data in Section 10.9.6. Modify the code to allow inclusion of the variable `day_of_week`, and fit the RNN. Compute the test R^2.

```r
arframed <- data.frame(day = NYSE[-(1:5), "day_of_week"], arframe)
arfitd <- lm(log_volume ~ ., data = arframed[istrain, ])
arpredd <- predict(arfitd, arframed[!istrain, ])
1- mean((arpredd - arframe[!istrain, "log_volume"])^2) / V0
```

```
## [1] 0.4643177
```

The test R-squared for this model including the factor `day of the week` is about 0.464.