Project 3

Anton Holm

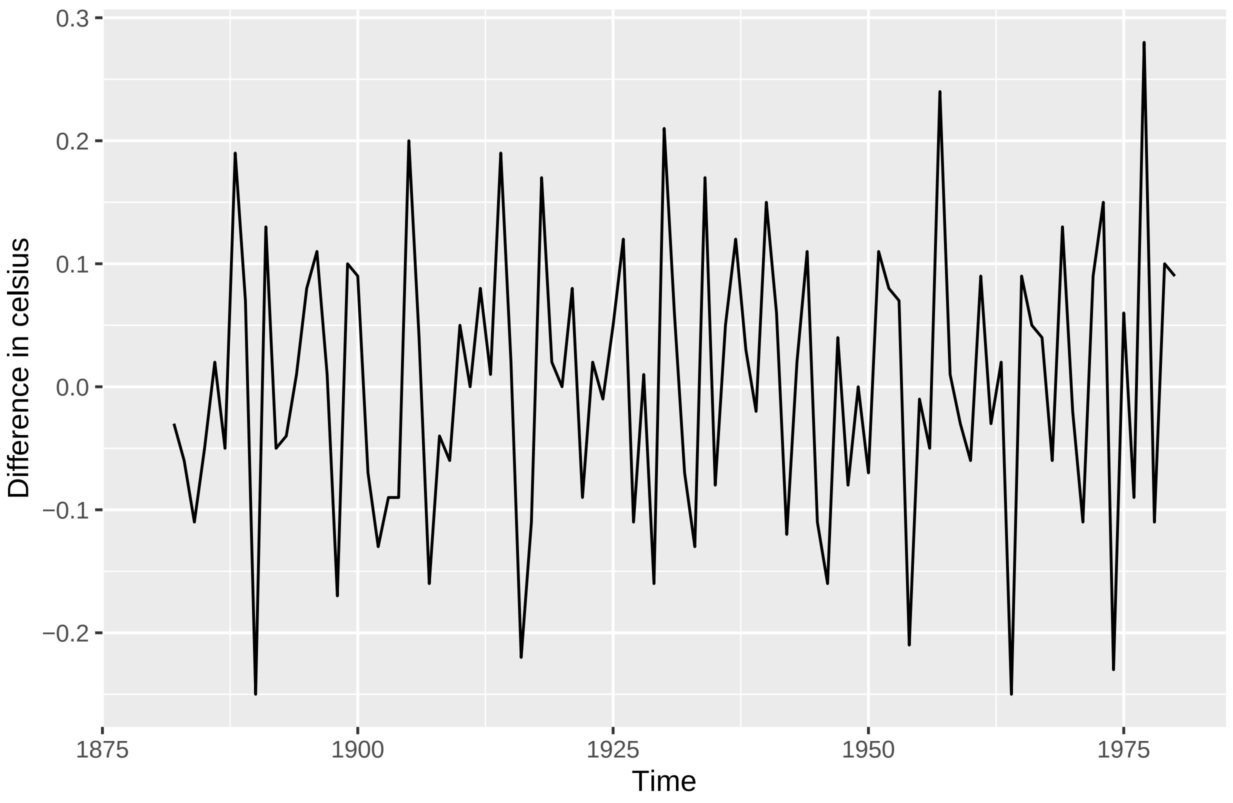
2022-02-28

First of all we get the stationary temperature instead of the non-stationary ones as suggested in the project description. Below we can see part of the data.

```
##   Time    V2   y_t y_t-1
## 1 1882 -0.11 -0.03  0.08
## 2 1883 -0.17 -0.06 -0.03
## 3 1884 -0.28 -0.11 -0.06
## 4 1885 -0.33 -0.05 -0.11
## 5 1886 -0.31  0.02 -0.05
## 6 1887 -0.36 -0.05  0.02
```

Next we divide the data into training and test data maintaining the continuous segment of the time series. Since all data past year 1980 is the test data, we should keep this hidden. Thus, the data I want to model would be the training data as seen in Figure 1.

Figure 1: Difference in global temperature 1880−1980



Next, I normalize my data, making sure it lies between -1 and 1 to fit the tanh activation function. To do this I use the code from "http://rwanjohi.rbind.io/2018/04/05/time-series-forecasting-using-lstm-in-r/".

The response variable for time t would then be $y(t)$, the temperature difference from time $t-1$ to time t. The predictor variables would be the sequence of temperature differences from previous year (as many as needed), e.g. $y(t-1), y(t-2), ...$ From the instruction, we only use $y(t-1)$ as our input.

Now it is time to build our LSTM model. I will use a one-dimensional input with only $y(t-1)$ as a predictor. I use a batch-size of 1. Since I only use $y(t-1)$ as a predictor, i.e. one-step time lagg, $\tau = 1$. Further more we only have 1 "feature", i.e. 1 year temperature difference. This is the 3-D vector that goes into the batch_input_shape argument of the lstm layer below. Since this is a regression problem I use MSE as the loss function. The Forget layer, the layer that decides what new information to store in the new cell and the output layer all have sigmoid activation functions (red, blue and dashed red line in the attached schematic). This is done since sigmoidal push the value between 0 and 1, 0 meaning we completely ignore/forget about a certain attribute and 1 is we want to remember/learn everything about it. When updating the cell state I use tanh. See attached notes for a more detailed explanation about what every part of the cell does. Furthermore, we have one input and one output hence this is a one-to-one model. I also use a learning rate equal to 0.015 and decay of 0.000001. The learning rate was tested for multiple values where smaller values made the convergence very slow while too large values made the loss jump up and down a lot, probably meaning we are missing the local minimum by overshooting it. The decay was chosen in a similar fashion where we do not want to decay the learning rate too fast making it too small before we find a local minimum. Finally I use one hidden layer with one hidden unit and one output unit since this is regression. I also use a glorot initialization in the LSTM layer and tested with different seeds to make sure I did not start at a local minimum.

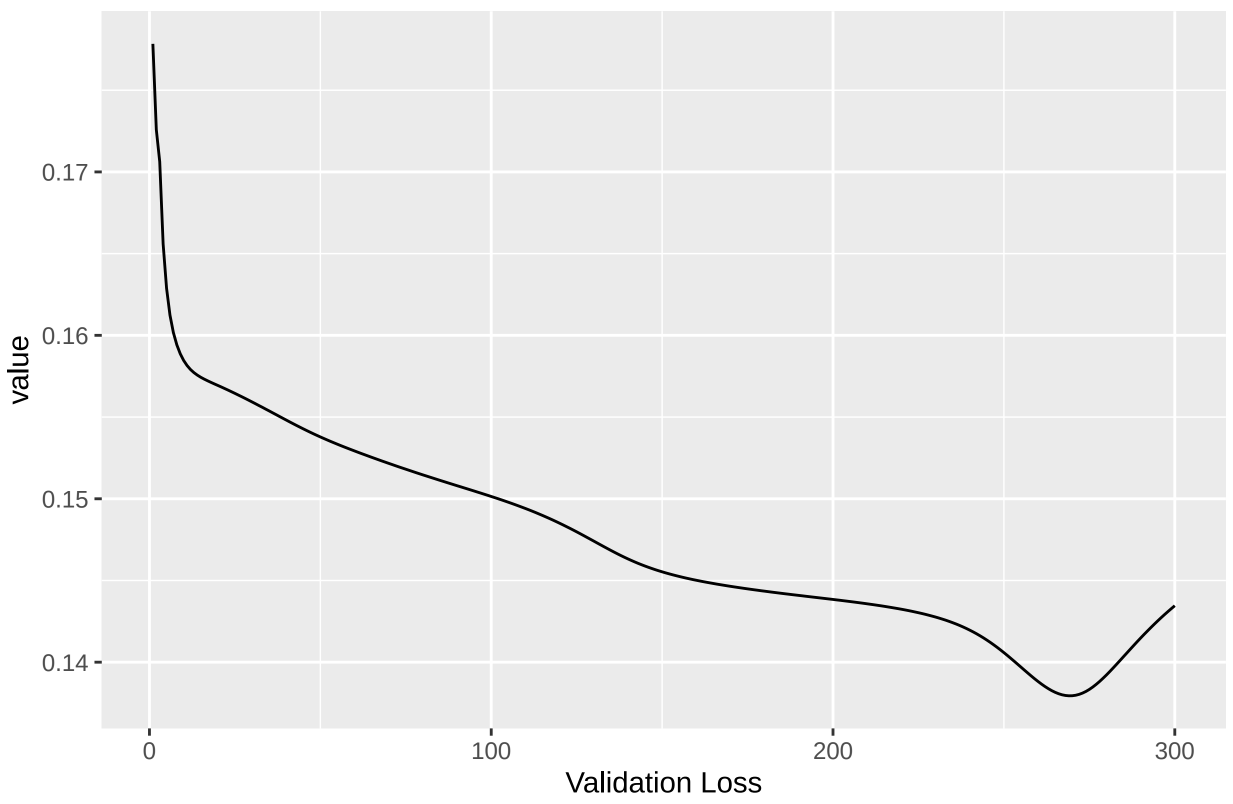
```
## Loaded Tensorflow version 2.7.0
```

```
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  lstm (LSTM)                        (1, 1)                          12
##
##  dense (Dense)                      (1, 1)                          2
##
## ================================================================================
## Total params: 14
## Trainable params: 14
## Non-trainable params: 0
## ________________________________________________________________________________
```

Below is a summary of the model structure where we have added a FFNN with unit activation function and 1 dimensional input and output layer.

Finally we train the model for 300 epochs where we see that the validation loss starts to increase again (See figure 2). At each epoch we need to reset the state of our cell and as such we use `reset:states` function after each epoch. I pick the model with the lowest validation loss.

Figure 2: Training history

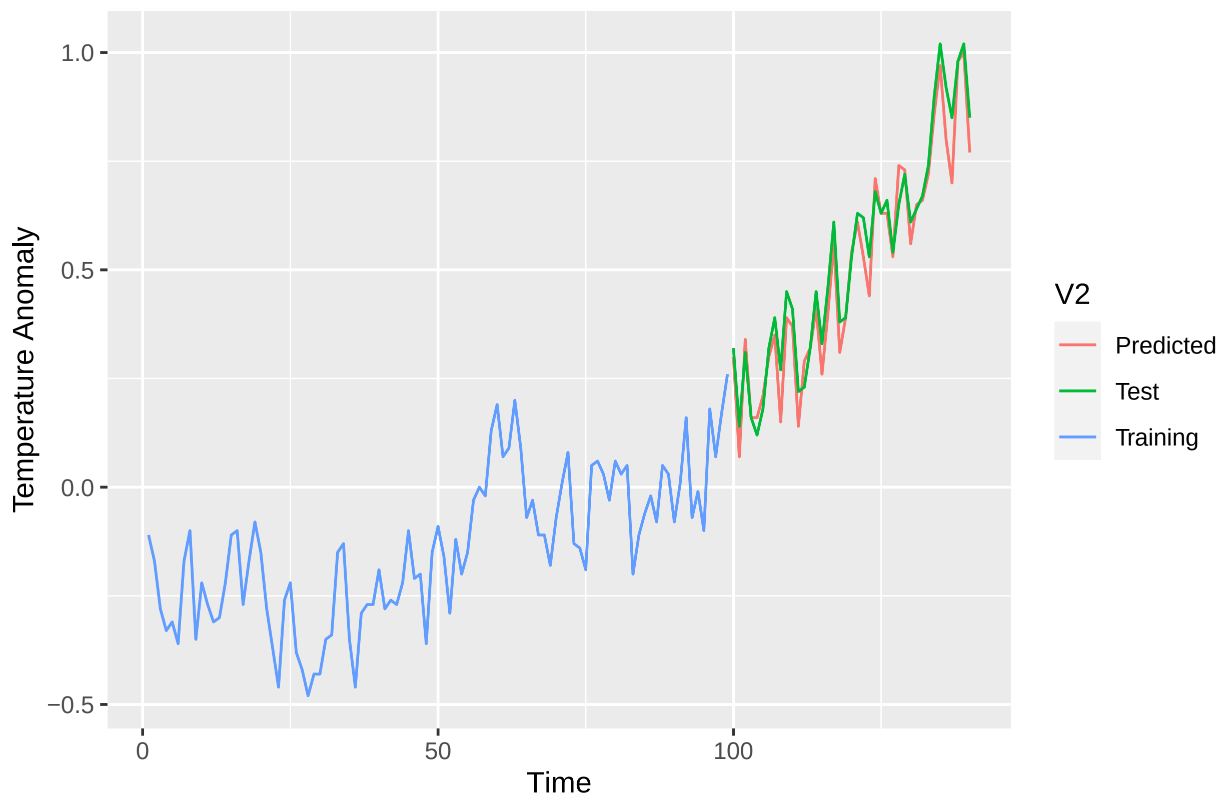


Finally we use our model to do predictions, remembering to scale back the values as well as transforming data back from $y(t)$ to $x(t)$ by the equation $y(t) = x(t) - x(t-1)$. Figure 3 shows the result of our model which coincides very well with the true outcome. When plotting our predictions against the testdata we see that the predictions are one year ahead. Hence we scale them back one year and see that the predictions are close to the true testdata. We also note that the predicted data should start one year after the test data since if $y(t)$ is the first datapoint in the testset, we would need $y(t-1)$ (only in trainingset) to predict it.

There could be several reasons for why we are able to predict so well. First of all, the data is very simple, only having one feature. Furtermore we have data dating quite far back. One of the reasons why an LSTM model works is since we have sequencial data. The LSTM model construct both a short-term and long-term memory making it very suitable for a time-series such as the one we work with.

```
## Warning: Removed 239 row(s) containing missing values (geom_path).
```

Figure 3: True and Estimated Temperature Anomaly



Code

```
#Year 1880 is removed since we don't have the difference of 1880 and 1879 and as such can not predict u
data <- read.table("Temperature_Anomaly_1880_2021.txt")
df <- data %>%
  mutate(`y_t` = c(NA, diff(V2))) %>%
  mutate(`y_t-1` = lag(`y_t`, default = 1)) %>%
  rename("Time" = V1) %>%
  slice(-c(1:2)) #Remove rows of years where we don't have input values (due to lagg)
saveRDS(df, file = "df.rdata")

training <- df %>% filter(Time <= 1980)
test <- df %>% filter(Time > 1980)
```

```
train_data_plot <- training %>%
  ggplot(aes(x = Time, y = y_t)) +
  geom_line() +
  labs(title = "Figure 1: Difference in global temperature 1880-1980", x = "Time", y = "Difference in c
  xlim(1880, 1980)

saveRDS(train_data_plot, file = "Training_plot.rdata")
```

```
#Scaling
scaling <- function(train, test){
  mini = -1
  maxi = 1
  x = train
  std_train = ((x - min(x) ) / (max(x) - min(x)))
  std_test  = ((test - min(x) ) / (max(x) - min(x)))

  scaled_train = std_train *(maxi - mini) + mini
  scaled_test = std_test *(maxi - mini) + mini
  return( list(scaled_train = as.vector(scaled_train), scaled_test = as.vector(scaled_test) ,scaler= c(
}

Scaled = scaling(training[,3:4], test[,3:4])
x_train = Scaled$scaled_train[, 2]
y_train = Scaled$scaled_train[, 1]

x_test = Scaled$scaled_test[, 2]
y_test = Scaled$scaled_test[, 1]

#Make the data fit the dimensions needed in the model
dim(x_train) <- c(length(x_train), 1, 1)
dim(y_train) <- c(length(y_train), 1, 1)
```

```
invert_scaling <- function(scaled, scaler, feature_range = c(0, 1)){
  min = scaler[1]
  max = scaler[2]
  t = length(scaled)
  mins = feature_range[1]
  maxs = feature_range[2]
  inverted_dfs = numeric(t)

  for( i in 1:t){
    X = (scaled[i]- mins)/(maxs - mins)
    rawValues = X *(max - min) + min
    inverted_dfs[i] <- rawValues
  }
  return(inverted_dfs)
}
```

```
set_random_seed(931031)
model1 <- keras_model_sequential()

model1 %>%
  layer_lstm(1, batch_input_shape = c(1, 1, 1), activation = "tanh", recurrent_activation = "sigmoid", 
             kernel_initializer = "glorot_uniform") %>%
  layer_dense(units = 1) #FFNN with linear activation function and 1 dimension input and output.

model1 %>% compile(
  loss = 'mean_squared_error',
  optimizer = optimizer_adam(learning_rate = 0.015, decay = 0.000001)
)

summary(model1)
```

```
set_random_seed(931031)

History = c()

for(i in 1:300){
  history <- model1 %>%
    fit(x = x_train,
        y = y_train,
        batch_size = 1,
        epochs = 1,
        verbose = 2,
        shuffle = FALSE,
        callbacks = callback_model_checkpoint(monitor = "loss",
                                              filepath = 'Models\\saved-model.{epoch:02d}-{loss:.5f}.hdf5',
                                              mode = "min", save_best_only = FALSE)) # maintain sequence of
  History[i] <- history$metrics[[1]]
  model1 %>%
    reset_states()
  }

History <- as_tibble(History) %>%
  rowid_to_column(var = "epochs") %>%
  ggplot(aes(x = epochs, y = value)) +
  geom_line() +
  labs(x = "Validation Loss", title = "Figure 2: Training history")
saveRDS(History, file = "History_plot.rdata")
best_model <- load_model_hdf5(filepath = "Models\\saved-model.01-0.13794.hdf5",
                              custom_objects = list("RandomNormal" = initializer_glorot_normal))
```

```r
#Make predictions and construct dataframe for graph
df <- readRDS("df.rdata")
dim(x_test) <- c(length(x_test), 1, 1)
predicted <- best_model %>%
  predict(x_test, batch_size = 1)
pred_rescaled <- invert_scaling(predicted, Scaled$scaler, c(-1,1))
pred_temp <- pred_rescaled + df$V2[100:140]

testing2 <- cbind(c(rep(NA, nrow(y_train)+2),df$V2[100:140]), "Test")
predicted2 <- cbind(c(rep(NA, length(y_train)+2), pred_temp), "Predicted")

#Graph to show how well model predicted
pred_graph <- as_tibble(cbind(data$V2[1:101],"Training")) %>%
  mutate(V1 = as.numeric(V1)) %>%
  add_row(V1 = rep(NA, 41), V2 = rep("Training", 41)) %>%
  add_row(V1 = as.numeric(testing2[,1]), V2 = testing2[,2]) %>%
  add_row(V1 = round(as.numeric(predicted2[,1]), 2), V2 = predicted2[,2]) %>%
  mutate(Time = rep(1880 + 0:141, 3)) %>%
  ggplot(aes(x = Time, y = V1, col = V2)) +
  geom_line() +
  labs(y = "Temperature Anomaly", title = "Figure 3: True and Estimated Temperature Anomaly")

saveRDS(pred_graph, file = "Predicted_Graph.rdata")
```