R Notebook

loading required library and data

library(tibble)  
library(readr)  
library(keras)  
jena\_climate\_2009\_2016 <- read\_csv("jena\_climate\_2009\_2016.csv")

## Parsed with column specification:  
## cols(  
## `Date Time` = col\_character(),  
## `p (mbar)` = col\_double(),  
## `T (degC)` = col\_double(),  
## `Tpot (K)` = col\_double(),  
## `Tdew (degC)` = col\_double(),  
## `rh (%)` = col\_double(),  
## `VPmax (mbar)` = col\_double(),  
## `VPact (mbar)` = col\_double(),  
## `VPdef (mbar)` = col\_double(),  
## `sh (g/kg)` = col\_double(),  
## `H2OC (mmol/mol)` = col\_double(),  
## `rho (g/m\*\*3)` = col\_double(),  
## `wv (m/s)` = col\_double(),  
## `max. wv (m/s)` = col\_double(),  
## `wd (deg)` = col\_double()  
## )

# as data is large we willuse only small portion of data  
data <- jena\_climate\_2009\_2016[1:20000,]  
data <- data.matrix(data[,-1])  
  
  
# creating training data for 15000 observation  
  
train\_data <- data[1:15000,]  
  
# scaling data on mean and std. dev.  
mean <- apply(train\_data, 2, mean)  
std <- apply(train\_data, 2, sd)  
  
data <- scale(data, center = mean, scale = std)

Generator yielding timeseries samples and their targets

generator <- function(data, lookback, delay, min\_index, max\_index,  
 shuffle = FALSE, batch\_size = 128, step = 6) {  
 if (is.null(max\_index))  
 max\_index <- nrow(data) - delay - 1  
 i <- min\_index + lookback  
 function() {  
 if (shuffle) {  
 rows <- sample(c((min\_index+lookback):max\_index), size = batch\_size)  
 } else {  
 if (i + batch\_size >= max\_index)  
 i <<- min\_index + lookback  
 rows <- c(i:min(i+batch\_size, max\_index))  
 i <<- i + length(rows)  
 }  
 samples <- array(0, dim = c(length(rows),  
 lookback / step,  
 dim(data)[[-1]]))  
 targets <- array(0, dim = c(length(rows)))  
 for (j in 1:length(rows)) {  
 indices <- seq(rows[[j]] - lookback, rows[[j]],  
 length.out = dim(samples)[[2]])  
 samples[j,,] <- data[indices,]  
 targets[[j]] <- data[rows[[j]] + delay,2]  
 }  
 list(samples, targets)  
 }  
}

Preparing the training, validation, and test generators

lookback <- 1440  
step <- 6  
delay <- 144  
batch\_size <- 128  
train\_gen <- generator(  
 data,  
 lookback = lookback,  
 delay = delay,  
 min\_index = 1,  
 max\_index = 15000,  
 shuffle = TRUE,  
 step = step,  
 batch\_size = batch\_size  
)  
val\_gen = generator(  
 data,  
 lookback = lookback,  
 delay = delay,  
 min\_index = 15001,  
 max\_index = 17000,  
 step = step,  
 batch\_size = batch\_size  
)  
test\_gen <- generator(  
 data,  
 lookback = lookback,  
 delay = delay,  
 min\_index = 17001,  
 max\_index = 19000,  
 step = step,  
 batch\_size = batch\_size  
)  
  
# to check how many steps to draw from val\_gen in order to see the entire validation set  
  
val\_steps <- (17000 - 15001 - lookback) / batch\_size  
  
# to check howmany steps to draw from test\_gen in order to see the entire test set  
test\_steps <- (19000 - 17001 - lookback) / batch\_size

Computing the common-sense baseline MAE

evaluate\_naive\_method <- function() {  
 batch\_maes <- c()  
 for (step in 1:val\_steps) {  
 c(samples, targets) %<-% val\_gen()  
 preds <- samples[,dim(samples)[[2]],2]  
 mae <- mean(abs(preds - targets))  
 batch\_maes <- c(batch\_maes, mae)  
 }  
 print(mean(batch\_maes))  
}  
(evaluate\_naive\_method())

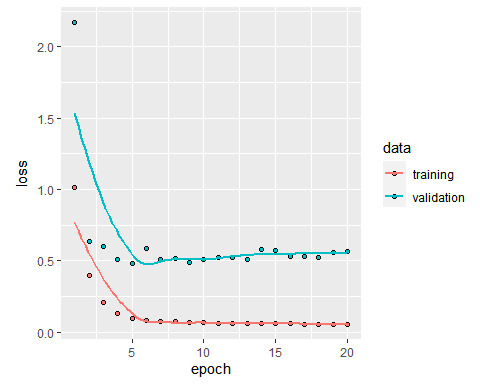
## [1] 0.4259496

## [1] 0.4259496

Training and evaluating a densely connected model

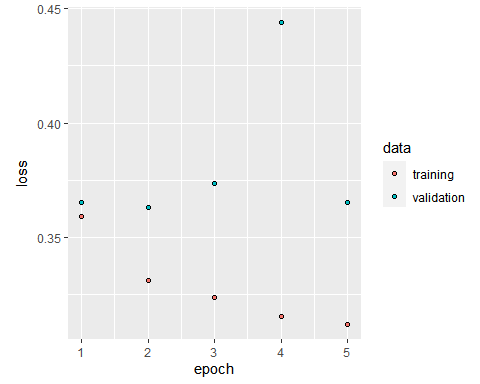
model <- keras\_model\_sequential() %>%  
 layer\_flatten(input\_shape = c(lookback / step, dim(data)[-1])) %>%  
 layer\_dense(units = 32, activation = "relu") %>%  
 layer\_dense(units = 1)  
  
model %>% compile(  
 optimizer = optimizer\_rmsprop(),  
 loss = "mae"  
)  
  
history <- model %>% fit\_generator(  
 train\_gen,  
 steps\_per\_epoch = 500,  
 epochs = 20,  
 validation\_data = val\_gen,  
 validation\_steps = val\_steps  
)  
  
plot(history)

## `geom\_smooth()` using formula 'y ~ x'

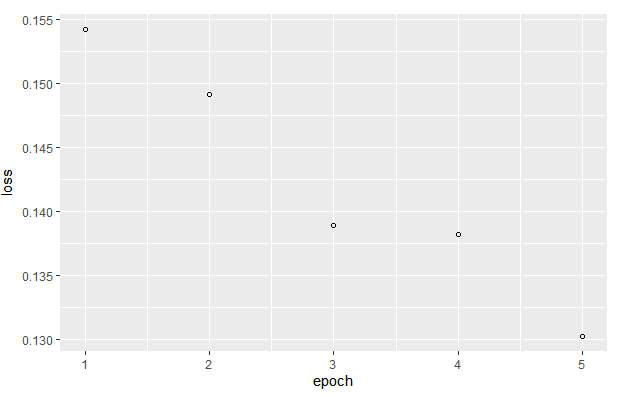


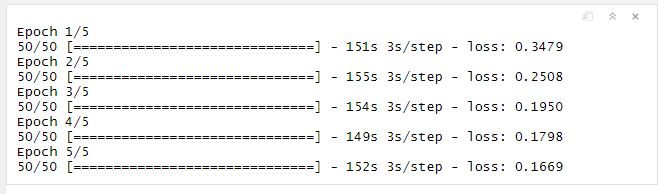
Training and evaluating a model with layer\_gru by chnaging numer of units

model <- keras\_model\_sequential() %>%   
 layer\_gru(units = 64,   
 dropout = 0.1,   
 recurrent\_dropout = 0.5,  
 return\_sequences = TRUE,  
 input\_shape = list(NULL, dim(data)[[-1]])) %>%   
 layer\_gru(units = 128, activation = "relu",  
 dropout = 0.1,  
 recurrent\_dropout = 0.5) %>%   
 layer\_dense(units = 1)  
model %>% compile(  
 optimizer = optimizer\_rmsprop(),  
 loss = "mae"  
)  
history <- model %>% fit\_generator(  
 train\_gen,  
 steps\_per\_epoch = 50,  
 epochs = 5,  
 validation\_data = val\_gen,  
 validation\_steps = val\_steps  
)  
  
plot(history)



model %>% fit\_generator(  
 test\_gen,  
 steps\_per\_epoch = 50,  
 epochs = 5,  
 validation\_steps = test\_steps  
)





From test result we see loss on test is .1669 which is less than baseline MAE of .45 which shows GRU model was able to reduce loss on partial data