LAPORAN TUGAS RNN DAN LSTM

TOPIK DALAM ANALISIS DATA DERET WAKTU

Oleh :

Yolla Faradhilla 05111950010029

Hafara Firdausi 05111950010040

Dosen Pengampu :

Dr. Ahmad Saikhu, S.Si., M.T.

MAGISTER INFORMATIKA

FAKULTAS ELEKTRO DAN INFORMATIKA CERDAS

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

SURABAYA

2020

Recurrent Neural Network (RNN)

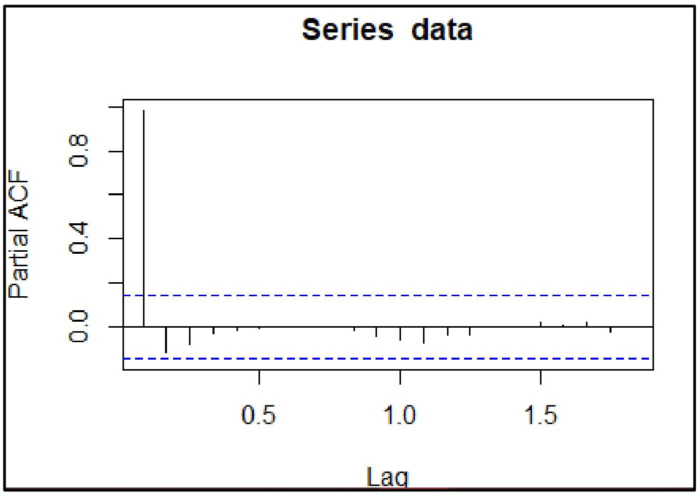
Forecasting the Whole Bird Spot Price of Chicken

- Data : data chicken dari package atsa

```
> data
Aug 2001 Sep 2001 Oct 2001 Nov 2001 Dec 2001 Jan 2002 Feb 2002 Mar 2002 Apr 2002 May 2002 Jun 2002
   65.58    66.48    65.70    64.33    63.23    62.94    62.92    62.73    62.50    63.35    63.80
Jul 2002 Aug 2002 Sep 2002 Oct 2002 Nov 2002 Dec 2002 Jan 2003 Feb 2003 Mar 2003 Apr 2003 May 2003
   64.21    64.11    64.04    63.00    61.90    61.49    62.27    63.13    63.86    63.53    64.60
Jun 2003 Jul 2003 Aug 2003 Sep 2003 Oct 2003 Nov 2003 Dec 2003 Jan 2004 Feb 2004 Mar 2004 Apr 2004
   65.99    67.50    68.50    69.23    68.57    68.36    68.98    69.58    71.59    73.09    74.75
May 2004 Jun 2004 Jul 2004 Aug 2004 Sep 2004 Oct 2004 Nov 2004 Dec 2004 Jan 2005 Feb 2005 Mar 2005
   76.59    79.63    80.94    80.10    78.16    76.00    74.71    73.60    73.44    73.75    73.88
Apr 2005 May 2005 Jun 2005 Jul 2005 Aug 2005 Sep 2005 Oct 2005 Nov 2005 Dec 2005 Jan 2006 Feb 2006
   74.00    74.29    74.48    74.75    74.77    75.19    74.38    72.69    71.21    69.86    69.18
Mar 2006 Apr 2006 May 2006 Jun 2006 Jul 2006 Aug 2006 Sep 2006 Oct 2006 Nov 2006 Dec 2006 Jan 2007
   68.29    67.52    67.87    68.98    69.90    70.42    70.69    69.65    69.00    69.35    71.33
Feb 2007 Mar 2007 Apr 2007 May 2007 Jun 2007 Jul 2007 Aug 2007 Sep 2007 Oct 2007 Nov 2007 Dec 2007
   73.77    76.37    78.10    79.52    80.75    81.17    81.27    81.55    79.75    77.77    76.85
Jan 2008 Feb 2008 Mar 2008 Apr 2008 May 2008 Jun 2008 Jul 2008 Aug 2008 Sep 2008 Oct 2008 Nov 2008
   77.25    79.15    81.23    82.04    83.46    85.71    88.25    88.42    88.40    87.54    86.93
```

Gambar 1.1 Sample Data Chicken

- PACF data



Gambar 1.2 PACF Data Chicken

- Lag : 1, 2, 3, dan 4

	Lag.1	Lag.2	Lag.3	Lag.4	data
Aug 2001	NA	NA	NA	NA	4.18
Sep 2001	4.18	NA	NA	NA	4.20
Oct 2001	4.20	4.18	NA	NA	4.19
Nov 2001	4.19	4.20	4.18	NA	4.16
Dec 2001	4.16	4.19	4.20	4.18	4.15
Jan 2002	4.15	4.16	4.19	4.20	4.14

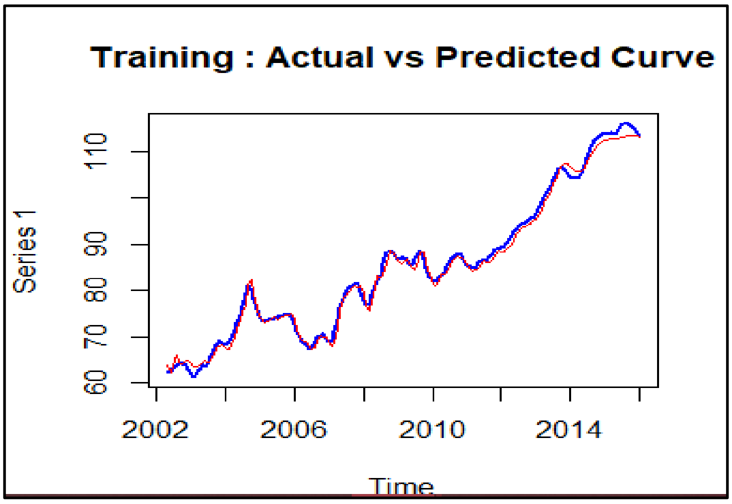
Gambar 1.3 Data pada Lag 1, 2, 3, dan 4

- RNN menggunakan package rnn dari CRAN
- Parameter RNN :
  - Seed : 256
  - Learning rate : 0.1
  - Hidden layer : 2 hidden layer dengan masing-masing 3 dan 5 node
  - Epoch : 3000
  - Network type : RNN
  - Sigmoid : Logistic

```
require(rnn)
set.seed(256)
model1<-trainr(Y=t(y_train),
               X=x_train,learningrate=0.1,
               hidden_dim= c(3,5),
               numepochs=3000,
               network_type="rnn",
               sigmoid="logistic")
```

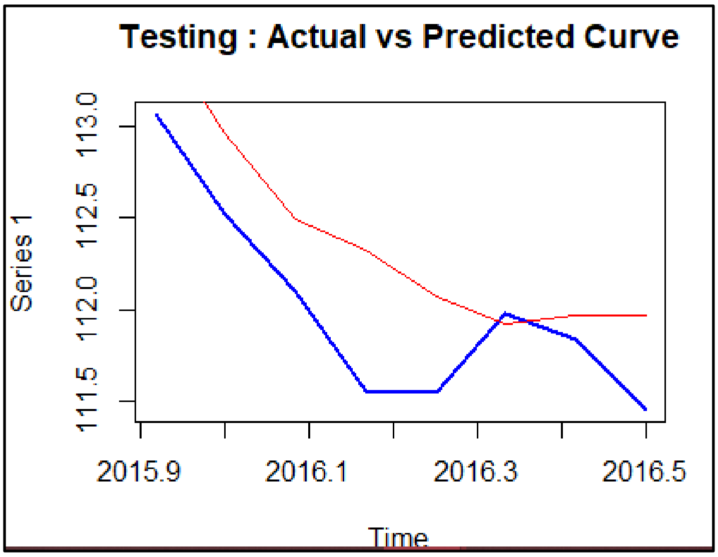
Gambar 1.4 Parameter RNN

- Hasil Training
  - Korelasi : 0.997
  - R^2 : 0.995
  - MSE : 1.511
  - MAPE : 0.016



Gambar 1.5 Hasil Training

- Hasil Testing
    - Korelasi : 0.9
    - R^2 : 0.81
    - MSE : 0.216
    - MAPE : 0.003



Gambar 1.6 Hasil Testing

Long Short-Term Memory (LSTM)

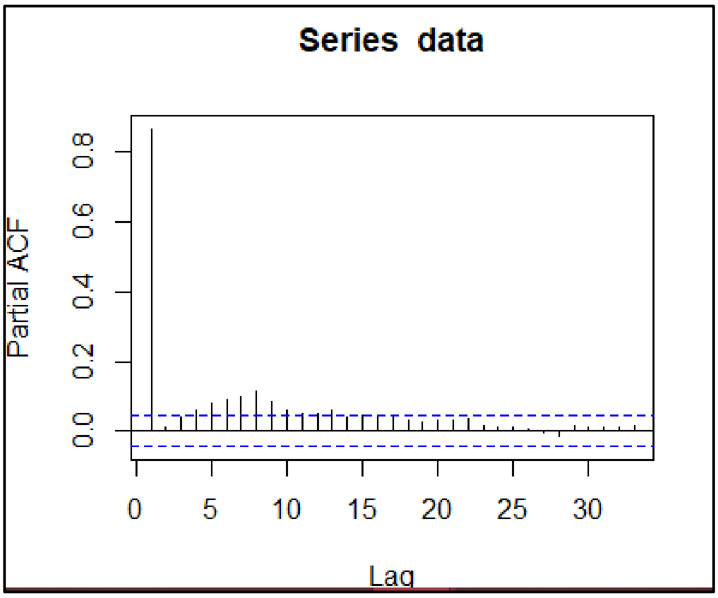
Forecasting Electrocardiogram Activity

- Data : data ecg dari package wmtsa

```
> head(data, n=50)
        1         2         3         4         5         6         7         8         9        10
-0.104773 -0.093136 -0.081500 -0.116409 -0.081500 -0.116409 -0.104773 -0.093136 -0.093136 -0.093136
       11        12        13        14        15        16        17        18        19        20
-0.093136 -0.093136 -0.093136 -0.093136 -0.058227 -0.093136 -0.104773 -0.093136 -0.081500 -0.093136
       21        22        23        24        25        26        27        28        29        30
-0.093136 -0.104773 -0.139682 -0.151318 -0.162955 -0.197864 -0.232773 -0.267682 -0.337500 -0.407318
       31        32        33        34        35        36        37        38        39        40
-0.418955 -0.453864 -0.488773 -0.500409 -0.535318 -0.546955 -0.535318 -0.535318 -0.523682 -0.523682
       41        42        43        44        45        46        47        48        49        50
-0.488773 -0.477136 -0.442227 -0.442227 -0.407318 -0.372409 -0.360773 -0.325864 -0.337500 -0.337500
```

Gambar 2.1 Sample Data ECG

- PACF data



Gambar 2.2 PACF Data ECG

- Lag : 1, 2, 3, dan 4

```
> head(round(x , 3 ) )
   Lag.1  Lag.2  Lag.3  Lag.4   data
1     NA     NA     NA     NA -0.105
2 -0.105     NA     NA     NA -0.093
3 -0.093 -0.105     NA     NA -0.082
4 -0.082 -0.093 -0.105     NA -0.116
5 -0.116 -0.082 -0.093 -0.105 -0.082
6 -0.082 -0.116 -0.082 -0.093 -0.116
```

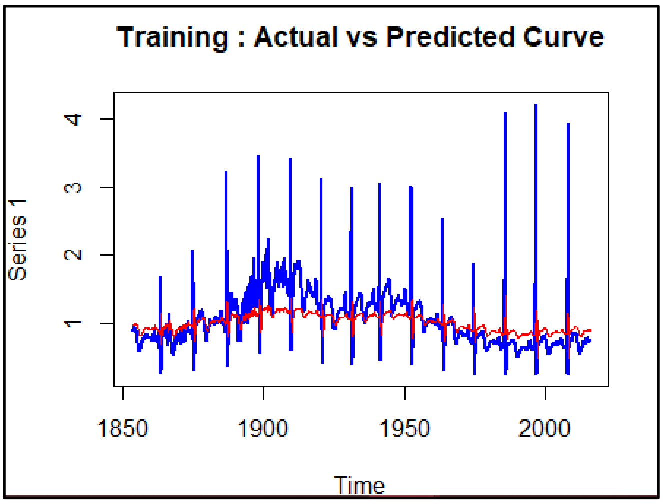
Gambar 2.3 Data pada Lag 1, 2, 3, dan 4

- LSTM menggunakan package rnn dari CRAN
- Parameter LSTM :
  - Seed : 256
  - Learning rate : 0.01
  - Hidden layer : 4 node
  - Epoch : 400
  - Momentum : 0.99
  - Network type : LSTM
  - Sigmoid : Tanh

```r
# specify model
require(rnn)
set.seed(256)
model1 <- trainr(Y = t(y_train),
                 X = x_train,
                 learningrate = 0.01,
                 hidden_dim = 4,
                 numepochs = 400,
                 momentum = 0.99,
                 network_type = 'lstm',
                 sigmoid = "tanh"
)
```

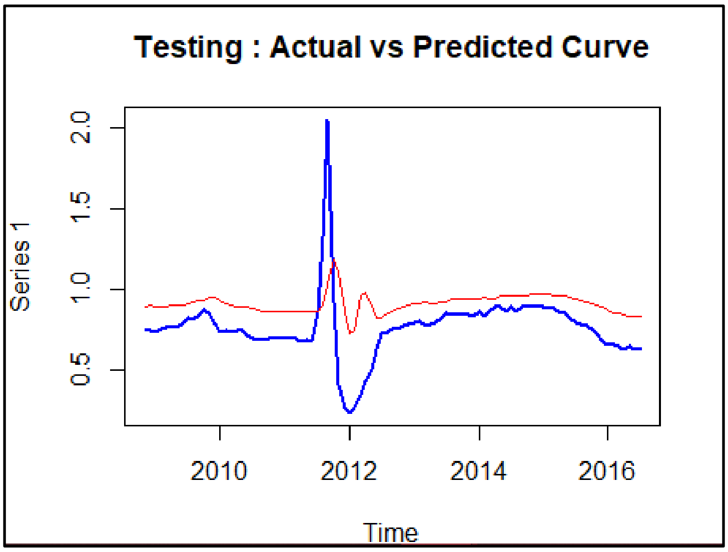
Gambar 2.4 Parameter LSTM

- Hasil Training
  - Korelasi : 0.787
  - R^2 : 0.619
  - MSE : 0.111
  - MAPE : 0.242



Gambar 2.5 Hasil Training

- Hasil Testing
    - Korelasi : 0.456
    - R^2 : 0.208
    - MSE : 0.055
    - MAPE : 0.239



Gambar 2.6 Hasil Testing

Predicting Eye Movements

In [1]:

```
# Load the necessary packages
library(zoo)
library(keras)
library(tensorflow)
```

```
Attaching package: 'zoo'


The following objects are masked from 'package:base':

    as.Date, as.Date.numeric
```

1. Load Data and Exploration

Data yang digunakan adalah data pergerakan mata dari package `crqa`

In [2]:

```
data("crqa", package = "crqa")
```

In [3]:

```
# datanya
typeof(RDts1)
```

'list'

In [4]:

```
# jumlah data
nrow(RDts1)
```

2000

In [5]:

```
head(RDts1)
tail(RDts1)
```

A data.frame: 6 × 1

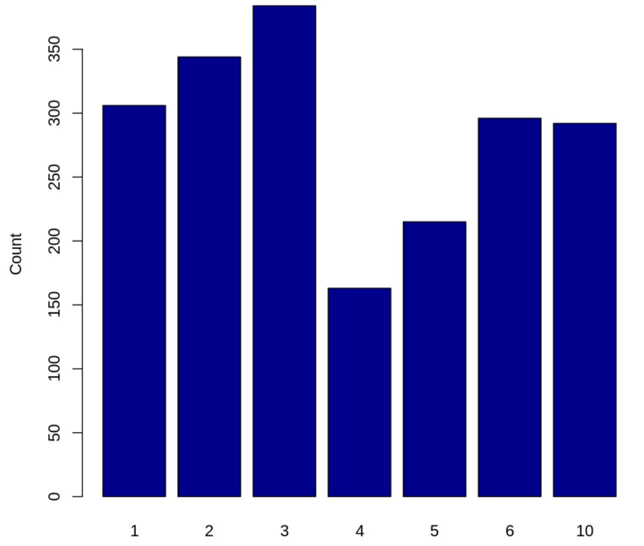
	V1
	<int>
1	10
2	2
3	2
4	10
5	10
6	10

A data.frame:
6 × 1

	V1
	<int>
1995	1
1996	1
1997	1
1998	1
1999	1
2000	1

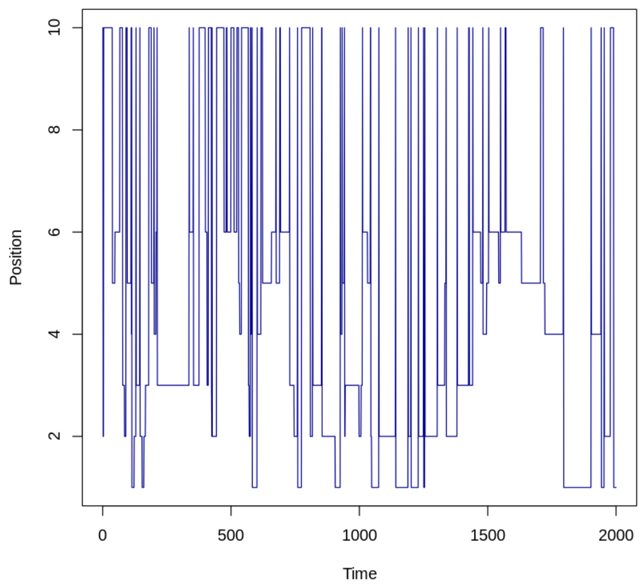
In [6]:

```
barplot(table(RDts1), ylab="Count", col="darkblue")
```



In [7]:

```
plot(as.ts(RDts1), col="darkblue", ylab="Position")
```



In [8]:

```
head(as.ts(RDts1))
```

10 · 2 · 2 · 10 · 10 · 10

2. Data Preparation

In [9]:

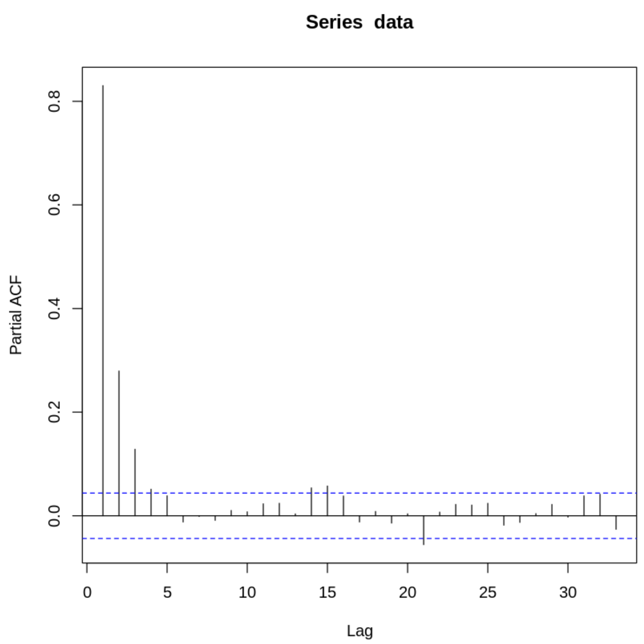
```
# change matrix to timeseries
data <- as.ts(RDts1)

head(data)
```

10 · 2 · 2 · 10 · 10 · 10

In [10]:

```
pacf(data)
```



2.1 Lagged

In [11]:

```
# lagged transform

# define function
lag_transform <- function(data, seq_len){
    for(i in 1:seq_len) {
        lagged <- c(rep(NA, i), data[1:(length(data)-i)])
        if(i == 1){
            DF <- as.data.frame(cbind(lagged, data))
        } else {
            DF <- as.data.frame(cbind(lagged, DF))
        }
    }
    return(DF)
}
```

```
# define parameter
seq_len = 1

supervised_data <- lag_transform(data, seq_len)
head(supervised_data, 10)
```

A data.frame: 10 × 2

	lagged	data
	<int>	<int>
1	NA	10
2	10	2
3	2	2
4	2	10
5	10	10
6	10	10
7	10	10
8	10	10
9	10	10
10	10	10

In [12]:

```
# remove NA value
supervised_data <- supervised_data[-(1:seq_len), ]

head(supervised_data, 10)
```

A data.frame: 10 × 2

	lagged	data
	<int>	<int>
2	10	2
3	2	2
4	2	10
5	10	10
6	10	10
7	10	10
8	10	10
9	10	10
10	10	10
11	10	10

2.2 Split Data into Train and Test

Dengan perbandingan train dan test 0.7 0.3

In [13]:

```
## split into train and test sets

total_data <- nrow(supervised_data)
jumlah_train_set <- round(total_data * 0.7, digits = 0)

train <- supervised_data[1:jumlah_train_set, ]
test  <- supervised_data[(jumlah_train_set+1):total_data,  ]

cat("Train set:", nrow(train), "\n")
```

```
cat("Test set:", nrow(test))
```

```
Train set: 1399
Test set: 600
```

2.3 Normalisasi Data

In [14]:

```
# normalize data

# define function
scale_data <- function(train, test, feature_range = c(0, 1)) {
  x <- train
  fr_min <- feature_range[1]
  fr_max <- feature_range[2]
  std_train <- ((x - min(x) ) / (max(x) - min(x)  ))
  std_test  <- ((test - min(x) ) / (max(x) - min(x)  ))

  scaled_train <- std_train *(fr_max -fr_min) + fr_min
  scaled_test <- std_test *(fr_max -fr_min) + fr_min

  return( list(scaled_train = as.vector(scaled_train), scaled_test = as.vector(scaled_test) ,scaler
= c(min =min(x), max = max(x))) )

}


Scaled = scale_data(train, test, c(-1, 1))
```

In [15]:

```
## function to inverse-transform
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)
}
```

2.4 Split Data into Input and Output

In [16]:

```
# split into input and output
# x_train = Scaled$scaled_train[, 1:seq_len]
# y_train = Scaled$scaled_train[, (seq_len+1)]

# x_test = Scaled$scaled_test[, 1:seq_len]
# y_test = Scaled$scaled_test[, (seq_len+1)]

x_train = Scaled$scaled_train[, 1]
y_train = Scaled$scaled_train[, 2]

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

3. Modeling

3.1 Define Model

In [17]:

```
# define parameter
batch_size = 1                # must be a common factor of both the train and test samples
units = 1                     # can adjust this, in model tuninig phase
nb_epoch = 100
```

In [18]:

```
# Reshape the input to 3-dim
dim(x_train) <- c(length(x_train), 1, 1)

# specify required arguments
X_shape2 = dim(x_train)[2]
X_shape3 = dim(x_train)[3]
```

In [19]:

```
# define model
model <- keras_model_sequential()

model%>%
    layer_lstm(units, batch_input_shape = c(batch_size, X_shape2, X_shape3), stateful= TRUE)%>%
    layer_dense(units = 1)
```

In [20]:

```
model %>% compile(
  loss = 'mean_squared_error',
  optimizer = optimizer_sgd( lr= 0.01),
  metrics = c('mean_squared_error', 'mean_absolute_error', 'mean_absolute_percentage_error')
)
```

In [21]:

```
summary(model)
```

```
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
________________________________________________________________________________
```

2.2 Fit Model

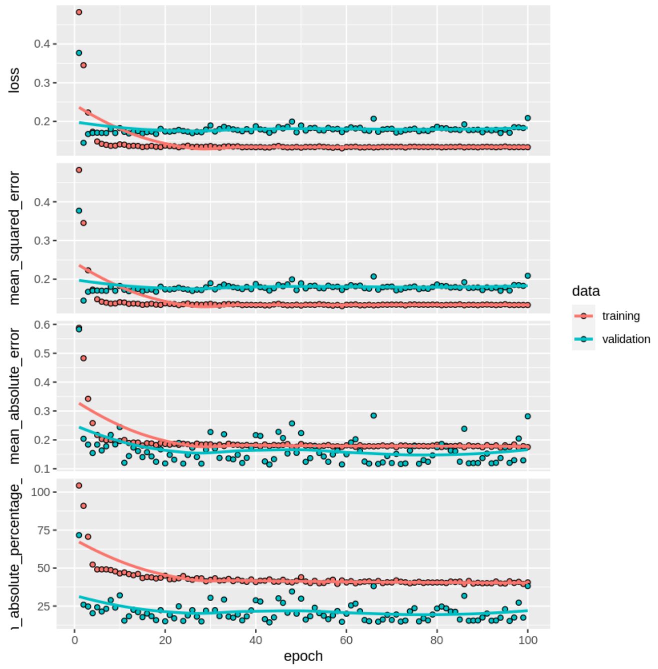
In [22]:

```
# Fit the model
history <- model %>% fit(
 x = x_train,
 y = y_train,
 epochs = nb_epoch,
 batch_size = batch_size,
 validation_split = 0.2,
 verbose = 1
)
```

In [23]:

```
plot(history)
```

```
`geom_smooth()` using formula 'y ~ x'
```



2.2 Evaluate Train

In [24]:

```
(score_train <- model %>% evaluate(x_train, y_train, batch_size=batch_size, verbose=0))
```

$loss
0.163901416085986
$mean_squared_error
0.163901060819626
$mean_absolute_error
0.259981960058212
$mean_absolute_percentage_error
73.5536880493164

In [25]:

```
mse_train <- score_train$mean_squared_error
rmse_train <- sqrt(score_train$mean_squared_error)
mae_train <- score_train$mean_absolute_error
mape_train <- score_train$mean_absolute_percentage_error

cat("MSE Train: ", mse_train, "\n")
cat("RMSE Train:", rmse_train, "\n")
cat("MAE Train:", mae_train, "\n")
cat("MAPE Train:", mape_train, "\n")
```

```
MSE Train:  0.1639011
RMSE Train: 0.404847
MAE Train: 0.259982
MAPE Train: 73.55369
```

In [26]:

```
# make prediction train

L = length(x_train)
scaler = Scaled$scaler
```

```
predictions_train = numeric(L)

for(i in 1:L){
    X <- x_train[i]
    dim(X) <- c(1,1,1)
    X <- tf$cast(X, tf$float32)
    yhat <- model %>% predict(X, batch_size=batch_size)
    # invert scaling
    yhat = invert_scaling(yhat, scaler,  c(-1, 1))
    # store
    predictions_train[i] <- yhat
}
```

In [27]:

```
unscaled_y_train = numeric(L)

for(i in 1:L){
    Y <- y_train[i]
    # invert scaling
    yhat = invert_scaling(Y, scaler,  c(-1, 1))
    unscaled_y_train[i] <- yhat
}
```

In [28]:

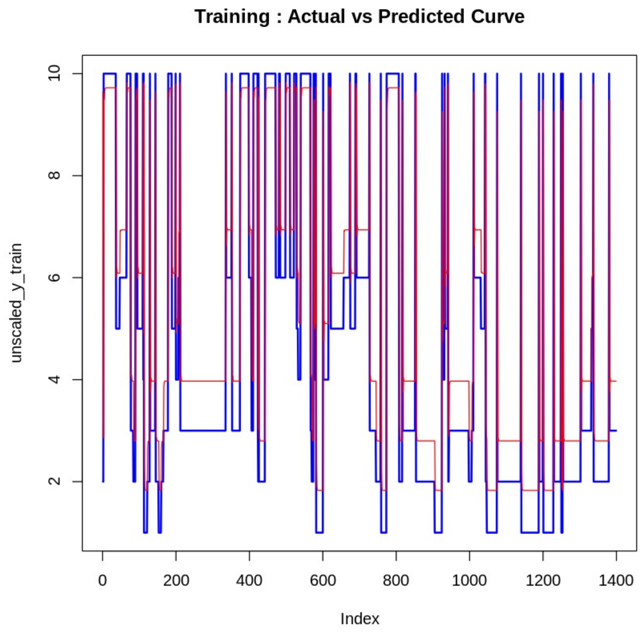
```
korelasi_train <- cor(unscaled_y_train, predictions_train)
r_sq_train <- korelasi_train^2

cat("Korelasi Train: ", korelasi_train, "\n")
cat("R-square Train: ", r_sq_train, "\n")
```

```
Korelasi Train:  0.8336987
R-square Train:  0.6950535
```

In [37]:

```
plot(unscaled_y_train, col="blue", type="l",main ="Training : Actual vs Predicted Curve", lwd = 2)
lines(predictions_train, type = "l", col = "red", lwd = 1)
```



2.2 Evaluate Test

In [30]:

```
# make prediction
```

```
L = length(x_test)
scaler = Scaled$scaler
predictions_test = numeric(L)

for(i in 1:L){
    X <- x_test[i]
    dim(X) <- c(1,1,1)
    X <- tf$cast(X, tf$float32)
    yhat <- model %>% predict(X, batch_size=batch_size)
    # invert scaling
    yhat = invert_scaling(yhat, scaler,  c(-1, 1))
    # store
    predictions_test[i] <- yhat
}
```

In [31]:

```
unscaled_y_test = numeric(L)

for(i in 1:L){
    Y <- y_test[i]
    # invert scaling
    yhat = invert_scaling(Y, scaler,  c(-1, 1))
    unscaled_y_test[i] <- yhat
}
```

In [39]:

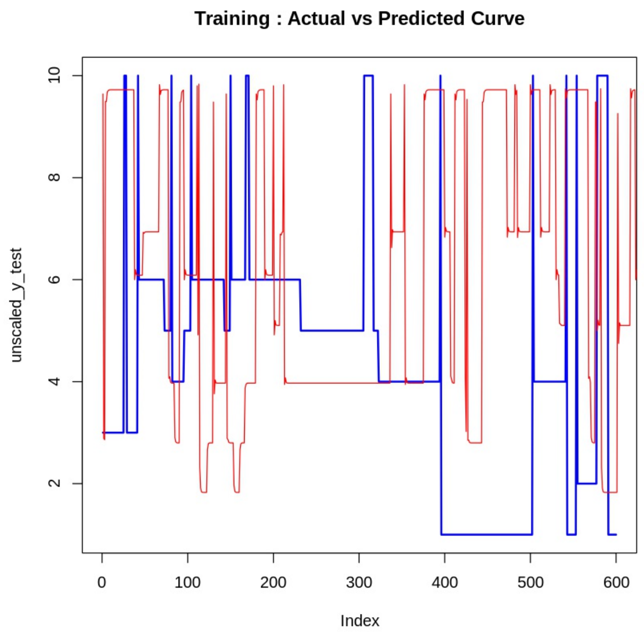
```
korelasi_test <- cor(unscaled_y_test, predictions_test)
r_sq_test <- korelasi_test^2

cat("Korelasi Test: ", korelasi_test, "\n")
cat("R-square Test: ", r_sq_test, "\n")
```

```
Korelasi Test:  0.8569178
R-square Test:  0.7343081
```

In [38]:

```
plot(unscaled_y_test, col="blue", type="l",main ="Training : Actual vs Predicted Curve", lwd = 2)
lines(predictions_train, type = "l", col = "red", lwd = 1)
```



In [34]:

```
# Reshape the input to 3-dim
dim(x_test) <- c(length(x_test), 1, 1)
```

In [35]:

```
(score_train <- model %>% evaluate(x_test, y_test, batch_size=batch_size, verbose=0))
```

$loss
0.114027401428223
$mean_squared_error
0.114027008414268
$mean_absolute_error
0.25246873497963
$mean_absolute_percentage_error
115.766967773438

In [36]:

```
mse_test <- score_train$mean_squared_error
rmse_test <- sqrt(score_train$mean_squared_error)
mae_test <- score_train$mean_absolute_error
mape_test <- score_train$mean_absolute_percentage_error

cat("MSE Test: ", mse_test, "\n")
cat("RMSE Test:", rmse_test, "\n")
cat("MAE Test:", mae_test, "\n")
cat("MAPE Test:", mape_test, "\n")
```

```
MSE Test:  0.114027
RMSE Test: 0.3376789
MAE Test: 0.2524687
MAPE Test: 115.767
```

References

- http://rwanjohi.rbind.io/2018/04/05/time-series-forecasting-using-lstm-in-r/