ELM: Extreme Learning Machines are single-hidden-layer feedforward networks where the input-to-hidden weights are randomly assigned and fixed; only the hidden-to-output weights are learned in closed form. This yields very fast training while retaining nonlinear modeling capacity via the hidden activation function. In time series, the lagged window acts as the input vector.

Objective: Fit and evaluate an ELM (Extreme Learning Machine) model for time-series forecasting with sliding windows, including data preparation, normalization, fitting, and visualization.

# Time Series Regression - ELM  
  
# Installing the package (if needed)  
#install.packages("tspredit")

# Loading the packages  
library(daltoolbox)  
library(tspredit)

# Series for study and sliding windows  
  
data(tsd)  
ts <- ts\_data(tsd$y, 10)  
ts\_head(ts, 3)

## t9 t8 t7 t6 t5 t4 t3 t2 t1  
## [1,] 0.0000000 0.2474040 0.4794255 0.6816388 0.8414710 0.9489846 0.9974950 0.9839859 0.9092974  
## [2,] 0.2474040 0.4794255 0.6816388 0.8414710 0.9489846 0.9974950 0.9839859 0.9092974 0.7780732  
## [3,] 0.4794255 0.6816388 0.8414710 0.9489846 0.9974950 0.9839859 0.9092974 0.7780732 0.5984721  
## t0  
## [1,] 0.7780732  
## [2,] 0.5984721  
## [3,] 0.3816610

# Series visualization  
library(ggplot2)  
plot\_ts(x=tsd$x, y=tsd$y) + theme(text = element\_text(size=16))



# Train-test split and projection (X, y)  
  
samp <- ts\_sample(ts, test\_size = 5)  
io\_train <- ts\_projection(samp$train)  
io\_test <- ts\_projection(samp$test)

# Preprocessing (global min-max normalization)  
  
preproc <- ts\_norm\_gminmax()

# Training the ELM model  
  
model <- ts\_elm(ts\_norm\_gminmax(), input\_size=4, nhid=3, actfun="purelin")  
model <- fit(model, x=io\_train$input, y=io\_train$output)

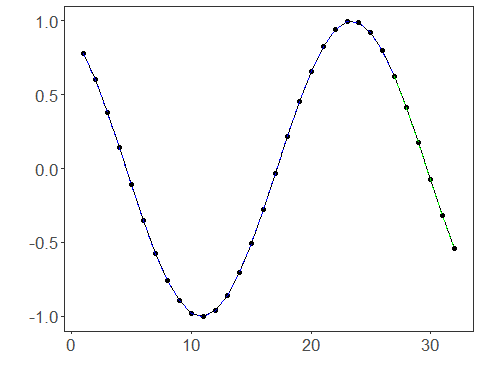
# Fit evaluation (train)  
  
adjust <- predict(model, io\_train$input)  
adjust <- as.vector(adjust)  
output <- as.vector(io\_train$output)  
ev\_adjust <- evaluate(model, output, adjust)  
ev\_adjust$mse

## [1] 2.484272e-27

# Forecast on test set  
  
prediction <- predict(model, x=io\_test$input[1,], steps\_ahead=5)  
prediction <- as.vector(prediction)  
output <- as.vector(io\_test$output)  
ev\_test <- evaluate(model, output, prediction)  
ev\_test

## $values  
## [1] 0.41211849 0.17388949 -0.07515112 -0.31951919 -0.54402111  
##   
## $prediction  
## [1] 0.41211849 0.17388949 -0.07515112 -0.31951919 -0.54402111  
##   
## $smape  
## [1] 2.056052e-12  
##   
## $mse  
## [1] 3.464323e-25  
##   
## $R2  
## [1] 1  
##   
## $metrics  
## mse smape R2  
## 1 3.464323e-25 2.056052e-12 1

# Plot results  
  
yvalues <- c(io\_train$output, io\_test$output)  
plot\_ts\_pred(y=yvalues, yadj=adjust, ypre=prediction) + theme(text = element\_text(size=16))



References - G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew (2006). Extreme learning machine: Theory and applications. Neurocomputing, 70(1–3), 489–501.