Conv1D: One-dimensional Convolutional Neural Networks (1D CNNs) extract local temporal patterns by convolving learnable filters across the input window. Stacked convolution and pooling layers can capture increasingly abstract features before a regression head maps to the next-step forecast. CNNs are effective when short- to mid-range motifs repeat over time and benefit from normalized inputs.

Objective: Train and evaluate a 1D CNN (Conv1D) for time-series forecasting with sliding windows, including normalization, fitting, and evaluation.

# Time Series Regression - 1D CNN (Conv1D)  
  
# Installing packages (if needed)  
  
#install.packages("tspredit")

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

# Training the 1D CNN  
  
model <- ts\_conv1d(ts\_norm\_gminmax(), input\_size=4, epochs=10000)  
model <- fit(model, x=io\_train$input, y=io\_train$output)

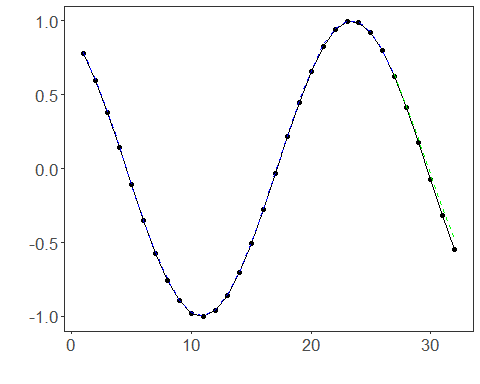
# Fit evaluation (train)  
  
adjust <- predict(model, io\_train$input)  
adjust <- as.vector(adjust)  
output <- as.vector(io\_train$output)  
ev\_adjust <- evaluate(model, output, adjust)  
ev\_adjust$mse

## [1] 6.871934e-05

# 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] 0.42627202 0.20462072 -0.03227533 -0.26520548 -0.47002587  
##   
## $smape  
## [1] 0.2652189  
##   
## $mse  
## [1] 0.002281668  
##   
## $R2  
## [1] 0.9802931  
##   
## $metrics  
## mse smape R2  
## 1 0.002281668 0.2652189 0.9802931

# 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 - Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324.