## Convolutional Autoencoder (encode-decode)

This example demonstrates how to use a 1D convolutional autoencoder to encode and reconstruct windows from a time series. After reducing from p to k dimensions, the model reconstructs back to p, enabling evaluation of reconstruction error.

Prerequisites - Python with PyTorch accessible via reticulate - R packages: daltoolbox, tspredit, daltoolboxdp, ggplot2

Quick notes - Reconstruction: compare input and output to verify local patterns are preserved. - Metrics: R2 and MAPE per column help measure quality across window steps.

# Convolutional Autoencoder transformation (encode-decode)  
  
# Considering a dataset with $p$ numerical attributes.   
  
# The goal of the autoencoder is to reduce the dimension of $p$ to $k$, such that these $k$ attributes are enough to recompose the original $p$ attributes. However, from the $k$ dimensions the data is returned back to $p$ dimensions. The higher the autoencoder quality, the more similar the output is to the input.   
  
# Installing packages  
#install.packages("tspredit")  
#install.packages("daltoolboxdp")

# Loading packages  
library(daltoolbox)  
library(tspredit)  
library(daltoolboxdp)  
library(ggplot2)

# Example dataset (series -> windows)   
data(tsd)  
  
sw\_size <- 5  
ts <- ts\_data(tsd$y, sw\_size)  
  
ts\_head(ts)

## t4 t3 t2 t1 t0  
## [1,] 0.0000000 0.2474040 0.4794255 0.6816388 0.8414710  
## [2,] 0.2474040 0.4794255 0.6816388 0.8414710 0.9489846  
## [3,] 0.4794255 0.6816388 0.8414710 0.9489846 0.9974950  
## [4,] 0.6816388 0.8414710 0.9489846 0.9974950 0.9839859  
## [5,] 0.8414710 0.9489846 0.9974950 0.9839859 0.9092974  
## [6,] 0.9489846 0.9974950 0.9839859 0.9092974 0.7780732

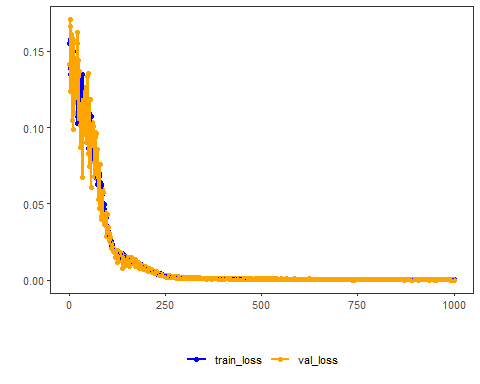
# Normalization (min-max by group)  
preproc <- ts\_norm\_gminmax()  
preproc <- fit(preproc, ts)  
ts <- transform(preproc, ts)  
  
ts\_head(ts)

## t4 t3 t2 t1 t0  
## [1,] 0.5004502 0.6243512 0.7405486 0.8418178 0.9218625  
## [2,] 0.6243512 0.7405486 0.8418178 0.9218625 0.9757058  
## [3,] 0.7405486 0.8418178 0.9218625 0.9757058 1.0000000  
## [4,] 0.8418178 0.9218625 0.9757058 1.0000000 0.9932346  
## [5,] 0.9218625 0.9757058 1.0000000 0.9932346 0.9558303  
## [6,] 0.9757058 1.0000000 0.9932346 0.9558303 0.8901126

# Train/test split  
samp <- ts\_sample(ts, test\_size = 10)  
train <- as.data.frame(samp$train)  
test <- as.data.frame(samp$test)

# Training autoencoder (reduce 5 -> 3)  
auto <- autoenc\_conv\_ed(5, 3)  
auto <- fit(auto, train)

fit\_loss <- data.frame(x=1:length(auto$train\_loss), train\_loss=auto$train\_loss,val\_loss=auto$val\_loss)  
  
grf <- plot\_series(fit\_loss, colors=c('Blue','Orange'))  
plot(grf)



# Testing the autoencoder  
# Show test samples and display reconstruction  
print(head(test))

## t4 t3 t2 t1 t0  
## 1 0.7258342 0.8294719 0.9126527 0.9702046 0.9985496  
## 2 0.8294719 0.9126527 0.9702046 0.9985496 0.9959251  
## 3 0.9126527 0.9702046 0.9985496 0.9959251 0.9624944  
## 4 0.9702046 0.9985496 0.9959251 0.9624944 0.9003360  
## 5 0.9985496 0.9959251 0.9624944 0.9003360 0.8133146  
## 6 0.9959251 0.9624944 0.9003360 0.8133146 0.7068409

result <- transform(auto, test)  
print(head(result))

## , , 1  
##   
## [,1] [,2] [,3] [,4] [,5]  
## [1,] 0.7532327 0.8559688 0.9248574 0.9485527 0.9597216  
## [2,] 0.8431355 0.9151731 0.9573855 0.9640068 0.9616455  
## [3,] 0.9057942 0.9409162 0.9657439 0.9626321 0.9466876  
## [4,] 0.9385246 0.9516720 0.9637739 0.9497952 0.9107729  
## [5,] 0.9541737 0.9527586 0.9504205 0.9164878 0.8336270  
## [6,] 0.9511613 0.9405294 0.9069749 0.8392485 0.7131780

# Reconstruction metrics per column: R2 and MAPE  
result <- as.data.frame(result)  
names(result) <- names(test)  
r2 <- c()  
mape <- c()  
for (col in names(test)){  
 r2\_col <- cor(test[col], result[col])^2  
 r2 <- append(r2, r2\_col)  
 mape\_col <- mean((abs((result[col] - test[col]))/test[col])[[col]])  
 mape <- append(mape, mape\_col)  
 print(paste(col, 'R2 test:', r2\_col, 'MAPE:', mape\_col))  
}

## [1] "t4 R2 test: 0.990353117401416 MAPE: 0.0270960835880134"  
## [1] "t3 R2 test: 0.972838460794404 MAPE: 0.0233588306631604"  
## [1] "t2 R2 test: 0.99309053169793 MAPE: 0.0159651280847722"  
## [1] "t1 R2 test: 0.993910412685348 MAPE: 0.0179303798188564"  
## [1] "t0 R2 test: 0.996032657153929 MAPE: 0.0175232001141548"

print(paste('Means R2 test:', mean(r2), 'MAPE:', mean(mape)))

## [1] "Means R2 test: 0.989245035946605 MAPE: 0.0203747244537914"

# Note: beware of divisions by values near zero when computing MAPE.