## Autoencoder (Encode) - Overview

This example demonstrates how to train a vanilla autoencoder to learn a latent representation (encoding) of a sliding window from a time series. The idea is to reduce the dimensionality from p attributes to k, preserving relevant information. You can use the encoded vectors as input to other tasks (such as clustering or prediction).

Prerequisites - Reticulate configured and Python with PyTorch installed - R packages: daltoolbox, tspredit, daltoolboxdp, ggplot2

Steps 1) Prepare the dataset (series windows) 2) Normalize data (avoid unbalanced scales) 3) Split into train and test 4) Train the autoencoder (reduce from 5 to 3 dimensions) 5) Inspect losses and transform data into latent codes

# Vanilla autoencoder transformation (encode)  
  
# 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.   
  
# Installing packages  
#install.packages("tspredit")  
#install.packages("daltoolboxdp")

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

# Example dataset (time series and 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

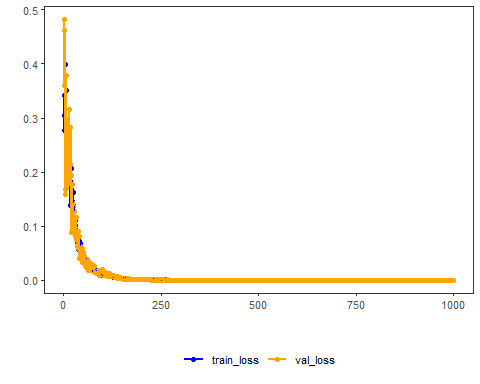
# Normalization (min-max by group) to stabilize training  
preproc <- ts\_norm\_gminmax()  
preproc <- fit(preproc, ts)  
ts <- transform(preproc, ts)  
  
ts\_head(ts)

## 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)

# Creating and training the autoencoder (reduce from 5 to 3 dimensions)  
auto <- autoenc\_e(5, 3)  
auto <- fit(auto, train)

# Visualizing loss curves (train/validation)  
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: encoding test examples  
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] [,2] [,3]  
## [1,] -0.8438497 -0.9710245 -0.5603727  
## [2,] -0.9606133 -1.0351816 -0.5179921  
## [3,] -1.0432230 -1.0567327 -0.4577575  
## [4,] -1.0834730 -1.0351111 -0.3873226  
## [5,] -1.0806378 -0.9714007 -0.3105585  
## [6,] -1.0402445 -0.8727744 -0.2311186