## Autoencoder (Encode-Decode) - Overview

The autoencoder jointly learns an encoder (p → k) and decoder (k → p) by minimizing reconstruction error. With sufficient capacity and regularization, the bottleneck enforces information compression so that reconstructions approximate inputs with low error.

This example shows an autoencoder that encodes and reconstructs the input. After training the reduction from p -> k dimensions, the model decodes back to p. The better the training, the closer the reconstruction is to the original (low reconstruction error).

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

Steps 1) Build time-series windows 2) Normalize data 3) Split into train and test 4) Train the AE (5 -> 3) and track losses 5) Reconstruct and compute metrics (R2, MAPE)

# Vanilla 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 is the output 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

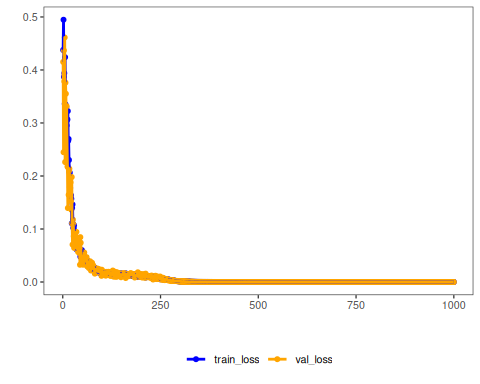
# Normalization (min-max by group)  
preproc <- ts\_norm\_gminmax()  
preproc <- fit(preproc, ts)  
ts <- transform(preproc, ts)  
  
ts\_head(ts)

## 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 (reduction 5 -> 3)  
auto <- autoenc\_ed(5, 3)  
auto <- fit(auto, train)

# Loss curves (train and 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: reconstruction of the test set  
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] [,4] [,5]  
## [1,] 0.7248489 0.8282283 0.9119520 0.9707282 0.9983931  
## [2,] 0.8307251 0.9111441 0.9706296 0.9981017 0.9965914  
## [3,] 0.9143901 0.9691466 0.9994863 0.9949566 0.9635265  
## [4,] 0.9701966 0.9988965 0.9951566 0.9624572 0.9010718  
## [5,] 0.9962895 0.9983780 0.9608707 0.9006096 0.8128038  
## [6,] 0.9918187 0.9669893 0.8990593 0.8139390 0.7045459

# Evaluating reconstruction quality: R2 and MAPE per attribute  
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: 0.999636775109796 MAPE: 0.00228681781040957"  
## [1] "t3 R2 test: 0.999509690133341 MAPE: 0.00317716159343264"  
## [1] "t2 R2 test: 0.999963330277838 MAPE: 0.00200280440000862"  
## [1] "t1 R2 test: 0.999972627359698 MAPE: 0.00159233648419616"  
## [1] "t0 R2 test: 0.99992897033067 MAPE: 0.0036558630443794"

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

## [1] "Means R2 test: 0.999802278642269 MAPE: 0.00254299666648528"

References - Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. (Chapter on Autoencoders)