## Adversarial Autoencoder (encode-decode)

This example shows how to train and use an Adversarial Autoencoder (AAE) in encode-decode mode: the model compresses windows from p to k dimensions and then reconstructs back to p, enabling evaluation of reconstruction error.

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

Quick notes - Evaluation: reconstruction quality measured, for example, via R2 and MAPE per window column. - Hyperparameters: num\_epochs, batch\_size influence convergence and adversarial stability.

# Installing example dependencies (if needed)  
#install.packages("tspredit")  
#install.packages("daltoolboxdp")

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

# Example dataset (series -> windows)  
data(tsd)  
  
sw\_size <- 5 # sliding window size (p)  
ts <- ts\_data(tsd$y, sw\_size) # convert series into windows with p columns  
  
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

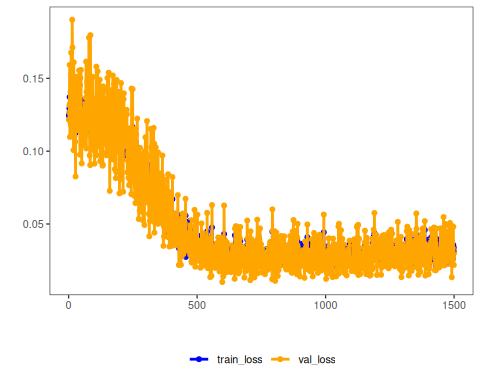
# Normalization (min-max by group)  
preproc <- ts\_norm\_gminmax()  
preproc <- fit(preproc, ts)  
ts <- transform(preproc, ts)  
  
ts\_head(ts)

## 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)

# Creating the adversarial autoencoder (encode-decode): 5 -> 3 -> 5 dimensions  
auto <- autoenc\_adv\_ed(5, 3, batch\_size = 3, num\_epochs = 1500)  
  
# Training the model  
auto <- fit(auto, train)

# Learning curves (train and validation loss per epoch)  
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 (reconstruction)  
# Show samples from the test set and the generated reconstruction (p columns)  
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,] 0.8507356 0.9051839 0.9285110 0.9303031 0.8938683  
## [2,] 0.8782777 0.9289893 0.9488168 0.9507282 0.9176667  
## [3,] 0.8899800 0.9384462 0.9565561 0.9584369 0.9273930  
## [4,] 0.8875695 0.9365563 0.9549674 0.9568641 0.9254043  
## [5,] 0.8707420 0.9227523 0.9434187 0.9453349 0.9112893  
## [6,] 0.8364844 0.8921487 0.9166197 0.9182562 0.8810522

# Reconstruction metrics per column: R2 and MAPE  
# Note: MAPE can be sensitive to values close to zero.  
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: 0.328654930328106 MAPE: 0.173053299511841"  
## [1] "t3 R2 test: 0.88796368264239 MAPE: 0.0958384385327353"  
## [1] "t2 R2 test: 0.959651392512862 MAPE: 0.0432026044053782"  
## [1] "t1 R2 test: 0.906253242931916 MAPE: 0.137711846279584"  
## [1] "t0 R2 test: 0.846019655121928 MAPE: 0.327083187717858"

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

## [1] "Means R2 test: 0.78570858070744 MAPE: 0.155377875289479"