## Adversarial Autoencoder (encode-decode)

An AAE couples reconstruction loss with an adversarial game in latent space. A discriminator distinguishes samples from the encoder vs. the prior, while the encoder seeks to fool it, aligning the aggregate posterior with the chosen prior distribution.

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

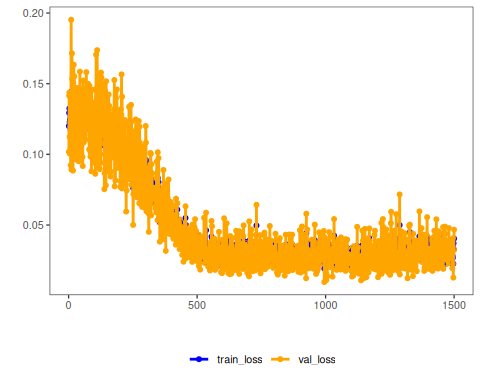
# Normalization (min-max by group)  
preproc <- ts\_norm\_gminmax()  
preproc <- fit(preproc, ts)  
ts <- transform(preproc, ts)  
  
ts\_head(ts)

## 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,] 0.8576337 0.9096393 0.9260868 0.9232257 0.8842417  
## [2,] 0.8845979 0.9329191 0.9469912 0.9454218 0.9097760  
## [3,] 0.8958009 0.9418942 0.9547969 0.9536744 0.9199443  
## [4,] 0.8939812 0.9405263 0.9535182 0.9523173 0.9183015  
## [5,] 0.8776266 0.9272050 0.9416381 0.9397333 0.9031091  
## [6,] 0.8444961 0.8976923 0.9144374 0.9107234 0.8711892

# 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: 0.332585734131919 MAPE: 0.166033800236882"  
## [1] "t3 R2 test: 0.888406976329992 MAPE: 0.0900354408271632"  
## [1] "t2 R2 test: 0.960114424827771 MAPE: 0.0431308898107746"  
## [1] "t1 R2 test: 0.911522627223471 MAPE: 0.129756572477852"  
## [1] "t0 R2 test: 0.850139987464046 MAPE: 0.314092789103655"

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

## [1] "Means R2 test: 0.78855394999544 MAPE: 0.148609898491265"

References - Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., & Frey, B. (2015). Adversarial Autoencoders. arXiv:1511.05644.