GAN

Stefan Huber

December 2022

1 Introduction

In order to generate the time series of a similar distribution as the original ones, we chose to use a Generative Adversarial Network (GAN). The GAN framework consists of two models, the generator and the discriminator. They learn by “playing” against each other: The generator generates new time series and tries to trick the discriminator into predicting them as real ones, while the predictor learns to distinguish them. Ideally in this process, the predictor learns the distribution of real data and the generator generates series that share exactly this distribution, such that ultimately the discriminator’s best choice is to pick at random.
Regarding the architecture of generator and discriminator, we draw significant inspiration from Savasta & Politano (2020). We used a Wasserstein GAN, as this improves training stability and the loss directly correlates with the sample quality, so for more advanced performance evaluation we later can just use the loss function of the generator rather than only see whether the model can fool human eyes.

2 A first Generator

A first Generator The architecture is mainly a combination of linear and convolutional layers and was designed by Fernando (2019), but as Savasta Politano (2020) we use RMSprop as optimizer rather than adam. In contrast to them however, we completely remove spectral normalization as it caused model collapse.
The generator takes a random noise vector of length 50 as input returns an XXX [edit]. It consists of several layers, each of which applies a different transformation to the input data. The first layer is a linear transformation, which applies a matrix multiplication and a bias term to the input data. This is followed by a Leaky ReLU nonlinearity, which applies a nonlinear function element-wise to the input data. The next layer adds an extra dimension to the input data, which is necessary for the subsequent convolutional layers. Then three 1D convolutional layers follow, which apply a convolution operation to the input data. This operation is a way of extracting local spatial information from the input data. The convolution is followed by an upsampling operation, which increases the size of the data. The upsampling is followed by a dropout layer, which randomly zeroes out some of the input data. This prevents overfitting and increases robustness. Then a fourth convolutional layer follows.
Finally a Leaky ReLU applies an element-wise nonlinear function to the input data and the data get squeezed down to a single dimension, and the last layer is a linear transformation, which applies a matrix multiplication and a bias term to the input data.

3 The Discriminator