Generating Time-Series with Adversarial Networks

February 16, 2024

Andrea Di Marco

Abstract

A good generative model for time-series data not only should capture the conditional dynamics of transitions, but also its open-loop rollouts and preserve the joint distribution of trajectories. Autoregressive models explicitly factor the distribution of sequences into a product of conditionals, while useful in the context of forecasting, this approach is fundamentally deterministic, and is not truly generative in the sense that new sequences can be randomly sampled from them without external conditioning. This project suggests an architecture that combines the flexibility of the unsupervised paradigm with the control afforded by supervised training. The code is available here.

1. Introduction

This work proposes a general architecture that can be changed in order to generate Time-Series of various domains. Using an embedding network to provide a reversible mapping between features and latent representations, reducing the high-dimensionality of the adversarial learning space. This capitalizes on the fact that the temporal dynamics of even complex systems are often driven by fewer and lower-dimensional factors of variation. The following model is based on (Yoon et al., 2019).

2. Related Work

Autoregressive recurrent networks trained via the maximum likelihood principle (Williams & Zipser, 1989) are prone to potentially large prediction errors when performing multi-step sampling, due to the discrepancy between closed-loop training (conditioned on ground truths) and open-loop inference (conditioned on previous guesses).

Multiple studies have inherited the GAN framework within the temporal setting. The first (Mogren, 2016) applied the GAN architecture to sequential data, generated recurrently, taking as inputs a noise vector and the previously generated samples. A multitude of applied studies have since utilized these frameworks to generate synthetic sequences in very diverse domains such as finance (Simonetto, 2018) and biosignals (Haradal et al., 2018).

3. Architecture

Unlike (Yoon et al., 2019) this work implements different loss functions and *seven* different modules working together to achieve the goal, instead of five. This was done to make the training more stable and prevent the Generator ($\mathbb{G}$) module to collapse and return the same sequence regardless of the input. Each module's implementation is roughly the same, with different hyperparameters: a recurrent network (*RNN*, *GRU* or *LSTM*) followed by fully connected layers.

Embedder ($\mathbb{E}$) Its purpose is to embedd the original sequence $\mathcal{X}$ into its latent representation $\mathcal{H} = \mathbb{E}(\mathcal{X})$. The loss function for this module is:

$$\mathcal{L}_{\mathbb{E}} = \lambda_1 |\mathcal{X} - \mathbb{R}(\mathbb{E}(\mathcal{X}))| + \lambda_2 |\mathbb{E}(\mathcal{X}) - \mathbb{S}(\mathbb{E}(\mathcal{X}))| \quad (1)$$

Recovery ($\mathbb{R}$) Aims to map the latent representation $\mathcal{H}$ back to its feature representation $\tilde{\mathcal{X}} = \mathbb{R}(\mathcal{H})$. The loss function for this module is:

$$\mathcal{L}_{\mathbb{R}} = \lambda_1 |\mathcal{X} - \mathbb{R}(\mathbb{E}(\mathcal{X}))| - \lambda_2 log(\mathbb{D}_{\mathbb{F}}(\mathbb{R}(\mathbb{E}(\mathcal{X})))) \quad (2)$$

Generator ($\mathbb{G}$) Takes a sequence $\mathcal{Z}$ sampled from a brownian motion with low dimensionality, and generates a latent representation $\hat{\mathcal{H}} = \mathbb{G}(\mathcal{Z})$. Note that to go from the noise to a sequence in the feature space the process is $\mathbb{R}(\mathbb{G}(\mathcal{Z}))$. The loss function for this module is:

$$\mathcal{L}_{\mathbb{G}} = \lambda_1\alpha_1 + \lambda_2\alpha_2 + \lambda_3\alpha_3 + \lambda_4\alpha_4 + \lambda_5\alpha_5 + \lambda_6\alpha_6 \quad (3)$$

Where:

$$\alpha_1 = log(\mathbb{D}_{\mathbb{L}}(\mathbb{S}(\mathbb{G}(z_i))))$$
$$\alpha_2 = log(\mathbb{D}_{\mathbb{L}}(\mathbb{G}(x_i)))$$
$$\alpha_3 = log(\mathbb{D}_{\mathbb{F}}(\mathbb{R}(\mathbb{G}(z_i))))$$
$$\alpha_4 = |\mathbb{G}(\mathcal{Z}) - \mathbb{S}(\mathbb{G}(\mathcal{Z}))|$$
$$\alpha_5 = D_{KL}[\mathbb{R}(\mathbb{G}(\mathcal{Z}))||\mathcal{X}]$$
$$\alpha_6 = |\mathcal{Z} - \mathbb{N}(\mathbb{G}(\mathcal{Z}))|$$

Email: Di Marco <dimarco.1835169@studenti.uniroma1.it>.

Deep Learning and Applied AI 2023, Sapienza University of Rome, 2nd semester a.y. 2022/2023.

Supervisor ($\mathbb{S}$) Provides feedback to the modules, it serves the purpose of inducing the space spanned by the Generator ($\mathbb{G}$) to be the same as the one spanned by the Embedder ($\mathbb{E}$). The loss function for this module is:

$$\mathcal{L}_{\mathbb{S}} = \lambda_1 |\mathbb{E}(\mathcal{X}) - \mathbb{S}(\mathbb{E}(\mathcal{X}))| + \lambda_2 \beta_1 \tag{4}$$
$$\beta_1 = D_{KL}\left[\mathbb{S}(\mathbb{G}(\mathcal{Z})) \| \mathbb{S}(\mathbb{E}(\mathcal{X}))\right]$$

Latent Discriminator ($\mathbb{D}_{\mathbb{L}}$) The first discriminator module aims to distinguish the latent representation $\mathcal{H}$ returned by the Embedder ($\mathbb{E}$) from the latent representation $\hat{\mathcal{H}}$ returned by the Generator ($\mathbb{G}$). The loss function for this module is:

$$\mathcal{L}_{\mathbb{D}_{\mathbb{L}}} = \lambda_1 \gamma_1 + \lambda_2 \gamma_2 + \lambda_3 \gamma_3 \tag{5}$$

Where:

$$\gamma_1 = log(1 - \mathbb{D}_{\mathbb{L}}(\mathbb{S}(\mathbb{G}(\mathcal{Z}))))$$
$$\gamma_2 = log(1 - \mathbb{D}_{\mathbb{L}}(\mathbb{G}(\mathcal{Z})))$$
$$\gamma_3 = log(\mathbb{D}_{\mathbb{L}}(\mathbb{E}(\mathcal{X})))$$

Noise Recovery ($\mathbb{N}$) Given the latent representation $\hat{\mathcal{H}}$ aims to recover the noise $\mathcal{Z}$ that generated it, the main purpose of this module is preventing the Generator ($\mathbb{G}$) from returning the same sequence regardless of the input. The loss function for this module is:

$$\mathcal{L}_{\mathbb{N}} = |\mathcal{Z} - \mathbb{N}(\mathbb{G}(\mathcal{Z}))| \tag{6}$$

Feature Discriminator ($\mathbb{D}_{\mathbb{F}}$) The last module aims to distinguish the real sequences $\mathcal{X}$, from the synthetic sequences $\hat{\mathcal{X}} = \mathbb{R}(\mathbb{G}(\mathcal{Z}))$. This module became necessary as tests showed that the Generator ($\mathbb{G}$) often mapped the noise $\mathcal{Z}$ to sequences $\hat{\mathcal{H}}$ that were deemed authentic by the Latent Discriminator ($\mathbb{D}_{\mathbb{L}}$) but when expanded by the Recovery module ($\mathbb{R}$) they did not match the right behavior of the Time-Series. The loss function for this module is:

$$\mathcal{L}_{\mathbb{D}_{\mathbb{F}}} = \lambda_1 \delta_1 + \lambda_2 \delta_2 + \lambda_3 \delta_3 \tag{7}$$

Where:

$$\delta_1 = log(1 - \mathbb{D}_{\mathbb{L}}(\mathbb{R}(\mathbb{S}(\mathbb{G}(\mathcal{Z})))))$$
$$\delta_2 = log(1 - \mathbb{D}_{\mathbb{L}}(\mathbb{R}(\mathbb{G}(\mathcal{Z}))))$$
$$\delta_3 = log(\mathbb{D}_{\mathbb{F}}(\mathcal{X}))$$

4. Results

Anomaly Detection To prove the effectiveness of the GAN it should be possible to train an anomaly detector like (Kurt et al., 2021) on the real data and the alarm rates should be similar between the real samples $\mathcal{X}$ and the generated samples $\hat{\mathcal{X}} = \mathbb{R}(\mathbb{G}(\mathcal{Z}))$, indeed this is the case with a 1.72% *False Alarm Rate* on $\mathcal{X}$ and a 2.96% *True Alarm Rate* on $\hat{\mathcal{X}}$. My C++ implementation of (Kurt et al., 2021) is available here.

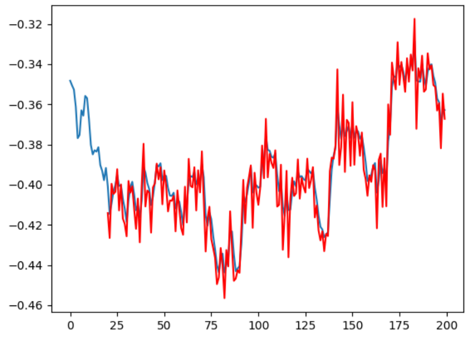

Figure 1. Forecasting test

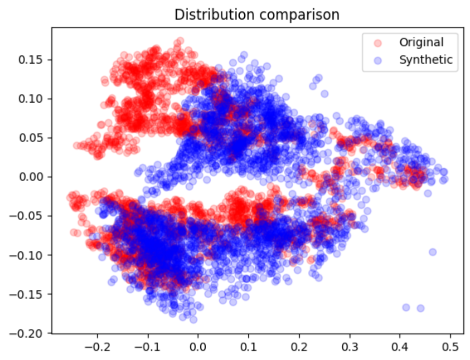

Figure 2. Visualization of the distributions.

Forecasting Another feat that should be possible to achieve with an ideal GAN is to train a simple regressive network on the generated data $\hat{\mathcal{X}} = \mathbb{R}(\mathbb{G}(\mathcal{Z}))$ and then use it to predict the next values of the real sequences extracted from $\mathcal{X}$, this is indeed possible. The picture (1) shows the first 200 prediction on one dimension, with an average absolute error of 0.06381 on the whole multi-dimensional sequence, where the blue time-series is sampled from $\mathcal{X}$ and red one is the prediction.

Distributions To be able to see the differences in the spanned space between $\mathcal{X}$ and the synthetic ones $\hat{\mathcal{X}}$, even when dealing with high-dimensionality, figure (2) shows the results of tuning PCA to map $\mathcal{X}$ on two dimensions and then projecting $\hat{\mathcal{X}}$ with the same learned weights. Even if not perfectly matching, the overlap is significant.

Conclusions The amount of samples and computational power needed can be too much when wanting to simulate a system. Still, to maintain confidentiality, it might be tempting to send a trained model instead of the complete simulation to run some tests.

References

Haradal, S., Hayashi, H., and Uchida, S. Biosignal data augmentation based on generative adversarial networks. In *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 368–371, 2018. doi: 10.1109/EMBC.2018.8512396.

Kurt, M. N., Ylmaz, Y., and Wang, X. Real-time nonparametric anomaly detection in high-dimensional settings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(7):2463–2479, July 2021. ISSN 1939-3539. doi: 10.1109/tpami.2020.2970410. URL http://dx.doi.org/10.1109/TPAMI.2020.2970410.

Mogren, O. C-rnn-gan: Continuous recurrent neural networks with adversarial training, 2016.

Simonetto, L. Generating spiking time series with generative adversarial networks : an application on banking transactions. 2018. URL https://api.semanticscholar.org/CorpusID:53511789.

Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989. doi: 10.1162/neco.1989.1.2.270.

Yoon, J., Jarrett, D., and van der Schaar, M. Time-series generative adversarial networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf.