Erasmus University Rotterdam

Interest rate risk simulation after EONIA-€STER transition using a macro-finance temporal and latent representation based Generative Adversarial Network.

Author

G.T.F. (Lars) ter Braak

(512956)

University supervisor

Prof. dr. Chen Zhou

Second assessor

Prof. dr. Michel van der Wel

Abstract

This paper discusses a Generative Adversarial Network (GAN) focused on multivariate time series with the objective to predict the Value at Risk (VaR) in the Euro Overnight Index Average (EONIA) and the financial risk implications of the EONIA-€STER transition. The research starts with a comparative study between EONIA and €STER in terms of theoretical and self-proposed stylized facts of short rates. We show their differences exceeding the first and second statistical moment. Subsequently, we use a macro-finance temporal and latent representation focused GAN, denoted by TimeGAN, to simulate future extreme short rates scenarios. Using the trained TimeGAN we assess the European Central Bank (ECB)'s proposed mapping from €STER to EONIA as €STER +8.5bps. TimeGAN with Wasserstein-1 Gradient Penalty and TimeGAN with Positive Label Smoothing and Feature Matching predict acceptable 20-day VaR estimates for EONIA based on the Kupiec backtest. Despite acceptable predictive performance, the 20-day EONIA simulations are less diverse than a one-factor Vasicek or Variance-Covariance model. Based on the Discriminator of the TimeGAN with Positive Label Smoothing and Feature Matching, we can not reject the hypothesis that the ECB's proposed mapping EONIA-€STER is correct.

Keywords: EONIA-€STER transition, macro-finance short rate modelling, temporal GAN

9th October 2020

The markets can stay irrational longer than you can stay solvent - John Maynard Keynes

Contents

List of Figures

List of Tables

Acronyms

€STER EURO Short Term Rate. i–v, 1–44

Adam	Adaptive Moment Estimation. 8, 24, 40, 41
ADF	Augmented Dickey Fuller. 14, 15
ARIMA	AutoRegressive Integrated Moving Average. 3
CAE	Contractive Autoencoder. 34, 35
CNN	Convolutional Neural Network. 23
DAE	Denoising Autoencoder. 34, 35
ECB	European Central Bank. ii, 1, 2, 4, 13–16
EONIA	Euro Overnight Index Average. i–v, 1–44
GAN	Generative Adversarial Network. ii, 1, 3–7, 9–11, 18, 23, 24, 38
GDP	Gross Domestic Product. ii, 2, 14
GFC	Great Financial Crisis. 14
GRU	Gated Recurrent Unit. 7, 38
HICP	Harmonised Index of Consumer Prices. ii, 14
IRD	Interest Rate Derivatives. 1
JSD	Jensen-Shannon Divergence. 9, 10
KLD	Kullback-Leibler Divergence. 9, 10, 33, 35
LSTM	Long Short Term Memory. 3, 7, 17, 18, 23, 24, 36–38, 42
MSE	Mean Squared Error. 7, 16
OIS	Overnight Index Swap. 1
PCA	Principal Component Analysis. 3, 32–35
RNN	Recurrent Neural Network. iv, 23, 35–38
SAE	Sparse Autoencoder. 34, 35
SAGAN	Self-Attention Generative Adversarial Network. 23
SN-GAN	Spectrally Normalized Generative Adversarial Network. 24
STL	Season and Trend decomposition using LOESS. 14, 15
t-SNE	t-distributed Stochastic Neighbor Embedding. iii, iv, 16, 20, 22, 31–33
TimeGAN	Time Series GAN. iii, iv, 1, 3, 4, 6–10, 12, 16, 18–20, 22, 23, 31, 35, 40–42
UMAP	Uniform Manifold Approximation and Projection. 23
VAR	Vector AutoRegression. 3
VaR	Value at Risk. iv, 1, 16, 18–20, 23, 32, 33
WGAN	Wasserstein GAN. 10
WGAN-GP	Wasserstein GAN with Gradient Penalty. 10, 11

1 Introduction

As of 2020, Euro Overnight Index Average (EONIA) is the short rate underlying EURO-denominated contracts and will be fully replaced by EURO Short Term Rate (€STER) on 3rd of January, 2022. The methodology for the construction of €STER is different from EONIA and therefore differences could be present after the EONIA-€STER transition. In order to check for EONIA-€STER differences, €STER was measured but not implemented during a period called the pre-€STER period ranging from March 15th, 2017 until September 30th, 2019. On October 2nd the European Central Bank (ECB) published the first €STER post diem. The EONIA-€STER transition period is schematically shown in Figure 1.

Figure 1. Visualization of the EONIA-€STER transition period. At October 2nd, 2019, EONIA is replaced by €STER +8.5bps and €STER is implemented after pre-€STER. After January 3rd, 2022, ECB will discontinue the publication of EONIA.

Based on an analysis of the pre-€STER period, the ECB decided to define the mapping from €STER to EONIA as €STER +8.5bps. The short rate is used in finance i.a. for Overnight Index Swap (OIS) discounting of interest rate derivatives in the multi-curve framework. The European Securities and Markets Authority published in *ESMA Annual Statistical Report - EU Derivatives Markets - 2019* (2019) that more than 70% of total notional amount of derivatives is concentrated in Interest Rate Derivatives (IRD). The EONIA-€STER transition is thus relevant for a major part of the derivatives industry where risk models were calibrated on EONIA dynamics. If we use the ECB's suggested mapping, then the risk models might produce inaccurate risk measures. Given the short deployment of €STER, classical methodologies for calculation of the risk metric Value at Risk (VaR) are potentially biased and unstable. This paper focuses on the TimeGAN model, a model-free simulation technique, developed by Yoon et al. (2019) allows us to assess the applicability of the ECB's suggested mapping for risk modelling. First, we discuss the short-rate modelling literature and related risk models. Thereafter we discuss the TimeGAN model and its application to risk modelling and we assess the ECB's mapping from EONIA to €STER.

2 Related work

Interest rates, as opposed to stocks, can not rise indefinitely because too extreme interest rates hamper the economy. The Vasicek (1977) models captures the mean reversion in interest rates using an Ornstein-Uhlenbeck stochastic process. Extensions are proposed by Cox et al. (1985) who include a non-negativity constraint and Hull and White (1990) who impose a no-arbitrage constraint in the yield curve. The assumption of a single source of market risk is insufficient for risk modelling. Therefore Longstaff and Schwartz (1992) propose a two-factor model including a stochastic mean component and L. Chen (1996) proposes a three-factor model including a stochastic volatility and a stochastic mean component. Duffee

and Stanton (2012) were critical on the assumption that unobservable latent factors completely define the evolution of short rates and suggested to include macroeconomic variables in the short rate modelling. Joslin et al. (2014) state that macroeconomic variables are important for forecasting short rates and Ang and Piazzesi (2003) conclude that the short end of the yield curve is mostly influenced by macroeconomics. Both papers propose to include inflation and economic growth based on the Taylor (1993) rule.[1][2][3] Fleming and Remolona (1999) connect macroeconomics to market microstructure and show that trading activity and liquidity are related to scheduled macroeconomic announcements. The trading activity is measured as the total volume of unsecured overnight borrowings of panel banks.[4] Goyenko and Ukhov (2009) state that liquidity of short-term bonds is first to react to changes in monetary policy and therefore it is hypothesized that the short rate is most effected by liquidity. Beber et al. (2009) state that in the Euro area liquidity is the main driver of interest rates in times of market stress while credit quality has more explanatory power for cross-sectional difference in short rates. Bühler and Trapp (2009) state that credit risk premia and liquidity are positively correlated and by modelling liquidity we implicitly capture any excess credit risk premia. Liquidity is estimated as the bid-ask spread on short-term Euribor tenors using Equation 2 where $r_t^+(\tau)$ denotes the highest quote and $r_t^-(\tau)$ the lowest quote of the panel banks used in the Euribor calculation for the specific tenor τ.[5] Table 2 shows a brief overview of the available data while a more elaborate exploration follows in Section 5.

$$\text{Spread}_t(\tau) = r_t^+(\tau) - r_t^-(\tau) \tag{2}$$

Table 2. *Short rate and short rate risk factors. The new methodology for the calculation of €STER allows for the inclusion of a liquidity component directly related to €STER.*

Short rate	Macroeconomic risk factors	Market microstructure risk factors
€STER	Inflation	Liquidity Euribor
EONIA	Output gap	Transaction volume
		Liquidity €STER

[1] Cieslak and Povala (2015) endorse the inclusion of inflation in short rate modelling and Cooper and Priestley (2009) endorse the inclusion of economic activity in short rate modelling.

[2] The Taylor (1993) rule links the growth of real Gross Domestic Product (GDP), y_t, and the rate of inflation, π_t, to the target short-term nominal interest rate, i_t, using Equation (1). In (1) π_t^* is the desired rate of inflation, r_t^* is the assumed equilibrium real interest rate and $\bar{y}_t$ is the the long-term growth of real GDP, commonly referred to as the potential output, as determined by a linear trend.

$$i_t = \pi_t + r_t^* + a_\pi(\pi_t - \pi_t^*) + a_y(y_t - \bar{y}_t) \tag{1}$$

[3] Rudebusch and Wu (2008) model the yield curve with inflation and output gap as explanatory variables.

[4] The EONIA and €STER methodology states that a representative panel of banks in the Euro area must provide daily quotes of the short rate. The panel bank composition is published on the European Money Markets Institute website.

[5] The Euribor spreads are used as an estimate of the EONIA liquidity due to a lack of data on the EONIA panel bank contributions. The complementary liquidity estimate of €STER is only used in sensitivity analysis and differs from the liquidity estimate of Euribor as stated in Equation 2. The €STER liquidity is calculated based on the spread between the 25^{th} and 75^{th} percentile of the €STER as published by the ECB.

We combine the mean reversion from the models before 2000, the macroeconomic risk factors from the models after 2000, and the market microstructure risk factors in one model. We see the explanatory variables and the short rate in Table 2 during a time period of length T as an image generated by the underlying process.

A non-linear technique that has been proven useful in image generation is the Generative Adversarial Network (GAN) model proposed by Goodfellow et al. (2014) and its extension for multivariate timeseries, the TimeGAN model proposed by Yoon et al. (2019).[6] The TimeGAN model simulates model-free future scenarios of the variables and thus allows for risk modelling of the short rate. The model consists of three components.

1. Autoencoder model. The first step is to apply an Autoencoder model, first proposed by Ballard (1987). The Autoencoder models performs a non-linear dimension reduction which is in line with the common belief that only limited amount of risk factors drive the underlying process of the short rate. We provide a more elaborate explanation of the difference between the linear dimension reduction technique Principal Component Analysis (PCA) and the non-linear dimension reduction technique Autoencoder in Appendix E. Here we also explain the current state-of-the art of Autoencoders and our motivation to use a specific network architecture. The dimension reduction reduces the curse of dimensionality that commonly serves as an error in forecasting applications.

2. GAN model. The second step is to perform a model-free simulation of the low-dimensional representation generated by the Autoencoder model using a GAN.[7] The GAN model allows us to implicitly maximize the likelihood without the need to assume an underlying distribution. We discuss the current state of the art and the used network architecture in Section 4.

3. Long Short Term Memory (LSTM) model. The third step is to match the temporal dynamics of the simulations of the low-dimensional representation produced by the GAN model with the low-dimensional representation of the real data. Hochreiter and Schmidhuber (1997) propose the Long Short Term Memory (LSTM) network architecture to model the temporal dynamics which is comparable to a non-linear variant of the Vector AutoRegression (VAR) model.[8] The LSTM model is trained on the low-dimensional representation of the real data and subsequently used to predict the low-dimensional representation of the simulations. Any deviations between the predictions and the simulations expose inadequacies in the temporal dynamics of the simulations. We provide a more elaborate description of the LSTM and recurrent neural networks in general in Appendix F.

[6] The underlying process can also be modelled with a Vector AutoRegression (VAR)(p) model. Nevertheless, the underlying assumptions would include i.a. stationarity of the individual time series and a linear relation in the lag variables. It is hypothesized that the variables could have a non-linear relationship and therefore the VAR(p) model is not applicable.

[7] As a comparison, simulation can be performed using a variance-covariance method, empirical historical simulation or a Monte Carlo simulation where we assume an underlying distribution of the data. The GAN nevertheless allows us to implicitly maximize the likelihood of the distribution whereby no specification of the underlying distribution is needed. Hence, GAN is referred to as a model-free simulation technique.

[8] Namin and Namin (2018) evaluated the performance of AutoRegressive Integrated Moving Average (ARIMA) models versus LSTM models and noted that the performance of LSTM is superior. VAR models need stationary data due to the linear relationship in autocorrelation and cross-correlation whereas LSTM has the ability to handle non-linear data and also incorporate long term memory which by default decays in normal VAR models.

To assess the ECB's proposed mapping from €STER to EONIA as €STER +8.5bps, we make a more elaborate discussion of Step 2 in the TimeGAN model. Step 2 is based on two models, namely a generator model and a discriminator model. The generator model produces fake simulations and the discriminator model classifies the simulations and real data as either fake or real. The trained generator model is able to fool the discriminator almost surely. The trained discriminator model classifies the ECB's proposed mapping. An in-depth elaboration of the TimeGAN model follows in Section 4.

3 Problem formulation

To assess the risk associated with the EONIA-€STER transition I formulated the following research questions:

> *Does the TimeGAN model applied on short rates provide a better prediction for interest rate risk than traditional simulation techniques and theoretical financial short rate models?*
>
> *Does the ECB's proposed EONIA-€STER mapping provide accurate interest rate risk predictions for EONIA after €STER implementation?*

The research questions are answered in the following steps. Firstly, we conduct the comparison for EONIA. Next, we expand the set of explanatory variables to include macroeconomic and market microstructure risk factors to assess if the interest rate risk prediction improves. After that, we implement three improved optimization techniques to assess the influence on the interest rate risk predictions. Using, the discriminator in the best performing TimeGAN model, we assess the ECB's proposed mapping. Lastly, we adjust the generator according to the EONIA-€STER transition. Using the modified generator and the pre-trained TimeGAN model, we assess the impact of the transition on interest rate predictions.

The plan of the remainder of this thesis is as follows. Section 4 starts off with an in-depth discussion of the GAN and TimeGAN model. Section 5 describes the data and explores theoretical and self-proposed stylized facts. Section 6 continues with the methodology including some deepening analysis. Lastly, Section 7 discusses the results and Section 8 provides the conclusion and proposes further research.

4 GAN literature and proposed model

4.1 Regular GAN and TimeGAN

Goodfellow et al. (2014) introduce the regular GAN. Here, we explain the regular GAN applied on short rates EONIA and €STER and risk factors referred to as data. Let $\mathcal{X}$ be a non-empty set of all possible outcomes of the data at a specific point in time. Let us define the real data $X \in \mathcal{X}^T$ to be the multidimensional time series of length T with k short rate related features, that can be instantiated with specific values denoted by $x_{1:T}$. In the GAN model setup, two models with adversarial objectives are simultaneously trained. The first of the two models in the standard GAN is a generator model that has the objective to generate data that is non-distinguishable from real data. The generator G is defined as the function $G_{\theta_g} : W \to \mathcal{X}^T$, where $W \in \mathbb{R}^T$ follows a time-dependent stochastic Wiener process that can be instantiated with specific values denoted by $w_{1:T}$. The function G_{θ_g} is described by a recurrent neural network with parameters θ_g. The second of the two models is a discriminator model that has the objective to discriminate between real and fake data. The discriminator D is defined as the function $D_{\theta_d} : \mathcal{X}^T \to (0,1)^T$. The function D_{θ_d} is described by a recurrent neural network with parameters θ_d. Figure 2 shows a schematic overview of the standard GAN model.

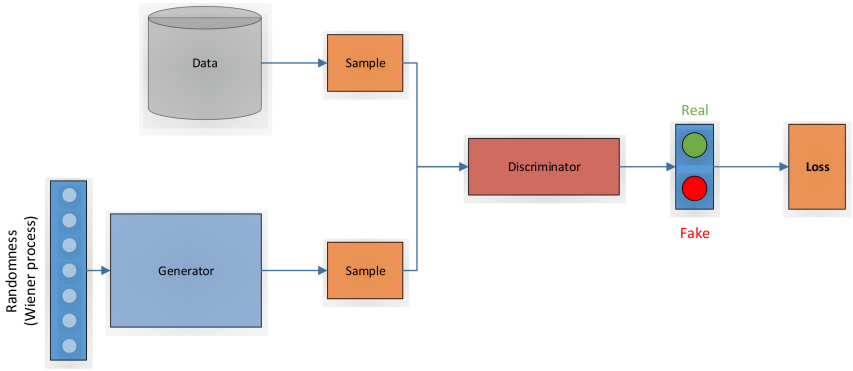

Figure 2. Schematic overview of a regular GAN model as proposed by Goodfellow et al. (2014).

The performance of a GAN is measured as the correct classification of the real and fake data samples by the discriminator. Real data instances $x_{1:T}$ are sampled from the dataset with underlying distribution $p_{\mathcal{X}}(\cdot) \in \text{Prob}(\mathcal{X}^T)$ while fake data instances are generated by applying the generator on samples $w_{1:T}$ coming from underlying distribution $p_W(\cdot) \in \text{Prob}(\mathbb{R}^T)$. The loss function, $\mathcal{L}$, for this classification problem is the binary cross-entropy loss and the resulting objective is a minimax optimization problem,

$$\mathcal{L} = \min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{\mathcal{X}}} \log D_{\theta_d}(x_{1:T}) + \mathbb{E}_{w \sim p_W} \log(1 - D_{\theta_d}(G_{\theta_g}(w_{1:T}))) \right]. \tag{3}$$

The loss function is evaluated using the sample mean

$$\hat{\mathcal{L}} = \min_{\theta_g} \max_{\theta_d} \left[\frac{1}{N} \sum_{i=1}^{N} \log D_{\theta_d}(x_{1:T}^{(i)}) + \sum_{i=1}^{N} \log(1 - D_{\theta_d}(G_{\theta_g}(w_{1:T}^{(i)}))) \right], \tag{4}$$

where i indicates a sample of the dataset. The objective of the generator coincides with the minimization of the loss while the objective of the discriminator coincides with the maximization of the loss.

The only source of feedback in the regular GAN for the generator is the discriminator's classification of the generated data samples. Given that there resides more information in multivariate time series data next to the discriminator's classification, Yoon et al. (2019) propose the Time Series GAN (TimeGAN) model. Here we discuss the TimeGAN applied on short rates. The model uses an intermediate embedding space, $\mathcal{H}^T$, to learn a low-dimensional embedding of dimension $p \times T$ of the real data where $p << k$. Additionally, the generator is trained to exhibit temporal relations equal to the low-dimensional embedding distribution of real data. These two auxiliary objectives are combined with the standard GAN objective, i.e. to generate data that is non-distinguishable from real data. Figure 3 shows the TimeGAN model architecture.

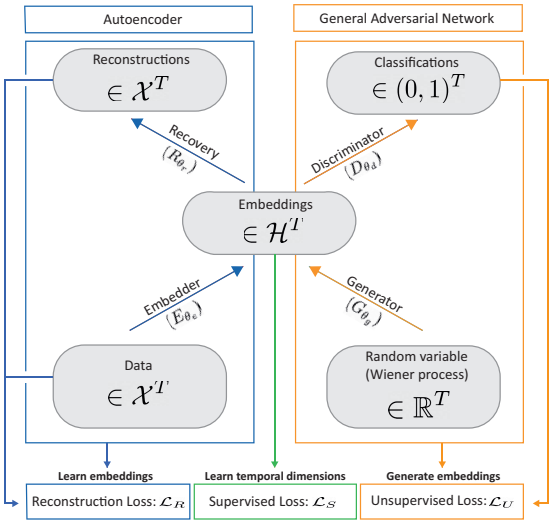

Figure 3. Simplified block diagram of component functions and objectives in TimeGAN applied on short rates. Real data sequences $x_{1:T} \in \mathcal{X}^T$ and random variables $w_{1:T} \in \mathbb{R}^T$ are respectively mapped to latent embeddings $h_{1:T} \in \mathcal{H}^T$ and $\hat{h}_{1:T} \in \mathcal{H}^T$. From embeddings $h_{1:T}$, reconstructions $\tilde{x}_{1:T} \in \mathcal{X}^T$ are computed. The unsupervised learning is performed on the low-dimensional embedding space $\mathcal{H}^T$.

First, let us discuss the Autoencoder part of TimeGAN, which is the left part in Figure 3. The TimeGAN model maps real time series data, $x_{1:T} \in \mathcal{X}^T$, to a low-dimensional embedding space, $\mathcal{H}^T$, using the embedder model. The embedder model is defined as the function $E_{\theta_e} : \mathcal{X}^T \to \mathcal{H}^T$. The function E_{θ_e} is a recurrent neural network with parameters θ_e. In order to map from the low-dimensional embedding space to real space, the recovery model is defined as the function $R_{\theta_r} : \mathcal{H}^T \to \mathcal{X}^T$. The function R_{θ_r} is described by a recurrent neural network with parameter θ_r. The goal of the Autoencoder part is to provide a low-dimensional representation of the high-dimensional feature space. This is justified by the assumption that the process is driven by an underlying low-dimensional representation of the data.

Next, let us discuss the GAN part of TimeGAN, which is the right part in Figure 3. The standard generator G_{θ_g} proposed by Goodfellow et al. (2014) is altered such that the generator maps the random variable $w_{1:T}$ to the low-dimensional embedding space, $\mathcal{H}^T$, instead of a direct mapping to the real space, $\mathcal{X}^T$. Here, the generator is defined as the function $G_{\theta_g} : W \to \mathcal{H}^T$. Given this indirect simulation approach, the TimeGAN model additionally performs representation learning. The representation learning

allows to understand similarities between short rates in a low-dimensional manifold.

In order to apply the GAN potential, the discriminator works in the embedding space, $\mathcal{H}^T$. The discriminator is defined as the function $D_{\theta_d} : \mathcal{H}^T \to (0,1)^T$. The function D_{θ_d} is described by a bidirectional recurrent neural network due to the fact that the classification is post-hoc.[9] Lastly, temporal relations are learned via a supervisor network trained on the embedding space, $\mathcal{H}^T$, as generated by the embedder E_{θ_g} and tested on the embedding space, $\mathcal{H}^T$, as generated by the generator G_{θ_g}. The supervisor is defined as the function $S_{\theta_s} : \mathcal{H}^T \to \mathcal{H}^T$. The function S_{θ_s} is described by a recurrent neural network. Given this setup, TimeGAN is able to simultaneously learn and generate representations while iterating across time and so preserving real-world temporal relations in the generator. The embedder, recovery, generator, and discriminator models are 3-layer Long Short Term Memory (LSTM) recurrent neural networks to avoid gradient vanishing (Hochreiter & Schmidhuber, 1997). The supervisor model is a 2-layer LSTM model.[10]

4.2 Training and optimization of TimeGAN

To preserve a reversible mapping between feature space, $\mathcal{X}^T$, and embedding space, $\mathcal{H}^T$, the Autoencoder part of TimeGAN produces reconstructions $\tilde{x}_{1:T}$ of embeddings $h_{1:T}$ of real data $x_{1:T}$. The loss function is the Mean Squared Error (MSE). The Autoencoder objective is minimization of the reconstruction loss,

$$\mathcal{L}_R = \mathbb{E}_{x_{1:T} \sim p_{\mathcal{X}}} ||x_{1:T} - \tilde{x}_{1:T}||_2, \tag{5}$$

where $\mathcal{L}_R$ is evaluated by the sample mean

$$\hat{\mathcal{L}}_R = \frac{1}{N} \sum_{i=1}^{N} ||x_{1:T}^{(i)} - \tilde{x}_{1:T}^{(i)}||_2. \tag{6}$$

To preserve temporal relations in the embedding space, $\mathcal{H}^T$, the stepwise conditional distribution, $p(h_t|h_{t-1}, ..., h_1)$, is captured by the supervisor model. The probability distribution of $h_{1:T}$ is denoted by $p_{\mathcal{H}}(\cdot) \in \text{Prob}(\mathcal{H})$. The loss function is the MSE between predicted embeddings, $\tilde{h}_t$, and real embeddings, h_t. The supervisor model objective is minimization of the supervised loss,

$$\mathcal{L}_S = \mathbb{E}_{h_{1:T} \sim p_{\mathcal{H}}} ||h_{1:T} - \tilde{h}_{1:T}||_2, \tag{7}$$

where $\mathcal{L}_S$ is evaluated by the sample mean

$$\hat{\mathcal{L}}_S = \frac{1}{N} \sum_{i=1}^{N} ||h_{1:T}^{(i)} - \tilde{h}_{1:T}^{(i)}||_2. \tag{8}$$

Unsupervised GAN training ensures the generator produces fake embeddings, $\hat{h}_{1:T}$. The discriminator provides classifications $y_{1:T}$ for real embeddings and $\tilde{y}_{1:T}$ for fake embeddings. The probability distribution of $\hat{h}_{1:T}$ is denoted by $p_{\hat{\mathcal{H}}} \in \text{Prob}(\hat{\mathcal{H}})$. The loss function is the negative binary cross-entropy. The generator model objective is the minimization of the unsupervised loss while the discriminator model objective is the maximization of the unsupervised loss,

$$\mathcal{L}_U = \mathbb{E}_{h_{1:T} \sim p_{\mathcal{H}}} \log y_{1:T} + \mathbb{E}_{\hat{h}_{1:T} \sim p_{\mathcal{H}}} \log(1 - \tilde{y}_{1:T}), \tag{9}$$

[9] We desribe the inner workings of a bidirectional recurrent neural network in Appendix F.

[10] Yoon et al. (2019) implement a LSTM model, a GRU model and an extension to the LSTM model. The goal of this paper is not address the differences between the recurrent neural network architectures and hence only LSTM is implemented.

where $\mathcal{L}_U$ is evaluated by the sample mean

$$\hat{\mathcal{L}}_U = \frac{1}{N}\sum_{i=1}^{N} \log y_{1:T}^{(i)} + \frac{1}{N}\sum_{i=1}^{N} \log(1 - \tilde{y}_{1:T}^{(i)}). \tag{10}$$

Figure 4 visualizes the resulting TimeGAN training schema. The gradient descent during the back-propagation step is performed by the Adaptive Moment Estimation (Adam) optimizer proposed by Kingma and Ba (2014).[11]

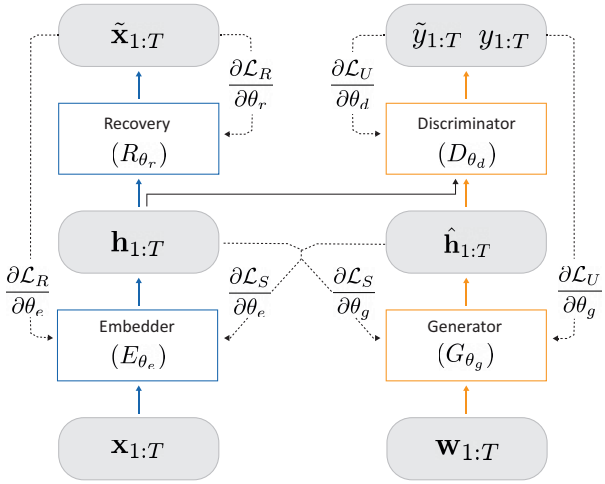

Figure 4. Training scheme for TimeGAN. Solid lines indicate forward propagation of data, and dashed lines indicate backpropagation of gradients.

Yoon et al. (2019) suggest to jointly train the Autoencoder and the supervisor in TimeGAN by optimizing objective function

$$\min_{\theta_e, \theta_r}(\lambda \mathcal{L}_S + \mathcal{L}_R), \tag{11}$$

where $\lambda \geq 0$ is a hyperparameter that balances the supervisor loss, $\mathcal{L}_S$, and recovery loss, $\mathcal{L}_R$. The hyperparameter λ preserves the temporal relation in the embedding space with respect to the recovery function and λ will undergo hyperparameter tuning.[12] The generator, discriminator and the supervisor models are simultaneously jointly trained by optimizing objective function

$$\min_{\theta_g}(\eta \mathcal{L}_S + \max_{\theta_d} \mathcal{L}_U)). \tag{12}$$

where $\eta > 0$ is a hyperparameter that balances the unsupervised loss, $\mathcal{L}_U$, and the supervisor loss, $\mathcal{L}_S$. The hyperparameter η preserves the temporal relations in the generated embedding space and η will undergo hyperparameter tuning.[13] By jointly optimizing the objectives in (11) and (12), TimeGAN learns to embed (short rate and risk factors), generate (embedding representations), and iterate (across time) low-dimensional embeddings.

[11] Adaptive Moment Estimation (Adam) is a stochastic gradient descent that combines momentum and adaptive learning rates per parameter and has shown good performance amongst a lot of optimizers when applied on a lot of different datasets. The hyperparameter β_1 controls the momentum and the hyperparameter β_2 controls the adaptive learning rates. For a more in-depth analysis of the hyperparameters β_1 and β_2 or the performance of Adam, we refer to Kingma and Ba (2014).

[12] Yoon et al. (2019) state that λ does not effect the results. Nevertheless, the model setup in this study is different and hence a difference could be present. The initial value is fixed at $\lambda = 1$.

[13] The paper uses $\eta = 10$, but states that the performance is insensitive to η. Given that the objective is to preserve the temporal dynamics and simulate new short rates simultaneously. The initial value is fixed at $\eta = 1$.

4.3 Interpretation of GAN loss

To interpret the unsupervised loss, $\mathcal{L}_U$, for the GAN part of TimeGAN, two theorems introduced by Goodfellow et al. (2014) are introduced.

Theorem 1. *For a given generator G, the optimal discriminator D_G^* is:*

$$D_G^*(h_{1:T}) = \frac{p_{\mathcal{H}}(h_{1:T})}{p_{\mathcal{H}}(h_{1:T}) + p_{\hat{\mathcal{H}}}(h_{1:T})} \tag{13}$$

Proof. The objective for discriminator D_G, given generator G, is to maximize (3) with respect to θ_d, i.e.:

$$\begin{aligned}
&\max_{\theta_d} \left[\mathbb{E}_{h_{1:T}\sim p_{\mathcal{H}}} \log D_{\theta_d}(h_{1:T}) + \mathbb{E}_{w_{1:T}\sim p_W} \log(1 - D_{\theta_d}(G_{\theta_g}(w_{1:T})))\right] \\
= &\max_{\theta_d} \left[\int_{h_{1:T}} p_{\mathcal{H}}(h_{1:T}) \log(D_{\theta_d}(h_{1:T}))dh_{1:T} + \int_{w_{1:T}} p_W(w_{1:T}) \log(1 - D_{\theta_d}(G_{\theta_g}(w)))dw_{1:T}\right] \\
= &\max_{\theta_d} \left[\int_{h_{1:T}} p_{\mathcal{H}}(h_{1:T}) \log(D_{\theta_d}(h_{1:T})) + p_{\hat{\mathcal{H}}}(h_{1:T}) \log(1 - D_{\theta_d}(h_{1:T}))dh_{1:T}\right]
\end{aligned}$$

Therefore, the following must hold.

$$\frac{p_{\mathcal{H}}(h_{1:T})}{D_G^*(h_{1:T})} - \frac{p_{\hat{\mathcal{H}}}(h_{1:T})}{1 - D_G^*(h_{1:T})} = 0$$

Given that $(p_{\mathcal{H}}(h_{1:T}), p_{\hat{\mathcal{H}}}(h_{1:T})) \in \mathbb{R}^2 \backslash \{0, 0\}$, the function achieves its maximum for $D_g^*(h_{1:T})$ at (13). ■

Theorem 2. *The global minimum of* (3) *is found when $p_{\mathcal{H}} = p_{\hat{\mathcal{H}}}$. At that point* (3) *achieves the value* $-2\log 2$.

Proof. First, let us insert the optimal discriminator D_G^* into the minimax optimization problem.

$$\begin{aligned}
&\max_{\theta_d} \left[\mathbb{E}_{h_{1:T}\sim p_{\mathcal{H}}} \log D_{\theta_d}(h_{1:T}) + \mathbb{E}_{w_{1:T}\sim p_W} \log(1 - D_{\theta_d}(G_{\theta_g}(w_{1:T})))\right] \\
= &\ \mathbb{E}_{h_{1:T}\sim p_{\mathcal{H}}} \log D_G^*(x) + \mathbb{E}_{w_{1:T}\sim p_W} \log(1 - D_G^*(G_{\theta_g}(w_{1:T}))) \\
= &\ \mathbb{E}_{h_{1:T}\sim p_{\mathcal{H}}} \log \frac{p_{\mathcal{H}}(h_{1:T})}{p_{\mathcal{H}}(h_{1:T}) + p_{\hat{\mathcal{H}}}(h_{1:T})} + \mathbb{E}_{h_{1:T}\sim p_{\hat{\mathcal{H}}}} \log \frac{p_{\hat{\mathcal{H}}}(h_{1:T})}{p_{\mathcal{H}}(h_{1:T}) + p_{\hat{\mathcal{H}}}(h_{1:T})}
\end{aligned}$$

For $p_{\mathcal{H}} = p_{\hat{\mathcal{H}}}$, $D_G^*(h_{1:T}) = \frac{1}{2}$. Hence the global minimum of (3) is:

$$= \mathbb{E}_{h_{1:T}\sim p_{\mathcal{H}}} \log \frac{1}{2} + \mathbb{E}_{h_{1:T}\sim p_{\hat{\mathcal{H}}}} \log \frac{1}{2} = -2\log 2$$

This means that the value of (3) when optimizing with respect to θ_g for an optimized discriminator D_G^* equals to:

$$= -2\log 2 + \mathrm{KL}(p_{\mathcal{H}} || \frac{p_{\mathcal{H}} + p_{\hat{\mathcal{H}}}}{2}) + \mathrm{KL}(p_{\hat{\mathcal{H}}} || \frac{p_{\mathcal{H}} + p_{\hat{\mathcal{H}}}}{2}) = -2\log 2 + 2 \cdot \mathrm{JSD}(p_{\mathcal{H}} || p_{\hat{\mathcal{H}}})$$

where $\mathrm{KL}(a||b)$ denotes the Kullback-Leibler Divergence (KLD) between probability distribution a and b.[14] $\mathrm{JSD}(a||b)$ denotes the Jensen-Shannon Divergence (JSD) between probability distribution a and b. Given that $\mathrm{JSD}(a||b) \geq 0$, the optimization of the generator G_{θ_g} with a perfect discriminator $D_{\theta_d}^*$ coincides with the optimization of the JSD. ■

Theorem 2 provides the lower bound of $-2\log 2$ for the unsupervised loss $\mathcal{L}_U$. The lower bound for recovery loss $\mathcal{L}_R$ and supervised loss $\mathcal{L}_S$ is equal to 0 which is self-evident.

[14] The division by 2 in the second argument of both Kullback-Leibler Divergence (KLD) terms is due to the fact that the argument should be a proper probability distribution.

4.4 Improved optimization of TimeGAN

Optimization of (3) is known to be slow and difficult, see Salimans et al. (2016) and Hazan et al. (2017). The generator and discriminator are trained simultaneously to find a Nash equilibrium to a two-player non-cooperative game. Parameters θ_g and θ_d are updated independently with no respect to the other player in the game. Updating the gradient of both models concurrently cannot guarantee convergence. The first technique for improved optimization focuses on the definition of the loss function $\mathcal{L}_U$ for the GAN part of TimeGAN. Arjovsky et al. (2017) propose the Wasserstein-1 distance that has a smoother and more interpretable value space in comparison to KLD and JSD. Regular GAN training optimizes the asymmetric KLD and for a perfect discriminator the symmetric JSD is optimized as we prove in Theorem 2. Nevertheless, a completely non-overlapping $p_{\mathcal{H}}(\cdot)$ and $p_{\hat{\mathcal{H}}}(\cdot)$ imposes a saturated gradient problem for both KLD and JSD. The Wasserstein-1 distance is informally defined as the "cost" of the optimal transport plan to move all "mass" from distribution $p_{\mathcal{H}}(\cdot) \in \text{Prob}(\mathcal{H}^T)$ to distribution $p_{\hat{\mathcal{H}}}(\cdot) \in \text{Prob}(\hat{\mathcal{H}}^T)$. A transport plan is defined as the joint distribution $\gamma(h_{1:T}, \hat{h}_{1:T})$ whose marginals are respectively $p_{\mathcal{H}}$ and $p_{\hat{\mathcal{H}}}$. The infimum over the set of joint distributions is the Wasserstein-1 distance,

$$W(p_{\mathcal{H}}, p_{\hat{\mathcal{H}}}) = \inf_{\gamma \in \prod(p_{\mathcal{H}}, p_{\hat{\mathcal{H}}})} \mathbb{E}_{(h_{1:T}, \hat{h}_{1:T}) \sim \gamma} \left[||h_{1:T} - \hat{h}_{1:T}|| \right], \tag{14}$$

For all joint distributions $\gamma(h_{1:T}, \hat{h}_{1:T})$, the distance is the ℓ-1 norm between the real data, $h_{1:T}$, and the generated data, $\hat{h}_{1:T}$. It is impossible to evaluate $\inf_{\gamma \in \prod(p_{\mathcal{H}}, p_{\hat{\mathcal{H}}})}$ analytically and therefore Arjovsky et al. (2017) use the Kantorovic-Rubinstein duality,

$$W(p_{\mathcal{H}}, p_{\hat{\mathcal{H}}}) = \sup_{||f||_L \leq 1} \mathbb{E}_{h_{1:T} \sim p_{\mathcal{H}}}[f(h_{1:T})] - \mathbb{E}_{\hat{h}_{1:T} \sim p_{\hat{\mathcal{H}}}}[f(\hat{h}_{1:T})]. \tag{15}$$

where $||f||_L \leq 1$ represents a set of 1-Lipschitz functions. The discriminator is the function f_{θ_d} where θ_d refers to the parametrization of the recurrent neural network. Given the depth of the discriminator model architecture, we assume that the discriminator can approximate all 1-Lipschitz functions. Therefore, the maximum approximates the supremum from (15) almost surely,

$$W(p_{\mathcal{H}}, p_{\hat{\mathcal{H}}}) = \max_{\theta_d : ||f_{\theta_d}||_L \leq 1} \mathbb{E}_{h_{1:T} \sim p_{\mathcal{H}}}[f_{\theta_d}(h_{1:T})] - \mathbb{E}_{\hat{h}_{1:T} \sim p_{\hat{\mathcal{H}}}}[f_{\theta_d}(\hat{h}_{1:T})]. \tag{16}$$

Taking the maximum yields the Wasserstein GAN (WGAN) loss function,

$$W(p_{\mathcal{H}}, p_{\hat{\mathcal{H}}}) = \mathcal{L}_U^{WGAN} = \mathbb{E}_{h_{1:T} \sim p_{\mathcal{H}}}[D_{\theta_d}(h_{1:T})] - \mathbb{E}_{\hat{h}_{1:T} \sim p_{\hat{\mathcal{H}}}}[D(\hat{h}_{1:T})], \tag{17}$$

where $\mathcal{L}_U^{WGAN}$ is evaluated using the sample mean

$$\hat{\mathcal{L}}_U^{WGAN} = \frac{1}{N} \sum_{i=1}^{N} D_{\theta_d}(h_{1:T}^{(i)}) - D_{\theta_d}(\hat{h}_{1:T}^{(i)}) \tag{18}$$

Gulrajani et al. (2017) note that the violation of the 1-Lipschitz constraint hurts performance and propose Wasserstein GAN with Gradient Penalty (WGAN-GP). They impose a constraint on the gradient for $||\nabla_{\tilde{h}_{1:T}} D_{\theta_d}(\tilde{h}_{1:T})|| \neq 1$, where $\tilde{h}_{1:T}$ is a convex combination between $h_{1:T}$ and $\hat{h}_{1:T}$. The probability distribution of $\tilde{h}_{1:T}$ is denoted by $p_{\tilde{\mathcal{H}}} \in \text{Prob}(\tilde{\mathcal{H}})$. Gulrajani et al. (2017) implement a ℓ-2 regularization term on the constraint which yields the WGAN-GP loss function,

$$\mathcal{L}_U^{WGAN-GP} = \mathbb{E}_{h_{1:T} \sim p_{\mathcal{H}}}[D_{\theta_d}(h_{1:T})] - \mathbb{E}_{\hat{h}_{1:T} \sim p_{\hat{\mathcal{H}}}}[D_{\theta_d}(\hat{h}_{1:T})] + \lambda \, \mathbb{E}_{\tilde{h}_{1:T} \sim p_{\tilde{\mathcal{H}}}}[(||\nabla_{\tilde{h}_{1:T}} D_{\theta_d}(\tilde{h}_{1:T})|| - 1)^2], \tag{19}$$

where we evaluate $\mathcal{L}_U^{WGAN-GP}$ by the sample mean,

$$\hat{\mathcal{L}}_U^{WGAN-GP} = \frac{1}{N}\sum_{i=1}^{N}\left[D_{\theta_d}(h_{1:T}^{(i)}) - D(\hat{h}_{1:T}^{(i)}) + \lambda(||\nabla_{\tilde{h}_{1:T}} D_{\theta_d}(\tilde{h}_{1:T}^{(i)})|| - 1)^2\right], \tag{20}$$

where λ is a hyperparameter that balances the gradient penalty and the Wasserstein-1 loss. The value $\lambda = 10$ is implemented as suggested by Gulrajani et al. (2017). The GAN objective from (12) is updated to include the WGAN-GP loss,

$$\min_{\theta_g}(\eta\mathcal{L}_S + (\max_{\theta_d}\mathcal{L}_U^{WGAN-GP})). \tag{21}$$

The second technique for improved optimization is Feature Matching as proposed by Salimans et al. (2016). Feature matching is a technique that inspects whether the generator's output, $\hat{h}_{1:T}$, matches expected statistics of real data embeddings, $h_{1:T}$.[15] The statistical difference between the real and generated data samples is the Feature Matching loss,

$$\mathcal{L}_{FM} = ||\,\mathbb{E}_{h_{1:T}\sim p_{\mathcal{H}}} f(h_{1:T}) - \mathbb{E}_{\hat{h}_{1:T}\sim p_{\hat{\mathcal{H}}}} f(\hat{h}_{1:T}))||_2, \tag{22}$$

where f denotes the function to compute the statistic which we specify later. The Feature Matching loss is evaluated by the sample mean,

$$\hat{\mathcal{L}}_{FM} = \frac{1}{N}||\sum_{i=1}^{N} f(h_{1:T}^{(i)}) - \sum_{i=1}^{N} f(\hat{h}_{1:T}^{(i)})||_2. \tag{23}$$

Yoon et al. (2019) implement feature matching with mean and variance as statistics. The objective of this study is to predict short rate risk and therefore feature matching is implemented with skewness,

$$f(h_{1:T}) = \mathbb{E}_{h_{1:T}}\left(\frac{h_{1:T} - \mathbb{E}[h_{1:T}]}{\sigma[h_{1:T}]}\right)^3,$$

and kurtosis,

$$f(h_{1:T}) = \mathbb{E}_{h_{1:T}}\left(\frac{h_{1:T} - \mathbb{E}[h_{1:T}]}{\sigma[h_{1:T}]}\right)^4.$$

The GAN objective as defined in (12) is updated to include feature matching,

$$\min_{\theta_g}(\eta\mathcal{L}_S + +\kappa\mathcal{L}_{FM} + (\max_{\theta_d}\mathcal{L}_U)) = \max_{\theta_g}(-\eta\mathcal{L}_S - \kappa\mathcal{L}_{FM} + (\min_{\theta_d} -\mathcal{L}_U)). \tag{24}$$

where $\kappa \geq 0$ is a hyperparameter that balances the losses.

The third technique for improved optimization is label smoothing. Regular GAN training defines the label 1 for real data and 0 for fake data. Label smoothing softens the label values, e.g. $\alpha = 1 - x$ for real and $\beta = x$ for fake where $0 < x < 0.5$. Hazan et al. (2017) have shown that label smoothing reduces the tendency to mode collapse. The optimal discriminator $D^*_{\theta_d}(h_{1:T})$ from (13) is updated,

$$D^*_{\theta_d}(h_{1:T}) = \frac{\alpha \cdot p_{\mathcal{H}}(h_{1:T}) + \beta \cdot p_{\hat{\mathcal{H}}}(h_{1:T})}{p_{\mathcal{H}}(h_{1:T}) + p_{\hat{\mathcal{H}}}(h_{1:T})}.$$

[15] During model collapse, the generator only produces one specific sample which the discriminator classifies as a real data sample. The discriminator Nevertheless, if you compare the statistics of the real and fake data samples, you would note thesee those are different. Hence, feature matching is a way to avoid mode collapse. Mode collapse can potentially also be avoided by using a Variational Autoencoder. Nevertheless, the Variational Autoencoder also has a generative nature due to stochasticity in the recovery network. Therefore we could not completely isolate the generative effect of the GAN part in TimeGAN and hence we did not implement this.

Hazan et al. (2017) state that the presence of $p_{\hat{\mathcal{H}}}(h_{1:T})$ in the numerator is problematic because, in areas where $p_{\mathcal{H}}(h_{1:T})$ is approximately zero and $p_{\hat{\mathcal{H}}}(h_{1:T})$ is large, erroneous samples from $p_{\hat{\mathcal{H}}}(h_{1:T})$ have no incentive to move nearer to the data. Therefore Hazan et al. (2017) propose to only smooth positive labels to α, leaving negative labels set to 0, i.e. $\beta = 0$. This yields the new optimal discriminator $D^*_{\theta_d}(h_{1:T})$,

$$D^*_{\theta_d}(h_{1:T})\Big|_{\beta=0} = \frac{\alpha \cdot p_{\mathcal{H}}(h_{1:T})}{p_{\mathcal{H}}(h_{1:T}) + p_{\hat{\mathcal{H}}}(h_{1:T})}. \tag{25}$$

4.5 Algorithm pseudocode

Algorithm 1 in Appendix G shows the pseudocode for the pre-training of the TimeGAN applied on short rates. Algorithm 2 in Appendix G shows the pseudocode for the actual training of TimeGAN applied on short rates. None of the improved optimization techniques proposed in Section 4.4 are described in the pseudocode to ensure readability. Also note that the generator optimization is different from Equation 12.[16] Python code for implementation of TimeGAN applied on short rates can be found at https://github.com/Larsterbraak/TimeGAN. All training simulations are performed for a total of N training iterations over the complete dataset, called epochs.[17] For each epoch, the dataset is split up in multiple batches of size m, called minibatches. The backpropagation is then performed for every minibatch in the dataset.

[16] In the first steps of the optimization, the generator will produce easy to classify fake data samples. In order to avoid saturating gradients with respect to θ_g in this stadium the objective $\min_{\theta_g}(\eta\mathcal{L}_S + \mathcal{L}_U)$ is changed to $\max_{\theta_g}(-\eta\mathcal{L}_S + -\mathcal{L}_U)$. This trick is common in the literature. The maximization of $\log(x)$ produces far better gradients for a bad generator.

[17] An epoch is a term used in machine learning and indicates the number of passes of the entire training dataset the machine learning algorithm has completed.

5 Data

At the time of writing, EONIA data ranges from January 4th, 1999 until March 12th, 2020. Pre-€STER data ranges from March 15th, 2017 to September 30th, 2019. €STER data ranges from October 1st, 2019 until March 12th, 2020. Figure 5 shows the 25th and 75th percentile, the transaction volume, number of transactions, and €STER during pre-€STER period.[18]

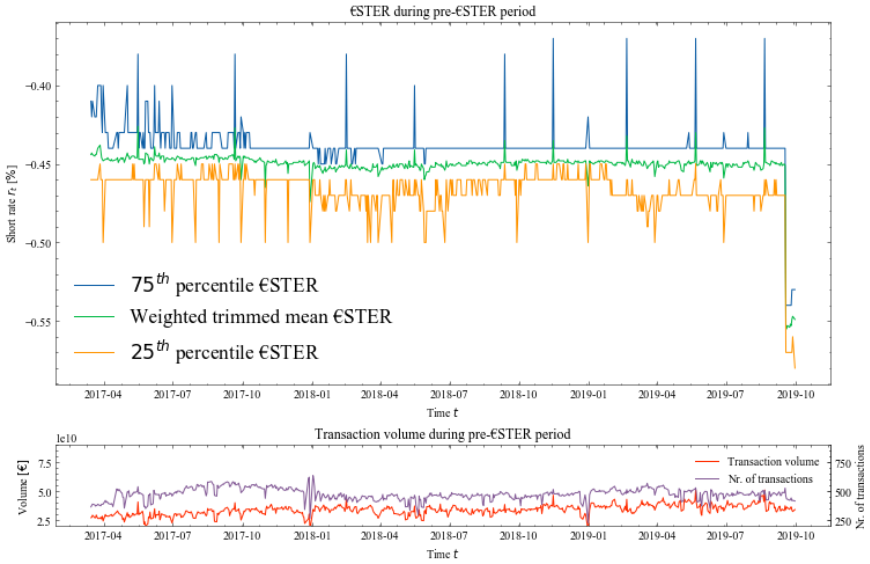

Figure 5. €STER during pre-€STER period. The weighted trimmed mean of panel bank contributions, i.e. €STER: The transactions with lower than 25th percentile and higher than 75th percentile volume are trimmed. Subsequently, the volume-weighted mean of the remaining 50% transactions is calculated.

Figure 6 shows the comparison across EONIA, pre-€STER and €STER since inception of pre-€STER on March 15th, 2017. Figure 7 shows the histogram for the daily differences in EONIA, pre-€STER, and €STER. The figure shows fatter tails for EONIA than for €STER.

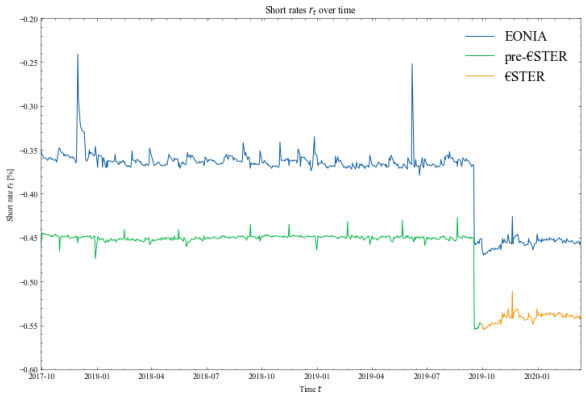

Figure 6. EONIA, pre-€STER and €STER over the period September 30th, 2017 until March 12th, 2020. Pre-€STER period motivated the ECB to implement a 8.5bps spread between EONIA and €STER.

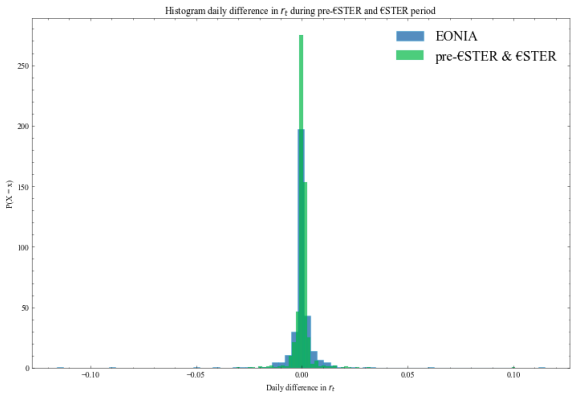

Figure 7. Histogram of daily differences EONIA, pre-€STER and €STER over the period September 30th, 2017 until March 12th, 2020. EONIA shows fatter tails than €STER short rate.

[18] EONIA and €STER data is published in the ECB's statistical data warehouse.

Figure 8 shows inflation as measured by the Harmonised Index of Consumer Prices (HICP) and the real GDP as defined in (1).[19] Figure 9 shows the liquidity estimate over the period 2004 to 2019 as calculated by (2) for different tenors.[20] The spreads are very noisy and manual inspection of the data reveals it contains fat-finger outliers.[21] Nevertheless, both the illiquidity in the Great Financial Crisis (GFC) and the decrease in spread but not total exclusion of spread after the GFC is clearly present.

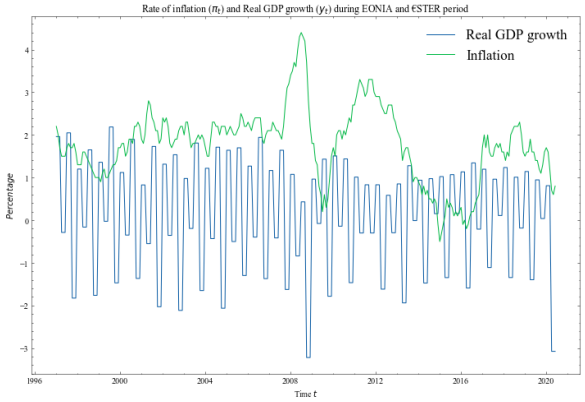

Figure 8. Monthly annualized inflation as measured by the HICP and the seasonal quarterly real GDP growth in the Euro area during the period 1997-2019.

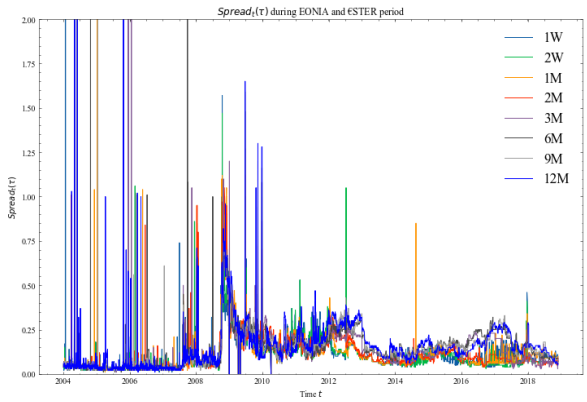

Figure 9. Euribor $\text{Spread}_t(\tau)$ during the period 2004-2019 for the tenors of 1 week, 2 weeks, 1 month, 2 months, 3 months, 6 months, 9 months, and 12 months.

It is hypothesized that the differences between EONIA and €STER exceed differences measurable in the first two statistical moments and we therefore need a non-linear model-free simulation technique to capture the distributional differences accurately. We perform a comparison between EONIA and €STER based on stylized facts known in industry and self-proposed stylized facts.[22] The tendency to negative returns is measured by **Skewness (S)**. The lower the skewness, the higher the likelihood of negative returns and vice versa. The concentration is measured by **Kurtosis (K)**. The higher the kurtosis, the fatter the tails of the return distribution are. Mean reversion is measured by the **Hurst exponent (H)**, as proposed by Hurst (1951). It measures the persistence in a time series, i.e. long-term memory. The higher the Hurst exponent, the stronger a time series tends to regress to the mean. **Stationarity (A)** is measured by the Augmented Dickey Fuller (ADF) test, as proposed by Dickey and Fuller (1979). The higher the p-value of the ADF test, the higher the likelihood of a non-stationary time series. Apart from these known stylized facts, we propose three stylized facts. We introduce the Season and Trend decomposition using LOESS (STL) as proposed by Wang et al. (2006). The STL decomposition allows us

[19] The ECB defines the HICP as the inflation target and the annualized rate of change in the price per month, i.e. annualized inflation, is published by the statistical agency Eurostat. HICP data is published on the Eurostat website. The researcher acknowledges the existence of other inflation-based measures such as the Consumer Price Index (CPI), the Producer Price Index (PPI), Spot market Commodity Price Inflation (PCOM) and other real-activity based measures such as e Index of Help Wanted Advertising in Newspapers(HELP), the rate of unemployment (UE), the growth rate of employment (EMPLOY), and the growth rate in industrial production (IP) as used in Ang and Piazzesi (2003) but chooses to include only HICP and real GDP growth because it is hypothesized that these are accurate aggregates.

[20] Euribor bid-ask spreads data for different tenors are published at the European Money Markets Institute website.

[21] It is hypothesized that dropout and normalization based on percentiles instead of minimum and maximum values in the TimeGAN model decreases the influence of outliers on the model robustness.

[22] All stylized facts are calculated based on the daily returns of the short rate to ensure comparability.

to measure the strength of the different components of the time series. The STL decomposition measures the **Strength of trend (FT)**, F_t, and **Strength of seasonality (FS)**, F_s. A higher value for F_t and F_s indicates a stronger relation. We use the definition of Hyndman et al. (2015) for the **spikiness (SP)** of daily returns. A higher value indicates a more spiky distribution. Table 3 shows the stylized facts for EONIA, EONIA during the pre-€STER time period, pre-€STER, the ECB's mapping and €STER.

We observe that pre-€STER returns have a lower variance[23], a higher skewness and a higher kurtosis than the ECB's proposed mapping. Therefore, pre-€STER has more tendency to produce positive returns while the ECB mapping is more similar to a normal distribution. The Hurst exponent remains relatively constant implying no change in the mean reversion. Furthermore, we observe that pre-€STER returns are non-stationary whereas the ECB's proposed mapping is stationary. Pre-€STER returns have a higher value for Strength of Seasonality and a lower value for Spikiness than the ECB's proposed mapping.

Table 3. *Stylized facts for daily returns in EONIA, pre-€STER, EONIA during pre-€STER period, ECB's proposed mapping of €STER = EONIA - 8.5bps, and €STER.*

Short rate	N	V	S	K	H	A	FT	FS	SP
EONIA	5425	0.530	22.49	2013.74	0.001	0.00	0.89	0.968	6.53 e-10
EONIA (pre-€STER)	648	9.0 e-4	4.13	104 .71	-0.025	0.00	0.00	0.336	3.86 e-11
pre-€STER	648	1.5 e-4	9.23	167.09	-0.028	0.93	0.00	0.109	8.59 e-12
ECB mapping	648	5.8 e-4	3.29	95.72	-0.025	0.00	0.00	0.241	8.62 e-11
ESTER	114	9.0 e-5	0.25	22.37	-0.220	0.00	0.00	0.117	1.36 e-11

N: Number of data points. V: Variance. S: Skewness, K: Kurtosis, H: Hurst exponent, A: p-value of ADF test. FT: Strength of trend, FS: Strength of seasonality, SP: Spikiness.

6 Methodology

TimeGAN's EONIA VaR estimates and TimeGAN's diversity of EONIA simulations are evaluated for the model specifications proposed in Section 4.4. Figure 10 shows the train-test split for the dataset.

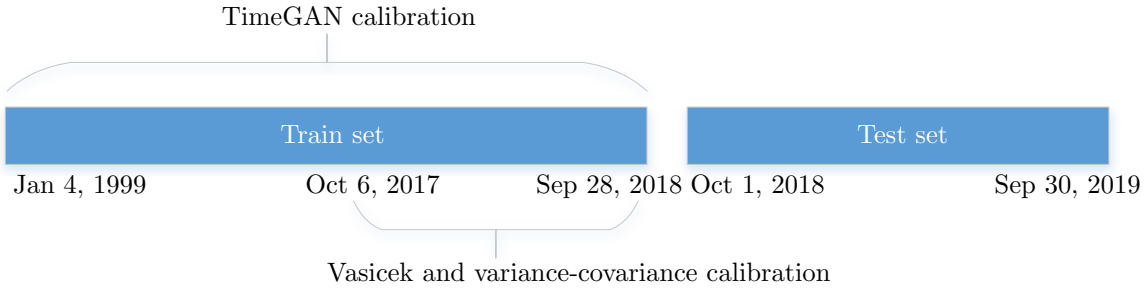

Figure 10. Train-test split for EONIA and EONIA risk factors dataset as specified in Table 2.

TimeGAN calibration is performed on the full training set. For comparison, a Vasicek model and

[23]The EONIA (pre-€STER) returns are non-normal. (Jarque-Bera test (Jarque & Bera, 1980) with both p-values of 0.0.). Therefore, the equality of variance is tested using the Levene test (Levene, 1960). The p-value is 0.0 and therefore we reject the equality of variances.

a variance-covariance technique are calibrated on the last 250 days of the training set.[24] The Vasicek model is only calibrated on EONIA and differences between TimeGAN VaR and Vasicek VaR estimates provide an insight in the effect of macroeconomic and market microstructure variables on VaR estimation. The variance-covariance technique assumes that the risk factors follow a multivariate normal distribution and have a linear effect on EONIA. The difference between TimeGAN VaR estimates and the variance-covariance VaR estimates provides an insight in the effect of a non-linear model in comparison to a linear model in the risk factors. The calibrated TimeGAN[25] model produces $10,000$ simulations of 20 days of EONIA and EONIA risk factors of which T-VaR at the 99.5^{th} percentile for EONIA is computed.[26] This procedure is repeated for 250 trading days in the test dataset and the T-VaR estimates are evaluated by backtesting with Basel Committee's Coverage Test (Kupiec, 1995).[27]

To qualitatively evaluate overfitting and mode collapse during training, Borji (2019) proposes to inspect the Nearest Neighbours (NN) of generated samples in terms of MSE with respect to real EONIA data. The simulated short rate samples, denoted by $R_{\theta_r}(G_{\theta_g}(\mathbf{w}_{1:T}))$, have a NN in the real EONIA data in terms of Mean Squared Error (MSE),

$$\underset{\mathbf{x}_{1:T}\in\mathcal{X}^T}{\arg\min}\, ||\mathbf{x}_{1:T} - R_{\theta_r}(G_{\theta_g}(\mathbf{w}_{1:T}))||_2. \tag{28}$$

The diversity of TimeGAN simulations is measured as the number of distinct nearest neighbours, denoted by D_{NN}, with respect to real EONIA data for 250 simulations of T days for a calibrated TimeGAN. As a qualitative metric of diversity the temporal dimension in $\mathbf{x}_{1:T}$ and $R_{\theta_r}(G_{\theta_g}(\mathbf{w}_{1:T}))$ is flattened using the dimension reduction technique t-distributed Stochastic Neighbor Embedding (t-SNE).[28]

Training TimeGAN provides a discriminator which is capable of distinguishing between real and fake EONIA data sample. The discriminator is applied on data samples 20 trading days of pre-€STER +8.5bps to validate the ECB's proposed mapping from €STER to EONIA. The discriminator provides probabilities for classification of realness, denoted by $\hat{r}_t$, for every trading day in the data sample. Given that almost every trading day is present in multiple data samples $\mathbf{x}_{1:T}$, the realness score for a specific t is the arithmetic average of all realness score predictions. Figure 18 shows how the results are displayed.

It is hypothesized there exists a relation between the stylized facts described in Table 3, and the realness score $\hat{r}_t$. If the non-linear stylized facts also hold predictive power next to the linear stylized facts, the use of TimeGAN can be justified. The stylized facts are regressed on the realness score,

$$r_t = \alpha + \beta_0 \cdot V + \beta_1 \cdot S + \beta_2 \cdot K + \beta_3 \cdot H + \beta_4 \cdot A + \beta_5 \cdot FT + \beta_6 \cdot FS + \beta_7 \cdot SP + \epsilon. \tag{29}$$

[24] The Vasicek model is calibrated by Kalman-Filter estimation as proposed by Amin (2012).

[25] The number of dimensions used for the dimension reduction step in the TimeGAN will be a hyperparameter.

[26] A description and derivation of the risk metric VaR can be found in Appendix D.B.

[27] For the calculation of VaR in the Vasicek model we solve the stochastic differential equation $dr_t = a(b - r_t)dt + \sigma dW_t$ using the method of Integrating Factors. Under the P-measure, Equation 26 gives an expression for the mean and variance of the short rate r_t. Equation 27 gives an expression for the VaR at confidence level p. The $\pm$ is used in interest rate VaR calculations because both an upwards as well as an downwards move poses risk for a portfolio.

$$\mathbb{E}[r_t] = r_0 e^{-at} + b(1 - e^{-at}) \quad \text{Var}[r_t] = \frac{\sigma^2}{2a}(1 - e^{-2at}) \tag{26}$$

$$\text{VaR}_p(r_t) = \mathbb{E}[r_t] \pm \sqrt{\text{Var}[r_t]}\Phi^{-1}(p) \tag{27}$$

[28] A derivation of the t-SNE algorithm as proposed by Maaten and Hinton (2008) can be found in Appendix D.A.

7 Results

Source code is written in TensorFlow 2.2 and Python 3.6.0 and computations are performed on the LISA high performance computer cluster.[29] Source code can be found in Appendix C.

7.1 Model selection

The hyperparameter optimization results are based on 5-fold cross-validation on the EONIA training data set. We train autoencoder architectures for different dropout regularization parameters, different number of layers and different number of latent space dimensions as to find an optimal parametrization of the autoencoder. Figure 11 shows the mean recovery loss for the autoencoder fitted on the training data for the different network architectures in the hypothesis space conditioned on a dropout of 0.1.[30]

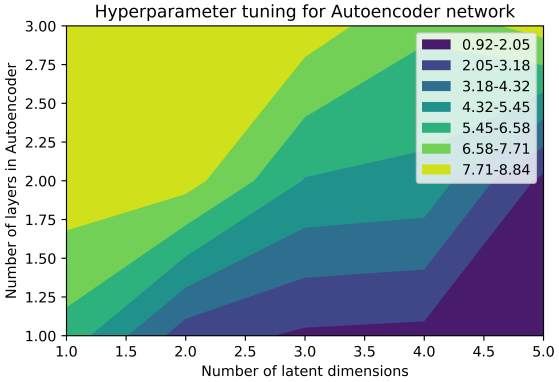

Figure 11. Recovery loss for different number of layers and different number of latent dimensions in autoencoder conditioned on dropout of 0.1.

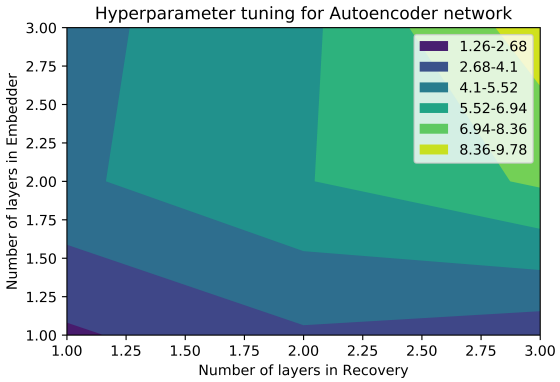

Figure 12. Recovery loss for different number of layers in the Embedder and Recovery conditioned on dropout of 0.1 and latent space dimension of 3.

Figure 11 shows that the 1-layer Embedder and 1-layer Recovery combined with three latent variables results in a low recovery loss for a relatively low latent space dimension.[31] Figure 12 shows the mean recovery loss for the autoencoder fitted on the training data for different number of layers in the Embedder and Recovery conditioned on dropout of 0.1 and latent space dimension of 3. Figure 12 shows that the 1-layer Embedder and 1-layer Recovery results in the lowest recovery loss. We choose a 1-layer LSTM as the architecture for the Embedder network and the Recovery network. We set the dropout regularization parameter at 0.1 and the latent space dimensionality at 3. Figure 13 shows the latent space representation in the optimal autoencoder for the period October 6$^{\text{th}}$, 2017 until November 2$^{\text{nd}}$, 2017 including EONIA.[32]

We train Supervisor architectures for different dropout regularization parameters and different number of layers based on the latent space representation learned by the optimal autoencoder model. Figure 14 shows the mean supervisor loss for the Supervisor fitted on the training data for different architectures.

[29] The LISA computer cluster contains 4 NVIDIA TITAN RTX GPUs that are used in parallel in distributed training.

[30] The 3-layer Embedder model consists of respectively 10 units, 7 units, and hidden dimension units in the top to lower layer of the stacked LSTM. The 2-layer Embedder removes the 7 units layer and the 1-layer Embedder model consists of hidden dimension units. The Recovery model has the exact opposite network architecture compared the Embedder model.

[31] Figure 24 and 25 in Appendix B show that a higher dropout regularization parameter results in a higher recovery loss.

[32] Appendix B.B shows that for higher latent space dimensions the latent variables start to co-move.

Figure 14 shows that the 1-layer Supervisor conditioned on a dropout regularization parameter of 0.1 results in the lowest supervisor loss. We choose a Generator that has more capacity than the Discriminator in terms of stacked LSTM layers. We implement a bidirectional LSTM for the Discriminator architecture because the classification does not have to adhere to normal temporal dynamics. Final architectures for all TimeGAN models are shown in Appendix H.[33]

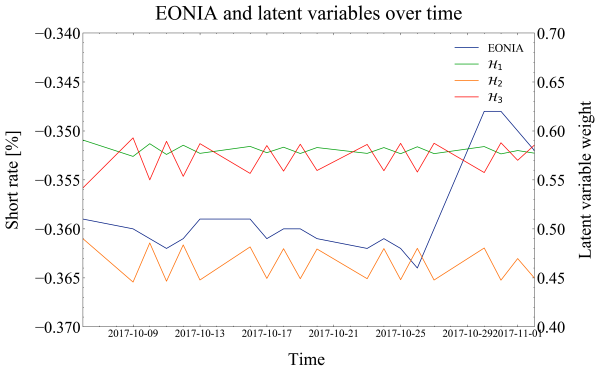

Figure 13. Latent space representations learned by optimal autoencoder for the period October 6$^{\text{th}}$, 2017 until November 2$^{\text{nd}}$, 2017.

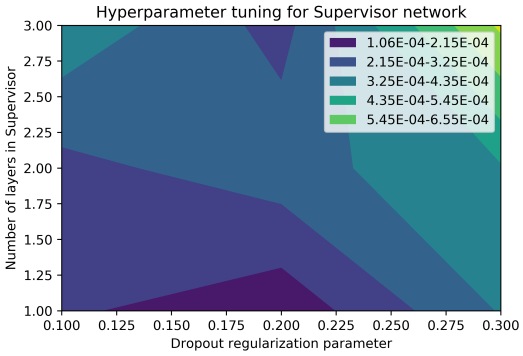

Figure 14. Supervisor loss for different number of layers and different dropout regularization parameters conditioned on optimal autoencoder.

7.2 Kupiec Test TimeGAN

We describe the model hyperparameters used in TimeGAN training in Appendix H. Table 4 shows the number of exceedances of T-VaR on the test EONIA data for the TimeGAN, Vasicek and the Variance-Covariance method.[34] We backtest the VaR estimates by comparing the number of exceedances to the allowed range for the Kupiec test as specified in Table 8. We observe that TimeGAN is able to estimate 20-day upper VaR, but unable to estimate 20-day lower VaR. TimeGAN is incapable of estimating 1-day and 10-day VaR for upper and lower direction.

Table 4. *Number of exceedances of 99 % - VaR in 250 EONIA trading days for TimeGAN during the period October 1st, 2018 until September 30th, 2019.*

Metric	Method	Number of exceedances					
		T (upper)			T (lower)		
		1	10	20	1	10	20
Coverage Test	TimeGAN	55	12	5	121	205	241
	Vasicek	3	1	2	1	0	0
	Variance-Covariance	4	6	5	3	7	5

We publish a visualization of the TimeGAN training in TimeGAN Tensorboard.

Table 5 shows the number of exceedances for T-VaR on the test EONIA data for the three possible

[33] We do not optimize hyperparameters of the Generator and Discriminator because this influences the GAN training.

[34] We notice that the standardized probability distribution of EONIA is leptokurtic. Therefore we decide to model the VaR using a t-distribution with the fitted degrees of freedom of 1.9 for Vasicek and the Variance-Covariance method.

improved TimeGAN models as specified in Section 4.4. We also include a TimeGAN model with Positive Label Smoothing and Feature Matching, referred to as TimeGAN with PLS+FM. We publish a visualization of respectively the TimeGAN with Feature Matching training, the TimeGAN with Positive Label Smoothing training, TimeGAN with PLS+FM, and the TimeGAN with Wasserstein-1 Gradient Penalty training in TimeGAN Feature Matching Tensorboard, TimeGAN Positive Label Smoothing Tensorboard, TimeGAN Feature Matching + Positive Label Smoothing Tensorboard and TimeGAN Wasserstein-1 Gradient Penalty Tensorboard.[35] We observe that TimeGAN with PLS+FM and TimeGAN with Wasserstein-1 Gradient Penalty estimates lower and upper 20-day VaR within acceptance range. We observe that the other models are able to estimate either upper or lower 20-VaR within range.

Table 5. *Number of exceedances of 99 % - VaR in 250 EONIA trading days for improved TimeGAN models during the period October 1st, 2018 until September 30th, 2019.*

Metric	Method	Number of exceedances					
		T (upper)			T (lower)		
		1	10	20	1	10	20
Coverage Test	TimeGAN	55	12	5	121	205	241
	with Wasserstein Gradient Penalty	55	27	9	164	3	1
	with Positive Label Smoothing	37	9	3	213	241	247
	with Feature Matching	55	126	241	61	11	3
	with PLS + FM	86	12	4	40	8	6

7.3 Diversity of TimeGAN simulations

Table 6 shows D_{NN} in the EONIA test data set for 250 simulations for all TimeGAN models. For comparison, D_{NN} in the Vasicek and the Variance-Covariance simulations are included. We observe that the diversity for all TimeGAN models is lower than the Vasicek and Variance-Covariance method. Figure 15 and Figure 16 show four NNs for the TimeGAN and TimeGAN with Wasserstein-1 Gradient Penalty models. We observe that TimeGAN is incapable of learning a complex structure while TimeGAN with Wasserstein-1 Gradient Penalty is capable of learning a more complex structure. Four nearest neighbours for every TimeGAN model are visualized in Appendix B.C and B.D.

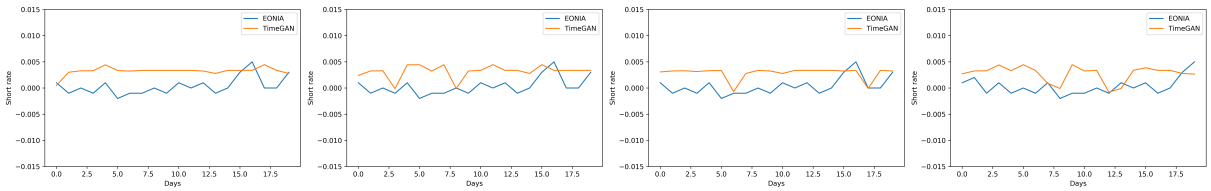

Figure 15. Four nearest neighbours in EONIA test data set for TimeGAN.

[35] We must note that the discriminator and generator loss in the Wasserstein-1 gradient penalty Tensorboard can not be compared to the other Tensorboards because a different loss metric, i.e. Wasserstein-1 loss, is implemented. For the Wasserstein-1 loss, the critic wants to give a high value for fake samples and a low value for real samples.

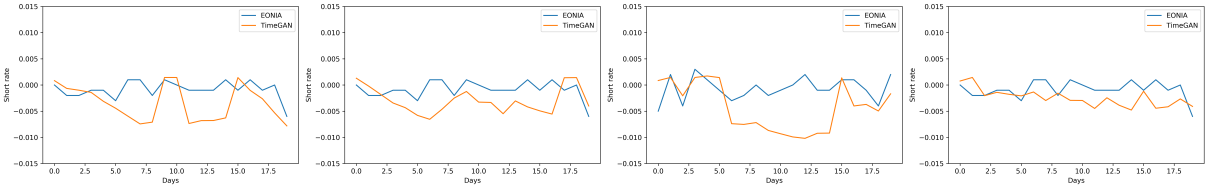

Figure 16. Four nearest neighbours in EONIA test data set for TimeGAN with Wasserstein-1 Gradient Penalty.

Table 6. *Number of diverse nearest neighbours in EONIA data for 250 simulations for TimeGAN and TimeGAN model improvements as proposed in Section 4.4.*

Metric	Method	D_{NN} for T		
		1	10	20
Diversity	TimeGAN	3	21	6
	TimeGAN with Wasserstein Gradient Penalty	3	19	19
	TimeGAN with Positive Label Smoothing	2	1	3
	TimeGAN with Feature Matching	7	14	4
	TimeGAN with FM + PLM	7	35	8
	Vasicek	44	121	145
	Variance-Covariance	37	53	58

7.4 t-SNE visualizations of TimeGAN simulations

We visualize the 20-day simulations of the TimeGAN model with Wasserstein-1 Gradient Penalty because this model specification provides acceptable VaR estimates and converges faster than TimeGAN with PLS+FM. Figure 17 shows the t-SNE projections of the simulations from the TimeGAN with Wasserstein-1 Gradient Penalty.[36]. We observe that the low-dimensional manifolds of the real and fake data partially overlap.[37]

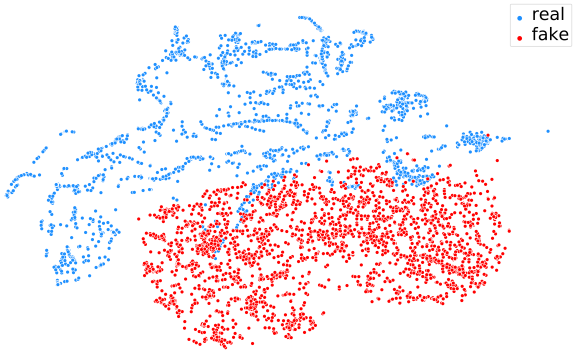

Figure 17. t-SNE for real EONIA data and fake EONIA data simulated by TimeGAN with Wasserstein Gradient Penalty after training 150 epochs.

[36] We implement t-SNE with perplexity of 40 and iterations of 300 as used in Maaten and Hinton (2008). We explain t-SNE in depth in Appendix D.A.

[37] Figure 36, Figure 37, Figure 38, and Figure 39 in Appendix B.E show respectively the t-SNE projections for the other TimeGAN models. We observe that these t-SNE visualizations show non-overlapping manifolds.

7.5 Classification of ECB's proposed EONIA-€STER mapping

Figure 18 shows the €STER during the pre-€STER period with corresponding realness score. We base the realness score of the ECB's proposed mapping on the classification of the Discriminator in TimeGAN with PLS+FM. We observe that some periods of the pre-ester period are strictly classified as real with a few exceptions in the middle of the period. We observe that none of the periods is classified as fake.

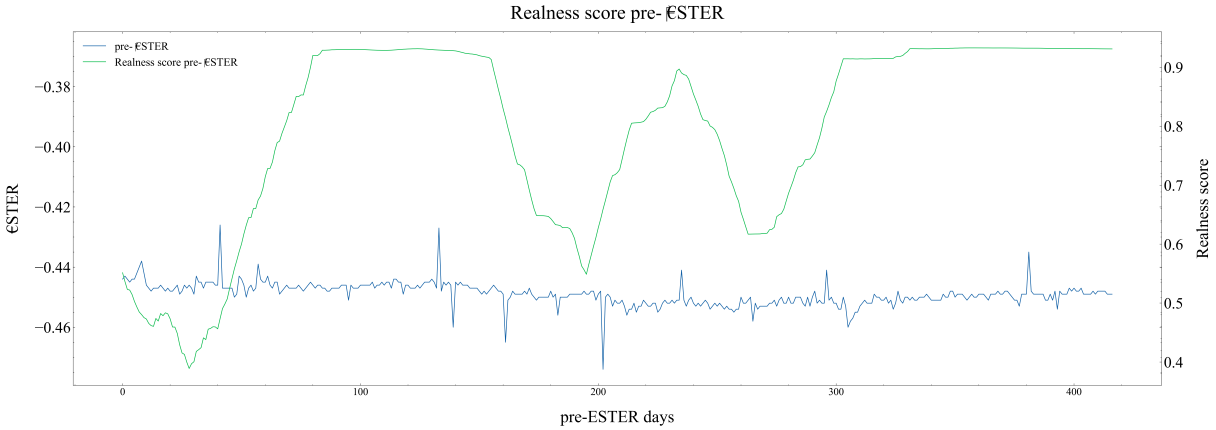

Figure 18. Realness score pre-€STER during pre-€STER based on TimeGAN with PLS+FM Discriminator.

7.6 Importance risk drivers and correlation matrix stylized facts

Figure 19 shows the correlation matrix between the stylized facts and the realness score provided by the Discriminator of the TimeGAN with PLS+FM. The abbreviations in Figure 19 corresponds to the abbreviations in Table 3. Figure 20 shows the importance of the independent variables in the realness prediction of €STER during the pre-€STER period. The importance of the independent variables is based on the XGboost regression decision tree as proposed by T. Chen and Guestrin (2016) where we treat the Discriminator as a black box.[38] We observe in Figure 19 that the variance and stationarity are strongly negatively correlated with the realness score and that spikiness is mildly negatively correlated with the realness score. Next to that, we observe in Figure 20 that longer term liquidity measures and inflation are of most importance to the realness score based on a static XGboost regression tree.

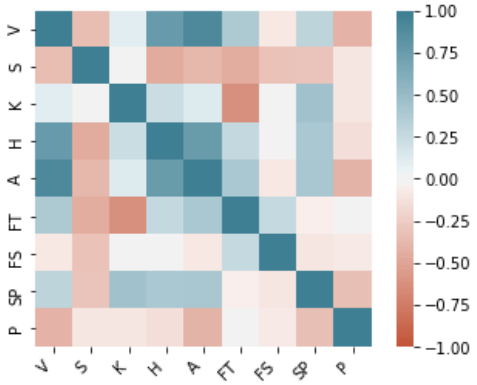

Figure 19. Correlation matrix of the stylized facts and the realness score.

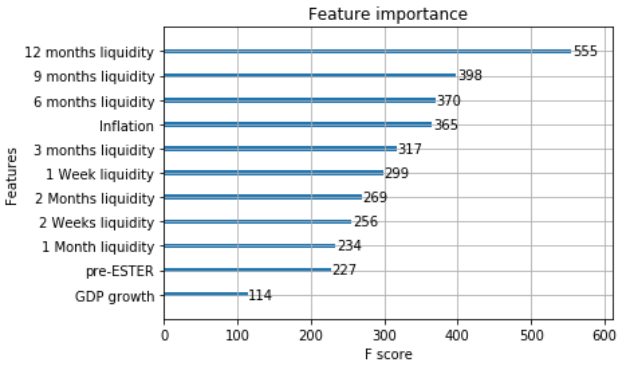

Figure 20. Independent variable importance in €STER realness score during pre-€STER period.

[38] We discuss the lack of interpretability of the black box in the discussion and propose further research increase clarity.

8 Conclusion

8.1 Conclusion

TimeGAN with Wasserstein-1 Gradient Penalty and TimeGAN with PLS+FM produce acceptable 20-day VaR estimates for EONIA based on the Kupiec backtest. The 20-day EONIA simulations are less diverse than a one-factor Vasicek or Variance-Covariance model. Secondly, based on a TimeGAN with PLS+FM Discriminator model, we can not conclude that the ECB's proposed mapping EONIA-€STER is wrong. The main drivers of the realness score are the variance and the stationarity of EONIA.

8.2 Discussion

We implement the loss hyperparameters shown in Table 7 in the TimeGAN training phase. Due to the computational complexity of the model, we were unable to structurally optimize the hyperparameters. We performed a limited unstructured manual search, but we could perform a grid search or random search over the hyperparameter space. Bergstra and Bengio (2012) visualize the difference between grid search and random search over a hyperparameter space in $\mathbb{R}^2$ in Figure 21. They conclude that a random grid search over a high dimensional hyperparameter space allows for better optimization. Despite the advantage of a structural approach to hyperparameter optimization, these methods do not reduce the computational complexity of the hyperparameter optimization.

Table 7. *TimeGAN loss hyperparameters.*

Parameter
γ for $\mathcal{L}_U^{WGAN-GP}$
η for $\mathcal{L}_S$ w.r.t. $\mathcal{L}_U$
λ for $\mathcal{L}_S$ w.r.t $\mathcal{L}_R$
κ for $\mathcal{L}_{FM}$ w.r.t $\mathcal{L}_R$ and $\mathcal{L}_U$

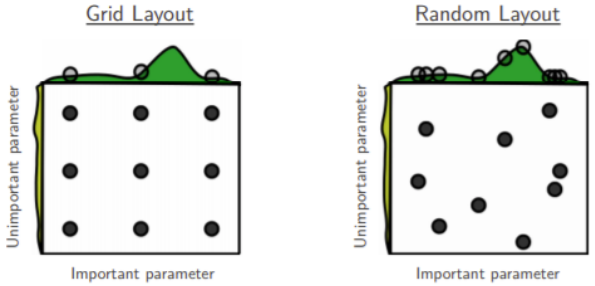

Figure 21. Difference grid search and random search.

Source: Bergstra and Bengio (2012)

A sequential hyperparameter optimization method that does reduce computational complexity is Bayesian optimization as proposed by Snoek et al. (2012). The optimization method obtains better performance of the model in fewer evaluations compared to grid search and random search. The improvement is due to the ability to reason about the performance of the model in the hyperparameter space before exploration of the space by computing the expected improvement. However, random search and grid search is embarrassingly parallel which results in a computation time reduction. We assume that the results in this study are generated by a TimeGAN model with a suboptimal loss hyperparameter setting due to the lack of a structural loss hyperparameter optimization method.

The second point of discussion concerns the t-SNE visualizations of the Generator simulations. Maaten and Hinton (2008) explain that the visualization of t-SNE is highly sensitive to the hyperparameters and that the algorithm is exposed to a certain level of stochasticity due to initialization. Besides that, it also

does not allow for a prediction step for test data. McInnes et al. (2018) propose the Uniform Manifold Approximation and Projection (UMAP) algorithm that tries to solve the aforementioned shortcomings. The UMAP algorithm produces a more stable dimension reduction of the high dimensional time series as generated by the Generator network in TimeGAN.

Next, we focus the discussion on the choice the training data set. We are interested in the VaR calculation which implicitly implies that it is beneficial to incorporate as much EONIA information as possible. Be that as it may, EONIA data characteristics change over time. An improvement to the model should be able to adapt to the dynamics. We expect a performance improvement if TimeGAN conditions on the current volatility or the previous trading period.

The last point of discussion focuses on the quality of the source code. Tensorflow 2.0 allows performance profiling of the source code on the LISA multi-GPU cluster. We were unable to profile the performance of TimeGAN because of access restrictions. Hence, we do not provide a guarantee that the model implementation in Tensorflow 2.0 is algorithmically efficient and attains the lowest running time possible.

8.3 Further research

We define four areas for further research. In the first place, we could extend TimeGAN with the attention model for increased performance. Secondly, we could increase the interpretability of the LSTM network. This is crucial for implementation of TimeGAN in VaR calculations due to regulations in finance. Thirdly, we could implement faster training techniques for GAN. This would allow for easier hyperparameter optimization due to decreased computational complexity. Lastly, we could condition on specific time periods in the training data. For instance the discrepancy between stressed versus non-stress training periods in terms of volatility. We briefly explore current research in these four directions and suggest further research.

Mnih et al. (2014) introduced the attention model in computer vision and Bahdanau et al. (2014) introduced the model in machine translation. The attention model implements an intermediary step where an attention map is created to pay attention to certain hidden states of the RNN. Given the sequence $x_{1:T}$, the RNN generates the hidden states $h_1, ..., h_T$. The attention model generates attention values α_i that are multiplied with the hidden states to form the context vector $c = \sum_i \alpha_i h_i$. The attention model is schematically depicted in Figure 22. The idea is that the context vector contains the information from every hidden state that is important for making a correct prediction. The attention model needs a separate RNN to interpret the attention map and make prediction y_1. One could also implement self-attention that implements attention within the layers of a RNN. This structure is schematically shown in Figure 23. Future research can investigate the performance of the attention model in combination with the GAN architecture. Research in this area is already active. For instance, Lai et al. (2018) propose the LSTNet model that combines a Convolutional Neural Network (CNN) with a LSTM with attention for multivariate time series forecasting. Shih et al. (2019) focus on applying attention especially to multivariate time series. Zhang et al. (2019) implements Self-Attention Generative Adversarial Network (SAGAN) which is a self-Attention model using a CNN for image generation in a GAN setting. Song

et al. (2017) implement self-attention with the LSTM model.

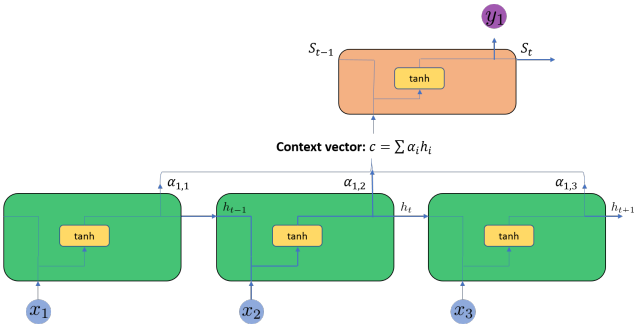

Figure 22. Schematic of the attention model with a vanilla RNN structure.

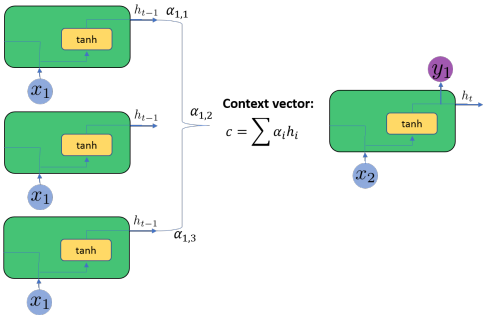

Figure 23. Schematic of the self-attention model with a vanilla RNN structure.

The assumption underlying the feature importance calculation is that the LSTM functions as a black box. We could inspect the LSTM in depth further for increased interpretability. Karpathy et al. (2015) increase interpretability of a LSTM trained on text data. They visualize the LSTM activations on text and conclude that around 10% of the memory state dimensions can be interpreted with the naked eye. We could implement and investigate what the latent variables actually represent. We tried to do this in the regression analysis but this does not say anything about the activations in the LSTM network. Moreover, if we were able to visualize the update, reset and forget gate, we would be able to see how long the dependencies are modelled in the LSTM. If the forget gate is very active this would say that a normal Vasicek or VAR(1) model also suffices. Common problems in GAN training are slow convergence and an unstable training process. We can leverage the power of the Adam optimizer but covariate shift could still be an issue when training the LSTM. Hence, hyperparameter optimization is difficult. We could leverage normalization techniques to speed up the convergence of LSTM models. Miyato et al. (2018) propose the Spectrally Normalized Generative Adversarial Network (SN-GAN) which implements spectral normalization as weight normalization to stabilize the weights in the discriminator network. Spectral normalization enforces the Lipschitz constant as a hyperparameter and hence alleviates the necessity of the Wasserstein-1 gradient penalty as implemented in Gulrajani et al. (2017). The main advantage is faster training because of faster convergence and a less computationally expensive algorithm due to the lack of the gradient penalty calculation. Ba et al. (2016) proposed layer normalization as a method that allows for normalization of the hidden states between two recurrent network cells. It basically implements the idea of batch normalization to recurrent neural networks. Cooijmans et al. (2016) allows for batch normalization in recurrent neural networks in his paper.

The model is now trained on the complete EONIA training dataset. The model could also be fitted only to stressed or non-stressed periods such that we can condition on this fact. (Arnelid et al., 2019) have tried to incorporate this notion of conditionality in autonomous driving sensor modelling. Fu et al. (2019) implement the conditionality on financial time series.

References

Amin, H. (2012). *Calibration of different interest rate models for a good fit of yield curves* (Master's thesis). Technische Universiteit Delft. the Netherlands.

Ang, A. & Piazzesi, M. (2003). A no-arbitrage vector autoregression of term structure dynamics with macroeconomic and latent variables. *Journal of Monetary Economics*, *50*(4), 745–787.

Arjovsky, M., Chintala, S. & Bottou, L. (2017). Wasserstein GAN. *arXiv preprint arXiv:1701.07875*.

Arnelid, H., Zec, E. L. & Mohammadiha, N. (2019). Recurrent Conditional Generative Adversarial Networks for Autonomous Driving Sensor Modelling, In *IEEE Intelligent Transportation Systems Conference*.

Ba, J. L., Kiros, J. R. & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bahdanau, D., Cho, K. & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473*.

Ballard, D. H. (1987). Modular Learning in Neural Networks, In *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*, Seattle, Washington, AAAI Press.

Beber, A., Brandt, M. W. & Kavajecz, K. A. (2009). Flight-to-Quality or Flight-to-Liquidity? Evidence from the Euro-Area Bond Market. *Review of Financial Studies*, *22*(3), 925–957.

Bengio, Y., Yao, L., Alain, G. & Vincent, P. (2013). Generalized Denoising Auto-Encoders as Generative Models, In *Advances in Neural Information Processing Systems*.

Bergstra, J. & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, *13*(1), 281–305.

Borji, A. (2019). Pros and cons of GAN evaluation measures. *Computer Vision and Image Understanding*, *179*, 41–65.

Bühler, W. & Trapp, M. (2009). Time-varying credit risk and liquidity premia in bond and CDS markets. University of Cologne, Center for Financial Research (CFR). No. 09-13.

Chen, L. (1996). Stochastic Mean and Stochastic Volatility: A Three-Factor Model of the Term Structure of Interest Rates and Its Applications in Derivatives Pricing and Risk Management. *Financial Markets, Institutions, and Instruments*, *5*, 1–88.

Chen, T. & Guestrin, C. (2016). Xgboost: A scalable tree boosting system, In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Cieslak, A. & Povala, P. (2015). Expected Returns in Treasury Bonds. *Review of Financial Studies*, *28*(10), 2859–2901.

Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç. & Courville, A. (2016). Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*.

Cooper, I. & Priestley, R. (2009). Time-Varying Risk Premiums and the Output Gap. *Review of Financial Studies*, *22*(7), 2801–2833.

Cox, J. C., Ingersoll Jr, J. E. & Ross, S. A. (1985). An Intertemporal General Equilibrium Model of Asset Prices. *Econometrica*, *53*(2), 363–384.

Dickey, D. A. & Fuller, W. A. (1979). Distribution of the Estimators for Autoregressive Time Series with a Unit Root. *Journal of the American Statistical Association*, *74*(366a), 427–431.

Duffee, G. R. & Stanton, R. H. (2012). Estimation of Dynamic Term Structure Models. *Quarterly Journal of Finance*, *2*, 1–51.

ESMA Annual Statistical Report - EU Derivatives Markets - 2019. (2019). European Securities and Markets Authority.

Fleming, M. J. & Remolona, E. M. (1999). Price Formation and Liquidity in the U.S. Treasury Market: The Response to Public Information. *Journal of Finance*, *54*(5), 1901–1915.

Fu, R., Chen, J., Zeng, S., Zhuang, Y. & Sudjianto, A. (2019). Time Series Simulation by Conditional Generative Adversarial Net. *arXiv preprint arXiv:1904.11419*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014). Generative Adversarial Nets, In *Advances in Neural Information Processing Systems*.

Goyenko, R. Y. & Ukhov, A. D. (2009). Stock and Bond Market Liquidity: A Long-Run Empirical Analysis. *Journal of Financial and Quantitative Analysis*, *44*(1), 189–212.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. & Courville, A. C. (2017). Improved Training of Wasserstein GANs, In *Advances in Neural Information Processing Systems*.

Hazan, T., Papandreou, G. & Tarlow, D. (2017). Adversarial Perturbations of Deep Neural Networks. In *Perturbations, Optimization, and Statistics* (pp. 311–342). MIT Press.

He, K., Zhang, X., Ren, S. & Sun, J. (2016). Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770–778).

Hinton, G. E. & Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, *313*(5786), 504–507.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780.

Hull, J. & White, A. (1990). Pricing Interest-Rate-Derivative Securities. *Review of Financial Studies*, *3*(4), 573–592.

Hurst, H. E. (1951). Long-term storage capacity of reservoirs. *Transactions of the American Society of Civil Engineers*, *116*, 770–799.

Hyndman, R. J., Wang, E. & Laptev, N. (2015). Large-Scale Unusual Time Series Detection, In *2015 IEEE International Conference on Data Mining Workshop (ICDMW). 1616-1619*.

Jarque, C. M. & Bera, A. K. (1980). Efficient tests for normality, homoscedasticity and serial independence of regression residuals. *Economics letters*, *6*(3), 255–259.

Joslin, S., Priebsch, M. & Singleton, K. J. (2014). Risk Premiums in Dynamic Term Structure Models with Unspanned Macro Risks. *Journal of Finance*, *69*(3), 1197–1233.

Karpathy, A., Johnson, J. & Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.

Kingma, D. P. & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.

Kupiec, P. (1995). Techniques for Verifying the Accuracy of Risk Measurement Models. *Journal of Derivatives*, *3*(2), 73–84.

Lai, G., Chang, W., Yang, Y. & Liu, H. (2018). Modeling Long-and Short-Term Temporal Patterns with Deep Neural Networks, In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*.

Levene, H. (1960). Contributions to probability and statistics. *Essays in honor of Harold Hotelling*, 278–292.

Longstaff, F. A. & Schwartz, E. S. (1992). Interest Rate Volatility and the Term Structure: A Two-Factor General Equilibrium Model. *Journal of Finance*, *47*(4), 1259–1282.

Maaten, L. v. d. & Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, *9*(Nov), 2579–2605.

Makhzani, A. & Frey, B. (2013). K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*.

McInnes, L., Healy, J. & Melville, J. (2018). UMAP: uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Miyato, T., Kataoka, T., Koyama, M. & Yoshida, Y. (2018). Spectral Normalization for Generative Adversarial Networks. *arXiv preprint arXiv:1802.05957*.

Mnih, V., Heess, N., Graves, A. Et al. (2014). Recurrent models of visual attention, In *Advances in Neural Information Processing Systems*.

Namin, S. S. & Namin, A. (2018). Forecasting Economic and Financial Time Series: ARIMA vs. LSTM. *arXiv preprint arXiv:1803.06386*.

Rifai, S., Vincent, P., Muller, X., Glorot, X. & Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction, In *Proceedings of the 28th International Conference on Machine Learning*.

Rudebusch, G. D. & Wu, T. (2008). A Macro-Finance Model of the Term Structure, Monetary Policy and the Economy. *Economic Journal*, *118*(530), 906–926.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. & Chen, X. (2016). Improved Techniques for Training GANs, In *Advances in Neural Information Processing Systems*.

Shih, S.-Y., Sun, F.-K. & Lee, H.-y. (2019). Temporal pattern attention for multivariate time series forecasting. *Machine Learning*, *108*(8-9), 1421–1441.

Snoek, J., Larochelle, H. & Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms, In *Advances in Neural Information Processing Systems*.

Song, H., Rajan, D., Thiagarajan, J. J. & Spanias, A. (2017). Attend and Diagnose: Clinical Time Series Analysis Using Attention Models. *arXiv preprint arXiv:1711.03905*.

Taylor, J. B. (1993). Discretion versus policy rules in practice. *Carnegie-Rochester Conference Series on Public Policy*, *39*, 195–214.

Vasicek, O. (1977). An equilibrium characterization of the term structure. *Journal of Financial Economics*, *5*(2), 177–188.

Vincent, P., Larochelle, H., Bengio, Y. & Manzagol, P.-A. (2008). Extracting and Composing Robust Features with Denoising Autoencoders, In *Proceedings of International Conference on Machine Learning*.

Wang, X., Smith, K. & Hyndman, R. (2006). Characteristic-Based Clustering for Time Series Data. *Data Mining and Knowledge Discovery*, *13*(3), 335–364.

Yoon, J., Jarrett, D. & van der Schaar, M. (2019). Time-series Generative Adversarial Networks, In *Advances in Neural Information Processing Systems*.

Zhang, H., Goodfellow, I., Metaxas, D. & Odena, A. (2019). Self-attention generative adversarial networks, In *International Conference on Machine Learning*.

Appendices

A Tables

Table 8. *Basel Committee backtesting from 1996b based on 99% Value-at-Risk and 250 days of data.*

zone	Exceedances	$\mathcal{P}(X < x)$
green	0	0.0811
	1	0.2858
	2	0.5432
	3	0.7581
	4	0.8922
yellow	5	0.9588
	6	0.9863
	7	0.9960
	8	0.9989
	9	0.9997
red	≥ 10	0.9999

B Figures

B.A Additional model selection results

Figure 24 and Figure 25 show the recovery loss for different hyperparameter settings conditioned on respectively dropout regularization of 0.2 and 0.3. We see that the recovery loss is higher in the subset of the hyperparameter space that is conditioned on a dropout regularization parameter of 0.2 or 0.3 than

the lowest value in Figure 11.

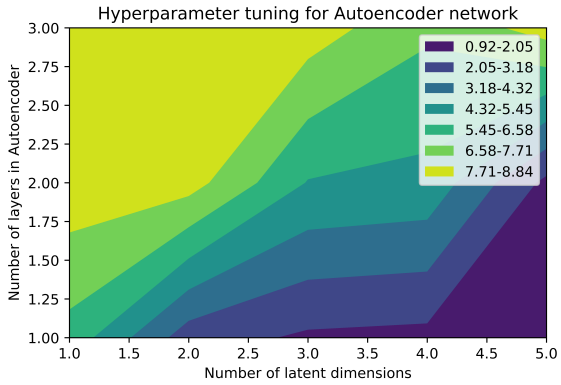

Figure 24. Recovery loss for different number of layers and different numbers of latent dimensions in autoencoder conditioned on dropout of 0.2.

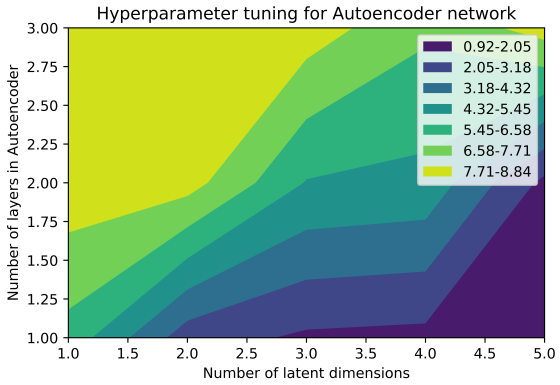

Figure 25. Recovery loss for different number of layers and different numbers of latent dimensions in autoencoder conditioned on dropout of 0.3.

B.B Latent space representation per latent space dimensionality

Figure 26 and Figure 27 show the latent space representation of an autoencoder with latent space dimensionality of respectively 4 and 5 during the 20-day trading period from October 6$^{\text{th}}$, 2017 until November 2$^{\text{nd}}$, 2017 including EONIA for comparison.

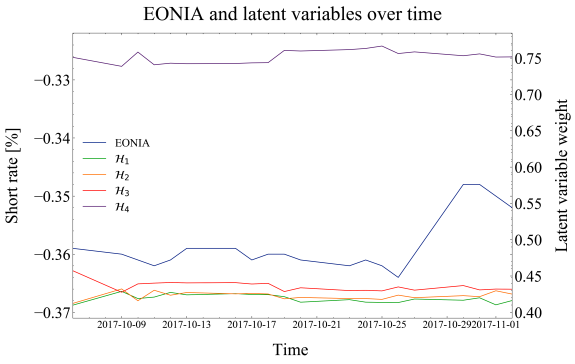

Figure 26. Latent space representation learned by autoencoder with latent space dimensionality 4.

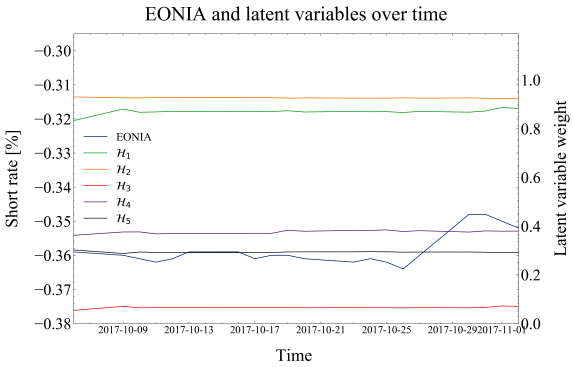

Figure 27. Latent space representation learned by autoencoder with latent space dimensionality 5.

B.C 20-day EONIA simulations for TimeGAN

Figure 28, Figure 29, and Figure 30 show five 20-day EONIA simulations produced by TimeGAN and their nearest neighbours after respectively 150, 900, and 1500 training iterations.

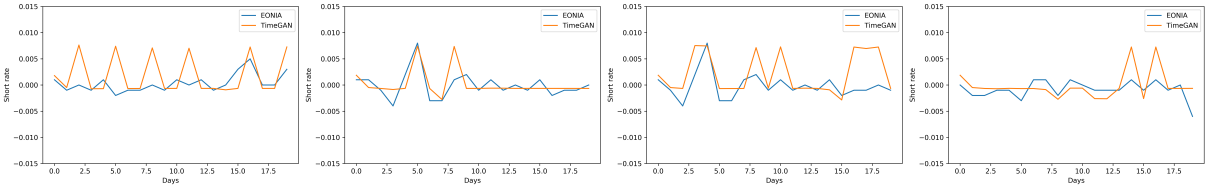

Figure 28. TimeGAN simulations of 20-day EONIA after 150 training epochs.

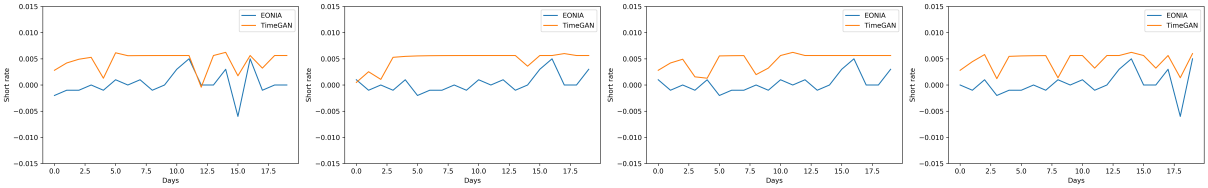

Figure 29. TimeGAN simulations of 20-day EONIA after 900 training epochs.

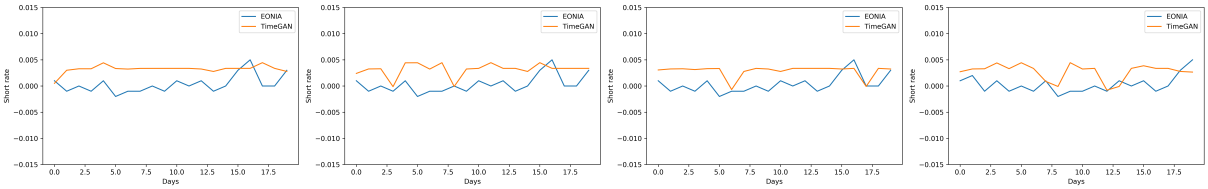

Figure 30. TimeGAN simulations of 20-day EONIA after 1000 training epochs.

B.D 20-day EONIA simulations for improved TimeGAN models

Figure 31, Figure 32, Figure 33, Figure 34 and Figure 35 show five 20-day EONIA simulations for respectively TimeGAN, TimeGAN with Positive Label Smoothing, TimeGAN with Feature Matching, TimeGAN with Feature Matching and Positive Label Smoothing, and TimeGAN with Wasserstein-1 Gradient Penalty and their nearest neighbours after 8,550 training iterations on the EONIA training data set.

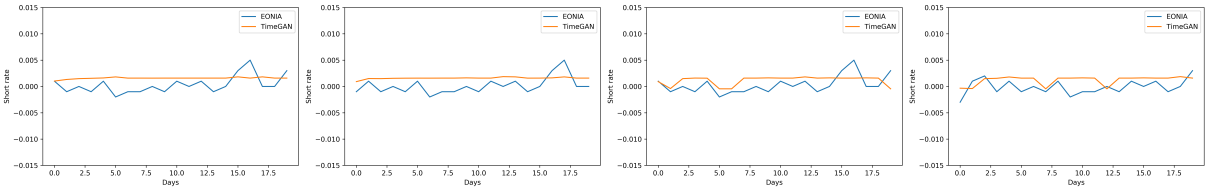

Figure 31. TimeGAN 20-day EONIA simulations.

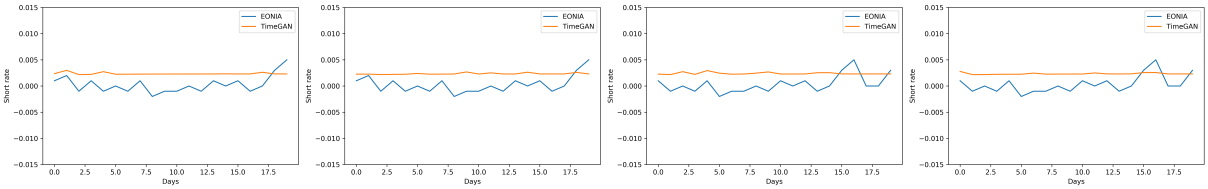

Figure 32. TimeGAN with Positive Label Smoothing 20-day EONIA simulations.

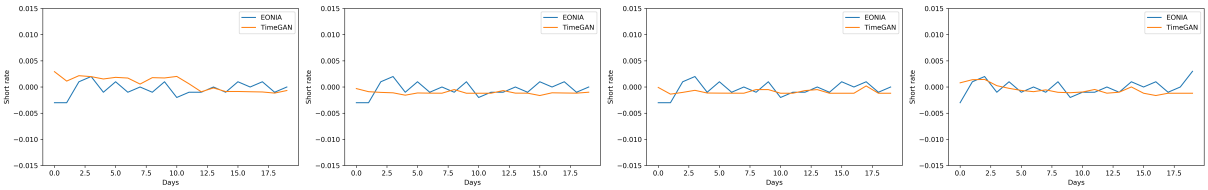

Figure 33. TimeGAN with Feature Matching 20-day EONIA simulations.

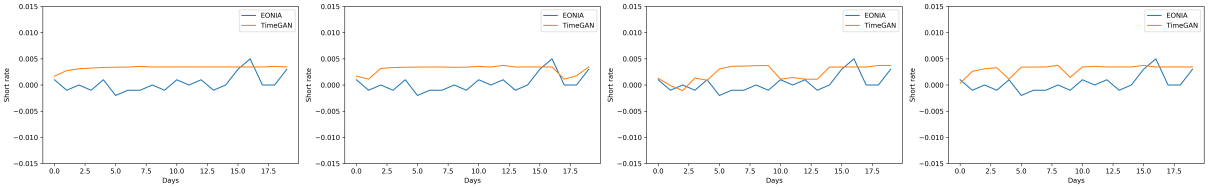

Figure 34. TimeGAN with Feature Matching and Positive Label Smoothing 20-day EONIA simulations.

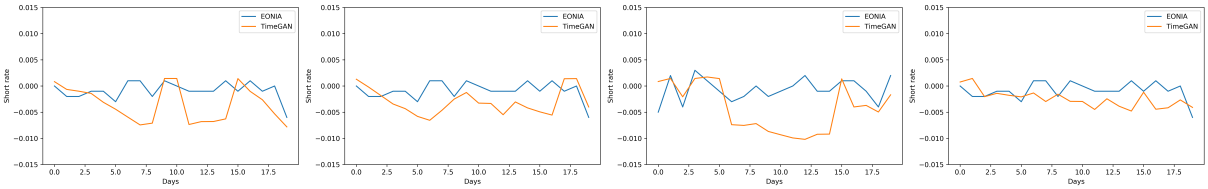

Figure 35. TimeGAN with Wasserstein-1 Gradient Penalty 20-day EONIA simulations.

B.E t-SNE visualizations for TimeGAN with FM, PLS and WGAN-GP

Figure 36, Figure 37, Figure 38, and Figure 39 show the t-SNE for respectively TimeGAN, TimeGAN with Feature Matching, TimeGAN with Positive Label Smoothing, and TimeGAN with Positive Label Smoothing and Feature Matching.

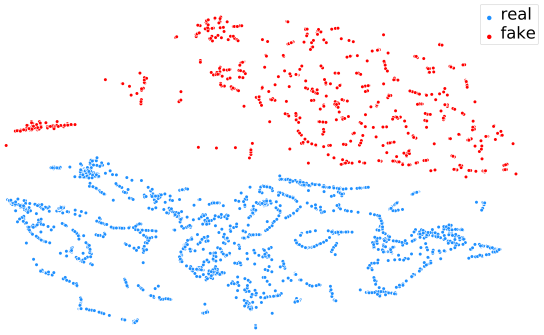

Figure 36. t-SNE for TimeGAN after 8550 training epochs.

Figure 37. t-SNE for TimeGAN with Feature Matching after 9150 training epochs.

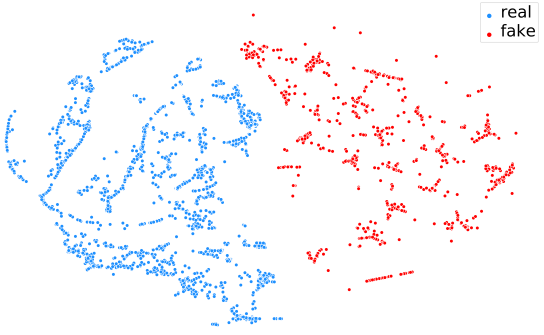

Figure 38. t-SNE for TimeGAN with Positive Label Smoothing after 9600 training epochs.

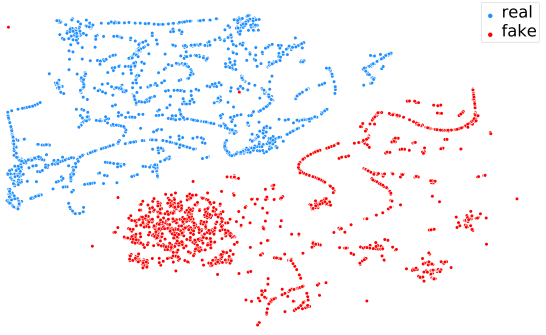

Figure 39. t-SNE for TimeGAN with Positive Label Smoothing and Feature Matching after 9150 epochs.

C Source code

Source code is hosted on github.com/Larsterbraak and is currently only available for university super-

visors. If you want access, feel free to e-mail **larsterbraak@gmail.com**.

D Derivations

This section shows derivations which are non-essential for the study but are used in the study. We start with the t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm that is used for the visualization of the multivariate time series as simulated by the Generator in TimeGAN. We conclude a discussion about Value at Risk (VaR) metric which we use as input in the Kupiec test.

D.A t-distributed Stochastic Neighbor Embedding

Maaten and Hinton (2008) propose the t-distributed Stochastic Neighbor Embedding (t-SNE) which is a neighbour graph based nonlinear dimensional reduction technique. The algorithm uses the local structure of the data instead of the global covariance matrix as is the case with a linear dimension reduction technique, e.g. Principal Component Analysis (PCA).

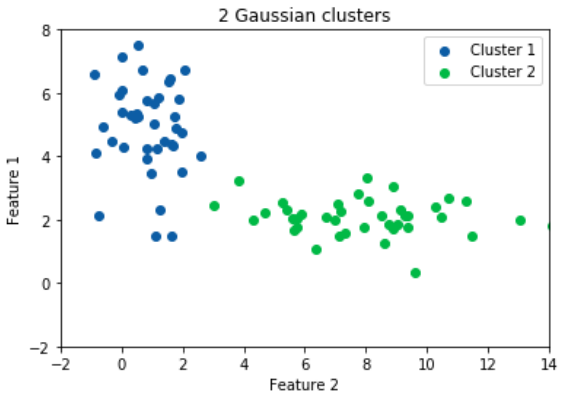

Figure 40. Two Gaussian clusters in $\mathbb{R}^2$.

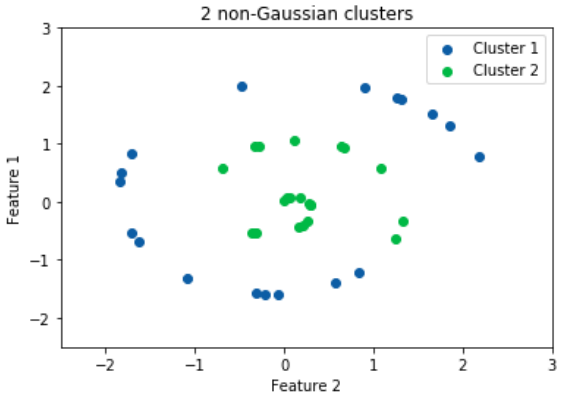

Figure 41. Two non-Gaussian clusters in $\mathbb{R}^2$.

PCA uses matrix factorization and would for instance perform well on the 2 Gaussian clusters illustrated in Figure 40. Nevertheless, Figure 41 shows 2 non-Gaussian clusters where the t-SNE generates a better dimension reduction. We assume that the multivariate time series used in this study is non-linear of nature and therefore we visualize using t-SNE algorithm. Let us define $x_{1:T}^{(1)} \cdots x_{1:T}^{(N)}$ to be high dimensional data points as used in this study. First, t-SNE constructs a probability distribution over $x_{1:T}^{(i)}$ and $x_{1:T}^{(j)}$ such that similar data points are assigned a higher probability while dissimilar points are assigned a lower probability. To assess the similarity of data points, we define the conditional probability $p_{j|i}$ that $x_{1:T}^{(i)}$ and $x_{1:T}^{(j)}$ are similar,

$$p_{j|i} = \frac{\exp(-||x_{1:T}^{(i)} - x_{1:T}^{(j)}||^2 \backslash 2\sigma_i^2)}{\sum_{k \neq i} \exp(\ ||x_{1:T}^{(i)} - x_{1:T}^{(k)}||^2 \backslash 2\sigma_i^2)}, \tag{30}$$

and set $p_{i|i} = 0$. The probability is proportional to the chance that $x_{1:T}^{(i)}$ would pick $x_{1:T}^{(j)}$ as it neighbours assuming that the neighbourhood of $x_{1:T}^{(i)}$ is a Gaussian with mean $x_{1:T}^{(i)}$ and variance σ_i. Now we define the probability p_{ij},

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}. \tag{31}$$

and note that $p_{ij} = p_{ji}$, $p_{ii} = 0$, and $\sum_{i,j} p_{ij} = 1$. Users specify the perplexity hyperparameter, which translates to the measure of effective neighbours for every $x_{1:T}^{(i)}$. A binary-search is performed which produces the bandwidth of the Gaussian kernel, i.e. σ_i, such that the predefined perplexity is satisfied. The σ_i parameter is therefore unique for every data point. Second, t-SNE initializes the data points in the low-dimensional space randomly. We define a probability distribution over points in the low-dimensional map. The algorithm learns the 2-dimensional map $y_{1:T}^{(1)} \cdots y_{1:T}^{(N)}$ with $y_{1:T}^{(i)} \in \mathbb{R}^2$ that reflects the similarities p_{ij} as well as possible. Let us define the probability q_{ij} between two points in the map $y_{1:T}^{(i)}$ and $y_{1:T}^{(j)}$,

$$q_{ij} = \frac{(1 + ||y_{1:T}^{(i)} - y_{1:T}^{(j)}||^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + ||y_{1:T}^{(k)} - y_{1:T}^{(l)}||^2)^{-1}}. \tag{32}$$

and set $q_{ii} = 0$. We use the the Cauchy distribution with $\gamma = 1$ with corresponding probability density function,

$$f(y_{1:T}^{(i)}; y_{1:T}^{(j)}, \gamma) = \frac{1}{\pi\gamma} \left[\frac{\gamma^2}{||y_{1:T}^{(i)} - y_{1:T}^{(j)}||^2 + \gamma^2} \right], \tag{33}$$

to measure the similarity between the low-dimensional points $y_{1:T}^{(1)} \cdots y_{1:T}^{(N)}$. The Cauchy distribution ensures that dissimilar objects are embedded far apart in the low-dimensional map and therefore avoid the so-called crowding problem. The locations of the points in the map are determined by minimizing the Kullback-Leibler Divergence (KLD) of the distribution P, i.e. the probability distribution of the high-dimensional space, from the distribution Q, i.e. the probability distribution of the low dimensional space. The KLD is defined as

$$KL(P||Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{34}$$

The minimization of the KLD with respect to the points y_i is performed using gradient descent with momentum. t-SNE is known to perform suboptimal on high dimensional data and therefore the first step includes the use of PCA or Autoencoder network to reduce the dimensionality to a level that t-SNE can handle. We use the fully trained Autoencoder network to reduce the dimensionality to $[Tx\mathcal{H}]$ whereby T represents the length of the time series and $\mathcal{H}$ represents the number of latent variables. The $[Tx\mathcal{H}]$-dimensional vector is then reduced to $\mathbb{R}^2$ using t-SNE.

D.B Value at Risk

Value at Risk (VaR) is a measure of the risk of loss for investments. Let X be a profit and loss distribution. The VaR at level $\alpha \in (0, 1)$ is the smallest number y such that the probability that $Y := -X$ does not exceed y is at least $1 - \alpha$. The VaR is defined as

$$\text{VaR}(X) = -\inf x \in \mathbb{R} : F_X(x) > \alpha = F_y^{-1}(1 - \alpha). \tag{35}$$

E AutoEncoder

The autoencoder model is a neural network with the objective to compress the data $x_{1:T}$ to a latent space $h_{1:T}$ and subsequently produce a recovery $\tilde{x}_{1:T}$. The autoencoder model is a nonlinear dimension reduction technique. The model consists of an encoder function f that produces latent variables $h_{1:T} = f(x_{1:T})$

and hence provides the mapping from data space to the latent space. The decoder function g produces the reconstruction $\tilde{x}_{1:T} = g(h_{1:T}) = (g \circ f)(x_{1:T})$ and hence provides the mapping from latent space to data space. The loss function is defined as the loss between the original data $x_{1:T}$ and the reconstruction $\tilde{x}_{1:T}$,

$$\mathcal{L}(x_{1:T}, \tilde{x}_{1:T}) = \mathcal{L}(x_{1:T}, (g \circ f)(x_{1:T})) = \mathbb{E}\,||x_{1:T}, \tilde{x}_{1:T}||_2, \tag{36}$$

where $\mathcal{L}$ denotes the mean-squared error loss function. We use an undercomplete autoencoder which means that the latent space dimensionality is smaller than the data dimensionality. Learning a undercomplete representation forces the model to capture the most salient features of the data $x_{1:T}$. The network architecture is schematically visualized in Figure 42.

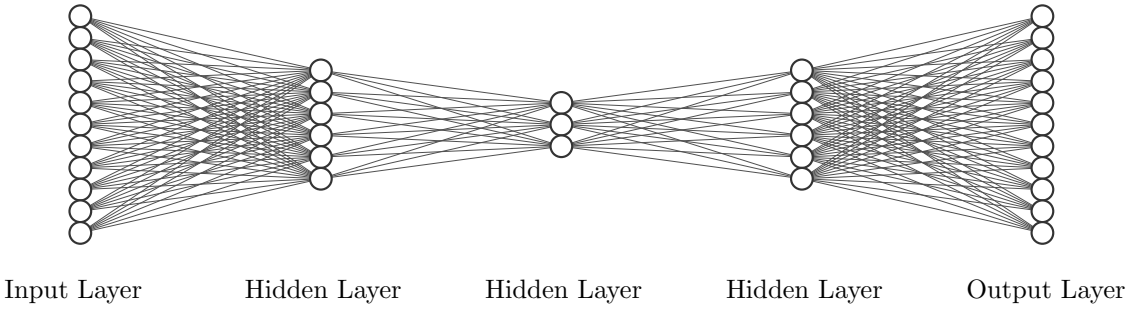

Figure 42. Autoencoder network with 3 layers with respectively 6, 3, and 6 neurons per layer. Both the input and output layer consist of 11 neurons which coincides with the data dimensionality.

An autoencoder model with a linear decoder corresponds to the Principal Component Analysis (PCA). We refer to an autoencoder with more than one hidden layer as a deep autoencoder. G. E. Hinton and Salakhutdinov (2006) show experimentally that deep Autoencoders yield much better compression than corresponding shallow or linear autoencoders. Nevertheless, if the encoder and decoder are very deep networks the autoencoder can learn to copy the input data $x_{1:T}$ without extracting useful information about the distribution of the data. Rather than limiting the model capacity by keeping the encoder and decoder shallow and the code size small, regularized autoencoders use a loss function that encourages the model to have other properties besides the ability to copy its input to its output. We first discuss the class of regularized autoencoder and in specific Sparse Autoencoder (SAE), the Denoising Autoencoder (DAE), and the Contractive Autoencoder (CAE). Next, we discuss generative autoencoders and in specific the Variational Autoencoder. Next, we discuss the applicability for short rate time series and the eventual model implemented in this research.

Makhzani and Frey (2013) propose the SAE in which the loss function involves a sparsity penalty $\Omega(h_{1:T})$ on the code layer $h_{1:T}$, in addition to the reconstruction error,

$$\mathcal{L}(x_{1:T}, (g \circ f)(x_{1:T})) + \Omega(h_{1:T}). \tag{37}$$

Vincent et al. (2008) propose the DAE that focuses on the loss function directly to improve the representation learned rather than adding a sparsity penalty Ω. The DAE takes a partially corrupted version of the input data, $\tilde{x}_{1:T}$, and train the model to construct the original data $x_{1:T}$. In this way,

the model is denoising the input. Different denoising techniques for x are available but we implement the dropout technique as proposed by G. E. Hinton et al. (2012) in this study. The loss function now becomes

$$\mathcal{L}(x_{1:T}, (g \circ f)(\tilde{x}_{1:T})) \tag{38}$$

Rifai et al. (2011) propose the CAE that defines the regularization by penalizing derivatives with respect to the data $x_{1:T}$,

$$\Omega(h_{1:T}) = \lambda \, \mathbb{E} \, ||\nabla_{x_{1:T}} h_{1:T}||^2. \tag{39}$$

This forces the model to learn a function that does not change much when $x_{1:T}$ changes slightly.

Apart from regularized autoencoder, there are also generative Autoencoders such as the Variational Autoencoder. The loss function then becomes

$$\mathcal{L}(\theta, \phi, x, z) = \mathbb{E}_{q_\phi(z|x)} \left[\log p_\theta(x|z)\right] - D_{KL}(q_\theta(z|x)||p(z)), \tag{40}$$

where the first term is the reconstruction loss and the second term the Kullback-Leibler Divergence (KLD) between the distribution and a prior distribution $p(z)$. This is done such that the latent space covers a range and does not collapse to the identity matrix. It is common practice to use the multidimensional Gaussian for this.

Makhzani and Frey (2013) observe that when representations are learnt in the SAE model, improved performance is obtained on classification tasks. Bengio et al. (2013) show that the DAE model forces f and g to implicitly learn the structure of $p_{data}(x)$. Hence, we implement the DAE model with dropout regularization. We explicitly reject the Variational Autoencoder due to its generative nature. We would be unable to distinguish where the generative power comes from in the TimeGAN model. Furthermore, we chose to implement a deterministic version of the autoencoder in order to stay as close to a non-linear version of PCA as possible.

F Recurrent neural networks

Here we explain the Recurrent Neural Network (RNN). Let us start with a regular neural network which is defined by the function f_w where w represents the parametrization in the network. We try to approximate the function f_w that links the explanatory variables, x_t, and the dependent variables, y_t. In the forward pass, we make a prediction $\hat{y}_t = f_w(x_t)$ and calculate the loss $\mathcal{L}(\hat{y}_t, y_t)$. We compute the gradients of the loss with respect to the weights in the network. In the backward pass we adjust the weights based on the gradients. A RNN has, apart from the explanatory variable x_t an additional hidden state, h_t, that also serves as input for f_w. The hidden state allows for a recurrence relation between consecutive RNN cells. The forward pass in a RNN now includes the recurrence relation and the RNN cell also outputs the current hidden state h_t,

$$\hat{y}_t, h_t = f_w(h_{t-1}, x_t). \tag{41}$$

A schematic visualization of the RNN and the updating equations are shown in Figure 43. In the first step, we compute the current cell state based on the previous cell state and the current explanatory variables. In the second step, we compute the prediction based on the current cell state.

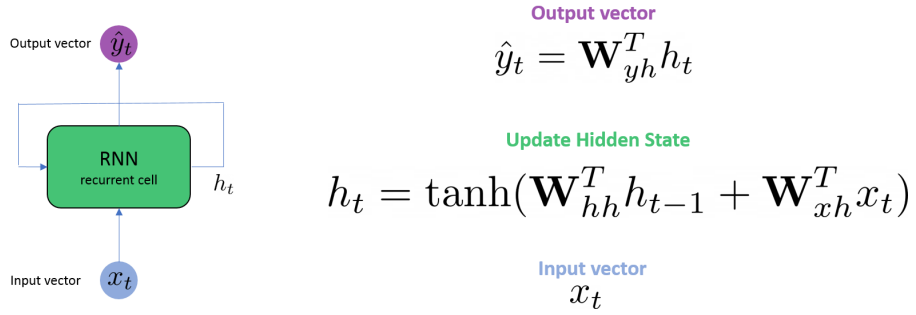

Figure 43. Overview of a Recurrent Neural Network architecture including updating equations.

Figure 44 shows a RNN unrolled over time. The forward pass, the loss computation, and the backpropagation are visualized. The backpropagation over time leads to a vanishing or exploding gradient problem due to $\lim_{n\to\infty} W^n = 0$ or $\lim_{n\to\infty} W^n = \infty$. Hochreiter and Schmidhuber (1997) propose a different network architecture that includes gated cells that control the information flow in the RNN cell in order to overcome this problem.

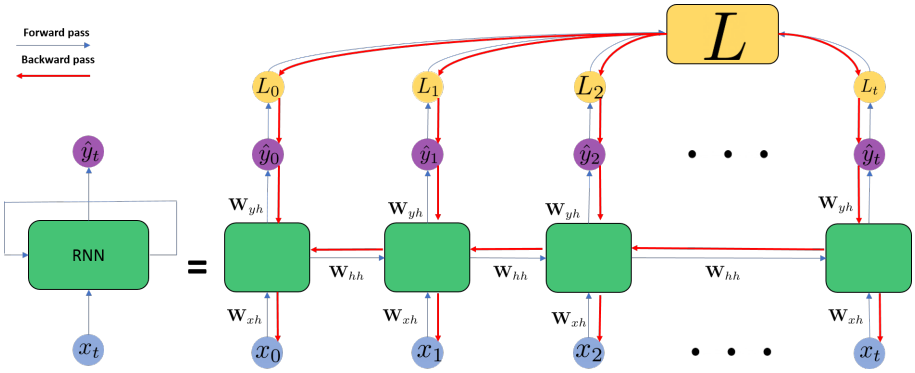

Figure 44. RNN computational graph across time and resulting backpropagation through time step.

The RNN and LSTM network architecture are respectively schematically visualized in Figure 45 and 46. Hochreiter and Schmidhuber (1997) include a long term cell state, c_t, that is updated using gated cells. We now deep dive in the LSTM network architecture and how it overcomes the vanishing gradient problem.

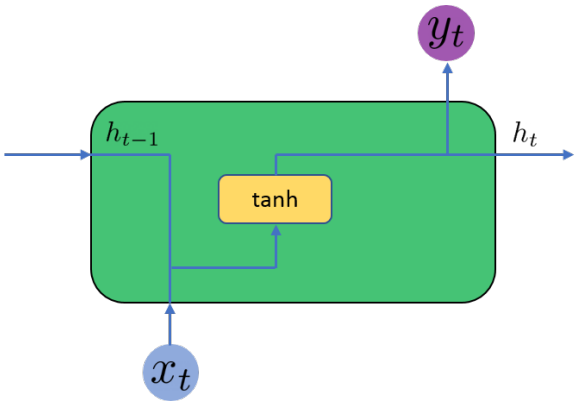

Figure 45. Simple computation node in the repeating modules of the standard RNN.

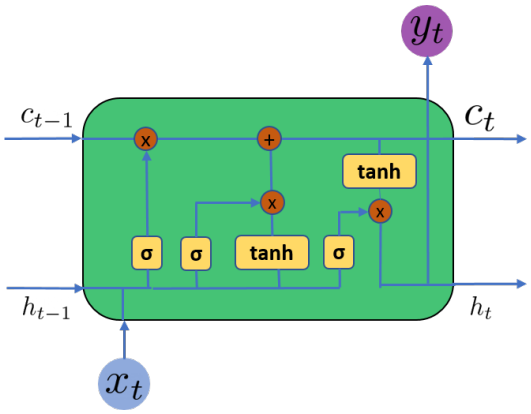

Figure 46. LSTM computation blocks that control the flow of information in the RNN cell.

The different steps in the LSTM network architecture are indicated by the blocks as visualized in Figure 47. The steps consists of 1) Forget, 2) Store, 3) Update, and 4) Output. We discuss each in detail.

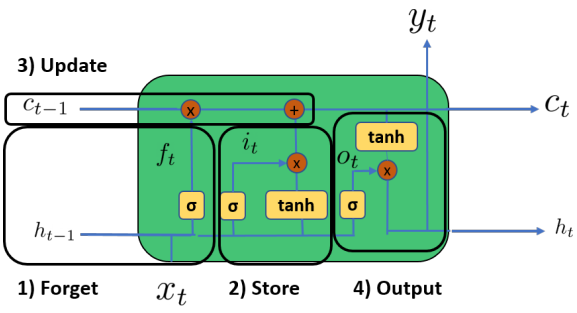

Figure 47. Inner working of the LSTM recurrent neural network cell.

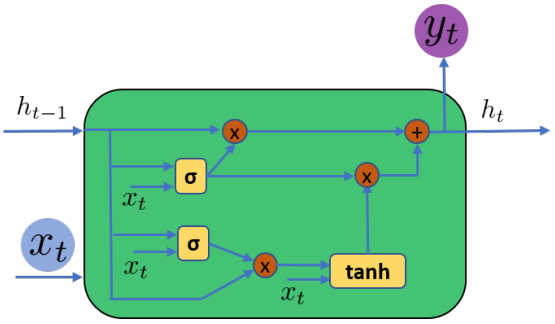

Figure 48. Inner working of the GRU recurrent neural network cell.

Firstly, the irrelevant information is forgotten based on the current explanatory variables and the previous cell state,

$$\textbf{Step 1:} \quad f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{42}$$

where σ denotes the sigmoid activation function and W_f and U_f are weight matrices. Secondly, the relevant information is stored based on the current explanatory variables and the previous cell state,

$$\textbf{Step 2:} \quad i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \tag{43}$$

where W_i and U_i are weight matrices. Thirdly, the proposed cell state, $\tilde{c}_t$ is computed based on the current explanatory variables and the previous hidden state,

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c),$$

where tanh denotes the hyperbolic tangent activation function. The new cell state, c_t, is computed based on the previous and proposed cell state, the forget gate and the store gate,

$$\textbf{Step 3:} \quad c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t, \tag{44}$$

where $\circ$ denotes the Hadamard product. Eventually, the output gate is computed based on the current hidden state and the explanatory variables,

$$\textbf{Step 4:} \quad o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \tag{45}$$

and we compute the current hidden state based on the current cell state and the output gate,

$$h_t = o_t \circ \tanh(c_t),$$

where the output y_t is computed in accordance to a regular RNN cell.

During the years, multiple improvements to the recurrent neural networks have emerged. Cho et al. (2014) propose the Gated Recurrent Unit (GRU) that is schematically visualized in Figure 48. The GRU model does not use the cell state c_t and hence is less heavy parametrized than the LSTM. The forget gate is based on the complement to 1 of the update gate. Cho et al. (2014) indicate that the performance is similar to LSTM. We discuss the updating equations of the GRU here.
We compute the read gate, r_t, based on the current explanatory variables and the previous hidden state,

$$\textbf{Step 1:} \quad r_t = \sigma(W_r x_t + U_r h_{t-1} + b_z), \tag{46}$$

where σ denotes the sigmoid activation function. Next we compute the forget gate, z_t, based on the current explanatory variables and the previous hidden state,

$$\textbf{Step 2:} \quad z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z). \tag{47}$$

Next we compute the proposed hidden state, $\hat{h}_t$, based on the current explanatory variables, the read gate and the previous cell state,

$$\hat{h}_t = \tanh(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h) \tag{48}$$

where $\circ$ denotes the Hadamard product. We compute the current hidden state, h_t, based on the update gate, the previous hidden state and the proposed hidden state,

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \hat{h}_t. \tag{49}$$

A RNN has the restriction that a hidden state can only pass on information to a successive recurrent neural network cell. This is logical in the case of time series because at time t we do not know the information at time $t + 1$. A bidirectional recurrent neural network does not have this restriction and the hidden state can propagate back in time as visualized in Figure 49. The discriminator in the GAN network does not have the restriction that it can not look at data in the future. Hence, we implement a bidirectional LSTM for the discriminator model.

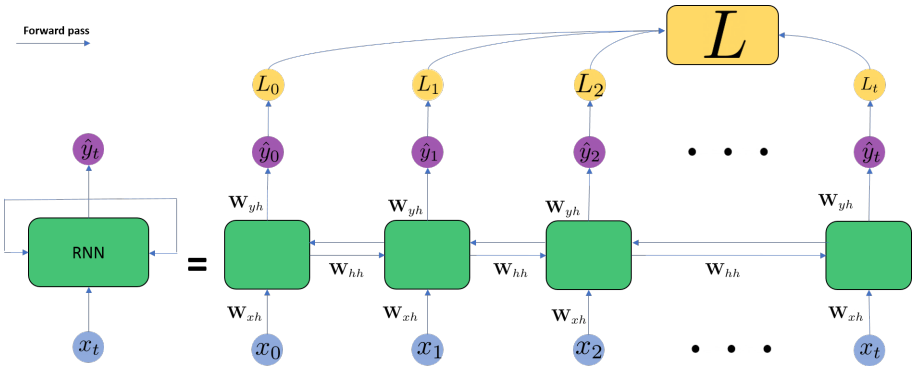

Figure 49. Forward pass in a bidirectional recurrent neural network cell. The hidden state h can flow pro- and contra-temporally through the network.

G Algorithms

Algorithm 1 Pre-training TimeGAN for short rates

Input: Adam parameters β_1 & β_2, minibatch size m, loss parameters η & λ, and total epochs N

Initialize: $\theta_e, \theta_r, \theta_s$ using He et al. (2016)

(1) Pre-training autoencoder

for N **do**

  for number of minibatches **do**

    Sample minibatch of m samples $\mathbf{x}_{1:T}^{(1)} \cdots \mathbf{x}_{1:T}^{(m)}$ from real data distribution $p_{\mathcal{X}}(\mathbf{x})$

    Provide embedding $\mathbf{h}_{1:T}^{(1)} \cdots \mathbf{h}_{1:T}^{(m)}$ by applying embedder E_{θ_e} on $\mathbf{x}_{1:T}^{(1)} \cdots \mathbf{x}_{1:T}^{(m)}$.

    Provide recovery $\tilde{\mathbf{x}}_{1:T}^{(1)} \cdots \tilde{\mathbf{x}}_{1:T}^{(m)}$ by applying recovery R_{θ_r} on $\mathbf{h}_{1:T}^{(1)} \cdots \mathbf{h}_{1:T}^{(m)}$.

    Update θ_e by descending gradient of $\mathcal{L}_R$ w.r.t. θ_e using Adam:

$$\nabla_{\theta_e} \frac{1}{m} \sum_{i=1}^{m} \sum_{t} ||x_t - \tilde{x}_t||_2$$

    Update θ_r by descending gradient of $\mathcal{L}_R$ w.r.t. θ_r using Adam:

$$\nabla_{\theta_r} \frac{1}{m} \sum_{i=1}^{m} \sum_{t} ||x_t - \tilde{x}_t||_2$$

  end for

end for

(2) Pre-training supervisor

for N **do**

  for number of minibatches **do**

    Sample minibatch of m samples $\mathbf{x}_{1:T}^{(1)} \cdots \mathbf{x}_{1:T}^{(m)}$ from real data distribution $p_{\mathcal{X}}(\mathbf{x})$

    Provide embedding $\mathbf{h}_{1:T}^{(1)} \cdots \mathbf{h}_{1:T}^{(m)}$ by applying embedder E_{θ_e} on $\mathbf{x}_{1:T}^{(1)} \cdots \mathbf{x}_{1:T}^{(m)}$.

    Predict embeddings $\tilde{\mathbf{h}}_{1:T}^{(1)} \cdots \tilde{\mathbf{h}}_{1:T}^{(m)}$ by applying supervisor S_{θ_s} on $\mathbf{h}_{1:T}^{(1)} \cdots \mathbf{h}_{1:T}^{(m)}$.

    Update θ_s by descending gradient of $\mathcal{L}_S$ w.r.t. θ_s using Adam:

$$\nabla_{\theta_s} \frac{1}{m} \sum_{i=1}^{m} \sum_{t} ||h_t - \tilde{h}_t||_2$$

  end for

end for

Algorithm 2 TimeGAN for short rates

Input: Adam parameters β_1 & β_2, minibatch size m, loss parameters η & λ, total epochs N, pre-trained model parameters θ_e, θ_r & θ_s, sequence length T and discriminator iterations k

Initialize: θ_g, θ_d using He et al. (2016)

(3) TimeGAN training

for N **do**

  for number of minibatches **do**

(3.1) Generator training

    Sample minibatch of m fake data samples $\hat{\mathbf{h}}_{1:T}^{(1)} \cdots \hat{\mathbf{h}}_{1:T}^{(m)}$ from generator G_{θ_g}

    Predict embeddings $\tilde{\mathbf{h}}_{1:T}^{(1)} \cdots \tilde{\mathbf{h}}_{1:T}^{(m)}$ by applying supervisor S_{θ_s} on $\hat{\mathbf{h}}_{1:T}^{(1)} \cdots \hat{\mathbf{h}}_{1:T}^{(m)}$

    Predict $\hat{y}_{1:T}$ by applying discriminator D_{θ_g} on $\hat{\mathbf{h}}_{1:T}^{(1)} \cdots \hat{\mathbf{h}}_{1:T}^{(m)}$

    Update θ_g by ascending gradient of $-\eta\mathcal{L}_S - \mathcal{L}_U$ w.r.t. θ_g using Adam:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} -\eta \left(\sum_{t=1}^{T} ||\hat{h}_t^{(i)} - \tilde{h}_t^{(i)}||_2 \right) + \log(\hat{y}_{1:T})$$

(3.2) Embedder training

    Sample minibatch of m real data samples $\mathbf{x}_{1:T}^{(1)} \cdots \mathbf{x}_{1:T}^{(m)}$ from real data distribution $p_{\mathcal{X}}(\mathbf{x})$

    Provide embedding $\mathbf{h}_{1:T}^{(1)} \cdots \mathbf{h}_{1:T}^{(m)}$ by applying embedder E_{θ_e} on $\mathbf{x}_{1:T}^{(1)} \cdots \mathbf{x}_{1:T}^{(m)}$.

    Provide recovery $\tilde{\mathbf{x}}_{1:T}^{(1)} \cdots \tilde{\mathbf{x}}_{1:T}^{(m)}$ by applying recovery R_{θ_r} on $\mathbf{h}_{1:T}^{(1)} \cdots \mathbf{h}_{1:T}^{(m)}$.

    Predict embeddings $\tilde{\mathbf{h}}_{1:T}^{(1)} \cdots \tilde{\mathbf{h}}_{1:T}^{(m)}$ by applying supervisor S_{θ_s} on $\mathbf{h}_{1:T}^{(1)} \cdots \mathbf{h}_{1:T}^{(m)}$.

    Update θ_e by descending gradient of $\lambda\mathcal{L}_S + \mathcal{L}_R$ w.r.t θ_e using Adam:

$$\nabla_{\theta_e} \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{T} \lambda ||h_t^{(i)} - \tilde{h}_t^{(i)}||_2 + ||x_t - \tilde{x}_t||_2$$

    Update θ_r by descending gradient of $\lambda\mathcal{L}_S + \mathcal{L}_R$ w.r.t θ_r using Adam:

$$\nabla_{\theta_r} \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{T} \lambda ||h_t^{(i)} - \tilde{h}_t^{(i)}||_2 + ||x_t - \tilde{x}_t||_2$$

(3.3) Discriminator training

    for k steps **do**

      Sample minibatch of m real data samples $\mathbf{x}_{1:T}^{(1)} \cdots \mathbf{x}_{1:T}^{(m)}$ from real data distribution $p_{\mathcal{X}}(\mathbf{x})$

      Provide embedding $\mathbf{h}_{1:T}^{(1)} \cdots \mathbf{h}_{1:T}^{(m)}$ by applying the embedder E_{θ_e} on $\mathbf{x}_{1:T}^{(1)} \cdots \mathbf{x}_{1:T}^{(m)}$.

      Sample minibatch of m fake data samples $\hat{\mathbf{h}}_{1:T}^{(1)} \cdots \hat{\mathbf{h}}_{1:T}^{(m)}$ from generator G_{θ_g}

      Predict $y_{1:T}$ by applying discriminator D_{θ_g} on $\mathbf{h}_{1:T}^{(1)} \cdots \mathbf{h}_{1:T}^{(m)}$

      Predict $\hat{y}_{1:T}$ by applying discriminator D_{θ_g} on $\hat{\mathbf{h}}_{1:T}^{(1)} \cdots \hat{\mathbf{h}}_{1:T}^{(m)}$

      Update θ_d by ascending gradient of $\mathcal{L}_U$ w.r.t. θ_d using Adam:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \log y_{1:T} + \log(1 - \hat{y}_{1:T})$$

    end for

  end for

end for

H Model architectures and hyperparameters

The network architecture of the Embedder, Recovery, Supervisor, Generator, and Discriminator are shown respectively in Table 9, Table 10, Table 11, Table 12, and Table 13. The building block in all models is the LSTM network architecture. Weights are initialized using the technique proposed by He et al. (2016). Table 14 shows the model hyperparameters for every version of the TimeGAN model.

Table 9. *Embedder model network architecture.*

layer	layer type	parameters	output shape
0	Input	normalization:min-max	(, data dim.)
1	LSTM	units:10	(, 10)
2	Dropout	rate:0.1	(, 10)
3	Dense	units:hidden dim., activation:Sigmoid	(, hidden dim.)

Table 10. *Recovery model network architecture.*

layer	layer type	parameters	output shape
0	Input	-	(, hidden dim.)
1	LSTM	units:10	(, 10)
2	Dropout	rate:0.1	(, 10)
3	Dense	units:data dim., activation:Sigmoid	(, data dim.)

Table 11. *Supervisor model network architecture.*

layer	layer type	parameters	output shape
0	Input	-	(, hidden dim.)
1	LSTM	units:hidden dim.	(, hidden dim.)
2	Dropout	rate:0.1	(, hidden dim.)
3	Dense	units:hidden dim, activation:Sigmoid	(, hidden dim.)

Table 12. *Generator model network architecture.*

layer	layer type	parameters	output shape
0	Input	(data dim.)	(batch size, hidden dim.)
1	LSTM	units:10	(, 10)
2	Dropout	rate:0.1	(, 10)
3	LSTM	units:7	(,7)
4	Dropout	rate:0.1	(,7)
5	LSTM	units:hidden dim.	(, hidden dim.)
6	Dropout	rate:0.1	(, hidden dim.)
7	Dense	units:hidden dim, activation:Sigmoid	(, hidden dim.)

Table 13. *Discriminator model network architecture.*

layer	layer type	parameters	output shape
0	Input	-	(batch size, hidden dim.)
1	Bidirectional LSTM	units:7	(batch size, 7)
2	Dropout	rate:0.1	(batch size, 7)
3	Bidirectional LSTM	units:hidden dim	(batch size, hidden dim.)
4	Dropout	rate:0.1	(batch size, hidden dim.)
5	Dense	units:1,activation:Sigmoid	(batch size,1)

Table 14. *Model hyperparameters for training all TimeGAN models.*

Parameter	value
batch size	128
epochs	20,000
Generator iterations (WGAN-GP model)	1
Discriminator iterations (WGAN-GP model)	5
Optimizer Generator and Discriminator (WGAN-GP)	RMSprop
learning rate for Generator optimizer (WGAN-GP)	5e-5
learning rate for Discriminator optimizer (WGAN-GP)	5e-5
Optimizer Generator and Discriminator (GAN, Feature Matching, Positive Label smoothing)	Adam
learning rate Generator (GAN, Feature Matching, Positive Label Smoothing)	1e-4
learning rate Discriminator (GAN, Feature Matching, Positive Label Smoothing)	1e-3
Optimizer (Embedder & Recovery)	Adam
learning rate (Embedder & Recovery)	0.01
Optimizer (Supervisor)	Adam
learning rate (Supervisor)	0.05
RMSprop decay rate	0.9
Adam β_1 (Embedder, Recovery, Supervisor, and Generator)	0.9
Adam β_1 (Discriminator)	0.5
Adam β_2 (Embedder, Recovery, Supervisor, Generator, and Discriminator)	0.999
Parameter γ for gradient penalty in Wasserstein-1 gradient penalty loss $\mathcal{L}_U^{WGAN-GP}$	1
Parameter η for supervised loss $\mathcal{L}_S$ w.r.t. unsupervised loss $\mathcal{L}_U$	1
Parameter λ for supervised loss $\mathcal{L}_S$ w.r.t recovery loss $\mathcal{L}_R$	1
Parameter κ for feature matching loss $\mathcal{L}_{FM}$ w.r.t recovery loss $\mathcal{L}_R$ and unsupervised loss $\mathcal{L}_U$	1