Quantitative Finance

ISSN: 1469-7688 (Print) 1469-7696 (Online) Journal homepage: www.tandfonline.com/journals/rquf20

Fin-GAN: forecasting and classifying financial time series via generative adversarial networks

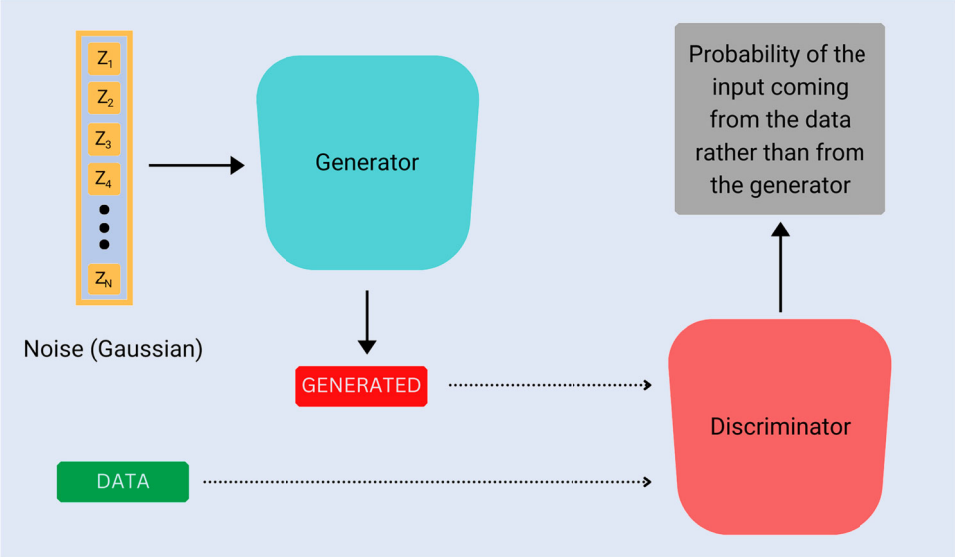
Milena Vuletić, Felix Prenzel & Mihai Cucuringu

To cite this article: Milena Vuletić, Felix Prenzel & Mihai Cucuringu (2024) Fin-GAN: forecasting and classifying financial time series via generative adversarial networks, Quantitative Finance, 24:2, 175-199, DOI: 10.1080/14697688.2023.2299466

To link to this article: https://doi.org/10.1080/14697688.2023.2299466

Published online: 31 Jan 2024.

Quantitative Finance, 2024
Vol. 24, No. 2, 175–199, https://doi.org/10.1080/14697688.2023.2299466



Fin-GAN: forecasting and classifying financial time series via generative adversarial networks

MILENA VULETIĆ*†‡, FELIX PRENZEL† and MIHAI CUCURINGU†‡§¶

†Mathematical Institute, University of Oxford, Andrew Wiles Building, Woodstock Rd, Oxford, OX2 6GG, UK
‡Oxford-Man Institute of Quantitative Finance, University of Oxford, Eagle House, Walton Well Rd, Oxford, UK
§Department of Statistics, University of Oxford, 24-29 St Giles', Oxford, OX1 3LB, UK
¶The Alan Turing Institute, 96 Euston Rd, London, NW1 2DB, UK

(*Received 18 January 2023; accepted 16 December 2023; published online 31 January 2024*)

We investigate the use of Generative Adversarial Networks (GANs) for probabilistic forecasting of financial time series. To this end, we introduce a novel economics-driven loss function for the generator. This newly designed loss function renders GANs more suitable for a classification task, and places them into a supervised learning setting, whilst producing full conditional probability distributions of price returns given previous historical values. Our approach moves beyond the point estimates traditionally employed in the forecasting literature, and allows for uncertainty estimates. Numerical experiments on equity data showcase the effectiveness of our proposed methodology, which achieves higher Sharpe Ratios compared to classical supervised learning models, such as LSTMs and ARIMA.

Keywords: GANs; Financial returns; Time series forecasting; Classification

JEL Classifications: G17, C15, C22, C32, C45, C53

1. Introduction

Time series forecasting has been a core topic of interest for many years, spanning both industry and academia. Most real-world processes are naturally endowed with a time-series structure. In finance and economics, these can be the prices of financial instruments, inflation rates, and many other key macroeconomic indicators. Trading strategies are based on informed beliefs on whether the price of a certain financial instrument will exhibit an upwards or downwards move, along with the magnitude of the move. Hence, time series forecasting and classification are of utmost importance in financial settings. Traditionally, this task has been achieved using methods that rely solely on point estimates. In this work, our goal is to move beyond such classical approaches, and provide *probabilistic forecasts for the future distribution of the target variable* via Generative Adversarial Networks (GANs) (Goodfellow *et al.* 2014).

Classical machine learning and time-series approaches do not allow for probabilistic forecasts, or have strong assumptions on the form of the future distribution of the target variable. This leads to the inability to capture uncertainty, which is crucial for a variety of applications ranging from risk management to healthcare and self-driving cars.

*Corresponding author. Email: milena.vuletic@maths.ox.ac.uk

To this end, we employ the ForGAN (Koochali *et al.* 2019) architecture, which combines GANs with recurrent neural networks (RNNs) (Medsker and Jain 2001) in order to generate probabilistic forecasts. We further introduce a novel economics-driven loss function for the generator in order to improve Sharpe Ratio performance. In particular, we incorporate extra terms into the generator loss function, placing it into a *supervised learning* setting, and rendering it more suitable for *classification tasks* in financial applications. We denote by Fin-GANs the GANs utilizing the ForGAN architecture and trained via the novel economics-driven loss function.

We compare our approach with a classical deep learning method for time-series forecasting, LSTM (Hochreiter and Schmidhuber 1997), and with a standard time-series model used in econometrics, ARIMA (Tsay 2005). We further benchmark our results against long-only strategies and against the same ForGAN architecture trained via the binary cross-entropy (BCE) loss. Aditionally, we investigate the effect of the economics-driven loss function terms on the LSTM, denoting the approach by LSTM-Fin. To our best knowledge, no prior LSTM-based approaches have been trained via PnL-inspired loss functions.

An extensive set of numerical results demonstrate that our proposed methodology attains superior performance on daily stock ETF-excess returns and on daily ETF raw returns, when compared to ARIMA, LSTM, LSTM-Fin, and ForGAN trained using the BCE loss. The aforementioned loss function furthermore shifts the generated data distributions, performing classification and providing uncertainty estimates on the direction of the movement. In addition, the Fin-GAN loss helps alleviate mode collapse, which GANs are notoriously known to suffer from.

Our approach allows for scalability, and potentially modeling of the joint co-movements of different stocks. We explore the notion of universality, in the spirit of Sirignano and Cont (2019). We consider 22 different stocks, across different sectors, and 9 sector ETFs, with each industry sector having at least two different stocks included in the training data. We train the networks on the data set created by pooling together the data on all 31 tickers. We perform a similar experiment using stocks that belong to a single sector (XLP). We test our models both on the stocks included in the training set, and on stocks not seen by the model during the training stage. We find that even for the latter category of new stocks, it is possible to achieve significant Sharpe Ratios.

Summary of main contributions We place Generative Adversarial Networks into a supervised learning setting by introducing a novel economics-based loss function for the generator. Our approach, denoted as Fin-GAN, outperforms a suite of benchmarks on daily stock market data in terms of Sharpe Ratios achieved, while producing distributional forecasts and uncertainty estimates. Furthermore, Fin-GAN allows for learning from co-movements of assets, and alleviates issues around mode collapse.

1.1. Paper outline

The remainder of the paper is organized as follows. We first survey related works in section 2. We cover the main theory pertaining to GANs in section 3. Section 4 reviews different performance metrics in the financial context, providing the motivation for the new loss function. Subsequently, we define the Fin-GAN loss function in section 5, and discuss data and implementation considerations in section 6. We detail the baseline methods in section 7, before moving towards numerical experiments on one-step-ahead forecasting of daily stock excess returns and daily ETF returns in section 8. Finally, section 9 is a summary and future work outlook.

2. Related literature

Time series modeling has long been of interest in both industry and academia. A good overview of classical time-series modeling approaches can be found in Box *et al.* (2015). In recent years, the rise of popularity and interest in machine learning unsurprisingly resulted in an increasing number of applications to time series. However, employing machine learning for time-series modeling in financial settings has been long studied (Refenes *et al.* 1994).

There is a plethora of literature on the use of deep learning in finance. Guresen *et al.* (2011) use neural networks for stock market forecasting, while Sirignano and Cont (2019) utilize LSTMs to argue for the universality of features of asset prices. An area of particular interest for deep learning applications is limit order book (LOB) modeling. Sirignano (2019) introduces spatial neural networks, aiming to model the joint distribution of future states of limit order books conditional on their current states. Tsantekidis *et al.* (2017a) utilize Convolutional Neural Networks for stock price movement forecasting in high-frequency, large-scale data extracted from LOBs, while Tsantekidis *et al.* (2017b) focus on the same problem using a Recurrent Neural Network approach. TransLOB Wallbridge (2020) and DeepLOB Zhang *et al.* (2019b) introduce more complex architectures with high prediction accuracy. Zhang and Zohren (2021) move beyond short-interval LOB forecasting, and focus on multi-horizon settings. Lucchese *et al.* (2022) introduce deepVOL, with the aim of forecasting mid-prices using a volume representation of the LOB. Deep Hedging (Buehler *et al.* 2019) has also sparked interest in deep learning use for hedging in both industry and academia.

GANs (Goodfellow *et al.* 2014) have so far been used for portfolio analysis (Mariani *et al.* 2019), high-frequency stock market prediction (Zhou *et al.* 2018), fine-tuning and calibration of financial trading strategies (Koshiyama *et al.* 2020), and more. QuantGAN (Wiese *et al.* 2020), StockGAN (Assefa *et al.* 2020), FINGAN Takahashi *et al.* (2019) and Buehler *et al.* (2020) focus on generating synthetic financial time series. Li *et al.* (2020) and Prenzel *et al.* (2022) both employ GANs for order flow dynamics in LOBs. GANs have also been used for anomaly (manipulation) detection in financial markets (Leangarun *et al.* 2018), and stock market prediction via GANs has been explored and compared to other methods (Zhang *et al.* 2019a, Romero 2018).

When it comes to GANs adapted to time series, Yoon *et al.* (2019) introduce TimeGAN and Xu *et al.* (2020) propose COT-GAN. Further to the generator and discriminator, TimeGAN utilizes embedding and recovery networks in order to map the data to and from a high-dimensional latent space. COT-GAN introduces a customized loss function based on a regularized Sinkhorn distance, stemming from causal optimal transport theory. However, neither TimeGAN nor COT-GAN are tailored for a forecasting task, and are mainly used for data augmentation. Our methodology has the potential to enhance both approaches, as TimeGAN and COT-GAN could be trained by incorporating the Fin-GAN loss, which we introduce in section 5, adapting them to the task of probabilistic forecasting of (excess) returns.

We focus on the work of Koochali *et al.* (2019), who introduced ForGAN, a conditional GAN suitable for a probabilistic time series forecasting task. This method allows for uncertainty estimates, which the classical time-series forecasting methods are unable to provide.

Another approach to achieve the same end-result relies on Gaussian Processes (Roberts *et al.* 2013), which are, however, very computationally expensive, and cannot capture any forecast distributions other than Gaussian. Hence, we focus on the ForGAN architecture, and in order to improve the Sharpe Ratio performance, we introduce a novel loss function for the generator. Training via customized loss functions is not uncommon in the GAN literature. For example, Cont *et al.* (2022) propose a value function suitable for estimating portfolio tail risks, Bhatia *et al.* (2021) include an extra term to the generator loss in order to generate extreme samples, and Vuletić and Cont (2023) incorporate a smoothness penalty for generating smooth arbitrage-free implied volatility surfaces.

Machine learning models in the univariate setting are able to successfully capture interactions between the input features and the target, even in a nonlinear setting. As previously mentioned, the most popular approaches include deep learning (Goodfellow *et al.* 2016, Wu *et al.* 2020), and tree-based methods such as GBRT (Chen and Guestrin 2016, Ke *et al.* 2017, Dorogush *et al.* 2018). However, being able to model nonlinear *cross-asset* interactions has been a task of significant interest in the recent literature (Wu *et al.* 2023, 2022). The majority of such cross-asset models are mostly linear in nature and come with theoretical guarantees, but cannot effectively model the nonlinear feature-target interaction. A GAN-based approach also has the potential to capture cross-asset interactions in order to improve the overall forecasting task.

3. Generative adversarial networks

Generative Adversarial Networks (GANs) were introduced by Goodfellow *et al.* (2014) and are currently state-of-the-art for various tasks, most notably image and video generation. They also have various applications to science (Mustafa *et al.* 2019), video games (Wang *et al.* 2019), photo editing (Zhang *et al.* 2020), video quality enhancement (Galteri *et al.* 2019), and many other areas. They belong to the family of generative models and their task is generating new samples coming from the unknown data distribution.

3.1. Classical GAN

A classical way to view GANs is through a game theory lens. Suppose we have two players, the generator G and the discriminator D, playing a game against each other. G is generating what it believes is a realistic-looking sample, whereas D is determining whether or not the sample given to it is coming from the data or from the generator. The goal of G is to learn to make D believe that its outputs are coming from the data, and generate samples which look as authentic as possible. On the other hand, D is trying to learn how to properly distinguish between the data samples and those coming from the generator. The generator G improves the generated samples using the feedback it receives from the discriminator D. In GANs, both the generator G and the discriminator D are differentiable functions with respect to their parameters, and they are represented by neural networks. G is given some noise as input, usually Gaussian, and returns a *generated* sample as the output. D takes either the sample generated by G, or a *real* sample coming from the data as input, and outputs the probability of its input being *real*. GANs are trained via an adversarial process which consists of simultaneously training two different models, a generator and a discriminator. In the case of neural networks, this can be efficiently achieved using backpropagation.

Suppose that we have some data $\boldsymbol{x}$ coming from an unknown probability distribution p_{data}, and that we wish to generate more samples from p_{data}.

DEFINITION 3.1 *The generator $G(\boldsymbol{z};\theta_g)$ is a differentiable mapping from the latent space to the data space. It is represented by a neural network, taking input noise $\boldsymbol{z} \sim p_z(\boldsymbol{z})$. The probability distribution to which the outputs of G belong is denoted as p_g.*

REMARK 3.1 Usually, $\boldsymbol{z}$ is a multivariate normal.

DEFINITION 3.2 *The discriminator $D(\boldsymbol{x};\theta_d)$ is a differentiable mapping from the data space to* $(0, 1)$*, represented by a neural network. $D(\boldsymbol{x})$ represents the network's estimated probability of $\boldsymbol{x}$ coming from p_{data} rather than from p_g.*

The discriminator D is trained to maximize the probability of assigning the correct label to both samples from p_{data} and p_g. The generator G is trained to either minimize the log-probability of D correctly assigning its outputs as coming from p_g, or to maximize the log probability of the discriminator being mistaken.

The two networks have separate cost functions which they aim to minimize. Both networks are only able to update their own respective parameters in order to minimize their corresponding cost functions, although the cost function of each network depends on the parameters of the other network. They are required to keep the parameters of the other network fixed while performing optimization. Usually, the cost function taken for the discriminator is the cross-entropy loss.

DEFINITION 3.3 *The cost function for the discriminator is* $J^{(D)}(\theta_d, \theta_g)$ *is defined as*

$$J^{(D)}(\theta_d, \theta_g) = -\frac{1}{2}\mathbb{E}_{p_{data}}[\log[D(\boldsymbol{x}; \theta_d)]] - \frac{1}{2} \times \mathbb{E}_{p_z}\left[\log[1 - D(G(\boldsymbol{z}; \theta_g); \theta_d)]\right]. \quad (1)$$

Learning the zero-sum game with this cost for the discriminator and $J^{(G)} = -J^{(D)}$ for the generator minimizes the Jensen–Shannon divergence between p_{data} and p_g. The game converges to a Nash equilibrium if both policies can be updated in the function space, which is usually not the case in practice, as argued by Goodfellow (2017). When convergence takes place, the discriminator views all outputs as being equally likely to come from p_{data} or p_g, therefore having its outputs converging to $\frac{1}{2}$. However, Goodfellow *et al.* (2014) and Goodfellow (2017) argue that maximizing the cross-entropy loss instead of optimizing the zero-sum game should be favored. This approach corresponds to the generator minimizing the log probability of the discriminator being mistaken. That is, the generator is trying to obtain as high of a score from the discriminator as possible.

DEFINITION 3.4 *The cross-entropy loss for the generator is* $J^{(G)}(\theta_d, \theta_g)$ *is defined as*

$$J^{(G)}(\theta_d, \theta_g) = -\frac{1}{2}\mathbb{E}_{p_z}\left[\log[D(G(\boldsymbol{z}; \theta_g); \theta_d)]\right]. \quad (2)$$

A further argument made by Goodfellow *et al.* (2014) for using the cross-entropy loss for the generator, i.e. maximizing $\log[D(G(\boldsymbol{z}; \theta_g); \theta_d)]$, is that the zero-sum loss may not provide sufficiently large gradients for G to be able to learn well. Early in training, samples from G can be easily distinguishable as not coming from the data, and D can strongly classify them as such, making $\log[1 - D(G(\boldsymbol{z}; \theta_g); \theta_d)]$ saturated. This makes it more difficult for G to learn. Changing the objective to maximizing $\log[D(G(\boldsymbol{z}; \theta_g); \theta_d)]$ results in the same equilibrium point of the dynamics of D and G, while allowing for gradients higher in magnitude, allowing the generator G to learn more. It is common to train the discriminator more, performing k discriminator updates for each generator update. However, Goodfellow (2017) argues that $k = 1$ should be used. An illustration of a standard GAN can be found in figure 1.

The main problem that arises when training GANs is *mode collapse*, also known as the *Helvetica scenario*. It can be full and partial. When it happens, the generator yields the same outputs (full mode collapse) no matter the noise, or the outputs all have similar motifs (partial mode collapse). Durall *et al.* (2021) argues that mode collapse is related to the generator converting towards sharp local minima. When the generator starts reproducing similar outputs and the discriminator cannot identify them as stemming from the generator, due to potentially converging to local minima, the generator receives a high rating from the discriminator. This can lead to weak gradients and close to no updates to the generator network, which can ultimately result in mode collapse.

It is worth noting that keeping the generator and the discriminator in balance is of utmost importance. This is particularly true when the discriminator overpowers the generator. When the discriminator is too successful, the generator cannot learn much from the feedback it receives, due to weak gradients.

In general, GANs are very difficult to train (Arjovsky and Bottou 2017), hence it is natural to consider changing the training objective. Various objectives have been shown to perform well (Huszár 2015), including all f-divergences (Nowozin *et al.* 2016). However, the most commonly used

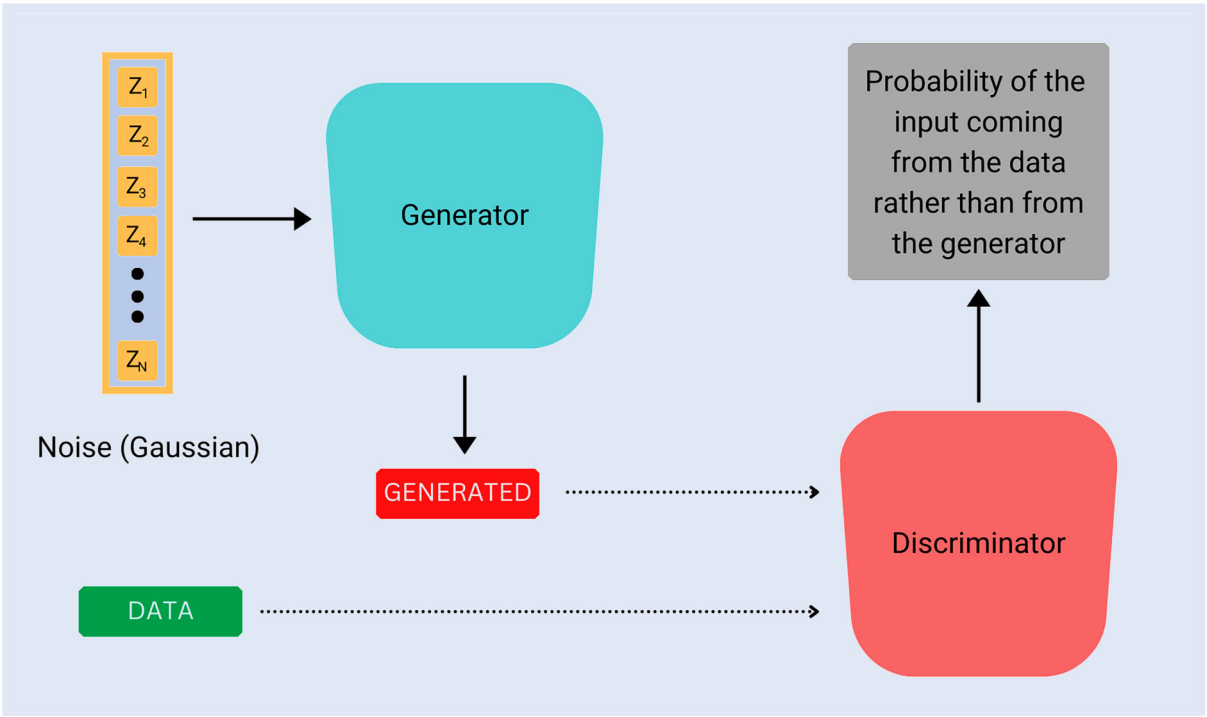


Figure 1. An illustration of a standard GAN.

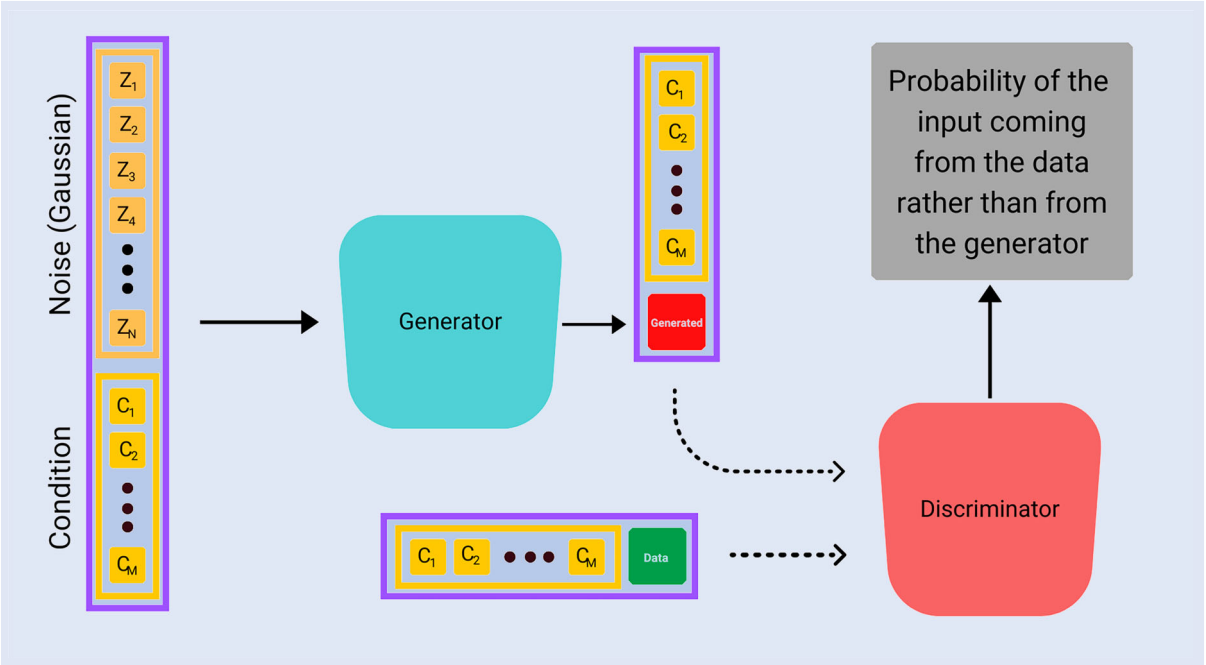


Figure 2. An illustration of a conditional GAN pipeline.

GAN variation, which is not the standard cross-entropy or zero-sum GAN, is a Wasserstein GAN (W-GAN), introduced by Arjovsky *et al.* (2017). It minimizes the Wasserstein-1 distance between p_g and p_{data}. Usually, Wasserstein GANs with Gradient Penalty (Gulrajani *et al.* 2017) are preferred, as they softly enforce the Lipschitz constraint. This approach includes a gradient penalty to the training objective of the discriminator.

3.2. *Conditional GAN*

Conditional generative adversarial networks are a natural extension to standard GANs and were first mentioned in the initial GAN paper (Goodfellow *et al.* 2014), and first explored by Mirza and Osindero (2014). Instead of having all data samples from the same underlying distribution, we assume that the data $\boldsymbol{x}$ is coming from a conditional distribution $p_{data}(\boldsymbol{x}|\text{condition})$, given some deterministic condition. The same principle applies in this case. The only difference in the architecture is that both the generator and the discriminator are given further deterministic information (condition) alongside the standard inputs. When this condition is a label, it is usually encoded as a one-hot-vector. The discriminator/critic is also given the same condition. An illustration of a conditional GAN can be found in figure 2.

As an intuitive illustrative example, we may aim to generate photos of cats and dogs. We have available labels for cats and dogs given as one-hot vectors. The generator would receive noise and a label (say for cats) as an input, and output what it thinks is a photo of a cat. The discriminator would take as input a photo of a cat from the original data, as well as the one generated by the generator network, both alongside the label indicating that a photo should be that of a cat. The discriminator would then give feedback on both the cat photo coming from the real data, and the one coming from the generator, and learn.

3.3. *ForGAN*

ForGAN is a conditional GAN whose architecture is well-suited for probabilistic forecasting of time series, and was introduced by Koochali *et al.* (2019). The idea is to use a conditional GAN to make a probabilistic forecast of x_{t+1} given the previous L values of the series, $x_t, x_{t-1}, \ldots, x_{t-(L-1)}$, which we denote by $\boldsymbol{x}_{t+1}^{-L}$. This allows for much more information to be leveraged, compared to standard point estimates, and renders the approach amenable to obtaining uncertainty estimates.

In order to exploit the time-series structure, both the generator and the discriminator first pass their inputs through a recurrent neural network layer. The ForGAN architecture includes either a gated recurrent unit (GRU) (Cho *et al.* 2014) or a long short-term memory (LSTM) cell (Hochreiter and Schmidhuber 1997), depending on the data. In our work, we chose to work with the LSTM cell because of its prevalence in finance. Similar architectures have already been successfully used for stock market prediction tasks, for example, Zhang *et al.* (2019a). Diagrams illustrating the ForGAN architecture using LSTM cells are shown in figure 3.

4. Performance measures

There are a number of ways to measure model performance. Based on applications, not all performance metrics are equally important. In order to compare one approach to another, it is crucial to understand which properties of the data we care about the most, in order to be able to construct appropriate performance metrics.

4.1. *Standard approach for GANs*

Measuring the performance of GANs is very difficult as we often do not know what distribution the real data is coming

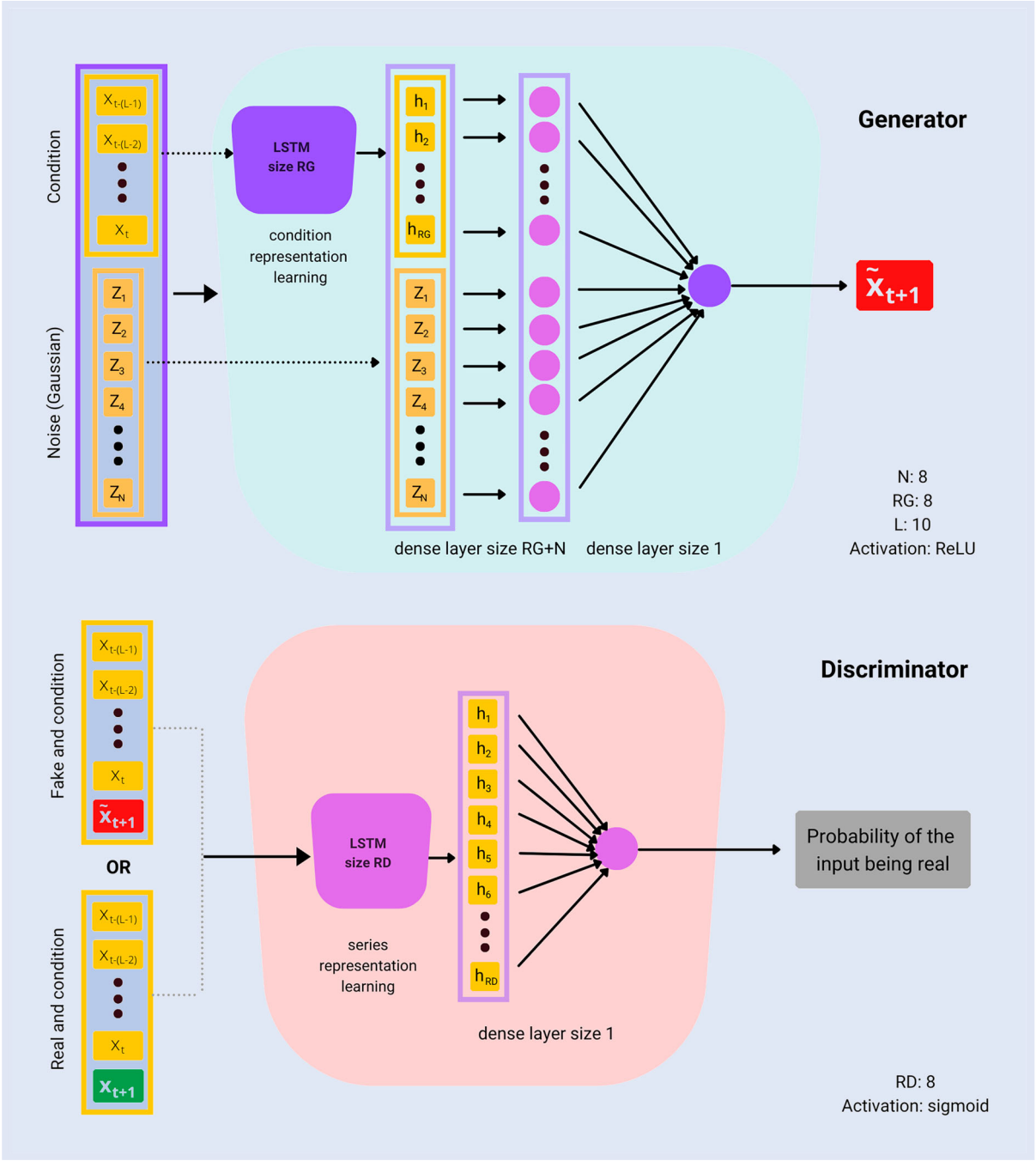


Figure 3. ForGAN architecture using an LSTM cell.

from. Usually, when working with images, videos, or sound, a number of human annotators are there to give feedback on both the generated and real samples. Alternatively, *inception score* (Salimans *et al.* 2016) can be used. It is based on the Kullback–Leibler divergence between the conditional data distribution given the label and the marginal label distribution. This score also takes into account the diversity of generated samples. Unfortunately, none of the approaches mentioned above can be applied to our setting.

4.2. *Standard approach for time series*

Traditionally, when the task at hand is time series forecasting, one opts for performance metrics such as MSE (mean squared error), RMSE(root mean squared error), and MAE (mean absolute error).

To define MAE and RMSE, suppose that the real value we wish to estimate is $\boldsymbol{x}$ and that our forecast is $\hat{\boldsymbol{x}}$. This can be one GAN output given a single noise sample, or the mean or the mode of the generated distribution given the corresponding condition. Then, the mean absolute error and the root mean squared error are defined as

$$MAE(\boldsymbol{x},\hat{\boldsymbol{x}}) = \frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{x}_i|,$$

$$RMSE(\boldsymbol{x},\hat{\boldsymbol{x}}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2}. \tag{3}$$

4.3. Financial data, forecasting returns

As we are working with financial data, MAE and RMSE are not as representative as they are in traditional time-series settings. Forecasting in a financial context is oftentimes performed for the purpose of determining the side of a trade (i.e. whether to buy or sell an asset), which provides motivation for considering a metric based on profit and loss (PnL). While both MAE and RMSE can be very low, it could still be possible to misclassify the side/directionality of a large move in the underlying asset price, leading to a severe PnL loss. Hence we incorporate the PnL as one of our main performance metrics. At each point in time, we produce a forecast of the true value of the (ETF-excess) returns time series, and place a trade according to our forecast. If our forecast of ETF-excess returns is positive, we long the stock and short the corresponding ETF. Conversely, if it is negative, we place the opposite trade. This sequence of trades results in a time series of PnLs. To minimize risk and maximize profit, one aims for the PnL time series to be positive, and have low variance. The larger the average of the PnL series is, the higher the profitability of our strategy. The lower the variance of the PnL series, the lower the risk associated with the strategy. Altogether, this leads us to employ the *annualized Sharpe Ratio* as our main performance measure. We remark that this is also the metric commonly used by portfolio managers to assess the risk-adjusted performance of their strategies.

Next, we define the above terms in the context of the data we are using. When considering stocks, we work with the market-excess returns, while for ETFs, we use raw returns. Assuming that asset A has price S_t^A at time t, its return at time $t+1$ is defined as

$$r_{t+1}^A = \log\left(\frac{S_{t+1}^A}{S_t^A}\right). \tag{4}$$

If the sector to which asset A belongs, has the corresponding ETF I whose return at time t is r_t^I, then the excess return of the asset A at time t is defined as

$$re_t^A = r_t^A - r_t^I. \tag{5}$$

If we expect A to have a higher return than I, then we long one share of stock A and short one share of ETF I. We do analogously for the opposite case. Hence, if our forecast of re_{t+1}^A is $\hat{re}_{t+1}^A$, the PnL of this strategy (at time $t+1$) is

$$PnL_{t+1} = \text{sign}(\hat{re}_{t+1}^A) re_{t+1}^A, \tag{6}$$

where we use the left-continuous version of the sign function

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ -1 & \text{if } x < 0. \end{cases} \tag{7}$$

To streamline the analysis and comparisons, we consider the PnL in basis points and normalize by dividing by the number of total trades placed. In our setting, the number of trades is the size of the validation/test set. If there are n values of re^A that we aim to forecast, denoted $re_1^A, \ldots, re_n^A$, we consider the PnL per trade (in basis points), and for simplicity refer to it as the *pPnL* henceforth. Formally, this amounts to

$$pPnL = \frac{10000}{n} \sum_{i=1}^{n} \text{sign}(\hat{re}_i^A) re_i^A. \tag{8}$$

We note that the *pPnL* defined by (8) does not take into account the uncertainty about the sign of the forecast, and only considers point estimates.

As we have access to full probability forecasts, we would ideally place larger weights on the returns for which we have higher confidence, and lower on those for which we have lower certainty about. Hence, we estimate the probability of the (ETF-excess) return forecast being positive and negative by taking $B = 1000$ different noise samples for the same condition. The two probability estimates are denoted by p_u and p_d, and are more precisely defined as

$$p_u^i = \frac{1}{B}\sum_{j=1}^{B} \mathbb{1}_{\hat{re}_i^A(\boldsymbol{z}^j)\geq 0}, \quad p_d^i = \frac{1}{B}\sum_{j=1}^{B} \mathbb{1}_{\hat{re}_i^A(\boldsymbol{z}^j)<0}, \tag{9}$$

where $\hat{re}_i^A(\boldsymbol{z}^j)$ the is generator's output given the previous L values of the time-series $\boldsymbol{re_i^{A-L}}$ and noise $\boldsymbol{z}^j$. We can now define an alternative PnL, which we can construct as the PnL of a weighted strategy which takes into account our certainty in the sign of the forecast, as

$$wPnL_2^i = 10000(p_u^i - p_d^i) re_i^A. \tag{10}$$

This strategy corresponds to going p_u^i long stock and p_u^i short ETF, and p_d^i short stock and p_d^i long ETF. This is the PnL of an equivalent strategy, consisting of taking an average bet of each of the generator's outcomes given appropriate conditions across different noise samples. Hence, we are penalizing higher uncertainty more, and if we are not very certain about the outcome, we attain significantly lower potential loss.

Note that we are multiplying the PnL by 10,000, in order to obtain the PnL value in basis points. Furthermore, we consider daily data, for which we have available two sets of returns per day (open-to-close and close-to-open), which motivates us to consider the daily PnL

$$wPnL^i = wPnL_2^{2i} + wPnL_2^{2i+1}, \quad i = 1, \ldots, \left\lfloor \frac{n}{2} \right\rfloor. \tag{11}$$

The more symmetric distributions we learn, the closer *wPnL* is to zero. We use the weighted strategy introduced above to compute the annualized Sharpe Ratio, which we use as our main performance metric

$$SR_w = SR_w = \sqrt{252}\frac{PnL_w}{\left(\frac{1}{\lfloor \frac{n}{2} \rfloor}\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor}(wPnL^i - PnL_w)^2\right)^{1/2}},$$

$$PnL_w = \frac{1}{\lfloor \frac{n}{2} \rfloor}\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} wPnL^i. \tag{12}$$

When working with models which provide point estimates only, we are unable to employ the weighted strategy, hence we instead consider

$$PnL^i = 10000\left(\text{sign}(\hat{re}_{2i}^A) re_{2i}^A + \text{sign}(\hat{re}_{2i+1}^A) re_{2i+1}^A\right),$$

$$i = 1, \ldots, \left\lfloor \frac{n}{2} \right\rfloor, \tag{13}$$

and the corresponding annualized Sharpe Ratio

$$SR = \sqrt{252} \frac{PnL_m}{\left(\frac{1}{\lfloor \frac{n}{2} \rfloor} \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} (PnL^i - PnL_m)^2 \right)^{1/2}},$$

$$PnL_m = \frac{1}{\lfloor \frac{n}{2} \rfloor} \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} PnL^i. \tag{14}$$

Annualized Sharpe Ratios of above 1 are considered to be *good*, above 2 are considered to be *very good* and those above 3 are referred to as *excellent*. When reporting the PnL, we use the mean daily PnL, that is PnL_w, or PnL_m, depending on the setting.

REMARK 4.1 Note that we do not take transaction costs into account.

5. Fin-GAN loss function

The main contribution of our work lies in introducing a novel economics-driven loss function for the generator, which places GANs into a **supervised learning** setting. This approach allows us to move beyond traditionally employed methods for pointwise forecasting and move towards probabilistic forecasts, which offer uncertainty estimates. Furthermore, we are evaluating the performance based on the sign of the predictions, meaning that we are interested in **classification**, which our loss function is more suitable for.

5.1. Motivation

The main aim of GANs is to create synthetic samples mimicking real data. However, an important question to ask is which properties of the real data would we like to reproduce via simulations. In our case, it is the sign of the forecast. We are more interested in correctly classifying returns/excess returns, especially when the magnitude of the price moves is large, rather than in producing forecasts close to the realized value. The motivation is that one may produce a forecast which is close to the realized value of the time series, but has the opposite sign. In financial time series forecasting, *correctly estimating the sign when it matters the most* is often more important than forecasting close to the realized value. We aim to provide the generator with more information, such that it better replicates the sign of the data, which is the crucial component of the task. The main reasons and benefits of using the novel loss function terms can be summarized as follows:

- shift the generated conditional distributions in the correct direction,
- render GANs more amenable to a classification task,
- provide effective regularization,
- help the generator learn more, especially in the case of weak gradients,
- help evade mode collapse; by making the generator loss surface more complex, the generator is likely to converge towards sharp local minima.

We have already seen several examples of different loss functions, along with the benefits they bring. Most notably, the W-GAN-GP (Gulrajani *et al.* 2017) adds an extra term to the discriminator loss as a way to enforce the Lipschitz constraint. Adding loss terms to the generator is less common, but it was employed in the recent work by Bhatia *et al.* (2021) for generating extreme samples, and in VolGAN (Vuletić and Cont 2023) for simulating arbitrage-free implied volatility surfaces. Other customized loss functions have also shown to be useful in finance, such as Tail-GAN (Cont *et al.* 2022), which focuses on the estimation of portfolio tail risks.

5.2. Novel loss function

Assume that our GAN aims to forecast x_i, where $i = 1, \ldots, n_{batch}$. For each x_i, given a noise sample $\boldsymbol{z}$ and the condition consisting of the previous L values of the time-series $\boldsymbol{x}_i^{-L}$, the output of the generator is

$$\hat{x}_i = G(\boldsymbol{z}, \boldsymbol{x}_i^{-L}). \tag{15}$$

Let $\boldsymbol{x} = (x_1, \ldots, x_{n_{batch}})$ and $\hat{\boldsymbol{x}} = (\hat{x}_1, \ldots, \hat{x}_{n_{batch}})$ be the forecast of $\boldsymbol{x}$. We define three loss function terms, namely PnL, MSE, and Sharpe Ratio based, which we use to train the generator.

5.2.1. PnL term. The first and most obvious choice of a loss function term to included in the generator's training objective (to maximize) is the PnL. However, the sign function is not differentiable, rendering it impossible to perform backpropagation. In order to alleviate this issue, we propose a smooth approximation to the pPnL, which we denote as PnL^*, defined as

$$PnL^*(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \frac{1}{n_{batch}} \sum_{i=1}^{n_{batch}} PnL_a^*(x_i, \hat{x}_i), \tag{16}$$

where PnL_a^* is a smooth approximation to the PnL for a particular forecast

$$PnL_a^*(x_i, \hat{x}_i) = \tanh(k_{tanh}\hat{x}_i)x_i, \tag{17}$$

and $\tanh(x)$ is defined by

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}. \tag{18}$$

The hyperparameter k_{tanh} controls the accuracy of our approximation. As values of excess returns are usually small, it is desirable for k_{tanh} to be large enough to have a good approximation, but at the same time small enough to have strong gradients which the generator can learn from. Note that in the expression for the PnL term, we consider the PnL per trade. This is not the PnL averaged over days as the training data set is shuffled at the start of every epoch. We use $k_{tanh} = 100$.

5.2.2. MSE term. Even though we are mainly interested in the sign of our forecast, we would also like to have $\hat{x}_i$ close to x_i. In order to enforce this, we add an MSE term to the training objective

$$MSE(\boldsymbol{x},\hat{\boldsymbol{x}}) = \frac{1}{n_{batch}} \sum_{i=1}^{n_{batch}} \left(x_i - \hat{x}_i\right)^2. \tag{19}$$

5.2.3. Sharpe ratio term. As our main measure of performance is the annualized Sharpe Ratio, we introduce an SR-based loss function term for the generator to maximize. However, due to the prohibitive computational cost, we did not implement the weighted strategy during training, and instead worked with point estimates, given one noise sample per condition window. We do not pair up the trades in two, as the consecutive data points in the training set do not necessarily come in that order originally due to shuffling. It is very important to pick mini-batches randomly, or equivalently to shuffle the training data set at the start of every epoch, in order for the stochastic gradient descent to converge. We hence define the Sharpe Ratio term to be maximized by the generator during training, denoted by SR^*

$$SR^*(\boldsymbol{x},\hat{\boldsymbol{x}}) = \frac{\frac{1}{n_{batch}} \sum_{i=1}^{n_{batch}} PnL_a^*(x_i,\hat{x}_i)}{\sqrt{\frac{1}{n_{batch}} \sum_{i=1}^{n_{batch}} \left(PnL_a^*(x_i,\hat{x}_i) - \frac{1}{n_{batch}} \sum_{j=1}^{n_{batch}} PnL_a^*(x_i,\hat{x}_i)\right)^2}}. \tag{20}$$

5.2.4. Standard deviation (of the PnL series) term. Maximizing the Sharpe Ratio means maximizing the PnL and minimizing the standard deviation of the PnLs. Hence, it is natural to consider a standard deviation term, which we define as

$$STD(\boldsymbol{x},\hat{\boldsymbol{x}}) = \sqrt{\frac{1}{n_{batch}} \sum_{i=1}^{n_{batch}} \left(PnL_a^*(x_i,\hat{x}_i) - \frac{1}{n_{batch}} \sum_{j=1}^{n_{batch}} PnL_a^*(x_i,\hat{x}_i)\right)^2}. \tag{21}$$

As the loss term defined above refers to the standard deviation of approximate PnLs, it should only be included in the training objective if the PnL^* term is included, due to the hierarchy principle. The SR^* term and the PnL^* combined with the STD term convey the same information, so the three should not be utilized at the same time. However, although the goal of maximizing SR^* is the same of simultaneously maximizing PnL^* and minimizing STD, the gradient norms are different, which may result in different behavior during training.

5.2.5. An economics-driven loss function for the generator. Finally, we integrate all of the aforementioned loss function terms defined in equations (16), (19) and (20), and arrive at the Fin-GAN loss for the generator

$$\begin{aligned} L^G(\boldsymbol{x},\hat{\boldsymbol{x}}) = J^{(G)}(\boldsymbol{x},\hat{\boldsymbol{x}}) - \alpha PnL^*(\boldsymbol{x},\hat{\boldsymbol{x}}) + \beta MSE(\boldsymbol{x},\hat{\boldsymbol{x}}) \\ - \gamma SR^*(\boldsymbol{x},\hat{\boldsymbol{x}}) + \delta STD(\boldsymbol{x},\hat{\boldsymbol{x}}), \end{aligned} \tag{22}$$

where $J^{(G)}$ is one of the standard GAN losses (zero-sum, cross-entropy or Wasserstein), and the hyperparameters α,β,γ are non-negative. The loss for the discriminator remains $J^{(D)}$. In prior numerical experiments, the Wasserstein loss suffered from explosion, hence we opted to use the binary cross entropy loss for $J^{(G)}$. We aim to minimize a classical generator loss function ($J^{(G)}$), maximize the PnL-based loss function term, minimize the MSE term, and maximize the SR-based loss function term, or jointly maximize the PnL and minimize the STD term. The loss function combinations which we investigate are PnL^*, $PnL^*\&STD$, $PnL^*\&MSE$, $PnL^*\&SR^*$, $PnL^*\&MSE\&STD$, $PnL^*\&MSE\&SR^*$, SR^*, $SR^*\&MSE$, MSE, BCE only. The standard deviation term refers to the PnL term, so due to the hierarchy principle, it is included only if the PnL term is. When it comes to Fin-GAN, there needs to be an economics-driven loss term included, hence for Fin-GAN we only consider the following options: PnL^*, $PnL^*\&STD$, $PnL^*\&MSE$, $PnL^*\&SR^*$, $PnL^*\&MSE\&STD$, $PnL^*\&MSE\&SR^*$, $SR^*, SR^*\&MSE$. In other words, we impose the following conditions on $\alpha,\beta,\gamma,\delta$:

- $\max(\alpha,\gamma,\delta) > 0$ (*at least one additional term other than MSE is included*).
- If $\beta > 0$ then $\max(\alpha,\gamma,\delta) > 0$ (*if the MSE term is included, then another term is included*).
- If $\delta > 0$, then $\alpha > 0$ (the *STD* term is included only if the PnL^* term is).
- $\min(\gamma,\ \delta) = 0$ (*SR^* and STD terms are never included at the same time*).

We refer to GANs utilizing the ForGAN architecture and the loss function defined by equation (22) and the rules above as Fin-GANs.

5.2.6. Gradient norm 'matching'. In order to avoid hyperparameter tuning to find $\alpha,\beta,\gamma,\delta$, we use approach similar to that of Vuletić and Cont (2023), treating all terms as *equally important*. That is, we first train the networks using $J^{(G)}$ only, but we calculate the norms of gradients of all terms in (22) with respect to the parameters of the generator θ_g. We then set $\alpha,\beta,\gamma,\delta$ to be the means of observed ratios (or zero if they are not included in the objective) of the gradient norms of the BCE term to the gradient norms of the corresponding loss function terms. We then train the networks using the possible combinations with the calculated values of $\alpha,\beta,\gamma,\delta$. The final combination of the novel terms is then determined by evaluating the Sharpe Ratio on the validation set. Further training details are explained in section 6, and the Fin-GAN algorithm is given in Algorithm 1.

5.3. Benefits and challenges of the Fin-GAN loss function

The novel loss function terms do indeed shift the generated distributions, help evade mode collapse, and improve Sharpe Ratio performance, which we demonstrate in an extensive set of numerical experiments. Including the new loss function terms introduces four new hyperparameters to be tuned, rendering the optimization a more challenging problem. However, the *gradient norm matching* approach mitigates this issue.

Algorithm 1 Fin-GAN algorithm

Input: Hidden dimension of the generator, hidden dimension of the discriminator, noise dimension. Target size, condition window, sliding window. Discriminator learning rate, generator learning rate. Training data, validation data, testing data. Number of epochs for gradient matching n_{grad}, number of epochs for training n_{epochs}, minibatch size n_{batch} , mode collapse threshold ϵ. Number of samples for the weighted strategy B.

Step 0: Take the first n_{batch} items from the training dataset as the reference batch for data normalization. Initialize the generator *gen* and the discriminator *disc* networks.

Step 1: Matching the gradient norms to find $\alpha, \beta, \gamma, \delta$.

for n_{grad} epochs **do**
  Split the training data into the minibatches.
  for number of minibatches **do**
    Calculate norms of gradients of the *BCE*, PnL^*, *MSE*, SR^*, *STD* terms with respect to θ_g.
    Label them as $grad_0, grad_\alpha, grad_\beta, grad_\gamma, grad_\delta$.
    Update *disc* and then *gen* parameters via RMSProp and the BCE loss.
  end for
end for
$\alpha \leftarrow \text{mean}(grad_0/grad_\alpha)$; $\beta \leftarrow \text{mean}(grad_0/grad_\beta)$;
$\gamma \leftarrow \text{mean}(grad_0/grad_\gamma)$; $\delta \leftarrow \text{mean}(grad_0/grad_\delta)$

Step 2: Training and validation.

Label the possible loss function combinations (PnL^*, PnL^*&*STD*, PnL^*&*MSE*, PnL^*&SR^*, PnL^*&*MSE*&*STD*, PnL^*&*MSE*&SR^*, SR^*, SR^*&*MSE*) as L_i, for $i = 1, \ldots, 8$, respectively.

for $i = 1, \ldots, 8$ **do**
  $gen_i \leftarrow gen$; $disc_i \leftarrow disc$.
  for n_{epochs} **do**
    Split the training data into the minibatches.
    for number of minibatches **do**
      Update *disc* via RMSProp and $J^{(D)}$.
      Update *gen* via RMSProp and L_i.
    end for
  end for
  Take B samples from gen_i for each day in the validation set.
  $SR^i_{val} \leftarrow$ Sharpe Ratio of the weighted strategy on the validation set.
end for
$i^* \leftarrow \text{argmax}\{SR^i_{val} : i = 1, \ldots 8\}$; $gen^* \leftarrow gen_{i^*}$.

Step 3: Evaluation on the test set.

For each time t in the test set, take B i.i.d. outputs from gen_{i^*}, $\hat{r}^i_t$, $i = 1, \ldots, B$.
Point estimate for each time $\tilde{r}_t \leftarrow \text{mean}(\hat{r}^i_t)$.
$MAE \leftarrow MAE(r_t, \tilde{r}_t)$; $RMSE \leftarrow RMSE(r_t, \tilde{r}_t)$;
Implement the weighted strategy, pair up days into two.
Calculate the annualized Sharpe Ratio SR_w and the mean daily PnL PnL_w.
if $std(\{\hat{r}^i_0\}_{i=1,\ldots,B}) < \epsilon$ or $std(\{\tilde{r}_t\}_{t \in \text{test set}}) < \epsilon$ **then**
  Mode collapse.
else
  No mode collapse.
end if

In addition, the loss surface becomes more complex, raising convergence challenges. In initial numerical experiments, when exploring a wider range of hyperparameters α, β, γ, the MSE and the Sharpe Ratio terms appeared to encourage the generator to produce very narrow distributions. It is not immediately obvious why the Sharpe Ratio term with a high γ coefficient would result in mode collapse, as it encourages the *PnLs* to have a low standard deviation, and not the generated samples. However, including the PnL-based term helped alleviate the issue of mode collapse, which is related to convergence to sharp local minima (Durall *et al.* 2021). The PnL term helps the generator escape such areas of the loss surface.

When *gradient norm matching* was used to tune the hyperparameters $\alpha, \beta, \gamma, \delta$, there was no mode collapse in Fin-GAN no matter the initialization of network weights. However, we noticed that using Xavier initialization (Glorot and Bengio 2010) instead of He initialization (He *et al.* 2015) helped reduce the number of iterations of ForGAN with mode collapse. With He initialization, this was occuring in 67% of the cases. utilizing normal Xavier initialization resolved this issue.

The classical generator loss term, i.e. the feedback received from the discriminator, allows learning conditional probability distributions, instead of only pointwise forecasts.

The remaining loss function terms promote sign consistency (enforced by the PnL and Sharpe Ratio terms), while at the same time targeting to remain close to the realized value (MSE).

If the hyperparameters $\alpha, \beta, \gamma, \delta$ are too large, the feedback of the discriminator becomes negligible, and we move towards the standard supervised learning setting, using a generator-only model. On the other hand, if $\alpha, \beta, \gamma, \delta$ are too small, we are not providing any extra information to the generator, thus remaining in a classical GAN setting. The additional information about the data communicated through the novel loss function terms is especially invaluable in the case of weak gradients coming from the discriminator/critic feedback, as the generator can still make progress in this case. This all together leads to a better classification of the data in terms of directional price movements, i.e. correctly classifying the sign of future time series values, alongside uncertainty estimates of the movement direction. The gradient norm matching procedure ensures that all loss terms are treated as equally important, and that the corresponding hyperparameters are neither too large, nor too small.

6. Data description and implementation considerations

In this section, we outline the main characteristics of the data sets we use, and further describe the ForGAN architecture. Next, we discuss the training setup, including the hyperparameter choice as well as optimization. Finally, we demonstrate the general behavior of the Fin-GAN model, which recurrently surfaced throughout the numerical experiments.

6.1. Data description

We consider daily stock ETF-excess returns and daily raw ETF returns, extracted from CRSP on Wharton Research Data Services. The time frame of interest is Jan 2000-Dec 2021. All time series have the same training-validation-testing split, namely $80-10-10$. That is, the training period is 3rd Jan 2000–9th Aug 2017, the validation period is 10th Aug 2017–22nd Oct 2019, and the testing period is 23rd Oct 2019–31st Dec 2021. These three time periods are very different from an economic perspective, and it is important to note that the test data includes the start of the Covid-19 pandemic, rendering the problem more challenging. We consider close-to-open and open-to-close returns sequentially, and refer to each one of them as one unit of time. In other words, one input time series alternates intraday open-to-close and overnight close-to-open returns. The sliding window is a one time unit, the condition window is ten time units (corresponding to five days of returns, i.e. one full trading week), and the prediction window is one time unit ahead.

The prices are adjusted by division with the cumulative adjustment factor provided (*cfacpr*), and the returns are capped at $\pm 15\%$, to alleviate the effect of potential outliers. To pre-process the data, we first create one time series consisting of consecutive close-to-open and open-to-close returns, then chronologically split the data into three disjoint shorter time series (training-validation-testing), and we appropriately further process the three resulting time series. At this stage, we create a matrix for each of the three data sets; each such matrix containing 11 columns, the first 10 corresponding to the condition window $(x_1, \ldots, x_{10})$, and the last column being the target (x_{11}). A sliding window of one time unit implies that the kth value in the column j is the $k+1$st value in the column $j-1$.

Table 1. Tickers and ETFs used for numerical experiments.

Sector name	ETF ticker	Stock tickers
Consumer Discretionary	XLY	AMZN, HD, NKE
Consumer	XLP	CL, EL, KO, PEP
Energy	XLE	APA, OXY
Financial	XLF	WFC, GS, BLK
Health Care	XLV	PFE, HUM
Industrial	XLI	FDX, GD
Technology	XKL	IBM, TER
Materials	XLB	ECL, IP
Utilities	XLU	DTE, WEC

We consider ETF-excess returns for 22 different stocks across nine sectors, and **raw returns** of the nine sector ETFs. All stocks are current members of the S&P500 index. The tickers, along with their sector memberships, are shown in table 1.

6.2. ForGAN architecture

For the single-stock/ETF numerical experiments, the classical ForGAN architecture utilizing LSTM cells, as shown in figure 3, was used. Due to the small size of the data sets used, the corresponding layer dimensions and the noise dimension are all set to 8. The generator uses ReLU as activation function, while the discriminator uses the sigmoid.

6.3. Training

Avoiding further hyperparameter tuning, the optimizer used throughout is RMSProp (Hinton *et al.* 2012), and normal Xavier initialization (Glorot and Bengio 2010) is used for the weights. The discriminator/critic and the generator are trained *using an alternating direction method, having* $k = 1$ *discriminator update per one generator update. We use reference batch normalization, picking the first* 100 *samples from the training data serving as the reference batch, in order to not be anticipative. Mini-batch size is also set to* 100. *The learning rates of both networks are set to* 0.0001. *We train for* 25 *epochs at the start, using the BCE loss only in order to implement the gradient norm matching, and find the values for* $\alpha, \beta, \gamma, \delta$ *coefficients. We then branch away from the generator and the discriminator by training via every suitable loss combination for* 100 *more epochs. At the end of the training stage, we calculate Sharpe Ratio on the validation set, and choose the loss function combination, and the optimized generator which maximizes it. Due to differences in economic periods for validation and testing, it may not be the case that the networks which would perform well on*

the test set are chosen. However, this approach allows us to ensure good performance, regardless of periods of crisis. The code is implemented in PyTorch. The Fin-GAN algorithm is summarized in Algorithm 1.

REMARK 6.1 One may explore a wider range of hyperparameters α, β, γ, δ, learning rates lr_g, lr_d, hidden dimensions *RG*, *RD*, noise dimension *N*, as well as completely different architectures, by evaluating Sharpe Ratio on the validation set, and opting for the hyperparameter/architecture maximizing it.

6.4. Gradient stability

Before incorporating the additional loss function terms, we first investigate the behavior of the gradient norms during training, in order to check for exploding and vanishing gradients. We train ForGAN, performing updates on the BCE loss using RMSProp (Hinton *et al.* 2012), for 25 epochs, the period used for *gradient norm matching*. In figure 4, we display sample gradient norms of each term equation (22). The data used in this example is the daily excess stock returns time series of PFE. We observe that there are no vanishing or explosion phenomena, and that the gradient norms are on a similar scale. This also holds true for other tickers used in our experiments.

6.5. Mode collapse

As we are considering time series data, we identify two different types of mode collapse, *distribution mode collapse* and *mean mode collapse*, each of which can be full or partial.

- *Distribution mode collapse* refers to a collapse in the generated distribution given a condition,
- *Mean mode collapse* refers to a collapse in the distribution of the means of generated distributions across all conditions.

It is a major concern when both full mean and full distribution mode collapse occur simultaneously, as this leads to only having a single point estimate. In order to simplify the notion of mode collapse in our setting, we consider that mode collapse occurs if the out-of-sample generated scenarios for a particular condition (time point), that is the time series of the generated means, have standard deviation below a threshold ϵ, which we set to 0.0002. Using Xavier initialization for the network weights (Glorot and Bengio 2010) helped alleviate the mode collapse of ForGAN. The additional terms of the Fin-GAN loss function have shown to have effective regularization, as there was no mode collapse in any of the experiments, no matter if the networks were initialized according to Xavier or He initialization (He *et al.* 2015). It is important to note that ForGAN suffered from either type of mode collapse in 67% of simulations when He initialization was used. Furthermore, in the initial numerical experiments (prior to considering the STD term), in which a wider range of hyperparameters was used instead of performing *gradient matching*, we noticed that the MSE and SR terms with high β and γ were likely to cause mode collapse. However, the inclusion of the PnL term would resolve this issue. The consequence of the high hyperparameter next to the MSE term resulting in mode collapse is not surprising as the MSE term is encouraging the generated values to be close to the target. The Sharpe Ratio term, even though promoting small standard deviation, is not promoting narrow distributions of forecasts, but PnLs.

6.6. Generated distributions

Samples of generated distributions when training using different Fin-GAN loss function term combinations are displayed in figure 5. All of the distributions shown are generated using the same training data (excess returns of PFE), and the same condition in the test set (they have the same target). Throughout the numerical experiments, training via just the BCE loss on its own would produce more symmetrical distributions, whereas the incorporated Fin-GAN loss function terms would help shift the forecast distributions. We also investigate the generated means and compare them with the true distributions of the target in figure 6. We note that in this particular example, the PnL and STD combination has a similar behavior to the SR term on its own. This is not surprising as both convey the same information, although they have different gradients. Both options have the biggest distribution-shifting effect in this example. We note that apart from the PnL with STD, and SR loss terms, all of the generated means resemble the true target distribution. Furthermore, including the *MSE* term prompted the distribution of the means to be closer to that of the true returns.

Figure 5 allows us to illustrate the weighted strategy that GANs are able to employ due to uncertainty estimates. In this particular forecast, the SR, as well as PnL and STD, would result in long ETF and short stock with weight one, whereas the PnL, MSE and SR combination would have a similar trade of a slightly lower weight, and the other combinations would result in very small trades due to uncertainty about the sign of the forecast. Although we are considering the Fin-GAN loss terms only in figures 5 and 6, it is important to note that the average correlation between the BCE loss and the BCE loss with the MSE term added to it is 99.7%.

We observe that it is beneficial to train on a combination of the loss terms, as they are able to produce shifts in generated distributions and evade mode collapse. The benefit of jointly utilizing the Fin-GAN loss function terms is backed up by the Sharpe Ratio performance, which we analyse in section 8.

It is important to highlight that the configuration chosen as optimal during the validation stage will vary from instance to instance. Figures 5 and 6 are illustrative examples of resulting distributions obtained by training via different objectives.

7. Baseline algorithms

Apart from the standard ForGAN (GAN with the ForGAN architecture trained via the BCE loss), we compare our Fin-GAN model with more standard supervised learning approaches to time series forecasting: ARIMA and LSTM. For completeness, and to demonstrate that the task at hand

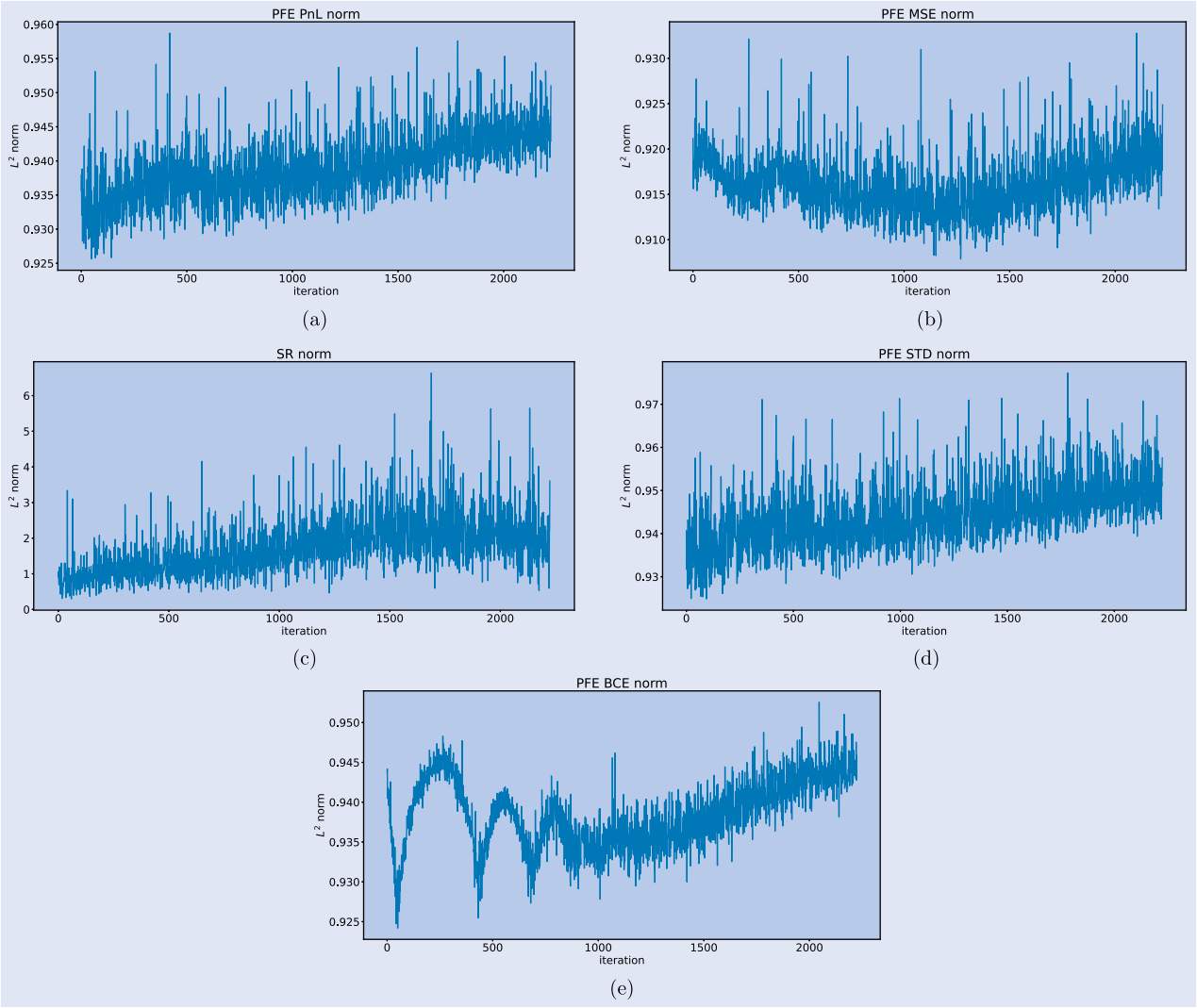


Figure 4. Sample generator gradient norms during training of different terms (PnL, MSE, SR, STD, BCE) with respect to θ_g. Updates were performed using the BCE loss only. (a) PnL term. (b) MSE term. (c) SR term. (d) STD term and (e) BCE term.

is non-trivial, we further include PnL and Sharpe Ratio values of long-only strategies, where the predicted sign is $+1$ for each observation. All comparisons have the same training-validation-testing split as those used in the Fin-GAN experiments.

7.1. ARIMA

We first recall the *auto regressive integrated moving average* (ARIMA), a very classical time series model (Tsay 2005). The parameters p, d, q in ARIMA(p,d,q) correspond to the autoregressive, differencing, and moving average parts, respectively. The differencing coefficient d indicates how many times we need to difference our initial time series in order to reach a time series X_t which has no unit roots. To determine d, we perform the augmented Dickey–Fuller (ADF) test. In our case, all of the time series had p-values smaller than 10^{-6}, indicating stationarity. This is not surprising, since we are working with returns and excess returns, which are already differenced log-prices time series. Therefore, in our setting, we set $d = 0$ throughout, hence working with ARMA(p,q) models of the form

$$X_t = \alpha_1 X_{t-1} + \cdots + \alpha_p X_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q}, \tag{23}$$

where $\epsilon_1, \ldots, \epsilon_t$ are white noise terms, θ_i are the moving average parameters, and α_j are the autoregressive parameters. In order to determine the most suitable p and q, we plot the autocorrelation (ACF) and partial autocorrelation (PACF), which indicate that $p, q \in \{0,1,2\}$. We fit ARMA(p,q) with $(p,q) \in \{(1,0),(1,1),(2,0),(2,1),(2,2)\}$ and choose the model with the lowest Akaike Information Criterion (AIC).

7.2. LSTM

Long short-term memory networks, or LSTMs (Hochreiter and Schmidhuber 1997), belong to the class of recurrent neural networks. They help with the vanishing/exploding gradients problem, and are particularly well-suited for working with time series. At every time t, there is the input $x(t)$, the previous cell state $c(t-1)$ and the previous hidden state

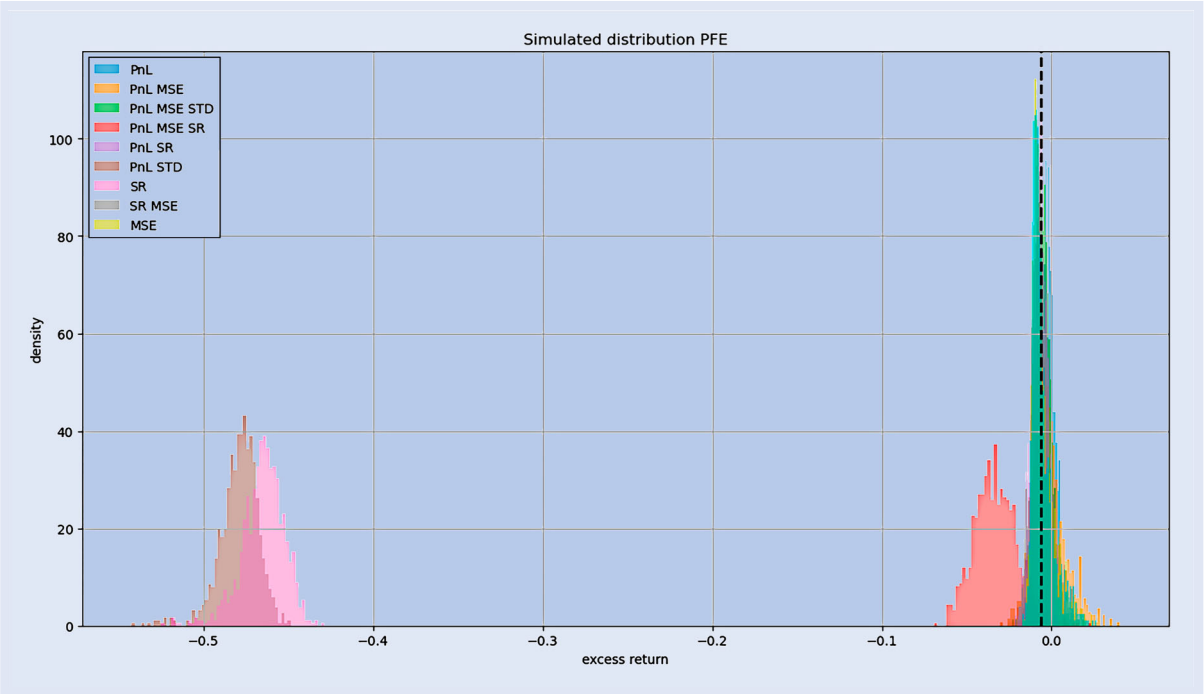


Figure 5. An illustration of how the Fin-GAN loss function terms can shift generated distributions. All the distributions shown are generated using the same condition window. The black vertical line is the true value, the target. The data used are the ETF-excess returns of PFE.

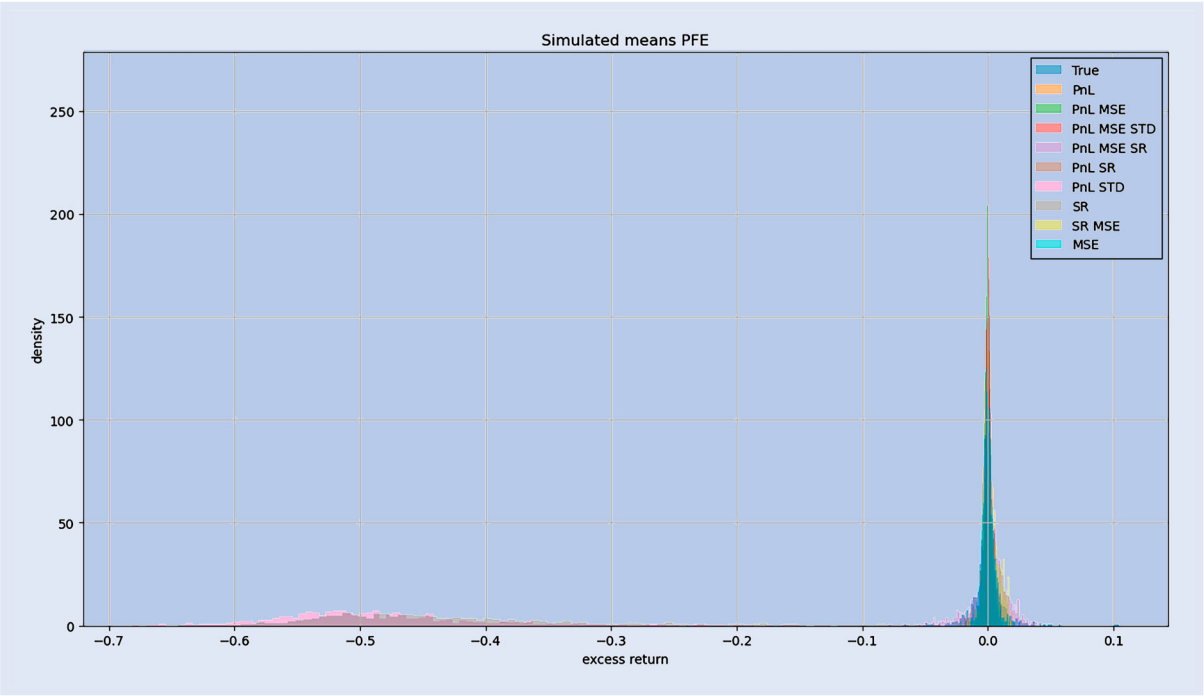


Figure 6. Illustration of generated out-of-sample (one-step-ahead) means on the test set, obtained by training on different loss function combinations. Training data: PFE excess returns. The loss function with the best Sharpe Ratio performance on the validation set is PnL-MSE-SR, for this particular instance.

$h(t-1)$ (the previous LSTM output). These are then processed by the forget gate, the input gate, and the output gate, in order to create the new cell state $c(t)$ and the new hidden state, the output, $h(t)$. An illustration of the LSTM cell architecture is shown in figure 7.

For training, we use the same setup as for Fin-GAN. That is, we use RMSProp as the optimizer of choice, train for 125 epochs in total. We are in a regression setting now, so the LSTM is trained to minimize the MSE loss. The rate is once again 0.0001.

7.2.1. LSTM-Fin. We also train LSTM on the appropriate combinations of the (baseline) *MSE* loss, PnL^*, SR^*, and *STD* loss terms, in order to have a better comparison with

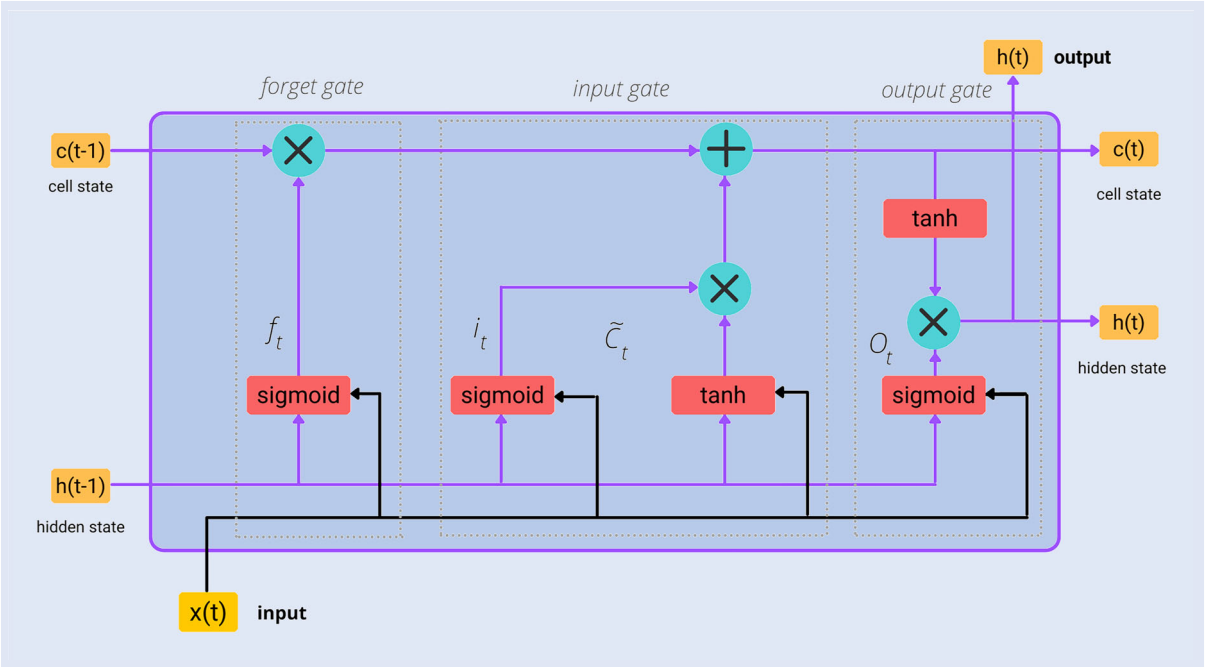


Figure 7. Illustration of an LSTM cell.

Fin-GAN. We use the same methodology as we did in the Fin-GAN setting, also performing gradient norm matching to determine the values of the hyperparameters corresponding to the newly included loss terms. That is, we consider the loss function (24), including and excluding the PnL^*, SR^*, and STD loss terms via the same rules that apply to Fin-GAN, determining the values of the hyperparameters α, γ, δ using the same gradient norm matching procedure, and determining the final loss function combination to be used for testing by evaluating the Sharpe Ratio on the validation set.

$$L^{Fin}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = MSE(\boldsymbol{x}, \hat{\boldsymbol{x}}) - \alpha PnL^*(\boldsymbol{x}, \hat{\boldsymbol{x}}) - \gamma SR^*(\boldsymbol{x}, \hat{\boldsymbol{x}}) + \delta STD(\boldsymbol{x}, \hat{\boldsymbol{x}}). \quad (24)$$

7.3. *Long-only strategies*

We also remark on the results of long stock and short ETF (for the stock data); and long-only ETF (for the ETF data), which corresponds to constantly forecasting $+1$ for the sign.

8. Numerical experiments

In this section, we first consider a single-ticker setting. That is, we train Fin-GAN and the baselines on ETF-excess returns/raw returns of a particular stock/ETF. We then explore the notion of universality, in the spirit of Sirignano and Cont (2019), on three different sets of stocks. We pool the data across the tickers used in the single-ticker setting, and train Fin-GAN as if the data was coming from the same source. We repeat this methodology on stocks belonging to the same sector (Consumer Staples), with and without the XLP raw returns used in the training stage. In the universal setting, we further investigate the performance of Fin-GAN on stocks not previously seen by the model during training.

8.1. *Single stock & ETF settings*

We first examine the summarized performance of the models under consideration: Fin-GAN, ForGAN (trained via the BCE loss), LSTM, LSTM-Fin, ARIMA, and the long-only approach. The statistics of interest are the annualized Sharpe Ratio, mean daily PnL, MAE, and RMSE. All statistics are summarized by computing the mean and the median across the 31 data sets (22 stocks and 9 ETFs) used in the numerical experiments. Additionally, we report on the *portfolio Sharpe Ratio* by first summing the daily Pnls across all tickers in the universe, and then computing the Sharpe Ratio of the resulting daily PnL time series. These summary statistics are shown in table 2.

We remark that the highest mean, median, and portfolio Sharpe Ratios are achieved by Fin-GAN, followed by LSTM. The median Sharpe Ratio achieved by Fin-GAN is close to twice that of LSTM. Furthermore, we note that despite the Sharpe Ratio being the highest when using the Fin-GAN approach, the highest mean PnLs are achieved by the LSTM, and the highest median PnLs by ARIMA. This altogether demonstrates the significant reduction in variance and higher consistency of the generated PnL by Fin-GAN, compared to the other models. It is also a consequence of placing smaller trades compared to other strategies due to the ability to leverage the uncertainty estimates and implement the weighted strategy. We stress the beneficial effect of the novel loss function terms on performance, evidenced by the mean, median, and portfolio Sharpe Ratios achieved by Fin-GAN being significantly higher than those achieved by ForGAN trained on the BCE loss only.

Table 2. Summary of performance metrics over the models across the stocks and ETFs.

	Fin-GAN	ForGAN	LSTM	LSTM-Fin	ARIMA	Long-only
Mean SR	**0.540**	0.033	0.467	0.341	0.206	0.182
Median SR	**0.413**	-0.092	0.214	0.170	0.204	0.194
Portfolio SR	**2.107**	0.172	2.087	0.942	0.612	0.618
Mean PnL	2.978	0.25	**4.123**	2.361	2.059	2.350
Median PnL	1.890	-0.673	1.959	1.735	**2.245**	1.975
Mean MAE	0.044	0.052	0.007	0.007	**0.007**	
Median MAE	0.008	0.009	**0.007**	0.007	0.007	
Mean RMSE	0.049	0.056	0.012	0.012	**0.012**	
Median RMSE	0.012	0.014	0.011	0.011	**0.011**	

Notes: SR refers to the annualized Sharpe Ratio, and PnL refers to the mean daily PnL. MAE and RMSE represent the mean absolute error and the mean root squared error, respectively. Highlighted are the best-performing results according to each metric.

We further note that ARIMA attains the best performance in terms of RMSE and achieves the best mean MAE, while LSTM achieves the lowest median MAE. We also remark that the non-GAN models have their MAE and RMSE summary statistics on the same scale, while the mean MAE and RMSE are significantly higher for the GAN models. However, the medians are on a similar scale to the other models, and errors are reduced when moving from ForGAN to Fin-GAN. Investigating RMSE and MAE in figures 9 and 8, respectively, we find that the main cause of such high mean MAE and RMSE achieved by Fin-GAN is performance on IBM. However, Fin-GAN achieves higher Sharpe Ratio (figure 10) and PnL (figure 11) than the other models in this case. Similarly, even though ARIMA attains values close to the realized ones, it does so on the opposite side of the real line, when the movements in the underlying are large, resulting in lower PnL and lower Sharpe Ratios. We note that the summary statistics achieved by the non-deep learning approaches, ARIMA and long-only, are similar, and that they achieve better performance than ForGAN trained in a classical way. The long-only column indicates that the task at hand is non-trivial, given that the only ticker on which a Sharpe Ratio above one is achieved is XLK. This is not surprising given the continued increase in stocks belonging to the technology sector during the Covid-19 pandemic. Furthermore, the positive effect that the economics-driven loss function terms had in the GAN setting is not visible in the case of LSTMs.

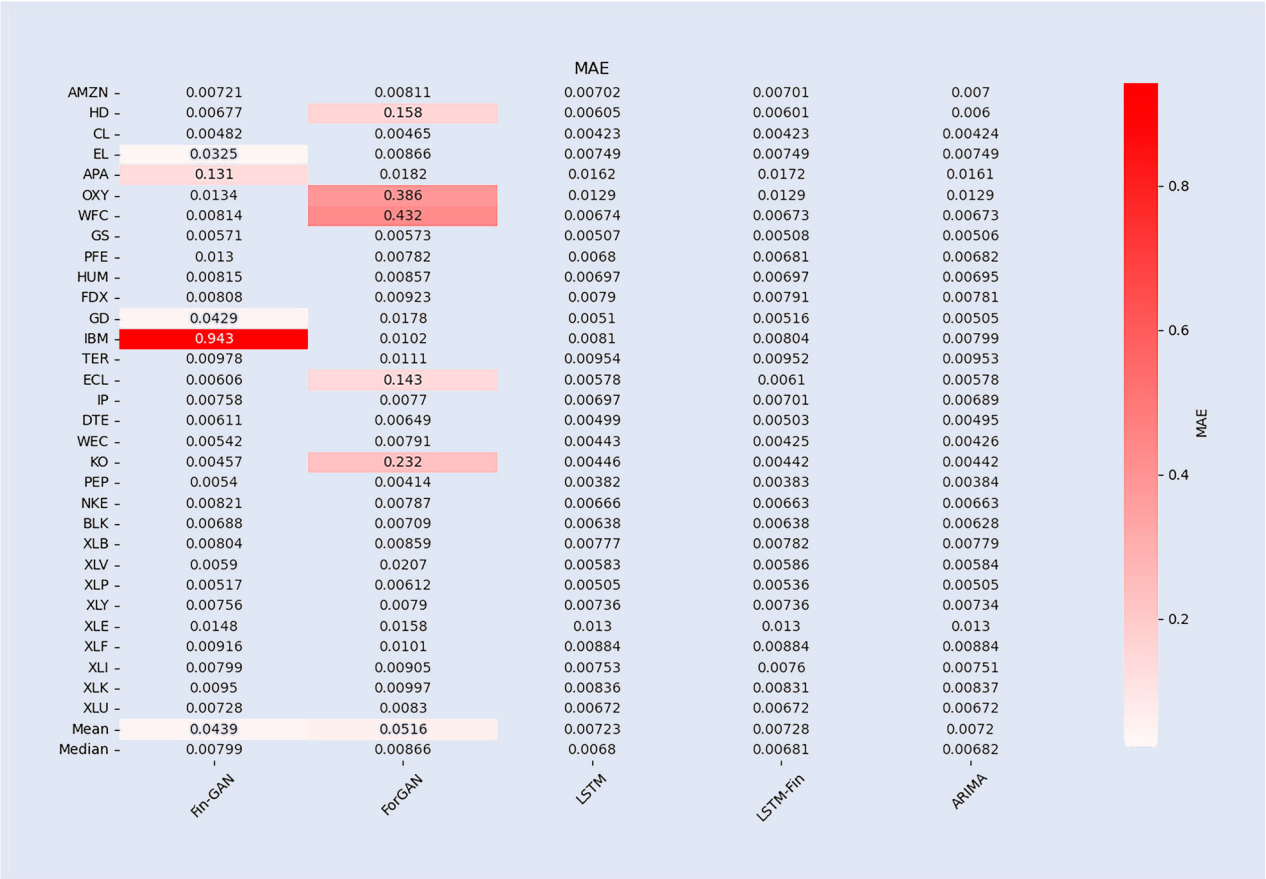


Figure 8. MAE obtained by different methods.

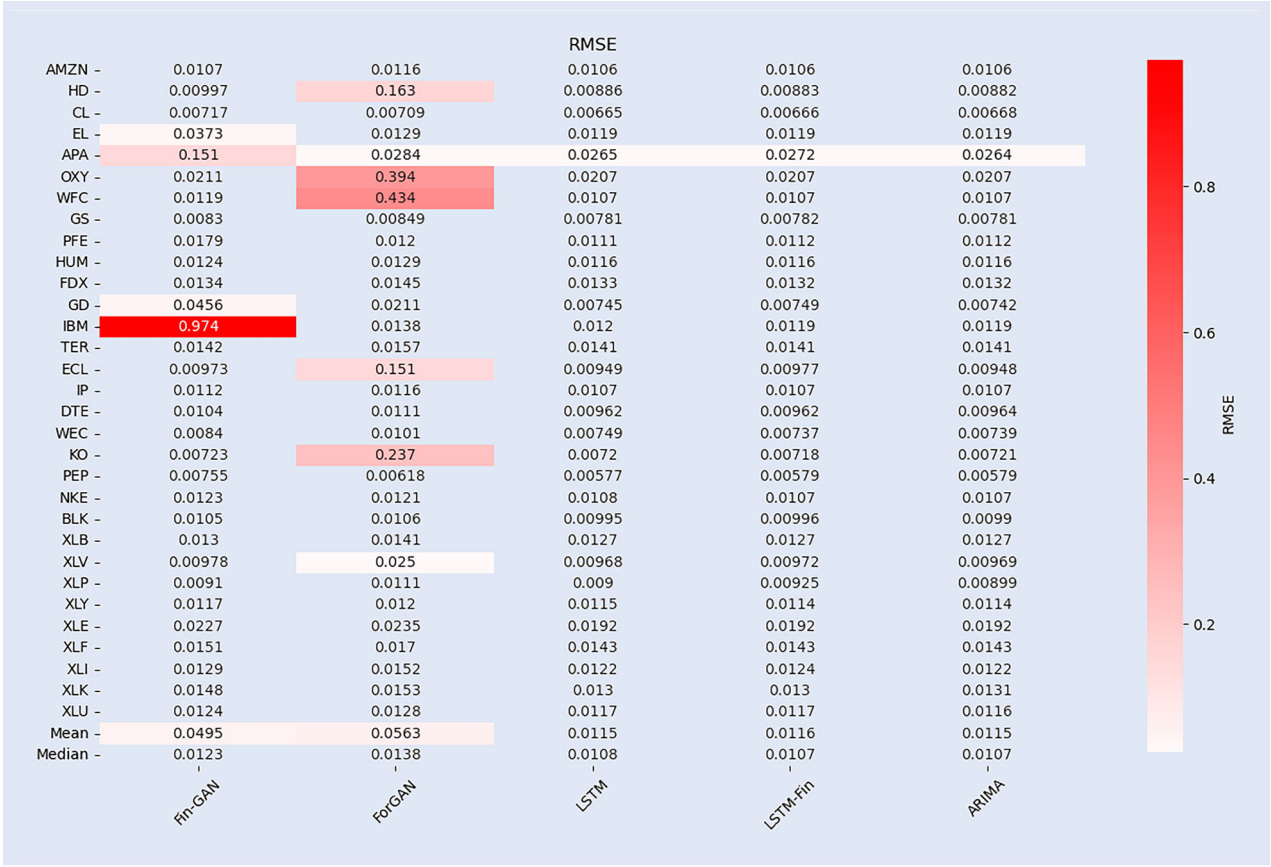


Figure 9. RMSE obtained by different methods.

Figure 10. SR (annualized Sharpe Ratio) obtained by different methods on the test set.

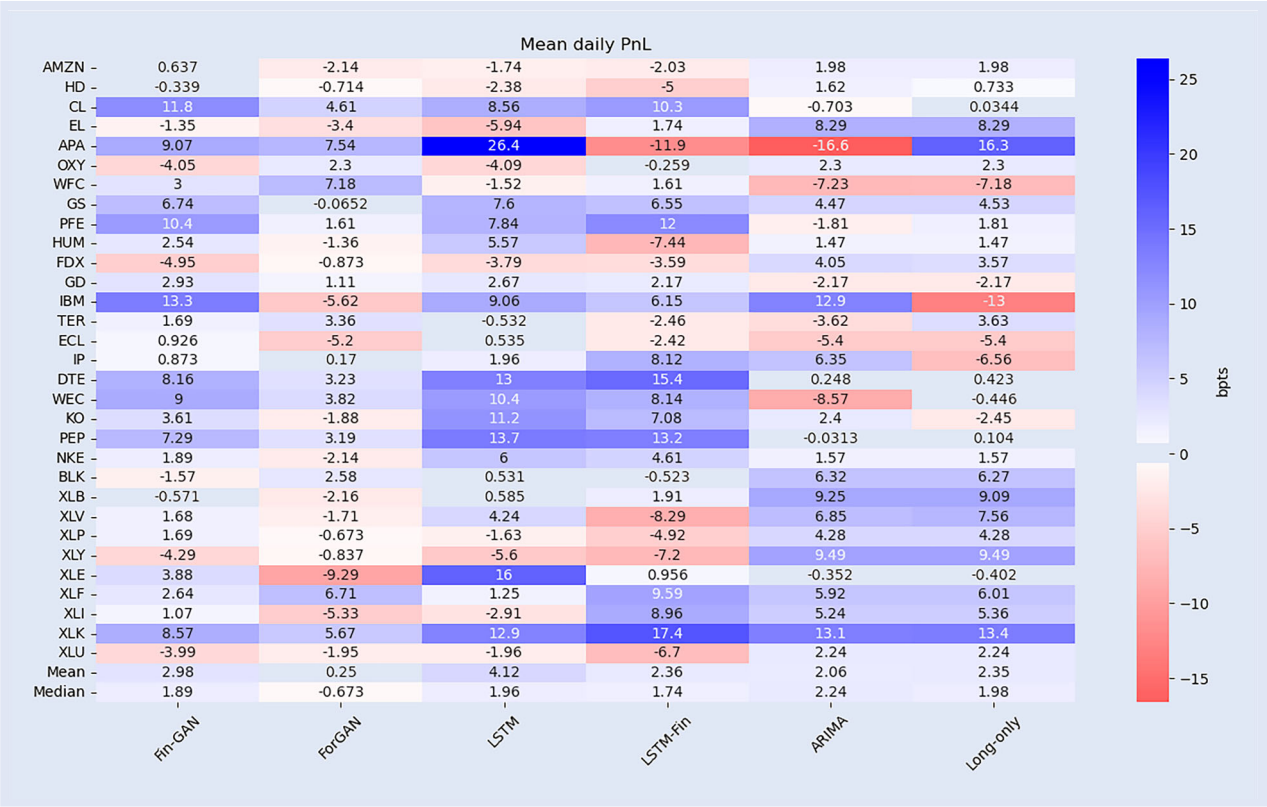


Figure 11. Mean daily PnL in basis points obtained by different methods.

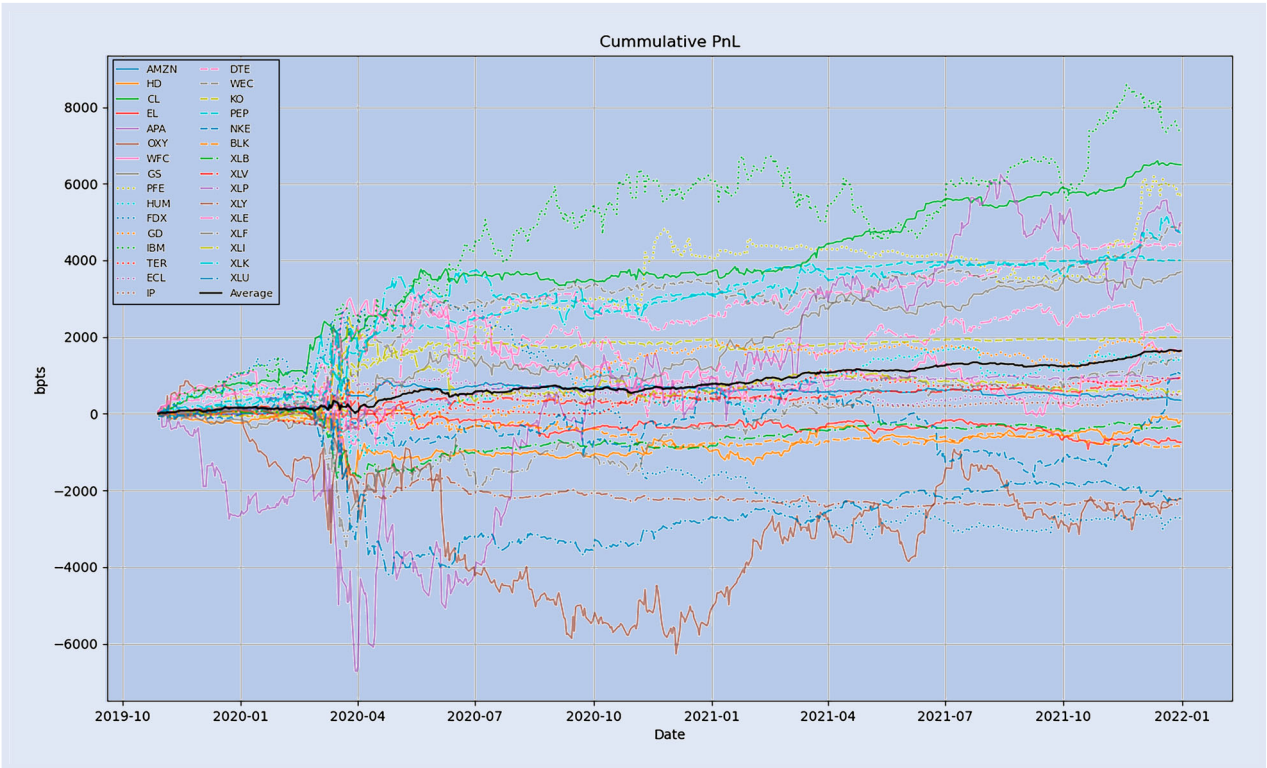


Figure 12. Cumulative PnL across tickers achieved by different loss function combinations of Fin-GAN. The portfolio PnL is the average PnL displayed in black, multiplied by the number of instruments. A comparison of the overall portfolio performance across the baselines (and the Fin-GAN loss function combinations) is shown in figure 13.

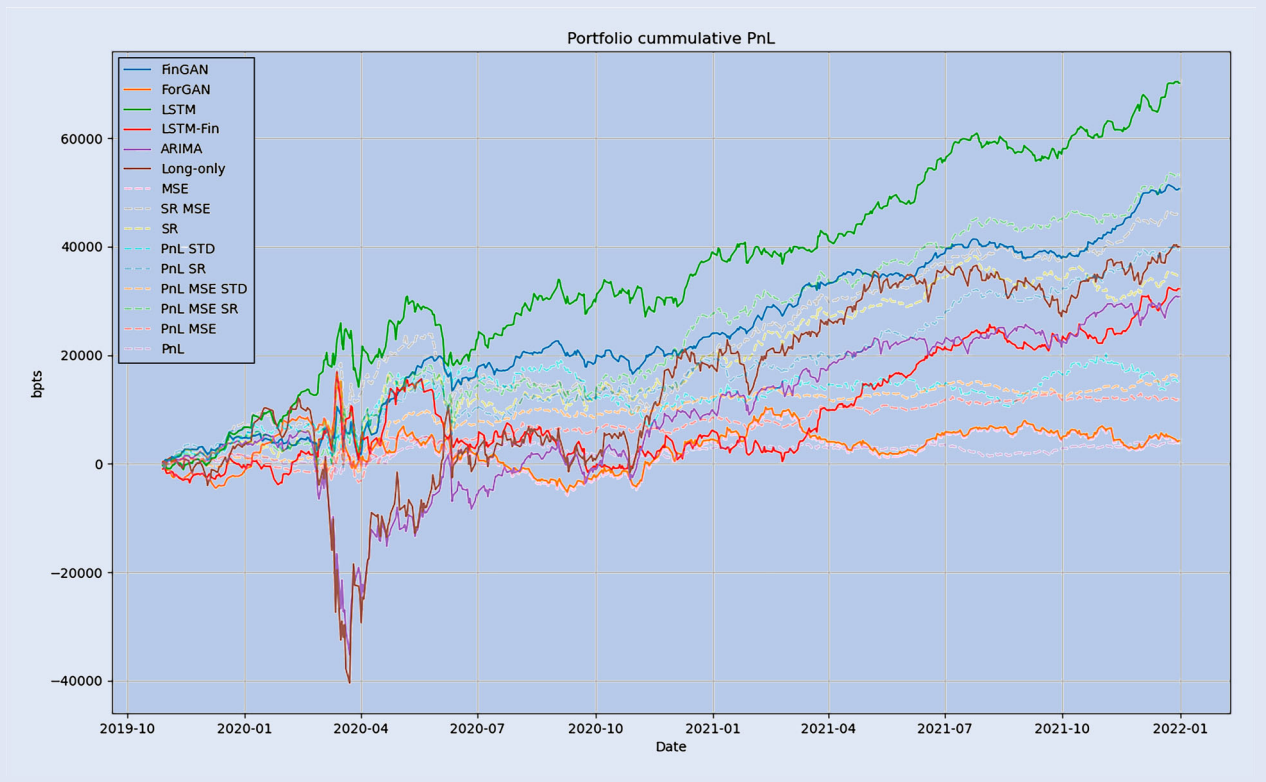


Figure 13. Portfolio cumulative PnL of different models. Dashed lines correspond to PnL paths generated by the appropriate Fin-GAN loss function combinations (including MSE alone).

The above experiments have demonstrated that the Fin-GAN approach outperforms, on average, ForGAN, LSTM, ARIMA, and long-only baselines. We now further investigate the behavior of Sharpe Ratios at the individual ticker level. The annualized Sharpe Ratio performance across the models and tickers is displayed in figure 10. We observe that the Fin-GAN model outperforms all other baselines, being able to achieve very competitive Sharpe Ratios, especially in light of the fact that we are dealing with single-instrument portfolios. Sharpe Ratios achieved by Fin-GAN are more stable than those of LSTM, and the only ticker with Sharpe Ratio below -1 is *XLY* (Consumer Discretionary). This is not surprising, as this sector ETF had a crash at the start of the Covid-19 pandemic, and was consistently growing during the time used for validation. On the data sets on which LSTM achieves Sharpe Ratios above 1 or 2, Fin-GAN does so as well, achieving similar Sharpe Ratios, or a group higher in the case of CL. Altogether, we observe clear benefits from using Fin-GAN when compared to the other methods, in terms of Sharpe Ratio performance. The breakdown of the Fin-GAN cumulative PnL performance by ticker is displayed in figure 12. We remark that the Covid pandemic had a different effect on different stocks, and that performance in March–June 2020 is the main cause of volatility in the attained Pnls.

In terms of the overall performance, Fin-GAN and LSTM outperform all the other methods under consideration. Cumulative PnL plots of the corresponding portfolios are shown in figure 13. The deep learning models recover from the Covid-19 shock much faster than ARIMA and long-only do, and we note that most of the volatility in the generated PnLs is stemming from the pandemic. In figure 13 we display the cumulative PnLs achieved by different Fin-GAN loss combinations, including MSE with BCE term only, which on average has a 99.7% correlation with ForGAN. It is evident that the overall performance is increased by using validation, rather than using the same loss combination from the start for every data set. The LSTM has a higher portfolio PnL, but the path is more volatile, especially from March 2020 until June 2020, resulting in a lower Sharpe Ratio than Fin-GAN.

Concerning the mean daily PnL performance displayed in figure 11, we note that ARIMA and the long-only approach have a tendency of producing mean daily PnLs on the same scale, while LSTM attains the highest PnLs on average. However, the PnLs achieved by LSTM fluctuate significantly from ticker to ticker. The mean daily PnLs achieved by the GAN approaches have a much lower standard deviation (4.85 for Fin-GAN, 4.00 for ForGAN) than LSTM (7.48), LSTM-Fin (7.53), ARIMA (6.26), and long-only (5.97), showcasing the benefits of being able to leverage the uncertainty estimates for developing a weighted strategy with dynamic sizing.

Next, we comment on the breakdown of the chosen loss function combinations in the Fin-GAN performance. As shown in figure 5, it can be beneficial to utilize the introduced loss function terms together, both for shifting distribution purposes, achieving a better approximation to the data distribution, and for alleviating mode collapse. We remark that the inclusion of the MSE term is very prominent, and that interestingly, the PnL and Sharpe Ratio combination was never chosen. The most common choice was Sharpe Ratio with MSE, in 29% of the cases. The breakdown of the number

Table 3. Number of chosen (highest SR on the validation set) Fin-GAN loss function term combinations across the data.

Combination	Count
PnL	3
SR	3
PnL & MSE	5
PnL & SR	0
SR & MSE	9
PnL & MSE & SR	4
PnL & STD	5
PnL & STD & MSE	2

of times each of the combinations was used is shown in table 3.

We compare the Sharpe Ratio performance of the different Fin-GAN loss function combinations on the test set, and show the results in figure 14. Due to the differences in validation and test set, sometimes a sub-optimal loss function combination is chosen during the validation stage. However, we note that performing validation in order to choose which terms to include in the training objective improves the Sharpe Ratio performance of the model.

As previously discussed, SR and the combination of PnL and STD (PnL & STD) terms convey the same information, but have different gradients, and may result in different forecasts. Hence, we investigate the average Pearson correlation of the obtained out-of-sample PnLs (test set) by different loss term combinations in figure 15. It is not surprising that there is a high correlation between combinations with the MSE term included, and the corresponding ones with the MSE term excluded. The correlation between the SR term and PnL & STD term is 33.5%, indicating that the two learn very different distributions, while once the MSE term is included, the correlation increases significantly.

An important finding is that the correlation between the MSE term with the BCE loss and the BCE loss is 99.7% on average, implying that ForGAN trained via the binary cross entropy loss is already aiming to produce outputs close to the real values. This behavior could be simply explained by the structure of the data and the task at hand: since there is a *true target* for each condition, that is, the empirical conditional distribution is point mass $p_{data}(x_{t+1}|x_t^{-L}) = \delta(x_{t+1})$ if L is large enough. Hence, it is not surprising that the forecast values close to the target would receive high scores from the discriminator, encouraging mode collapse in a classical GAN setup. This further supports the decision to use ForGAN as the baseline, its suitability for probabilistic forecasting, and the necessity to customize it to a financial setting.

8.2. *Universality*

We test for universality in our approach, in the spirit of Sirignano and Cont (2019). However, due to computational cost, we are only able to perform numerical experiments on a small universe of stocks and ETFs, the 31 previously used. We do not attempt to learn from the cross-asset correlations, but rather consider each time series individually. That is, we employ the single-asset model, where we combine (pool) the data from all 31 tickers. We test the performance of the model on all stocks used for training, and on four additional unseen stocks. The time periods for the training, validation,

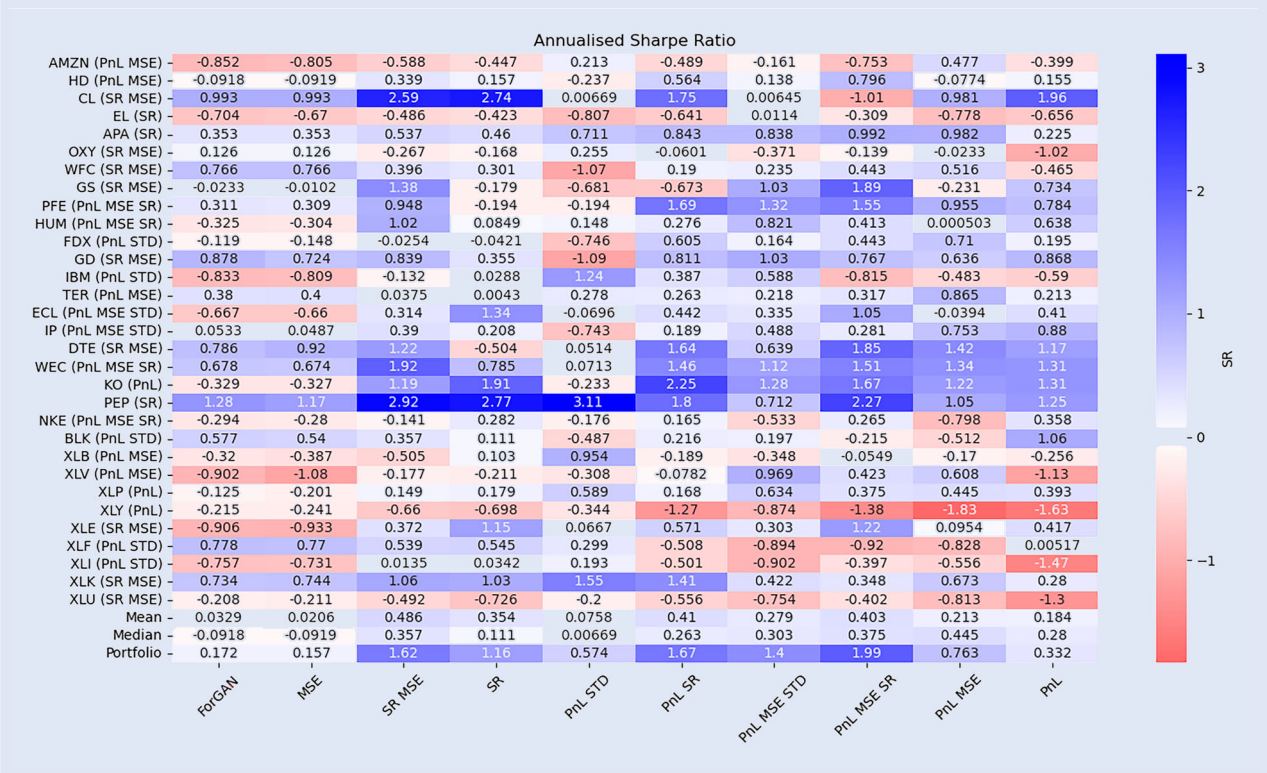


Figure 14. SR (annualized Sharpe Ratio) obtained by different loss combinations of Fin-GAN on the test set. The chosen loss combination (the combination achieving the highest Sharpe Ratio on the validation set) is reported in parentheses, for each ticker.

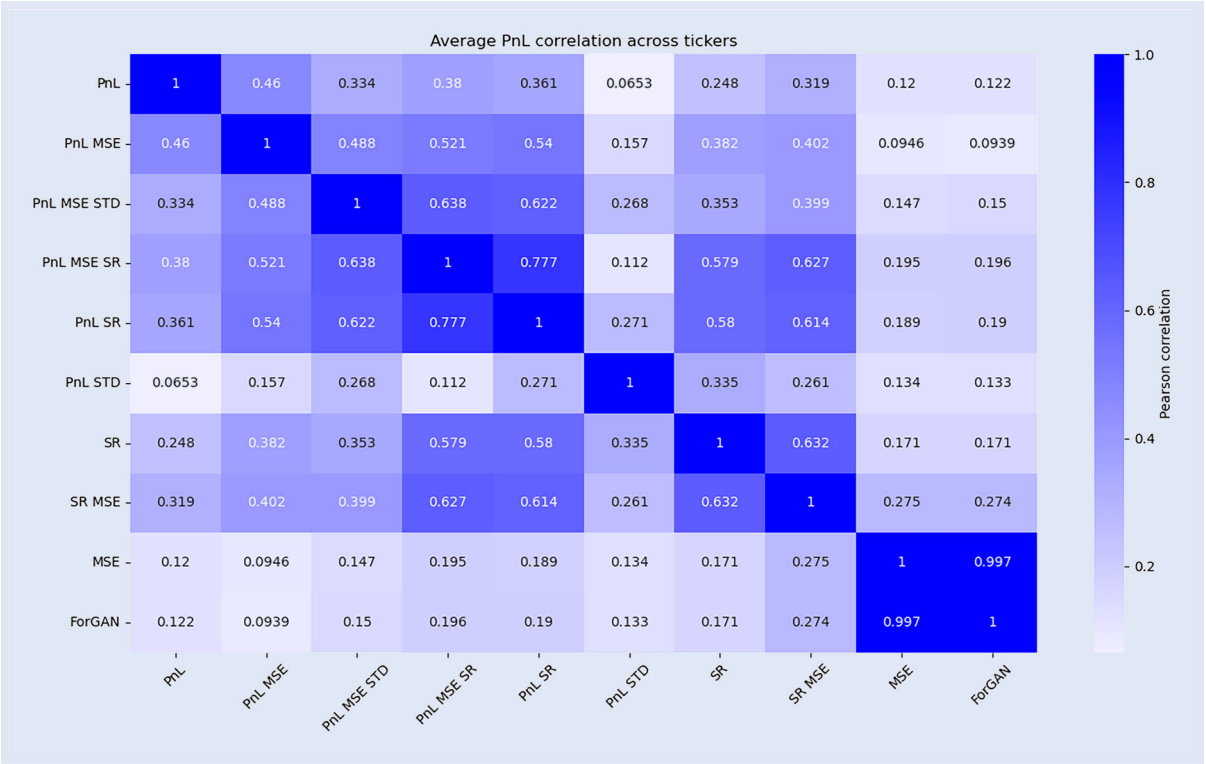


Figure 15. Mean out-of-sample (test) correlation of PnLs across different tickers of the Fin-GAN loss term combinations.

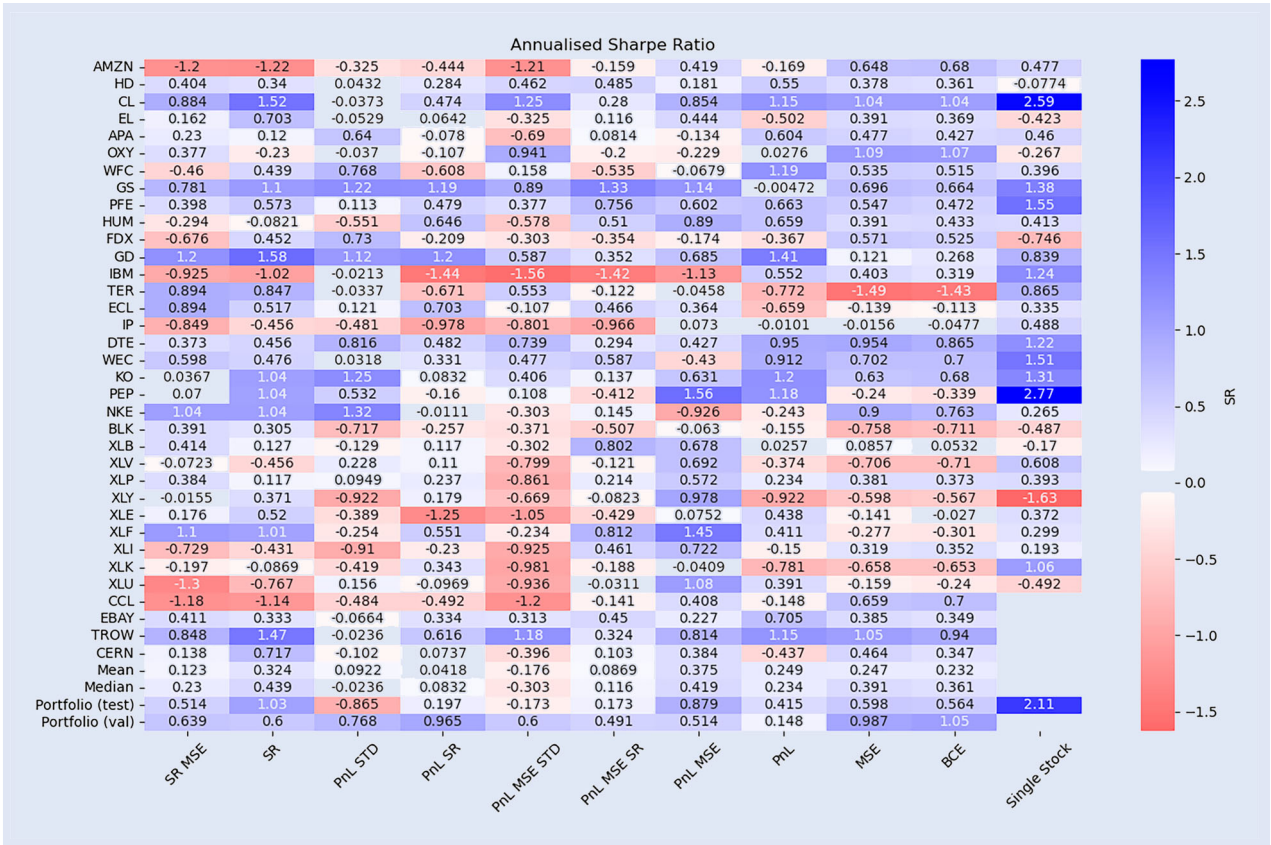


Figure 16. Summary of Sharpe Ratio performance for individual stocks in the universal model. Each column represents a different combination of the loss function terms. The *Single Stock* column shows the best Fin-GAN performance when trained on a particular stock/etf. CCL, EBAY, TROW and CERN have not been seen by the model during the training stage.

and test sets across different tickers, as well as the Fin-GAN methodology remain the same. A summary of the Sharpe Ratio performance of the small universal model is shown in the heatmap displayed in figure 16. The *Single Stock* column refers to the highest Sharpe Ratio achieved by the single-stock model when training on a particular ticker only. Stocks CCL, EBAY, TROW, and CERN have not been seen by the model during the training stage. We note that it is possible to achieve competitive Sharpe Ratios even on unseen stocks. For example, Sharpe Ratios achieved on THROW data are often above 1, despite the fact that the universal model has never seen the EBAY data. Figure 16 indicates that some of the stocks included in the training of the universal model benefit from the pooled training with other stocks, but not all. This is most visible for XLY, where the achieved Sharpe Ratios are either significantly less negative, or even positive. The overall performance not increasing could be due to our universe of stocks being small, and not having strong correlations. One would expect that cross-asset interactions are more informative when stocks belong to the same sector.

We repeat the same analysis using stocks from the Consumer (XLP) sector. We perform one set of numerical experiments with XLP data (raw returns) included in the training data, and another set of experiments without it. The Sharpe Ratio performance without the XLP data included is shown in the heatmap displayed in figure 17, while same performance when the XLP data is included is shown in figure 18. Stocks unseen by the model are SYY and TSN. Similarly to the previously discussed universal model, Sharpe Ratios achieved on the unseen data can be very competitive, see for example TSN. When XLP data is included, the model chosen by validation is PnL & SR, achieving a good overall portfolio SR of 1.43 on the test set. Although not all stocks have a positive Sharpe Ratio, this combination achieves good and even very good SRs, including the Sharpe Ratio of 1.55 on unseen TSN data. An important observation is that the Sharpe Ratio on XLP data, in this case, is 1.11, compared to 0.393 in the single-stock setting. Other ETFs might as well benefit from being trained alongside their constituents. When XLP data is removed from the pool of stocks used for training, the model of choice is SR. The portfolio Sharpe Ratio reduces to 1.18, but remains competitive. Comparing the heatmaps in figures 17 and 18, we conclude that there is a clear benefit from including the ETF data in the training process of Fin-GAN, but not of ForGAN. Training alongside the XLP data results in more stable Sharpe Ratios compared to the model without it.

9. Conclusion and future outlook

We have shown that GANs can be successfully employed for probabilistic time series forecasting in the context of financial data, where the directionality (sign) of the forecast is of main interest, particularly ahead of large price jumps. We introduced a novel economics-driven generator loss function, which includes suitably weighted Profit and Loss (PnL), standard deviation of the PnL, MSE, Sharpe Ratio-based loss

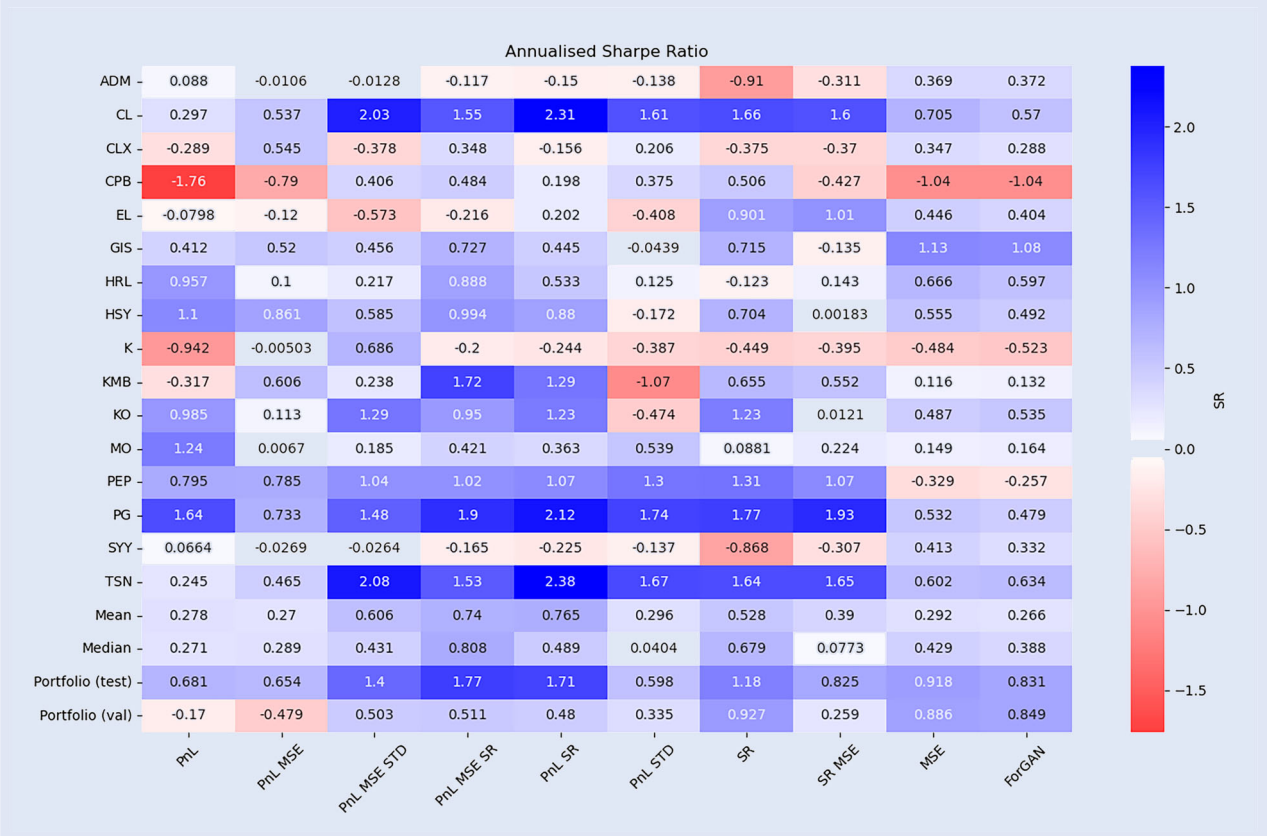


Figure 17. Summary of SR performance on stocks constituents of the XLP sector. Each column represents a different combination of loss function term. SYY and TSN have not been seen by the model during the training stage.

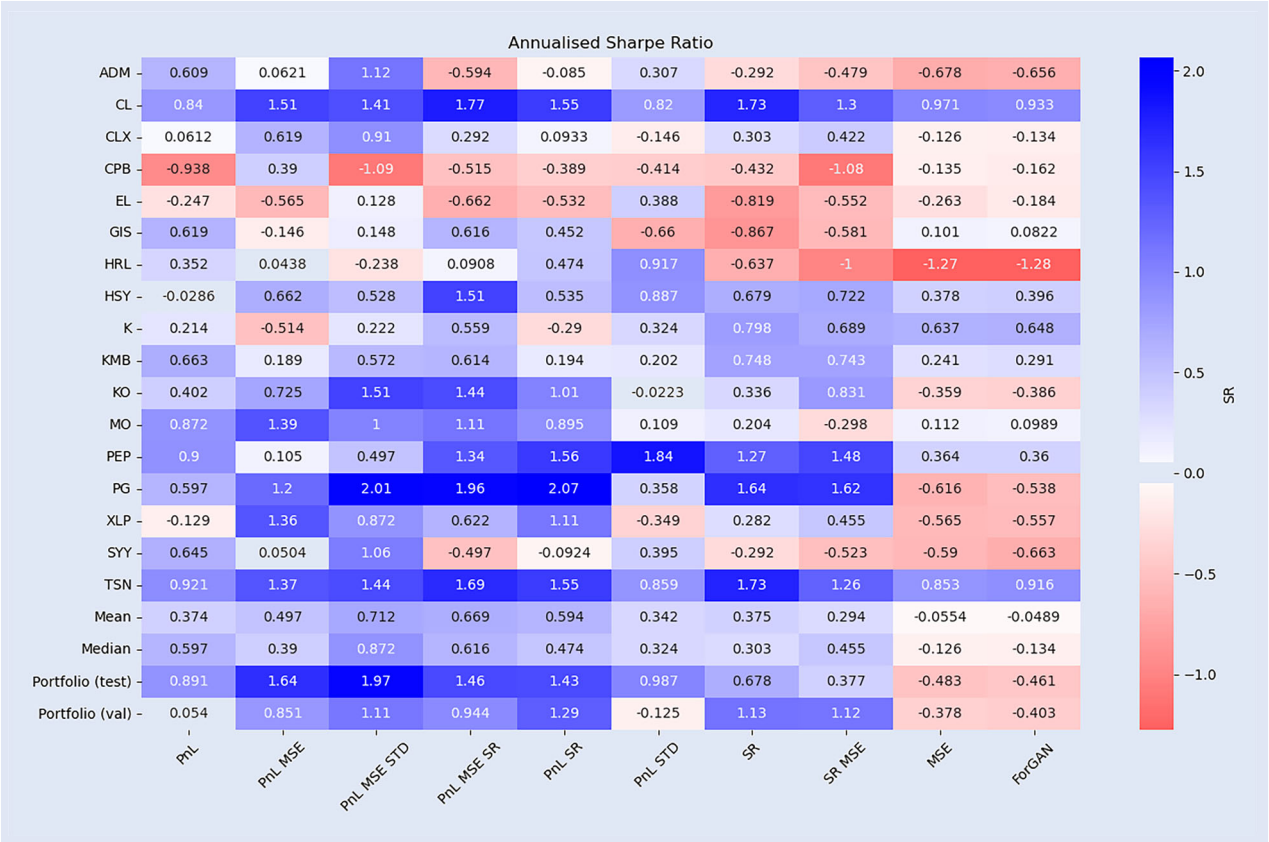


Figure 18. Summary of SR performance on the stocks belonging to the XLP sector, with XLP data included in the training data. Each column represents a different combination of loss function terms. SYY and TSN have not been seen by the model during the training stage.

function terms, rendering GANs more suitable for a classification task, and placing them into a supervised learning setting.

In a comprehensive set of numerical experiments, we compared our approach to a standard GAN model, LSTM, ARIMA and long-only strategies, and demonstrated superior performance of our model in terms of Sharpe Ratios achieved.

Furthermore, we considered a universal model in three settings: pooling the data from stocks belonging to different sectors and sector ETFs, considering one sector with the sector ETF included in the training data and with the sector ETF excluded from the training stage. Even though the universe of stocks was small, we note that good performance is possible even on unseen stocks. Furthermore, performance in the single-sector setting increased when the sector ETF was included in the training data.

Our proposed Fin-GAN methodology was shown to be able to significantly improve Sharpe Ratio performance, shift generated distributions, as well as help alleviate mode collapse issues, the latter of which is a standard challenge in many GAN-based approaches.

Our work raises a number of interesting future directions. It could be insightful to explore how the addition of the new loss function terms impacts path simulations, and how to leverage cross-asset interactions, learn from correlations and explore the notion of universality in a larger universe of stocks. Moving beyond equity data and exploring other classes of assets (such as options), potentially in a joint framework, is another interesting avenue for future work. Our proposed model and architecture could benefit from being trained on higher-frequency data, for eg, intraday minutely data. An additional area of interest would be to explore how one might combine the Fin-GAN loss with COT-GAN (Xu *et al.* 2020) and Time-GAN (Yoon *et al.* 2019). Exploring more complex architectures, a larger hyperparameter space, and longer training could further yield improved results.

Acknowledgments

We thank the anonymous referees, Justin Sirignano, and audiences from the Mathematical Institute and Department of Statistics at the University of Oxford, Oxford-Man Institute of Quantitative Finance, ETH Zurich, and Man AHL, whose feedback and comments have improved our work.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

Milena Vuletić's research is supported by BNP Paribas through the *EPSRC Centre for Doctoral Training in*

Mathematics of Random Systems: Analysis, Modelling and Simulation [ESPRC Grant EP/S023925/1].

Data availability

The data used is extracted from CRSP on Wharton Research Data Services and is available here.

Code availablility

The code is available on GitHub: https://github.com/milenavuletic/Fin-GAN.

References

Arjovsky, M. and Bottou, L., Towards principled methods for training generative adversarial networks. In *International Conference on Learning Representations*, 2017.

Arjovsky, M., Chintala, S. and Bottou, L., Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pp. 214–223, 2017 (PMLR).

Assefa, S.A., Dervovic, D., Mahfouz, M., Tillman, R.E., Reddy, P. and Veloso, M., Generating synthetic data in finance: Opportunities, challenges and pitfalls. In *Proceedings of the First ACM International Conference on AI in Finance*, pp. 1–8, 2020.

Bhatia, S., Jain, A. and Hooi, B., ExGAN: Adversarial generation of extreme samples. *Proc. AAAI Conf. Artif. Intell.*, May 2021, **35**(8), 6750–6758. doi:10.1609/aaai.v35i8.16834.

Box, G.E., Jenkins, G.M., Reinsel, G.C. and Ljung, G.M., *Time Series Analysis: Forecasting and Control*, 2015 (John Wiley & Sons: Hoboken, NJ).

Buehler, H., Gonon, L., Teichmann, J. and Wood, B., Deep hedging. *Quant. Finance*, 2019, **19**(8), 1271–1291.

Buehler, H., Horvath, B., Lyons, T., Perez Arribas, I. and Wood, B., Generating financial markets with signatures. Available at SSRN 3657366, 2020.

Chen, T. and Guestrin, C., Xgboost: A scalable tree boosting system. In *KDD*, 2016.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y., Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, October 2014 (Association for Computational Linguistics: Doha, Qatar).

Cont, R., Cucuringu, M., Xu, R. and Zhang, C., Tail-GAN: Nonparametric scenario generation for tail risk estimation. Papers 2203.01664, arXiv.org, March 2022,

Dorogush, A.V., Ershov, V. and Gulin, A., CatBoost: Gradient boosting with categorical features support. arXiv preprint arXiv:1810.11363, 2018.

Durall, R., Chatzimichailidis, A., Labus, P. and Keuper, J., Combating mode collapse in GAN training: An empirical analysis using hessian eigenvalues. In *VISIGRAPP*, pp. 211–218, January 2021. doi:10.5220/0010167902110218.

Galteri, L., Seidenari, L., Bertini, M., Uricchio, T. and Del Bimbo, A., Fast video quality enhancement using GANs. In *Proceedings of the 27th ACM International Conference*, pp. 1065–1067, October 2019. doi:10.1145/3343031.3350592.

Glorot, X. and Bengio, Y., Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010 (JMLR Workshop and Conference Proceedings).

Goodfellow, I.J., NIPS 2016 tutorial: Generative adversarial networks. ArXiv, abs/1701.00160, 2017.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., Generative adversarial nets. In *Advances in Neural Information Processing Systems*, Vol. 27, edited by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence and K.Q. Weinberger, 2014 (Curran Associates, Inc.: Montreal).

Goodfellow, I., Bengio, Y. and Courville, A., *Deep Learning*, 2016 (Springer US: New York).

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. and Courville, A.C., Improved training of wasserstein GANs. In *Advances in Neural Information Processing Systems*, Vol. 30, edited by I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, 2017 (Cornell University Library: Ithaca, NY).

Guresen, E., Kayakutlu, G. and Daim, T.U., Using artificial neural network models in stock market index prediction. *Expert Syst. Appl.*, 2011, **38**(8), 10389–10397.

He, K., Zhang, X., Ren, S. and Sun, J., Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015.

Hinton, G., Srivastava, N. and Swersky, K., Neural networks for machine learning, lecture 6. Coursera, 2012.

Hochreiter, S. and Schmidhuber, J., Long short-term memory. *Neural Comput.*, December 1997, **9**, 1735–1780. doi:10.1162/neco.1997.9.8.1735

Huszár, F., How (not) to train your generative model: Scheduled sampling, likelihood, adversary? ArXiv, abs/1511.05101, 2015.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q. and Liu, T.-Y., Lightgbm: A highly efficient gradient boosting decision tree. In *NeurIPS*, 2017.

Koochali, A., Schichtel, P., Dengel, A. and Ahmed, S., Probabilistic forecasting of sensory data with generative adversarial networks–ForGAN. *IEEE Access*, 2019, **7**, 63868–63880.

Koshiyama, A., Firoozye, N. and Treleaven, P., Generative adversarial networks for financial trading strategies fine-tuning and combination. *Quant. Finance*, September 2020, **21**, 1–17. doi:10.1080/14697688.2020.1790635.

Leangarun, T., Tangamchit, P. and Thajchayapong, S., Stock price manipulation detection using generative adversarial networks. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 2104–2111, 2018. doi:10.1109/SSCI.2018.8628777.

Li, J., Wang, X., Lin, Y., Sinha, A. and Wellman, M., Generating realistic stock market order streams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34, pp. 727–734, 2020.

Lucchese, L., Pakkanen, M. and Veraart, A., The short-term predictability of returns in order book markets: A deep learning perspective, 2022.

Mariani, G., Zhu, Y., Li, J., Scheidegger, F., Istrate, R., Bekas, C. and Malossi, A.C.I., PAGAN: Portfolio analysis with generative adversarial networks. Papers 1909.10578, arXiv.org, September 2019.

Medsker, L.R. and Jain, L., Recurrent neural networks. *Des. Appl.*, 2001, **5**, 64–67.

Mirza, M. and Osindero, S., Conditional generative adversarial nets. ArXiv, abs/1411.1784, 2014.

Mustafa, M., Bard, D., Bhimji, W., Lukić, Z., Al-Rfou, R. and Kratochvil, J.M., CosmoGAN: Creating high-fidelity weak lensing convergence maps using generative adversarial networks. *Comput. Astrophys. Cosmol.*, May 2019, **6**(1), 1–13. doi:10.1186/s40668-019-0029-9.

Nowozin, S., Cseke, B. and Tomioka, R., f-GAN: Training generative neural samplers using variational divergence minimization. *Adv. Neural Inf. Process. Syst.*, 2016, **29**.

Prenzel, F., Cont, R., Cucuringu, M. and Kochems, J., Dynamic calibration of order flow models with generative adversarial networks. In *3rd ACM International Conference on AI in Finance*, pp. 446–453, 2022.

Refenes, A.N., Zapranis, A. and Francis, G., Stock performance modeling using neural networks: A comparative study with regression models. *Neural Netw.*, 1994, **7**(2), 375–388.

Roberts, S., Osborne, M., Ebden, M., Reece, S., Gibson, N. and Aigrain, S., Gaussian processes for time-series modelling. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, 2013, **371**(1984), 20110550. doi:10.1098/rsta.2011.0550.

Romero, R.A.C., *Generative Adversarial Network for Stock Market Price Prediction*, 2018 (CS230: Deep Learning, Winter 2018, Stanford University: Stanford, CA).

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X. and Chen, X., Improved techniques for training GANs. In *Advances in Neural Information Processing Systems*, Vol. 29, edited by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon and R. Garnett, 2016 (Curran Associates, Inc.: Barcelona, Spain).

Sirignano, J. and Cont, R., Universal features of price formation in financial markets: Perspectives from deep learning. *Quant. Finance*, 2019, **19**(9), 1449–1459. doi:10.1080/14697688.2019.1622295.

Sirignano, J.A., Deep learning for limit order books. *Quant. Finance*, 2019, **19**(4), 549–570.

Takahashi, S., Chen, Y. and Tanaka-Ishii, K., Modeling financial time-series with generative adversarial networks. *Phys. A Stat. Mech. Appl.*, 2019, **527**, 121261.

Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M. and Iosifidis, A., Forecasting stock prices from the limit order book using convolutional neural networks. In *2017 IEEE 19th Conference on Business Informatics (CBI)*, Vol. 1, pp. 7–12, 2017a (IEEE).

Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M. and Iosifidis, A., Using deep learning to detect price change indications in financial markets. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pp. 2511–2515, 2017b (IEEE).

Tsay, R.S., *Analysis of Financial Time Series*, 2005 (John Wiley & Sons: Hoboken, NJ).

Vuletić, M. and Cont, R., VolGAN: A generative model for arbitrage-free implied volatility surfaces. Available at SSRN, 2023.

Wallbridge, J., Transformers for limit order books. arXiv preprint arXiv:2003.00130, 2020.

Wang, X., Yu, K., Wu, S., Gu, J., Liu, Y., Dong, C., Qiao, Y. and Loy, C.C., ESRGAN: Enhanced super-resolution generative adversarial networks. In *Computer Vision – ECCV 2018 Workshops*, edited by L. Leal-Taixé and S. Roth, pp. 63–79, 2019 (Springer International Publishing: Cham).

Wiese, M., Knobloch, R., Korn, R. and Kretschmer, P., Quant GANs: Deep generation of financial time series. *Quant. Finance*, April 2020, **20**(9), 1419–1440. doi:10.1080/14697688.2020.1730426.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C. and Philip, S.Y., A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.*, 2020, **32**(1), 4–24.

Wu, Q., Brinton, C.G., Zhang, Z., Pizzoferrato, A., Liu, Z. and Cucuringu, M., Equity2Vec: End-to-end deep learning framework for cross-sectional asset pricing. In *Proceedings of the Second ACM International Conference on AI in Finance*, ICAIF '21, New York, NY, USA, 2022 (Association for Computing Machinery). doi:10.1145/3490354.3494409.

Wu, Q., Li, J., Liu, Z., Li, Y. and Cucuringu, M., Symphony in the latent space: Provably integrating high-dimensional techniques with non-linear machine learning models. In *Proceedings of the 2023 AAAI Conference on Artificial Intelligence*, 2023.

Xu, T., Wenliang, L.K., Munn, M. and Acciaio, B., COT-GAN: Generating sequential data via causal optimal transport. *Adv. Neural Inf. Process. Syst.*, 2020, **33**, 8798–8809.

Yoon, J., Jarrett, D. and Van der Schaar, M., Time-series generative adversarial networks. *Adv. Neural Inf. Process. Syst.*, 2019, **32**.

Zhang, Z. and Zohren, S., Multi-horizon forecasting for limit order books: Novel deep learning approaches and hardware acceleration using intelligent processing units. arXiv preprint arXiv:2105.10430, 2021.

Zhang, K., Zhong, G., Dong, J., Wang, S. and Wang, Y., Stock market prediction based on generative adversarial networks. *Procedia Comput. Sci.*, 2019a, **147**, 400–406. doi:10.1016/j.procs.2019.01.256.

Zhang, Z., Zohren, S. and Roberts, S., Deeplob: Deep convolutional neural networks for limit order books. *IEEE Trans. Signal Process.*, 2019b, **67**(11), 3001–3012.

Zhang, H., Sindagi, V. and Patel, V.M., Image de-raining using a conditional generative adversarial network. *IEEE. Trans. Circ. Syst. Video Technol.*, 2020, **30**(11), 3943–3956. doi10.1109/TCSVT.2019.2920407.

Zhou, X., Pan, Z., Hu, G., Tang, S. and Zhao, C., Stock market prediction on high-frequency data using generative adversarial nets. *Math. Probl. Eng.*, April 2018, **2018**, 1–11. doi:10.1155/2018/4907423.