Modeling Financial Time-Series with Generative Adversarial Networks

Harsh Bhatt
Associate Quant Researcher at Quant Club

Highlights

- Generative adversarial networks for financial time-series modeling are proposed.
- The model learns and generates the time-series in a data-driven manner.
- The model is able to reproduce stylized facts of the financial time-series.

Article Info
Keywords:
Financial market
Deep learning
Generative adversarial networks

Abstract
Financial time-series modeling is a challenging problem as it retains various complex statistical properties, and the mechanism behind the process remains largely unrevealed. In this paper, a deep neural networks-based approach, Generative Adversarial Networks (GANs) for financial time-series modeling, is presented. GANs learn the properties of data and generate realistic data in a data-driven manner.GAN model produces a time-series that recovers the statistical properties of financial time-series, such as the Linear unpredictability, Heavy-tailed price return distribution, Volatility clustering, Coarse-fine volatility correlation, Gain/loss asymmetry.

1 Introduction

Imagine trying to predict stock prices——it is like trying to forecast the weather, but with money on the line. That's where this innovative technology called *Fin-GAN* comes in. Think of it as a super-smart crystal ball that doesn't just tell you if a stock might go up or down, but also how confident it is about its prediction. This research introduces *Fin-GAN*, an innovative approach to financial time series forecasting that addresses key limitations in traditional methods. Combining Generative Adversarial Networks (GANs) with specialized economics-driven loss functions, it generates probabilistic forecasts, moving beyond the simple point estimates common in classical approaches.

The model's architecture builds on *ForGAN*, integrating recurrent neural networks for time series analysis. What sets *Fin-GAN* apart is its novel loss function, specifically designed for financial applications, which improves Sharpe Ratio performance and helps prevent mode collapse—a common issue in GAN training. Tested *Fin-GAN* on a diverse dataset of 22 stocks and 9 sector ETFs, demonstrating its versatility across different market sectors. The results show superior performance compared to established methods like LSTM and ARIMA, particularly in forecasting daily stock ETF-excess returns and raw returns.

Classical GAN

A simple way to understand GANs is by thinking of it like a teacher-student scenario. The "generator" (G) is like a student trying to create realistic drawings, while the "discriminator" (D) is the teacher who has to decide whether the drawing is genuine or made by the student. The student's goal is to trick the teacher into thinking their drawings are real, while the teacher's job is to tell the difference between real and fake drawings. The student (G) improves by getting feedback from the teacher (D). Both G and D are neural networks, and they learn through a back-and-forth process. The student gets some random input, like noise, and tries to generate a new drawing. The teacher looks at the drawing and decides if it's real or fake, then gives feedback. Over time, the student gets better at making convincing drawings, and the teacher gets better at spotting the fakes. 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.This process of training both models together is called adversarial learning, and it works well with neural networks through a technique called backpropagation.

Definition 3.1: The generator $G(z;\theta_g)$ is like a machine that creates data. It takes random noise z, which comes from a known probability distribution (usually something simple like a normal distribution), and transforms it into data that appears real. This process is performed through a neural network. The data created by the generator follows its own probability distribution, which we denote as p_g.

Definition 3.2: The discriminator $D(x;\theta_d)$ is another neural network with a distinct role. It takes in data x and attempts to determine whether this data is "real" (from the actual dataset) or "fake" (produced by the generator). The discriminator outputs a value between 0 and 1, representing the probability that the input data is real.

The discriminator D is trained to accurately distinguish between real data (from p_{data}) and fake data generated by the generator G (from p_g). Its objective is to correctly label real data as "real" and fake data as "fake," effectively maximizing its classification accuracy. Conversely, the generator G is trained to deceive the discriminator by producing data so realistic that D cannot reliably differentiate it from real data. To achieve this, the generator either minimizes the discriminator's ability to correctly identify its outputs as fake or maximizes the likelihood of the discriminator mistakenly labeling fake data as real. In Generative Adversarial Networks (GANs), the discriminator D and generator G optimize separate cost functions:

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

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

Using cross-entropy loss for the generator G provides stronger gradients, enhancing the learning process, especially when the discriminator D initially performs well. However, mode collapse can occur when G outputs repetitive data, leading to weak feedback from D. Therefore, balancing the updates for D and G is crucial to prevent mode collapse and maintain effective training dynamics.

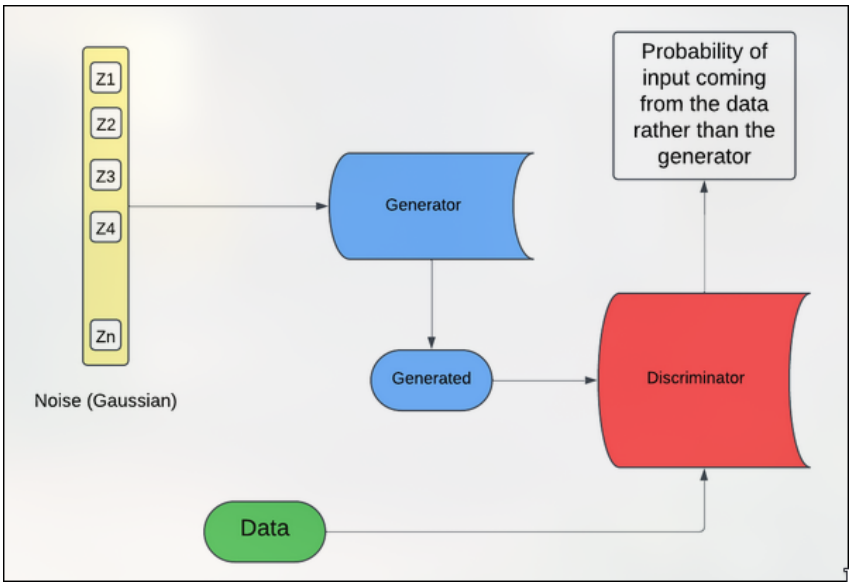

Figure 1: An example of standard GAN

ForGAN

ForGAN is a type of conditional GAN designed for forecasting time series data in a probabilistic way, introduced by Koochali et al. (2019). Unlike standard models that make single-point predictions, ForGAN can generate a range of possible outcomes for future data points, taking into account the past values of the series. For example, it predicts the value of a time series at time t+1 based on the previous L time points, like xt, xt-1, ..., xt-(L-1). This approach allows ForGAN to give more accurate predictions and uncertainty estimates, making it useful in areas like finance. To handle time series data, ForGAN uses a recurrent neural network (RNN) layer in both the generator and the discriminator. The architecture includes either a Gated Recurrent Unit (GRU) or a Long Short-Term Memory (LSTM) cell. In this case, the authors chose LSTM cells because they are commonly used for finance-related tasks. Similar models have been successfully used for predicting stock market trends, like in Zhang et al. (2019a). The diagrams in figure 3 illustrate the ForGAN architecture using LSTM cells.

Conditional GANs

In regular GANs, the generator creates data from a single distribution, but in conditional GANs, we assume that the data comes from a distribution based on a specific condition, like a label or some other information. The key difference in conditional GANs is that both the generator and discriminator receive additional information — the condition — along with the usual inputs. For example, if we want to generate pictures of cats and dogs, the

generator would take in both random noise and a label (e.g., a "cat" label) as inputs. It would then try to create an image of a cat. The discriminator, on the other hand, would take both real and generated images of cats (along with the label) and try to determine which one is real. The condition (often a label, like "cat" or "dog") helps the generator focus on creating specific types of images, and the discriminator uses it to evaluate whether the generated image matches the expected label. This setup makes cGANs more flexible and useful when we want to generate specific types of data, like images of certain animals or objects.

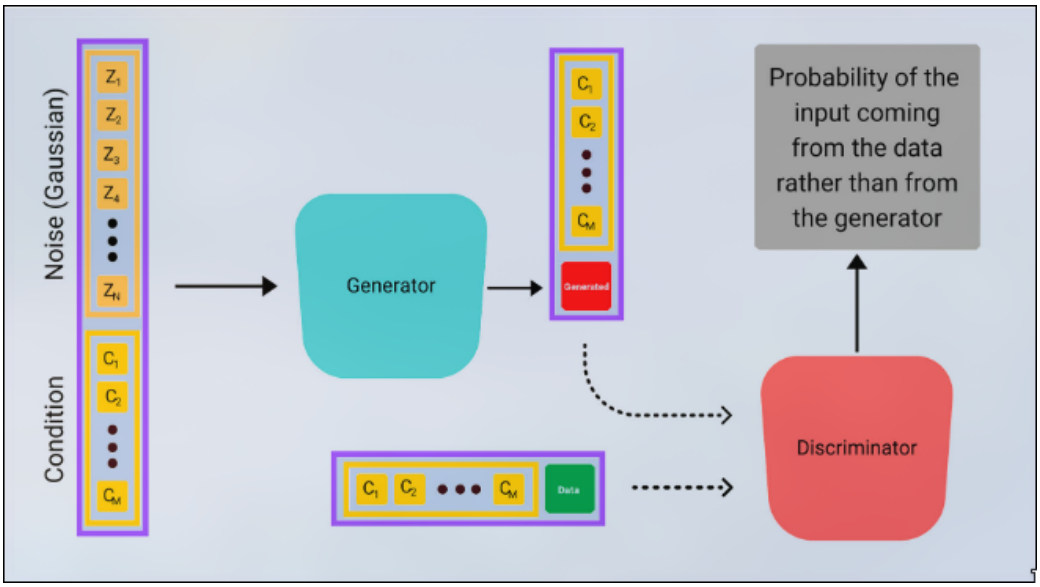

Figure 2: Conditional GAN

2 Performance Measures

When forecasting time series data, we often use performance metrics like Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE) to evaluate the accuracy of our predictions. Suppose the real value we want to estimate is x, and the predicted value is $\hat{x}$. This prediction could be a single output from a Generative Adversarial Network (GAN) based on random input, or it could represent the average or most likely prediction from the generated data. In both cases, the goal is to minimize the error between the real and predicted values. Lower MAE and RMSE values indicate more accurate forecasts.

2.1 Mean Absolute Error (MAE)

The Mean Absolute Error measures the average magnitude of the errors in a set of predictions, without considering their direction. It is defined as:

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

2.2 Root Mean Squared Error (RMSE)

The Root Mean Squared Error measures the square root of the average of squared differences between prediction and actual observation. It is defined as:

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

Using cross-entropy loss for the generator G provides stronger gradients, enhancing the learning process, especially when the discriminator D initially performs well. However, mode collapse can occur when G outputs repetitive data, leading to weak feedback from D. Therefore, balancing the updates for D and G is crucial to prevent mode collapse and maintain effective training dynamics.

2.3 Forecasting Return

When dealing with financial data, traditional metrics like Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) don't fully capture the nuances of real-world trading scenarios. In finance, the primary goal of forecasting is to decide whether to buy or sell an asset. This makes Profit and Loss (PnL) a more practical performance metric. Here's why: Even if MAE or RMSE are low, a model can still fail to predict the correct direction of significant price movements. Such missteps can result in substantial financial losses. To address this, we focus on PnL as a key measure of success. At any given time, we forecast the (ETF-excess) returns of an asset and place trades based on the prediction. If the forecast is positive, we buy the stock and sell the related ETF; if negative, we do the opposite. These trades create a time series of PnL values. To ensure a successful trading strategy, the PnL time series should show consistent profitability (a high average PnL) with minimal fluctuations (low variance). A higher average PnL means greater profitability, while lower variance indicates reduced risk. To combine these goals, we use the annualized Sharpe Ratio as our primary performance metric. The Sharpe Ratio is widely used in portfolio management to evaluate the risk-adjusted returns of trading strategies.

Analysis Components

- **Stocks**: We use market-excess returns (the return of the stock minus the return of its sector ETF).
- **ETFs**: We work with their raw returns directly.

Assuming that asset A has price $S_A(t)$ at time t, its return at time $t+1$ is defined as

$$r_t^A = \log\left(\frac{S_A(t+1)}{S_A(t)}\right)$$

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

$$\text{ret}_t^A = r_t^A - r_I(t)$$

If we predict that asset A will have a higher return than its corresponding ETF I, the strategy involves buying one share of stock A (going long) and simultaneously selling one share of the ETF I (going short). Conversely, if the forecast indicates that I will outperform

A, we reverse the positions—selling stock A and buying ETF I. This strategy generates a profit or loss (PnL) based on the correctness of the forecasted excess return, denoted as $\hat{re}_t^A$. At any given time $t+1$, the PnL is calculated by multiplying the actual excess return re_{t+1}^A by the direction of the trade, which is determined by the sign of the forecasted excess return $\hat{re}_{t+1}^A$.

2.3.1 Profit and Loss (PnL)

The formula for PnL at time $t+1$ is as follows:

$$\text{PnL}_{t+1} = \text{sign}\left(\hat{re}_{t+1}^A\right) \cdot re_{t+1}^A$$

The sign function determines the trade direction:

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

To simplify the analysis and allow for easier comparisons across strategies, the Profit and Loss (PnL) is expressed in basis points (1 basis point = 0.01%). Additionally, the PnL is normalized by dividing it by the total number of trades placed, which is equivalent to the size of the validation or test set. This normalized metric is referred to as the PnL per trade (pPnL), providing a standardized measure of the strategy's performance.

If there are n excess returns $re_1^A, re_2^A, \ldots, re_n^A$ that we aim to forecast, the pPnL is calculated as the average of the PnL values, scaled by a factor of 10,000 to convert it into basis points. This calculation only considers the directionality of the forecast (whether it is positive or negative) and the actual excess return, without accounting for the uncertainty in the forecast.

2.3.2 PnL per Trade (pPnL)

$$\text{pPnL} = \frac{10,000}{n} \sum_{i=1}^{n} \text{sign}\left(\hat{re}_i^A\right) \cdot re_i^A$$

The previously defined pPnL metric only considers point forecasts and ignores the uncertainty associated with predictions. To improve this, we leverage full probability forecasts to adjust for the confidence of each prediction. For predictions with higher confidence, we assign greater weights, while lower weights are given to less certain predictions. This is achieved by estimating the probabilities of the forecasted excess return $\hat{re}_i^A$ being positive or negative using $B = 1000$ noise samples under the same conditions. The probabilities for the forecast being positive (p_u^i) or negative (p_d^i) are calculated as the proportion of noise samples that result in positive or negative forecasts. These probabilities are then used to create a weighted PnL (wPnL), which adjusts for forecast confidence.

2.3.3 Probabilities and Weighted PnL

The formulas for calculating probabilities and weighted PnL are as follows:

$$p_u^i = \frac{1}{B} \sum_{j=1}^{B} \mathbf{1}\left(\hat{re}_i^A(z_j) \geq 0\right), \quad p_d^i = \frac{1}{B} \sum_{j=1}^{B} \mathbf{1}\left(\hat{re}_i^A(z_j) < 0\right)$$

Weighted PnL per Trade

$$\text{wPnL}_i^2 = 10,000 \cdot (p_u^i - p_d^i) \cdot re_i^A$$

Where:

- p_u^i: Probability of the excess return forecast being positive.
- p_d^i: Probability of the excess return forecast being negative.
- re_i^A: Actual excess return of asset A for the i-th trade.

For daily data, where there are two returns per day (e.g., open-to-close and close-to-open), the daily weighted PnL is calculated as:

$$\text{wPnL}_i = \frac{\text{wPnL}_{2i}^2 + \text{wPnL}_{2i+1}^2}{2}, \quad i = 1, \ldots, \left\lfloor \frac{n}{2} \right\rfloor$$

A higher confidence in the forecast results in a larger PnL contribution. Greater uncertainty reduces the potential loss by penalizing less confident predictions. The PnL is multiplied by 10,000 to express the values in basis points.

2.3.4 Sharpe Ratio

In financial forecasting, achieving symmetric distributions in the learned models ensures that the weighted Profit and Loss (wPnL) approaches zero. To evaluate the performance of a trading strategy, we use the annualized Sharpe Ratio as the primary metric. This metric measures the risk-adjusted return of the strategy by comparing the average PnL to its variability (standard deviation). For models that allow weighted strategies based on forecast confidence, the annualized Sharpe Ratio is calculated using the formula:

$$\text{SR}_w = \sqrt{252} \cdot \frac{\text{PnL}_w}{\sqrt{\frac{1}{\lfloor \frac{n}{2} \rfloor} \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} (\text{wPnL}_i - \text{PnL}_w)^2}}$$

For models that only provide point forecasts (without probabilities), the PnL per trade (PnL_i) is calculated directly using the sign of the forecast and the actual excess return. The corresponding Sharpe Ratio is:

$$\text{PnL}_i = 10,000 \cdot \left(\text{sign}\left(\hat{re}_{2i}^A\right) \cdot re_{2i}^A + \text{sign}\left(\hat{re}_{2i+1}^A\right) \cdot re_{2i+1}^A \right)$$

$$\text{SR} = \sqrt{252} \cdot \frac{\text{PnL}_m}{\sqrt{\frac{1}{\lfloor \frac{n}{2} \rfloor} \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} (\text{PnL}_i - \text{PnL}_m)^2}}$$

Interpretation of Sharpe Ratios

- **Above 1**: Good performance.
- **Above 2**: Very good performance.
- **Above 3**: Excellent performance.

When reporting the PnL, we typically use the mean daily PnL (PnL_w or PnL_m, depending on the model).

Note: Transaction costs are not considered in these calculations, meaning the reported metrics represent idealized results. In practice, including transaction costs could slightly reduce the performance metrics.

2.4 Fin-GAN

The Fin-GAN introduces a novel, economics-driven loss function to improve financial time series forecasting. Unlike traditional methods focused on minimizing pointwise errors (like Mean Squared Error), this approach emphasizes probabilistic forecasts and classification accuracy, particularly the sign of the forecast. Correctly identifying whether returns are positive or negative—especially during significant price movements—takes precedence over predicting values close to realized prices. The primary goal of this loss function is to enhance the generator's ability to classify the sign of returns correctly, ensuring accurate decision-making in financial trading. The loss function aims to address key challenges in financial forecasting by:

- Shifting the generated forecasts in the correct direction (towards accurate signs).
- Adapting GANs for classification tasks where the sign matters most.
- Improving generator learning, particularly in weak gradient cases.
- Preventing mode collapse by introducing a more complex generator loss surface, encouraging convergence to sharp minima.
- Regularizing the model for better generalization.

This innovative approach builds on ideas from previous works like W-GAN-GP (Lipschitz constraints) and Tail-GAN (portfolio tail risk estimation), while specifically tailoring the loss to financial forecasting needs. The Fin-GAN loss function integrates three terms—PnL-based, MSE-based, and Sharpe Ratio-based—to train the generator effectively. For simplicity, we focus on the PnL term, which incorporates the economic importance of accurate sign predictions.

Generator Forecasting

The generator aims to forecast values $\hat{x}^i$ for $i = 1, \ldots, n_{\text{batch}}$, based on:

- Noise sample z.
- Conditioning on the previous L values of the time series (x_i^{-L}).

The forecast is given by:

$$\hat{x}^i = G(z, x_i^{-L})$$

Incorporating Profit and Loss (PnL) into the Generator's Objective

To incorporate the profit and loss (PnL) into the generator's objective, the standard PnL function (which uses the non-differentiable sign function) is replaced with a smooth approximation, denoted PnL*. This makes it differentiable and suitable for backpropagation.

The approximated PnL term is defined as:

$$\text{PnL}^*(x, \hat{x}) = \frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} \text{PnL}_a^*(x_i, \hat{x}_i)$$

Where PnL_a^* for a single forecast is approximated as:

$$\mathrm{PnL}_a^*(x_i, \hat{x}_i) = \tanh(k_{\tanh} \cdot \hat{x}_i) \cdot x_i$$

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 dataset is shuffled at the start of every epoch. We use $k_{\tanh} = 100$.

Mean Squared Error (MSE) Term

While the primary focus is on predicting the correct sign of the forecast, it is also important for the predicted value ($\hat{x}^i$) to be close to the actual value (x_i). To enforce this, the Mean Squared Error (MSE) term is included in the loss function. This term penalizes large deviations between the forecast and the actual values, encouraging the generator to produce realistic forecasts.

The MSE term is defined as:

$$\mathrm{MSE}(x, \hat{x}) = \frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} (x_i - \hat{x}_i)^2$$

Sharpe Ratio (SR) Term

A key metric in financial performance, measuring the risk-adjusted return. To align the generator's training with this goal, a Sharpe Ratio-based term (SR^*) is introduced in the loss function. This term encourages the generator to maximize the annualized Sharpe Ratio by balancing higher profits with lower risk (standard deviation).

The Sharpe Ratio term is defined as:

$$\mathrm{SR}^*(x, \hat{x}) = \frac{\frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} \mathrm{PnL}_a^*(x_i, \hat{x}_i)}{\sqrt{\frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} \left(\mathrm{PnL}_a^*(x_i, \hat{x}_i) - \frac{1}{n_{\text{batch}}} \sum_{j=1}^{n_{\text{batch}}} \mathrm{PnL}_a^*(x_j, \hat{x}_j) \right)^2}}$$

Maximizing the Sharpe Ratio involves not only increasing the average PnL but also minimizing its variability (standard deviation). To explicitly focus on reducing the variability of the PnL series, a Standard Deviation (STD) term is added to the loss function. This term penalizes higher deviations in the PnL series, pushing the generator towards more stable predictions.

The STD term is defined as:

$$\mathrm{STD}(x, \hat{x}) = \sqrt{\frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} \left(\mathrm{PnL}_a^*(x_i, \hat{x}_i) - \frac{1}{n_{\text{batch}}} \sum_{j=1}^{n_{\text{batch}}} \mathrm{PnL}_a^*(x_j, \hat{x}_j) \right)^2}$$

This term should only be used in conjunction with the PnL^* term due to their interdependence, and it should not be combined with the SR^* term, as they convey similar information.

Fin-GAN Loss Function

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:

$$L_G(x, \hat{x}) = J(G)(x, \hat{x}) - \alpha \cdot \text{PnL}^*(x, \hat{x}) + \beta \cdot \text{MSE}(x, \hat{x}) - \gamma \cdot \text{SR}^*(x, \hat{x}) + \delta \cdot \text{STD}(x, \hat{x})$$

where $J(G)$ is one of the standard GAN losses (zero-sum, cross-entropy, or Wasserstein), and the hyperparameters $\alpha, \beta, \gamma, \delta$ 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*MSESTD, PnL*MSESR*, 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*MSESTD, PnL*MSESR*, 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.

Gradient Norm Matching

In order to avoid hyper-parameter tuning to find $\alpha, \beta, \gamma, \delta$, we use an 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 equation (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.

The model's training became more challenging because its loss function became more complicated. In early tests, adjusting the settings for certain hyperparameters (α, β, γ) showed that the Mean Squared Error (MSE) and Sharpe Ratio components caused the generator to produce very limited and narrow results. It was unclear why a high γ value in the Sharpe Ratio led to mode collapse, which occurs when the model only generates similar outputs, since the Sharpe Ratio was supposed to reduce the variability in profits and

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 $\bar{r}_t \leftarrow \text{mean}(\hat{r}^i_t)$.
$MAE \leftarrow MAE(r_t, \bar{r}_t)$; $RMSE \leftarrow RMSE(r_t, \bar{r}_t)$;
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 $std(\{\bar{r}_t\}_{t \in \text{test set}}) < \epsilon$ **then**
  Mode collapse.
else
  No mode collapse.
end if

Figure 3: An example of a placeholder image. Replace 'example-image' with your image file name.

losses, not the generated data itself. However, adding a profit-and-loss (PnL) term helped prevent this mode collapse by allowing the generator to move away from problematic areas in the loss function. When they used a technique called gradient norm matching to fine-tune the hyperparameters $\alpha, \beta, \gamma, \delta$, the model did not experience mode collapse, no matter how the network's initial weights were set. Additionally, they discovered that using Xavier initialization for the network's weights instead of He initialization significantly reduced the chances of mode collapse, which was happening 67% of the time with He initialization. Switching to Xavier initialization effectively solved 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 the model moves towards a standard supervised learning setting, using a generator-only model. On the other hand, if $\alpha, \beta, \gamma, \delta$ are too small, the model does not pro-

vide 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 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.

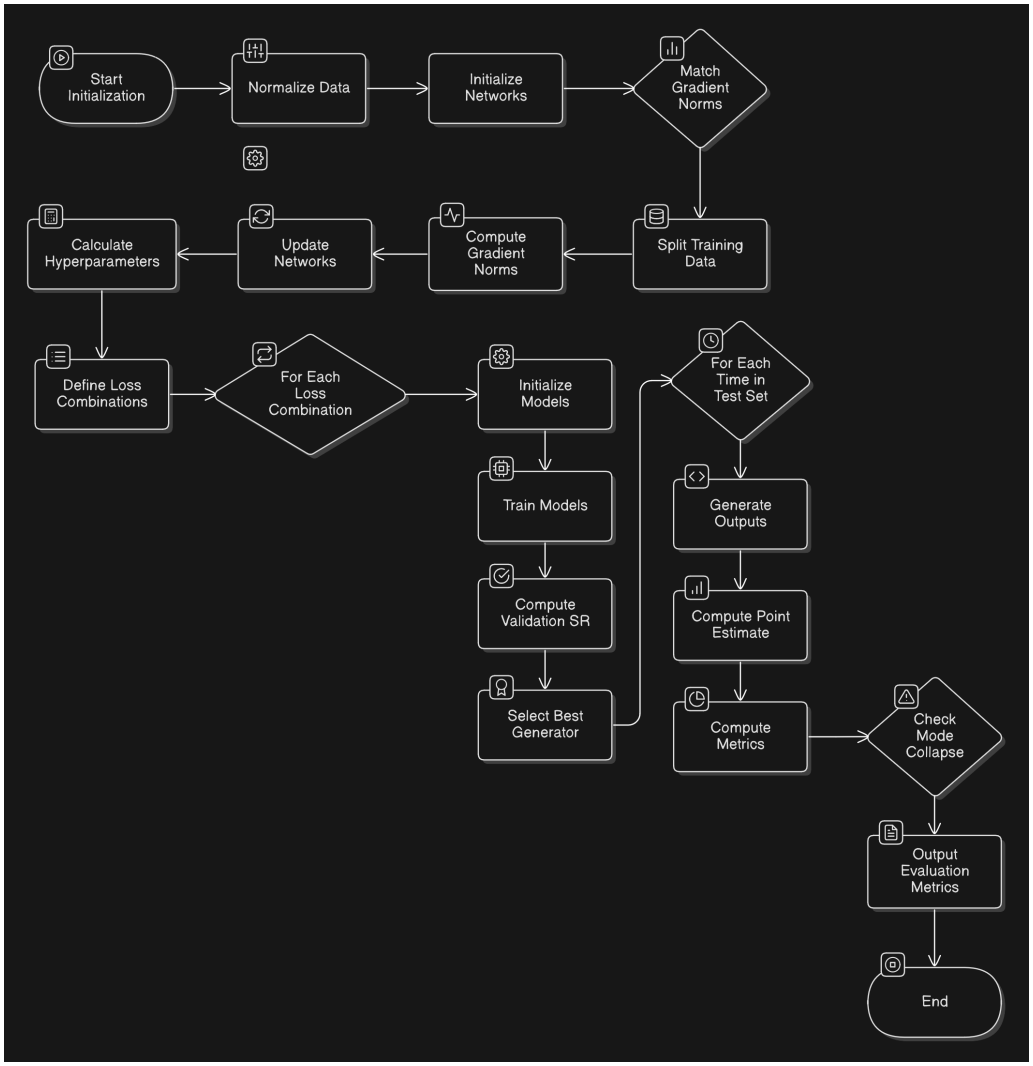

Figure 4: An example of standard GAN

Data Description

The dataset used in this study includes daily stock ETF-excess returns and daily raw ETF returns, sourced from Yahoo Finance Services. The analysis spans from January 2010 to December 2021, with the data split into three periods: training (80%), validation (10%), and testing (10%).

- **Training period:** January 1, 2010, to August 9, 2017.

- **Validation period:** August 10, 2017, to October 22, 2019.
- **Testing period:** October 23, 2019, to December 31, 2021.

These periods differ significantly in economic conditions, with the test data including the early stages of the COVID-19 pandemic, adding complexity to the forecasting task.

The dataset captures two types of returns sequentially:

- **Close-to-open (overnight) returns.**
- **Open-to-close (intraday) returns.**

These returns are treated as distinct time units. Each input time series alternates between these returns, using a sliding window of one time unit and a condition window of ten time units (five trading days). The model predicts one time unit ahead. All stock prices are adjusted using the cumulative adjustment factor (`cfacpr`), and returns are capped at $\pm 15\%$ to reduce the impact of outliers.

Preprocessing

For preprocessing, the data is converted into a continuous time series of returns. It is then split chronologically into three disjoint segments: training, validation, and testing. Each segment is further processed into a matrix format, where each row represents a sliding window. The matrix structure is as follows:

- The first ten columns contain the returns from the condition window:

$$(x_1, x_2, \ldots, x_{10}).$$

- The last column contains the target return: x_{11}

3 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 datasets, the corresponding layer dimensions and the noise dimension are all set to 8. The generator uses ReLU as the activation function, while the discriminator uses the sigmoid function.

3.1 Training

The Fin-GAN model is optimized using RMSProp as the optimizer and employs Xavier initialization to set the weights, ensuring stable training without additional hyperparameter tuning. The training alternates between updates to the generator and discriminator, with one discriminator update for every generator update ($k = 1$). To avoid data leakage, reference batch normalization is used by selecting the first 100 samples from the training dataset as a fixed reference batch. The minibatch size is set to 100, and the learning rates for both networks are set to 0.0001.

The training process begins with an initial phase of 25 epochs, where only the Binary Cross-Entropy (BCE) loss is used. This phase implements gradient norm matching, which determines the values of the hyperparameters $\alpha, \beta, \gamma, \delta$, controlling the importance of additional loss terms. After this, the training splits into multiple branches, each using a different combination of loss functions. Each branch is trained for 100 additional epochs. At the end of this phase, the Sharpe Ratio is calculated on the validation set for each branch, and the loss function combination and generator model that maximize the Sharpe Ratio are selected.

3.2 Gradient Stability

Before adding the additional loss terms to the Fin-GAN training process, the stability of gradient norms during training is analyzed to ensure there are no issues with exploding or vanishing gradients. This investigation focuses on the behavior of gradients when training the ForGAN model using the Binary Cross-Entropy (BCE) loss with the RMSProp optimizer for 25 epochs, corresponding to the period used for gradient norm matching.

In the analysis, sample gradient norms of each term in the loss function (Equation 22) are plotted. For this example, the daily excess stock returns time series of Pfizer (PFE) is used. The results show that the gradient norms remain stable, without any signs of exploding (becoming excessively large) or vanishing (becoming extremely small). Moreover, the gradient norms of all terms are observed to be on a similar scale, indicating balanced behavior during training.

3.3 Mode Collapse

In the context of time series data, mode collapse refers to a failure in the generator's ability to produce diverse outputs. This can happen in two ways:

- **Distribution Mode Collapse:** The generated output for a specific condition (time point) collapses into a narrow distribution, losing variability within that condition.
- **Mean Mode Collapse:** The means of the generated distributions across all conditions collapse, leading to a lack of variability across different conditions.

If both types of collapse occur fully and simultaneously, the generator produces only a single-point estimate, which is highly undesirable for probabilistic forecasting. To detect mode collapse in this setup, the standard deviation of the out-of-sample generated means for a particular condition is calculated. If this standard deviation falls below a threshold ($\epsilon = 0.0002$), mode collapse is considered to have occurred.

Using Xavier initialization for the network weights proved effective in mitigating mode collapse in ForGAN. Furthermore, the additional loss terms introduced in the Fin-GAN model acted as strong regularizers, preventing mode collapse in all experiments, regardless of whether the networks were initialized with Xavier or He initialization. In contrast, ForGAN experienced some form of mode collapse in 67% of simulations when He initialization was used.

During early numerical experiments (before adding the STD term), it was observed that the MSE and Sharpe Ratio (SR) terms, when assigned high weights (β and γ), increased the likelihood of mode collapse. This happened because the MSE term encouraged generated values to closely match the targets, reducing variability. However, the inclusion of the PnL term effectively resolved this issue.

3.4 Generated Distributions

The generated distributions resulting from training with different combinations of Fin-GAN loss terms were analyzed using the excess returns of Pfizer (PFE) as training data and the same condition from the test set as the target. The results show that training solely with the Binary Cross-Entropy (BCE) loss produced more symmetrical forecast distributions, while incorporating additional Fin-GAN loss terms shifted the distributions, allowing the generator to better capture directional price movements.

A comparison of the generated means with the true target distribution reveals that certain loss combinations, such as PnL with STD and the Sharpe Ratio (SR) term, caused significant shifts in the forecast distributions. This is expected since both these terms prioritize directional accuracy and risk-adjusted performance. On the other hand, adding the MSE term resulted in generated means that closely resembled the true target distribution, as the MSE emphasizes minimizing the difference between forecasts and actual values.

Additionally, the inclusion of loss terms like PnL, MSE, and SR enabled the model to better balance uncertainty in predictions. For instance, these combinations led to trades with slightly lower weights compared to more confident forecasts like those trained with PnL and STD or SR alone, which resulted in stronger trades (long ETF and short stock) due to higher confidence in forecast direction.

Figures 5 and 6 serve as illustrative examples of how different training objectives influence the generated distributions.

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 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.

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.

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.

$$L_{\text{Fin}}(x, \hat{x}) = \text{MSE}(x, \hat{x}) - \alpha PnL^*(x, \hat{x}) - \gamma SR^*(x, \hat{x}) + \delta STD(x, \hat{x}). \tag{1}$$

Comparison of Intraday Cumulative PnL and Cumulative PnL for TCS

The first plot (Figure 9) showcases the **Intraday Cumulative PnL** for TCS, comparing metrics like PnL, MSE, SR, and STD. It highlights short-term dynamics with moderate variations. The second plot (Figure 10) evaluates the **Cumulative PnL**, demonstrating broader performance trends where ForGAN significantly outperforms others, indicating better long-term profitability. While the first plot emphasizes intraday volatility, the second

plot focuses on cumulative growth and stability over time, showcasing ForGAN's superior optimization for sustained profitability.

Comparison of Cumulative PnL for TCS: FinGAN vs LSTM

The first plot (Figure 12) demonstrates the cumulative PnL performance of TCS using the FinGAN model. It showcases significant variations and highlights the model's optimization in terms of metrics such as PnL, MSE, SR, and STD. The second plot (Figure 13) shows the cumulative PnL using the LSTM model, which provides a broader perspective of its long-term growth and stability. The comparison underscores FinGAN's superior ability to manage cumulative profitability, with LSTM showing more gradual improvements.

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, and Sharpe Ratio-based loss function terms, rendering GANs more suitable for a classification task and placing them into a supervised learning setting.

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

[b]0.8

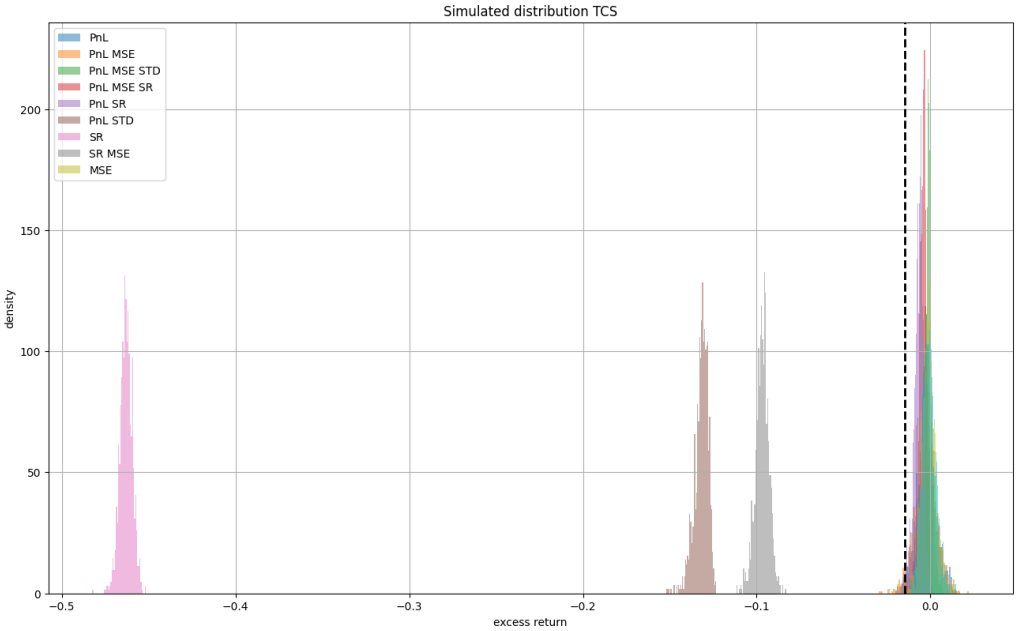

Figure 5: Intraday Cumulative PnL for TCS

[b]0.8

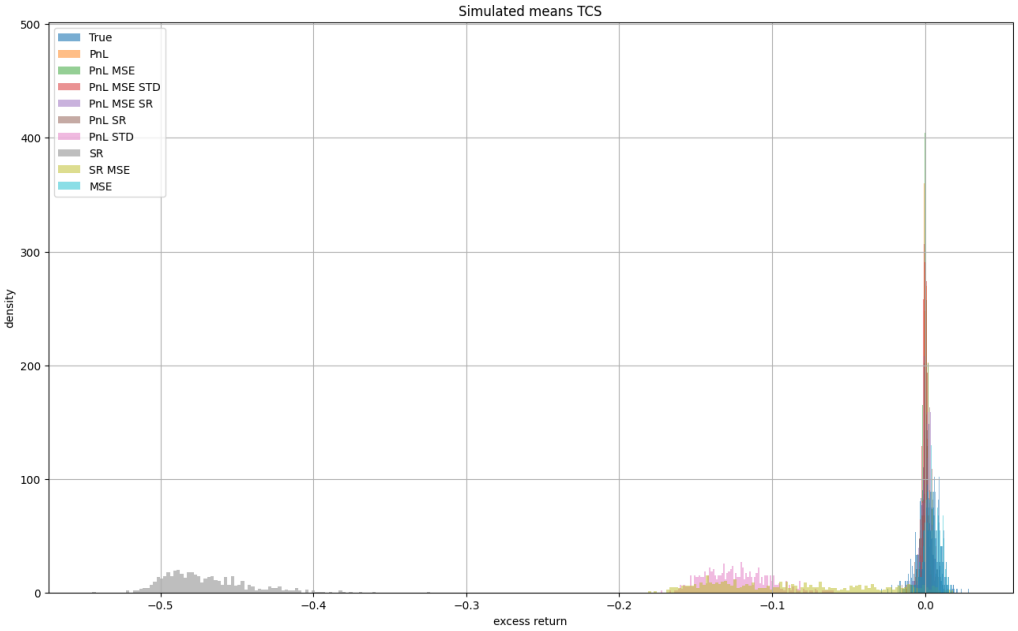

Figure 6: Cumulative PnL for TCS

Figure 7: Comparison of Intraday Cumulative PnL and Cumulative PnL for TCS.

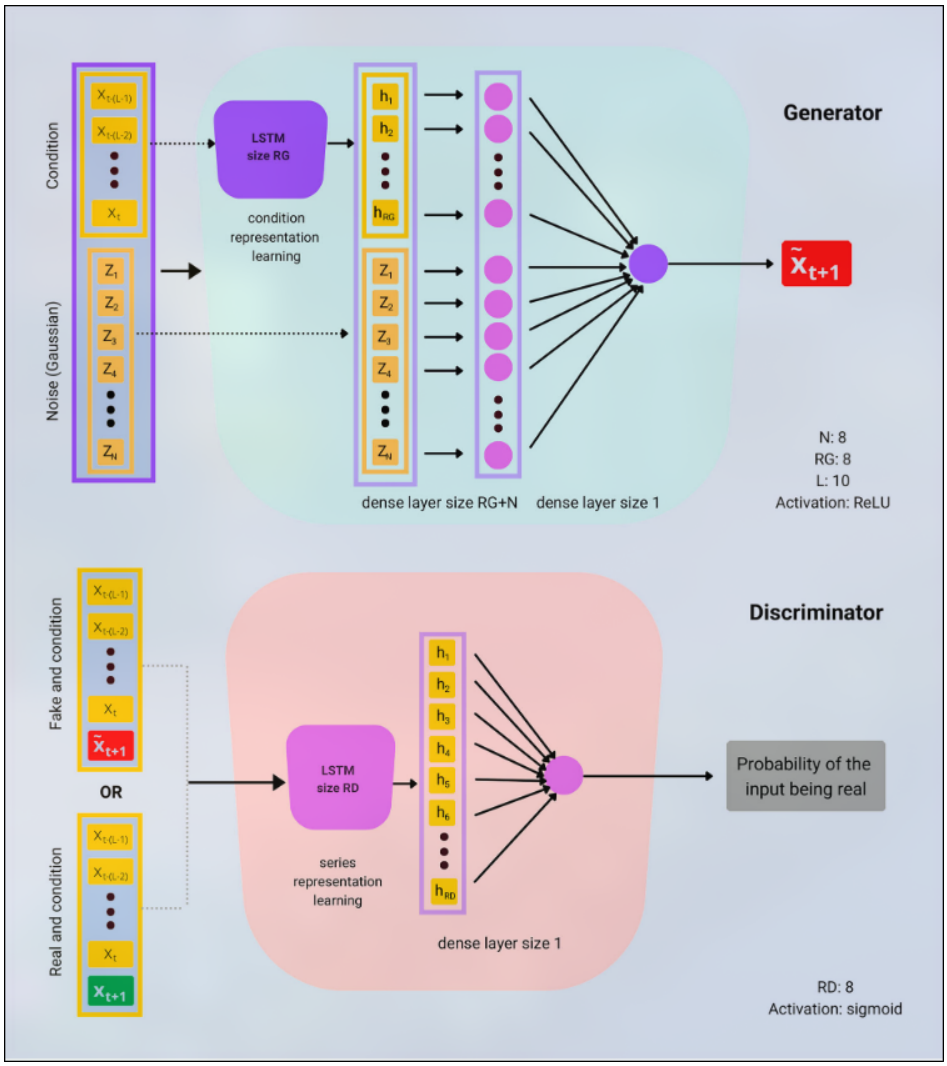

Figure 8: An example of standard GAN

[b]0.8

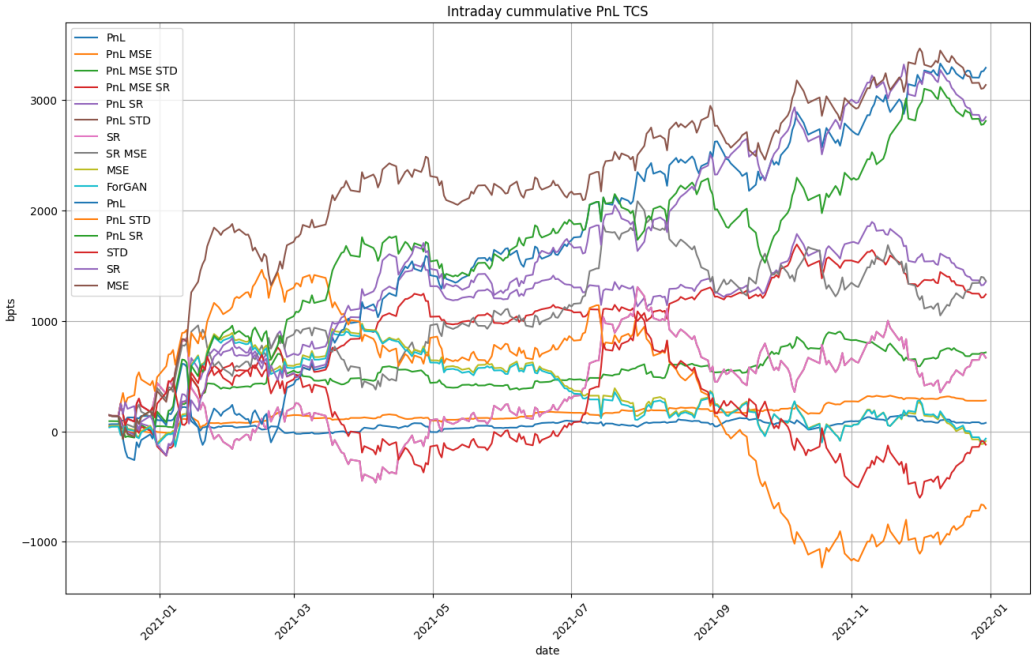

Figure 9: Intraday Cumulative PnL for TCS

[b]0.8

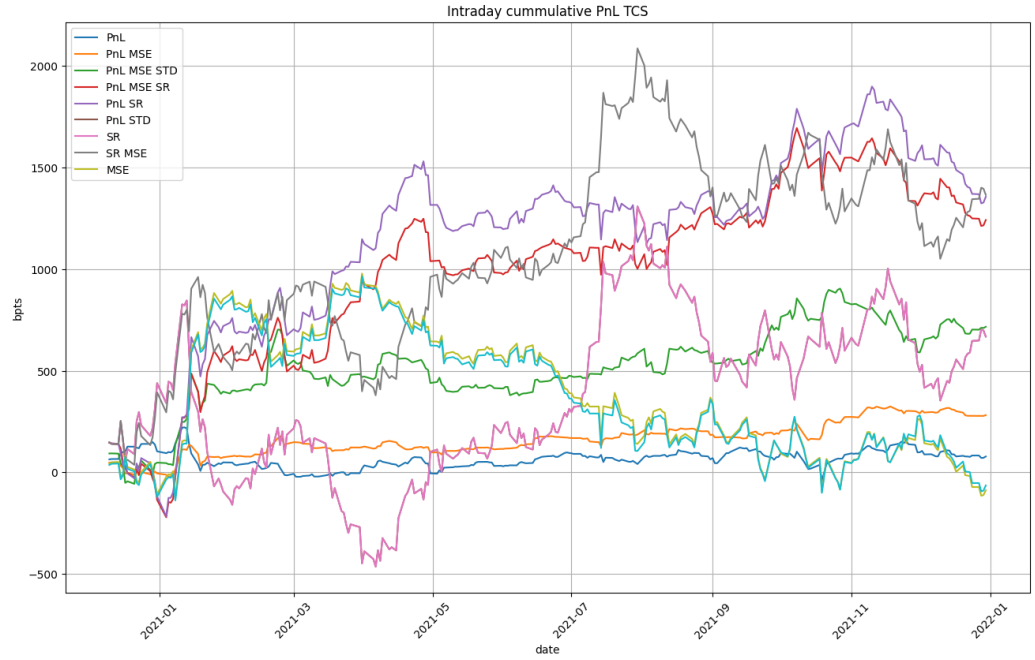

Figure 10: Cumulative PnL for TCS

Figure 11: Comparison of Intraday Cumulative PnL and Cumulative PnL for TCS.

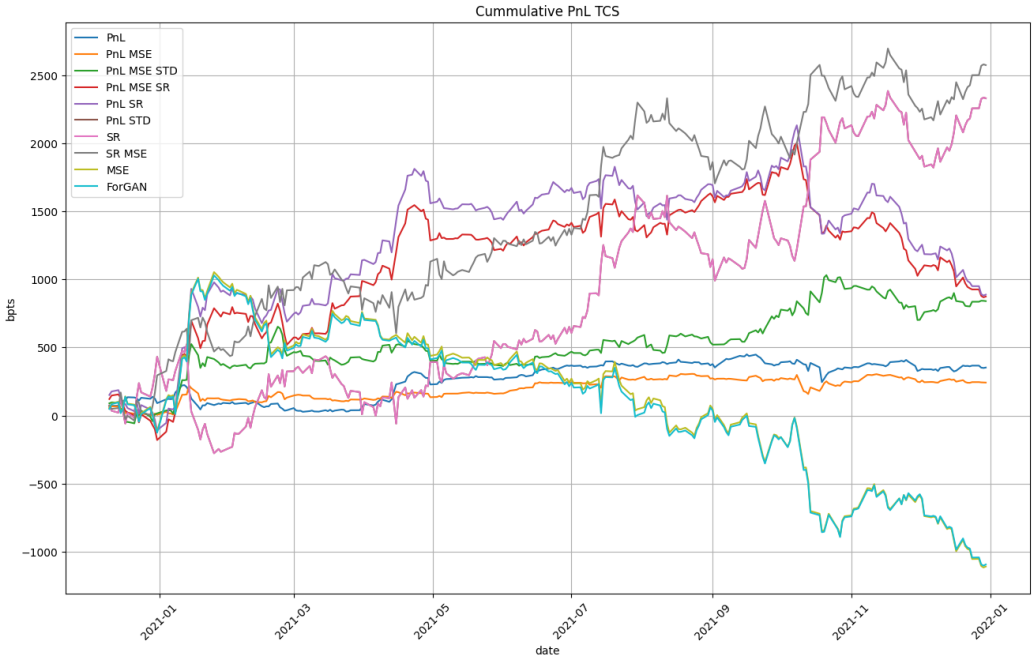

Figure 12: Cumulative PnL for TCS using FinGAN

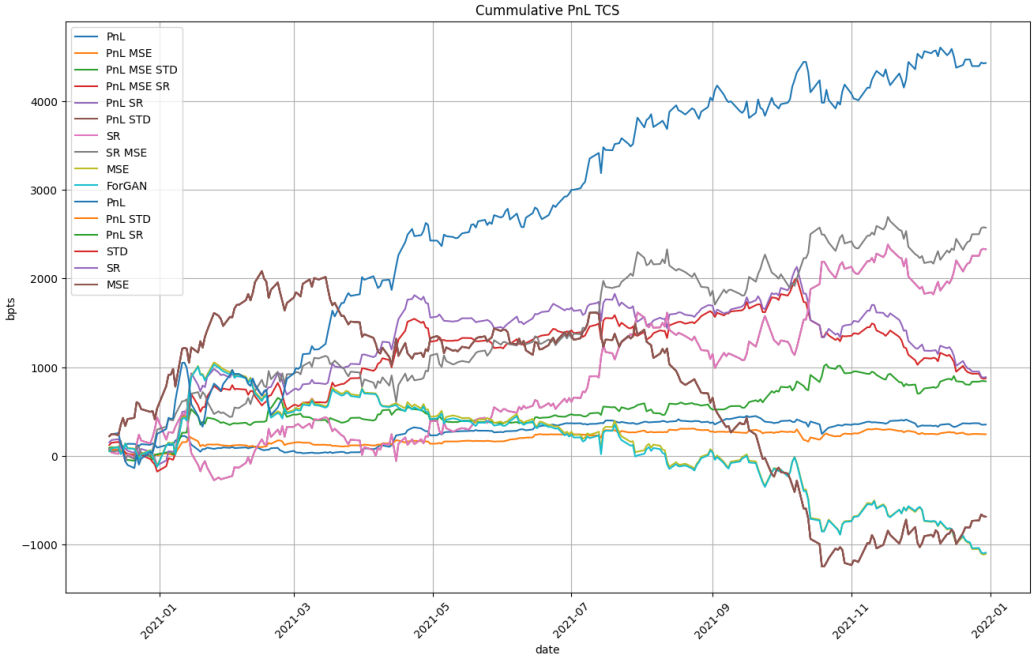

Figure 13: Cumulative PnL for TCS using LSTM

Figure 14: Comparison of cumulative PnL for TCS using FinGAN and LSTM.