CSCD94 Midterm Report - Targeted Manipulation

Dominik Luszczynski[1]

[1] University of Toronto Department of Computer Science Toronto, ON, Canada

1 Introduction

The increased use of machine learning (ML) and artificial intelligence (AI) models in high risk sectors, such as healthcare and finance, make it imperative that these predictive models are robust. However, most research has been geared towards improving the performance models, rather than considering their security. In fact, deep neural networks (DNN) are highly susceptible to adversarial attacks [1]. Adversarial attacks occur when an attack slightly modifies the input to a model, typically by adding noise, which causes a model to produce an incorrect result [2]. Adversarial attacks are extremely problematic in noise sensitive domains such as healthcare, finance and computer vision. For example, a reliable and safe self-driving car must be able to identify a stop sign, even if some noise has been added to its vision. Adversarial attacks come in two forms: white and black-box attacks. A white box attack occurs when the attacker has full access to all information about the model, including parameters, which enables the attacker to exploit gradient information [3]. A black-box attack occurs when the model is hidden from the attacker, and they do not have any knowledge about the structure or parameters [3], [4].

Most contributions to adversarial attack research has been focused on image and text classification, while research about attacks on time series data is still in its early stages [3]. There have been recent advancements in the time series domain, with Gallager et al., attacking a simple three layered Convolutional Neural Network (CNN) forecasting the Google Stock from 2006 to 2018 with Fast Gradient Sign Method [5]. Moreover, Rathore et al., applied the Fast Gradient Sign Method and the Basic Iterative Method on a classification model trained on 54 different time series datasets related to healthcare, vehicle sensors and electrical equipment [6]. Rathore et al., discovered that time series classification models are susceptible to adversarial attacks, and defense strategies such as adversarial training (including adversarial examples in the training set) are effective [6]. The introductory research conducted shows how susceptible many time series models are to adversarial attacks; however, they typically focus on classification problems, small-scale models and do not consider the believability of an adversarial example [3], [7].

A significant challenge regarding adversarial attacks in the time series domain is that, unlike images, perturbations made on time series are more noticeable. To combat this challenge, Shen and Li introduced Stealthy attacks which rely on using cosine similarities to ensure temporal characteristics of time series remain intact [3]. Furthermore, Pialla et al., introduced the Smooth Gradient Method which involves constraining the noise optimization by a regularization term penalizing non-smoothness [8]. As a result, these attack methods are able to bypass human and machine detection; however, there is a tradeoff between the amount of error caused by the attack versus making it look believable.

In an attempt to generate more believable attack inputs, Generative Adversarial Networks (GANs) have been used to create adversarial attacks [9]. GANs typically comprise of two networks, a generator and a discriminator (or critic), where the generator creates synthetic data, while the discriminator determines whether it is real or fake [10]. At the end of training, the GAN should be able to generate believable synthetic data that would ideally bypass detection. Researchers have experimented with adding a second critic to the GAN architecture, which is the model that the attacker is aiming to fool [9]. However, similar to adversarial attacks, the majority of research on GANs have been on images, and there have been even fewer studies using GANs to generate adversarial examples for time series forecasting [9], [11], [12]. There have been similar adversarial network attacks on time series data, as S. Baluja and I. Fischer developed an Adversarial Transformation Network to develop adversarial examples, which balance an input-space loss (L_2 loss) with an output space loss (how much the model's prediction has changed) [13].

This research aims to build upon previous studies using GANs to create adversarial examples, by adapting existing GAN frameworks for generating adversarial attacks on images to financial stock data.

The use of GANs enables an attacker to generate realistic and believable time series data, ideally without sacrificing how much error the model produces. Moreover, existing adversarial attacks made for time series data are performed on relatively simple architectures, such as a shallow CNN [3], [5]. Thus, this research is aimed to target more state of the art forecasting models, such as the N-HiTS model [14]. However, rather than training a model to forecast a single stock, the N-HiTS model is trained on over 300 recordings of stock data to improve the robustness of the model, thus making adversarial attacks more difficult. Figure 1 provides a visual display of the adversarial GAN architecture. Furthermore, this research aims to investigate existing adversarial attack strategies and their performance on the N-HiTS architecture.

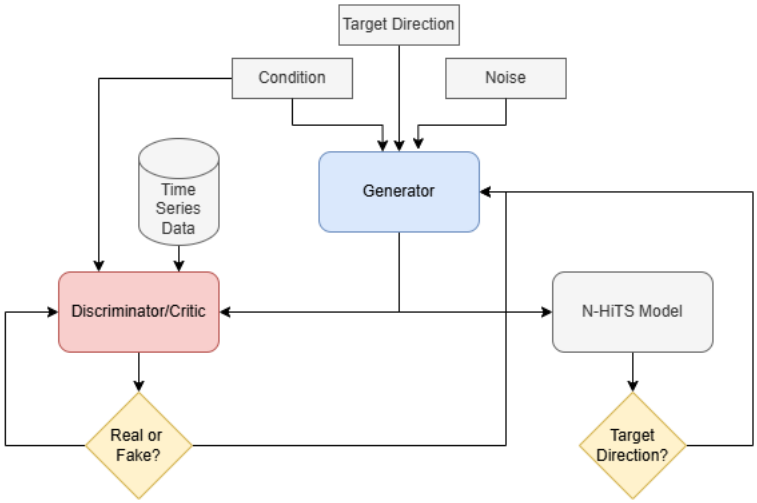


Figure 1: High-level architecture diagram of the proposed targeted adversarial GAN

2 Forecasting Model

2.1 Dataset and Preprocessing

All financial stock data was collected from the Center for Research Security Prices (CRSP). Specifically, the daily adjusted price (adjprc) of all S&P 500 stocks was collected from CRSP for the 2015:05:01-2025:05:01 period. To determine the training, validation and testing set split, a stratified split was made by creating eight evenly distributed bins based on the "Stock Price" collected from [15]. The training, validation and testing set comprises of 360, 48, and 72 recordings respectively, or equivalently 75%, 10% and 15% of the dataset. Any stocks with less than 600 days of recordings were omitted. In order to enrich the feature set for the forecasting model, several features were derived from the adjusted price:

- Rolling mean with window sizes of 5, 10, 20
- Rolling standard deviation with window sizes of 5, 10, 20
- Log Returns
- Rate of Change with a delta of 5
- Exponential moving averages with window sizes 5, 10 and 20

Furthermore, from the date of each adjprc, the day of the week was extracted as a categorical feature because it has been shown that there are differences in volatility between the beginning and end of the trading week [16]. In order to preserve the computational graph for adversarial attacks, all feature generation was performed in PyTorch. Extensive feature selection was not performed as the focus of this research project are the adversarial attacks.

2.2 N-HiTS Model

N-HiTS is a novel projection model which builds upon the N-BEATS architecture, simultaneously improving computational performance and accuracy by sampling the time series at different rates [14].

The N-HiTS architecture uses Multi-layer Perceptrons (MLPs) for each block of the time series to estimate coefficients for the backcasts and forecasts [14]. The backcasts are used to clean future blocks, while the forecasts are summed to generate the predictions [14]. However, the N-HiTS model introduces novel components to each block, like using MaxPool with some kernel size k_b for each block b, claiming that it helps the MLP focus on low frequency and large scale contents of the time series, which allows it to sample different rates [14]. After the pooling layer, a non-linear regression is applied to estimate the backcasts and forecasts, and to eventually generate the final predictions, hierarchal interpolation is performed [14]. Given that N-HiTS is making predictions at different frequencies, the hierarchal interpolation assigns each stack of predictions with an expressiveness ratio, representing the number of predictions in the stack, then sums the stacks of predictions according to the magnitude of the expressiveness ratio [14].

The N-HiTS model was implemented through the PyTorch-Forecasting library. The model was trained for 100 epochs, with an encoder length of 300 and a prediction length of 50, meaning it uses the previous 300 days of data to forecast the next 50. The hyperparameters used for the N-HiTS model are stated in Table 3 in the Appendix. A PyTorch Lightning Tuner was used to determine the optimal learning rate, while weight decay and the hidden size were determined experimentally. Batch normalization and early stopping with a patience of 15 were used to help prevent overfitting. Several loss functions were experimented with, such as Mean Squared Error (MSE), L1 Loss, Mean Absolute Percentage Error (MAPE), and Symmetric Mean Absolute Percentage Error (SMAPE); however, Quantile Loss with quantiles 0.01, 0.1, 0.05, 0.5, 0.95, 0.99, and 0.999 gave the best results.

2.3 N-HiTS Results

Using the test set, results were computed by comparing the true adjusted price with the adjprc forecast generated from the N-HiTS model. The predictions were made on the entire recording starting from the 300th day, as the N-HiTS model requires a look-back period of 300 days. Given that the model produces 50 day forecasts, N-HiTS model makes predictions over a rolling window of 300 days. Corresponding predictions on the same day are then averaged for the final prediction. The model achieved an average Mean Absolute Error (MAE) of 6.62, a Root Mean Squared Error of 9.43 and a Mean Absolute Percentage Error of 5.29%. To illustrate, Figure 2 shows a predicted forecast generated by the N-HiTS model compared to the actual adjusted price. Although the model does not capture the small fluctuations of the actual adjusted price, the overall trend of the stock is learned. The lack of small fluctuations in predictions can be explained by the N-HiTS model being trained on 360 different stocks, rather than a single stock, improving the generalizability of the forecasting model while sacrificing learning the local noise of the stock.

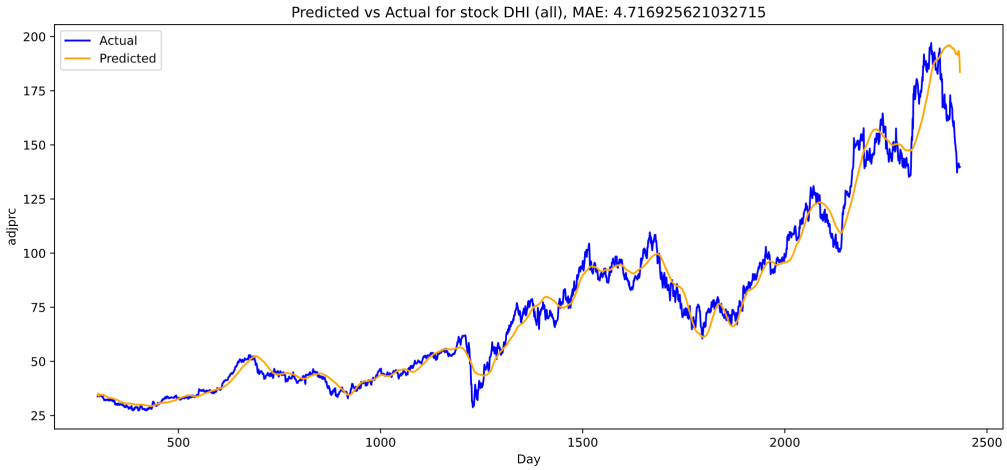


Figure 2: Example forecast for the *DHI* Stock generated by the N-HiTS model

3 White Box Adversarial Attacks

In this section, only white box adversarial attacks are considered, with the premise that, if the Targeted Adversarial GAN is able to perform similarly to a white box attack, then it is a valid and effective adversarial strategy. Furthermore, throughout this section the adversarial input will be denoted as x_{adv} and the corresponding ground truth adjusted price will be denoted as adjprc.

Most iterative adversarial attacks follow a similar pattern, as described in Algorithm 1, where the attacker would iteratively probe the model with its adversarial input, x_{adv}, compute a loss, and update x_{adv} with respect to the gradient of the loss. However, each baseline attack either updates x_{adv} or computes the loss differently. Note that L1 Loss was used in Algorithm 1, as the goal was to maximize the MAE, however any regression loss function would work.

Algorithm 1 Generic Iterative Adversarial Attack Algorithm on the N-HiTS model

Input: Stock Forecasting model **model**, time series recording **df**, number of iterations **iter** $\in \mathbb{R}$, maximum perturbation $\boldsymbol{\epsilon} \in \mathbb{R}_{\geq 0}$, and the step size $\boldsymbol{\alpha} = 1.5 \cdot \boldsymbol{\epsilon}/\textbf{iter} \in \mathbb{R}_{\geq 0}$

adjprc ← getAdjprc(df)
x_{adv} ← adjprc
days ← getDays(df)
for i **in** iter **do**
    x_{adv} ← requires grad
    model ← zero grad
    features ← $getFeatures(x_{adj}$, days)
    output ← $model$(features)
    predictions ← $predictionsOverMovingWindow(output)$
    loss ← $LossFunction$(predictions, adjprc)
    loss.backward()
    with no grad:
        $x_{adv} \leftarrow AttackMethod(x_{adv}, \alpha)$
    detach x_{adv}
end for
return x_{adv}

3.1 FGSM

The Fast Gradient Sign Method (FGSM) is the simplest way to create an adversarial attack. The FGSM involves computing the loss with respect to the forecasting predictions and the ground truth label (adjprc) then, using the sign of the gradients, x_{adv} is adjusted to maximize the loss [17]. That is:

$$x_{adv} = x + \alpha \cdot sign(\nabla_{x_{adv}} loss) \quad [17]$$

Note that the FGSM is done in a single step, thus, the step size $\alpha = \epsilon$. Normally, when a neural network performs gradient descent, the gradients determine the direction in which to move the weights. Since the gradient term is subtracted from the weights, the weights move opposite of the gradient's sign. However, the FGSM computes the loss with respect to each element in the time series x_{adv}, and instead of subtracting the gradient, it is added, leading to the maximization of the loss [17].

3.2 BIM

The FGSM is a special case of the Basic Iterative Method (BIM), also called the Iterative Fast Gradient Sign Method (I-FGSM), where the number of iterations is 1 and the step size is ϵ [1]. The BIM applies the FGSM repeatedly for a desired number of iterations, but with a smaller step size α [18]. The BIM follows the same generic algorithm shown in Algorithm 1, initializing the adversarial attack x_{adv} to the unperturbed adjprc, and then proceeds to iteratively update x_{adv} by taking the sign of the gradient, and moving x_{adv} in the direction that maximizes the loss. However, to ensure that x_{adv} is in the ϵ-neighborhood of the original time series, x_{adv} is clipped between adjprc - ϵ and adjprc + ϵ [18].

3.3 MI-FGSM

The Momentum Iterative Fast Gradient Sign Method (MI-FGSM), makes use of the momentum technique used to accelerate gradient descent algorithms [1]. By accumulating the direction of the loss function gradients across iterations, the momentum techniques allow gradient descent algorithms to avoid getting stuck in a local minima [1]. Let g_t be the variable containing the gradients up to iteration t and let μ be the decay factor [1]. Initialize $g_0 = 0$, and x_{adv} is updated by first computing g_{t+1} as:

$$g_{t+1} = \mu \cdot g_t \cdot \frac{\nabla_{x_{adv}} loss}{||\nabla_{x_{adv}} loss||_1}$$

then is moved in the direction of the g_{t+1}:

$$x_{adv} = x_{adv} + \alpha \cdot sign(g_{t+1})$$

[1]. Finally, similar to BIM, x_{adv} is clipped to stay within adjprc - ϵ and adjprc + ϵ. The gradient of the loss is normalized by the 1-norm since each iteration may produce different magnitudes of gradients [1]. The current implementation of the MI-FGSM uses $\mu = 0.35$, which was determined experimentally; however, more experimentation will be performed for the final paper.

3.4 Stealthy Iterative Method

An important factor to consider when designing an adversarial attack is that the perturbed time series should be relatively undetectable. Although the size of ϵ controls the magnitude of the noise added to input, Z. Shen and Y. Li propose the Stealthy Iterative Method (SIM) to ensure the temporal characteristics of the time series remain intact [3]. Z. Shen and Y. Li extend the BIM by introducing cosine similarity comparisons between x_{adv} and adjprc [3]. First, if the cosine similarity of adjprc and x_{adv} is larger than the similarity between adjprc and adjprc + ϵ, then x_{adv} is kept as is, otherwise $x_{adv} \leftarrow$ adjprc + ϵ [3]. Then, if the cosine similarity between adjprc and x_{adv} is larger than the cosine similarity between adjprc - ϵ, x_{adv} is kept as is, otherwise $x_{adv} \leftarrow$ adjprc $- \epsilon$ [3].

3.5 Targeted Iterative Method

Although the goal for many adversarial attacks is to maximize the error of the model, in realistic scenarios the attacker would want to choose the direction in which the model fails. Thus, instead of simply maximizing the error, the Targeted Iterative Method (TIM) is another extension on the BIM which allows the attacker to add noise to adjprc, resulting in the forecast going in the direction that the attacker desires. The TIM takes in a direction d and margin γ and calculates a new target as:

$$tar = \text{adjprc} + d \cdot \gamma \qquad [3]$$

Now, rather than modifying x_{adv} to go in the direction that maximizes the loss between the predictions and adjprc, the loss between the predictions and tar is computed instead [3]. Given that the goal is to minimize the error between the predictions and tar, for a given step size α, x_{adv} is updated as:

$$x_{adv} = x_{adv} - \alpha \cdot sign(\nabla_{x_{adv}} loss) \qquad [3]$$

3.6 C&W

The Carlini and Wagner attack (C&W) deviates from the general approach to adversarial attacks. Rather than iteratively modifying x_{adv}, a C&W attack aims to find the optimum noise vector η that is added to the adjprc [8]. Specifically, the C&W attack aims to solve the following optimization problem:

$$\min ||\eta||_2 + f(x_{adv} + \eta)$$

where f is a function that enforces an incorrect prediction [8]. Since this research focuses on a regressive task, f was chosen to be the L1 Loss between the attack predictions and a target tar defined in Section 3.5. In the following sections, the C&W attack performed was targeted below the original adjprc.

3.7 Attack Results

Given that the N-HiTS model requires a look-back period of 300 days, a segment of 400 days was chosen as it gives 100 days of predictions made from averaging a moving window. Attacks were performed on the first 400 days of each recording in the test set, due to the low MAE for the normal forecast. Predictions made on the unperturbed adjusted price had an average MAE of 3.37. Table 1 shows the results for each baseline attack method described in Section 3. Notably the MI-FGSM produced the most error with an average MAE of 5.68, while the SIM performed the worst with an average MAE of 3.57. This is not surprising as the SIM is designed to produce an adversarial input with similar temporal characteristics to the original adjprc. Finally, given that the TIM and C&W methods are targeted methods, it is not surprising that there is no significant increase in the average MAE because targeting the adversarial attack above or below the adjprc may actually improve the prediction. For example, if the prediction from the N-HiTS model is above the ground truth, targeting the attack below the adjprc will improve the forecast.

Table 1: Average metrics for different attack methods performed on the first 400 days of each recording, with $\epsilon = 1$. The best metrics are bolded.

Attack	*MAE*	*RMSE*	*MAPE*
Normal	3.37	4.04	0.052
FGSM	4.79	6.01	0.079
BIM	5.66	6.88	0.098
MI-FGSM	**5.68**	**6.90**	**0.099**
SIM	3.57	4.59	0.058
TIM (Up)	4.30	5.32	0.076
TIM (Down)	4.15	5.20	0.068
C&W	3.93	4.99	0.064

Furthermore, as shown in Figure 3, all attack methods improve with larger sizes of ϵ. Notably, BIM and MI-FGSM are able to obtain a significant increase in MAE with $\epsilon = 0.25$, both with MAEs of 4.04. Figures 5 and 6 in the Appendix illustrate forecasts of the various attack methods on the *DHI* and *POOL* stocks. The attacks were carried out on the first 400 days of the recording.

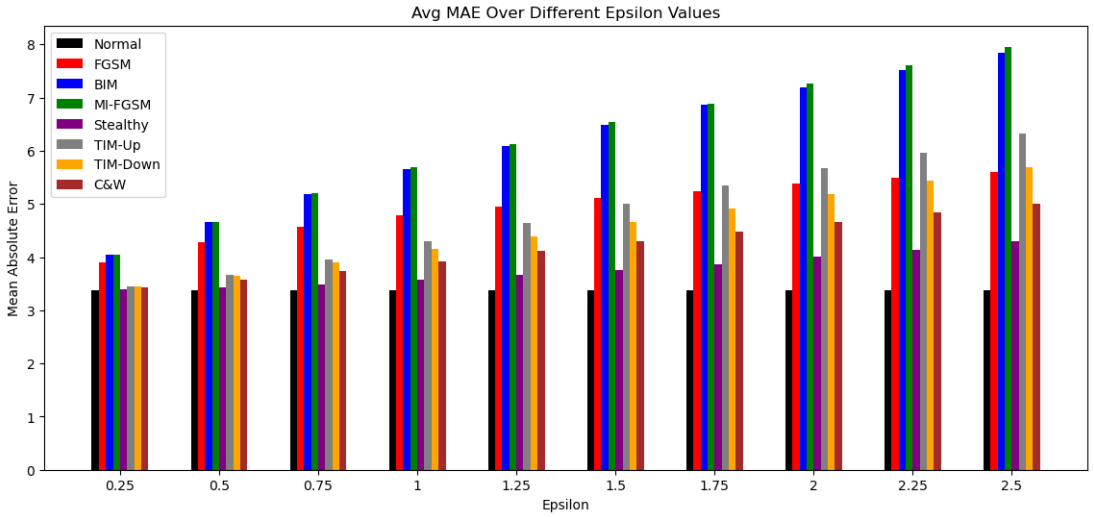


Figure 3: Average MAE for different attack methods performed on the first 400 days of each recording, with varying ϵ sizes.

4 Generative Adversarial Networks

GANs were originally created by I. Goodfellow et al., and describe a neural network with two submodels, a generator and discriminator [19]. The generator G, aims to create synthetic data by learning the distribution of the real data, while the discriminator D, aims to distinguish the real from the synthetic

data [19]. The objective function of a vanilla GAN is:

$$\min_G \max_D \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[(\log(1 - D(G(z)))] \qquad [19]$$

Essentially, the discriminator aims to maximize performance when classifying real data, while the generator aims to maximize the performance of its synthetic data being labeled as real by the discriminator.

To become familiar with the training and implementation of GANs, multiple simple versions of the GAN architecture were implemented such as a Deep Convolutional GAN (DCGAN) and a Wasserstein GAN (WGAN), before finally developing a Conditional WGAN (C-WGAN). All GANs implemented are trained using the stock data extracted as in Section 2.1 with the ticker A; however, all GAN implementations generate 50 days of log returns rather than adjprc. Random continuous intervals of 50 days were selected for each sample of training data. Brief introductions to the DCGAN and WGAN are given in the Appendix.

4.1 DCGAN

The current implementation of the DCGAN, contains four blocks of strided transpose 1D convolutions and strided 1D convolutions for the generator and discriminator respectively. Each convolution in the generator has an increasing stride, then the first 50 samples in the sequence length is taken. Furthermore, experiments have shown that, rather than the discriminator outputting a single value for each sample, a value is outputted for every point in the sequence, which is then averaged to determine a final value. Finally, binary cross entropy was used for the loss function, where real data were given a value of 1 and synthetic a label of 0. Hyperparameter tuning was not performed as this was an introductory implementation. All hyperparameters are described in Table 4 in the Appendix.

4.2 WGAN

Through experimentation, the WGAN was tested with multiple different architectures to help capture long term dependencies. Moving away from the DCGAN architecture, rather than increasing the dimensionality of a small noise vector, the noise vector was initialized to the same size as the desired synthetic data (50 days). Then, the noise vector was passed through generator with a 4-layered Temporal Convolutial Network (TCN) architecture. The TCN architecture was chosen to help capture long term dependencies. The critic (former discriminator) has a hybrid architecture made up of alternating TCN and Gated Recurrent Network (GRU) blocks, with a similar focus on long term dependencies. All hyperparameters are listed in Table 5 in the Appendix. Thorough hyperparameter tuning will be performed in future work.

4.3 C-WGAN

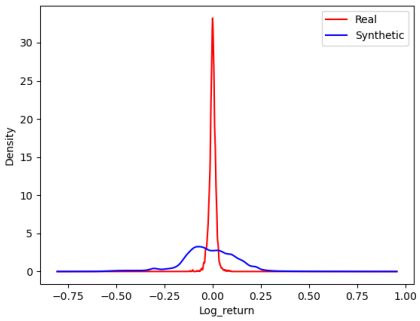
A further extension to the WGAN is to include a condition on the generated data [20]. To condition the WGAN, the previous 50 days of log returns is concatenated with the noise vector in the feature dimension for both the generator and the critic. The same architecture for the WGAN is used for the C-WGAN with the 4 layered TCN for the generator and the TCN/GRU hybrid for the discriminator. Hyperparameters are defined in Table 5 in the Appendix.

4.4 GAN Results

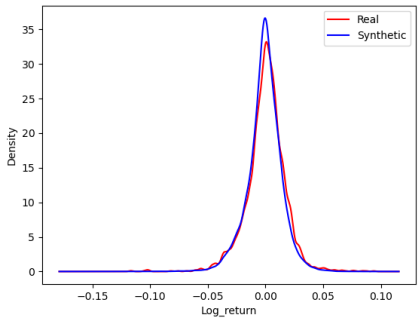
To evaluate the GANs, 2000 log returns (50 day interval) from the real stock and 2000 log returns generated by the GAN were sampled, and the mean (μ), standard deviation (σ), Inter-quartile range (IQR), skew, and raw kurtosis were measured. In addition, the Maximum Mean Discrepancy was measured between the real and fake samples [21]. As illustrated in Table 2, the WGAN is the best performing GAN with an MMD of 0.0004. Surprisingly, the C-WGAN performs slightly worse than the WGAN, as seen by the large skew. The underperformance is likely caused by the need to finetune the architecture and carefully condition the added complexity to the model. Unsurprisingly, the DCGAN performs the worst because its architecture is derived from image-based GANs, thus, they lack the capture of long term dependencies in the time series. Furthermore, Figure 4 displays the Kernel Density Estimate (KDE) plots of each GAN. As confirmed by Table 2, the WGAN most closely resembles the real data, while the DCGAN performs the worst. The large skew for the C-WGAN is also exemplified in Figure 4 as the main cause for the deviation from the real distribution.

Table 2: Statistical metrics based on sampling the real and generated data 2000 times.

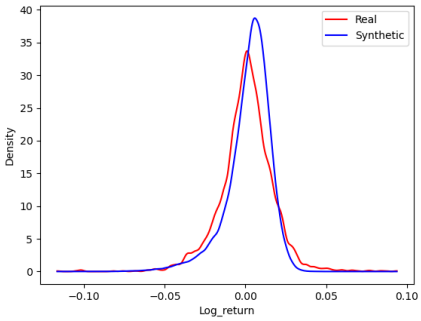
Data	μ	σ	*IQR*	*Skew*	*Kurtosis*	*MMD*
Real	0.0005	0.0160	0.0184	-0.3042	4.6254	0
DCGAN	-0.0162	0.1431	0.1699	-0.3892	3.9882	0.4999
WGAN	-0.0007	0.01453	0.0162	-0.3131	3.9447	0.0004
C-WGAN	0.0022	0.01327	0.01464	-0.9966	4.6560	0.0006



(a) DCGAN KDE



(b) WGAN KDE



(c) C-WGAN KDE

Figure 4: KDE Plots of the distributions between the real and synthetic data generated from each GAN, after sampling 2000 times.

5 Future Work and Conclusion

In the upcoming weeks, the research will focus on adding a second critic to the GAN, which is the N-HiTS model to be fooled. The GAN-based adversarial attack will be a targeted attack, similar to the TIM and C&W, where the target for the time series will be defined and the N-HiTS forecast will be judged based on the target with some loss function that has yet to be determined. However, given that 300 days of information is needed to make a prediction, either the C-WGAN's output sequence length will be changed to at least 300, or C-WGAN will repeatedly sample 50 days of log returns, where the condition will be the previously generated output. Furthermore, comparisons will be made between the adversarial C-WGAN in a black box and white box setting, determining the effectiveness of each attack setting. Moreover, hyperparameter tuning, in addition to experimenting with more complicated and novel GANs such as the TimeGAN and Sig-WGAN, will be performed [10], [20]. Current implementations of GANs do not have extensive hyperparameter or architectural tuning, limiting the possible effectiveness of these models.

Finally, if time permits, some baseline black-box attacks will be implemented to compare the black-boxed version of the adversarial C-WGAN. Currently, only white-box methods are implemented as comparative baselines, but this is not necessarily a fair comparison to the black-boxed adversarial C-WGAN, if the performance is not up to the same level. Furthermore, other time-series related adversarial attacks, such as the Smooth Gradient Method, should be implemented [8].

Appendix

5.1 Baseline Attacks on Individual Stocks

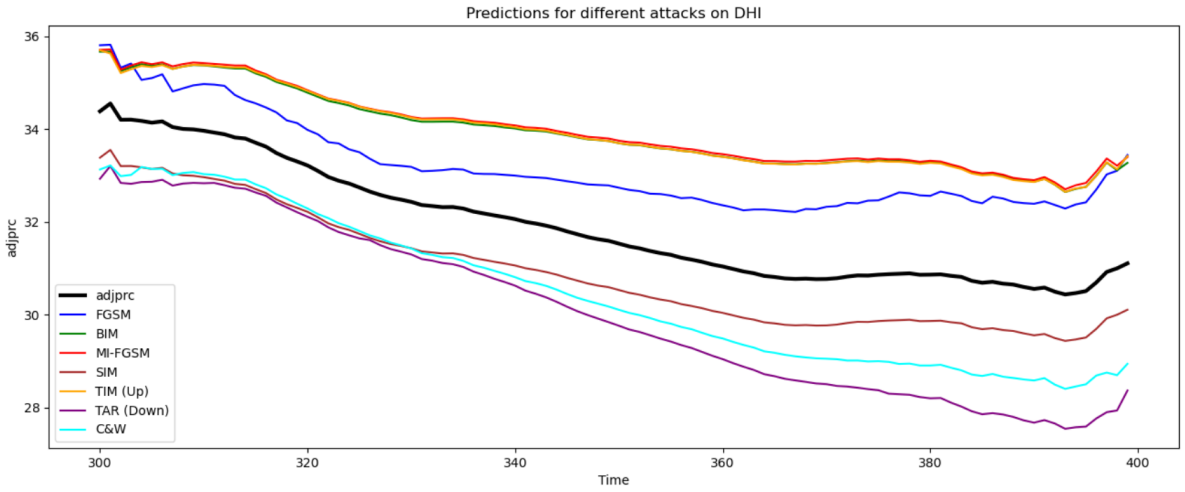


Figure 5: Various attacks performed on the stock *DHI*. Attacks were performed on the first 400 days of the recording.

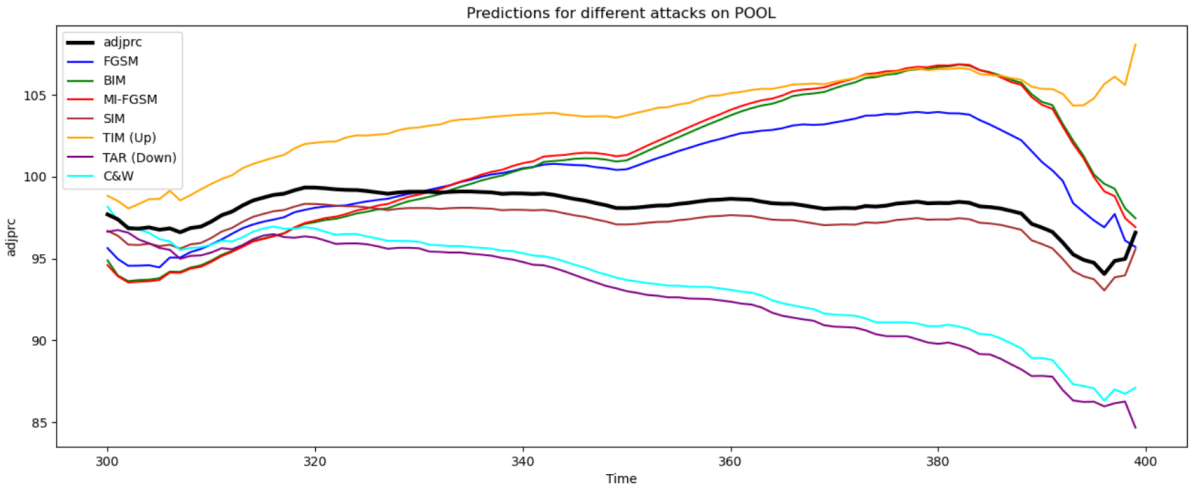


Figure 6: Various attacks performed on the stock *POOL*. Attacks were performed on the first 400 days of the recording.

5.2 GAN Explanations

5.2.1 DCGAN

The DCGAN is an extension of the original GAN, where the generator and discriminator are made using transpose and normal convolutions [22]. The generator takes a small noise vector as input, and passes the vector through multiple strided transpose convolutions in order to increase the dimensionality to match the real data [22]. The discriminator typically performs the opposite transformation, using strided convolutions to reduce the dimensionality to a single value, representing the probability of the sample being real or fake [22].

5.2.2 WGAN

In practice, vanilla GANs are typically notoriously hard to train, due to the possible discontinuities between the divergence the generator optimizes and the generators parameters [23]. Thus, the objective function was changed to the approximate the Wasserstein distance which, under mild assumptions, is continuous everywhere [23]. Specifically, the objective function is defined as:

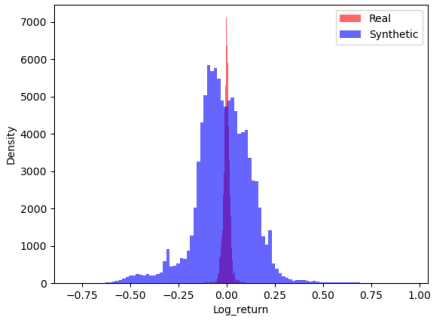
$$\min_G \max_D \mathbb{E}_{x \sim p_{data}(x)}[D(x)] - \mathbb{E}_{z \sim p_z(z)}[D(G(z))] \qquad [23]$$

However, the original implementation of the WGAN enforced the Lipchitz constraint using gradient clipping, which can be problematic due to vanishing gradients and optimization difficulties [23]. Therefore, a gradient penalty term was added to the objective function to overcome the optimization difficulties, defined as

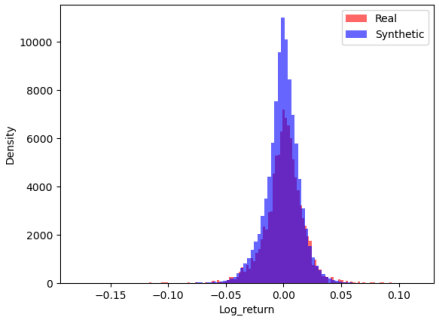
$$L = \mathbb{E}_{\tilde{x} \sim P_g}[D(\tilde{x})] - \mathbb{E}_{x \sim P_r}[D(x)] + \lambda \cdot \mathbb{E}_{\hat{x} \sim P_{\hat{x}}}[(||\nabla_{\hat{x}} D(\hat{x})|| - 1)^2]$$

where $\tilde{x}$ is a synthetic sample, and λ controls the magnitude of the gradient penalty [23].

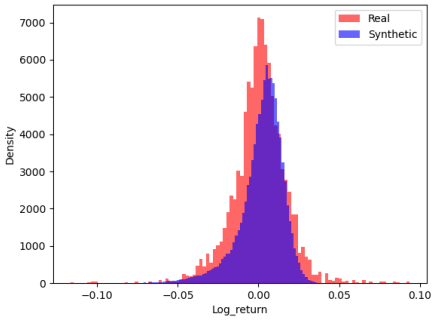
5.3 Extra GAN Results



(a) DCGAN Histogram



(b) WGAN Historgram



(c) C-WGAN Histogram

Figure 7: Histograms of the distributions between the real and synthetic data generated from each GAN.

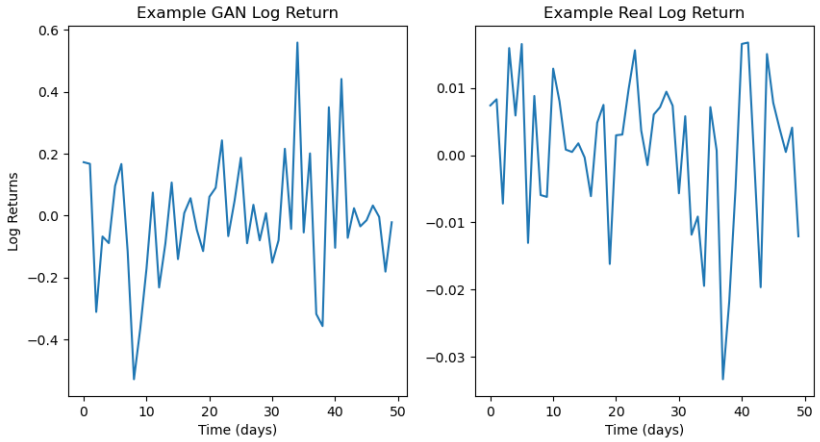


Figure 8: An example DCGAN return compared to a real return.

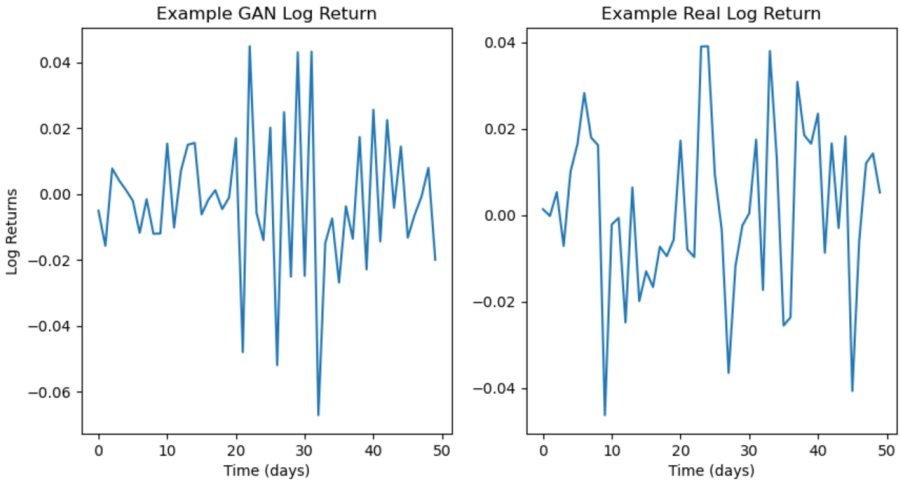


Figure 9: An example WGAN return compared to a real return.

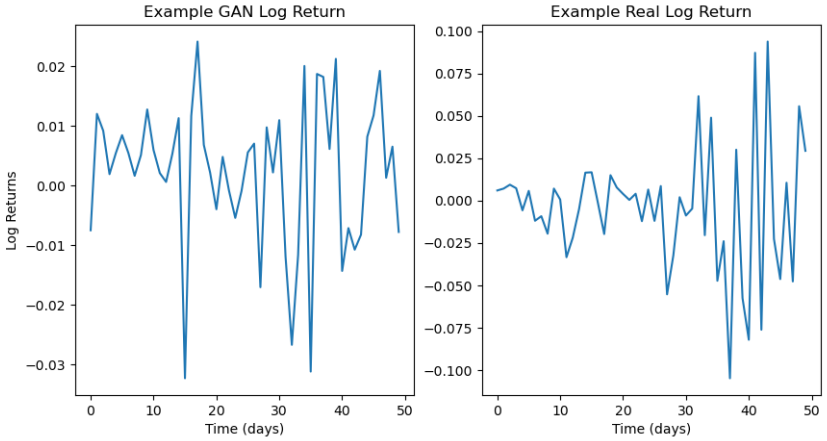


Figure 10: An example C-WGAN return compared to a real return.

5.4 Model Hyperparameters

Table 3: Hyperparameters used for the N-HiTS model

Hyperparameter	*Value*
Learning Rate	10^{-3}
Weight Decay	10^{-4}
Hidden Size	64
Batch Normalization	True
Early Stopping	True
Max/Min Encoder Length	300
Max/Min Prediction Length	50
Batch Size	500

Table 4: Hyperparameters used for the DCGAN model

Hyperparameter	*Value*
Batch Size	32
Noise Dimension	8
Generator Learning Rate	10^{-4}
Discriminator Learning Rate	10^{-5}
Betas	(0.5, 0.99)
Dropout	0.25
Batch Normalization	True
Leaky ReLU Negative Slope	0.2
Number of Layers	4
Kernel Size	5
Hidden Size	(128, 256, 128, 64)

Table 5: Hyperparameters used for the WGAN and C-WGAN models

Hyperparameter	*Value*
Batch Size	32
Number of Critic Iterations	5
λ	5
Generator Learning Rate	10^{-4}
Critic Learning Rate	10^{-4}
Betas	(0, 0.9)
Dropout	0.2
Batch Normalization	False
Leaky ReLU Negative Slope	0.2
Generator Number of TCN Layers	4
Generator Hidden Size	(128, 256, 128, 64)
Generator TCN Kernel Sizes	(3, 5, 5, 3)
Generator TCN Dilation	(1, 2, 4, 8)
Critic Number of Layers	5
Critic Number of TCN Layers	3
Critic Number of GRU Layers	2
Critic Hidden Size	(128, 256, 256, 128, 64)
Critic TCN Kernel Sizes	(3, 5, 5)
Critic TCN Dilation	(1, 2, 4)

References

[1] Y. Dong et al., "Boosting adversarial attacks with momentum," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, 2018. DOI: 10.1109/CVPR.2018.00957.

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014. DOI: 10.48550/arxiv.1412.6572.

[3] Z. Shen and Y. Li, "Temporal characteristics-based adversarial attacks on time series forecasting," *Expert systems with applications*, vol. 264, no. 125950, 2025. DOI: 10.1016/j.eswa.2024.125950.

[4] N. Ghaffari Laleh et al., "Adversarial attacks and adversarial robustness in computational pathology," *Nature communications*, vol. 13, no. 1, 2022. DOI: 10.1038/s41467-022-33266-0.

[5] M. Gallagher et al., "Investigating machine learning attacks on financial time series models," *Computers & security*, vol. 123, no. 102933, 2022. DOI: 10.1016/j.cose.2022.102933.

[6] P.Rathore et al., "Untargeted, targeted and universal adversarial attacks and defenses on time series," in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9207272.

[7] J.Zhang et al., "Are time-series foundation models deployment-ready? a systematic study of adversarial robustness across domains," 2025. DOI: 10.48550/arxiv.2505.19397.

[8] G. Pialla et al., "Time series adversarial attacks: An investigation of smooth perturbations and defense approaches," *International journal of data science and analytics*, vol. 19, no. 1, pp. 129–139, 2025. DOI: 10.1007/s41060-023-00438-0.

[9] J. Chen et al., "Mag-gan: Massive attack generator via gan," *Information sciences*, vol. 536, pp. 67–90, 2020. DOI: 10.1016/j.ins.2020.04.019.

[10] J. Chen et al., "Time series data augmentation for energy consumption data based on improved timegan," *Sensors*, vol. 25, no. 2, 2025. DOI: 10.3390/s25020493.

[11] L. Wang and K.-J. Yoon, "Psat-gan: Efficient adversarial attacks against holistic scene understanding," *IEEE transactions on image processing*, vol. 30, no. 9524508, 2021. DOI: 10.1109/TIP.2021.3106807.

[12] H. Sun, S. Wu, and L. Ma, "Adversarial attacks on gan-based image fusion," *Information fusion*, vol. 108, no. 102389, 2024. DOI: 10.1016/j.inffus.2024.102389.

[13] S. Baluja and I. Fischer, "Adversarial transformation networks: Learning to generate adversarial examples," 2017. DOI: 10.48550/arxiv.1703.09387.

[14] C. Challu et al., "N-hits: Neural hierarchical interpolation for time series forecasting," 2022. DOI: 10.48550/arxiv.2201.12886.

[15] StockAnalysis. "A list of all stocks in the s&p 500 index." (2025), [Online]. Available: https://stockanalysis.com/list/sp-500-stocks/ (visited on 06/15/2025).

[16] J. Zhang, Y. Lai, and J. Lin, "The day-of-the-week effects of stock markets in different countries," *Finance research letters*, vol. 20, pp. 47–62, 2017. DOI: 10.1016/j.frl.2016.09.006.

[17] J. Sen and S. Dasgupta, "Adversarial attacks on image classification models: Fgsm and patch attacks and their impact," 2023. DOI: 10.48550/arxiv.2307.02055.

[18] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," 2016. DOI: 10.48550/arxiv.1607.02533.

[19] I. J. Goodfellow et al., "Generative adversarial networks," 2014. DOI: 10.48550/arxiv.1406.2661.

[20] J.Liao et al., "Sig-wasserstein gans for conditional time series generation," *Mathematical finance*, vol. 34, no. 2, pp. 622–670, 2024. DOI: 10.1111/mafi.12423.

[21] C. Esteban, S. L. Hyland, and G. Rätsch, "Real-valued (medical) time series generation with recurrent conditional gans," 2027. DOI: 10.48550/arxiv.1706.02633.

[22] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015. DOI: 10.48550/arxiv.1511.06434.

[23] I. Gulrajani et al., "Improved training of wasserstein gans," 2017. DOI: 10.48550/arxiv.1704.00028.