Financial Data Augmentation Using Generative Models

Olivier Filion
ofilion@andrew.cmu.edu

Michael Agaby
magaby@andrew.cmu.edu

Owen Wang
owenw@andrew.cmu.edu

Abstract

Deep generative models have seen great improvements in recent years. This project builds on recent work in time-series generation with generative adversarial models (GANs) to create a synthetic dataset of stock price data. The model we implement using transformers is able to more closely match the distribution of the real data and better capture relationships between the features when compared to other generative techniques.

1 Introduction

A current trend in machine learning is to build deep models that require huge amounts of data to train effectively. One application of these models that has received a lot of attention is financial trading. The algorithmic trading market is predicted to reach $18.8 billion by 2024 (MarketsandMarkets). However, availability of historical data is limited. For example, to model the daily movement of one of the most famous indexes in the American stock market, the Standard and Poor's 500 (S&P 500), there are only a little over 16,000 data points available since the index's conception in 1957 (Valetkevitch, 2013). Moreover, it is not always advised to use data from too far in the past due to the risk of a distribution shift caused by evolving economic trends (Izzo et al., 2021).

We approach this challenge with data augmentation. We employ generative adversarial networks (GAN's) specialized for time-series data to generate synthetic time-series for daily returns of the S&P 500 that behave similarly to the original data and that can be used for training. We compare the synthetic data to the original dataset using various summary statistics such as mean and correlation between the generated time-series features, as well as PCA and t-SNE visualizations.

To build our dataset, we downloaded historical price data for the S&P 500 (Yahoo!, 2022b) and the Chicago Board of Exchange's volatility index (VIX) (Yahoo!, 2022a) which contain open prices, low prices, high prices, closing prices, adjusted closing prices and volume traded on each trading day between April 1st, 1960 and December 31st, 2021 for the S&P 500 and between February 1st, 1990 and December 31st, 2021 for the VIX. We then processed this data to obtain a dataset of stationary features, which include the relative changes between consecutive closing prices (returns), between the open price and closing price, lowest price and high price on a same day, the volume traded normalized using the mean and standard deviation of the previous 30 days, the VIX index and the relative change in the VIX index between the market open and close. We then restrict our dataset to the period between March 1st, 1995 and December 31, 2019. In summary, we are working with a dataset containing 6254 days of data, each with 7 stationary features.

2 Background

In the midway report, we implemented a baseline model to tackle this problem. The model we considered was a standard GAN, similar to the one first proposed by Goodfellow et al. (2014). We found that the time series generated looked similar to real data but lacked diversity, which was evident

Preprint. Under review.

when plotting the generated data against the real data using PCA and t-SNE. Also, the model failed to respect some basic constraints of the real data. For example, the fact that the lowest price on a day cannot be greater than the opening price.

In section 4, we show how we improve our model to beat the baseline. In section 5, we compare the data generated by our improved model to this baseline.

3 Related work

3.1 Generative models

Generative models can learn the distribution of features in their training data and are able to produce new examples, preserving the characteristics of the original dataset. Deep generative models achieve this goal using neural networks.

There are two classes of deep generative models that have received the most attention in recent years: variational autoencoders (VAEs) (Kingma and Welling, 2014) and generative adversarial networks (GANs) (Goodfellow et al., 2014). In VAEs, the encoder outputs a probability distribution over the latent space and the decoder attempts to reconstruct the original example using a sample drawn from that latent distribution (Doersch, 2016). In GANs, two models compete wherein the generator tries to create samples that the discriminator is unable to tell apart from real data.

Multiple variations of the basic GAN framework have been proposed to tackle various issues. For example, conditional GANs (CGANs) (Mirza and Osindero, 2014) can generate data conditioned on a specific class, Wasserstein GANs (WGANs) (Arjovsky et al., 2017) improve the stability of training of traditional GANs, and TimeGANs (Yoon et al., 2019) and time series GANs (TSGANs) (Smith and Smith, 2020) build on previous ideas to generate realistic time series data. Finally, Wiese et al. (2020) propose QuantGAN, which uses TCN with skip connections and data preprocessing to generate financial time series data, tackling the some challenges associated with return distributions, such as the fact that they are usually not normal (heavy tails), display volatility clustering (cycles of activity) and a levarage effect (negative correlation between volatility and return) (Wiese et al., 2020).

Of particular interest to our work is the TimeGAN model proposed by Yoon et al. (2019). Their model introduces a latent space to facilitate the generation of time series. They jointly train an autoencoder and the GAN to minimize a combination of three losses, the usual GAN unsupervised loss, the reconstruction loss of the autoencoder and a supervised loss to help generate time series that respect the time dependencies of the original data. The various components of their model are implemented using gated recurrent units (GRUs) (Cho et al., 2014). We expand on their work by using models based on attention that have achieved great results in recent years on various sequence modelling tasks.

3.2 Transformers

One problem that often arises in sequence models is the incapacity of the system to remember longer input sequences (Kyunghyun Cho, 2014). Attention mechanisms, which allow for access to the state of each element of the input sequence, such as the one proposed by Dzmitry Bahdanau (2014), were introduced to help alleviate this problem with great success. Transformers (Vaswani et al., 2017) were introduced to eliminate the use of recurrent models in such tasks. They make use of multi-headed self attention in order to provide context to each element of the input sequence and thus do not need to process the data in order (allowing them to be trained more efficiently). Vaswani et al. (2017) show that transformer models can outperform RNN+attention models with significantly lower training cost.

3.3 Data augmentation

Data augmentation refers to various procedures used to make a dataset larger and more diverse by generating new data, which is usually used to reduce overfitting. It has been used extensively in vision tasks. Krizhevsky et al. (2012) use transformations such as cropping and varying the intensity of the pixels on ImageNet data to reduce overfitting. Antoniou et al. (2017) show that GANs can be used to improve the test accuracy of classifiers on image data where basic transformations were previously performed on the data. Nishizaki (2017) improved the performance of a model trained on acoustic data using VAEs.

There has also been some work done on augmenting time series datasets, including financial time series. Fons et al. (2020) present some basic transformations that can be applied to time series data such as time warping, reversing, and adding Gaussian noise. These methods were shown to increase the accuracy of a binary classifier on the returns of the S&P 500. Naritomi and Adachi (2020) apply a WGAN with a generator and a discriminator designed with Long-Short-Term-Memory networks (LSTMs) (Hochreiter and Schmidhuber, 1997) to generate high-frequency stock time series data and improve a binary classifier. Most of the work done on generating financial time series data has been focused on generating returns between time periods. We expand on this by also generating other relevant features.

4 Methods

4.1 Data preprocessing

We use min-max scaling on our data. For each feature j, we calculate $a_j = \max_{i,t} \mathbf{x}_{i,t,j}$ and $b_j = \min_{i,t} \mathbf{x}_{i,t,j}$ and scale data points with

$$x_{i,t,j} \leftarrow \frac{x_{i,t,j} - b_j}{a_j - b_j} \tag{1}$$

By restricting our output to the range $(0, 1)$, we will enforce logical constraints, like the non-positivity of open to low moves, which was an issue we had in our baseline model.

4.2 Network architecture

The basic structure of the network is based on Yoon et al. (2019). The block diagram as presented in their paper is in figure 1. The network consists of 5 components, an embedder, a generator, a recovery function, a discriminator and a supervisor. The embedder maps the real sequences to the latent space, the recovery function learns to decode the latent space and recover real sequences, the generator maps the noise vectors to the latent space, the discriminator learns to distinguish between real and fake data in the latent space, and the supervisor learns to make next-step predictions on the latent space sequences.

In the Yoon et al. (2019) paper, they implement all 5 components with a GRU. We improve on their model by replacing the GRU by a more recent transformer-like multi-headed self-attention architecture. The use of transformers is motivated in part by the fact that the time series that we generate are fairly long (30 timesteps), and some tasks could require even longer sequences. Transformers tend to perform better on longer time series than RNN's, since their attention mechanism allow them to better model long-term dependencies.

The transformer architecture we implemented consists of 3 encoder layers (see figure 2). The encoder layers first apply layer normalization (Ba et al., 2016) to the input, followed by multi-headed self attention (3 heads) with a skip connection (He et al., 2015). Layer normalization is applied again and the output is passed into a feedforward network with another skip connection. The feedforward network passes input through 2 linear layers with GELU (Hendrycks and Gimpel, 2016) activation between, then applies dropout (set to 0.3). Finally the output is passed through a linear layer in order to reshape it to the appropriate size.

We use layer normalization, which computes the mean and variance over the layer rather than the batch, as it is often preferred in sequence tasks. We use skip connections as well, as they help combat vanishing/exploding gradient problems and also provide the model with a way to combine representations at different levels to gain a more top-down understanding, similar to the way humans might process a sequence.

The transformer computes a positional encoding which allows the model to make use of sequence order. The positional encoding we used is identical to the one first proposed by Vaswani et al. (2017). Let t be the position in the sequence, the positional encoding $\mathbf{p_t} \in \mathcal{R}^d$, where d is the embedding dimension, is computed as:

$$\mathbf{p_t} = \begin{cases} \sin(\omega_k \cdot t) & i = 2k \\ cos(\omega_k \cdot t) & i = 2k + 1 \end{cases}$$

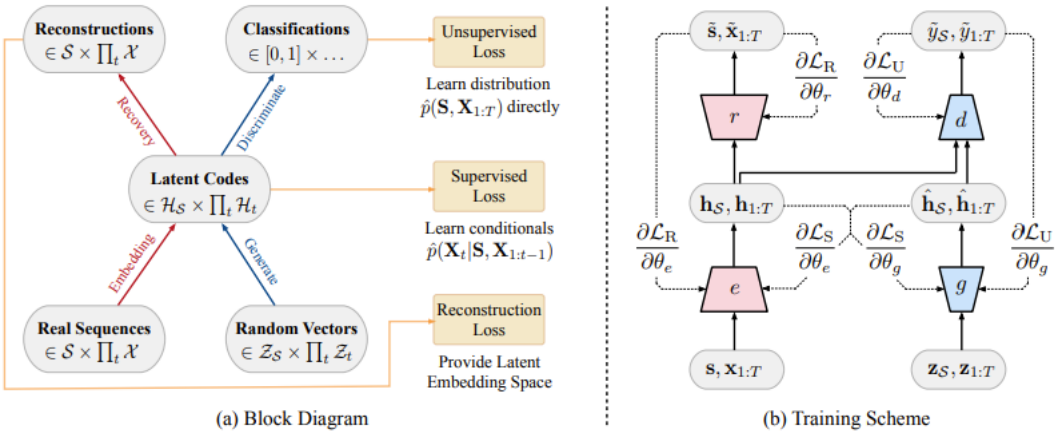


Figure 1: Diagram of the basic TimeGAN model and its training scheme. The model learns to map the random noise vectors and the real time series to a latent space before recovering them. Figure taken from Yoon et al. (2019).

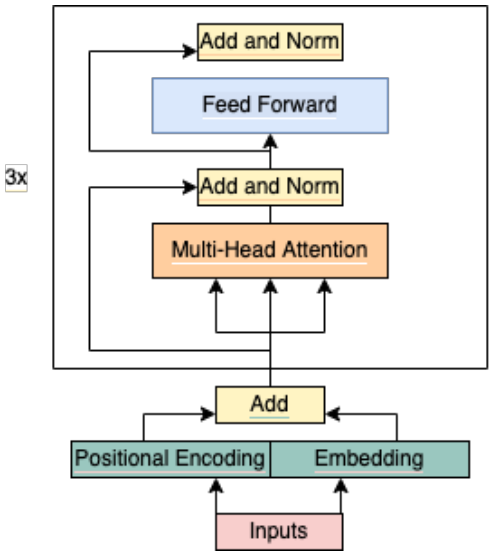


Figure 2: Diagram of the transformer encoder architecture

where

$$\omega_k = \frac{1}{10000^{2k/d}}$$

and d is the dimension of the encoding. We also tried using learned time representations such as Time2Vec (Kazemi et al., 2019) but did not find a significant change in performance.

The transformer passes the input through an embedding layer which applies layer normalization followed by a linear transformation with dropout and ReLU activation and sums it with the positional encoding before passing this through the 3 encoder layers.

This architecture is used for the generator, the embedder, the recovery function and supervisor. The discriminator additionally passes the output from the transformer through a linear layer with GELU activation, before passing it through another linear layer with output dimension 1 and finally through a sigmoid function.

		Return	Open-Close	Open-Low	Open-High	Norm. Vol.	VIX	VIX Op.-Cl.
Mean	Real	3.72×10^{-4}	3.29×10^{-4}	-6.65×10^{-3}	6.32×10^{-3}	3.18×10^{-2}	1.97×10^{1}	-2.82×10^{-3}
	GAN	1.48×10^{-2}	1.28×10^{-3}	2.69×10^{-2}	-2.07×10^{-2}	1.73×10^{-1}	-1.94×10^{1}	3.60×10^{-3}
	GRU	1.38×10^{-2}	1.14×10^{-3}	-7.68×10^{-3}	7.71×10^{-3}	6.46×10^{-2}	2.01×10^{1}	-2.52×10^{-3}
	Trans	6.83×10^{-4}	6.14×10^{-4}	-6.85×10^{-3}	6.72×10^{-3}	-3.88×10^{-2}	1.94×10^{1}	-4.07×10^{-4}
Std. Dev.	Real	1.16×10^{-2}	1.11×10^{-2}	8.20×10^{-3}	7.26×10^{-3}	1.07×10^{0}	8.10×10^{0}	5.82×10^{-2}
	GAN	6.07×10^{-2}	4.66×10^{-2}	3.86×10^{-2}	3.19×10^{-2}	4.46×10^{0}	3.41×10^{1}	2.29×10^{-1}
	GRU	9.73×10^{-3}	1.04×10^{-2}	7.93×10^{-3}	7.08×10^{-3}	1.13×10^{0}	7.94×10^{0}	4.29×10^{-2}
	Trans	1.13×10^{-2}	1.08×10^{-2}	7.18×10^{-3}	6.47×10^{-3}	1.02×10^{0}	7.50×10^{0}	4.90×10^{-2}

Table 1: Comparison of the mean and standard deviation of the 7 features in the data set in the real data and the generated data. The GAN rows use data generated with a traditional GAN (baseline from midway report), the GRU rows use data generated with the GRU TimeGAN implementation from Yoon et al. (2019) and the Trans rows use data generated with our transformer-based TimeGAN.

4.3 Training

We reimplemented the training code for the timeGAN model available in the ydata-synthetic data repository (Santos et al., 2022), which is based on the Yoon et al. (2019) paper, in PyTorch (Paszke et al., 2019).

The training is done in three steps. The first step trains the autoencoder portion of the model (the embedder and the recovery function). The autoencoder is trained to minimize the mean-squared error between the original data and the reconstructed sequences.

$$\mathcal{L}_R(\mathbf{X}, \hat{\mathbf{X}}) = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \|\mathbf{x}_{i,t} - \hat{\mathbf{x}}_{i,t}\|_2^2 \tag{2}$$

where N is the size of the dataset, T is the number of timesteps in the time series, $\mathbf{X}$ is the original data and $\hat{\mathbf{X}}$ is the reconstructed data.

The second training step aims to train the supervisor to predict the next step in the latent space. The original data $\mathbf{X}$ is passed through the embedder to obtain its latent space representation $\tilde{\mathbf{X}}$ and then through the supervisor to obtain next-step predictions $\tilde{\mathbf{Y}}$. This step also optimizes a mean-squared error loss.

$$\mathcal{L}_S(\mathbf{X}, \tilde{\mathbf{Y}}) = \frac{1}{N(T-1)} \sum_{i=1}^{N} \sum_{t=1}^{T-1} \|\tilde{\mathbf{y}}_{i,t} - \tilde{\mathbf{x}}_{i+1,t}\|_2^2 \tag{3}$$

Finally, the third step of the training algorithm attemps to jointly train the two previously mentioned losses, $\mathcal{L}_R$ and $\mathcal{L}_S$ and the unsupervised discriminator loss, $\mathcal{L}_U$, which we take to be binary cross entropy. This steps also includes a moment loss which attempts to match the mean and variance of the generated data to those of the real data.

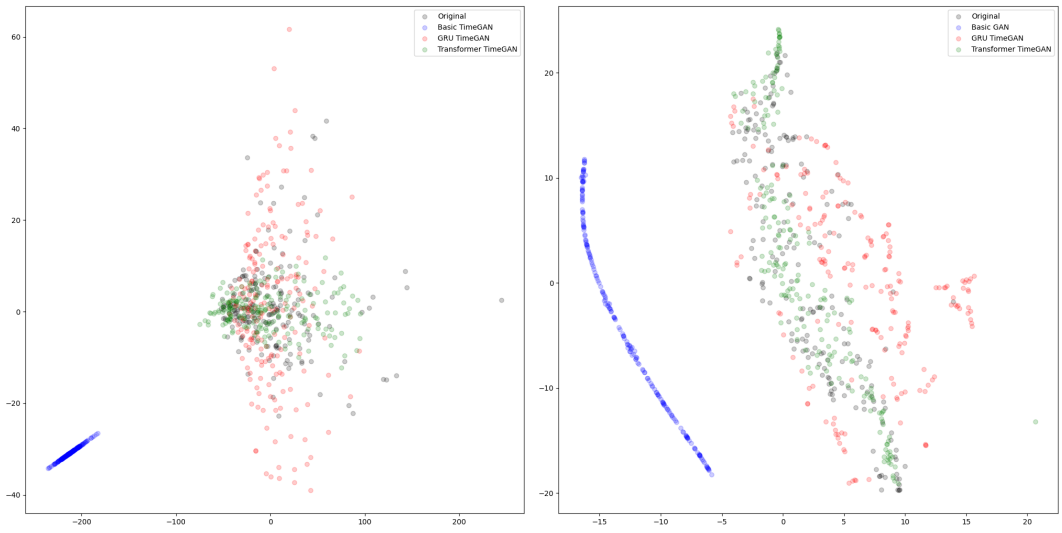
The optimization is done with an Adam optimizer (Kingma and Ba, 2014) at all three steps, with a learning rate of 5×10^{-4}, a batch size of 128, a hidden dimension of 30 and a noise dimension of 32 (noise sampled from a $\mathcal{U}[0, 1]$ distribution). We train the model for 1000 iterations.

4.4 Sampling

Another improvement we implemented for this specific task is in our sampling technique when generating new data. A major issue encountered in our baseline model, and for GAN's in general, is sample diversity (Arjovsky et al., 2017). That is, we may tend to see a mode collapse, where the generator learns to output very few plausible examples. To try to counter this, we train the model as specified above $M > 1$ times. Then, to generate K new time series, we generate $\frac{K}{M}$ examples from each TimeGAN model trained. This allows the generated dataset to be more diverse even when each individual generative model trained is not fully diverse. We set $M = 5$. This also helps reduce the dependence on the random initialization of the neural networks.

5 Results

We present our results by comparing some summary statistics and visualizations of the real data and the generated data from our baseline GAN, GRU TimeGAN Yoon et al. (2019) and our Transformer-



(a) PCA (b) t-SNE

Figure 3: PCA and t-SNE visualizations of the distribution of the data. Black points are the original data, blue points are the data generated by our baseline GAN, red points are generated from the GRU TimeGAN and green points are generated from our transformer TimeGAN.

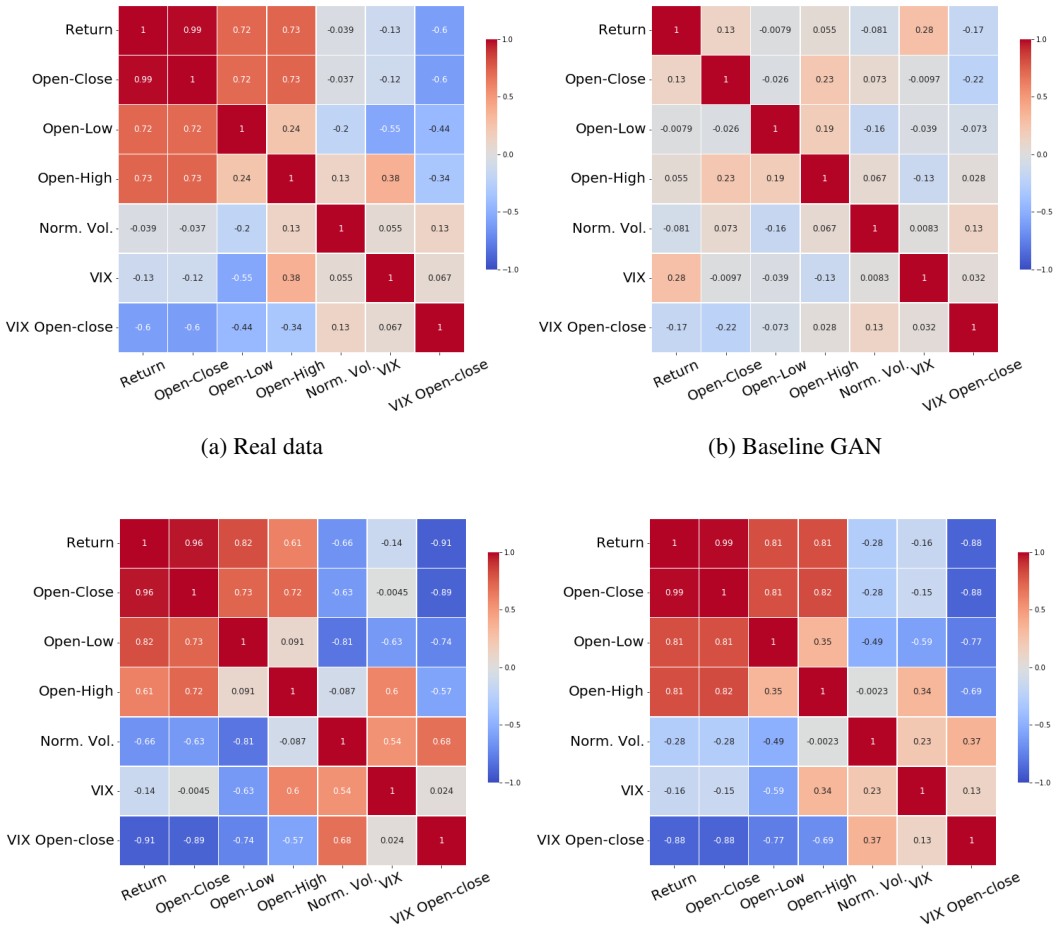
TimeGAN. Table 1 presents the mean and standard deviation of the features in the different datasets. We notice that the while the mean of the features for our model are not always aligned with the mean of the real data, they seem to be in the right order of magnitude more often than for the original TimeGAN. For example, the mean return of the TransformerTimeGAN is 6.83×10^{-4} compared to the mean of the true returns which is 3.72×10^{-4}. GRU TimeGAN produces returns with a mean of 1.38×10^{-2} which is significantly further from the true value. Also, the standard deviations for our model are consistently lower than those of the real data. However, they are relatively close which suggests that our model does a decent job of capturing the variability of the features in the data.

Figure 3 shows PCA and t-SNE visualizations of 200 data points sampled from each dataset. These visualizations clearly show that the GAN models designed specifically for time series generate much more realistic data. Both the GRU and TransformerTimeGANs generate data that seem to cover most of the distribution of the real data in the PCA plot, although the GRU-based model generates many more outliers. The t-SNE plot shows that the transformer generates data that much better covers the true distribution.

Figure 4 presents correlation visualizations between the features in the datasets. Both TimeGAN models seem prone to generating data with stronger correlations than what we observe in the real data. However, the GRU-based model also creates strong correlations that don't actually exist (between the normalized volume and the open-close move of the VIX) and ignores correlations that do exist (between the open-high move and the open-low move). In general, although TransformerTimeGAN also presents some issues with capturing the exact correlations that exist in the data, it appears to do a better job than the GRU TimeGAN. For example, there is very little or no correlation in the true data between the returns and the normalized volume and the data generated by both models shows a negative correlation between these two features, but the false correlation is much stronger in the data generated by the GRU TimeGAN. This shows that our TransformerTimeGAN does a better job at capturing the true distribution of the data.

Figure 5 shows an example of a generated data over 30 days for each feature. Both TimeGAN models produce examples that are within the same bounds as the original data.

These results are in line with our expectations since transformer models are becoming more and more dominant in sequential tasks and our sampling technique allows the model to generate more varied data than it would otherwise, by training the models multiple times.



(a) Real data

(b) Baseline GAN

(c) GRU TimeGAN

(d) Transformer TimeGAN

Figure 4: Comparison of the correlation matrices of the features between the real and generated data

6 Discussion and analysis

6.1 Analyzing results

The PCA, t-SNE, and correlation plots show that the TransformerTimeGAN model is able to generate data with a similar distribution to true financial data and that could potentially be used for training predictive models. The t-SNE plot also shows that the GAN models specifically designed for time-series generation significantly outperform the classic GAN architecture. Attention models, and in particular transformers, are becoming the standard in a broad range of time-series problems, and our results are in line with this trend. The data produced by the TransformerTimeGAN seems to be more similar, and thus more useful for training, to the original data than the data generated by the GRU TimeGAN. In particular, the t-SNE plot shows that the distribution of the data generated by the TransformerTimeGAN is significantly more similar to the original data than the GRU TimeGAN. T-SNE attempts to preserve the local structure of data in a non-linear fashion. The fact that the TransformerTimeGAN seems to be more similar to the true data in the t-SNE plots may indicate the transformer's ability to better learn long-range dependencies that allow it to better conserve complex structure in the data compared to the GRU TimeGAN.

Figure 5 allows us to qualitatively assess an example of generated data. We see that in particular the data generated by TransformerTimeGAN could conceivably be mistaken for true data and thus could be useful for training a predictive model.

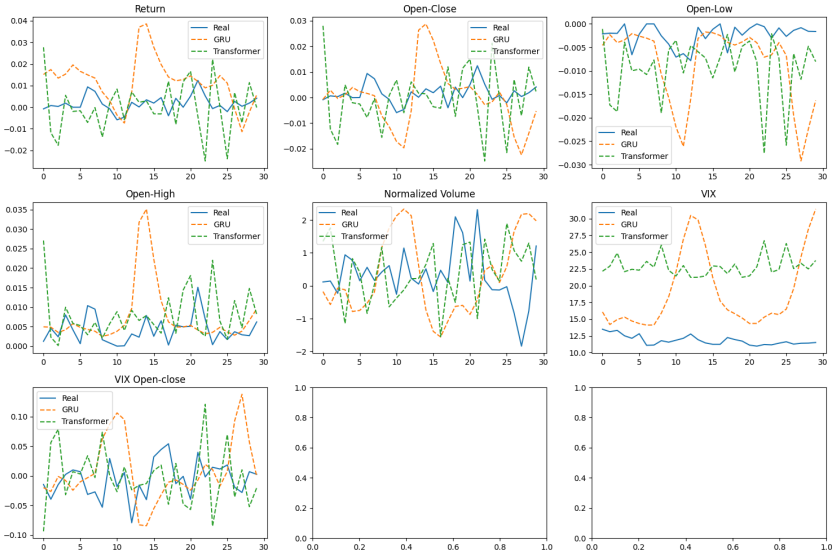


Figure 5: Examples of time series generated by both GRU and Transformer TimeGAN, compared to a sample from the real dataset.

6.2 Limitations and future directions

Although the generated data is qualitatively and quantitatively similar to true data, it is unclear whether using it to augment financial datasets would lead to better results. Due to time-constraints, we did not test whether or not it improved predictive model results. This could be tested by evaluating the performance of multiple models in a binary prediction task such as whether next day's close is higher or lower than the current day when trained using this augmented dataset and when trained only on the original data. Such synthetic samples could also be used for backtesting, where a predictive model is tested on historical data, in situations where historircal data is limited.

A limitation of our model that was mentioned in section 5 is that it is prone to exaggerate the strength of correlations between features. Features that are not inherently correlated (like the move between open and close which has to be correlated with the return by definition) will usually at most be weakly correlated in the real data due to the amount of noise in financial data. Adding higher moments to the moment loss or some loss function directly linked to the correlation between features during training could help solve alleviate this issue.

Also, our model makes a pretty strong assumption that should be noted. We use min-max scaling on our input data and constrain our output to the range $(0, 1)$, which has the advantage that logical constraints will be respected (e.g. move from open price to high price must be greater than or equal to 0). However, it adds additional constraints to our generated data that we might not want, such as the fact that no generated time series will have returns higher or lower than historical maximums.

Additionally, TransformerTimeGAN could also be used to generate synthetic time-series data in other domains than finance, such as climate data for example.

7 Teammates and work division

Work was split among all three team members and the main idea of the project was contributed by everyone. Olivier wrote the TimeGAN implementation with Michael's help. Owen helped debug it and run experiments. Michael implemented the transformer code. All three worked on the report.

8 Access to our code

Our code is available in a public GitHub repository.

References

Antreas Antoniou, Amos Storkey, and Harrison Edwards. 2017. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Carl Doersch. 2016. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*.

Yoshua Bengio Dzmitry Bahdanau, Kyunghyun Cho. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Elizabeth Fons, Paula Dawson, Xiao-jun Zeng, John Keane, and Alexandros Iosifidis. 2020. Evaluating data augmentation for financial time series classification. *arXiv preprint arXiv:2010.15111*.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems*, 27.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus).

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Zachary Izzo, Lexing Ying, and James Zou. 2021. How to learn when data reacts to your model: performative gradient descent. In *International Conference on Machine Learning*, pages 4641–4650. PMLR.

Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. 2019. Time2vec: Learning a vector representation of time.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Diederik P Kingma and Max Welling. 2014. Auto-encoding variational bayes.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

Dzmitry Bahdanau Yoshua Bengio Kyunghyun Cho, Bart van Merrienboer. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.0473v7*.

MarketsandMarkets. Algorithmic trading market by trading type (forex, stock markets, etf, bonds, cryptocurrencies), component (solutions and services), deployment mode (cloud and on-premises), enterprise size, and region - global forecast to 2024. *MarketsandMarkets*.

Mehdi Mirza and Simon Osindero. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Yusuke Naritomi and Takanori Adachi. 2020. Data augmentation of high frequency financial data using generative adversarial network. In *2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, pages 641–648. IEEE.

Hiromitsu Nishizaki. 2017. Data augmentation and feature extraction using variational autoencoder for acoustic modeling. In *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1222–1227. IEEE.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Francisco Santos, Luis Portela Afonso, Gonçalo Martins Ribeiro, Fabiana, Arunn Thevapalan, Umberto, CeShine Lee, Guan Wang, and Archit Yadav. 2022. ydata-synthetic. `https://github.com/ydataai/ydata-synthetic.git`.

Kaleb E Smith and Anthony O Smith. 2020. Conditional gan for timeseries generation. *arXiv preprint arXiv:2006.16477*.

Caroline Valetkevitch. 2013. Key dates and milestones in the s&p 500's history. *Reuters. Viitattu*, 15:2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer. 2020. Quant gans: Deep generation of financial time series. *Quantitative Finance*, 20(9):1419–1440.

Yahoo! 2022a. Cboe volatility index (^vix) charts, data & news. Accessed on 2022-03-03.

Yahoo! 2022b. S&p 500 (^gspc) charts, data & news. Accessed on 2022-03-03.

Jinsung Yoon, Daniel Jarrett, and Mihaela Van der Schaar. 2019. Time-series generative adversarial networks. *Advances in Neural Information Processing Systems*, 32.