Time Series Synthesis using Generative Adversarial Networks

Jan Mark Dannenberg , Lydia Y. Chen[1] , Aditya Kunar[2] , Zilong Zhao[2]
TU Delft

Abstract

Generative Adversarial Networks are widely used as a tool to generate synthetic data and have previously been applied directly to time-series data. However, relying solely on the binary adversarial loss is not sufficient to ensure the model learns the temporal dynamics of the data. TimeGAN [1] introduces an additional reconstruction and supervised loss to tackle this issue and efficiently and effectively capture the step-wise dependencies of the data. We have been able to reproduce results similar to those of the original TimeGAN paper [1], after fixing several issues in the provided implementation by the authors of TimeGAN. Furthermore, we propose two novel improvements to TimeGAN. Firstly we updated its implementation to Tensorflow 2 to ensure compatibility across systems. Secondly by re-weighting the training iterations over the three training phases we are able to reduce the overall training time up to 29% and produce results equal or better compared to the benchmark.

1 Introduction

Nowadays data is one of the most valuable resources for a lot of companies. The business insights gained are extremely useful and can be crucial to success. However, big data often impedes personal privacy [2]. This creates a challenge to find a way to adhere to government regulations such as the European General Data Protection Regulation (GDPR) [3].

One of the solutions that addresses this problem is the generation of synthetic data that statistically represents real data. General Adversarial Networks [4] (GANs) are a way to achieve this generation of synthetic data. GANs are networks of a discriminator and a generator which compete with each other in a min-max game in order to learn. The discriminator tries to separate real data (from the training set) from fake data (generated by the sequence generator). The generator tries to fool the discriminator by creating data samples from random noise. By competing against each other they both try to improve and this can result in an optimised generator that can create very realistic looking synthetic data.

Time-series data is dependent on time and often has step-wise dependencies; variables can have a certain relationship across time. For example, stock data contains correlations between variables over time. These temporal dynamics should be preserved when learned by a GAN in order to generate good synthetic data. There have been several approaches to use GANs to generate time-series data such as RCGAN [5], C-RNN-GAN [6] and WaveGAN [7] [8]. However, these approaches rely solely on the standard GAN loss, which is not sufficient to guarantee that the model will learn the temporal dynamics of the data efficiently and effectively [1].

This paper is about utilizing GANs to generate time-series data and mainly centers around the TimeGAN algorithm [1] which was recently developed for this purpose. The first goal of this research is to reproduce the results achieved for TimeGAN in the original TimeGAN paper [1]. Furthermore, we will propose novel improvements to the existing algorithm to ensure compatibility across systems and to reduce the overall training time. We found that by re-weighting the iterations used for training over the different training phases of TimeGAN the overall performance can be increased. We are able to achieve speedups up to 29% without loss in quality of results. Choosing a smaller speedup with corresponding weights, based on computational resources available, can even produce better results compared to the original TimeGAN configuration benchmark.

This paper is divided into five sections. First, the *Related work* section will dive deeper into the components and underlying concepts of TimeGAN and alternative approaches. The method used to conduct this research will be discussed in the *Methodology* section. The *Reproducing results* section will present the results obtained by reproducing the results of the TimeGAN paper [1]. Then we will propose two novel improvements to the TimeGAN algorithm in the following section. Finally we will conclude by summing up the results of the research and suggesting follow-up research.

2 Related work

In this section we will cover the basic concepts of Generative Adversarial Networks [4] and illustrate how this framework has been utilized for time-series synthesis [6] [5]. Then we will give a preliminary of TimeGAN [1] by going over the different components and losses it depends on. The training process and its different stages will be discussed as well, since this is relevant for our proposed improvements.

2.1 Generative Adversarial Networks

Generative Adversarial Networks consist of two main components; a generator and a discriminator [4]. Both are implemented by a neural network. The generator takes random noise as input and outputs synthetic data. This synthetic data is mixed with real training data and used as input for the discriminator. The job of the discriminator is then to tell if the data presented to it is either real or fake. In this manner the discriminator and the generator act as adversaries as the generator tries to fool the discriminator and the discriminator tries to learn whether the provided data is real or fake. This creates a min-max game where two parties try to respectively minimize and maximize a loss function. By using this adversarial loss to train both components the generator can create high fidelity synthetic data.

2.2 Generative Adversarial Networks for time-series data

The architecture of GANs has been applied directly to generate synthetic time-series data. C-RNN-GAN [6] has used the concept of recurrent neural networks, implemented by LSTM networks [9], for the generator and discriminator. In this way the outputted synthetic data relies on the previous steps and is thus generated recurrently. Another similar approach is Recurrent Conditional GAN (RCGAN) [5] which has a very similar architecture compared to C-RNN-GAN, but has some minor differences. These differences include no longer relying on previous input but adding a dependency on additional input [5].

2.3 Preliminary of TimeGAN

The TimeGAN algorithm [1] was chosen for this research because it was published fairly recently. In addition, the results the authors presented were significantly better than that of alternative approaches such as C-RNN-GAN [6], RCGAN [5] and WaveGAN [7]. According to the TimeGAN authors relying solely on the binary adversarial feedback of the sequence discriminator of the GAN framework may not be enough for the sequence generator to learn the temporal dynamics of the data. Therefore, TimeGAN combines the concepts of generative adversarial networks, auto-regressive models for sequence prediction and time-series representation learning [1] to create a model that efficiently and effectively can create high fidelity synthetic data.

Besides consisting of a sequence discriminator and a sequence generator, TimeGAN makes use of two other main components: an embedding and recovery function [1]. The first maps from feature into latent space, which allows the adversarial components to train in a lower-dimensional space and learn the step-wise dependencies in the data. The recovery function allows to convert back into the feature space. Note that the discriminator and generator train (and thus take input and output) using the latent vector representations. These mapping functions (embedding and recovery) are respectively implemented by a recurrent neural network and a feed-forward network.

Another addition to the normal GAN framework is the use of a supervised loss for the generator. The generator receives actual data in the latent space and has to generate the next step (also in the latent space). This ensures that the generator learns the step-wise dependencies and can generate synthetic data with similar step-wise transitions.

TimeGAN relies on three different loss functions. First there is the reconstruction loss for the embedding and recovery functions which ensures that data is efficiently encoded and decoded into and from the latent vector space [1]. Secondly, the unsupervised adversarial loss for the generator and discriminator forces the generator to create realistic data sequences. Lastly, the supervised loss where the generator is trained in closed-loop mode ensures that the generator learns the temporal dynamics of the time-series data, by forcing the generator to generate realistic next-step sequences. This supervised loss is also applied to the embedding function to preserve the temporal dynamics in the encoding of real sequences. A block diagram of the training of TimeGAN can be found in Figure 1. In this figure the way to generate synthetic data after training is illustrated as well, where the generator produces sequences from random noise in the latent vector space which is then translated to the feature space by the recovery function.

The TimeGAN algorithm learns over three different phases of training as shown in Figure 2. First, only the embedding and recovery networks are trained in the *Embedding Phase*. In the second phase, the *Supervised Phase*, the generator is trained with the supervised loss only. In the last phase all the components are trained in joint fashion. In this *Joint Phase* the algorithm jointly learns to encode, iterate and generate time-series data. Since this last phase combines the first two phases and adds the additional adversarial loss training it is the most time-consuming phase. All phases are trained over the same amount iterations, as all phases have equal weighting in the original TimeGAN implementation [1].

Compared to the previously mentioned alternative approaches utilizing the GAN framework, like RCGAN [5] and C-RNN-GAN [6], TimeGAN naturally has computational overhead by adding components to the architecture. By optimizing additional losses for these components (embedding loss, recovery loss, supervised loss) it logically follows that TimeGAN differs in training time.

3 Methodology

This section will illustrate the approach used in order to reproduce the results from the TimeGAN paper [1] and to evaluate our improvement. We will first cover the metrics used to evaluate the performance of TimeGAN and to reproduce the results of the original paper [1]. The metrics used to measure the performance of the novel improvements will also be discussed in the first part of this section. Then we will go into detail on the different data sets used in our research. Finally, we will discuss the implementation and method used for reproducing the results of TimeGAN, which is also the base for our improvements to the existing algorithm.

Exactly reproducing the results of the TimeGAN paper [1] is not very likely, since Generative Adversarial Networks inherently have a sense of randomness [4]. One would not be able to have the same trained GAN model after two sepa-

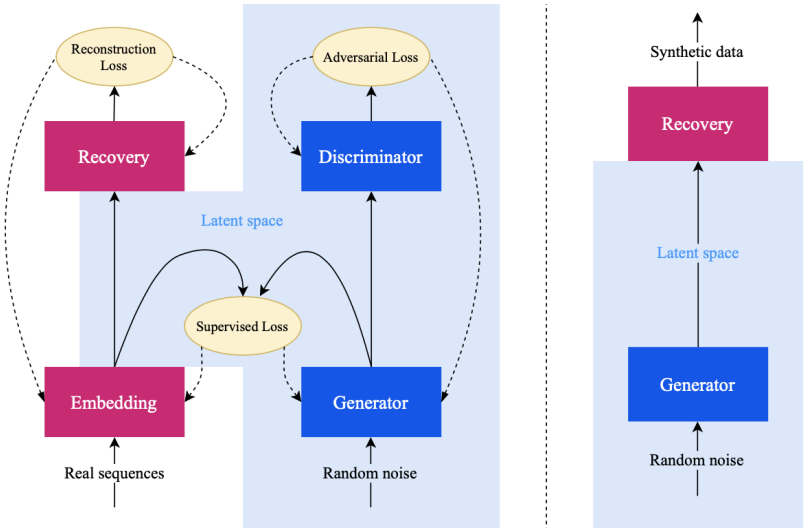

Figure 1: Block diagram of training (left) and generating (right) synthetic data for TimeGAN. Dashed arrows indicate to which components loss functions are applied to. Normal arrows indicate the data flow through and out of the different components.

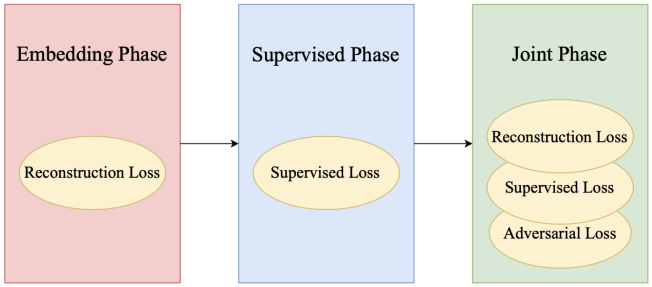

Figure 2: Different phases of training of TimeGAN.

rate runs of training, even when using exactly the same hyper parameters/configuration. Therefore the goal is not to have exactly the same results, but rather results that are similar and in the same range.

3.1 Metrics

To evaluate the TimeGAN algorithm [1] we will resort to the use of a discriminative and a predictive score, as proposed in the TimeGAN paper [1]. The discriminative score is used to indicate the similarity between the synthetic and original data. We first label each sequence in the original data set real and label each sequence in the synthetic data set as not real. We then train a classification model to distinguish between the the real and synthetic data as a standard supervised machine learning task. The classification error is then calculated on a test set, which gives a measurement of similarity between the synthetic and original data. The classification model should be compatible with time-series data, therefore it is implemented by optimizing a 2-layer LSTM.

This discriminative score is a measurement that could be applied to any GAN in general, since it reflects how good the synthetic generated data is. The predictive score is more unique to time-series data GANs as it shows how well the model captured the temporal dynamics/step-wise dependencies of the data. This metric is based on the *train on synthetic, test on real* framework [10] [6]. To calculate this metric we first train a a sequence-prediction model on the generated data set. After training this model (like the discriminative score implemented by optimizing a 2-layer LSTM) we evaluate its performance on the original data set. We obtain the predictive score by calculating the mean absolute error.

Note that the discriminative and predictive score are both obtained by calculating the mean of iterating over multiple post-hoc trained networks. For both the discriminative and predictive score holds the lower the better.

Since we are interested in reducing the time for the improvement we also consider the time it takes to train the TimeGAN algorithm over the different phases as a metric. Besides the total time the training takes, for each phase we calculate the time metric by timestamp of the end of that phase minus the timestamp of the start of that phase.

Besides using these three metrics we also use PCA [11] and t-SNE [12] analyses to visualize how well the synthetic data distribution resembles that of the original data set in 2-dimensional space. By visualizing the results we get a better insight in how well the model learns the distribution of the data, rather than just a numeric metric. These visualization also allow us to observe how a model evolves over any arbitrary number of training iterations.

3.2 Data sets

We will consider three different time-series data sets to evaluate the performance of TimeGAN. They differ in the combination they have of various properties like periodicity, regularity of time steps, level of noise and the correlation across time and features. Note that these data sets have also been used in the TimeGAN paper [1]:

- **Sines:** This data set consists of multivariate sinusoidal sequences which have different frequencies and phases. Therefore this data is periodic, continuous-valued and

the features are independent of each other. This data set is generated and takes a dimension value and a sample number as its arguments.

- **Stocks**: This data set consists of the Google stocks data from 2004 to 2019 [13]. Opposed to the sines data set this set is not periodic and features are not independent of each other. The data set has six features, consisting of high, low, volume, opening price, closing price and adjusted closing price.
- **Energy**: The last data set we use is the UCI Appliances energy prediction data set [14]. This data set has a higher level of noise compared to the previous two. It also consists of a lot more features compared to the stock data set and thus has a higher number of dimensions.

Using these different data sets will allow to evaluate TimeGAN on different domains with a variety of characteristics.

3.3 Implementation

To reproduce the results of the TimeGAN paper [1] we make use of an execution script which takes all hyper parameters of TimeGAN and a data set as input, trains the TimeGAN using these parameters and provided data set and subsequently generates synthetic data and uses it to calculate the previously mentioned metrics. These measures are then saved to text files and image files for respectively the numeric metrics and visualization analyses.

To evaluate the proposed improvement we further extended this execution script by adding additional hyper parameters needed as input and adding additional output for calculating the time metrics. All implementations used in this research are coded based on the Tensorflow platform [15]. For the implementations mentioned in the following two sections using Tensorflow 2 we were able to run our experiments using a GPU. For the older implementations using Tensorflow we were forced to use CPUs, as the GPU support in combination with CUDA (on the machine availabe for our research) only applied to Tensorflow 2.2+.

4 Reproducing results

To evaluate the performance of the TimeGAN algorithm, our research goal was to reproduce the results achieved for TimeGAN in Table 2 of the research paper [1]. As discussed in the *Methodology* section, these results consist of a discriminative and a predictive score. These metrics are calculated for the trained TimeGAN model on different domains. Originally, the authors of TimeGAN evaluated its performance on four different data sets in the research paper [1]: sine, stock, energy and an events data set. As mentioned in the TimeGAN paper [1], the events data set consisting of lung cancer pathways is private and is not accessible for our research. Therefore reproducing has been limited to the first three data sets, which are also mentioned in the *Methodology* section. This section further discusses the difficulties we faced reproducing the results and the differences across multiple implementations of TimeGAN we used. Finally, the obtained results are presented and compared to the original TimeGAN paper results [1].

4.1 Observations on two implementations

Reproducing the results achieved for TimeGAN by the original authors was complicated. The authors provided a code base with their TimeGAN implementation on GitHub [16]. However, running this implementation did not provide any good results for any of the data sets. Figure 3 shows the results that have been visualized for the stock data set and as can be seen the synthetic data is not diverse and does not fit the real data distribution at all at any number of epochs. In Figure 4 the discriminative and predictive score have been plotted every 5000 training iterations. We observed that there is no learning curve and the scores appear to be random and validated this with the research of others on TimeGAN [17]. Writing our own implementation of TimeGAN would not be feasible, due to lack of experience and time.

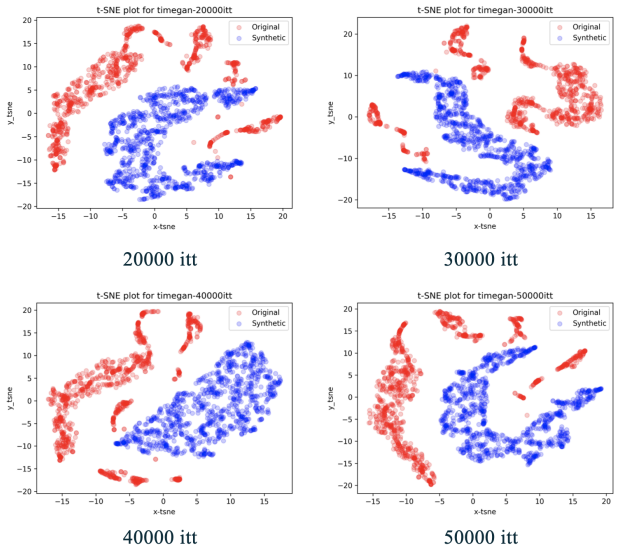

Figure 3: t-SNE analyses for original TimeGAN implementation at 20000, 30000, 40000 and 50000 epochs for stock data set.

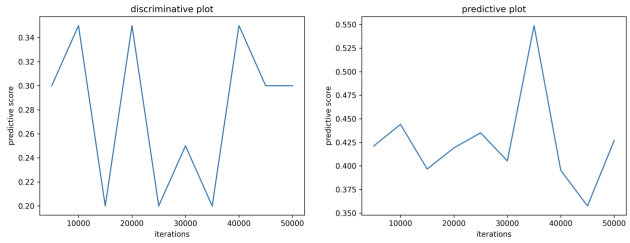

Figure 4: Discriminative and predictive score plotted every 5000 epochs for original TimeGAN implementation for stock data set.

Another implementation of TimeGAN is provided by YData on GitHub [18] [19]. This implementation used Tensorflow 2, just like the orignal implementation of the authors of TimeGAN [1]. However, this code base is highly optimised for and centered around the stock data set in the input

used for the general GAN/loss parameters. We have adapted this implementation to fit in our execution script and produce the metrics needed for reproducing. We found that this model of TimeGAN suffers from over-fitting when training more than 25.000 - 30.000 epochs as can been observed from the visualisation of the results for the stock data set in Figure 5.

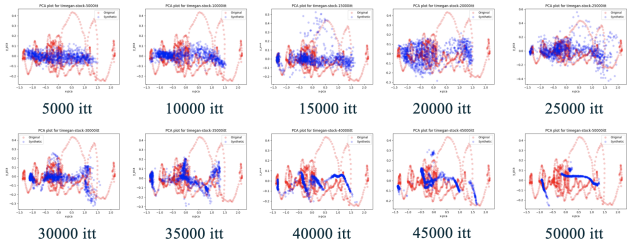

Figure 5: PCA anlyses for alternative implementation for stock data set every 5000 epochs. (Blue is synthetic data, red is original data)

From here on, the *original implementation* will refer to the code base provided by the authors of the TimeGAN paper and the *alternative* implementation will refer to the code base provided by YData [19]. We found that the model obtained by the alternative implementation is able to capture the temporal dynamics of the data quite effectively as we found that the predictive score of the synthetic data is within fairly close range of the predictive score of the original data and that of the experiments of the TimeGAN paper [1]. However, the discriminative scores obtained were not quite as consistent and similar compared to the results achieved in the paper, but still far more accurate than the results produced by the original implementation. The visualization of the synthetic data in Figure 6 of those results shows that the model has indeed learned the data distribution of the real training data.

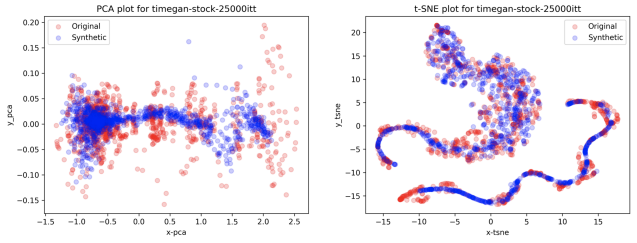

Figure 6: PCA anylysis (left) and t-SNE anlysis (right) of alternative implementation for stock data set.

4.2 Explanations on differences

To further investigate the difference in performance between the alternative and original implementation and to be able explain the difference in results, we came up with three hypotheses we put to the test.

Hypothesis 1. First of all, the implementation of TimeGAN depends on several hyper parameters such as number of layers, batch size and hidden dimensions. In our research we could have been using badly optimised hyper parameters and as the configurations used were not mentioned in the TimeGAN paper or in its supplemental materials, this could have been the cause for the bad performance. To test this hypothesis we first tried running TimeGAN with a lot of different configurations and evaluated the performance with the discriminative and predictive score metrics. Through the joint effort with [17], we found that the parameter search can not improve the performance of the original implementation. Indeed, authors of TimeGAN later on confirm that the correct hyper parameters are provided in the original implementation.

Hypothesis 2. Secondly, upon further investigation and comparison between the original and alternative implementation we found that the loss functions used for the adversarial loss of the generator and discriminator differ. The original implementation used the sigmoid cross entropy loss function whereas the alternative implementation used binary cross entropy loss function. Loss functions play a major role in GANs and can greatly affect the performance of a GAN and could thus be a possible cause for the bad results. However, this appears to be no more than a Tensorflow version difference. In the version used by the original implementation the binary cross entropy loss function is not available and in the version used by the alternative implementation the sigmoid cross entropy loss function is not available. After looking into the source code implementation of both those loss functions, we found that they are equivalent and this difference was not the cause for the bad results.

Hypothesis 3. Lastly, the original implementation did not seem to take the input time information into account, whereas the alternative implementation did use this input. Although provided in all the components of the original implementation as input T, this information was not used in any of these networks. When further investigating this with the authors of the TimeGAN paper [1] and an independent researcher, we found that due to insufficient code review a pull request to update the original implementation to Tensorflow 2 had been incorrectly accepted. Continuing from this pull request the time information was not longer taken into account in all of the five components of TimeGAN, which was the core cause for the poor results. After reverting these changes, we were not only able to reproduce results close to the results in the paper [1], but also significantly better than the results obtained by the alternative implementation. In Table 1 the discriminative and predictive scores are shown for respectively the (unfixed) original implementation, the alternative implementation and the fixed original implementation and the results of the original TimeGAN paper [1].

4.3 Final results

Using the fixed original implementation was fairly simple for the sine and stock data set. For these first two data sets the authors provided the hyper parameters in the Github repository of TimeGAN [16]. However, for the energy data set they did not provide any configuration they used for their research. When tweaking all the hyper parameters of TimeGAN, we discovered that the number of layers used for each component of the algorithm has the most impact when working with higher dimensional data, such as the energy data set. Using trial and error we jointly found that TimeGAN performed best on the energy data set when using 6 as input parameter for the

	Original implementation	Alternative implementation	Fixed implementation	*TimeGAN paper*
Discriminative score (Lower the better)	0.250	0.161	**0.141**	*0.102*
Predictive score (Lower the better)	0.323	0.038	**0.039**	*0.038*

Table 1: Discriminative and predictive scores for stock data set of respectively the original (unfixed) original implementation, the alternative implementation and the fixed original implementation and the results from the TimeGAN paper.

number of layers [17], although not providing results similar to the TimeGAN paper [1] for the discriminative score.

In the end, we were able to successfully reproduce results reasonably similar to the results of the TimeGAN paper [1] with the fixed original implementation. However, we found that results can be quite inconsistent and tend to fluctuate. This variance in results is also depicted in Figure 8 and will be touched upon again in the *Proposed improvements* section. The results of reproducing are showcased next to the original results from the TimeGAN paper [1] in Table 2.

		Sine	Stock	Energy
Discriminative score	Our results	0.036	0.141	0.402
(Lower the better)	*TimeGAN paper*	*0.011*	*0.102*	*0.236*
Predictive score	Our results	0.097	0.039	0.252
(Lower the better)	*TimeGAN paper*	*0.093*	*0.037*	*0.273*

Table 2: Discriminative and predictive scores for sine, stock and energy data set compared to results of TimeGAN paper.

We learned that in this field of research providing the hyper parameters/configuration used is equally important as providing the original code used to be able to reproduce results. Therefore, we added the hyper parameters used for our results in the `README.md` of our published code base. Furthermore, we experienced that code review is crucial and should be taken seriously to ensure the quality and correctness of the code.

5 Proposed improvements

In this section our proposed improvements will be illustrated and discussed. We propose two novel improvements to the existing TimeGAN algorithm [1]. The first improvement only concerns compatibility and does not alter the TimeGAN algorithm. The second improvement centers around the computational overhead and longer total training time due to the architectural changes of TimeGAN compared to alternative approaches. For both improvements we made use of and built upon the original fixed TimeGAN implementation GitHub repository [16].

5.1 Tensorfow 2 compatibility

As a first improvement to the TimeGAN algorithm we converted the original code base to be compatible with Tensorflow 2. As mentioned in the *Reproducing Results* section the previous attempt to update the code base to Tensorflow 2 was incorrect as it did not produce the correct results any longer. After these breaking changes were reverted we converted all necessary parts in the code to Tensorflow 2 in the correct manner. All the layers and activation functions used for the components of TimeGAN were converted to the corresponding Tensorflow 2 implementation or an external library implementation.

This improvement allows the algorithm to be compatible with more modern systems which support the use of Tensorflow 2. Since Tensorflow 2 enables the developer to make use of multiple widely used libraries, such as Keras, TimeGAN can more easily be extended and improved in the future. In addition, also GPU support (e.g. using CUDA) is easier to setup using Tensorflow 2. For our research this meant we could now successfully run the correct original TimeGAN implementation with using a GPU.

5.2 Re-weighting the iterations over training phases

TimeGAN adds a several components and losses to the standard GAN framework. As mentioned in the Related Work section, a supervised loss is introduced to ensure that the generator learns the temporal dynamics of the data. Furthermore, TimeGAN makes use of an embedding and recovery network to translate sequence vectors from feature space into latent space and vice versa. This reconstruction network yields the reconstruction loss, which forces TimeGAN to effectively accurate encode and decode data.

Observation. These additional losses yield three different training phases (covered in the *Related work* section); *Embedding Phase*, *Supervised Phase* and *Joint Phase*. In Figure 2 the different phases are depicted with the corresponding losses trained in that particular phase. All these different phases use the same amount of epochs to train as they have equal weights (e.g. when provided 50000 iterations as a hyper parameter, all phases will train for 50000 iterations). The last phase is significantly more time-consuming than the first two phases, as can be observed from the time distribution in Figure 7. The authors of TimeGAN did not mention anything about training time of TimeGAN in the original TimeGAN paper [1] nor about the way the training iterations are proportioned over the different phases of training, which made it an interesting topic for our improvement research.

Hypothesis. We propose that the iterations each phase uses are weighted differently with respect to the others. Since the first two phases are relatively fast, we researched the impact in results of decreasing the iterations used in the last (most time-consuming) phase. Our hypothesis is that by distributing the reduced iterations of the last phase over the first two phases of training, we can compensate and achieve equal or

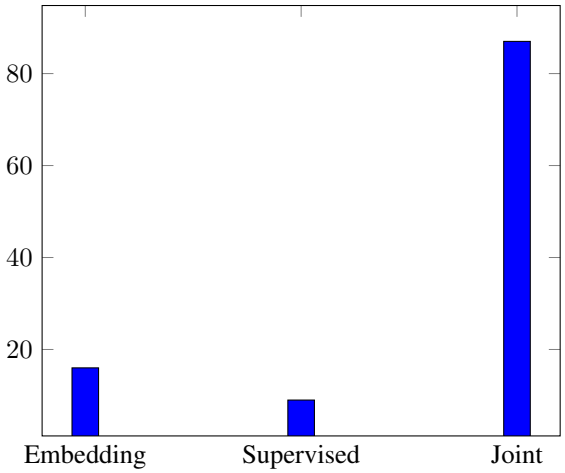

Figure 7: Time in minutes for TimeGAN for every phase of training for a run of 50000 iterations for each training phases.

better results. TimeGAN adds the reconstruction and supervised loss [1] (compared to alternative approaches [5] [6]) to ensure that the model learns the temporal dynamics of the data. By increasing the amount of iterations for the first two phases, that train these additional losses, we focus more on those additions and gives us reason to believe this will improve the overall performance of TimeGAN. Because the first two phases of training are less time-consuming the overall training time can be reduced. We set up 9 different configurations which all use 150000 iterations in total, but are distributed in different ways over the three training phases. All combinations have in common that they reduce the training iterations for the *Joint Phase*. These configurations are shown in Table 3.

	Embedding	**Supervised**	**Joint**
C1	72500	72500	5000
C2	70000	70000	10000
C3	67500	67500	15000
C4	65000	65000	20000
C5	62500	62500	25000
C6	60000	60000	30000
C7	57500	57500	35000
C8	55000	55000	40000
C9	52500	52500	45000
Benchmark	*50000*	*50000*	*50000*

Table 3: Different configurations for distributing 150000 iterations over the different training phases of TimeGAN.

Implementation. We implemented the distribution over phases by adding parameters to the TimeGAN algorithm which are used to specify the iterations for each training phase. To evaluate the performance of our proposed improvement we made use of the discriminative and predictive metrics. In addition, we measure the total training time of TimeGAN to show the overall decrease in training time. The total time training depends on the number of epochs and other hyper parameters such as number of layers. In addition, the system used and its capabilities hugely affect the overall training time. Therefore, we will measure the increase or decrease in time (and the other metrics as well) in percentages for the same hyper parameters (other than the given configurations for distributing the iterations).

As a benchmark to compare the results of our experiments to, we will use the results we obtained for reproducing the results of the TimeGAN paper [1] presented in Table 2. These results were obtained with 50000 iterations of training for all three phases. This benchmark is also represented in Table 3.

Evaluation results. We found that the overall performance can be increased by not equally distributing the iterations over respectively the *Embedding Phase*, the *Supervised Phase* and the *Joint Phase*. The results obtained for the different configurations can be found plotted in Figure 8 for the stock data set. Qualitative results for configurations *C1* up to *C5* are not consistent, especially the discriminative scores (e.g. they have a huge variance), due to lack of training on the *Joint Phase*. Furthermore, those results do not improve the overall performance of TimeGAN, even though they provide a enormous speedup for training the algorithm.

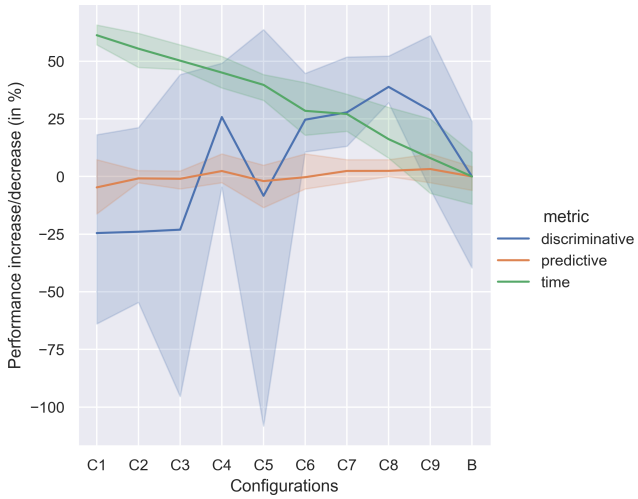

Figure 8: Increase or decrease in percentages of TimeGAN of three performance metrics compared to benchmark configuration for the different configurations of Table 3 for the stock data set.

However, configurations *C6* up to *C8* provide quite consistent results equal or better compared to the benchmark configuration. An average speedup for the training of TimeGAN ranging from 9% up to 29% can be achieved using these configurations. Moreover, the discriminative scores improve significantly and are more consistent compared to the benchmark. We believe that this is because the model starts overfitting after training the *Joint Phase* for more than 40000 iterations, which will cause more inconsistent and possibly worse results.

The predictive scores do not improve or decrease a lot for any of the configurations. This is mostly due to the fact that we train the *Supervised Phase* more in all configurations,

which can compensate for the reduced training in the *Joint Phase*. In the *Supervised Phase* the model learns to produce similar step-wise transitions, which is exactly what is evaluated in the predictive score.

Validation on sine data set. We evaluated the performance of our proposed optimal configurations on the sine data set to further validate our hypothesis. The speedups obtained ranged from 15% up to 24%, although the qualitative metric scores stayed roughly the same ranging from 5% decrease up to 4% increase. However, to achieve a maximal speedup configuration *C6* still remains the best option.

In addition, we applied linear regression to all obtained results to calculate the re-weighting factor to be used when using the benchmark results from Table 2 as a threshold to compare to. We used the linear equations to find the intersect point with the benchmark results. This confirmed that indeed the maximal speedup achieved while still having consistent and equal results compared to the benchmark is at a ratio of *2 : 2 : 1* respectively for the *Embedding*, *Supervised* and *Joint Phase*, which we also obtained by our results plotted in Figure 8. Using ratios that allow more iterations for the *Joint Phase* (up to *11 : 11 : 8*) can produce even better qualitative results compared to the benchmark but do not provide large speedups in training time. However, it should be noted that using linear regression analysis on our results is limited due to having relatively few data points.

Proposed improvement. To conclude, we found that our addition to TimeGAN not only reduces the training time but also significantly improves the overall performance of TimeGAN for using configurations *C6* up to *C8*. Therefore, we propose re-weighting the first two phases with respect to the last phase with a factor between 1.4 and 2 based on the available computational resources, in order to achieve optimal and consistent results for TimeGAN and reduce training time. This range of factors corresponds to respectively the evaluated configurations and calculated ratios, *C6* and *2 : 2 : 1* up to *C8* and *11 : 11 : 8*. Using re-weighting factor 2 will provide the greatest speedup but will not improve the overall qualitative results, whereas re-weighting factor 1.4 will provide a smaller speedup and overall better results. This improvement is not simply hyper parameter tuning, since in the original TimeGAN paper [1] the weighting of training iterations over the different training phases was not even noticed as a possibility.

6 Responsible Research

In the field of artificial intelligence reproducibility and transparency can be a major issue [20], since the results heavily depend on the algorithm code and the setup used. Especially in the case of Generative Adversarial Networks it can be even more complicated, since GANs inherently have a sense of randomness due to the fact they take random noise as input to generate synthetic data [4].

A major part of our research centered around reproducing results from the TimeGAN paper [1]. As discussed in the *Reproducing Results* section we found that it was hard to reproduce the results of the paper due to human error, insufficient code review on a pull request which caused the implementation to no longer produce good results, the fact that not all hyper parameters originally used were provided.

To prevent these issues to arise when reproducing this paper, we have published the entire code base containing all implementations used in this research. Furthermore, we provided all hyper parameters used in our experiments to produce results in this paper in the `README.md` of our repository. In addition, all data sets used in this research are publicly available [13] [14].

However, naturally GANs still have a lot of randomness to them, since to obtain the synthetic data the generator model still takes random noise as input [4], which will most likely result in slightly different results. The post-hoc classifier model for the discriminative score and the post-hoc sequence-prediction model for the predictive score are both models that still need to be trained on the outputted synthetic data set. This is also a cause that results may slightly differ.

7 Conclusion

To conclude, we found that TimeGAN [1] effectively learns the temporal dynamics of time-series data and is able to generate realistic looking synthetic data. While trying the reproduce results from the original paper [1], we found that the provided code base [16] was not producing the correct results. Different causes for the inconsistent results compared to an alternative implementation [19] were discussed, such as seemingly different loss functions and bad hyper parameters. However, due to human error a pull request had wrongly been accepted and the time information was neglected by different components of TimeGAN. After correcting necessary parts in the implementation, we were able to reproduce results similar to the results achieved in the TimeGAN paper [1] for the sine and stock data set. For the energy data set we could not produce similar discriminative score, as we were not able to find good hyper parameters for this data set. Providing not only the code but the hyper parameters used as well and sufficient code review is crucial to successfully reproduce results achieved elsewhere.

We proposed two separate novel improvements to the existing algorithm. Firstly, we updated the existing code base to Tensorflow 2, in order to be compatible across multiple systems and easier allow for improvements and extensions to TimeGAN in the future. Furthermore, we proposed the re-weighting of iterations across the three different phases of training of TimeGAN (*Embedding Phase*, *Supervised Phase*, *Joint Phase*) opposed to the normal ratio *1 : 1 : 1* with which the iterations are distributed. We found that the *Joint Phase* is the most time-consuming and by decreasing the iterations used for this phase and increasing the iterations over the other phases, the overall performance of TimeGAN can be improved. For a maximal speedup in training time (29%) the iterations should be weighted in a *2 : 2 : 1* ratio for respectively the *Embedding*, *Supervised* and *Joint Phase*. Other ratios up to *11 : 11 : 8* which allow slightly more training for the *Joint Phase* can be used, based on the computational resources available, to produce consistent and better or equal results compared to the original *1 : 1 : 1* ratio.

Further research could be conducted into the importance

of the different components of TimeGAN and swapping out several components, such as the reconstruction networks and supervised network. Another suggestion, we unfortunately could not work on due to a lack of time, is applying the concepts of differential privacy [21] to the TimeGAN algorithm.

References

[1] D. J. Jinsung Yoon and M. V. D. Schaar, "Time-series generative adversarial networks," *NeurIPS*, 2019.

[2] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," *2008 IEEE Symposium on Security and Privacy (sp 2008)*, p. 111–123, 2008.

[3] General Data Protection Regulation. European Commission. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679

[4] M. M. B. X. D. W.-F. S. O. A. C. Ian J. Goodfellow, Jean Pouget-Abadie and Y. Bengio, "Generative adversarial nets," *In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, 2014.

[5] S. L. H. Cristóbal Esteban and G. Rätsch, "Real-valued (medical) time series generation with recurrent conditional gans," 2017.

[6] O. Mogren, "C-rnn-gan: Continuous recurrent neural networks with adversarial training," 2016.

[7] J. M. Chris Donahue and M. Puckette, "Adversarial audio synthesis," 2019.

[8] M. B. Giorgia Ramponi, Pavlos Protopapas and R. Janssen, "T-cgan: Conditional generative adversarial network for data augmentation in noisy time series with irregular sampling," 2018.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, p. 1735–1780, 1997.

[10] J. J. Jinsung Yoon and M. van der Schaar., "Pate-gan: Generating synthetic data with differential privacy guarantees." *International Conference on Learning Representations*, 2019.

[11] F. B. Bryant and P. R. Yarnold, "Principal-components analysis and exploratory and confirmatory factor analysis," 1995.

[12] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research - 9 November*, p. 2579–2605, 2008.

[13] "Alphabet Inc. (GOOG) Stock Historical Prices Data," Jun 2021. [Online]. Available: https://finance.yahoo.com/quote/GOOG/history?p=GOOG&guccounter=1

[14] "UCI Machine Learning Repository: Appliances Energy Prediction Data Set." [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction

[15] "Tensorflow: An end-to-end open source machine learning platform." [Online]. Available: https://www.tensorflow.org/

[16] J. Yoon, "TimeGAN implementation on GitHub." [Online]. Available: https://github.com/jsyoon0823/TimeGAN

[17] Z. Z. A. K. Marcus Plesner, Lydia Y. Chen, "Federated Time-series Generative Adversarial Networks (FeTGAN)," 2021.

[18] "Build better datasets for AI with synthetic data." [Online]. Available: https://ydata.ai/

[19] Y. AI, "YData - YData-synthetic GitHub." [Online]. Available: https://github.com/ydataai/ydata-synthetic

[20] B. Haibe-Kains, G. A. Adam, A. Hosny, F. Khodakarami, L. Waldron, B. Wang, C. Mcintosh, A. Goldenberg, A. Kundaje, C. S. Greene, and et al., "Transparency and reproducibility in artificial intelligence," *Nature*, vol. 586, no. 7829, 2020.

[21] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, no. 3–4, p. 211–407, Aug. 2014. [Online]. Available: https://doi.org/10.1561/0400000042