Time series Synthesis using GANs - A take on DoppelGANger

Auke Schaap, Lydia Y. Chen, Zilong Zhao, Aditya Kunar
Delft University of Technology

June 27, 2021

Abstract

With a growing need for data comes a growing need for synthetic data. In this work we reproduce the results of DoppelGANger [16] in synthesising time series data with metadata. We identify a key issue in the comparison made in [16] of DoppelGANger to TimeGAN, RNNs, AR and HMM models, which creates a new avenue of time series synthesis using GANs. We show that not all results of [16] can be reproduced. We furthermore find that DoppelGANger does not adequately capture measurement-metadata correlations of our dataset. Sample size reduction is shown to be an effective tool to reduce training time while still attaining accurate results, and the key parameter S is tuned further. Finally we show that execution on CPU has similar training times as execution on GPU by [16], suggesting that the original code can be improved, and we release our version of the models ourselves, to enable easy reproduction. In closing points we shine light on possible future improvements that we were unable to test ourselves, and conclude that DoppelGANger is a promising model that opens the door to new unseen applications of GANs for time series synthesis.

1 Introduction

In the current society data is becoming more and more important everyday. With this growing need for data comes a need for privacy and data analysis, and hence data-driven research. A specific part of this research is the generation of synthetic data that closely resembles real data. An advantage of this type of data is for instance that there might be less privacy concerns and it is hence easier to publish. We specifically look at time series data of network traffic. From this data we can e.g. better predict when and where dynamic system resources are necessary.

Starting from this we might ask ourselves how to generate high fidelity synthetic time series data of network traffic? Recent research uses generative adversarial networks (GANs) to successfully generate high fidelity synthetic data. [23] builds a neural network that maps time series to vector embeddings, and after training outputs sequences of embeddings rather than samples. This model is however not designed for time series with additional metadata. [16] designs a model that does take metadata as input and which produces highly similar synthetic data with metadata. It performs better at this task than [23], hence based on this research we have the following research question:

> *Can the performance of DoppelGANger in sythesising time series data with metadata, and the comparison to other alternative methods, including autoregressive models, Hidden Markov models and RNNs, be reproduced?*

This question consists of the following subquestions which will be answered in this paper:

1. *Can we achieve the same results as [16] on existing data?*
2. *Does the model capture all correlations in the data?*
3. *In what way can we expand on current research?*

This paper therefore aims to reproduce the fidelity results of [16] on the existing and on new datasets, to verify its claims.

Contributions: Our primary contribution consist of the reproduction of the results of [16]. Firstly, we reproduce the results of [16] on the WWT dataset and we find that we achieve similar results; DoppelGANger captures all correlations. We do however achieve less accurate results on the GCUT dataset than [16]. We show that DoppelGANger nevertheless outperforms the AR and RNN models, as is also shown by [16].

Our second contribution is a mix of improvements that are not present in [16]. We start by explaining an inherent difference in the workings of DoppelGANger and the models the authors of [16] compare it to, and conclude that a better comparison is necessary. We show that DoppelGANger does not adequately capture measurement-metadata correlations on the GCUT dataset, similar to the other models. We furthermore quantify the Pearson Correlation on the GCUT dataset and find that decreasing the sample size does not always achieve worse results. We find that decreasing the sample size is an effective tool to reduce training time, for the WWT dataset. We provide implementations for the models that run on CPU and show that this is not slower than training the model on GPU, therefore making a present dependency redundant and opening the way to further improvements. Lastly

we tune the batch size parameter S and find that S = 5 achieves the best results.

The rest of the paper mentions related work, explains some background of the research, it formulates the working of DoppelGANger, explains the results and improvements and concludes the research.

2 Related work

This section discusses other research related to our work. Time series synthesis has been done by machine-learned models and non-machine-learned models. We will address each separately.

2.1 Non-machine-learned models

Non-machine-learned models consist of simulation-based approaches and mathematical models. Simulation models [5, 13, 17, 18, 20] work by building a realistic copy of the real world problem and simulating the results. This poses the disadvantage that is is incredibly hard to build a realistic model of the real world that produces similar results after simulation. On top of that they are not at all general; they have to be customised to every real world scenario. This makes much less useful than a general solution. Mathematical models [2, 3, 14, 4, 22] are models that stem from statistical or stochastic research. These have been shown to model certain problems accurately. The authors of [16] mention for instance that the Hierarchical Bundling Model models inter-arrival times of datacenter jobs better than the widely-used Poisson process [14]. Similarly to simulation models these models are not general. Different mathematical models need to be used to model different real world scenarios.

2.2 Machine-learned models

Machine-learned models generally are models that learn parameters from data by training. Examples of this are: Multilayer perceptrons, autoregressive (AR) models, Hidden Markov models, recurrent neural networks (RNNs) [1, 12] and GANs [10]. As mentioned in section 3.3.2 there is an inherent difference in the problem these models address. Notably, none model metadata and time series together.

Notable GANs that model single time series separately are RCGAN [8], which uses RNNs to generate time series and can use metadata as input, and TimeGAN [23], which, as the authors of [16] describe it, trains an additional neural network that maps time series to vector embeddings and outputs sequences of embeddings rather than samples [23].

We chose to reproduce DoppelGANger for a number of reasons. First and foremost we chose this to learn more about GANs and their application on time series. Furthermore, it offers a comparison to a wide number of models, which motivated us to learn more. It also has a widespread application; starting from network traffic, but possibly reaching far more.

3 Background of the research

This section explains how GANs work, explains some difficulties to it's workings and explains how DoppelGANger deals with these difficulties.

3.1 GANs

General adversarial networks are a type of neural network that can be used to generate data. A state-of-the-art and well known example - even outside of research literature - is named StyleGAN [15, 21]. This GAN generates realistic images of people that do not exist. They are so realistic that one can hardly tell the difference between real and fake people. A GAN can also be used to generate different data than images [16], but image data is a nice metaphor to explain the inner workings of the GAN.

A GAN works by having two neural networks work together [10]. One part is called the *generator* and the other is called the *discriminator*. Each learning cycle the generator generates data as best as it can. The discriminator tests the generated data alongside real data and learns to spot the 'counterfeit' data; the data the generator created. This feedback is then fed back into both parts. This way the generator learns to improve his counterfeiting skills while the discriminator learns to spot his mistakes even better. The end result is (supposed to be) data that can not be distinguished from real data.

3.2 Using GANs for time series synthesis

Our interest lies in using GANs for time series synthesis. A time series is simply a series of data points over a period of time. This could for example be the price of a stock during a day, the temperature each day of a month, or the amount of cars that pass a traffic camera every minute for an hour. This research will be exploring the use of GANs on time series data with metadata. Metadata is additional data corresponding to a measurement. An example dataset that is used is the number of daily views of Wikipedia articles [11]. The measurements are the views per day; the metadata is the corresponding domain name.

When using GANs to synthesise time series some difficulties arise. We will address each difficulty shortly. The first well known difficulty is *mode collapse* [10, 16]. Mode collapse occurs when the data is diverse but the GAN only outputs one certain 'mode' of the data, instead of the multiple underlying modes. A second difficulty is the *capturing of long-term effects* [23, 16]. This is the case because the MLP architecture fails to capture long-term effects properly, as is observed by [16, 23, 8, 9]. Furthermore a difficulty is the *capturing of complex relations between the time series and the metadata* that goes along

with it. Most new research focuses on generating measurements without metadata and can not easily be adapted to generate both metadata and measurements jointly. It is paramount to realise that generating independent time series for each measurement-dimension will break their correlations [16].

3.3 DoppelGANger

In this section we will explain how DoppelGANger handles time series synthesis.

3.3.1 The model

We have chosen refer to the original paper [16] for the explanation of the model. For an in depth explanation see Section 4 of [16].

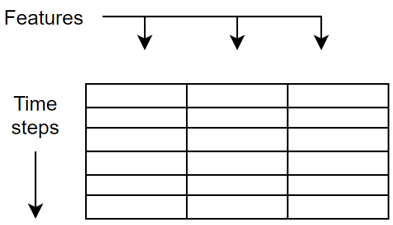

Figure 1: A schematic indicating the type of time series.

3.3.2 Spatial and temporal correlation

Before addressing these difficulties, we need to clarify the type of time series that [16] deals with. DoppelGANger builds a model that takes in n samples $\mathcal{D} = \{s_1, s_2, ..., s_n\}$ and produces m samples $\mathcal{D}' = \{r_1, r_2, ..., r_m\}$. The process is illustrated in Figure 2. For any two samples $s_i, s_j \in D$, the number of features and the length of data are all the same. If we zoom into each $s_i \in D$, itself is a time series tabular data, as can be seen in Figure 3. Each row in s_i is a time point, and the time is increasing with the increasing of the row numbers. For a corresponding row in any two samples $s_i, s_j \in D$, it records two events in parallel in the same time . All above characters of D are also applied for the generation D'. This is our goal, as we have networking data about e.g. multiple tasks per event type (GCUT) or multiple pages per domain name (WWT). [1]

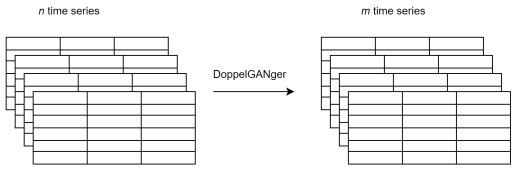

Figure 2: A schematic indicating the type of time series DoppelGANger deals with.

A different kind of time series is the type TimeGAN [23] and (generally) AR models and RNNs deal with. TimeGAN (and RNNs and AR models) produce a model that takes in 1 sample $\mathcal{D} = s$ and produces m samples $\mathcal{D}' = \{r_1, r_2, ..., r_m\}$. The difference between these two types is shown in Figure **??**. This problem generalizes to different scenario's, poses different problems and might have not have the same relevance to certain metrics.

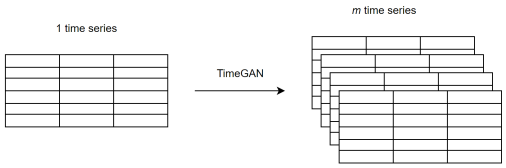

Figure 3: A schematic indicating the type of time series TimeGAN deals with.

Most importantly these problems are correlated differently. In the first scenario we will find temporal correlations, i.e. correlation within one sample, and multiple spatial correlations, i.e. correlation between different samples and correlations between different features. However, in the second scenario we will find temporal correlations, i.e. correlation within one sample, but only single spatial correlations, correlations between different features. There can not be correlation between different samples, as there is only one sample.

All the models DoppelGANger is compared to in [16] fall into this second category. Due to this it should not come as a surprise that all (modified) models that DoppelGANger is compared to, perform worse at the current task. They are designed for a different task and hence their comparison to DoppelGANger seems inaccurate. This undermines the claims made in [16] and leaves room for further experiments with DoppelGANger and further research on GANs for this purpose. This does not at all mean that the models proposed in [16] are poor; the opposite is true! This might be an entirely new application for GANs in time series synthesis that is not explored yet.

4 Reproducing

This section will cover the results achieved by our reproduction of [16].

4.1 Aims

Our main aims were to reproduce the fidelity performance of [16]. In that sense we want to show that DoppelGANger produces high-quality samples while learning temporal correlations, cross-measurement correlation, the metadata distributions and measurement-metadata correlations. We show this on the WWT and GCUT datasets; our new financial dataset is evaluated in Section 5. As

[1] To illustrate this difference we have omitted the metadata from this formulation. Including metadata for completeness, each sample s_i is actually a tuple $s_i = (m_i, d_i)$ of metadata m_i and measurements $d_i = \{f_i^1, f_i^2, ..., f_i^{t_i}\}$ of length t_i. This obfuscation only adds additional correlation between metadata and correlation between metadata and time series, which is already addressed by [16].

mentioned in section 3.3.2 we find the comparison to other models imprecise, and encourage further research on this topic. For completeness, to further support our point and for lack of a better option we have chosen to still compare the model to an autoregressive model, Markov model and an RNN.

4.2 Implementation

To reproduce DoppelGANger we chose to use the implementation provided at [6]. This implementation does not (yet) provide other models, but we contacted the author and received the source code for them. All models are implemented in Python using `tensorflow` 1.14. We were unable to get the model of the HMM working. For reasons mentioned in Section 3.3.2 as well as being restricted in time we chose to omit this model from our comparisons.
Modification beyond DoppleGAnger. The original code uses a dependency called `GPUTaskScheduler`. This dependency schedules execution on GPU; without it DoppelGANger can only be trained and not be used to generate data. We could not get this dependency to work, hence we have rewritten the code to accommodate for execution without this dependency. The code now only runs on CPU, but can now easily be ran by someone who seeks to reproduce the results of [16] further. Empirically we find that our average training times are similar to the times of [16] with the dependency, as can be seen in Table 1. [16] mentions an average training time of 17 hrs on the WWT dataset for their code using 50000 samples and 400 epochs. We have achieved an average of 8 hrs on the WWT dataset using 50000 samples and 200 epochs, while training models with different parameters in parallel. Because of this parallel training this number can only serve as an upper bound. As a result of this we do not see the need of this dependency and believe that a further speed up can be achieved by the author.

	Average training time (hh:mm)	
Model	*WWT*	*GCUT*
AR (5000 samples)	18:40	6:43
RNN (5000 samples)	3:17	4:08
DoppelGANger (CPU)		
1000	0:21	1:41
2000	0:33	2:20
5000	1:27	3:29
10000	1:56	5:21
20000	3:33	9:47
50000	8:22	18:38

Table 1: (Average) training times of our models on the WWT and GCUT data. Some models were trained in parallel, so these can only serve as upper bounds. All models were trained for 200 epochs.

Notably we find that the AR model trains faster on the GCUT dataset than on the WWT dataset. This is not expected, as the GCUT dataset is not only much larger, but other models also train slower on it than on the WWT dataset. Where the WWT data consist of 50000 samples of 1 feature of length 506, the GCUT dataset consists of 50000 samples of 9 features of length 2500. This discrepancy raises eyebrows on the implementation of the AR model and further supports our claims in Section 3.3.2.

Our source code is available at [7]; this code includes the model for DoppelGANger, and AR, RNN and HMM models, as well as all scripts that were used to produce the results in this paper. There is a dataset included as well as scripts to provide your own data.

4.3 Datasets, testbed and parameters

4.3.1 Datasets

As [16] mentions these datasets are chosen to exhibit different combinations of challenges: (1) correlations within time series and metadata, (2), multi-dimensional measurements, and (3) variable measurement lengths. On top of that we have introduced a new dataset of financial data, to test the performance of DoppelGANger on time series synthesis outside of the scope of network traffic.
Wikipedia Web Traffic (WWT): The WWT dataset consists of 50000 samples of length 550 that track the amount of daily views of a Wikipedia page, from July 1st, 2015 to December 31st, 2016 [11]. The metadata consists of three parts: the corresponding domain name, the type of access (e.g. mobile/desktop), and the type of agent (e.g. spider) [16]. Each sample consists of a single feature, the amount daily views of a page. The shape of the data is therefore `(50000, 550, 1)`.
Google Cluster Usage Traces (GCUT): The GCUT dataset consists of 50000 samples of variable length that track resource usage of tasks of a Google Cluster of 12.5 thousand machines, over the period of 29 days in May 2011 [19]. The metadata consist of the exit code of each task. Each sample consists of nine features of for example CPU rate, memory usage and disk space usage. These samples were logged per second, with mean and maximum logged every 5 minutes. When a task ends it is padded with zeroes, and the shape of the data is therefore `(50000, 2500, 9)`.

4.3.2 Testbed

All models were trained on a Google Cloud Virutal Machine with 8 vCPUs and 64 GB of RAM. As mentioned in section 4.2 our implementation only uses a CPU; this makes reproduction more accessible while training times (listed in Table 1) are similar.

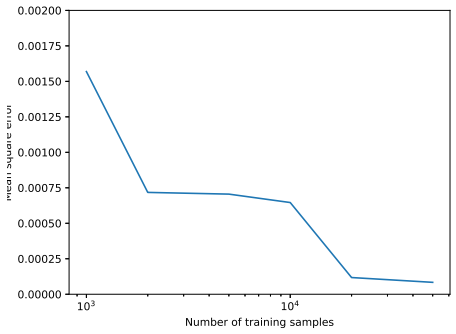

Figure 4: Plot of the MSE vs the samples of the WWT dataset.

4.3.3 Parameters

As a starting point for our experiments, we used parameters similar to [16]. Notably, however, we chose S = 10. Furthermore, because our resources were limited, models were trained on a subset of the data, with only 200 epochs. This number was chosen as a trade-off between time and accuracy; [16] mentions that convergence was achieved after 400 epochs. To verify this approach we have run several experiments that had more feasible running time. Similar to [16], Figure 4 shows the mean square error (MSE) of the average autocorrelation of the WWT dataset plotted against the sample size. We have run each sample set size three times and their MSE is plotted. In Table 2 this is show quantitatively. We find that similar to [16] the MSE deceases as the sample size increases.

Method	MSE
RNN	0.063091
AR	0.008374
DoppelGANger	
1000	0.001569
2000	0.000717
5000	0.000705
10000	0.000646
20000	0.000117
50000	0.000084

Table 2: Table of the MSE of web dataset

Empirically we find that our model achieves a lower MSE than [16] for both AR and RNN models as well as for DoppelGANger with 50000 samples. For the RNN this is a factor 2 (0.063 to 0.122), for the AR model this is a factor 33 (0.0084 to 0.2777), and for DoppelGANger this is a factor 11 (0.00008 to 0.0009). This is not expected, as we use fewer epochs and hence our results should be less accurate. Possible reasons for this could be:

(i) The model of [16] overfits on the data.

(ii) There is a high variance in our results and three runs are not conclusive.

(iii) There is an error in our calculation of the MSE.

These numbers do, however, show that DoppelGANger is already able to produce quite similar data when using only a subset of the 50000 samples. As a compromise between accuracy and execution speed we therefore chose to run all experiments on the WWT dataset on a uniform subset of 5000 samples.

Additionally we have also performed this experiment on the GCUT dataset. Figure 5 shows the MSE of the CDF of the Pearson correlation between CPU rate and assigned memory usage at different sample sizes. We have run each sample set size three times and their MSE is plotted. In Table 3 this is shown quantitatively.

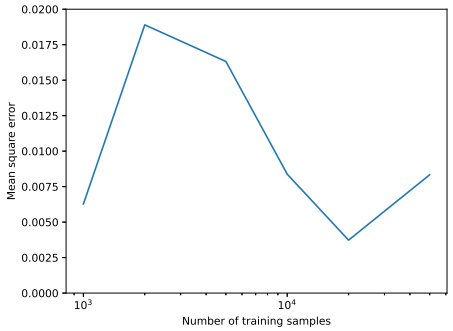

Figure 5: Plot of the MSE vs the samples of the GCUT dataset.

Method	MSE
RNN	0.031923
AR	0.274099
DoppelGANger	
1000	0.006276
2000	0.018899
5000	0.016320
10000	0.008379
20000	0.003729
50000	0.008338

Table 3: Table of the MSE vs the samples of the GCUT dataset.

The result of this experiment is not what we expected. We expected a trend of a decrease in in MSE as the sample size increases, similar to 4 on the WWT dataset. As visible in Figure 5, this downward trend is not present. One possible explanation for this is that it could be overfitting, or because the model has not converged yet. We can, however, not compare this with [16] as this quantitative analysis is new. It would be interesting to research this more.

These numbers do show that DoppelGANger is able to produce similar data for the the GCUT dataset when using a subset of the 50000 samples. Prior to the results of this experiment we chose to run all subsequent experiments on the GCUT dataset on a uniform subset of 5000 samples. After this result it seems we could have even chosen a subset of 1000 samples, as it achieves a better result. This decision poses, however, not a problem, as all achieved results of DoppelGANger have a smaller MSE

than the other models.
Experiments: For all our experiments we thus have, so far, the following parameters: 200 epochs, sample size of 5000, Batch size (S) of 10.

4.4 Results

This section shows that we have successfully reproduced the results of [16], as well as additional results.

4.4.1 Summary

As a summary, this papers shows, in line with [16], that: *(1)* The DoppelGANger model trains as expected on the WWT, albeit with some annotations. *(2)* DoppelGANger captures all relevant correlations on both datasets better than all other models.

Beyond the original paper we show that:

(i) We analyse the measurement-metadata correlation.

(ii) We find a different result for the GCUT dataset by quantifying the correlation by the MSE.

(iii) Comparison to AR, RNN and HMM models, as well as TimeGAN [23] and RCGAN [8] is deemed to be imprecise (Section 3.3.2).

(iv) Decreasing the sample size is an effective tool to reduce training time, for the WWT dataset.

(v) Training times on CPU and GPU are shown to be similar, so improvements can be made to the implementation of the model.

(vi) The parameter S is tuned (Section 5.3).

(vii) We provide a new dataset that unfortunately we were not yet able to produce results on.

4.4.2 Temporal correlations

The WWT dataset contains time series of length 550 and therefore contains temporal correlations over a long time span. This dataset inhibits correlation over a period of a year, and hence a plot of the average autocorrelation of each time series should show activity around day 365. As seen in figure 6 this is the case. This correlation should be captured by DoppelGANger and this is also shown in Figure 6. When we compare this to the results achieved by [16] we find that the DoppelGANger models capture the autocorrelation quite similarly, while the AR and RNN models behave differently in our results. Such discrepancies can be explained in the following ways:

(i) Our explanations in Section 3.3.2 that these models are not built for this and hence behave strangely.

(ii) The behaviour of these models, unlike the behaviour of DoppelGANger, is impacted severely by the sample size.

(iii) The behaviour of these models, unlike the behaviour of DoppelGANger, is impacted severely by the number of epochs and overfitting.

Empirically we find that DoppelGANger captures the weekly and annual patterns much better than the other models, similar to the result presented in [16].

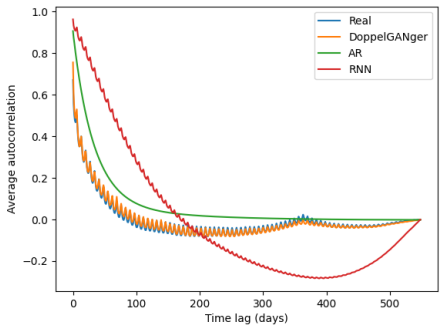

Figure 6: Average autocorrelation of the WWT dataset. This shows that DoppelGANger successfully captures the temporal correlations (annual and weekly) present in the WWT dataset.

4.4.3 Cross-measurement correlation

The GCUT dataset contains time series of varying lengths, with 9 different features of e.g. the CPU rate and the assigned memory usage. These features are heavily correlated, and hence high fidelity synthetic data should also inhibit this correlation. To show this we have computed the Pearson correlation between the measurements of the GCUT dataset, similar to [16], in Figure 7.

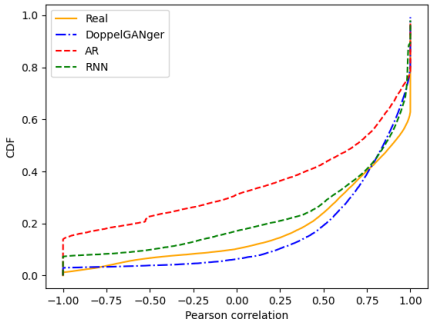

Figure 7: The cumulative distribution function of the Pearson Correlation between the CPU rate and the assigned memory usage of the GCUT dataset

This better shows what is mentioned in Section 4.3.3 and stems from 3. Initially it shows that DoppelGANger does capture some of the cross-measurement correlations present in the GCUT dataset. When we compare our results to the results achieved by [16] we find that [16] achieves better results than our model. As mentioned in Section 4.3.3 this could be due to having run less epoch, but not because of our smaller sample size. When looking at the tails of the plot we can conclude that there are some specific time series that inhibit either complete negative

or positive correlation [2]. As the GCUT dataset contains a lot of time series of extremely short length, as visible in Figure 8, we can explain this the following way. These extremely short time series consist of so few measurements that the correlation can only be completely positive or negative. DoppelGANger captures this best of all models because it captures the distribution of the metadata and measurements better than the other models. Nevertheless the discrepancies mentioned in Section 4.3.3, we still find that DoppelGANger captures the cross-measurement correlations better than the other models, similar to [16]. This does however benefit from further research.

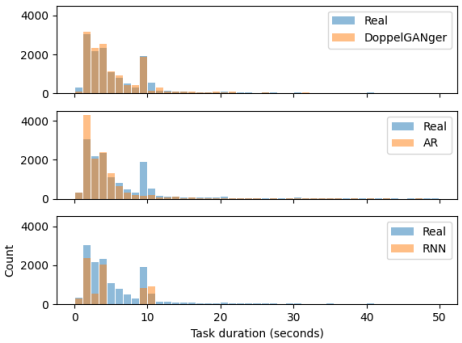

Figure 8: The distribution of the task duration of the GCUT dataset. DoppelGANger captures the distribution, but the AR and RNN models do not.

4.4.4 Metadata distribution

When a synthetic data is generated that has metadata, we expect the distribution of the metadata in the generated samples to be similar to the distribution of the metadata in the real samples. Figure 8 shows the distribution of the metadata of the real samples and the generated samples. We find that DoppelGANger captures the metadata distributions of the GCUT datasets. The RNN does not capture the distribution sufficiently, and the AR model fails to capture the second mode at a duration of 10 seconds. We therefore find that, similar to [16], DoppelGANger outperforms the AR and RNN models.

4.4.5 Measurement-metadata correlation

The original paper does not analyse measurement-metadata correlations. The GCUT dataset should have a lot of these, as different event types (`EVICT`, `FAIL`, `FINISH`, `KILL`) have different real world effects related to the measurements. We separated each event type and calculated the Pearson correlation between the CPU rate and the assigned memory usage, and have plotted the CDF. We show 2 different visualisations: Figure 9 is ordered by event type; it shows how the model captures the type of correlation in the specific event. Figure 10 is ordered by model; it shows how the correlation captured varies per event type.

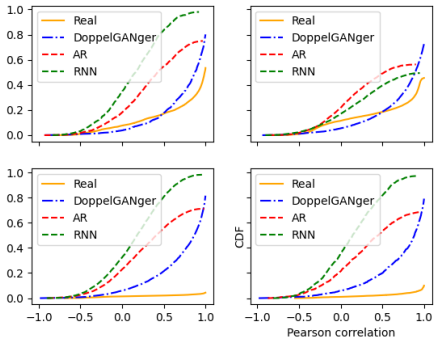

Figure 9: Measurement-metadata correlation ordered by event type. This shows the CDF of the Pearson Correlation between the CPU rate and the assigned memory usage.

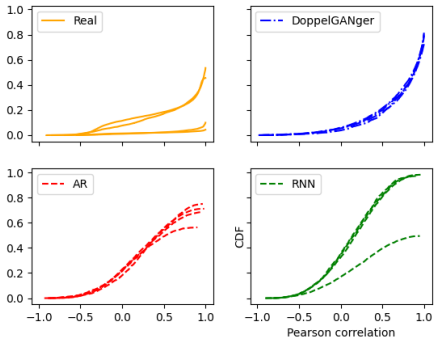

Figure 10: Measurement-metadata correlation ordered by model. This shows the CDF of the Pearson Correlation between the CPU rate and the assigned memory usage.

We can see in the top left plot of Figure 10 that the correlation in the GCUT dataset has two distinct modes [3]. We find empirically that DoppelGANger captures one of these modes, as can be viewed in the top two plots of Figure 9. It does, however, not capture the other mode, as is shown by the bottom two plots of Figure 9. The other models do not capture either of the modes properly. This becomes even more clear when we look at the top right plot in Figure 10. The plot for DoppelGANger of the CDF of the correlation shows that there is little variance in the correlation captured per event type. From this we can conclude that DoppelGANger does not capture the metadata-measurement correlations present in the GCUT dataset.

5 Proposed improvements

This section proposes improvements to the result of [16]. It serves as an overview of the work beyond [16], as well as offers points for further research. All improvements are tested and results are shown. In summary we propose a

[2] This is indicated by the flat part of the line that goes straight up on the left side, as well as the flat end that goes straight up on the right side.

[3] Ergo, within the four event types two show the same correlation.

better method to compare DoppelGANger, experiments to tune special parameters of DoppelGANger, execution on CPU and analysis about execution time, and lastly an improved code base.

5.1 Comparing DoppelGANger

As explained in Section 3.3.2 we believe the comparison of DoppelGANger to AR, RNN, and HMM models is imprecise and we encourage new research in this area. We propose to flip the tables and to compare DoppelGANger to TimeGAN in a different setting. What if we take three time series with different metadata from the WWT dataset, trained TimeGAN on these, and generated a set of new synthetic time series. This should show if the variance present in our datasets, as well as the metadata distribution and the measurement-metadata correlations are sufficiently captured by the models, and if DoppelGANger outperforms TimeGAN. Due to time constraints it was not possible to do this analysis ourselves.

Furthermore we propose an interesting point for further research might be to evaluate the result of DoppelGANger using the metrics that TimeGAN was evaluated on [23].

Another approach to evaluate the result of DoppelGANger might be to demonstrate the result in practice and show it improves upon a working model.

5.2 Data

We have included a new dataset that can be trained and evaluated on. When we tried to analyse our results we found our generated data to be corrupted. Unfortunately we can therefore not evaluate DoppelGANger on a new dataset.

We will introduce the dataset and provide it at [7] so it can be used in the future. It is a financial dataset that consists of 2710 samples of length 506 that indicate price data of certain stocks over the period of Jan. 1st, 2019 to Jan 1st 2021. The metadata consists of the sector the stock belongs to (e.g. technology, transportation, finance). Each sample consists of 6 standard features: Open, High, Low, Close, and Adjusted Close prices as well as trading volume. Each of these features is recorded daily, only for trading days (hence $506 \neq 731$). The shape of the data is therefore `(2710, 506, 6)`.

5.3 Parameter tuning

As another proposed improvement we chose to further tune the batch size parameter S. In the original work the author the author briefly mentions that an S of 5 produces the best results. We test additional values of S close to 5, as well as larger deviations, to find if this can be improved. Figure 11 shows that similarly to [16] an S of 5 produces an accurate result. We also find, unexpectedly, that an S of 275 produces similar results. As [16] mentions, when S gets larger, the difficulty of synthesizing a batch of records at a single RNN pass also increase; for this reason we would expect the result to be worse. A possible explanation for this is that it is only a minor deviation from the expected result, as produced by our not completely converged model.

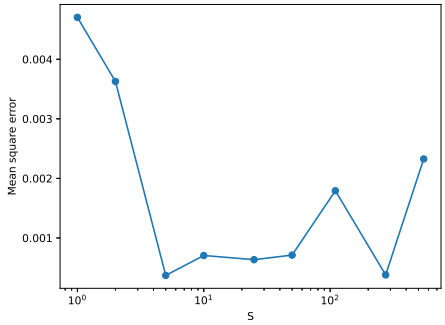

Figure 11: A plot of the MSE against the batch size parameter S.

Based on this explanation we empirically agree with the batch parameter S of 5.

5.4 GPU vs CPU

To run the code a dependency is necessary called `GPUTaskScheduler`. This dependency schedules execution on GPU; without it only a small part of the code can be run. We could not get this dependency to work, hence we could only train a model. Data could therefore initially not be generated. This could have been the case due to incorrect setup by us. As mentioned in Section 4.2 we modified the code to also run on CPU. This code is supplied at [7]. It also includes improved documentation, scripts to produce the plots in the paper and code for the other models.

6 Conclusion

While some of the results attained by [16] have been reproduced, others have yielded different results. Notably, DoppelGANger is compared to models that are not expected to perform well at the tasks at hand, and hence we believe that they do not offer sufficient support to establish the promise of [16]. This also means that in essence, [16] offers a new insight to time series synthesis using GANs that has not been seen yet, which opens the door to new unseen applications of GANs for time series synthesis.

References

[1] Oludare Isaac Abiodun et al. "State-of-the-art in artificial neural network applications: A survey". In: *Heliyon* 4.11 (2018), e00938. ISSN: 2405-8440. DOI: https://doi.org/10.1016/j.heliyon.2018.e00938. URL: https://www.sciencedirect.com/science/article/pii/S2405844018332067.

[2] Spiros Antonatos, Kostas Anagnostakis, and Evangelos Markatos. "Generating Realistic Workloads for Network Intrusion Detection Systems". In: *ACM SIGSOFT Software Engineering Notes* 29 (Apr. 2004). DOI: 10.1145/974044.974078.

[3] Brian F. Cooper et al. "Benchmarking Cloud Serving Systems with YCSB". In: *Proceedings of the 1st ACM Symposium on Cloud Computing*. Association for Computing Machinery, 2010. ISBN: 9781450300360. DOI: 10.1145/1807128.1807152. URL: https://doi.org/10.1145/1807128.1807152.

[4] Yves Denneulin, Emmanuel Romagnoli, and Denis Trystram. "A Synthetic Workload Generator for Cluster Computing." In: Jan. 2004. DOI: 10.1109/IPDPS.2004.1303297.

[5] Sheng Di and Franck Cappello. "GloudSim: Google Trace Based Cloud Simulator with Virtual Machines". In: *Softw. Pract. Exper.* 45.11 (Nov. 2015), pp. 1571–1590. ISSN: 0038-0644. DOI: 10.1002/spe.2303. URL: https://doi.org/10.1002/spe.2303.

[6] *DoppelGANger Github repository*. URL: https://github.com/fjxmlzn/DoppelGANger.

[7] *DoppelGANger reproduced*. URL: There % 20is % 20no%20link%20yet..

[8] Cristóbal Esteban, Stephanie L. Hyland, and Gunnar Rätsch. *Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs*. 2017. arXiv: 1706.02633.

[9] William Fedus, Ian Goodfellow, and Andrew M. Dai. *MaskGAN: Better Text Generation via Filling in the _*. 2018. arXiv: 1801.07736.

[10] Ian Goodfellow. *NIPS 2016 Tutorial: Generative Adversarial Networks*. 2017. arXiv: 1701.00160 [cs.LG].

[11] Google. *Web Traffic Time Series Forecasting*. 2018. URL: https://www.kaggle.com/c/web-traffic-time-series-forecasting.

[12] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9 (1997), pp. 1735–1780.

[13] Teerawat Issariyakul and Ekram Hossain. "Introduction to Network Simulator 2 (NS2)". In: Apr. 2012, pp. 21–40. ISBN: 978-1-4614-1405-6. DOI: 10.1007/978-1-4614-1406-3_2.

[14] Da-Cheng Juan et al. "Beyond Poisson: Modeling Inter-Arrival Time of Requests in a Datacenter". In: 2014. ISBN: 978-3-319-06604-2. DOI: 10.1007/978-3-319-06605-9_17.

[15] Tero Karras et al. *Analyzing and Improving the Image Quality of StyleGAN*. 2020. arXiv: 1912.04958 [cs.CV].

[16] Zinan Lin et al. "Using GANs for Sharing Networked Time Series Data". In: *Proceedings of the ACM Internet Measurement Conference* (Oct. 2020). DOI: 10.1145/3419394.3423643. URL: http://dx.doi.org/10.1145/3419394.3423643.

[17] Mario Lucic et al. *Are GANs Created Equal? A Large-Scale Study*. 2018. arXiv: 1711.10337 [stat.ML].

[18] Ismael Solis Moreno et al. "Analysis, Modeling and Simulation of Workload Patterns in a Large-Scale Utility Cloud". In: *IEEE Transactions on Cloud Computing* 2.2 (2014), pp. 208–221. DOI: 10.1109/TCC.2014.2314661.

[19] Charles Reiss, John Wilkes, and Joseph L Hellerstein. "Google cluster-usage traces". In: *Google Inc* (2011).

[20] Leszek Sliwko and Vladimir Getov. "AGOCS — Accurate Google Cloud Simulator Framework". In: *2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. 2016, pp. 550–558. DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0095.

[21] *StyleGAN example*. URL: https://thispersondoesnotexist.com/.

[22] Jianwei Yin et al. "BURSE: A Bursty and Self-Similar Workload Generator for Cloud Computing". In: *IEEE Transactions on Parallel and Distributed Systems* (2015). DOI: 10.1109/TPDS.2014.2315204.

[23] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. "Time-series Generative Adversarial Networks". In: 2019. URL: https://proceedings.neurips.cc/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf.