5.9% PlagScan by Turnitin Results of plagiarism analysis from 2023-02-16 11:27 UTC

ICDM2022Orlando.pdf

Date: 2023-02-16 11:24 UTC

✱ All sources 10 | Internet sources 10

☑	[0]	escholarship.org/content/qt2q51g25p/qt2q51g25p_noSplash_3b136de97bc43c6451f00da1723ade3d.pdf 3.7% 17 matches
☑	[1]	papers.neurips.cc/paper/8789-time-series-generative-adversarial-networks.pdf 3.8% 19 matches ⊞ 1 documents with identical matches
☑	[3]	arxiv.org/pdf/2107.11098 1.5% 7 matches
☑	[4]	arxiv.org/pdf/2111.08386 1.0% 6 matches
☑	[5]	www.vanderschaar-lab.com/synthetic-data-breaking-the-data-logjam-in-machine-learning-for-healthcare/ 0.6% 3 matches
☑	[6]	papers.neurips.cc/paper/6099-professor-forcing-a-new-algorithm-for-training-recurrent-networks.pdf 0.6% 3 matches
☑	[7]	arxiv.org/pdf/2006.05132 0.5% 3 matches
☑	[8]	repository.tudelft.nl/islandora/object/uuid:fe2e735d-35fa-4787-9a28-bfbf3a276146/datastream/OBJ/download 0.5% 3 matches
☑	[9]	arxiv.org/pdf/1810.05749 0.3% 2 matches
☑	[10]	arxiv.org/pdf/2209.06505 0.2% 1 matches

7 pages, 4539 words

PlagLevel: 5.9% selected / 5.9% overall

31 matches from 11 sources, of which 11 are online sources.

Settings

Data policy: *Compare with web sources, Check against my documents*

Sensitivity: *Medium*

Bibliography: *Consider text*

Citation detection: *Reduce PlagLevel*

Whitelist: --

TimeGAN for Data-Driven AI in high dimensional Industrial Data

Anonymous

Abstract—The availability of historical process data in predictive maintenance is often insufficient to train complex machine learning models. To address this issue, techniques for data augmentation and synthesis have been developed, including the use of Generative Adversarial Networks (GANs). In this paper, the authors apply the GAN-based approach to synthesize simulated time series data. Experiments are carried out to find a trade-off between the amount of labeled data needed and the accuracy of the synthetic data for downstream tasks. The authors find that using 40 % of the original data for training the GAN results in synthetic data that contains the same information for downstream tasks as the original data, leading to an estimated speedup of 60 % in the initial computing time. The results of the evaluation for the authors' own FEM simulation data, as well as for the Tennessee-Eastman benchmark dataset, are presented, demonstrating the potential of GANs in reducing time and energy in process development, while additionally interpolating a fixed parameter grid that is used for simulation purposes.

Index Terms—Generative Adversarial Networks, industrial AI, time-series generation, FEM simulation, data-driven AI, low data

I. Introduction

In the field of predictive maintenance, the availability and quality of historical process data is not sufficient to train complex machine learning models. In addition, the large number of parameters used in specialized neural network architectures requires a large number of training samples to provide reliable and accurate results. Because of this recurring problem with deep learning and machine learning in general, techniques are being developed to focus on the available data. Data augmentation and synthesis methods are successfully used to improve image recognition problems [1]. However, augmentation of data describing physical reality in a complex industrial process is not as easy as it is for images, since the physical reality must be preserved. One possible solution is to apply deep learning-based methods to synthesize simulated time series data to subsequently compare different partitions of the training data to find the amount of labeled data needed to generate accurate synthetic data for later use. In manufacturing, the modeling of new processes is often done with dedicated finite element method (FEM) software, which often takes a long time to compute the entire process. This step is and will always be very important in the development of new complex industrial processes since mathematical and physical conditions must be taken into account very precisely. When simulating a complex manufacturing process, a so-called process window is often defined in which the parameters for the simulation are set and then the simulation is performed. The high dimensional parameter grids would however take a lot of computing resources with specialized software to yield enough data to train machine learning models in downstream tasks. In addition to the number of resources used for those calculations, there are cases where only very specific parts of the simulated data are needed for the downstream tasks, i.e. those specifically useful for said downstream task. This may even result in the final simulated data not being used and thus energy and time resources not fully utilized. The computation time and required resources can be reduced by using Generative Adversarial Networks (GAN) to synthesize the required parts of the simulated data within this parameter grid. The GAN can then be used as an additional tool to quickly generate data that is important for a specific use case analysis. Generating sequences with the help of generative methods does not replace thorough use of numerical simulation software but can be an important tool to add to a simulation and process development toolbox. After a GAN has been trained on a limited set of numerically modeled data, synthesizing new data points or sequences in the context of time-series data is not a very resource-intensive process, as the trained generator model only needs to perform logical output tasks. This inference step typically takes $O(seconds)$ time compared to $O(hours)$ for numerical modeling. This time saving is also a cost and energy saving in process development and may enable or enhance the use of deep learning in subsequent tasks. Additionally a generative neural network can indirectly interpolate a parameter grid that is used for a FEM simulation by producing variations in the output data that translate into variations in the input parameter grid.

This paper is structured as follows: Section II describes similar work that has been done with generative neural networks. In Section III we explain the data, preprocessing steps and models that are used for the experiments carried out for this work. Section IV summarizes the findings of the experiments while discussing the trade-offs that can be done to maximize synthesizing power while reducing the time and energy consumed in the process. Section V gives an overview of the findings and an outlook of where the use of GANs for low data problems might also be applicable.

II.[0] Related Work

TimeGAN is a generative time series model trained antagonistically and collectively through a learned embedding space with monitored and unsupervised losses.[0] This approach overlaps several research directions and combines topics such as autoregressive models for sequence prediction, GAN-based methods for sequence generation, and time series representation learning [9]–[11]. In [6],[0] a novel mechanism to link the two strands of research to a generative model explicitly trained to preserve temporal dynamics is proposed.[0] It presents Time-Series Generative Adversarial Networks (TimeGAN), a natural framework for generating realistic time series data in different domains.

[3] A GAN is a generative model consisting of a generator and a discriminator, typically two Neural Network (NN) models. The generator takes input random vectors with specified dimensions and produces output vectors of the same dimension that are similar to the actual training data. The discriminator is a binary classifier that tries to discriminate between generated samples and real training samples. The generator and discriminator are alternately updated by backpropagation, playing a zero-sum game. That zero-sum game between the two components is played until equilibrium is reached. The Transformer architecture, based on multiple levels of self-awareness [12], has recently become a widely used deep learning model architecture. [15] has been shown to outperform many other popular neural network architectures, such as Convolutional Neural Networks (CNN) on images and Recurrent Neural Networks (RNN) on sequential data, and even to represent properties of a universal computing machine [13], [14], [15]. Some works try to use the transformative model in the design of the GAN model architecture with the aim of improving the quality of synthetic data or creating the most efficient training process [15], [16], [17] for image and text generation tasks. Since time-series data is not easily interpretable by humans, Principal Component Analysis (PCA) [18] and t-distributed stochastic neighbor embedding (t-SNE) [20][4] are used to map the multidimensional output sequence vectors into two dimensions to visually observe and qualitatively evaluate the similarity in the distribution of the synthetic data and real data instances. For a more quantitative comparison, several known signal properties are measured and compared to the transformer-generated ones as well as RNN-generated sequences with real sequences of the same class. [15] proposed several heuristics to more effectively train a transformer-based GAN model on time-series data, and quantitatively compared the quality of the generated sequences with real and with sequences generated by other state-of-the-art time-series GAN algorithms. [15] designed a conditional GAN with an architecture analogous to that of the DCGAN [20] to generate additional samples. In general, even without a data-centric mindset, it is safe to assume that significant effort has gone into preparing a dataset for machine learning to ensure it contains high-quality data points. Nonetheless, one can always expect to find invalid states, including those that are misclassified, ambiguous, or entirely unrelated. If the number of such samples is small compared to the core elements of the dataset, their presence has little impact on performance. There are two reasons for this: First, the gradients are averaged with respect to model parameters in each micro-batch. This reduces the effect of calculating an undesirable set of gradients for an invalid entry. Second, the effect on average gradients is more limited and is determined by the learning rate. Thus, if the vast majority of the data is correct, the deleterious effect of invalid samples becomes trivial. This desirable feature disappears as the percentage of invalid data points increases; the most common scenario in small data sets.[5] In such situations, taking additional steps to improve the quality of the data set has a significant impact on accuracy.

III. Data and Methodology

Data

The data used for the evaluation of the TimeGAN data synthesis experiments consists of two distinct datasets. Firstly the rather well-studied Tennessee Eastman (TE) dataset serves as a benchmark for high dimensional time series classification tasks [].

The Tennessee Eastman dataset is introduced by [3] and consists of ”fault-free” and ”faulty” datasets. Each data frame contains 55 columns (”fault number”, ”simulation-Run”, ”sample”, and the other columns contain the process variables). All 52 usable variables are numerical with a description of the meaning of each variable available under [2]. The complexity of this data set is comparable to the complexity of industrial processes. In the case of complex control systems, this work can be transferred to a specific case. The scheme of the chemical process is shown in figure 1. This dataset is used to evaluate the methodology on a large and complex dataset to demonstrate its applicability to other industrial datasets.

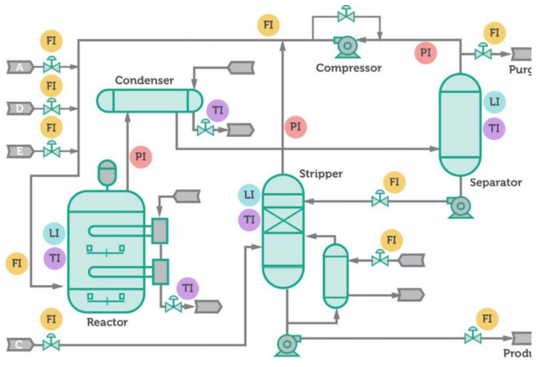

Fig. 1. Processdescription of the Tennessee-Eastman Process [2]

The second dataset studied is a self-generated FEM simulation data set from the pressing process, which is used in the automotive industry to produce high-rigidity steel parts for car bodies. The complete simulation dataset uses associated FEM models that calculate the heat transfer from a hot metal board to a die for molding, as well as numerically calculate the mechanical shifts of the metal. In this article, we will focus on the temperature data of this process, since this is a critical characteristic that needs to be monitored and/or correctly evaluated but cannot be measured directly. The temperatures are read by nodes on which temperature sensors are placed at a real processing plant. Figure 2 shows the location of the thermocouples in the molding tool.

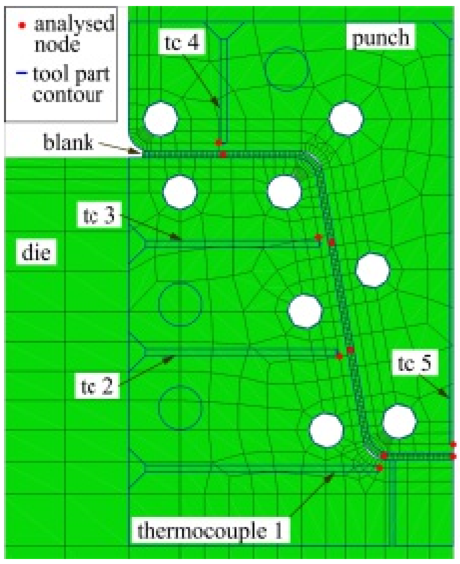

Fig. 2. Positions of the thermocouples (red dots) and nodes of the FE-model. The blank and contact area are modeled more finely than the outer parts of the punch and die to save computation time. The white circular areas are the cooling pipelines that are used for transferring heat from the tool out of the system in the real-world process.

From the point of view of a mechanical engineer, there is a process window in which parameters outside this window can never give satisfactory results. But within this multidimensional window of the process, there are many possible sets of parameters that can work. We modeled a parameter grid with four parameters, resulting in a total number of simulation attempts using the ABAQUS [8] simulation software totaling 20412. The total simulation time of all sequences was about 30 weeks (of one) processor time. Although we were able to parallelize this process on several processors, not all simulation runs converged due to the numerical instability of the process. A more detailed description of this modeling process and the subsequent follow-up task can be found in [darkened quote]. The parameter grid that we used to generate the training data can be found in the table I.

TABLE I
PARAMETER GRID USED FOR THE FEM SIMULATION

Parameter	minimum value	maximum value	number of steps
oven temperature	800 °C	1000 °C	21
forming die temperature	60 °C	170 °C	12
pressing force	10 MPa	25 MPa	16
holding time	4 s	10 s	4

From this data set, the question arose about how much of this FEM modeling data we would need to create a GAN that interpolates between already modeled data grid points. This data synthesis gives a little more variation in the training data for subsequent tasks.

To generate synthetic sequences, we pre-process the training data so that we have unique fixed-length sequences that are fed to the GAN. We also apply standard scaling to the training data so that the neural network can better process the values.

After preprocessing, we use the TimeGAN architecture described in [6] for data synthesis. The code used for this work is an adaptation of the TimeGAN [7] PyTorch implementation, and it can be found at [link redacted]. The parameters for training the timeline for different data sets are presented in the table II.

TABLE II
HYPERPARAMETERS USED FOR THE EXPERIMENTS

Parameter	Tennessee Eastman	FEM-Data
embedder epochs	500	5000
supervisor epochs	500	5000
GAN epochs	500	500
max sequence length	25	367
batch size	10000	10000
hidden dim	20	20
num layers	3	3
optimizer	adam	adam
learning rate	0.001	0.001

By trying to minimize the number of parameters in the network itself to prevent overfitting and demonstrate the strength of the architecture itself. Hidden units are long-term short-term memory (LSTM) cells. When evaluating the effectiveness of TimeGAN, we are guided by the evaluation metrics given in [6]. After TimeGAN finishes training, we generate a series of synthetic data k, where k is the length of the training data. With these balanced classes, we now train a two-layer LSTM classifier to match a variable from the generated data set. If this variable matches well for real data and for generated data, then the assumption is valid that data properties are preserved in synthetic (generated) data. We measure this with the mean squared error (MSE) and compare the MSE values between real and synthetic data. A low MSE means a good fit to the data and a small difference in MSEs for the model trained on real and synthetic data signifies well-matching synthetic data. Additionally, to measure the reliability of the model, a prediction is performed for each function in the original dataset to measure the mean MSE and standard deviation for the predicted value of the two-level LSTM. In addition to this metric, we visualize the data that is generated together with the real data. For that we apply, analogously to the original TimeGAN paper, t-SNE [4] and PCA [5] to the time-flattened sequences and plot the distributions into a single image. When the distribution of real and synthetic data is similar and closely clustered together we can assume, that a good distinction between these two classes is not possible for simple discriminative models.[4] This means that the generated data can be used for training downstream task models without introducing an unwanted bias.

Methodology

[6] proposed a generative model for time series data that preserves temporal dynamics. This model uses Generative Adversarial Networks (GANs) to fully account for temporal dependencies. At the same time, the learned embedding space is optimized along with the supervised and adversarial targets. A TimeGAN is composed of four main neural networks. Schematic diagrams of the layout of the TimeGAN architecture can be found in Figures 3 4 and 5.

[1] a) Embedding and Recovery Functions: [0] The functions learn the underlying temporal dynamics of the data. They are defined by the latent space vectors H_S H_X of S, X feature spaces as follows:

$$e : S \times \prod_t X \rightarrow H_s \times \prod_t H_x$$

Where $h_S, h_{1:T} = e(s, x_{1:T})$ e is computed by a RNN evaluation. Furthermore, the recovery function r is evaluated at each step by a feed-forward network: $\tilde{s} = r_s(h_s)$, and $\tilde{x}_t = r_x(h_t)$ where r_s and r_x are recovery networks. The generator network function (g generating function) is defined as:

$$g : Z_S \times \prod_t Z_X \rightarrow H_S \times \prod_t H_X,$$

g is implemented through a RNN, see Figures 3 and 4. The discrimination function d is defined as:

$$d : H_S \times \prod_t H_X \rightarrow [0, 1] \times_t \prod [0, 1].$$

[0] d function receives the static and temporal codes and returns a classification.

[1] b) Sequence Generator and Discriminator: The sequence generator outputs a latent representation into the embedding space denoted by Z_S, Z_X that are vector spaces with a predefined distribution.

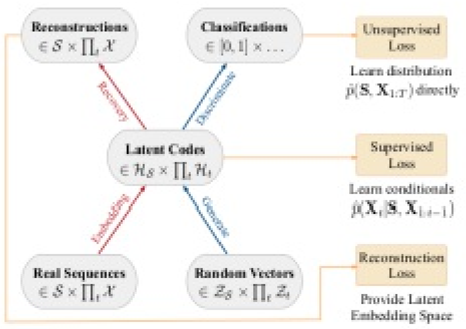
Reconstructions $\in \mathcal{S} \times \prod_t \mathcal{X}$
Classifications $\in [0,1] \times \dots$
Unsupervised Loss
Learn distribution $\hat{p}(\mathbf{S}, \mathbf{X}_{1:T})$ directly
Recovery
Discriminate
Latent Codes $\in \mathcal{H}_S \times \prod_t \mathcal{H}_t$
Supervised Loss
Learn conditionals $\hat{p}(\mathbf{X}_t|\mathbf{S}, \mathbf{X}_{1:t-1})$
Embedding
Generate
Real Sequences $\in \mathcal{S} \times \prod_t \mathcal{X}$
Random Vectors $\in \mathcal{Z}_S \times \prod_t \mathcal{Z}_t$
Reconstruction Loss
Provide Latent Embedding Space

Fig. 3.[0] Block diagram of component functions and objectives for the TimeGAN architecture. The different losses that are optimized and their connection to the reconstruction, classification and latent representations are shown. [6]

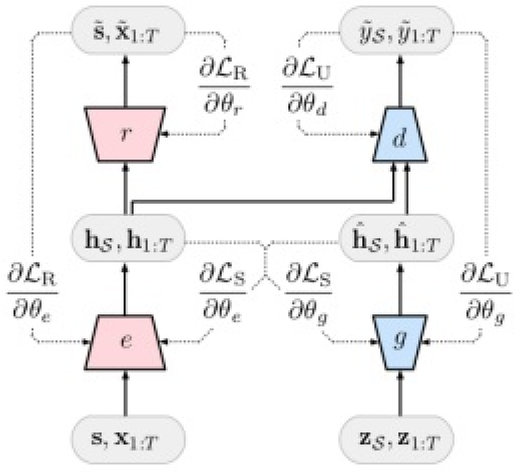
$\tilde{\mathbf{s}}, \tilde{\mathbf{x}}_{1:T}$
$\tilde{y}_S, \tilde{y}_{1:T}$
$\frac{\partial \mathcal{L}_R}{\partial \theta_r}$
$\frac{\partial \mathcal{L}_U}{\partial \theta_d}$
r
d
$\mathbf{h}_S, \mathbf{h}_{1:T}$
$\hat{\mathbf{h}}_S, \hat{\mathbf{h}}_{1:T}$
$\frac{\partial \mathcal{L}_R}{\partial \theta_e}$
$\frac{\partial \mathcal{L}_S}{\partial \theta_e}$
$\frac{\partial \mathcal{L}_S}{\partial \theta_g}$
$\frac{\partial \mathcal{L}_U}{\partial \theta_g}$
e
g
$\mathbf{s}, \mathbf{x}_{1:T}$
$\mathbf{z}_S, \mathbf{z}_{1:T}$

Fig. 4. Training scheme of the TimeGAN. The backpropagation steps of the different loss functions are shown where they act on the neural networks parts. The Network parts are denoted by recovery (r), discriminator (d), embedding (e) and generator (g). The hidden states are denoted with h. [6].

c) TimeGAN Jointly Learning: The reconstruction loss to accurately reconstruct $\tilde{s}, \tilde{x}_{1:T}$ from the original data s, x is defined as:

$$L_R = \mathbb{E}_{s, x_{1:T} \sim p}[\|s - \tilde{s}\|_2 + \sum_t \|x_t - \tilde{x}_t\|_2].$$

[0] This definition makes reversible mapping between feature and latent space feasible.

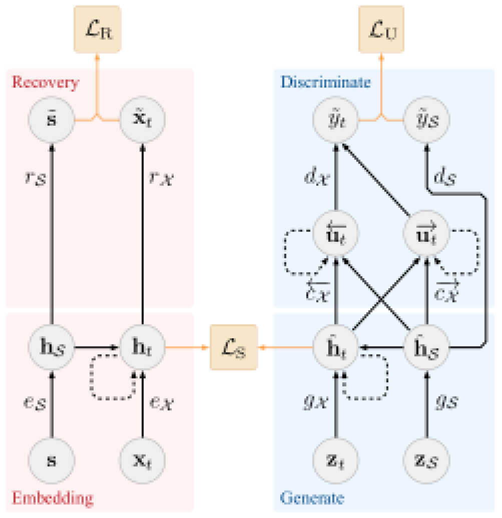

Fig. 5.[0] TimeGAN instantiated with RNNs.[0] Visualization with the shared loss function L_S which is calculated from the hidden states h of the generator/discriminator and embedding/recovery network parts. [6].

TABLE III

RESULTS OF THE TRAIN-SYNTHETIC-TEST-REAL EVALUATION OF THE FEM-SIMULATION DATA. RMSE OF THE HELPER LSTM TRAINED ON THE GIVEN DATA FOR THE FEATURE REGRESSION TASK. A LOWER DIFFERENCE HINTS AT A CLOSER MATCH BETWEEN SYNTHETIC AND ORIGINAL DATA.

GAN training data	Trained on original data	Trained on synthetic data	difference
10%	0.33 ± 0.2	0.29 ± 0.1	0.04 ± 0.3
20%	0.51 ± 0.2	0.57 ± 0.3	0.06 ± 0.5
30%	0.38 ± 0.1	0.35 ± 0.1	0.03 ± 0.2
40%	0.37 ± 0.1	0.33 ± 0.1	0.04 ± 0.2
50%	0.41 ± 0.1	0.32 ± 0.1	0.08 ± 0.2
60%	0.37 ± 0.1	0.32 ± 0.1	0.05 ± 0.2
70%	0.37 ± 0.1	0.33 ± 0.1	0.04 ± 0.2
80%	0.37 ± 0.1	0.32 ± 0.1	0.04 ± 0.2
90%	0.22 ± 0.1	0.46 ± 0.2	0.24 ± 0.3
100%	0.23 ± 0.1	0.47 ± 0.2	0.24 ± 0.3

d) Optimization:[0] $\theta_e, \theta_g, \theta_d$ are the parameters of the embedding, restoration, generator, and discriminator networks, respectively. The training is carried out by minimizing the cost function $\min_{\theta_e,\theta_r}(\lambda L_S + L_R))$.

Where λ is hyperparameter and set to 1, check IV.

IV. EXPERIMENTAL RESULTS

Running the experiments we find that not all of the simulation data is needed to train a GAN model that synthesizes data that closely resembles the original data. This means that a significant speedup from the order of CPU-weeks to the order of a single day in model training and minutes in new sample generation. We tested fractions from $10\ \%$ to $100\ \%$ of the original FEM-simulation data for training our TimeGAN. After training we generate as many synthetic sequences as we have original sequences to evaluate the similarity between original and synthetic data. We evaluate the trained models via the Train-Synthetic-Test-Real method to quantify the difference in information contained in the synthetic data compared to real data. We trained the LSTM to predict all of the features in the data with the other features as given parameters to simulate a regression downstream task. We then averaged the RMSE of the test predictions on the real data to aggregate the performance over all predictable features. We aim for the difference of the scores trained on original and synthetic data to be small while the score itself should also be small on the LSTM trained on synthetic data. The results of the evaluation for our own FEM simulation data can be seen in Table III. We find that the difference in predictive scores stagnates when using 40% of the original data as training data. The RMSE score of the LSTM also stagnates at that percentage of used training data. Secondary the performance of the LSTM drops significantly on the original data test set with a higher amount of training data used for the GAN training. This can be interpreted as an artifact of overtraining of the downstream model on the synthetic data. We conclude that a percentage of 40% of the original data could be used for training a GAN that yields synthetic data that contains the same information for downstream tasks as the original data. This in turn results in a estimated speedup in initial computing time of 60%.

Another, more qualitative method of showing the similarities between the original and synthetic data is transforming the sequences into a latent space with the t-SNE method and comparing the structures that arise from that latent space representation in a two-dimensional space. A lack of cluster-like structures that can be used to discriminate the original and synthetic data shows that the information in the two classes are very similar and a clear distinction between them is not possible.[7] This visualization can be seen in Figure 6.

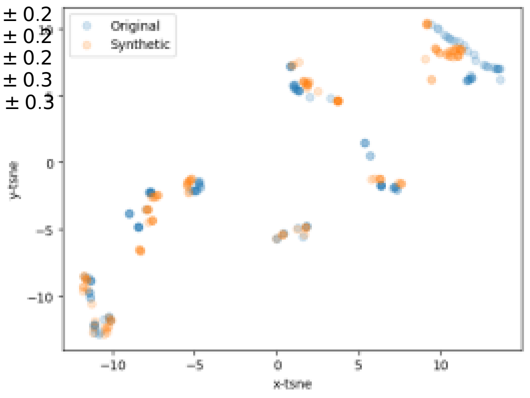

Fig. 6. t-SNE-plot for for the visualization of the similarity of the original and synthetic data. This plot uses the synthetic data from the TimeGAN model that has been trained on 100% of the original FEM-simulation data.

To benchmark our results against a well known dataset we run the same test procedure on the Tennessee-Eastman dataset. The results for the Train-Synthetic-Test-Real evaluation can be found in Table IV. We find that the performance of the GAN does not significantly depend on the percentage of used training data. This might be due to the initial size of the dataset and the information contained in the fractions used in the GAN training. When evaluating the generated sequences qualitatively with help of the t-SNE embedding of the sequences, as shown in Figure 7 one can see a close match between the original and synthetic data with some variations. This underlines the time-series generation capabilities of the TimeGAN trained in this work. In addition to the faster sequence generation and resulting time and energy savings the method has an added benefit. When simulating a rigid parameter grid one can only generate a sparse representation of the whole process. The use of a generative model that allows for variation in the output data also gives a possibility for an indirect interpolation of the parameter space. A schematic illustration of this grid interpolation is shown in Figure 8. This interpolation makes the parameter space less sparse and might lead to improvements in performance of downstream analyses, because of a more completely available process space.

V. DISCUSSION AND OUTLOOK

The results of this work show that Generative Adversarial Networks (GANs) have the potential to significantly improve the use of deep learning models in predictive maintenance.

TABLE IV
RESULTS OF THE TRAIN-SYNTHETIC-TEST-REAL EVALUATION OF THE TENNESSEE-EASTMAN DATASET. RMSE OF THE HELPER LSTM TRAINED ON THE GIVEN DATA FOR THE FEATURE REGRESSION TASK. A LOWER DIFFERENCE HINTS AT A CLOSER MATCH BETWEEN SYNTHETIC AND ORIGINAL DATA.

GAN training data	Trained on original data	Trained on synthetic data	difference
10%	0.03 ± 0.03	0.09 ± 0.08	0.06 ± 0.11
20%	0.04 ± 0.05	0.09 ± 0.07	0.05 ± 0.12
30%	0.03 ± 0.04	0.17 ± 0.17	0.14 ± 0.21
40%	0.04 ± 0.06	0.09 ± 0.10	0.05 ± 0.16
50%	0.04 ± 0.05	0.09 ± 0.09	0.05 ± 0.14
60%	0. ± 0.1	0. ± 0.1	0. ± 0.2
70%	0. ± 0.1	0. ± 0.1	0. ± 0.2
80%	0. ± 0.1	0. ± 0.1	0. ± 0.2
90%	0. ± 0.1	0. ± 0.2	0. ± 0.3
100%	0. ± 0.1	0. ± 0.2	0. ± 0.3

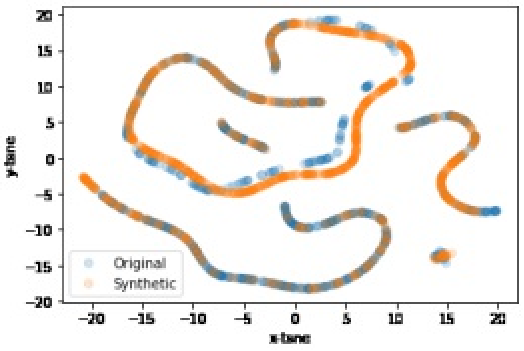

Fig. 7. t-SNE-plot for for the visualization of the similarity of the original and synthetic data. This plot uses the synthetic data from the TimeGAN model that has been trained on 60% of the original Tennessee-Eastman dataset.

In particular, we demonstrate that GANs can be used to synthesize simulated industrial time series data, which can then be used as training data for downstream tasks. The experiments carried out in this study show that the use of GANs can reduce the amount of labeled data needed for these tasks by around 60 %, as well as reduce the computing time and resources required for this process. Using GANs for synthetic data generation can lead to a significant speedup in the initial simulation computing time and due to that a reduction in energy consumption. These are important considerations in the industrial sector.

The trade-off between the amount of labeled data required and the usefulness of the synthetic data for downstream tasks is an important aspect to consider in future research. Further experiments can be carried out to determine the optimal amount of labeled data needed to produce synthetic data that meets the desired level of accuracy in specific downstream tasks. Additionally we showed a more general approach to test for the information contained in the synthetic data. Additionally, the use of GANs in other types of time series data, such as those generated by other complex industrial processes, can be investigated to determine the generalizability of this approach.

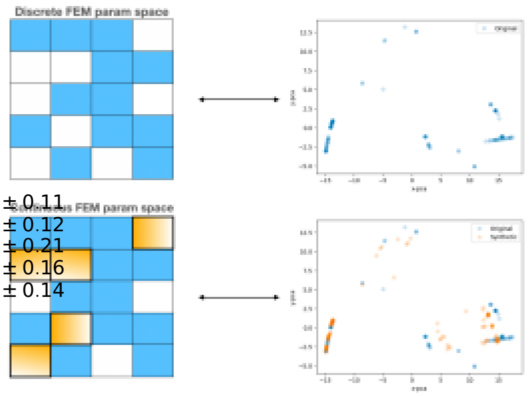

Fig. 8. Schematic illustration of a parameter grid used in FEM simulation and an indirect interpolation with a generative neural network. This indirect interpolation can be visualized using a PCA of the time-series data. The more space is filled in the PCA latent space between the original data, the more complete the process window is.

An end to end solution could be developed with industry partners for important use-cases.

REFERENCES

[1] Mohammad Motamedi, Nikolay Sakharnykh, Tim Kaldewey, A Data-Centric Approach for Training Deep neural networks with Less Data, CoRR. 2021

[2] Xiaolu Chen, Tennessee Eastman simulation dataset , IEEE Dataport, https://dx.doi.org/10.21227/4519-z502

[3] Rieth, Cory A. and Amsel, Ben D. and Tran, Randy and Cook, Maia B., Harvard Dataverse, Additional Tennessee Eastman Process Simulation Data for Anomaly Detection Evaluation https://doi.org/10.7910/DVN/6C3JR1

[4] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. Journal of machine learning research, 9(Nov):2579–2605, 2008

[5] Fred B Bryant and Paul R Yarnold. Principal-components analysis and exploratory and confirmatory factor analysis. 1995.

[6] Yoon, Jinsung, Jarrett, Daniel and van der Schaar, Mihaela, Time-series Generative Adversarial Networks, Advances in Neural Information Processing Systems 32 (NeurIPS 2019), https://proceedings.neurips.cc/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf

[7] Implementation of the TimeGAN in PyTorch, birdx0810, https://github.com/birdx0810/timegan-pytorch

[8] Michael Smith, Dassault Syst`emes Simulia Corp ABAQUS/Standard User's Manual, Version 6.9, 2009

[9] Samy Bengio, Oriol Vinyals,Navdeep Jaitly, Noam hazeerScheduled, Sampling for Sequence Prediction with Recurrent Neural Networks (CoRR), https://arxiv.org/abs/1506.03099

[10] Lamb, Alex and Goyal, Anirudh and Zhang, Ying and Zhang, Saizheng and Courville, Aaron and Bengio, Yoshua, A New Algorithm for Training Recurrent Neural Networks, 2016

[11] Bahdanau, Dzmitry and Brakel, Philemon and Xu, Kelvin and Goyal, Anirudh and Lowe, Ryan and Pineau, Joelle and Courville, Aaron and Bengio, Yoshua, An Actor-Critic Algorithm for Sequence Prediction, 2016

[12] Chernyavskiy, Anton and Ilvovsky, Dmitry and Nakov, Preslav, Transformers: "The End of History" for NLP?, 2021

[13] Dosovitskiy, Alexey and Beyer, Lucas and Kolesnikov, Alexander and Weissenborn, Dirk and Zhai, Xiaohua and Unterthiner, Thomas and Dehghani, Mostafa and Minderer, Matthias and Heigold, Georg and Gelly, Sylvain and Uszkoreit, Jakob and Houlsby, Neil, An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, 2020

[14] Devlin, Jacob and Chang, Ming-Wei and Lee, Kenton and Toutanova, Kristina, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018

[15] Li, Xiaomin and Metsis, Vangelis and Wang, Huangyingrui and Ngu, Anne Hee Hiong, TTS-GAN: A Transformer-based Time-Series Generative Adversarial Network, 2022

[16] Jiang, Yifan and Chang, Shiyu and Wang, Zhangyang, TransGAN: Two Pure Transformers Can Make One Strong GAN, and That Can Scale Up, 2021

[17] Shizhe Diao and Xinwei Shen and Kashun Shum and Yan Song and Tong Zhang, TILGAN: Transformer-based Implicit Latent GAN for Diverse and Coherent Text Generation, 2021

[18] Svante Wold and Kim Esbensen and Paul Geladi, Principal component analysis, https://www.sciencedirect.com/science/article/pii/0169743987800849

[19] van der Maaten, Laurens and Hinton, Geoffrey, Viualizing data using t-SNE, Vol 9, pages = 2579-2605

[20] Radford, Alec and Metz, Luke and Chintala, Soumith, Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015