universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Generating Reliable Process Event Streams and Time Series Data based on Neural Networks“

verfasst von / submitted by

Tobias Harald Herbert

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna, 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 921
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Informatik
Betreut von / Supervisor:	Univ.-Prof.Dipl.-Math.oec.Dr. Stefanie Rinderle-Ma
Mitbetreut von / Co-Supervisor:	

Acknowledgements

This work has been partly funded by the Austrian Research Promotion Agency (FFG) via the "Austrian Competence Center for Digital Production" (CDP) under the contract number 854187. This work has been supported by the Pilot Factory Industry 4.0, Seestadtstrasse 27, Vienna, Austria.

Abstract

Time series data is present in almost every domain and ranges from sensor data, sales data, stock data, climate data to astrophysics. The rise of connected devices in homes and the public space creates an abundance of time series data waiting to be utilized. Combining the abundance of data with the rapid development and increase in performance of machine learning models, makes it possible to utilize the data to find patterns, optimize existing processes and make scientific discoveries. However, one of the biggest challenges in machine learning is a lack of high quality data, and obtaining more data can be prohibitively expensive or due to the nature of context simply not possible. As a result, either a machine learning model can only be used at a later stage when more data is available or in other cases it is simply not feasible to use a machine learning model at all. The problem is a lack of quantity of training data for the machine learning models, which results in a low performance of the models. The goal is to boost small data sets with data that represents the underlying distribution. One way is to try to test out distributions that approximate the data well. However, this requires either a lot of manual labour and domain expertise, or a complex setup with vast parameterization and the effort needs to be repeated whenever a new process is introduced or an existing process changes. The goal of the proposed technique is to take a small data set as input, learn the characteristics of the data using machine learning and boost the data set to a desired size with minimal intervention and manual labour. Additionally, the solution should be modular and allow the usage of different machine learning architectures and interfaces that can accept different inputs and produce different outputs as well as the ability to add and remove processing steps. The proposed approach is a pipeline that takes event streams as log files as its input, perform pre-processing steps, train machine learning models, generate time series data and re-embed it into the originating log file. It focuses on automation from a business standpoint and provides an evaluation tool that provides research insights on the effects of different input data, machine learning models, overall performance and resource intensiveness. The results show that it is possible to take very small data sets and boost them with reliable data with minimal human intervention by using a recurrent neural network (RNN) to produce high quality data efficiently.

Kurzfassung

Zeitreihen sind in fast allen gewerblichen Bereichen vorhanden und reichen von Sensordaten, Verkaufsdaten, Aktienkursen, Klimadaten bis hin zu Astrophysik. Mit dem Aufkommen vernetzter Geräte, die sich in Häusern und im öffentlichen Raum befinden, waren noch nie so viele Zeitreihen Daten verfügbar. Die Kombination der Datenfülle mit der rasanten Entwicklung und Leistungssteigerung von Machine-Learning-Modellen ermöglicht es, die Daten zum Auffinden von Mustern, zur Effizienzsteigerung bestehender Prozesse und zu wissenschaftlichen Entdeckungen zu nutzen. Was aber, wenn all diese Techniken zur Verfügung stehen, aber Prozessen sich oft ändern und daher nie große Datenmengen produzieren? Das Problem, mit dem wir konfrontiert sind, ist ein Mangel an Trainingsdaten für die Machine-Learning-Modelle, was zu einer geringen Qualität der Modelle führt. Das Ziel, ist eine Möglichkeit, die Größe des Datensatzes mit aussagekräftigen Daten zu erhöhen. Eine Möglichkeit besteht darin, eine Verteilung zu ermitteln, die die Daten gut repräsentiert. Dies erfordert jedoch entweder viel Handarbeit und Domänen-Knowhow oder einen komplexen Aufbau mit umfangreicher Parametrierung. Der Aufwand hierfür muss wiederholt werden, wenn ein neuer Prozess eingeführt oder ein bestehender Prozess geändert wird. Das Ziel der vorgeschlagenen Technik besteht darin, einen kleinen Datensatz als Eingabe zu verwenden, die Eigenschaften der Daten zu lernen und den Datensatz möglichst automatisiert auf eine gewünschte Größe zu erhöhen. Darüber hinaus sollte es modular sein und die Verwendung verschiedener Machine-Learning-Modelle und Schnittstellen ermöglichen, sowie unterschiedliche Eingangsdaten akzeptieren und verschiedene Ausgabedaten erzeugen zu können. Zusätzlich sollte es möglich sein, Verarbeitungsschritte hinzuzufügen oder zu entfernen. Der vorgeschlagene Ansatz ist eine Pipeline, die Ereignisströme in Form von Logdaten als Eingabe verwendet, Vorverarbeitungsschritte durchführt, Machine-Learning-Modelle trainiert, Zeitreihen generiert und diese wiederum in die ursprüngliche Logdaten einbettet. Der Ansatz konzentriert sich auf die Automatisierung aus wirtschaftlicher Sicht und bietet ein Bewertungstool, das Forschungseinblicke zu den Auswirkungen verschiedener Eingabedaten, Machine-Learning-Modelle, Qualität und Ressourcenbeanspruchung liefert. Die Ergebnisse zeigen, dass es möglich ist, sehr kleine Datensätze zu nutzen und sie mit minimalen menschlichen Eingriffen mit sinnvollen Daten aufzuwerten, indem ein Recurrent neural network (RNN) verwendet wird, um effizient qualitativ hochwertige Daten zu erzeugen.

Contents

List of Tables

List of Figures

List of Algorithms

Listings

1 Introduction

This chapter is an extended version of the paper "Generating Reliable Process Event Streams and Time Series Data Based on Neural Networks". [HMRM21]

1.1 Motivation

In many domains such as medicine [Je20] and manufacturing [SRM20], continuous process monitoring and analysis are necessary to have stable and reliable processes and get timely alerts when it deviates from the expected. Especially with processes that produce very high dimensional data like machines or medical sensors that create data that describes the inner workings in detail, it is important to understand which of these metrics are relevant for certain outcomes of the process in order to find the sources for unexpected behavior and to optimize the process. Most of the sensor or machining data is produced as time series which allows to pinpoint failures or deviations to a specific moment and find the metrics that show abnormal behavior at or before the occurrence. Figure 1.1 takes the manufacturing domain as an example in which a computer numerical control (CNC) machine creates metal parts by removing material from a stock to produce identical parts with tight tolerances. Monitoring such a process using machine learning models is of interest for a multitude of reasons:

- Machine failure such as a broken tool head or a motor failure which results in a stop in the production
- Tool head deterioration or improper part cooling resulting in out of tolerance parts
- Unnecessary tool paths or slow feed rates resulting in a lower throughput

Such a machine is equipped with a logging server that produces process event streams that include time series data for the metrics of the different motor axis such as load, torque and rpm as well as related accessory process data while the machine is running. The machined parts are measured to obtain information about their quality and if they are within the defined tolerances. Together with the logged event stream, this creates a labeled data set in which each part has features from the production process and a label reflecting whether or not the part is within tolerance or if the machine experienced a problem while producing the part. Such a data set can be used to find the underlying reasons for machine failures and insufficient quality or to find the optimal machining speeds to have a high throughput while staying within the tolerances. The data set can also be used to predict anomalies based on the features extracted from the process event stream to monitor the machine and alert the machinist when the machine shows abnormal

behavior. Using online process mining together with techniques such as dynamic time warping (DTW) [SRM20] allows for easy access and usage of the produced data in analysis tasks.

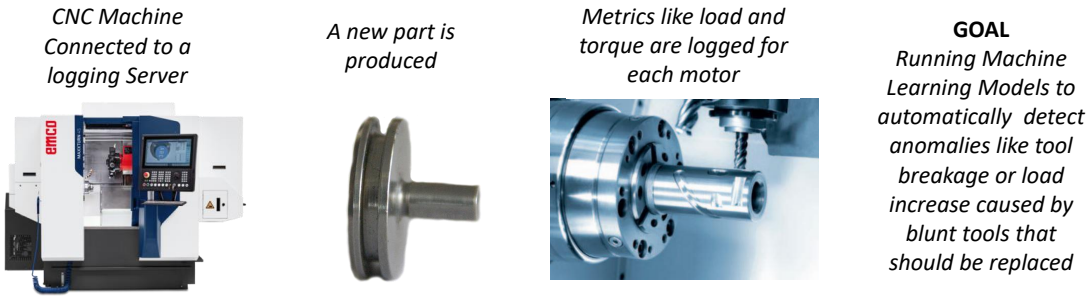


Figure 1.1: Machine outages and low yield rates due to out of tolerance parts cause high costs in production. machine learning models can find patterns that lead to failures or deterioration and can alert the machinist to take action before an incident occurs.

In order to utilize these powerful machine learning models for analysis and monitoring applications it is required to have reliable data that form the features for the input of the model. For a data set to be reliable it has to be of sufficient quality and size that the model can be trained in a way that it can generalize well to unseen data which is only possible if the training data reflects the distribution of the data well, i.e., it covers edge cases and contains meaningful variance. A minimum data set size is also required to be able to train the weights of the model for it to learn the repeating patterns that change the output of the model. Quality and quantity of the data set are often linked because a sufficiently large data set of low quality is required to undergo pruning to remove invalid and noisy data which reduces the size of the data set. Reasons for insufficient quality and quantity can be as follows:

- Changes in the process model causing the previous data to become invalid which results in a small data set for the new process.
- The data set is of high quality but becomes obsolete before it can grow large enough, e.g. small batch production.
- Unreliable data collection for sensor data which can be caused by:
  - Limited processing resources.
  - Network speed and reliability.
  - Noisy sensor readings caused by interference or human intervention.

The manufacturing domain is especially vulnerable to some of these issues as it often involve machines that were build before the more wide spread adoption of data collection. External systems need to be added to these machines to be able to collect data from the

inner workings of the machine such as metrics for the individual motors, temperatures and error codes. In reality this can be very difficult as many machines do not provide digital interfaces which means that only a subset of the available metrics can be logged and monitored and reliability is often a problem. Issues can also arise if the machine does provide its own data logging and interfaces as it is common that the same system that controls the machine also writes the log files which mean they share the computing resources and the system will most likely prioritize the machining process over the logging process which can cause gaps in the collected data which usually means these data instances can not be used for model training. Other domains have similar issues and cost is often hindering the collection of additional data or quality control and pruning of the data decreases the size of the data set. Furthermore, for some processes it is just inherently impossible to get larger data sets as the process changes too frequently, examples can be found in the medical domain as well as logistics and climate data. In the context of data produced by processes, most approaches for data generation do not include or prioritize time series data and are often based on probability distributions that require heavy parametrization which makes them labour intensive and often specialized to certain types of data and applications. The proposed approach is different as it utilizes neural networks to generate data and can optimize itself over time and requires only minimal user intervention as it is not based on a specific probability distribution but rather finds patterns in the data and can generate data from these learned patterns. This can also be a limitation as it relies on the input data to include data instances that reflect the underlying distribution well. Therefore the size of the input data set is less important than its characteristics. A very small data set with high variance that reflects the distribution well is more suited to this approach than a larger data set that is missing important edge cases or scenarios. It is however not required that the input data set is balanced which is one of its strength, e.g. in the medical domain a data set might consist of mostly healthy patients and only a small number of instances of a disease. Most machine learning approaches benefit or require a relatively balanced data set to not have a bias after training. In such a scenario, instead of removing data instances of healthy patients to create a balanced data set, the instances with the disease can become the basis for the data generation to balance the data set by boosting the smaller class of the existing data.

Machine learning algorithms can be used for monitoring, analyzing and optimizing processes [vdA16]. However, in many applications it is either expensive or not possible to gather enough data to train a high performing model. The idea behind the proposed approach is to broaden the applications where machine learning models can be used by boosting small data sets and make it possible to train models that perform well on unseen data. It can always be beneficial to check for patterns and outliers in log data, to optimize a process, learn more about its inner workings and to perform conformance checking to ensure the process in behaving as expected. Machine learning models can be applied to both online and offline applications, i.e., streaming the log file to the model to check for outliers and conformance in near real time or training a complex model that can analyze vast amounts of log data over night to identify sources of quality degradation.

These use cases can be very beneficial to the process but they require a certain amount of minimum training data in order to become useful. This is where the proposed approach can be introduced to the process life-cycle to broaden the time in which machine learning algorithms can be applied to the process. The open-source research implementation is publicly available at `https://cpee.org/genlog/`. This research tool gives access to all the data sets used in this thesis, ability to upload new data, start the pipeline and access the evaluation tools used and described in Chapter 6.

1.2 Research Questions

- A common problem in the field of data science is a lack of high quality data. Reasons for this can be that there is only a small data set available, or the data set is of low quality which requires pruning and in turn reduces the size of the data set. **How can the size of a small time series data set be increased with meaningful data in order to use it in machine learning applications?**
- Both traditional techniques for data generation as well as emerging machine learning algorithms are available to boost small data sets. They differ dramatically in the manual effort and domain knowledge required by the user, as well as the computational resources required to generate the data. **Which techniques exist that offer reliable time series data generation without the requirement of extensive domain knowledge and manual effort?**
- Finding the right machine learning techniques for data generation is a major part of the related work and the techniques differ strongly in their architecture and computational requirements. Furthermore, the machine learning techniques require a set of hyperparameters to improve the outcome (accuracy, ...), which again are often the result of manual fine tuning by a user. **What are the strengths and weaknesses of different techniques and can they be optimized by automatically choosing a suitable set of hyperparameters?**
- Time series data is used to predict anomalies or quality patterns. While an insufficient volume of data is available, these predictions might yield sub-optimal results. Boosting the data (i.e., artificially generating data from available time series) on the other hand might distort the predictions. **What is a suitable set of metrics to assess the quality of the generated data?**

1.3 Contributions

The contributions of this thesis are split into two major parts, one being the modular pipeline that automatically boosts small data sets with reliable data and the second being an web-based evaluation tool that allows the user to use various data sets to test different machine learning models and their hyperparameters. By automatically creating a detailed user-understandable generation report for each generation run (including the

automatically chosen hyperparameters and metrics for the used machine learning model), it is possible for a user to explore the potential properties of a generated data set and thus a specific use case. A big differentiator of the data generation approach is that it is not only a data generation model, but rather an end to end log file generation pipeline that takes log files of a specific type as the input, (1) extracts the relevant data, (2) processes the data, (3) trains the machine learning model, (4) generates the time series data and (5) re-embeds it into the original log file. This allows it to be used in any existing process or application that expects a certain type of log file - as the generated files have the same type, structure and content as the original files.
The web-based evaluation tool created as part of this thesis can be used to produce a full evaluation report of a specific data generation run, which includes (1) the input data set, (2) the chosen machine learning architecture and (3) the chosen hyperparameters of the model. It can be used in a production environment to find a model that performs well on the specific types of data, i.e., it finds a balance between quality and reliability of the generated data. It also determines the required computational resources and inference times in order to be usable in run-time or ex-post applications. Another use case is the exploration effect when slightly varying the input (e.g., changing a single variable, model architecture or hyperparamter). The observed effects might point to interesting edge cases and give the user an idea how potential future changes might affect production parameters such as part quality or maintenance requirements.

1.4 Methodology

This thesis uses design science [PTRC07] as a research methodology. Design science focuses on the successful creation of artifacts. This approach can be broken down into six individual steps:

- Motivate the solution to a problem by justifying the value of the solution.
- Infer objectives from the problem specification to be able to test the solution.
- Define the functionality of the artifact and produce it.
- Demonstrate how the solution addresses different instances of the problem.
- Observe and measure the ability of the solution to solve the problem in the real world.
- Communicate the novelty and capabilities of the solution to the research community.

This methodology is used in different granularities and in different aspects of the solution. It is used to verify that it is possible to generate reliable time series data instances based on a small input data set, given it contains a sufficient amount of variance. It also verifies that the entire process from data extraction, processing, model training, data generation and re-embedding can be automated. It shows through evaluation that the modular

architecture of the pipeline allows to use different machine learning models through the definition of interfaces. It demonstrates its ability to work successfully with data produced by real world processes by using the created artifacts in existing applications. The web-based evaluation tool can be used to communicate findings to the science community and providing an easy and transparent way to recreate the findings.
This structured approach manifests as follows:

- **Motivation**
  Constantly evolving processes make it difficult for machine learning models to have enough process log file training data to provide reliable results.
- **Objectives**
  Does the generated data follow the same distribution of the original data and does it introduce variance, making it an asset for data analysis?
- **Artifacts**
  Two separate artifacts, a data generation pipeline focusing on automation and a evaluation and research tool focusing on gaining insights in how different data, models, and parameters affect the resulting data.
- **Demonstration**
  Two problems are addressed, one being the generation of reliable process event streams with minimal user effort and the other being able to compare and test the capabilities of different machine learning algorithms.
- **Evaluation**
  An extensive set of evaluation metrics is introduced and applied to the data to verify how changes in the input and architecture affect the output, and which models are suited for which use case.
- **Communication**
  The web-based open-source evaluation tool provides the ability for anyone to run the pipeline with their own data and change the models and parameters to utilize the evaluation results for research purposes.

1.5 Structure of the thesis

The thesis is structured as follows:
Chapter 2 explores the current state of the art in the field of process event stream data and introduces different approaches to data generation. Different machine learning algorithms and their applications are introduced and discussed in the context of time series data generation.
Chapter 3 discusses the existing problems such as small or ill-balanced data sets in detail and lays the foundation for the concept of the proposed approach as well as highlighting different use cases such as using the full or only parts of the pipeline depending on what

the available input data is and what the desired output data should be.
Chapter 4 describes the prototypical implementation and the underlying reasoning for the architectural and technical decisions that were made as well as describing each step in the pipeline in detail with a focus on the machine learning models.
Chapter 5 is concerned with the implementation of the proposed approach in the context of the manufacturing domain, specifically the production or parts on a CNC machine and all the challenges that arise with real world data such as missing data in the time series, reassigned labels for sensors as well as the diversity of the data distributions for the different metrics.
Chapter 6 introduces the developed evaluation tool, the evaluation techniques and their results in the context of the manufacturing of machined parts. The different evaluation metrics are described and compared to each other and evaluated for their usefulness.
Chapter 7 answers the research questions and gives insights into how the findings can be applied and how they contribute the the field of machine learning and process data generation.
Chapter 8 concludes by reflecting on the posed problem and what was learned through the implementation and evaluation of the approach as well as looking into what else can be done with the approach and what limitations are present in its current form.

2 Related Work

This chapter is an extended version of the paper "Generating Reliable Process Event Streams and Time Series Data Based on Neural Networks" [HMRM21]

The process mining manifesto states that the provision of high quality event data is crucial (Guiding Principle 1 and Challenge 1) and that process mining should be combined with other analysis techniques (Challenge 9) [vdAe11]. The proposed approach contributes to both claims. The generation of process log data can be approached from two directions, i.e., through simulation based on process models (e.g., using Colored Petri Nets [MW17] with CPN tools[1]) or through executing process models in "test mode", e.g., with mocked services [SRHM16]. These approaches, however, rely on process models with estimated values, e.g., probabilities at alternative branchings, and do not generate sensor data. An approach for integrating event streams from heterogeneous data sources is presented in [KMR19], but no event streams are generated. Approaches such as [SRM20] analyze and exploit event and sensor streams, but do not generate them either. Analyzing the capability of the approach for predicting the next activity in a process execution in comparison to existing approaches such as [ERF17] using deep learning is part of future work.
Neural networks [Roj13] can be used in many applications such as face detection [PVZ15], general regression [S+91], image classification [CMS12] or predicting extreme weather in climate data sets [LRC+16]. Neural networks use many nodes which are connected with weights between the input and the output. Features are used as input and represent the available training data. In case the raw data is unstructured, noisy or redundant, it might be necessary to transform, clean, and pre-process the data in order for it to become features. A feature might be a numeric representation of classes e.g., different values for different classes, in this case the representation should be changed to a one-hot encoding as it would otherwise see numbers close to each other also closer related which is not the case for classes. In order to set the weights appropriately and to be able to perform classification, backpropagation [HN92] is used with a loss function and gradient descent [Ama93] to determine the optimal weights for the training data. Parametrization is important to avoid overfitting [Haw04] or underfitting of the data and allow the model to generalize well to unseen data. Deep learning [GBCB16] moves away from linear perceptrons and introduces hidden layers and non-polynomial activation functions which allows it to be a universal classifier, it does however only use a feedforward mechanism [SKP97] and observes its input as just a collection of inputs without a temporal relationship. A subset of deep neural networks are the convolutional neural networks (CNNs) which are heavily

[1] `http://cpntools.org/`

used in image classification [RW17], image segmentation [BKC17] and combining these two applications are used in autonomous vehicles [WIJK17] to detect where objects are in the field of view of the camera such as other vehicles or pedestrians. Another application is natural language processing (NLP) for sign language which identifies hand gestures [SBCN20] to provide help with virtual assistants. Transfer learning can be applied to deep neural networks to achieve fast training times by utilizing pre-trained models, resulting in very capable models that benefit from being trained on vast data sets whilst only requiring training of the new and unseen data for the application of the model. An example is the Inception [SVI+16] model from Google which can be a base model for advanced image classification. Recurrent neural networks (RNNs) [Man01][Gra13] are used in image generation [GDG+15], document modeling with sentiment classification [TQL15], text classification [LQH16] and symbol creation and recognition including challenging applications like Chinese characters [ZYZ+17]. Another use of an RNN is to employ the hidden state vector to create a temporal context [CMA94] which influences the output of each node together with the weights of the node and allows the use on time series data where the data has relationships over a time period. It can be used for time series prediction [ZX00] taking into account seasonality and other patterns as well as time series classification [HS03] and is architecturally well suited to deal with missing values [CPC+18] which are very common in practise and can cause problems for more traditional methods [DR81]. Having a hidden state vector also makes it a powerful concept for time series data generation, but it is are still not able to keep important information over a longer period of time [MJC+14]. During back propagation, recurrent neural networks suffer from the vanishing gradient problem [PMB13] which reduces the strength of the update to the weights based on how far the back the propagation is progressing. The weights towards the end of the model are updated first and the updates have a strong impact and further down the line the updates approach zero which essentially nullifies the updates at this stage. A long short-term memory (LSTM) [MVSA15][HS97] uses a global state to combat this problem and propagating the error to each node directly instead of travelling through all the nodes to reach the beginning of the model. Gates are used to control the flow of information to find the relevant patterns and ignore the rest of the data. The data is regularized with the hyperbolic tangent function and forget, input and output gates are created with the sigmoid function that controls the flow of information in both the forwards and the backwards direction which can be seen in Figure 3.3[Phi18]. LSTMs are not limited to time series data and can also be used for sequence labeling [MH16] and text classification [ZSLL15] which can be used on the process and task level for conformance checking and anomaly detection. Gated recurrent units (GRUs) which are related to LSTMs and are also based on RNNs can be used to predict bottlenecks [FGL+20] in a production setting and generally show similar levels of performance compared to LSTMs whilst requiring less training time due to their simpler internal architecture. Models based on RNN architectures using existing data can also address issues like overfitting and underfitting which can occur in process mining applications [vdARV+10] which commonly use rules and parameters to achieve a similar result.

Generative adversarial networks (GANs) [GPAM+14] and variational autoencoders (VAEs) [FVA14] are powerful concepts in data generation and are used to generate images from a class [DCSF15] or from textual description [ZXL+17] as well as combining a set of input images of a person with a textual description or parameters to change their appearance by altering their smile, facial hair or even gender [CCK+18]. Other domains where GANs are employed are speech enhancement [PBS17], super resolution [WYW+18] which is a technique to increase the resolution of an image with the goal to obtain the maximum amount of detail, or creating video based on speech input and images of a person [ODK+19]. Generative adversarial networks are starting to be used on time series data as well by changing the internal architecture to incorporate the temporal aspect of the data. Generative adversarial networks are comprised of two neural networks which are usually deep neural networks [LWL+17] in the context of complex data and can also be recurrent neural networks. Additionally, they can employ convoluational layers [KSH12] which is especially useful in the context of images and audio [Pic15]. The two networks - the generative network and the discriminative network - are competing in a zero-sum game [WGD+17] in which one agent wins, the other agent loses which makes them adversaries hence the name generative adversarial network. The generative network takes as input the data from a latent space and produces candidates which are then evaluated by the discriminative network which takes as input the candidate and samples of the true data. Both can have one or more loss functions and use backpropagation independently from each other to minimize their loss functions. In each iteration, either the generator or the discriminator are optimized and generally the model is considered to perform well when the discriminator continuously fails to distinguish between real and fake data samples. Depending on the chosen loss functions, the networks can operate unsupervised, semi-supervised or supervised depending whether the input data is labeled, unlabeled or mixed. Using a latent space for the input of the generative network is an important optimization step as otherwise the training cost would render them infeasible, however it can also serve the purpose of preconditioning the data to reduce noise in unlabeled data or to shift the focus to certain aspects in the data. In the context of time series data this preconditioning of the data can increase the focus on attributes of the data that are subject to change over time in contrast to static attributes. The use of preconditioning also comes with the risk of taking out important information from the data and should be used with caution either by consulting a domain expert to increase the knowledge on the data or increasing the focus on previous steps like finding correlations and patterns in the data. Compared to techniques like the LSTM, the cost of training on time series data is usually orders of magnitude higher due to the fact that training the two competing networks that are not from the ground up designed for time series data takes longer than training an LSTM to predict from one time step to the next. In most real world scenarios and applications it is not sufficient to just choose one variant of neural networks to solve a posed problem. What makes neural networks so powerful is the ability to combine the different architectures and approaches with each other. This can be done within a model by choosing different variants for different hidden layers. An example is video where convolutional layers are used for image segmentation and classification

i.e., working with spatial information and they are combined with recurrent layers that work on the temporal information. This is a powerful concept as it allows the model to learn from both spatial and temporal data. Models that are just designed to generate convincing images or are using inpainting [YLY+18] to fill holes in images, show severe artifacts and jittering when applied to video, as the model is not aware of the previous and the next frame, resulting in the generated information to be inconsistent over time. By adding the temporal context to the model, it is able to take into consideration how the scene changes which reduces artifacts and results in a more consistent output video [KWLK19].
Taking this concept even further, it is possible to take this model architecture and use it in a generative adversarial network. By using different and optimized model architectures to generate data, the GAN concept can be applied to a wide variety of data. This can be further aided by using e.g., variational autoencoders (VAEs) for preconditioning of the data. There is however a trade off for creating such a powerful framework, which is the computational requirement, which can quickly increase by orders of magnitude for more complex models. Therefore a major focus is to investigate the differences in model performance, computational requirements, prediction quality and ease of optimization as well as ability to integrate it into a highly automated setting. As a result, some architectures can be used in online i.e., near real time environments because of their fast training and inference times, while others are more suited in offline scenarios with a high data complexity which can benefit from a more comprehensive model architecture.

3 Solution Design

This chapter is an extended version of the paper "Generating Reliable Process Event Streams and Time Series Data Based on Neural Networks". [HMRM21]

As insufficiently large data sets are one of the biggest challenges in many machine learning applications, this thesis aims to take a small data set as input and produce a larger data set as the output that reflects the input data but adds meaningful variance to the data. The proposed solution is called GENLOG which stands for generation of log files and is designed as a pipeline which takes event streams as for example XES[1] format and outputs generated log files of the same format. Being modular allows the pipeline to add and remove processing steps and create interfaces to existing applications and processes. The machining of a new part as described in Figure 1.1 shows the importance of continuous operation and parts that are within tolerance to assure a high throughput. A machine learning based monitoring model can help to find patterns that lead to machine failures and quality deterioration of the parts and use classification to alert the machinist to take action and investigate the issue further. The problem in this use case is the fact that the produced parts are specialized and produced in small quantities, i.e., the type of part produced changes regularly, which results in the monitoring model to perform poorly as it does not have enough training data available to adjust its weights iteratively to a point where it can adapt well to unseen data. Figure 3.1 illustrates the life-cycle of the monitoring model from data collection to model training, deployment and monitoring of the machine. It highlights the difficulty of training the model due to a lack of training data and brings in GENLOG before the training step to learn the distribution of the data and to boost the data set. The pipeline is therefore embedded into the existing process of the monitoring model. Another difficulty in this scenario is the fact that even within the production of one type of part, the machining process evolves in order to optimize quality, throughput and stability of the process. Depending on the severity of the changes, GENLOG can continuously adapt to changes in the process without the requirement to make strict cuts and discarding older data. Depending on the configuration, the generated data can be based on mostly recent data, or on a broader time span and can be optimized with domain knowledge or left as default otherwise. The reason why a data set is small can be based on a number of different causes including:

- Noisy or invalid data renders an initially large enough data set too small after pruning.

[1] XES stands for eXtensible Event Stream and is the de facto process log standard, `http://www.xes-standard.org/`

- The data set is not balanced resulting in a bias in the trained model [FC18] and thus requiring the deletion of data instances from the larger class, reducing the data set size.
- Substantial changes to the process that generates the data resulting in a reset of the data set which can be due to:
  - The subject, e.g., type of part, of the process changes, changing the nature of the produced data.
  - The order of executing of the process is altered, making the data incompatible with the old data.
  - The process is optimized causing smaller changes which trigger the outlier detection of the model.
  - Changes in the log file structure like hierarchy or relabeling of metrics make it incompatible with older data.

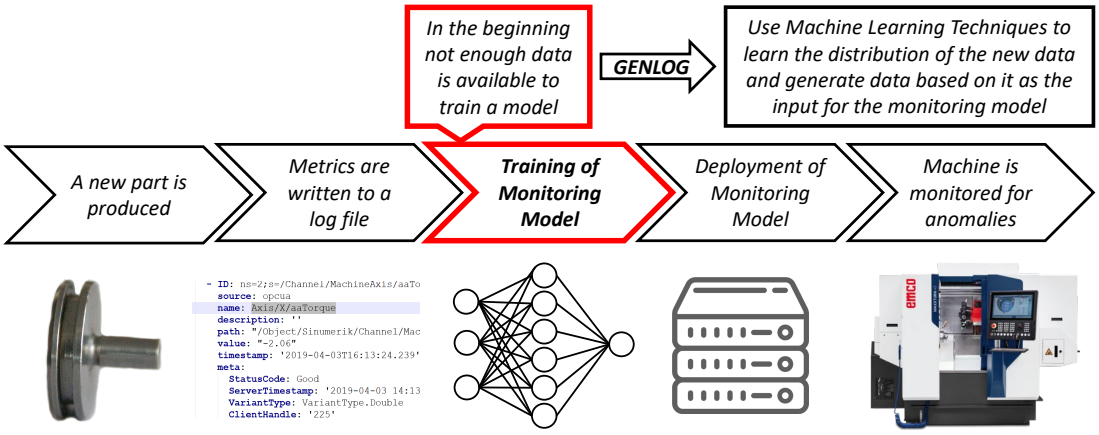


Figure 3.1: The life-cycle of the monitoring model shows that it cannot be used when a new type of part is produced or when the process changes substantially. GENLOG can be added to the life-cycle to allow the monitoring model to be trained from an earlier stage and be more robust to process changes

Figure 3.2 shows the pipeline of GENLOG, illustrating that the output is of the same type as the input which allows it to be embedded into existing processes and applications as the resulting generated log files have the same structure, hierarchy and data which allows the use in any process or application that expects the original log files. That is one way GENLOG is different from other approaches as it provides an end to end time series data generation solution for process event streams that does not require the implementation of additional software to continuously generate data for other processes. Its modularity and defined interfaces also allow easy replacement of processing steps, e.g., different machine learning algorithms, different input/output log file types or interfaces

to existing applications like web services or databases.

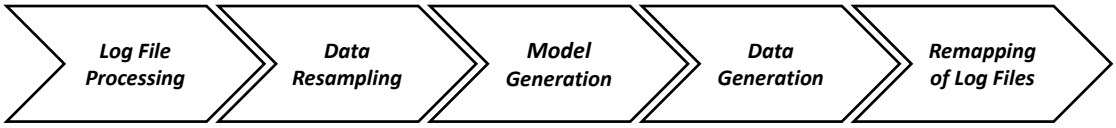


Figure 3.2: Modular pipeline implementation of GENLOG, facilitating a high degree of automation and flexibility for different machine learning algorithms and support for different file types and interfaces to other applications and services

The GENLOG pipeline starts by processing the input log files. The result of this step is a collection of time series data instances, one for each metric for each log file. Examples for metrics are sensor data for temperature, acceleration or power related measurements. Next the data is resampled to a common sample rate which is then used for binning using quantiles to create bins of equal size which is required for some of the machine learning architectures. The data is then ready to be used as input for the training of the neural network. At this stage, no human intervention was necessary and the only parameter used, was the list of metrics that should be extracted from the log files which can also be omitted if all metrics should be used. Now the user is presented with a dashboard that shows the metrics in terms of their grouping either by bins, batches or temporal, which are used to choose the data that will be the base for the data generation. The data is then generated using the inference method of the trained models and using existing data as input. At this stage the data is still a time series and the final and crucial step is to map it back into its original file format. Thus it is embedded into a template which results in a new log file with generated data that can be used by any process that uses log files as its input and it could even be used in GENLOG to create new models.
Depending on the use case, it is possible to train many LSTM models even down to the individual instances. It is also possible to aggregate the data first and then train a model on e.g., the median and standard deviation to reduce the effect of noise in the data. Binning is also an option to control both the length of a time series and the granularity of the model. Besides the trained models, additional variance can be achieved by using the existing data as input for the models inference method such that the generated data is a combination of model and input data. By using domain knowledge it is possible to control the distribution of the data generation in the user interface by omitting invalid or problematic data. In the GENLOG approach, models are trained at fine granularity and the user selects the data that should be used to generate new log data. It can also be implemented as a continuous training by using new data to train new models and thus over time both replacing artificial data with real data and improving the quality of the generated data by having access to more real data for training. This approach allows to train machine learning models from a much earlier stage where only a small amount of data is available and over time increase the quality of the prediction. This is especially relevant for domains which deal with new scenarios regularly such as the manufacturing domain where new parts are produced with no initial data available or the process model

changes which alters the distribution of the data.

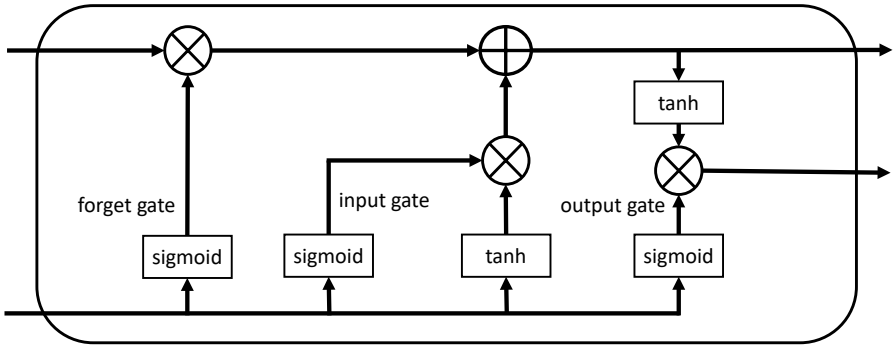


Figure 3.3: Conceptual Illustration of an LSTM

3.1 Concept

The idea behind GENLOG is a modular pipeline that allows easy addition and removal of processing steps. These steps include data extraction, preprocessing, model generation, data generation, post processing and re-embedding of data. This makes it applicable to many different applications. One possible scenario is that it is deployed in a fixed use case but different machine learning models are tested and used for data generation as can be seen in Figure 3.6, while a different scenario could be a fixed data generation model that is applied to different domains and applications to test its ability to generalize to different types of time series data. Depending on the machine learning model used, it can be deployed in a dynamic setting which requires near real time inference and short training times or a more comprehensive model that also takes into consideration a broader spectrum of the data at once including static data from e.g., an event stream and is used in a closer to offline application where the training time required by a model is a lower priority and inference times are not production critical. The definition of interfaces between the steps in the pipeline allows for removal, addition and exchange of processing steps without additional effort. With that, a new data source can be defined and added to the pipeline the same way a new machine learning model can be added or post processing steps that offer new export options or define a interface to an external application through a web service. One main purpose of GENLOG is to boost small data sets with reliable data to use it in existing analysis tools. Therefore the generated data, either the time series data directly or the time series data embedded into the original log file is the produced asset. Another purpose can be to use the trained model as an asset and using it in near real time in a production setting. In this case the log data is streamed and the time series is extracted on the fly and becomes the input for the machine learning model trained in GENLOG and the output is the value that the model predicts for this specific time step based on what it learned about the process and based on the previous

values that were recorded in this production run. It is then possible for every step to calculate how close the recorded values are to the expected values and dynamic time warping, which will be discussed in-depth in section 6.2.2 which allows to calculate a minimum path between two time series and is not sensitive to shifts and speed changes in the data. This enables the system to monitor how much the currently produced part is deviating from what the model learned from previously produced parts, which can directly be used to trigger alarms if the values deviate over a defined threshold. This approach has the potential to actually replace the monitoring model that it was originally designed to boost the small data set for. That is possible if the goal for triggering the alarm is merely to catch the attention of the machinist to investigate if there is a problem with the machine, rather than producing full classification with multiple classes. It is also a more lightweight approach in terms of processing resources required as it continuously monitors each incoming value during machining, rather than continuously taking all of the so far recorded values as input for classification in case of the existing monitoring model. Here a two stage approach would be most beneficial where the new dynamic model monitors if anything abnormal is happening and only then triggers the existing monitoring model to perform classification on the full data of the current production run to provide more information to the user in regard of what the problem is. That is one of the reasons why all produced assets (extracted and resampled time series, models, generated time series and re-embeded log files) are written to disk, as they all have the potential to be assets, rather than just being necessary processing steps in the pipeline.

3.2 Use cases

The motivation behind GENLOG is the specific need and use case of automating the boosting of small data sets to be used in machine learning algorithms. This involves accessing, extracting and processing of event streams to create the small data set to begin with. This is where GENLOG comes in as a full pipeline that takes an event stream as input and outputs multiple event streams with data similar to what it learned from the training data. This allows the use of the data in any existing and future mechanisms and algorithms that take the event streams as input. Therefore this use cases represents the full pipeline as can be seen in Figure 3.4 which offers the greatest benefit to the user as it handles tedious and cumbersome tasks and returns the result to the same format as the input which in turn also allows for multiple iterations of one data set through the pipeline which offers interesting insights into the data transformation that happens which will be covered in Section 6.3.4. This full pipeline however is not the only application of GENLOG as there are also other possible scenarios in which it can be used. These are represented by the partial pipelines which only take a subset of steps from the pipeline. Figure 3.5 shows the case that the input data already exists as a time series. The first step of accessing and extracting the time series from an event stream can be omitted which also opens more research opportunities as benchmark data sets can be used for comparison or the tool can simply be used to compare behavior and performance of different data generation models on a data set that is setup to cover more edge cases

without the need of artificially creating a fake event stream and then extracting the time series from it. One downside of omitting the event stream to time series step is the fact that it is not possible to use any accessory data that might be present in the event stream but is not present in the time series data. However, the intended focus of GENLOG is data that is changing over time and use cases in the realm of monitoring and near real time analysis rather than searching a very wide feature space for potential causes which would render it useless for dynamic scenarios that are present on the shop floor with machinery that is producing parts. The partial pipeline can be used as described above in a from time series to time series application, but can also include the embedding of the generated time series into an event stream. This is done by creating an event stream template which is used to write the time series into. In the context of batch production this is as simple as taking an event stream and declaring it to be a template which will cause the time series aspect of it to be replaced by the generated time series. This can be useful if other software or processors require an event stream as input, however it is important to point out that there is a disconnect between the time series data and the accessory data in the event stream as they were created independently from each other and so far has been used more in the context of creating abnormal data to see how it affects the pipeline.

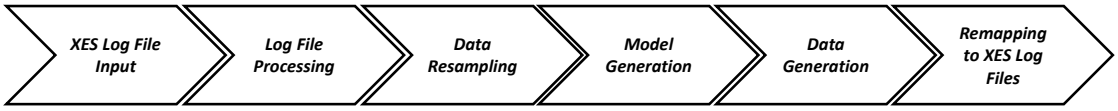


Figure 3.4: XES is the desired input and output format requiring extraction and re-embedding of time series data

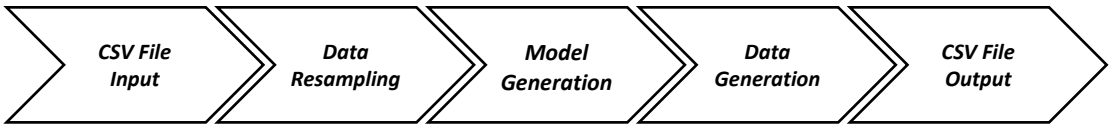


Figure 3.5: CSV is the desired input and output format eliminating the need for extraction and re-embedding allowing for faster pipeline execution

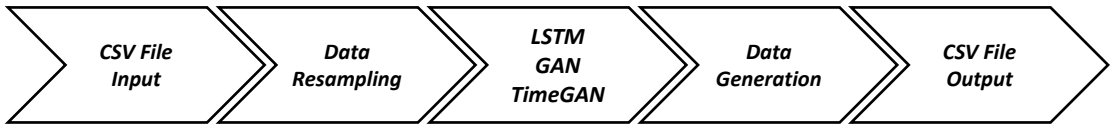


Figure 3.6: Evaluation of the performance of the different machine learning models is desired with focus on evaluation results

4 Prototypical Implementation

This chapter is an extended version of the paper "Generating Reliable Process Event Streams and Time Series Data Based on Neural Networks". [HMRM21]

The GENLOG pipeline depicted in Figure 3.2 and all the processing steps, model training, and data generation are implemented using python. The frontend makes heavy use of d3.js [Zhu17] which is used to dynamically create the user interface based on the batches and metrics and is used to plot the data for validation and visual data analysis as can be seen in Figure 5.1. The machine learning models are implemented with Tensorflow [Zac17] which uses CUDA [SWXC16] for GPU accelerated training. The open-source repository for the tool can be found here[1] and the web-based tool can be used for evaluation here[2]. The core idea is to automate as much as possible and at the same time provide users with a high degree of influence over what data is used as the basis for data generation, i.e., by removing or adding metrics and choosing from which batches or time spans data should be used. This flexibility enables users to react especially to concept drift [BvdAZP14], i.e., changes in the log data that reflect process change and evolution. An advantage of the long short-term memory is that while it does have the ability to use information from its long term memory, the weight it puts on this older data is reduced which is the goal for in the context of concept drift where a smooth progression in which the generated data reflects most strongly the more recent data while considering a longer period of time as the data source is desired. For the generative adversarial network and its variants the preconditioning can also take on the role of providing a similar behavior, but this requires deeper knowledge and experience and does not inherently include it as does the LSTM. Depending on the complexity of the data and the specific requirements, the GAN and its variants can be the more powerful concept and are able to tackle even very complex input whereas the LSTM works best in more dynamic applications where the data is less complex but aspects like short training and inferences times are required or desired. As the pipeline covers both scenarios as it provides both methods for model training and data generation and make it possible to choose the method most suited to the requirements or if it is not clear what the best method would be it can be assessed by trying both approaches which is one of the main purposes behind the evaluation tool as can be seen in Figure 6.18 which is focused on training with smaller data sets and providing training and inference metrics as well as visualizations of the results. Algorithm 1 shows the inputs and outputs of all steps in the pipeline and this also shows possible entry points as the first step can be omitted if the data is already in csv format and it

[1] https://github.com/tohe91/GENLOG
[2] https://cpee.org/genlog/

also shows that the output of the final step is equal to the input of the first steps which means it can also be used iteratively which is tested and discussed in chapter 8. Since GENLOG is built as a pipeline with interfaces between all steps, the assets created in each step are also available for different use cases. The resampled data is available as are the trained models, generated data and the final log files are all written to files for use in external systems.

Log File Processing
Input: Log files (YAML, XES, MXML)
Output: raw csv files

Data Resampling and Binning
Input: raw csv files
Output: resampled/binned csv files

Model Generation
Input: resampled csv files
Output: trained machine learning models

Data Generation
Input: csv files, machine learning models
Output: generated csv files

Remapping to Log File Format
Input: generated csv files
Output: Log files (YAML, XES, MXML)
.

Algorithm 1: Inputs and Outputs of the full pipeline

4.1 Log File Processing

In this step of the pipeline the input are log files [SRHM16, MSS10] which can include a wide range of information about the processes, markup data, server and networking specific information and more besides the time series data that should be extracted. The output of this step are csv files that hold the time series data. Log file formats include XES [20116], MXML [vDvdA05] and YAML [BKEI09]. The log files are processed based on a file structure that must be provided with the file as it is up the the external system and user how the log files are structured and therefore where in the structure the time series data is. Additionally, it might be of interest to provide a list of metrics that should be extracted as it might not be feasible or desirable to extract data, train models and generate data for all metrics present in the original log files. In the case of a specific deployment, the task of providing the file structure and selection of metrics would only be part of the initial setup and only require changes if the structure of the logs files changes, otherwise new log files that are placed in the corresponding log file folder will

automatically be processed and moved through the pipeline until the model generation step without further input by the user required. Algorithm 2 shows the input, output and processing steps for this part of the pipeline.

Input: Log files (YAML, XES, MXML)
Output: raw csv files
load data into memory
while *not end of file* **do**
  parse structure
  if *metric included in selection* **then**
    append to time series
  end
end

Algorithm 2: Log File Processing: Loading, processing and extracting time series data from a log file

4.2 Data Resampling and Binning

The input of this step are csv files holding the time series data, one file for each metric extracted. The output is resampled and optionally also binned and written to csv files. The reason why the input files have to be separated by metric is the fact that the frequency of measuring the metrics is inconsistent both over time and between metrics. The sample rate can differ between each metric and the measurements itself also vary depending the measuring device, server, networking and load on the it infrastructure and often arises because log data is written by the same device that is executing the process which can take resources away from log creation and towards the execution of the main process as well as data being transmitted through a network. To allow multiple metrics to be stored in one csv file and to unify the data for further processing, the data is resampled to a common sample size. If a generative adversarial network is used as the model, also data binning is performed as the model expects the same size for each sample. In the case of the LSTM, this step can be omitted. The scale of the values matters in multivariate data as the metrics together form the features for model training. In a univariate model, feature scaling is not necessary as only one metric is used as the feature. To train a multivariate model it is advised to normalize the data because some loss metrics can introduce a bias otherwise [JS11] or converge more slowly as it is the case with gradient descent. Algorithm 3 shows the input, output and processing steps for this part of the pipeline.

Input: raw csv files
Output: resampled/binned csv files
load data into memory
determine smallest sample rate
for *all time series files* **do**
  resample data to the common sample rate
  if *model == GAN || TimeGAN* **then**
    perform data binning to common bin size
  end
end

Algorithm 3: Data Resampling: Resampling the data to a common sample rate and in case of a GAN based model also perform data binning

4.3 Model Generation

The input in this step are the resampled and potentially binned csv files holding the time series data and the output are trained machine learning models (LSTM, GAN, TimeGAN) which are written to disk as compiled files. The models can be loaded into memory on demand which makes the inference for data generation easier. The idea behind the individual model architectures and parameterizations is not to adjust them for every metric and every new incoming data but rather to create models that generalize well and can deal well with volatile data as the process that generates it evolves. The architectures and parametrizations are chosen by cross validation and grid search. Optionally the user can chose to have the inference result of the trained models be displayed in the user interface to assess if the chosen machine learning model is a good fit for the data before proceeding to the Data Generation step. Algorithm 4 shows the input, output and processing steps for this part of the pipeline.

Input: resampled csv files
Output: trained machine learning models
User selects subset from the data for training
load time series data into memory
User selects (LSTM, GAN, TimeGAN)
for *all time series files* **do**
  train models
  compile and write model to disk
  if *verbose == true* **then**
    display chart for User with inference result overlaid by original data
  end
end

Algorithm 4: Model Generation: User chooses data and model and has the option of reviewing the inference results

4.3.1 LSTM

Preparing the data for training of the LSTM is done by creating a feature and a target vector. The target vector is created by shifting the values of the feature vector such that the target value is always the next value relative to the feature vector which is what the model is supposed to predict. The model is created by defining the size of the LSTM layer and its activation function. The activation function used is the rectified linear unit (ReLU) [HSS15] which provides better gradient propagation [GBB11] together with the Adam optimizer [KB14]. The size of the output layer is the number of features used and is one for a univariate time series. The loss function used is the mean squared error [CCH15]. A callback for early stopping [BHR18] is added to increase performance by monitoring the loss function and stopping the training before max iterations are reached if the loss function did not change significantly in the last number of iterations. Listing 4.1 shows the code for training the univariate LSTM model.

```
for i in range(num_of_models):
    model = Sequential()
    model.add(LSTM(15, activation='relu',
              input_shape=(n_steps, n_features)))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    callbacks = [EarlyStopping(monitor='loss', patience=10)]
    model.fit(X, y, epochs=100, verbose=0, callbacks=callbacks)

    yhat = model.predict(X_input, verbose=0)
```

Listing 4.1: LSTM model for univariate training

4.3.2 GAN

Listing 4.2 shows the code for training the generator network of the GAN model. It is a deep neural network architecture as it consists of multiple hidden layers. The hidden layers decrease in size, which represents different levels of focus on the data. This allows the model to learn about semi global patterns as well as learn more intricate details in the data. The weights of the hidden layers are initialized with the he_uniform [HZRS15] distribution, meaning the values for the weights are drawn from a uniform distribution, which results in slightly faster convergence compared to just using ones or zeros. The rectified linear unit layers between dense hidden layers set the negative values from the activations to zero and keep the positive values untouched which increases convergence speed and helps mitigating both vanishing and exploding gradient problems in backpropagation. This architecture was chosen after extensive testing using different kernel initializations, activation functions, number of layers, sizes of layers and optimizers. The activation function of the last layer in the discriminator network is the sigmoid funciton to allow the use of binary cross entropy as the loss function, which is well suited as the goal is to distinguish real from generated data samples. The chosen architecture is able to learn the most complex data in the evaluation which is the metric with the highest polynomial degree. Using the architecture that performs well over all data also means

that the training cost is very high as the same complex model architecture is also used on simpler data samples. Listing 4.3 shows the code for training the discriminator network of the GAN model. The main difference in the model architecture between the generator and discriminator is the output layer, as the generator is producing a time series and the discriminator is producing a binary classification, i.e., real or generated sample. Listing 4.4 shows the code for joint training of the generator and discriminator network of the GAN model including loss and optimization functions.

```
def define_generator(latent_dim, n_outputs=2):
    model = Sequential()
    model.add(Dense(1000, activation='relu', kernel_initializer='
    he_uniform', input_dim=latent_dim))
    model.add(LeakyReLU(alpha=0.3))
    model.add(Dense(500,  kernel_initializer='he_uniform'))
    model.add(LeakyReLU(alpha=0.3))
    model.add(Dense(250,  kernel_initializer='he_uniform'))
    model.add(LeakyReLU(alpha=0.3))
    model.add(Dense(100,  kernel_initializer='he_uniform'))
    model.add(LeakyReLU(alpha=0.3))
    model.add(Dense(50,  kernel_initializer='he_uniform'))
    model.add(LeakyReLU(alpha=0.3))
    model.add(Dense(n_outputs, activation='linear'))
    return model
```

Listing 4.2: Generator model of the generative adversarial network

```
def define_discriminator(n_inputs=2):
    model = Sequential()
    model.add(Dense(1000,  kernel_initializer='he_uniform',
            input_dim=n_inputs))
    model.add(LeakyReLU(alpha=0.3))
    model.add(Dense(500,  kernel_initializer='he_uniform'))
    model.add(LeakyReLU(alpha=0.3))
    model.add(Dense(250,  kernel_initializer='he_uniform'))
    model.add(LeakyReLU(alpha=0.3))
    model.add(Dense(100,  kernel_initializer='he_uniform'))
    model.add(LeakyReLU(alpha=0.3))
    model.add(Dense(50,  kernel_initializer='he_uniform'))
    model.add(LeakyReLU(alpha=0.3))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam',
            metrics=['accuracy'])
    return model
```

Listing 4.3: Discriminator model of the generative adversarial network

```

def define_gan(generator, discriminator):
    discriminator.trainable = False
    model = Sequential()
    model.add(generator)
    model.add(discriminator)
    model.compile(loss='binary_crossentropy', optimizer='adam')
```

```
8     return model
```

Listing 4.4: GAN model for joint training of the generative adversarial network

4.3.3 TimeGAN

The training of TimeGAN includes three main steps that are performed in series. First the embedding network is trained which is a variational autoencoder (VAE) that is responsible for creating an optimized latent space for the generator to work in as well as conditioning the space to learn temporal relationships.
Second is the training with supervised loss only, additionally introducing a stepwise supervised loss using the original data as supervision, encouraging the model to capture the stepwise conditional distributions in the data. This adds more information than just differentiating between real or synthetic samples which is a binary decision and rather learning with a loss function on the real data.
Third is the joint training in which the VAE and the generative adversarial network are trained together with both supervised and unsupervised loss. This results in the attempt to minimize the loss for the embedding network, the generator network both supervised and unsupervised and the discriminator loss.
Figure 4.1 [Jan20] shows the architecture of the approach and its different components and how they interact with each other. The serial execution of the three steps to train the VAE and the GAN first separately and then joint, combined with the minimization of all the loss functions results in a very expensive training process. The details on the training performance and resource requirements are discussed in Section 6.3.3 and compared with the other approaches in Section 6.4. Algorithm 5 shows how TimeGAN approaches the learning of the data through its different networks to generate synthetic time series data.

Input: time series as csv file
Output: generated synthetic csv files

1. Map between Feature Space and Latent Space
2. Generate Synthetic Latent Codes
3. Distinguish between Real and Synthetic Codes
4. Compute Reconstruction, Unsupervised, and Supervised Losses
5. Update via Stochastic Gradient Descent
6. Synthetic Data Generation

Algorithm 5: TimeGAN training: Steps taken to learn the data from the provided time series in the latent space and generate synthetic data as the output

TimeGAN is implemented in TensorFlow 1.15 and includes a predictive score, discriminative score and visualizations for assessing the quality of the results. It offers the following

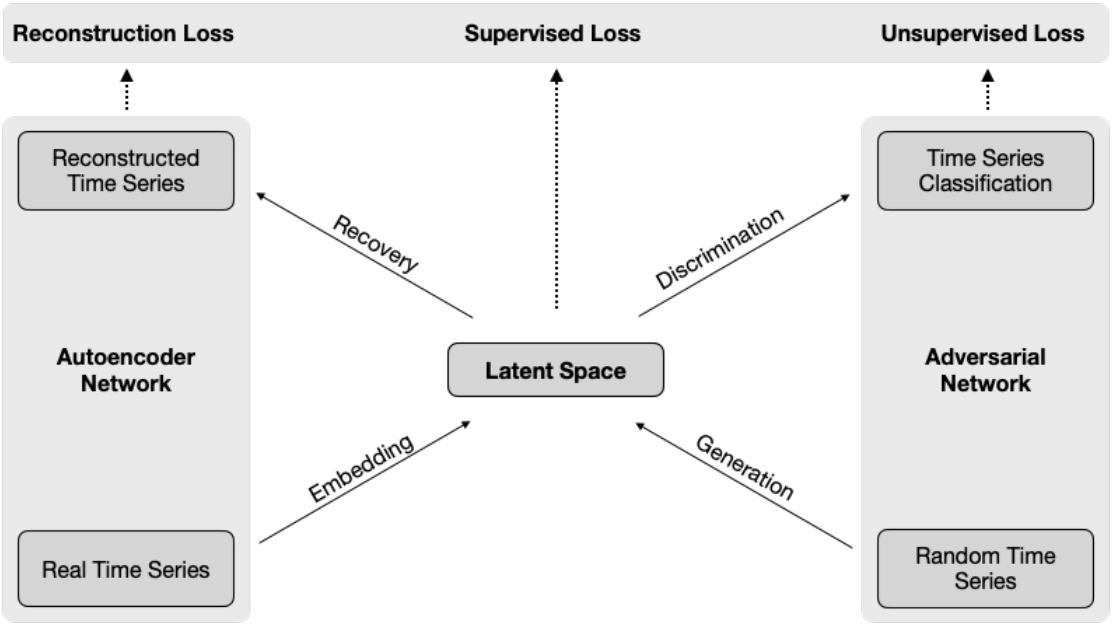


Figure 4.1: Conceptual illustration of TimeGAN created by Stefan Jansen in 2020 showing the embedding and adversarial network with its loss function

parameter options for training:

- module: gru, lstm, or lstmLN
- hidden_dim: hidden dimensions
- num_layer: number of layers
- iteration: number of training iterations
- batch_size: the number of samples in each batch

The most impactful parameter is module, which determines which model architecture is used for the layers for the generator and disciminator model. Both gated recurrent units and long short-term memories are recurrent neural networks and are similar in many ways, however they differ in their architecture. The LSTM uses cell states whereas the GRU uses a hidden state to transfer information and the LSTM uses three gates (input, output and forget) whereas the GRU only uses two gates (reset and update). These architectural differences result in the GRU usually having faster training times, but since they are quite similar to each other it is usually a good idea to try both and to see how they perform in the specific use case. The number of layers and hidden dimensions define how many hidden layers make up the networks and how big they are. Iterations and batch size define the number of epochs and the size of the batches used in the training of the networks. The loss function used for the generator and discriminator is sigmoid cross

entropy and the mean squared error for the supervised and embedder loss. All networks use Adam optimizer for optimizing gradient descent.

4.4 Data Generation

To generate new data, both a trained model and an original data instance are chosen uniformly at random from a set that can be defined by the user through the user interface, see Figure 5.2 and the predict method of the model is called with an original data instance as the input, see Listing 4.5. What makes this approach so powerful are the selective options that the user has. Often data can be grouped temporally by changes in the process and thus create data bins that correspond to a specific state of the process. By grouping the input data and the resulting trained models by the state of the process and other groupings like batch production or medical trials, the user can choose what should be the base for the newly generated data. This can be achieved by choosing the state of the process for each feature individually. In case there is no natural grouping apparent in the data, the user can chose a time window for each feature to represent the data basis for the data generation. This gives the user fine grained control over the generated data which can be beneficial if it is known that a specific feature was invalid or undesired for a given state of the process. Another approach the user can take is to specify if the generated data should be similar to the most recent data or if it should cover a broader time span. This can eliminate the need for manual pruning of data that is often required when using data analysis algorithms on raw log data. It can also be useful if manual pruning results in a data set that is too small for use in data analysis algorithms to produce meaningful results. By taking the pruned data and generating more data based on its distribution, a more reliable data set is created. Algorithm 6 shows the input, output and processing steps for this part of the pipeline.

```
# Inference for LSTM
X, y = split_series(raw_seq, n_steps)
X_input = X.reshape((X.shape[0], X.shape[1], n_features))
yhat = model.predict(X_input, verbose=0)

# Inference for GAN
x_input = randn(latent_dim * n)
x_input = x_input.reshape(n, latent_dim)
X = generator.predict(x_input)

# Inference for TimeGAN is simply calling the timegan() function
```

Listing 4.5: Inference for data generation for LSTM and GAN

Input: trained machine learning models, resampled csv files
Output: generated csv files
User selects subset from the data as input for inference
    load time series data into memory
User selects trained models (LSTM, GAN, TimeGAN)
    load machine learning models into memory
for *all time series files and models* **do**
    generate time series data for model and time series pairs
    if *verbose == true* **then**
        display charts for User with generated data overlaying original data
    end
end

Algorithm 6: Data Generation: User chooses data and model and has the option of reviewing the generated data

4.5 Remapping to Log File Format

In order to go from validating the generated data to using it in existing workflows it is important to map the data back to its original log file format. As the data is extracted and resampled in the beginning of the pipeline, now the steps need to be reversed by sampling it back and by embedding it into the original log file format. Templates can be created for this purpose, or the originating log file can be used for embedding. The templates can be either generic empty templates that the generated data is then embedded in or - in case of complex log files that also include other information that were not altered by our pipeline - a template is created from an existing log file and the values and timestamps are replaced with the generated data (complex template). When using an empty template the original mean sample rate can be used for each metric. For the complex template, the sample rate has to be adjusted to match the sample rate of the template. A detailed explanation will be given in the next section where the GENLOG approach is applied to the production of a metal part from a CNC turning machine on the shop floor. Algorithm 7 shows the input, output and processing steps for this part of the pipeline.

Input: generated csv files
Output: Log files (YAML, XES, MXML)
load generated time series data into memory
if *template exists* **then**
| resample time series data to match template
else
| resample time series data to match original log file
end
for *all time series files* **do**
| embed reampled time series data into log file
end

Algorithm 7: Remapping to Log File Format: Resampling time series data to match log files and embedding it into it

5 Evaluation in a Real World Scenario

This chapter is an extended version of the paper "Generating Reliable Process Event Streams and Time Series Data Based on Neural Networks". [HMRM21]

The GENLOG pipeline depicted in Figure 3.2 is applied in a production environment based on log data from the Competence Center for Digital Production (CDP)[1]. An EMCO "MaxxTurn 45 SMY" machines a part in the GV12 production. GV12 is a *"part of a valve from a gas motor. The complexity of this product is caused by the small tolerances, which necessitates the complex overall scenario"* [MPRE19]. The production process is executed and logged by the Cloud Process Execution Engine (CPEE) [MSS10, SRHM16]. The CPEE also logs metrics of the machine like the torque and load of the motors for each axis as well as spindle speed, spindle load, and other metrics from the turning process together with the process execution steps and meta data [EFMR19]. The produced log file originates in XES format and is then stored as a YAML file for further usage as can be seen in Listing 5.1. As YAML is used in other existing systems and analysis tools, the pipeline also includes this option as the idea is to be able to generate log files that can be used directly in existing workflows without the need for additional processing to change the file format. The production of the individual parts is organized in batches. Major changes to the production process are typically made between batches while it is common to make smaller changes also within a batch. These changes are typically directly related to machining the part to improve tolerances i.e., the quality of the part while a major change in the process could be the addition of a measuring step that is performed by the robot arm that is unloading the part from the machine which introduces the entire movement chain of the robot arm and the results of the measurements to the log files. Only metrics within the machining process are evaluated and aspects like the loading, unloading, measuring and post processing of the part are omitted as they do not contribute as much to the resulting quality of the part and aspects like tool degradation, tool failure and other production critical events usually happen within the machining process.

```
- ID: ns=2;s=/Channel/MachineAxis/aaLoad[u1,1]
        source: opcua
        name: Axis/X/aaLoad
        description:
        path: /Object/Sinumerik/Channel/MachineAxis/aaLoad[u1,1]
        "value: 25.5615234375"
        "timestamp: '2019-05-02T13:00:18.795+02:00'"
        meta:
```

[1] http://acdp.at

```
        StatusCode: Good
        ServerTimestamp: 2019-05-02 15:20:21 +0200
        VariantType: VariantType.Double
        ClientHandle: 42
    - ID: ns=2;s=/Channel/MachineAxis/aaLoad[u1,2]
      source: opcua
      name: Axis/Y/aaLoad
      description:
      path: /Object/Sinumerik/Channel/MachineAxis/aaLoad[u1,2]
      "value: 5.328369140625"
      "timestamp: '2019-05-02T13:00:18.797+02:00'"
      meta:
        StatusCode: Good
        ServerTimestamp: 2019-05-02 15:20:21 +0200
        VariantType: VariantType.Double
        ClientHandle: 42
```

Listing 5.1: Section of a YAML log file including time series data as value and timestamp

To allow the user to get an understanding of the data and to make a comparison between metrics, batches and aggregation types, a visual tool was developed to explore the data and to understand which metrics are of interest and should be boosted by GENLOG to be used in the monitoring model. Figure 5.1 shows the frontend of the tool. On the left side, the user can orchestrate the visualization by choosing different types, batches and modes to be displayed in the chart on the right side while on the bottom right the standard deviation is visualized over time for each metric to help with finding correlations in the data. An example is the torque and load for each axis, as they are negatively correlated which to an domain expert might make intuitive sense. The visualization helps to understand the data and to find anomalies. These anomalies can manifest themselves differently, by either being an isolated occurrence or by also affecting everything after the occurrence, e.g., a tool head breaks and the load is much lower than expected from that moment forward as the tool does not make contact with the part.

5.1 Log File Processing

The log files created by the turning machine shown in Listing 5.1 include all the metrics and information about the state of the machine, the state of the process, and meta data from the servers and production environment as well as the time series data for the different motors of the machine. The only requirement for the GENLOG pipeline to run is to provide the log files and their internal structure if it differs from the default and enter the metrics that should be extracted. If no metrics are specified all of the available metrics are selected by default, i.e., the user does not have to set any parameters. The pipeline then handles the extraction, resampling, and model generation without supervision or further input. The YAML files are processed as a stream and the combination of the provided structure and list of metrics that should be extracted enables the algorithm to extract all occurrences of a metric with its value and timestamp. With this approach the file only has to be processed once to extract all metrics which accelerates the data import.

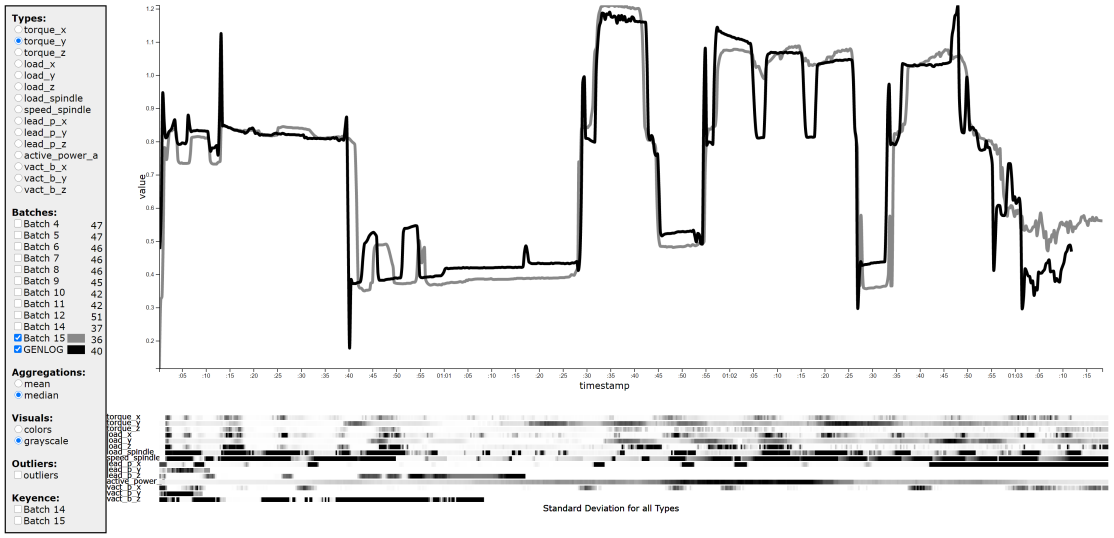


Figure 5.1: Data visualization tool intended for use before starting the pipeline to choose which batches and metrics should be selected and to compare the generated data with the original data

Applying GENLOG in this real world scenario showes several potential difficulties and the importance of encapsulating each step as an interface rather than creating a monolithic algorithm that has to run from start to finish. A problem that occurred is that at some point in the data logging process, the timestamp format was changed which required a more adaptable processing of the timestamps that can handle different timestamp formats and time zones. Another problem that had to be addressed was the fact that for the first 9 batches, one of the metrics was mislabeled which was corrected for subsequent batches which initially resulted in very faulty generated data as the training data was a combination of the metric in question with an completely unrelated metric. This required a hotfix that handled this metric differently for different batches. Due to changes in the process, new metrics were introduced at different points in time which also had to be accounted for as it made it necessary to check which metrics were present in which log files before processing. It might be of interest to investigate the viability of using clustering algorithms designed for time series like tslearn [TFV+20] to validate if a newly extracted time series for a metric matches with previous samples and if that is not the case, if there is another cluster that fits this new sample better. That could enable the pipeline to automatically trigger an alarm to the user if there is a anomaly like the relabeling of an existing metric which was the case in this real world scenario.

5.2 Data Resampling and Binning

The data is extracted and stored as time series data for each metric. The GENLOG pipeline enables the user to use visual analysis to find outliers, view different aggregations

of the data, and analyze how the metrics change over time and through changes in production by plotting the data. The production of the parts is organized in batches which creates a natural grouping that is used to organize the log data. Figure 5.1 depicts the output of GENLOG to find changes in the process based on duration and shape of the plots. In the background, the data is resampled to a common sample rate which is at least as high as the highest rate between all the metrics to make sure no data is lost in the resampling process. Data binning is used to group the samples into multiple bins based on their length which helps with mapping of the models and samples in the data generation step as the goal is to maximize the information extracted from both the trained model and the input data. Due to the fact that the log creation is performed on the machine itself which in turn means that log creation has to share computing resources with the manufacturing process, it can happen that there are gaps in the log file creation which have to be handled. By default the last value written before the gap would fill the time span until the next value is written again after resources allowed renewed log file creation. As this is not representative of what actually happened in that time span, the last value before the gap is set to a predefined gap signaling value which causes the resampling algorithm to fill the space with the gap signaling value. This ensures the ability to handle the missing values depending on what the user wants to achieve. Rules can be set for example a duration can be defined and all gaps that are shorter will be ignored, meaning the last written value will be used to fill the gap and if the gap is longer than the predefined duration the gap signaling value is written which can then be set to be ignored by the model training.

5.3 Model Generation

The models are generated automatically as part of the pipeline and a fine grain model generation is used to preserve the characteristics of the individual production runs and no aggregation or generalization is done, which means that for every production run a model is trained and the time series for all runs is used as input for the model. This is similar to how cross validation works where a portion of the data for testing is changed for every training. In the case of model training it ensures the maximum information extraction which is crucial in the case of small data sets. If more data is available the user can chose in the data generation step which subset of the data should be used to generate more data. Using an LSTM in this context is powerful as the different metrics have very different characteristics with some following a low polynomial function while others follow a high polynomial function with a high number of oscillations. This can be covered well by the LSTM because it predicts one data point at a time by using both recent information like an RNN and also older information which gives it its long term memory characteristics. Adding the early stopping functionality decreases the total run time of the model creation significantly as some metrics result in very small changes in the loss function earlier than others. For reference the creation of over 5800 LSTM models took just 5 hours on a standard laptop (i7-7700HQ CPU, 16GB RAM, NVIDIA GeForce GTX 1060) with CUDA (graphics accelerated training) enabled. The architecture and

parametrization of the models were automatically selected based on grid search cross validation which was a model that can handle the most complex metric well and the less complex metrics did not have to be trained longer than necessary because of the use of early stopping. The training time required for the standard GAN and for TimeGAN were orders of magnitude higher and early stopping could not be used as effectively as with the LSTM due to the higher complexity of the generator and discriminator model which requires much more computation before the model learns the data to the desired degree. The details on training time and performance for each architecture will be discussed in Chapter 6.

5.4 Data Generation

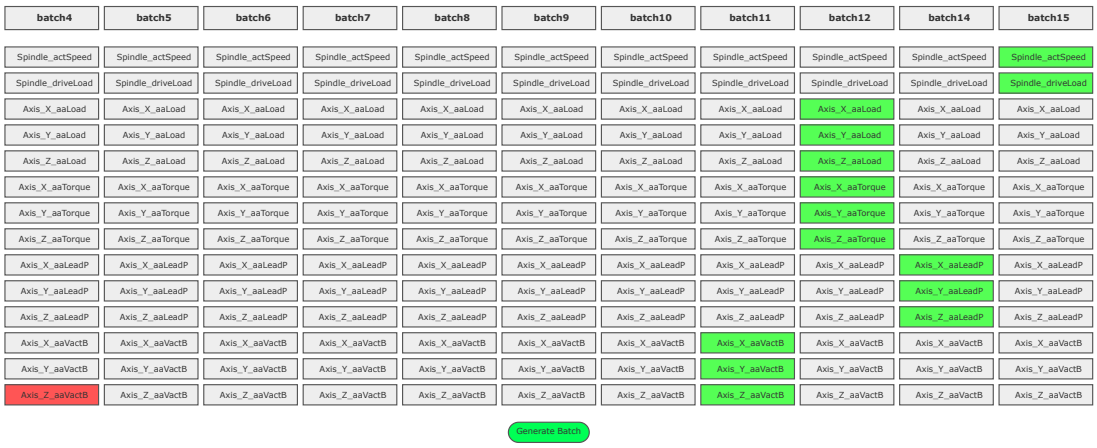


Figure 5.2: User Interface for data generation, **red**: model not trained yet - **gray**: model ready - **green**: model active for data generation

At this point the user is presented with a dashboard, depicted in Figure 5.2 showing all the metrics for the individual batches. In the generic case this is where a temporal window could be chosen if there was no grouping in batches or process changes made beforehand. The user interface shows for which metric-batch combination the models have already been trained and are ready for usage. The user can pick which batches should be the data basis for each of the metrics. In case only the most recent data should be used as the basis the user can simply select the latest batch which activates all metrics for this batch and the generated data will represent this batch in all metrics. In case of a new process that only has very few data instances produced at this point, the selection can be based on the entire available data. In the scenario that multiple batches are chosen, the individual models and input values are chosen uniformly at random and are combined at random. As an example, batch14 and batch15 are chosen for a metric, there can be four possible combinations, model14-data14, model-14-data15, model15-data14, model15-data15, which produces a high degree of variance even if the original data set

size was small because models and input data can be combined together. It is important to note that the generated data can be biased or overfitted if the input data set is very small and the data instances are very similar to each other and do not cover the spectrum well that is generally common in the process. However, the goal in this real world scenario is to be able to use the existing machine learning monitoring model from the start when a new part is entering production. The production scenario is similar to many data sets in the medical field where data sets are unbalanced meaning most parts produced are ok and there are no major incidents occurring in the production, similar to medical data sets where the majority of tested patients do not have the disease in question. If only classification is used to determine whether there is an incident in the machine or not, the model would suffer from the problem that it would be very biased towards a production run being ok as this is the majority of the input data. Using RNN based models, allows to determine for every time step how much it is deviating from the expected value which allows to monitor the production as soon as a part is produced successfully. The idea behind the boosting of the data set is that it happens continuously as the production progresses, new models are trained and the variance in the models and subsequently the generated data increases which makes the monitoring model less prone to wrong classifications as it is aware of variations in the production which are within the thresholds and are not considered problems. If a part is produced for long enough the need for boosting the data set can become obsolete and only real data is used without the need of synthetic data as is illustrated in Figure 5.3.

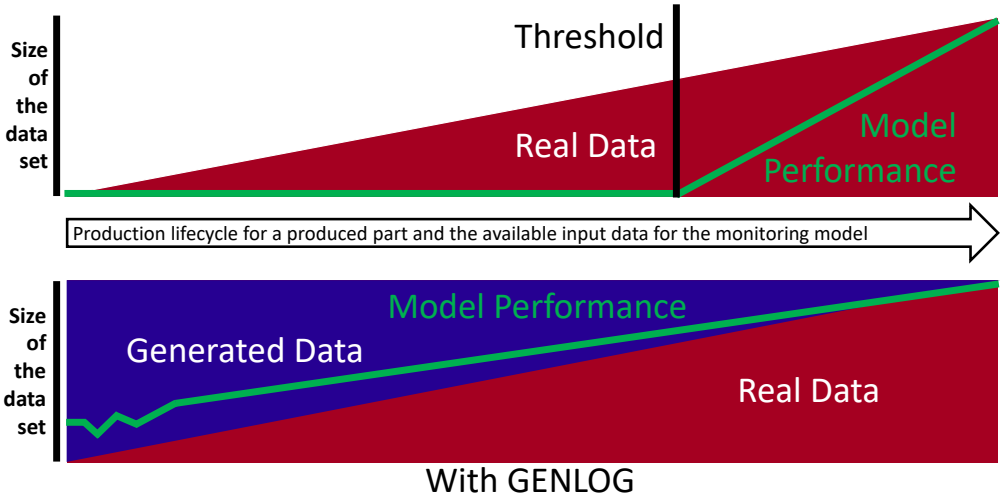


Figure 5.3: Boosting the data set allows for an earlier usage of the monitoring model as more data is available from the beginning. The reliability is low in the beginning but quickly stabilizes and then steadily increases over time

5.5 Remapping to Log File Format

The previous step is sufficient to validate how well the newly generated data represents the same distribution as the original data. In order to become useful for data analysis, the data needs to be embedded into the original file format. In the remapping stage the template with the same batch number and bin size is chosen and the generated data is embedded into it. The embedding is done by first calculating the duration of the process from the template in milliseconds. Second the number of logged values for each metric are summed up, thus resulting in the duration of the process and the number of logged values for each metric which allows to calculate a sample rate for each metric. Third the generated data of each metric is resampled with the calculated sample rate and therefore the generated data has the same amount of entries as the template for each metric. Fourth both value and timestamp for each metric are embedded into the template log file. The timestamp was created by taking the first timestamp of the template and then using the sample rate to increase each timestamp accordingly. As a result the generated log file has the same structure, length, meta data and similar variance as the original log file. Using templates allows for consistent and predictable results which allows comparisons between different pipeline runs with different parameters, machine learning models and data samples, which is crucial for meaningful analysis. In the production environment, instead of using templates, the log file is used that the model originates from. This allows for a much closer match between time series data and all other data present in the log file that relates to the machining process. A static template would not evolve with the process which would cause the time series data to drift away from the all other data in the log file which makes it less useful and potentially cause conflicting data.

6 Evaluation

This chapter is an extended version of the paper "Generating Reliable Process Event Streams and Time Series Data Based on Neural Networks". [HMRM21]

6.1 Evaluation Tool

The open-source web-based evaluation tool[1] is designed as a research tool to generate all the evaluation metrics and diagrams automatically such that when changes are made to the pipeline or new data is used, all assets relevant to the thesis are generated as a result. This removes the majority of the manual work in order to obtain the assets and allows for more rapid development and comparability between changes of code or data. It also allows others to verify the findings and use new and unseen data as input to evaluate its ability to generalize to data that it was never exposed to. It creates a layer of transparency as the tool that is used to create the assets for this thesis is publicly available to anyone. Due to performance limitations and the resulting extensive training times associated with it, it is at the time of writing not possible to train GAN or TimeGAN models in the web-based tool described in Section 6.1.1. It is however possible to build the project locally as described in Section 6.1.2 as it is open-source and available on Github[2].

6.1.1 Usage of the Tool

The web based evaluation tool shown in Figure 6.1 is designed to analyze the input log data, visually evaluate the time series results for the generated data in relation to the original data, examine the evaluation metrics and finally analyze the newly create embedded log files. The user has the ability to upload new log files or choose one or multiple log files from the list of uploaded files. The progress for each file can be monitored and the status indicates the current step in the pipeline and the progress within that step is displayed. Active pipelines can be canceled and finished training runs can be evaluated, downloaded or deleted. It is also possible to execute the GENLOG pipeline, download the newly generated log files, upload them and execute the pipeline again. This can be done repeatedly and offers interesting insights as discussed in Section 6.3.4. The tool is mainly written in python like the GENLOG pipeline itself and uses Flask and Flask Tables to serve the data to the frontend. The pipeline is executed within a Jupyter notebook and the the html export of the notebook is presented to the user for the evaluation including the pipeline steps taken, plots showing the generated data overlaid by the original data

[1] https://cpee.org/genlog/
[2] https://github.com/tohe91/GENLOG

as well as the evaluation metrics.
Figure 6.18 shows an example for the evaluation results and includes the individual generated data samples overlaid by the originating time series as well as all the evaluation techniques that will be discussed in this chapter. This allows the reproduction of the evaluation results and execution of further experiments through a simple web interface.

6.1.2 Installation of the Tool

The tool is open source and available on Github[3] and can also be run locally by either running

```
python app.py
```

which will present the tool on part 8000, or it can also be run within docker by running

```
docker-compose up
```

which exposes the tool on port 5000 by default. Using docker has the advantage that it contains all the dependencies within the image and only requires docker and docker-compose to be installed on the host system.

6.2 Evaluation Techniques

This section describes the different evaluation techniques used, why they were chosen and their pros and cons for evaluating the generated time series data. Section 6.2.5 compares the different approaches and discusses how they correlate to each other.

6.2.1 Euclidean Distance

The euclidean distance is used as a baseline similarity measure as it is very straight forward to implement given that the data is sampled to a common sample rate. Calculating the normalized mean euclidean distance for each time series allows for an easy and direct comparison of the similarity between each generated sample and originating time series of the used model. By computing the mean over the set of distances makes sure that the slight differences in length of the time series is not skewing the results and normalization is used to allow the comparison between different metrics.

- **Advantages**
  - Ease of implementation
  - Direct comparability within a metric and with other metrics
  - Fast computation
- **Disadvantages**
  - Differences in the length of the time series affects results
  - Not robust against shifts and speed changes in the time series

[3] https://github.com/tohe91/GENLOG

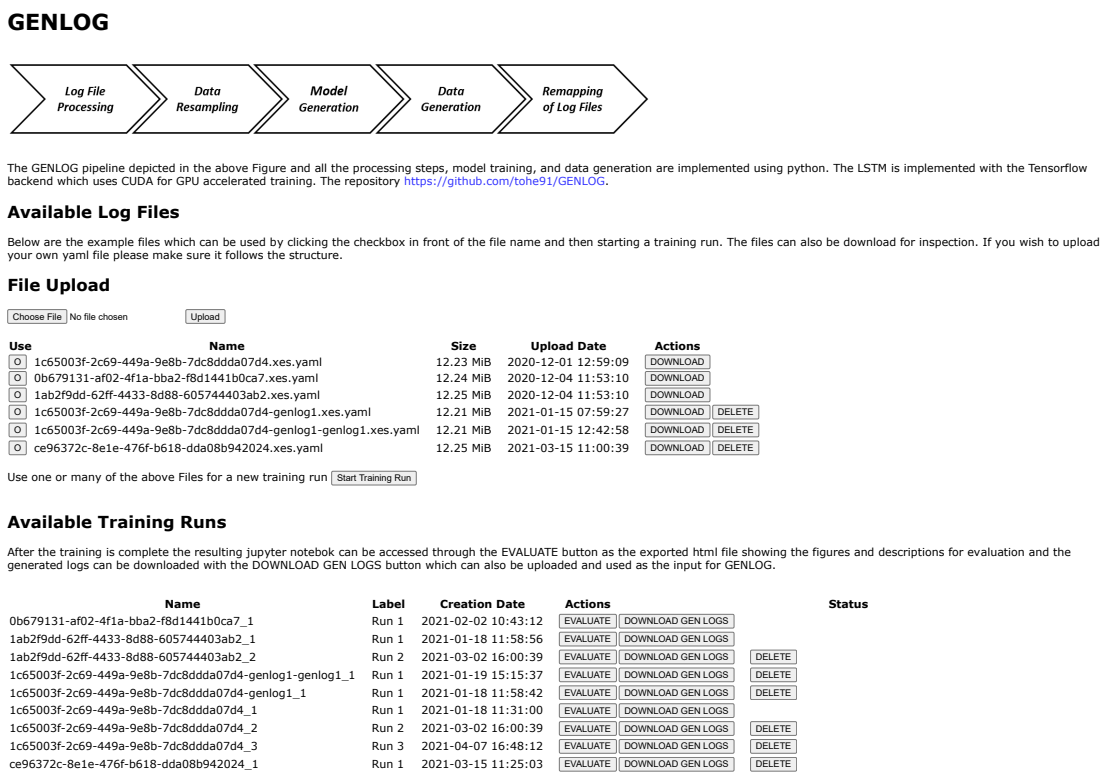


Figure 6.1: Web-based tool allowing the upload and download of log files and the executing of the GENLOG pipeline

6.2.2 Dynamic time warping

Dynamic time warping (DTW) [Mül07] is an algorithm designed to measure the similarity between two time series without the constraint that they have to have the same speed or be perfectly aligned. As discussed in the previous section, simply calculating the distance between two time series for each index is vulnerable to shifts and changes in speed of either of the time series. DTW removes these constraints by dynamically warping the time series to find the optimal match for each index. This makes it significantly more robust in many applications including the application that was evaluated as it is common for the machine operators to do speed overwrites in case they notice unusual sounds or think that the original speed is too conservative. It can also happen that the machine is briefly stoped in case there is an unforeseen problem or a change in the process can result in slight timing changes. For all these scenarios an algorithm that can cope with these mutations is desirable. The results can be plotted showing where in time the two time series differ and the minimum path is calculated which can be used to compare the different samples as can be seen in Figure 6.2. Normalization is necessary in order to make a comparison between different metrics. The computational cost is significantly higher compared to the euclidean distance but this can be reduced by calculating the

pairwise distance over the entire vector instead of the iterative implementation of the standard DTW.

- **Advantages**
  - Robust against shifts and speed changes in the time series
  - Not affected by small differences in length of the time series
  - Includes plotting for visual evaluation
- **Disadvantages**
  - High computational cost
  - Increased implementation effort or requirement to evaluate pre-existing implementations

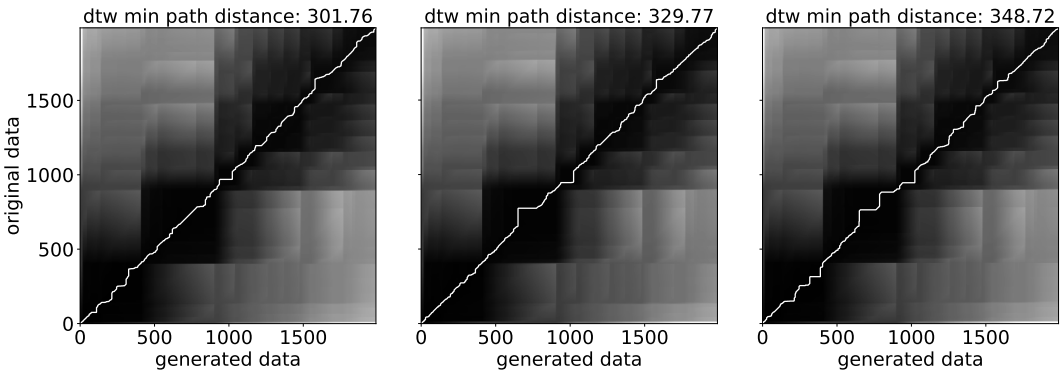


Figure 6.2: Visualization for three samples plotted against the originating data of dynamic time warping with minimum path through the warped data

6.2.3 Correlation Coefficients

Correlation coefficients are numerical measures of a specific type of correlation. The measurements range from -1 being perfectly negatively correlated, through 0 having no correlation at all, to 1 being perfectly positively correlated. In the context of data generation, the goal is to have a high positive correlation without suffering from overfitting. A correlation of 1 would mean that the data was merely copied, while a low or even negative value would mean that the model did not learn the data well. In order to actively utilize the measurements it is necessary to define a range for which a generated sample is rated as good. Whether or not it is possible to judge a generated sample purely by its correlation coefficient will be discussed in Section 6.2.5.

Pearson correlation coefficient

The pearson correlation coefficient (pearson) [BCHC09] calculates the linear correlation between two data sets. The range from -1..0..1 is achieved via normalization of the covariance as can be seen in Eq. 6.1 and is defined as the covariance of x and y is divided by the product of the standard deviation of x and y
where
n is sample size,
x_i, y_i are the respective data samples,
$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ are the sample means

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{6.1}$$

Spearman rank correlation coefficient

The spearman rank correlation coefficient (spearman) [Zar05] evaluates the monotonic relationship between two data sets compared to the linear relationship of pearson and it is using the rank of the values instead of using the data directly as with pearson which can be seen in Eq. 6.2
where
ρ is the Pearson correlation coefficient based on the rank of the values
σ_{rg_X} *and* σ_{rg_Y} standard deviations of the rank values
$\mathrm{cov}(\mathrm{rg}_X, \mathrm{rg}_Y)$ is the covariance of the rank values

$$r = \rho_{\mathrm{rg}_X, \mathrm{rg}_Y} = \frac{\mathrm{cov}(\mathrm{rg}_X, \mathrm{rg}_Y)}{\sigma_{\mathrm{rg}_X}\sigma_{\mathrm{rg}_Y}} \tag{6.2}$$

Kendall rank correlation coefficient

The kendall rank correlation coefficient (kendall) [Abd07] also uses ranks instead of raw data similar to spearman and cares about the similarity of the ranks rather than requiring the same rank and uses concordant and discordant pairs which are defined as:

concordant: ($x_i > x_j$ and $y_i > y_j$) or ($x_i < x_j$ and $y_i < y_j$)
discordant: if condition for concordant does not hold

The definition for kendall can be seen in Eq. 6.3, where

$$\tau = \frac{(\text{number of concordant pairs}) - (\text{number of discordant pairs})}{(\text{number of concordant pairs}) + (\text{number of discordant pairs})} \tag{6.3}$$

6.2.4 Visual Evaluation

Visual Evaluation is a manual evaluation technique and is dependent on experience and domain knowledge. Most users will have a good intuition whether or not a set of generated samples looks inline with the original data without any domain knowledge. Adding domain knowledge can elevate the quality of the evaluation significantly as important aspects and details of the data can be weighted differently. In the manufacturing domain used in the evaluation, specific peaks and outliers can offer very useful insights and are important to be preserved when data is generated while reducing noise can also be an asset. Utilizing the knowledge and experience of a domain expert is crucial in the initial phase of parameter optimization to ensure the generated data reflects the important aspects of the data well. There is also the opportunity to allow labeling by the domain expert and investigate which of the before mentioned evaluation techniques are most closely correlated to the labeling of the domain expert. It is important to evaluate when and to which extend the domain expert should be utilized to find the balance between cost and quality which can be found with initial labeling of generated data and automated weighting of the other evaluation metrics which are subsequently used for automatic evaluation.

6.2.5 Comparison of the Evaluation Techniques

The visual evaluation performed by a domain expert can be seen as the ground truth for the evaluation and this section dives into how the different evaluation techniques relate to each other and how they relate to the ground truth. By making the initial investment into labeling the generated data, it is then possible to calculate weights for the different evaluation techniques and create a loss function which can then be minimized in model training. By using a selection of trainable parameters and this loss function it is possible to optimize the trained models further.
Looking at Figure 6.4 reveals some interesting insights. Most notably is the fact that kendall has the highest average correlation, be it positive or negative, with all other techniques, which makes it the prime candidate for a scenario where only a single technique is used. Another important insight is that dynamic time warping and the euclidean distance have a relatively weak correlation which shows that shifts and or speed changes must be present in the data, which we know to be true. This also raises the question if it might be of interest to build an indicator that determines how much distortion is present in the data, by comparing the euclidean distance with DTW. The fact that pearson, spearman and kendall all come to very similar conclusions is expected as their definitions are also similar mostly differing in whether the data is used directly or if ranks are established.
Based on these results, the decision was made to use kendall rank correlation coefficient as the default evaluation technique. The user has the option to incorporate their own evaluation which will then be used as the ground truth and a weighting function is established for all the techniques to reflect the ground truth as well as possible. A distortion indicator is also added to highlight which data samples have a higher than normal distortion which can be of interest for further investigation by the user. This can

help to identify outliers or samples with specific abnormal occurrences, which can be used to group and boost these samples to increase the data set size in a more balanced way which can be vital depending on what the generated data is intended for e.g., in some form of a neural network that would suffer from a bias if the classes are not balanced in case the desired result is classification.

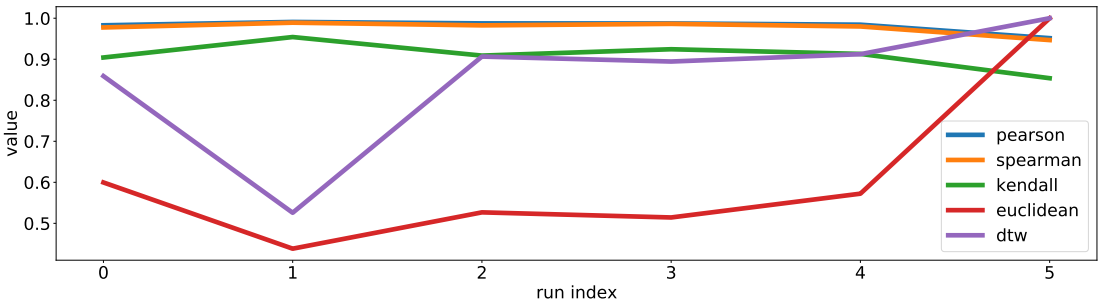


Figure 6.3: Results for the evaluation techniques for six generated data samples of the Y Axis Motor

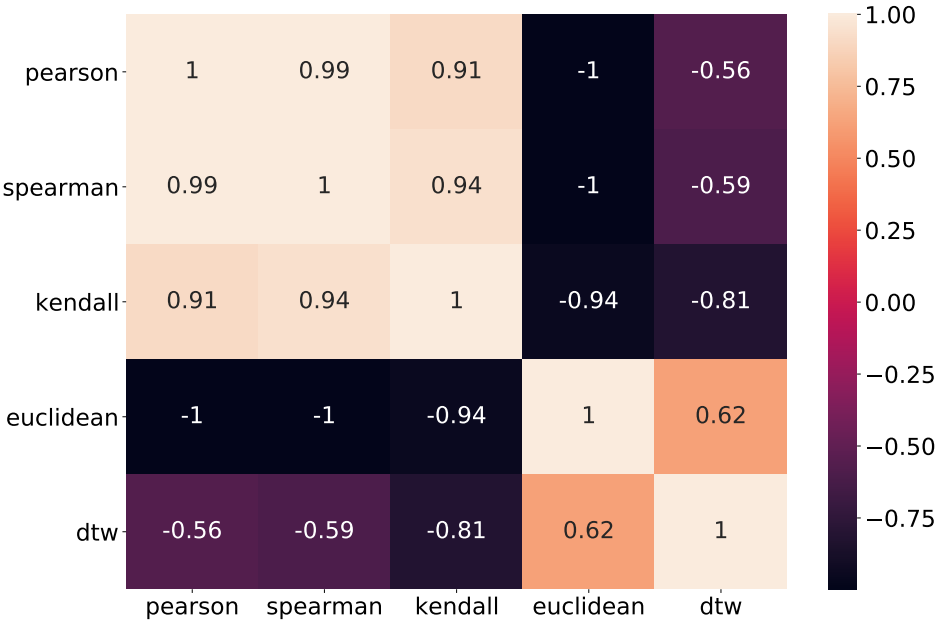


Figure 6.4: Heatmap of the correlation matrix for the evaluation techniques for the Y Axis Motor

6.3 Evaluation of the Results

Figures 6.5, 6.6, 6.7 show sample data for the x and y axis motor load as well as the spindle speed in the turning machine which were chosen because of their different characteristics. The x axis motor produces relatively high noise, specially in the end, while the y axis motor produces large jumps as well as many plateaus. The spindle speed is characterized by a combination of ramp ups and either ramp downs or relatively sudden drops. What make the spindle speed an interesting metric is that in the collected data the variance is very low as the machine is powerful enough to keep the desired speed when machining and thus the speed values are almost not affected by increased friction or other machining related factors. Together, the different metrics vary strongly in their characteristics and variance between samples and provide a diverse challenge to the model training which provides additional meaning to the evaluation.

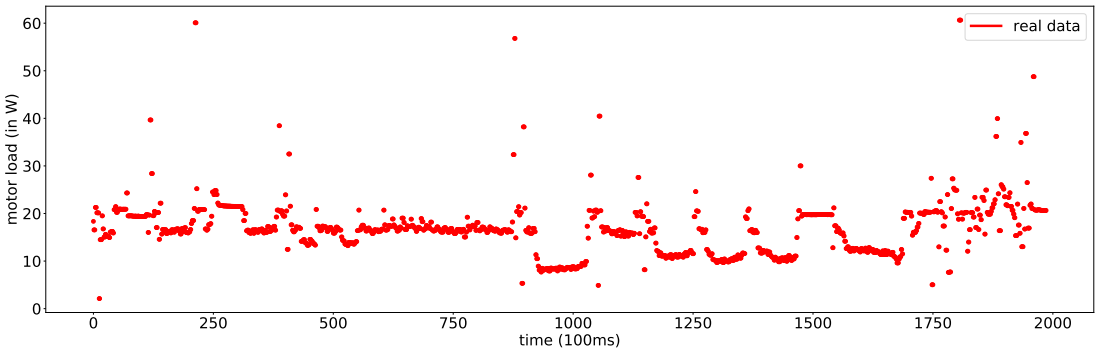


Figure 6.5: Sample data used for evaluation of the x axis motor load in the turning machine

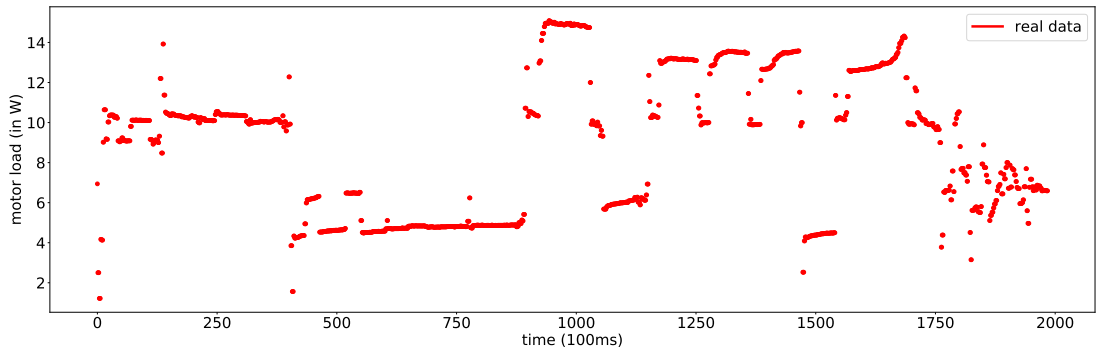


Figure 6.6: Sample data used for evaluation of the y axis motor load in the turning machine

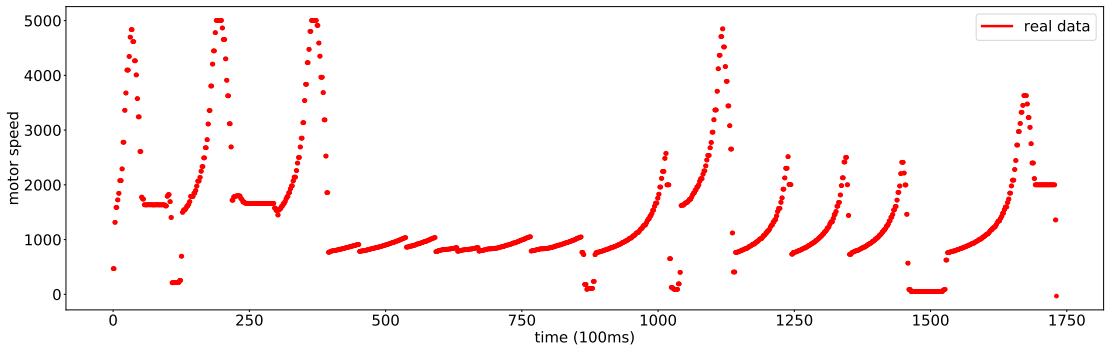


Figure 6.7: Sample data used for evaluation of the spindle speed in the turning machine

6.3.1 Evaluation of long short-term memory

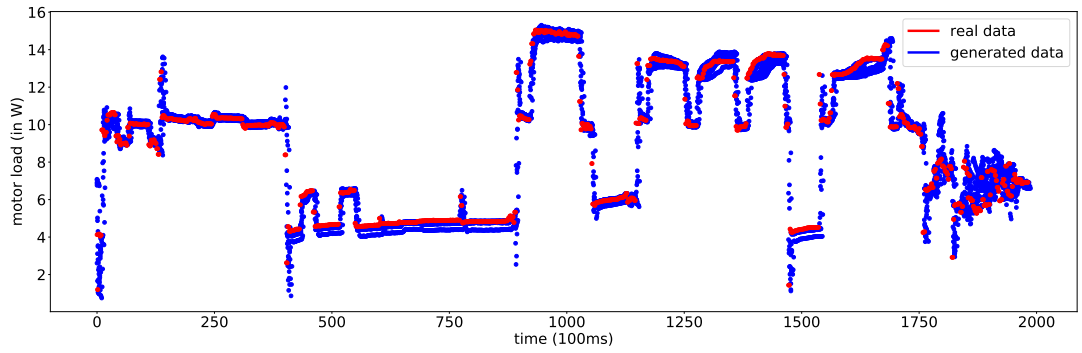


Figure 6.8: The real data is shown in red while blue shows the distribution of 100 generated data samples based on only 6 training samples for the load of the y axis motor in the turning machine

The results are encouraging as Figure 6.8 shows that the generated data samples follow the distribution of the input data whilst introducing variance by using the time series from other samples as input for the prediction. What makes the LSTM such as powerful technique for time series data generation, is the fact that its prediction depends on both what it learned from the training data, as well as on the input data it is presented with. Instead of just introducing variance through introduction of randomness or through combining multiple time series samples into a new altered sample, letting the model predict each time step allows it to handle changes in the process in a much more natural way. Assuming the data generation was merely a clever combination of samples based on some mathematical rules to increase variance in the data set in a somewhat meaningful way, the problem would be that it would adjust to changes in the process very slowly as only a small part of the data would be data from the new process, while the majority would be from the old process. By using an LSTM model, the newly trained model will

be trained with a focus on recent data, which helps it to adapt very quickly to changes in the process and as a result it makes predictions that reflect the new process heavily even if the input time series is from the old process. Depending on the conditioning of the model chosen, the model can have a very high focus on the most recent data and basically forcing every input sample into the new process where the input data just introduces some variance without having a large influence on the distribution of the generated data. It is equally possible to condition the model to adapt more slowly and smoothly to changes by varying the degree of aggregation of the training data. By training a model for every data sample the resulting data generation architecture is highly dynamic which can be an asset and a risk depending on the application and desired behavior. Using a moving window of different sizes for data aggregation with e.g., mean or median, noise in the data can be reduced as well as outliers can be reduced and smoothed out. Conditioning the data before training gives the user fine grain control over how the model will react to changes in the process. The time to generate the time series data is negligible as the bottleneck is the mapping back to the original file format. Additionally, it can also be feasible to train the models at the time of the data generation instead of beforehand as the training time required is rather short. This might make sense if the data is very large or if models at different aggregation levels are desired, meaning both fine grain models and models that were trained on data that was aggregated to a certain degree. Table 6.1 shows the correlation coefficients for generated data based on six input time series and the predictions of an individual model for data of the load of the y motor axis of the machine.

Y Motor	run 0	run 1	run 2	run 3	run 4	run 5	mean	std
pearson	0.983	0.991	0.987	0.987	0.984	0.951	0.981	0.013
spearman	0.977	0.989	0.983	0.986	0.980	0.947	0.977	0.014
kendall	0.904	0.954	0.909	0.924	0.913	0.854	0.910	0.030
mean	0.955	0.978	0.960	0.966	0.959	0.917	0.956	0.019

Table 6.1: Correlations for generated data for the load of the y axis motor based on LSTM model

X Motor	run 0	run 1	run 2	run 3	run 4	run 5	mean	std
pearson	1.0	0.503	0.464	0.438	0.468	0.458	0.555	0.200
spearman	1.0	0.711	0.661	0.640	0.684	0.585	0.714	0.134
kendall	1.0	0.608	0.535	0.541	0.565	0.466	0.619	0.175
mean	1.0	0.607	0.554	0.540	0.573	0.503	0.629	0.170

Table 6.2: Correlations for generated data for the load of the x axis motor based on LSTM model

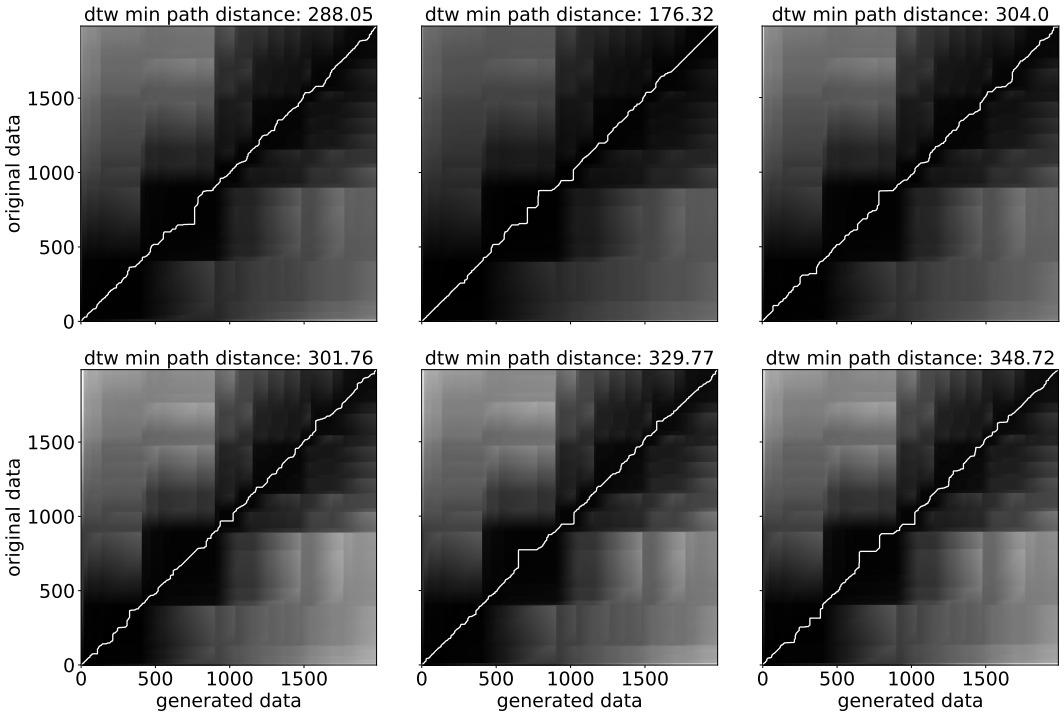


Figure 6.9: The graphs show the generated data from one model and six different time series as input plotted against the original time series the model was trained on and shows how much and where they differ as well the minimum path for dynamic time warping

6.3.2 Evaluation of generative adversarial network

Figure 6.10 and 6.11 show the data generated by the GAN model overlaid by the original data for the load of the y axis motor and spindle speed. The Figures show that the model does a good job approximating the data, it does however approximates the data from left to right as can be seen in Figure 6.12. The training cost is orders of magnitude higher than the LSTM model and the results shown in the Figures required 800000 iterations and took 42 hours on the reference machine. In case a simpler architecture with less hidden layers and nodes per layer is used the model fails to fit the data well. Another drawback is that the complexity required for the model depends on the complexity of the data e.g., polynomial degree which makes it difficult to use a fixed model for all metrics as this results in either exorbitant training time or poor performance for some metrics. The reason it is not possible to utilize early stopping to determine when the model reached a satisfactory score, is that data points further on the temporal axis are not approximated yet and the loss function would still be very high and dependent on the input data which means it is not possible to predict what an acceptable loss value for early stopping should be. A benefit of this approach is that in case not the entire time series is relevant but

Spindle Speed	run 0	run 1	run 2	run 3	run 4	run 5	mean	std
pearson	0.988	0.989	0.985	0.991	0.992	0.972	0.986	0.007
spearman	0.972	0.977	0.973	0.981	0.984	0.944	0.972	0.013
kendall	0.948	0.951	0.932	0.952	0.951	0.899	0.939	0.019
mean	0.969	0.972	0.963	0.975	0.976	0.938	0.966	0.013

Table 6.3: Correlations for generated data for spindle speed based on LSTM model

rather only a given duration from the beginning, than this approach converges faster as it uses the computation to approximate the data over the temporal axis rather than trying to approximate the entire data at once. Generally, the GAN approach is most useful when the training data is highly multivariate and thus requiring a complex model architecture and if also static i.e., non time series data should be utilized in the training. This can become relevant when this accessory data includes causality for patterns in the time series data. In case of the machining scenario, this could be the exact placement of the raw material into the machine that causes an increased or decreased resistance on the motors, or a measurement of the raw material that shows variation in the dimension which causes more or less material that needs to be removed to create the final part which in turn affects the load on the machine. When a model is also trained on this kind of data, and for inference this data is also used, the prediction accuracy can be increased as it holds more causal information. That is one of the reasons why the GENLOG pipeline is designed to utilize different models with minimum manual effort to be able to cope with a wide variety of use cases and data.

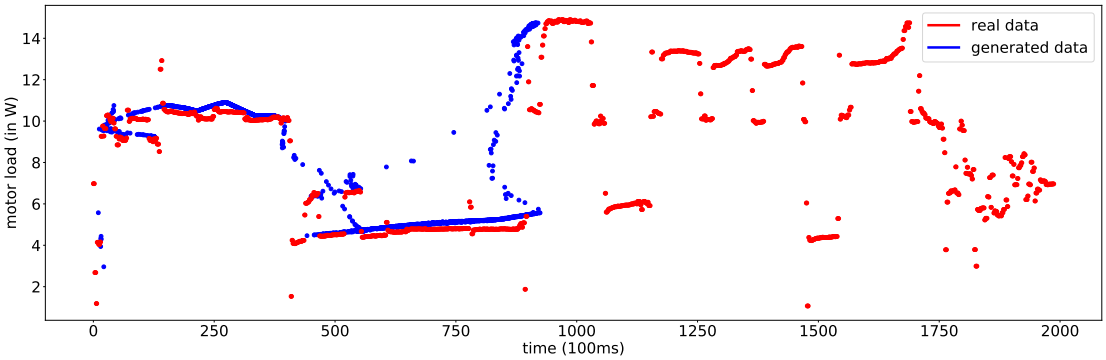


Figure 6.10: Blue shows the generated data and original data in red for the load of the y axis motor in the turning machine

6.3.3 Evaluation of time-series generative adversarial network

Table 6.4 shows an extract of tested parametrizations for TimeGAN on the data used for all the evaluation and Figure 6.14 shows an example result for the y axis motor data which shows that the data was not learned properly. In contrast to the standard

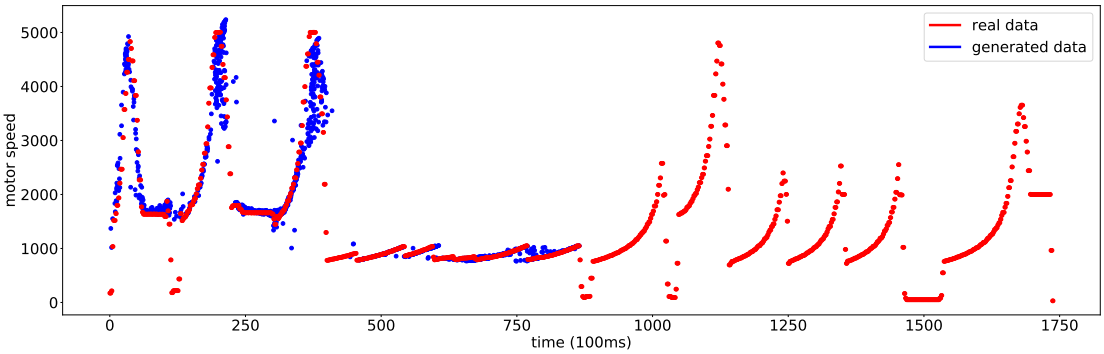


Figure 6.11: Blue shows the generated data and the original data in red for the spindle speed in the turning machine

GAN, TimeGAN uses an recurrent neural network architecture for the generator and discriminator which also explains that the standard GAN learns the data progressively through the temporal domain, meaning that it does not cover all the data in every iteration but rather builds the timeline trough the entire training progress. The training of TimeGAN is very resource intensive as first the embedding network is trained which is the VAE that is responsible for creating an optimized latent space for the generator to work in, as well as conditioning the space to learn temporal relationships. This step is comparatively quick but does share the fixed number of iterations for the training which means it does take a long time in case of a large number of iterations. The next step is the training with supervised loss only. Additionally, introducing a stepwise supervised loss using the original data as supervision, encourages the model to capture the stepwise conditional distributions in the data. This adds more information than just differentiating between real or synthetic samples which is a binary decision and rather learning with a loss function on the real data. This step is also relatively quick but is also executed with the same number of iterations which results in a linear increase in training time with an increase of iterations. The final step is the joint training in which the VAE and the GAN are trained together. The fact that the three steps are executed in series, means that there is no option for early stopping as every step will execute the specified number of iterations. Additionally, as the rate of convergence is dependent on the characteristics of the data, e.g., polynomial degree, finding an appropriate parametrization requires experimentation and differs for every metric. Combined with the generally very high computational cost, this approach does not work well in the desired application in which the pipeline is continuously executed with minimal human intervention.

6.3.4 Evaluation of iterative pipeline execution

As the output of the last step in the pipeline is a log file that represents the log file that was used as the input of the pipeline with the difference that it includes generated time series data instead of the original data, it raises the question of what happens if this generated log file is used again as input in the pipeline and doing this iteratively over and

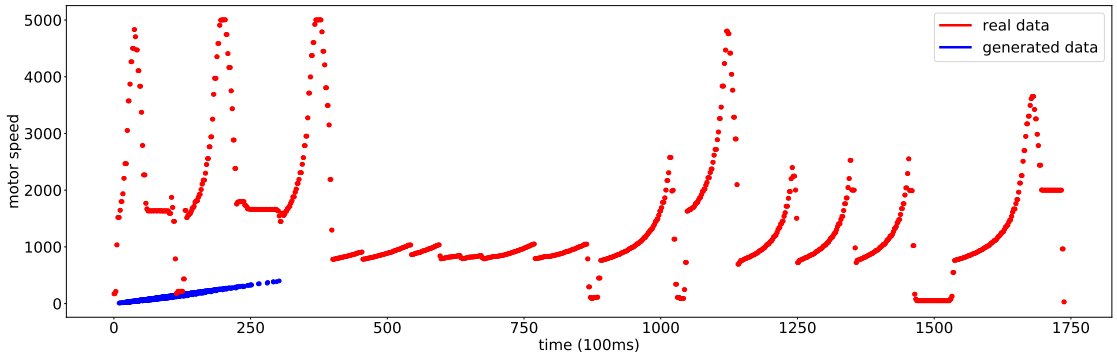


Figure 6.12: Generated data after 2000 iterations from the standard gan which shows an approximation from left to right

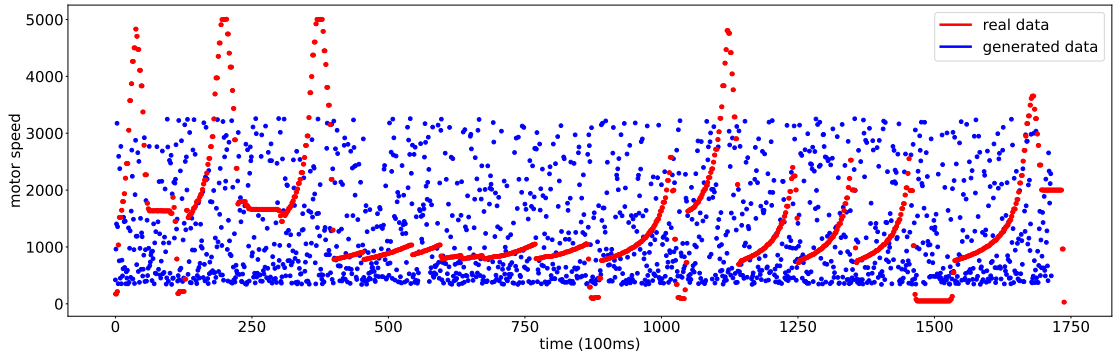


Figure 6.13: Generated data after 2000 iterations from timegan which shows an approximation for every point on the temporal axis

over again. This is both an interesting research question as well as a sanity check to make sure that the generated log files are not corrupted and can be used in the pipeline without any errors. Figures 6.15, 6.16 6.17 show the generated data overlaid by the originating data for the first, second and third iteration of GENLOG. Even though the original data set only contains six produced parts, it can be seen that running the pipeline iteratively adds variation to the data set without unexpected behavior like significant deviation from the input data. The second and third iteration were performed with the first out of six generated logs files, thus if the goal is a major boost in data set size, subsequent iterations can be performed with each generated log file. Extrapolating this example out, yields 36 log files after the initial iteration and up to $(6 + 36)^2 = 1764$ log files after the second iteration. With that it is possible to specify the desired data set size and the pipeline will perform the training by selecting data instances uniformly at random and once all log files from an iteration are used, it will move to the next iteration using the log files from the previous iteration as input and will stop once the set data set size is reached.

TimeGAN	training 0	training 1	training 2	training 3	training 4	training 5
module	lstm/gru	lstm/gru	lstm/gru	lstm/gru	lstm/gru	lstm/gru
hidden_dim	24	512	24	24	256	24
num_layers	3	3	3	3	6	3
iterations	10000	10000	300000	20000	30000	200000
batch_size	128	128	128	1024	128	24

Table 6.4: Tested parameterizations for TimeGAN all with similar unsatisfactory results

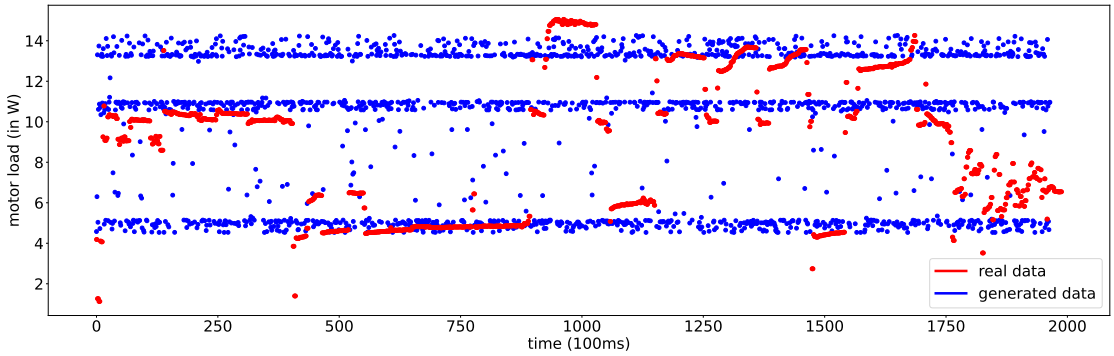


Figure 6.14: Blue shows the generated data after 100000 iterations which was not able to fit well to the original data in red for the load of the y axis motor in the turning machine

6.3.5 Practical implications

Applying the iterative functionality described in Section 6.3.4 over time provides the functionality described in Figure 5.3 in which a fixed data set size is specified and in the beginning the data set mostly consists of generated data and over time the data set consist more and more of real data and reliable generated data as a larger input data set yields an output that better represents the distribution of the data. This makes it a useful tool in practise rather than just an interesting research project as it allows the user to have a appropriately sized data set throughout the production life-cycle with minimal parametrization and interaction required, whilst also continuously improving the quality of the data set. In a practical use case, an output folder can be specified which always contains the desired number of log files which can be used by another process and it gets continuously updated whenever new data instances appear in the input folder. Equally it is possible to create a new interface that can connect an existing process to the generated data set. Examples can be: writing the time series to a database such as influxdb or to provide a REST web service that serves the data. After adding more features and abilities for the user to interact with the data in the applied machining scenario and after adding more evaluation and processing steps for research purposes, the two applications were separated into a production pipeline and a research pipeline, which share most their code and mostly differ in their user interfaces and therefore user interactions as well as

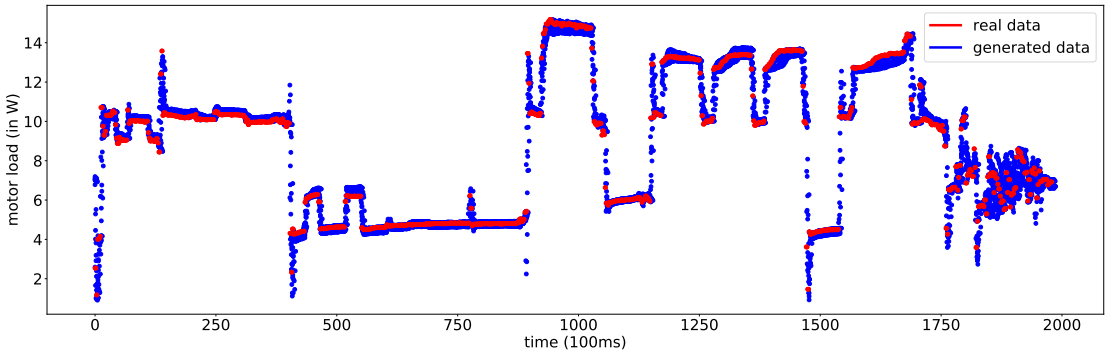


Figure 6.15: The results for the inital iteration of GENLOG based on the real log files

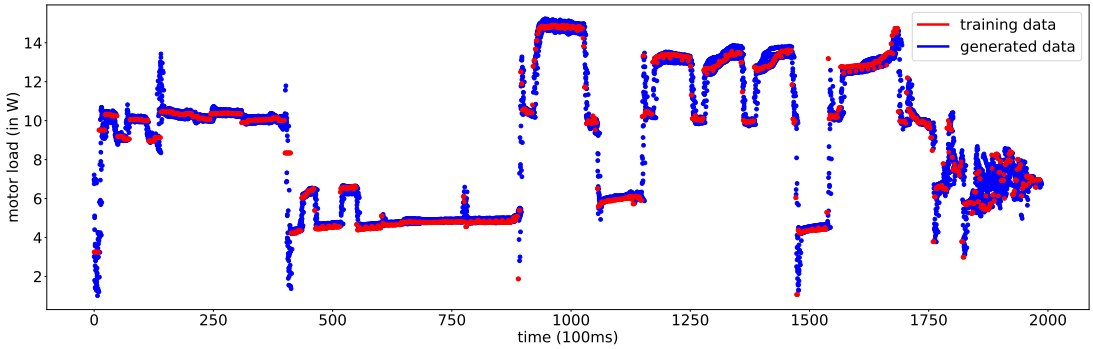


Figure 6.16: The results for the second iteration of GENLOG based on the output of the initial iteration

some processing aspects like using templates for remapping to log files in the research pipeline and using the originating log file of the model that generated the time series in the production pipeline. The main generated assets of the production pipeline are the boosted data set and the key performance indicators like input/output variance and correlation between euclidean distance and dynamic time warping while the research pipeline trains a single model and uses a selected fix set of input data to produce all the aforementioned evaluation metrics and the generated log files as assets.

6.4 Comparison of the results

For TimeGAN in every iteration all data points are estimated which is evident by comparing the 2000th iteration of the standard GAN in Figure 6.12 with TimeGAN in Figure 6.13 where the standard GAN creates a cluster in the beginning of the data and TimeGAN is close to the mean over all data points for every time step. This is potentially more useful as the training could terminate once the threshold based on the evaluation techniques is met as every iteration produces a viable result as it covers

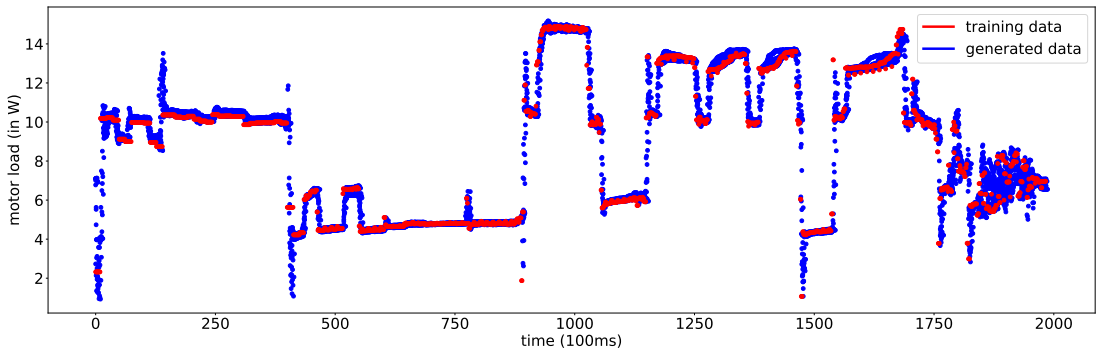


Figure 6.17: The results for the third iteration of GENLOG based on the output of the second iteration

the entire temporal range in contrast to the standard GAN which learns the data more accurately but progresses on the temporal axis. This differentiation could be relevant depending on the use case as it might not be required to learn the entire data sample. Training high performing LSTM models can be achieved in a fraction of the time of training the GAN models as they are inherently designed to work with time series data and can therefore solely learn the temporal patterns in the data instead of being a general learning algorithm that is merely applied to time series data. Working with individual time steps at the architectural level means that model complexity does not need to scale with data complexity to a high degree and it is a single model architecture compared to the multi model approach of TimeGAN that needs to be executed in series. However, this does only apply if the goal is to just generate time series data and use the generated data in analytical tasks, in case the task that data is generated for also takes into consideration other data from the log files, then a more complex model can also generated appropriate none time series data to create a log file that is valid or close to being valid for all its included information. This could be achieved by training additional LSTM models on these event stream data and creating a latent space and architecture that learns not only the temporal relationships within a metric or event information, but rather also learns the relationships, dependencies and correlations between metrics and event information.

LONG SHORT TERM MEMORY

In [1]:
```
from app import extract, resample, train, yaml
file = '1c65003f-2c69-449a-9e8b-7dc8ddda07d4.xes.yaml'
filename = '1c65003f-2c69-449a-9e8b-7dc8ddda07d4'
```

```
/home/demo/Projects/genlog
```

In [2]:
```
extract(file, filename)
```

```
PATH uploads/1c65003f-2c69-449a-9e8b-7dc8ddda07d4/single_runs/
single runs start
single runs end
---------------------------------
```

Out[2]: 'extraction finished'

In [3]:
```
resample(file, filename)
```

```
resample start
write resampled 1c65003f-2c69-449a-9e8b-7dc8ddda07d4_Axis_X_aaLoad.csv
write resampled 1c65003f-2c69-449a-9e8b-7dc8ddda07d4_Axis_Y_aaLoad.csv
resample end
---------------------------------
```

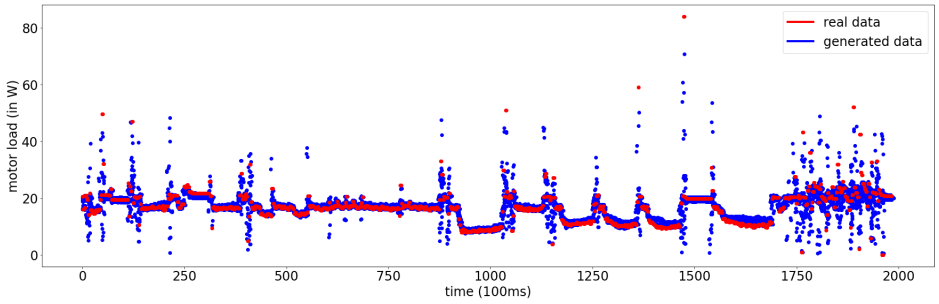
Out[3]: 'resampling finished'

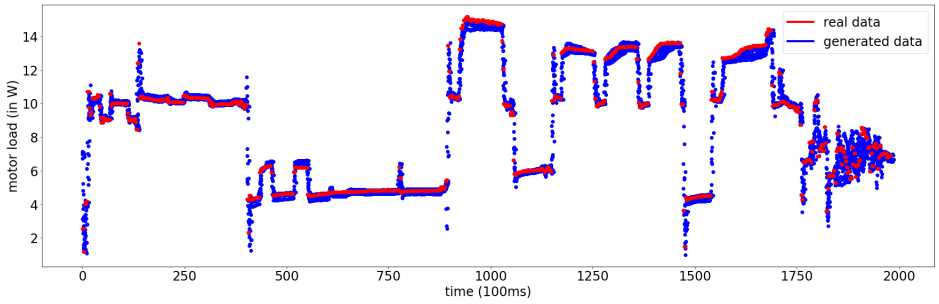
In [4]:
```
train(file, filename)
```

```
lstm training start


lstm training end
---------------------------------
```

Out[4]: 'training finished'





In [5]:
```
yaml(file, filename)
```

```
yaml start


yaml end
---------------------------------
```

Out[5]: 'yaml finished'

Figure 6.18: Evaluation of the results of a run of GENLOG using an LSTM as the model for data generation

7 Discussion of the Results

- **How can the size of a small time series data set be increased with meaningful data in order to use it in machine learning applications?**
  The use of a modular pipeline with defined interfaces allows the approach to be used with different log file types, different data generation architectures and different parametrization. The extensive evaluation metrics show that the generated data follows the distribution of the original data set while adding variance to the data. It also shows that when the input data set does not represent the distribution of the process well, the boosted data set will not change that but only allows the machine learning application that uses the data set to learn the existing patterns better. The fact that the GENLOG pipeline can be introduced in existing workflows and applications that use process log files shows the contribution to the field as it demonstrates its readiness for real world scenarios.

- **Which techniques exist that offer reliable time series data generation without the requirement of extensive domain knowledge and manual effort?**
  Machine learning algorithms such as neural networks have proven their ability to generalize well to unseen data and be powerful classification tools that do not depend on extensive domain knowledge of the user or requires complex setup or parametrization of more traditional techniques. The category of recurrent neural networks offers powerful concepts for data that includes temporal relationships such as time series data. The subcategory of long short-term memories (LSTMs) is a state of the art architecture that offers the ability to learn patterns at different levels of granularity, i.e., recurring patterns on different time scales and for different points in time. This type of model can also be integrated into generative adversarial networks to work with complex time dependent data sets.

- **What are the strengths and weaknesses of the different approaches and can they be automatically optimized?**
  Long short-term memories (LSTMs) provide fast training and inference times which makes it a powerful concept in online applications where the model needs to be able to adapt to changing processes and provide fast data generation capabilities. It is best used in dynamic environments with a low to medium data complexity. For complex highly multivariate data sets, using LSTMs within the context of a GAN can provide the ability to learn complex patterns and to reduce the complexity of the feature space by using a latent space that is optimized for features that are subject to change over time. This ability comes with a dramatic increase in computational

resource requirements and is therefore more suited for offline applications and processes that are not undergoing frequent changes. The combination of grid search to find suitable hyperparameter combinations with the use of the evaluation metrics to determine the quality of the generated data, allows for an automatic optimization of the model architecture and the resulting generated data.

- **What is a suitable set of metrics to assess the quality of the generated data?**
  The generation of evaluation assets is fully integrated in the pipeline and provides the full suite of evaluation metrics whenever the pipeline is executed. This is true for different input data sets, different machine learning models and parameterizations. These assets are used directly in this thesis and are the basis for verifying the validity of the proposed solution. As the evaluation tool is publicly available[1] and open-source[2], it is possible to easily replicate the results and evaluation presented in this thesis. The tool also provides a framework for exploring new machine learning models and compare them to specific solutions (i.e., an architecture with a defined set of hyperparameters) with minimal effort. This contributes to the still relatively young field of machine learning based time series data generation.

[1] https://cpee.org/genlog/
[2] https://github.com/tohe91/GENLOG

8 Conclusion

GENLOG is designed to be used in production environments to require only minimal user intervention to continuously produce data sets of useful size and quality even if the existing data set is very small or of poor quality. It is also designed to be a research tool that allows rapid and highly automated evaluation of different machine learning models and different parametrizations as well as evaluating the performance on data sets with different characteristics. It achieves both goals as it was successfully tested in a real world industrial context and it also provided the majority of the insights and figures for the thesis with minimal manual effort. By making the automatic evaluation part of the pipeline allowed for insight and comparability for every change to the pipeline, machine learning models, their hyperparameters and the data sets, which revealed details that would not have been discovered otherwise. This includes the comparison of runtime between the different models and evaluation techniques as well as correlations in the input and output data. An aspect that did not perform as intended is the class of generative adversarial networks, which were not able to provide an acceptable quality for the computational cost required. The standard GAN lacks the inherent architecture that an recurrent neural network provides to cope with temporal relationships, which rendered it inefficient as it required a very complex design to deal with the complexity of the data and it required more manual effort as the model complexity must match the data. The TimeGAN approach is designed for time series data, but its complex architecture also resulted in inefficient performance on the univariate data in the real world industrial scenario. These approaches for data generation certainly have their place and have the potential to outperform a standard RNN if the input data is highly multivariate, but in the context of continuous automated data generation, as of now, recurrent neural networks provide a much better ratio between quality and computational cost.
Therefore it can be shown that the GENLOG pipeline meets its goal of boosting small data sets with reliable data and that it is feasible to use it in real world scenarios as the computational cost is low and it can work highly automated. It also provides insights into the current state of the art time series data generation models and evaluation techniques. Another way to use a LSTM would be to take data from an event stream to predict the next value at each step. Then a data log would be created for the entire production run and compared to the results of the actual log. A Neural Network would be used as a discriminator to decide if this production run was inline with what it knows about the process or how much it diverges from the expected output. If remapping to the original file type is not required, a near real time prediction is also possible as a prediction takes only a couple of hundred milliseconds. In future work, a goal is to use it for a monitoring system to distinguish between process changes and malfunctions. Overall, the results show that it is feasible to use a pipeline such as GENLOG which does not rely extensively

on pre-configured parameter sets, but instead solely utilizes historic log data, in order to generate new time series for log data.

Bibliography

[20116] *IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams (1849-2016)*. IEEE, 2016.

[Abd07] Hervé Abdi. The kendall rank correlation coefficient. *Encyclopedia of Measurement and Statistics. Sage, Thousand Oaks, CA*, pages 508–510, 2007.

[Ama93] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.

[BCHC09] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[BHR18] Gilles Blanchard, Marc Hoffmann, and Markus Reiß. Optimal adaptation for early stopping in statistical inverse problems. *SIAM/ASA Journal on Uncertainty Quantification*, pages 1043–1075, 2018.

[BKC17] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[BKEI09] Oren Ben-Kiki, Clark Evans, and Brian Ingerson. Yaml ain't markup language (yaml™) version 1.1. *Working Draft 2008-05*, 11, 2009.

[BvdAZP14] R. P. Jagadeesh Chandra Bose, Wil M. P. van der Aalst, Indre Zliobaite, and Mykola Pechenizkiy. Dealing with concept drifts in process mining. *IEEE Trans. Neural Networks Learn. Syst.*, 25(1):154–171, 2014.

[CCH15] Ali Charkhi, Gerda Claeskens, and Bruce E Hansen. Minimum mean squared error model averaging in likelihood models. *Statistica Sinica*, 2015.

[CCK+18] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8789–8797, 2018.

[CMA94] Jerome T Connor, R Douglas Martin, and Les E Atlas. Recurrent neural networks and robust time series prediction. *IEEE transactions on neural networks*, 5(2):240–254, 1994.

[CMS12] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3642–3649. IEEE, 2012.

[CPC+18] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):1–12, 2018.

[DCSF15] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. *arXiv preprint arXiv:1506.05751*, 2015.

[DR81] William Dunsmuir and PM Robinson. Estimation of time series models in the presence of missing data. *Journal of the American Statistical Association*, 76(375):560–568, 1981.

[EFMR19] Matthias Ehrendorfer, Juergen-Albrecht Fassmann, Juergen Mangler, and Stefanie Rinderle-Ma. Conformance checking and classification of manufacturing log data. In *Business Informatics*, pages 569–577, 2019.

[ERF17] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. XES tensorflow - process prediction using the tensorflow deep-learning framework. In *Forum and Doctoral Consortium at International Conference on Advanced Information Systems Engineering*, pages 41–48, 2017.

[FC18] Fanny and Tjeng Wawan Cenggoro. Deep learning for imbalance data classification using class expert generative adversarial network. *Procedia Computer Science*, 135:60–67, 2018.

[FGL+20] Weiguang Fang, Yu Guo, Wenhe Liao, Shaohua Huang, Nengjun Yang, and Jinshan Liu. A parallel gated recurrent units (p-grus) network for the shifting lateness bottleneck prediction in make-to-order production systems. *Computers & Industrial Engineering*, 140, 2020.

[FVA14] Otto Fabius and Joost R Van Amersfoort. Variational recurrent auto-encoders. *arXiv preprint arXiv:1412.6581*, 2014.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[GDG+15] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. In *International Conference on Machine Learning*, pages 1462–1471. PMLR, 2015.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[Gra13] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[Haw04] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[HMRM21] Tobias Herbert, Juergen Mangler, and Stefanie Rinderle-Ma. Generating reliable process event streams and time series data based on neural networks. In Adriano Augusto, Asif Gill, Selmin Nurcan, Iris Reinhartz-Berger, Rainer Schmidt, and Jelena Zdravkovic, editors, *Enterprise, Business-Process and Information Systems Modeling*, pages 81–95, Cham, 2021. Springer International Publishing.

[HN92] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[HS03] Michael Hüsken and Peter Stagge. Recurrent neural networks for time series classification. *Neurocomputing*, 50:223–235, 2003.

[HSS15] Kazuyuki Hara, Daisuke Saito, and Hayaru Shouno. Analysis of function of rectified linear unit used in deep learning. In *International Joint Conference on Neural Networks*, volume 2015, pages 1–8. IEEE, 2015.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[Jan20] Stefan Jansen. Generative adversarial nets for synthetic time series data. https://github.com/stefan-jansen/machine-learning-for-trading/tree/master/21_gans_for_synthetic_time_series, 2020.

[Je20] Christian Janiesch and et al. The internet of things meets business process management: A manifesto. *IEEE Systems, Man, and Cybernetics Magazine*, 6:34–44, 2020.

[JS11] T Jayalakshmi and A Santhakumaran. Statistical normalization and back propagation for classification. *International Journal of Computer Theory and Engineering*, 3(1):1793–8201, 2011.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KMR19] Patrik Koenig, Juergen Mangler, and Stefanie Rinderle-Ma. Compliance monitoring on process event streams from multiple sources. In *International Conference on Process Mining*, pages 113–120, 2019.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[KWLK19] Dahun Kim, Sanghyun Woo, Joon-Young Lee, and In So Kweon. Deep video inpainting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5792–5801, 2019.

[LQH16] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101*, 2016.

[LRC+16] Yunjie Liu, Evan Racah, Joaquin Correa, Amir Khosrowshahi, David Lavers, Kenneth Kunkel, Michael Wehner, William Collins, et al. Application of deep convolutional neural networks for detecting extreme weather in climate datasets. *arXiv preprint arXiv:1605.01156*, 2016.

[LWL+17] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

[Man01] Danilo P Mandic. *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability*. John Wiley & Sons Incorporated, 2001.

[MH16] Xuezhe Ma and Eduard H. Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016.

[MJC+14] Tomas Mikolov, Armand Joulin, Sumit Chopra, Michael Mathieu, and Marc'Aurelio Ranzato. Learning longer memory in recurrent neural networks. *arXiv preprint arXiv:1412.7753*, 2014.

[MPRE19] Jürgen Mangler, Florian Pauker, Stefanie Rinderle-Ma, and Matthias Ehrendorfer. centurio.work - industry 4.0 integration assessment and evolution. In *Industry Forum at BPM*, volume 2428 of *CEUR Workshop Proceedings*, pages 106–117, 2019.

[MSS10] Juergen Mangler, Gerhard Stuermer, and Erich Schikuta. Cloud process execution engine - evaluation of the core concepts. *CoRR*, abs/1003.3330, 2010.

[Mül07] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[MVSA15] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, volume 89, pages 89–94. Presses universitaires de Louvain, 2015.

[MW17] Fabrizio Maria Maggi and Michael Westergaard. Designing software for operational decision support through coloured petri nets. *Enterprise IS*, 11(5):576–596, 2017.

[ODK+19] Tae-Hyun Oh, Tali Dekel, Changil Kim, Inbar Mosseri, William T. Freeman, Michael Rubinstein, and Wojciech Matusik. Speech2face: Learning the face behind a voice. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[PBS17] Santiago Pascual, Antonio Bonafonte, and Joan Serra. Segan: Speech enhancement generative adversarial network. *arXiv preprint arXiv:1703.09452*, 2017.

[Phi18] Michael Phi. towardsdatascience.com: Illustrated guide to lstm's and gru's: A step by step explanation. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21, 2018.

[Pic15] Karol J Piczak. Environmental sound classification with convolutional neural networks. In *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2015.

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[PTRC07] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[PVZ15] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. 2015.

[Roj13] Raúl Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[RW17] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[S+91] Donald F Specht et al. A general regression neural network. *IEEE transactions on neural networks*, 2(6):568–576, 1991.

[SBCN20] Dipanshu Someshwar, Dharmik Bhanushali, Vismay Chaudhari, and Swati Nadkarni. Implementation of virtual assistant with sign language using deep learning and tensorflow. In *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, pages 595–600. IEEE, 2020.

[SKP97] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[SRHM16] Florian Stertz, Stefanie Rinderle-Ma, Tobias Hildebrandt, and Juergen Mangler. Testing processes with service invocation: Advanced logging in CPEE. In *Service-Oriented Computing Workshops*, pages 189–193, 2016.

[SRM20] Florian Stertz, Stefanie Rinderle-Ma, and Juergen Mangler. Analyzing process concept drifts based on sensor event streams during runtime. In *Business Process Management*, pages 202–219, 2020.

[SVI+16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[SWXC16] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. Benchmarking state-of-the-art deep learning software tools. *CoRR*, abs/1608.07249, 2016.

[TFV+20] Romain Tavenard, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Rußwurm, Kushal Kolar, et al. Tslearn, a machine learning toolkit for time series data. *J. Mach. Learn. Res.*, 21(118):1–6, 2020.

[TQL15] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1422–1432, 2015.

[vdA16] Wil M. P. van der Aalst. *Process Mining – Data Science in Action, Second Edition*. Springer, 2016.

[vdAe11] Wil M. P. van der Aalst and et al. Process mining manifesto. In *Business Process Management Workshops*, pages 169–194, 2011.

[vdARV+10] W.M.P. van der Aalst, V. Rubin, H. Verbeek, B. Dongen, E. Kindler, and C. Günther. Process mining: a two-step approach to balance between

underfitting and overfitting. *Software & Systems Modeling*, 9(1):87–111, 2010.

[vDvdA05] Boudewijn F van Dongen and W.M.P.MP van der Aalst. A meta model for process mining data. *EMOI-INTEROP*, 160:30, 2005.

[WGD+17] Kunfeng Wang, Chao Gou, Yanjie Duan, Yilun Lin, Xinhu Zheng, and Fei-Yue Wang. Generative adversarial networks: introduction and outlook. *IEEE/CAA Journal of Automatica Sinica*, 4(4):588–598, 2017.

[WIJK17] Bichen Wu, Forrest Iandola, Peter H Jin, and Kurt Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 129–137, 2017.

[WYW+18] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, September 2018.

[YLY+18] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Generative image inpainting with contextual attention. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5505–5514, 2018.

[Zac17] Giancarlo Zaccone. *Deep Learning with TensorFlow*. Packt Publishing, Birmingham, 2017.

[Zar05] Jerrold H Zar. Spearman rank correlation. *Encyclopedia of biostatistics*, 7, 2005.

[Zhu17] Nick Zhu. *Data Visualization with D3 4.x Cookbook – Second Edition*. Packt Publishing, Birmingham, 2017.

[ZSLL15] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. A c-lstm neural network for text classification. *CoRR*, abs/1511.08630, 2015.

[ZX00] Jia-Shu Zhang and Xian-Ci Xiao. Predicting chaotic time series using recurrent neural network. *Chinese Physics Letters*, 17(2):88, 2000.

[ZXL+17] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 5907–5915, 2017.

[ZYZ+17] Xu-Yao Zhang, Fei Yin, Yan-Ming Zhang, Cheng-Lin Liu, and Yoshua Bengio. Drawing and recognizing chinese characters with recurrent neural network. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):849–862, 2017.

Acronyms

CNC computer numerical control. 1, 7, 28

CNN convolutional neural network. 9

DTW dynamic time warping. viii, xi, 2, 17, 41, 42, 44, 54

GAN generative adversarial network. viii, xv, 11, 12, 19, 21–25, 27, 35, 39, 49–51, 54, 55, 57, 59

GENLOG generation of log files. xi, xii, 13–20, 28, 31–34, 39, 41, 50, 52, 54–57, 59

GRU gated recurrent unit. 10, 26

kendall kendall rank correlation coefficient. 43, 44

LSTM long short-term memory. vii, viii, xi, xii, xv, 10, 11, 15, 16, 19, 21–23, 26, 27, 34, 35, 47, 49, 55, 56, 59

LSTMs long short-term memories. 26, 57

NLP natural language processing. 10

NN neural network. 3, 9, 11, 15, 57

pearson pearson correlation coefficient. 43, 44

ReLU rectified linear unit. 23

RNN recurrent neural network. iii, v, 10, 11, 26, 34, 36, 51, 57, 59

spearman spearman rank correlation coefficient. 43, 44

TimeGAN time-series generative adversarial network. vii–ix, xi, xiii, 22, 25, 26, 35, 39, 50, 51, 53–55, 59

VAE variational autoencoder. 11, 12, 25, 51

Glossary

Adam optimizer The Adam Optimizer is an optimization algorithm for stochastic gradient descent used in training deep neural networks. 23, 27

backpropagation Backpropagation, or "backward propagation of errors" is an algorithm that uses gradient descent on the loss function in regard to the weights of the model. 9, 23

clustering algorithm Clustering algorithms take unlabeled input data and perform groupings on the data to separate the data into clusters which can then be labeled by the user to created labeled data. Using unlabeled data as input makes them a subset of unsupervised learning compared to supervised learning which includes labels as the ground truth. 33

cross validation In the training of a machine learning model, cross validation is performed by splitting the data in a training and a test set and then iteratively choosing different test sets within the data to avoid overfitting in case the one test set is very biased. 22, 35

deep neural network Deep neural networks are neural networks with multiple hidden layers which allows them to be universal classifiers compared to a linear perceptron which is not. 9–11, 23, 72

gradient descent Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. 9, 21, 27

grid search In the training of a machine learning model, grid search is used to exhaustively try different parameter combinations and computing the results e.g., loss, precision, recall. Examples for parameters are the number of hidden layers, the size of the hidden layers, loss functions, number of iterations and optimizers. 22, 35, 58

hidden layer Hidden Layers in neural networks are located after the input layer and before the output layer and perform nonlinear transformations on the data by determining the weights based on their activation functions. 9

linear perceptron A Linear Perceptron is an algorithm that uses a predictor function to determine a set of weights with the input vector to produce a binary classification for data that can be linearly separated. 9

machine learning Machine learning focuses on data and algorithms that do not require explicit programming and instead are designed to learn on their own. This includes supervised, unsupervised, semi-supervised and reinforcement learning. iii, xi, 1–7, 13–19, 22, 36, 37, 57–59

mean squared error The mean squared error is the average squared difference between the estimated values and the actual value. 23, 27

one-hot encoding A one hot encoding transforms numeric categorical data e.g., 1=cat, 2=dog, into a binary representation in which all classes are 0 and only the current class is 1. This eliminates the bias that class 2 is closer to class 1 than to class 6. 9

overfitting Overfitting describes the case where a machine learning model performs well on the data it was trained on, but fails to generalize well to unseen data as it learned a bias in the training data. 9

reliable data In this context, reliable means that the data set is of sufficient size and quality that it can be used in machine learning applications. iii, 2, 16, 27

REST Representational state transfer (REST) is a stateless protocol to provide web resources such as HTML, XML and JSON and includes the HTTP methods GET, POST, PUT, and DELETE.. 53

sigmoid cross entropy Sigmoid cross entropy computes cross entropy loss for pre-sigmoid activations. 26

TensorFlow TensorFlow is an open-source library for machine learning with a focus on the accelerated training of deep neural networks utilizing graphic cards and tensor processing units. 25

transfer learning Transfer learning is a powerful concept within deep learning. A initial model is trained, usually with a very large data set, and subsequently when a model should be trained on similar data, it can be used as a starting point which reduces training time significantly. An example is an image classification model trained on thousands of classes with millions of pictures and the subsequent model can be trained to differentiate between unseen classes while benefiting from the already learned data from the initial vast data set. 10

underfitting Underfitting describes the case where a machine learning model does not perform well on the data it was trained on. This is can happen when the chosen architecture does not cover the complexity of the data or the parameters were not chosen well causing a slow convergence, e.g., a small learning rate. 9