On the Usage of Deep Learning for Modelling Energy Consumption in Simulation Models

Benjamin Woerrlein, Steffen Strassburger

Group for Information Technology in Production and Logistics, Ilmenau University of Technology, P.O. Box 100 565, Ilmenau 98693, Germany; *{benjamin.woerrlein; steffen.strassburger}@tu-ilmenau.de*

SNE 30(4), 2020, 165-174, DOI: 10.11128/sne.30.tn.10536
Received: July 15, 2020 (Selected ASIM SPL 2019 Postconf. Publ.); Revised: Oct. 7, 2020; Accepted: Oct. 20, 2020
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna
ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. With the increasing availability of data, the desire to interpret that data and use it for behavioral predictions arises. Traditionally, simulation has used data about the real system for input data analysis or within data-driven model generation. Automatically extracting behavioral descriptions from the data and representing it in a simulation model is a challenge for these approaches. Machine learning on the other hand has proven successful in extracting knowledge from large data sets and transforming it into more useful representations. Combining simulation approaches with methods from machine learning seems, therefore, promising. Representing some aspects of a real system by a traditional simulation model and others by a model generated from machine learning, a hybrid system model (HSM) is generated. This paper discusses such HSMs and suggests a specific HSM incorporating a deep learning method for predicting the power consumption of machining jobs.

Introduction

For a computer simulation of a real system it is indispensable to create a model of this system. System models are generally abstractions of the real-world system under observation and will focus on the most relevant parts, or attributes thereof that are of interest to the model designer. In traditional modelling approaches, the modeler is bound and potentially limited by the chosen modelling paradigm.

Machine learning (ML), in contrast to simulation, is a set of algorithms that provide an efficient way to aggregate rather big data sets and to find patterns within that data [1]. Those patterns can then further be used to describe the mechanism of the system of interest that emitted the initial data points. Applications of ML are not limited to sets of static data, as the most prominent picture-classification tasks, but can also be applied to dynamic data sets such as time series data. This duality results in ML methods being a promising match for hybrid systems modelling since a chosen simulation methodology can be complemented by an ML method with a different methodology and vice versa.

While ML methods are no simulation technique by definition, they can be used to design a data-driven model as a constituent part of a hybrid systems model (HSM) [2]. In this paper, we propose such an HSM that combines discrete event simulation (DES) with Sequence2Sequence (Seq2Seq) neural networks. This newly proposed HSM focuses on the realistic depiction of electrical power consumption of a job in a manufacturing cell that contains a waiting room and a machine tool.

The investigation of energy efficiency issues within simulation has become a widespread research approach. Existing studies are often based on the consideration of the power consumption of resources (machines, furnaces, etc.) by means of metrologically recorded operating conditions, which are regarded as constant over a defined period of time [3].

The power consumption of resources averaged over a period is then assigned to a resource state and can be mapped and analysed status-based with discrete event simulation approaches. It is questionable, for which application cases these quasi-static operating states provide enough closeness to reality! For the determination and smoothing of load peaks of many resources such an approach does not offer sufficient proximity to reality.

A solution for this is presented in [4]. It is based on the basic idea of combined simulation, which is also proposed, for example, in [5]. While the production and logistics part of the model is represented classically with discrete event simulation, the system dynamics approach is applied for electricity usage in [4]. This enables the time series of the measured power consumption to be reproduced in high resolution in the simulation, offering the advantage of a high-resolution overall picture of a production line's power consumption.

However, the disadvantage of this approach is that only the power consumption of measured jobs can be reproduced. Power consumption of upcoming differing job types cannot be predicted. Furthermore, the approach described in [4] does not depict cause-effect relationships between control parameters (e.g. half feed, slower heating phase, etc.) and the anticipated power usage.

Within the scope of this paper, we will therefore examine whether there are alternative possibilities for high-resolution forecasting of electricity usage that can overcome the disadvantages mentioned above. The focus of the investigation is on the field of ML learning. Particularly, a promising method is proposed based on deep learning.

The aim of the proposed method is to be able to predict time series for the power usage of differing jobs by means of appropriately trained artificial neural networks (ANN). The basic idea is to train an ANN with relevant control information (here: numerical control codes of the production jobs of a machine tool and machine states) and the high-resolution time series of power consumption measured for these jobs.

Then, in perspective, the ANN can forecast a time series of the expected power consumption for any job, possibly even a job with deviating numerical control codes. Then, these time series could be used in hybrid simulation models of the entire production system.

This paper presents a concept for the outlined procedure as well as a prototypical implementation and validation. The paper is structured as follows: Section 1 discusses related work concerning the combination of simulation and ML and introduces the specific ML approach used. In particular, the necessity and basic idea of a deep learning method, which can map asynchronous sequences of different lengths to each other, are presented. This approach was first mentioned in [6], but led to non-conclusive results.

Building on this, Section 2 proposes a concept and prototypical implementation for the overall architecture with its input and output sequences. Section 3 discusses the makeup and necessary preprocessing steps of the data that are required to lead to conclusive results of the model. A brief description of the application case is given in Section 4. A prototypical application of the concept is demonstrated in Section 5. A critical review of the results and a discussion of future work is given in Section 6.

1 Related Work

1.1 Combining Machine Learning and Simulation

The need for data-driven decision making in a dynamic environment results in a need for methods that allow simulation models to adapt over time by learning [7]. Classical simulation approaches, such as discrete event simulation, have traditionally used data about the real system. This was either done manually within the modelling process, e.g., in the context of input data analysis for modelling stochastic influences by fitting theoretical distributions to the real observations, or semi-automatically within data-driven model generation approaches for depicting structural aspects of the model [8]. Automatically extracting behavioral descriptions from the data and representing it in a simulation model can be considered a weak point of automatic simulation modelling approaches [9].

Previous work focused, therefore, on combining ML with traditional simulation modelling for mitigating this weakness.

The papers by Bergmann et al. [10, 11] present an approach for using trained artificial neural networks. These networks can be called from material flow simulation models to obtain a decision on which control strategy to apply within the simulation, depending on certain input parameters modelled in the simulation project.

Another example is given by Rabe et al. [12], where Reinforcement Learning was used alongside a simulation-based Decision Support System for logistics networks. Here, the actions of an agent were modelled through ML, to identify and select principles on which decision-making policies should be carried out by the agent.

In [13] a set of machine learning classification techniques is proposed as a method to generate metamodels for the simulation of sawmilling processes. Here, data-driven models of the sawing process are generated and used to determine what sets of lumber are derived from breaking down the logs in a sawing mill.

Within patients care pathway design for hip fracture, ML was used to identify clusters of patients, and their underlying characteristics to use that insight in the development stage of a simulation model [14]. Unsupervised machine learning was used to cluster a set of patients into subgroups that relate in common characteristics. Once groups of patients at risk being treated for fractured hips were discovered, that information was considered in the simulation model to increase the efficiency of the overall healthcare process through optimized coordination of care resources.

Finally, ML is a key constituent in the modelling of a digital twin, as it is stipulated for symbiotic simulation approaches, and further be referred to as the result of HSM in [15]. Here, ML enables a digital twin that is a virtual representation of a physical system, as it allows the systems simulation model under observation to adapt primarily according to the behaviour of variables controlled by the physical system in question, and not the intentions of the shareholder of the model. Further such hybrid simulation-ML environments can be used to predict the changes in state variables of a system, as ML methods can be trained on past changes in the same system.

These examples have in common that they allow the representation of certain isolated decisions by an ML model and to include that decision within the simulation.

A different – widely uninvestigated – area is the inclusion of entire time-series data delivered from an ML model into a simulation model. This new approach contrasts with classical time series data analytics and prediction in simulation modelling, which have been discussed extensively (e.g., in [16], where time series data were used to generate wait time predictions).

To motivate the potential necessity for machine-learning-based time series predictions, let us consider one of the basic characteristics of discrete event simulation approaches: State changes can only occur at specific event time stamps. Anything that would happen in the real system between two events cannot be depicted. However, for some activities, i.e., the time span between two events, it may be necessary or desirable to describe a progression of a state variable belonging to the activity (e.g., the progression of power consumption during processing). To depict this, a modelling paradigm outside DES would need to be used in combination with DES. If the time series can be described analytically, some form of combined (i.e., continuous and discrete) simulation could be used. Often large amounts of data cannot be used to derive an analytical model, giving rise to the use of ML.

However, large quantities of data are not seen as a liability, but a prerequisite in a machine learning methodology. ML methods, as shown above, represent an effective way to aggregate data at particular steps of a modelling and simulation study, and their further use within a generative aspect will be discussed here.

1.2 Recurrent Neural Nets and Encoder-Decoder Architectures

Artificial neural nets (ANN) are used to identify patterns in complex data structures. For this purpose, embedding layers of a neural network embed the data under observation and guide them through the hidden layers of an ANN. Hidden layers consist of hidden units, the actual neurons. These neurons are self-parameterizing units. The more hidden layers an ANN contains, the higher the degree of abstraction of the recorded information can be. If an ANN has more than one hidden layer, it can link abstractions gained in one layer to another layer, thus creating a more complex abstraction with each added layer. This deep staggering of neural layers is commonly referred to as Deep Learning [1].

If patterns change over time, this temporal sequence of patterns is understood as a sequence. For an ANN to be able to process temporal patterns, recurrent connections must be present in the network topology that enable feedback of abstract knowledge [17, 18]. Such feedback or recurrent neural networks (RNN) are particularly suitable for data that are presented in sequential form [1]. Accordingly, a neural cell must be provided, which on one hand retains its own state and can pass it on, and on the other hand has access to successor states and can classify them. The requirements for such a neural cell with memory are fulfilled by neural Long Short Term Memory (LSTM) cells [19], and their simplification Gated Recurrent Units (GRU) [20].

If the inputs and outputs of a deep learning method are sequences, one speaks of Sequence to Sequence (Seq2Seq) architectures. Here, one embedding layer of an RNN encodes the input sequence. If the input sequence is encoded into a specific neural layer, one speaks of an encoder. If a sequence is generated from of a neural layer, this part of a network topology is called a decoder [1].

The recurrent Encoder-Decoder model (RNN-ED) as described in Figure 1 encodes a sequence X_T of T values into a summary vector C that is then decoded into a sequence $Y_{T'}$ of T' values. The encoder and decoder are conjoined by the fixed sized vector C [20].

If the task of a Seq2Seq model is to map asynchronous sequences, i.e., such of different lengths, to one another, such structures are generally referred to as Encoder-Decoder networks [1]. If sequences of different lengths and different attributes are to be mapped to each other, they must be extended by an additional description, a *context* (cf. context *C* Figure 1). The context is an intermediate hidden layer between the hidden layers of the encoder and decoder [20].

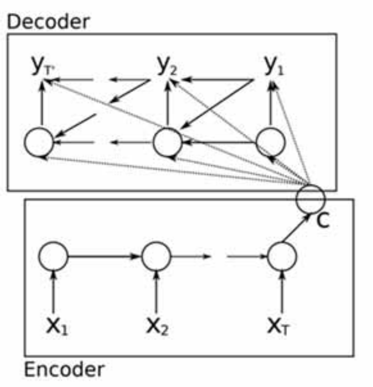

Figure 1: The Encoder-Decoder model as proposed by [20]. Note that $\boldsymbol{T}' \neq \boldsymbol{T}$.

Once all the values of X_i have been processed, the last hidden state is encoded into the summary vector C. The decoder now has two inputs C and $Y_{T'} = (y_1, \ldots, y_{T'})$ and learns the conditional distribution between them by updating its hidden state whilst reading in the values of $Y_{T'}$ and C, accordingly.

Here, the hidden state is linked to a *Softmax*-layer holding the unique tokens found in the training set $\mathbb{Y}$. Once training is finished the decoder can be initialized by any sequence X_i that can mapped to C, and generates a sequence $\widehat{Y_{T'}}$ therefrom [20, 21].

As the model learns to generate the next token y_{t+1} according to the previous token y_t and C, a stop condition needs to be added to keep the decoder from infinitely generating new tokens.

This is commonly done by placing a unique *end-of-sequence* (EOS) token at the end of the sequences Y_i in the training set $\mathbb{Y}$. Then ,once the trained decoder generates an EOS token, the sampling of new tokens is terminated [20, 21].

Further explanations of the encoder decoder used here can be found in [1, 20, 21].

2 Concept and Prototypical Implementation

For conceptual verification it is proposed to use such input and output sequences that are belonging to the same temporal-spatial entity. As a characteristic of a temporal-spatial entity, an activity is assumed that takes place at the same place and at the same time. For this purpose, the technological process of machining a job on a machine tool was identified.

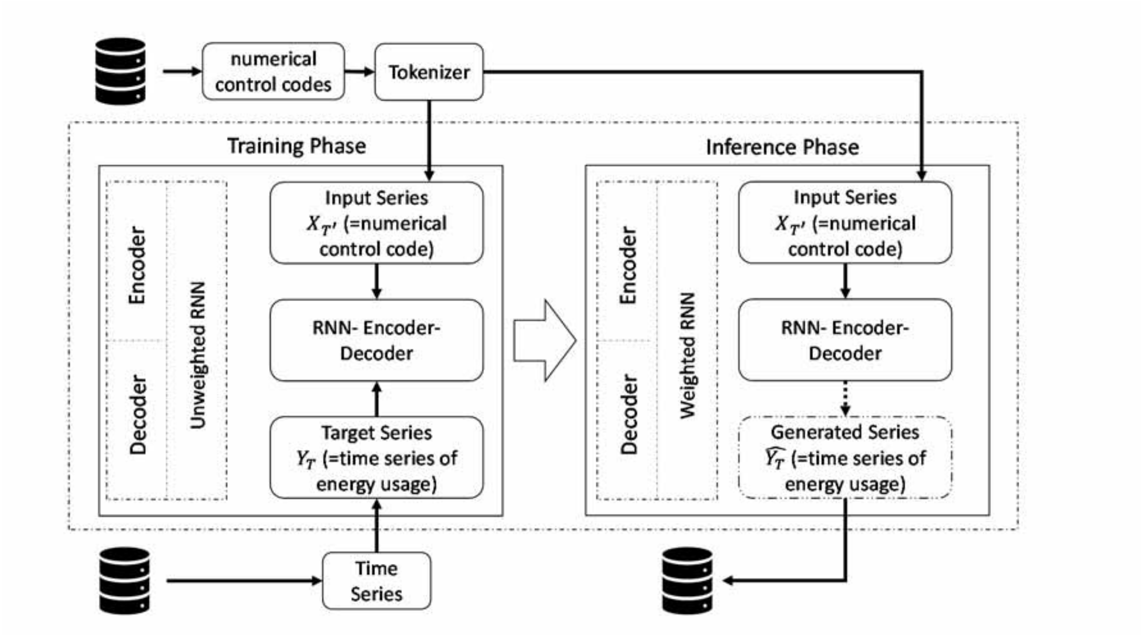
numerical control codes
Tokenizer
Training Phase
Inference Phase
Encoder
Decoder
Unweighted RNN
Weighted RNN
Input Series $X_{T'}$ (=numerical control code)
RNN- Encoder-Decoder
Target Series Y_T (=time series of energy usage)
Generated Series $\widehat{Y_T}$ (=time series of energy usage)
Time Series

Figure 2: Implementation and components of an RNN Encoder-Decoder-Architecture for asynchronous and asymbolic series.

2.1 Predicting Energy Consumption through Seq2Seq

Figure 2 shows the overall concept of the proposed method. In the training phase, an unweighted RNN, the Seq2Seq-model, is parameterized using the input and target sequences $\{X_i, Y_i\}$.

For the training data, 51 in-field measurements of the active power usage of a machine tool and their corresponding numerical control codes, along with machine states, i.e., *modes*, were taken.

In the training phase, an unweighted RNN, consisting of an RNN-ED, is parameterized using the input and target sequences $\{X_i, Y_i\}$. The task of the inference phase is to provide a meaningful power consumption profile $\widehat{Y_{T'}}$ explicitly quasi-continuously over time (see Figure 2).

2.2 Seq2Seq in Hybrid System Models

Furthermore, the RNN-ED is called within a discrete-event-oriented simulation in accordance with a hybrid simulation methodology (cf. Hybrid Systems Model in [22]). Here, the power consumption for each job within the machining room of a machine tool (see the jobs trajectory in Figure 3) is characterized by the described RNN-ED method.

For this purpose, the weighted RNN-ED is called as a constituent of the timeout function within the simulation model once the machine tool is seized by a job (see Figure 3). Here, the prediction of the time for which a job seizes the machine tool is solely achieved by the assignment of an input sequence X_T to the timeout function of a job's trajectory. A specific time series is subsequently generated for each job that passes through the machining room, as it consequently activates the timeout function.

This happens once the job has blocked the resource of the machine tool it is machined on. The resource remains blocked until the seize time according to the timeout function has been reached. Then, the job is released from the resource's trajectory and it can be seized by the next job.

In a simulation run with this hybrid simulation method, the same time series is generated as it is done with a stand-alone inference of the RNN-ED (see $\widehat{Y_{T'}}$ in Figure 2).

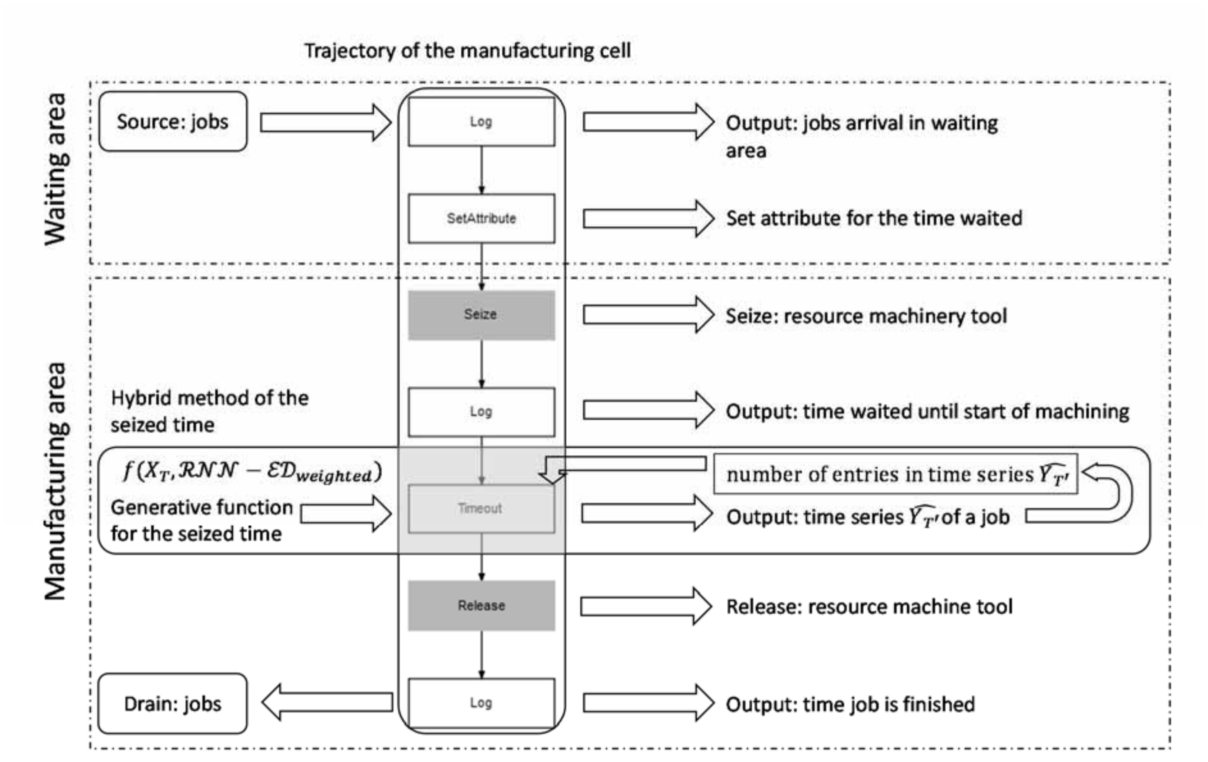
Trajectory of the manufacturing cell
Waiting area
Source: jobs
Log
Output: jobs arrival in waiting area
SetAttribute
Set attribute for the time waited
Manufacturing area
Seize
Seize: resource machinery tool
Hybrid method of the seized time
Log
Output: time waited until start of machining
$f(X_T, \mathcal{RNN} - \mathcal{ED}_{weighted})$
number of entries in time series $\widehat{Y_{T'}}$
Generative function for the seized time
Timeout
Output: time series $\widehat{Y_{T'}}$ of a job
Release
Release: resource machine tool
Drain: jobs
Log
Output: time job is finished

Figure 3: Visualization of a job's trajectory through a manufacturing cell. The constituents of the cell are modeled in a discrete-event-oriented fashion, while the manufacturing area's submodel contains a hybrid method adjacent to the trajectory's timeout function.

3 Preprocessing of Input Data

The machine tool's consumption of energy, and inherently the time it takes to process a job, was initially monitored every time a job is processed on it and then saved as time series data. The numerical control code that controls the machine's action for a job is monitored and saved as well.

In accordance with the findings in [6], both types of sequences need to be preprocessed for the RNN-ED to learn a meaningful context and the connection between them.

3.1 Setup of Input Sequences $\mathbb{X}$

A set of numerical control code and machine states for jobs is used as the input sequence X_T of an RNN-ED (see Figure 1). A numerical control code describes a sequence of necessary technological processes up to the completion of a job and can be understood as a direct description of a sequence of states underlying the process of machining jobs. The numerical control code decisively determines the behavior within the machining room of a machine tool. Furthermore, a job is only considered to be completed once the numerical control code has been completely processed.

The numerical code must first be translated into a sequence of numerical values that retains the structure of the targeted input sequence. This is realized by a so-called *tokenizer* (1). A tokenizer assigns a numeric value to each symbol or set of symbols present in the numerical code, e.g., based on the frequency of the symbol concerned.

$$[\dots G\,00, X0\;Y0\;Z0, \dots] \xRightarrow{Tokenizer} [\dots 1\;2\;3\;4\;5\;\;\dots] \qquad (1)$$

The tokenizer also removes symbols or sets of symbols that are assumed to have a low information content, such as commas, upper or lower case letters, etc. One way to limit the dimensions of the vector space is to dictate the tokenizer a maximum number of symbol sets (i.e., words) that can be mapped to a numerical vector. In our case, a word to vector (Word2Vec) tokenizer was used, which translates all symbols into a vector.

The input sequences are further extended with different modes $\{x_{11}, x_{12}\}$ in which the machine tool can be operated on. Those modes reflect a common work routine in machining a job. The numerical control code runs for the first time $\{roughing = x_{11}\}$ to chip a larger amount of excess material off and give the material its shape.

Afterwards the same numerical control code is run for several times $\{smoothing = x_{12}\}$ to smooth the surface of the now shaped material. Those two modes are reflected in time series of power consumption that are comparable in length but show very different characteristics in their features. The input sequence X_i is described accordingly as:

$$X_i = \{\{x_{11}, x_{12}\}, x_2\}$$

with x_2 being the numerical control code. The sequences of $\mathbb{X}$ are further tokenized to a list of integer values, where any unique word is represented by exactly one integer. This allows for modelling recurring patterns within the numerical control code.

3.2 Setup of a Set of Time Series $\mathbb{Y}$

The basis of values for the quasi-continuous time output series Y_T, is the current power consumption of a job when the numerical control code was processed (see Figure 2). The temporal power consumption gives concrete information about when and how much consumption must be expected before a decision has to be made about the machining of a job.

In contrast to [6], the set of time series $\mathbb{Y}$ has further been discretized. Discretization is the process of portioning continuous values into new discrete groups of values or *bins* that resemble the original values of the data.

This was necessary as the empirical results presented in [6] led to the conclusion that a uniform or long tail distribution, i.e., where the tail tends towards a discrete uniform distribution, P^f of value frequencies prevents the Seq2Seq model from learning a meaningful joint distribution.

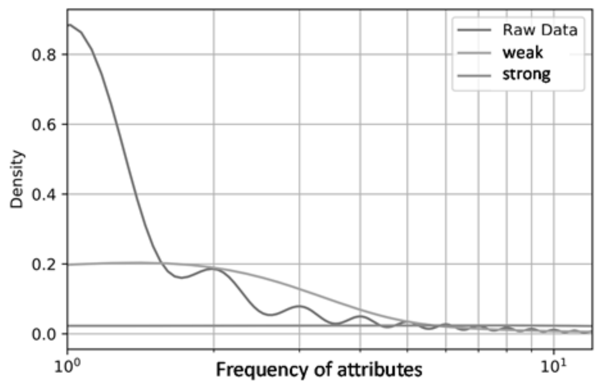

Figure 4: The KDE-plot shows that the raw data contained mostly unique values, while a strong discretization results in a uniform distribution, where the probability of a value belonging to any frequency is the same as for any other frequency. A weak discretization results in a heavy tail distribution of frequencies.

To find the right parameter, as to which degree needs to be discretized, several runs of training with alternative discretization parameters were conducted. The different discretization parameters were applied on the whole data set of time series and, then, classified according to the frequency *f* of the discretized values (Figure 4). For this purpose, the distribution of frequencies P^f was analyzed using a Gaussian kernel density estimator (KDE).

The proposed concept has been tried for all three frequency distributions and results only for the weak discretization in satisfying results.

4 Case Study

Tensorflow was used to call the tokenizer function and to implement the RNN-ED architecture. For the ease of use of *Tensorflow*, the API *Keras* was used. The *Keras* API was used as the interface to *Tensorflow*, because it provides a high level of clarity when presenting network architectures with a higher level of abstraction. The package *rSimmer* [23] was used for the discrete event modelling part.

The time series data $Y_{T'}$ was recorded under field conditions and has the same clocking of Δt – 500 ms – representing quasi-continuous recordings of the active power usage. The input sequences X_i represent the numerical control codes of the same jobs along with the machine states. A tokenizer was used to generate the vectorized input sequence from the input data.

The RNN-ED sequentially embeds the vectorized input sequence X_i and the time series $Y_{T'}$ of power consumption associated with the initial jobs. The trained net and its weightings are then saved. To use the weighted RNN-ED in the inference phase (see Figure 2), the vectorized input sequences X_i of a job are entered into the encoder. This results in the generation of a time series $\widehat{Y_{T'}}$ of the power consumption from a trained RNN-ED for a job.

As this paper focuses on the generation of the values for the time series $\widehat{Y_{T'}}$, instead of its integration, the simulation at runtime will no further be discussed here.

5 Results and Evaluation of the Seq2Seq Method

Metrics to compare the generated time series $\widehat{Y_i}$ and Y_i are the median length $len(\bar{Y}_i)$ and average $sum(\bar{Y}_i)$ of time series as found in the training set. Further, the time series have been visualized and features of characteristic patterns or labels have been added to those visualizations (see Figure 6). Adding features helps to compare the time series $\widehat{Y_i}$ and Y_i more intuitively on a visual level. The result for the raw data, which have not been discretized, aligns with the non-conclusiveness of [6]. The sequences created showed no meaningful course of values and further failed to produce an EOS-token, i.e., the method did not terminate the creation of new numeric values.

The results for the *strong* discretization as shown in Figure 5 are disappointing. An EOS token was created, as well as most other features, yet the generated series can clearly be distinguished from the training data (samples shown in Figure 6) and results in low scores in the metrics, accordingly.

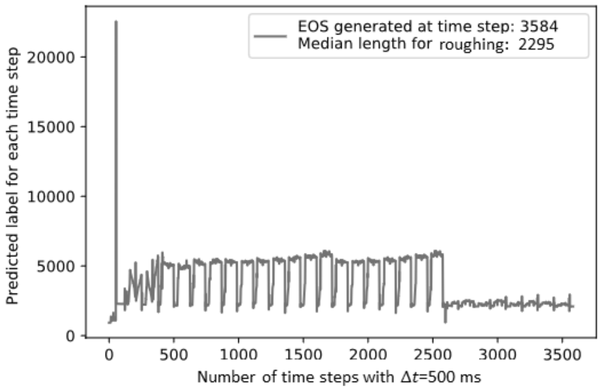

Figure 5: Result for a strong discretization and parameter setting $\{x_{11}, x_2\}$.

On the other hand, the *weakly* discretized time series shows high values in comparison to $len(\bar{Y}_i)$ and $sum(\bar{Y}_i)$:

$$\{x_{11}, x_2\}: \frac{len(\hat{y}=2258)}{len(\bar{y}_i=2295)} = 98.4\,\%; \frac{sum(\hat{y}=6847.7)}{sum(\bar{y}_i=6927.9)} = 98.8\,\%$$

$$\{x_{12}, x_2\}: \frac{len(\hat{y}=2204)}{len(\bar{y}_i=2256)} = 97.7\,\%; \frac{sum(\hat{y}=4843.3)}{sum(\bar{y}_i=4871.8)} = 99.4\,\%$$

On closer inspection of the time series $\widehat{Y_i}$ and Y_i for $\{x_{11}, x_2\}$, as displayed in Figure 6, a striking resemblance can be seen. The sequences displayed from the two sets clearly show the same patterns over the course of labels.

The time series generated accomplishes to mimic the course of labels as shown in the training set with a remarkable precision. It does not only achieve to reproduce an EOS token that matches the length of the time series found in the training set (green dot), a distinct peak-feature (red dot), as well as a string of subsequences (the dots of changing shades of blue), but also to generate them in the right order and dimension.

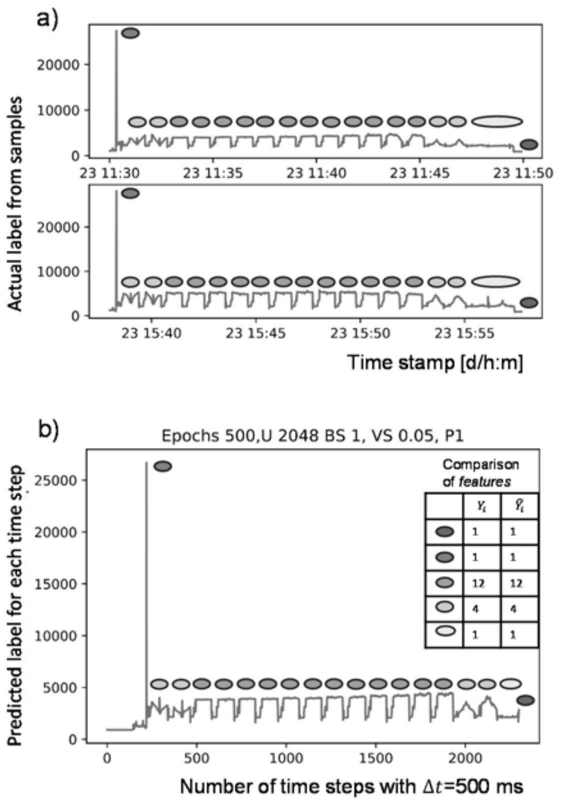

Figure 6: Comparison of (a) samples Y_i drawn from the training set $\mathbb{Y}$ and (b) the generated time series $\widehat{Y_\iota}$ for the weak discretization parameter and the sequence combination $\{x_{11}, x_2\}$. The table shown on the lower-right side compares the count of features against each other.

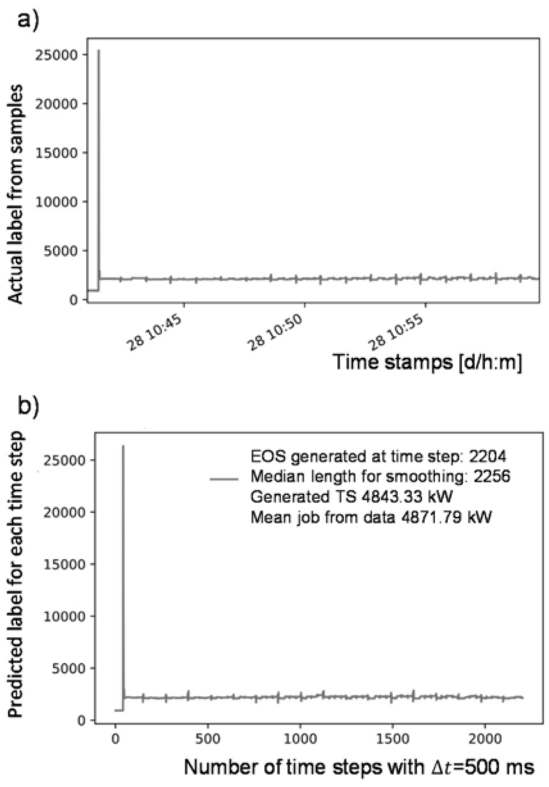

Figure 7: Comparison of (a) samples Y_i drawn from the training set $\mathbb{Y}$ and (b) the generated time series $\widehat{Y_\iota}$ for the weak discretization and the sequence combination $\{x_{12}, x_2\}$. No features were added as the time series holds few characteristics.

The time series $\widehat{Y_\iota}$ and Y_i for $\{x_{12}, x_2\}$, displayed in Figure 7, also clearly show that the Seq2Seq-model succeeded in catching the course of labels within the training set, even though the time series from the training set contained few features that could be learned in the first place.

6 Conclusion and Future Work

The functionality of the described approach was confirmed in the use case by chosen metrics. However, the generated time series still must be critically questioned and validated in further research work.

On the one hand, there is still a lack of evaluation methods for generative models of ML to check the generated time series entries for the meaningfulness of their entries. This is done at the moment by the observation and comparison of the generated time series through an application expert by optical inspection [1] as shown in Figure 6.

For a final evaluation of the methods used, it is advisable to increase the qualitative and quantitative data basis of the Seq2Seq-model. The data set used here is of a small size. Yet the set size is exemplary for real world settings that might change rapidly and in short periods of time.

On the other hand, machine learning algorithms tend to work better given that there is a lot of data to learn from. A framework in which the training set is extended by time series that have been altered to represent a ground truth of the training set of time series could solve that problem. *Dynamic Time Warping* could be used to generate such ground truth time series [24], which could then iteratively be added to the training set until an advantageous learning behavior could be displayed.

Additional end-of-sequence tokens could be used to describe events like machine failures. The *EOS* token used here simply marked the end of a finished job. Yet some jobs are prone to break due to system changes like wear and tear experienced by the tool.

Adding an alternative *EOS*, indicating machine failure, to the training set, along with data for the state of tools etc., might also answer the question whether a job can be executed given the current settings.

The method further allows for generating time series according to factorial combinations not found in the training data. As the decoder is not parametrized directly on the input sequences found in the training set, but on a summary thereof in form of the context vector, factorial combinations of input parameters can be used that are not represented in the initial training set. If the respective input parameters and their distinctive effect on the time series has been modeled, any combination thereof could be used. This would result in factor combinations of high interest to a simulation expert that could not be modeled in a generic simulation setting.

Hence, a suitable evaluation method must be added to the proposed solution, since validation cannot be guided by a (non-existent) ideal time series.

The further development of the ML method described here and its use for hybrid simulation models is currently the subject of ongoing research. Also, if the method is successfully established and validated, a solution could be developed that produces plausible power consumption forecasts for unknown jobs, e.g., based on their numerical control codes. This would have a high practical potential and would also be a breakthrough from a scientific point of view.

The transfer of the basic idea to other forms of control code and time series of other values is also conceivable and a possible subject for further investigations.

References

[1] Goodfellow I, Bengio Y, Courville A. *Deep Learning*. Boston, Massachusetts: MIT Press; 2016.

[2] Mustafee N, Brailsford S, Djanatliev A, Eldabi T, Kunc M, Tolk A. Purpose and benefits of hybrid simulation: Contributing to the convergence of its definition. In: Chan WKV, d'Ambrogio A, Zacharewicz G, Mustafee N, Wainer G, Page E, editors. *Proceedings of the 2017 Winter Simulation Conference*. Piscataway, NJ: IEEE; 2017. 1631–1645. doi: 10.1109/WSC.2017.8247903

[3] Holger H. *Eine Methodik zur modellbasierten Planung und Bewertung der Energieeffizienz in der Produktion.* Dissertation, University of Stuttgart. Stuttgart: Fraunhofer: 2013.

[4] Römer A, Rückbrod M, Strassburger S. Eignung kombinierter Simulation zur Darstellung energetischer Aspekte in der Produktionssimulation. In *ASIM 2018: 24. Symposium Simulationstechnik, 2018 October 2018, Hamburg*. Wien: ARGESIM/ASIM. 73–80.

[5] Peter T, Wenzel S. Simulationsgestützte Planung und Bewertung der Energieeffizienz für Produktionssysteme in der Automobilindustrie. In: Rabe M, Clausen U, editors. *Simulation in Production and Logistics 2015*. Stuttgart, Germany: Fraunhofer Verlag; 2015. 535–544.

[6] Woerrlein B, Bergmann S, Feldkamp N, Straßburger S. Deep-Learning-basierte Prognose von Stromverbrauch für die hybride Simulation. In:Putz M, Schlegel A, editors. *Simulation in Produktion und Logistik 2019*. Auerbach, Germany: Verlag Wissenschaftliche Scripten; 2019. 121–131.

[7] Biller B, Biller SR, Dulgeroglu O, Corlu CG. The role of learning on industrial simulation design and analysis. In: Chan WKV, d'Ambrogio A, Zacharewicz G, Mustafee N, Wainer G, Page E, editors. *Proceedings of the 2017 Winter Simulation Conference*. Piscataway, NJ: IEEE; 2017. 3287–3298. doi: /10.1109/WSC.2017.8248046

[8] Bergmann S, Stelzer S, Wüstemann S, Strassburger S. Model generation in SLX using CMSD and XML stylesheet transformations. In: Laroque C, Himmelspach J, Pasupathy R, Rose O, Uhrmacher AM, editors. *Proceedings of the 2012 Winter Simulation Conference*. Piscataway, NJ: IEEE; 2012. 1–11. doi: 10.1109/WSC.2012.6464981

[9] Bergmann S, Strassburger S. Challenges for the automatic generation of simulation models for production systems. In: *Proceedings of the 2010 Summer Computer Simulation Conference*. San Diego, CA, USA: Society for Computer Simulation International; 2010. 545–549.

[10] Bergmann S, Feldkamp N, Strassburger S. Emulation of control strategies through machine learning in manufacturing simulations. *Journal of Simulation*. 2017; 11(1):38–50.

[11] Bergmann S, Stelzer S, Strassburger S. On the use of artificial neural networks in simulation-based manufacturing control. *Journal of Simulation*. 2014; 8(1): 76–90.

[12] Rabe M, Dross F. A Reinforcement Learning approach for a Decision Support System for logistics networks. In: Yilmaz L, Chan WKV, Moon I, Roeder TMK, Macal C, Rossetti MD, editors. *Proceedings of the 2015 Winter Simulation Conference*. Piscataway, NJ: IEEE; 2015. 2020–2032. doi: 10.1109/WSC.2015.7408317

[13] Morin M, Paradis F, Rolland A, Wery J, Laviolette F. Machine learning-based metamodels for sawing simulation. In: Yilmaz L, Chan WKV, Moon I, Roeder TMK, Macal C, Rossetti MD, editors. *Proceedings of the 2015 Winter Simulation Conference*. Piscataway, NJ: IEEE; 2015. 2160–2171. doi: 10.1109/WSC.2015.7408329

[14] Elbattah M, Molloy O, Zeigler BP. Designing care pathways using simulation modeling and machine learning. In: Rabe M, Juan AA, Mustafee N, Skoogh A, Jain S, Johansson B, editors. *Proceedings of the 2018 Winter Simulation Conference.* Piscataway, NJ: IEEE; 2018. 1452–1463. doi: 10.1109/WSC.2018.8632360

[15] Onggo BS, Mustafee N, Smart A, Juan AA Molloy O. Symbiotic simulation system: Hybrid systems model meets big data analytics. In: Rabe M, Juan AA, Mustafee N, Skoogh A, Jain S, Johansson B, editors. *Proceedings of the 2018 Winter Simulation Conference.* Piscataway, NJ: IEEE; 2018. 1358–1369. doi: 10.1109/WSC.2018.8632407

[16] Mustafee N, Powell JH, Harper JH. RH-RT: A data analytics framework for reducing wait time at emergency departments. In: Rabe M, Juan AA, Mustafee N, Skoogh A, Jain S, Johansson B, editors. *Proceedings of the 2018 Winter Simulation Conference.* Piscataway, NJ: IEEE; 2018. 100–110. doi: 10.1109/WSC.2018.8632378

[17] Zell A. *Simulation neuronaler Netze*. 4th ed.. Munich, Germany: Oldenbourg; 2003.

[18] Brause RW. *Neuronale Netze*. *Eine Einführung in die Neuroinformatik.* 2nd ed. Wiesbaden, Germany: Vieweg + Teubner; 1995.

[19] Hochreiter S, Schmidhuber J. Long short-term memory. *Neural computation* 1997; 9(8): 1735–1780.

[20] Cho K, Merrienboer BV, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics; 2014.

[21] Sutskever I, Vinyals O, Le VQ. Sequence to Sequence Learning with Neural Networks. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems – Volume 2*. Cambridge, MA: MIT Press; 2014.

[22] Mustafee N, Powell JH. From hybrid simulation to hybrid systems modelling. In: Rabe M, Juan AA, Mustafee N, Skoogh A, Jain S, Johansson B, editors. *Proceedings of the 2018 Winter Simulation Conference.* Piscataway, NJ: IEEE; 2018. 1430–1439. doi: 10.1109/WSC.2018.8632528

[23] Lawson B, Leemis LM. An R package for simulation education. In: Chan WKV, d'Ambrogio A, Zacharewicz G, Mustafee N, Wainer G, Page E, editors. *Proceedings of the 2017 Winter Simulation Conference.* Piscataway, NJ: IEEE Press; 2017. 4175–4186.

[24] Petitjean F, Forestier G, Webb GI, Nicholson AE, Chen Y, Keogh E. Dynamic time warping averaging of time series allows faster and more accurate classification. In: *2014 IEEE International Conference on Data Mining,* Shenzhen, 2014, 470-479, DOI: 10.1109/ICDM.2014.27