# Chapter 17
# Unsupervised Learning and Simulation for Complexity Management in Business Operations


**Lukas Hollenstein, Lukas Lichtensteiger, Thilo Stadelmann, Mohammadreza Amirian, Lukas Budde, Jürg Meierhofer, Rudolf M. Füchslin, and Thomas Friedli**

**Abstract** A key resource in data analytics projects is the data to be analyzed. What can be done in the middle of a project if this data is not available as planned? This chapter explores a potential solution based on a use case from the manufacturing industry where the drivers of production complexity (and thus costs) were supposed to be determined by analyzing raw data from the shop floor, with the goal of subsequently recommending measures to simplify production processes and reduce complexity costs.

The unavailability of the data—often a major threat to the anticipated outcome of a project—has been alleviated in this case study by means of simulation and unsupervised machine learning: a physical model of the shop floor produced the necessary lower-level records from high-level descriptions of the facility. Then, neural autoencoders learned a measure of complexity regardless of any human-contributed labels.

In contrast to conventional complexity measures based on business analysis done by consultants, our data-driven methodology measures production complexity in a fully automated way while maintaining a high correlation to the human-devised measures.

Lukas Hollenstein and Lukas Lichtensteiger have contributed equally.

L. Hollenstein (✉) · L. Lichtensteiger · T. Stadelmann · M. Amirian · J. Meierhofer · R. M. Füchslin
ZHAW Zurich University of Applied Sciences, Winterthur, Switzerland
e-mail: hols@zhaw.ch; licn@zhaw.ch; stdm@zhaw.ch; amir@zhaw.ch; meeo@zhaw.ch; furu@zhaw.ch

L. Budde · T. Friedli
Institute of Technology Management, University of St. Gallen, St. Gallen, Switzerland
e-mail: lukas.budde@unisg.ch; thomas.friedli@unisg.ch



M. Braschler et al. (eds.), *Applied Data Science*,
https://doi.org/10.1007/978-3-030-11821-1_17

## 1 Introduction

One of the most important aspects of a data science project is the data itself. Its availability is *the* necessary condition for any successful data product that at its core relies on a successful analysis of this data. This fact seems obvious enough to be considered a truism, but nevertheless is the proverbial "elephant in the room" of countless data analytics projects. A recent survey among 70 data scientists showed that on average 36% of their projects have been negatively impacted by the unavailability of the data to be analyzed. The following paragraphs summarize the results from this poll.[1]

The survey has been conducted among the associates of the ZHAW Datalab.[2] The typical negative impact reported has been a delay of the project in the order of months, sometimes leading to a change of scope and goal up to the cancellation of the complete project (see Fig. 17.1). "Unavailability of data" here refers to the situation in which a data science project has been started under the requirement that specific data will be available at a certain point in the timeline of the project. The analysis of this data is the main part of the project and crucial to reach its goal, and all reasonable measures have been taken upfront to secure its availability. According to the survey, failing this requirement has usually one of the following reasons:

- *Measurement issues:* the data was meant to be collected in the course of the project but resource problems for staff to conduct measurements, the absence of specific events to be measured, or the unavailability of respective hardware hinder its collection.
- *Privacy issues:* the data is there but cannot be shared among the project partners due to new or unforeseen legal constraints.
- *Quality issues:* the raw data is available and shareable but the measurements themselves or the human-provided labels lack the required precision.

The effect of the unavailability of data is manifold: usually, it not only stretches the duration of an affected project by several weeks to years, it also leads to much more work on data curation at the expense of less time for the actual analysis and decreases the motivation on all parts of the project team, as was mentioned several times in the survey. It forces the data scientist to revert to suboptimal methods (with respect to the aspired project goal), and usually leads to lowered overall project goals up to a total cancellation of the endeavor. The matter is even more severe if data is not absent altogether, but some crucial parts are missing or its quality is far below the necessary standard. This ultimately leads to the same issues as outlined above; the

[1] While the survey and its evaluation have been conducted under controlled circumstances specifically for this chapter, we explicitly point out the small return rate of 10 questionnaires and hence the limited generality of conclusions; we report them because of their good correlation with our overall impression from numerous experiences with colleagues inside and outside our respective institutions.

[2] See www.zhaw.ch/datalab for a list of associates.

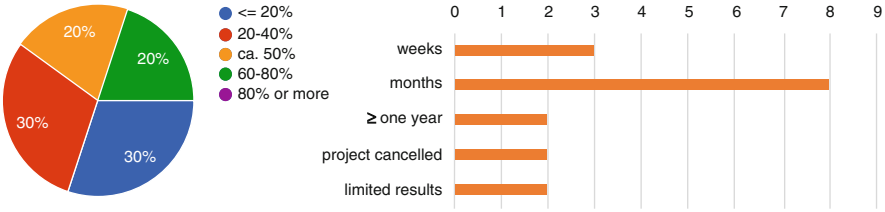


**Fig. 17.1** (Left) Survey results for the question "Which percentage of projects you worked on has roughly been affected by the unavailability of the data?". (Right) Answers to the question "How long have the affected projects typically been delayed (multiple answers allowed) by the unavailability of data?." Overall, the survey produced a 17% return rate (12 people), out of which 10 answered the above questions

more subtle form of the problem, however, may hinder project management to take appropriate measures early on, as was pointed out by several participants in the survey.

In this chapter, we provide a twofold contribution: first, we discuss a specific approach to partially overcome the above-mentioned issues of data unavailability by producing detailed data for machine learning out of a simulation model informed only by high-level process descriptions. Second, we introduce a novel measure of business operations complexity that can be evaluated fully automatically based on data and holds potential to inform business owners on how to reduce unwanted complexity. It exploits the idea that complexity and compressibility are highly anticorrelated (Schmidhuber 2008). Our case study from the area of Industry 4.0 is motivated by the assumption that in the time of growing mass customization in production (Fogliatto et al. 2012), variability in the product range leads to increased production complexity, which is a major driver of costs. (Note that there are scenarios where this assumption does not hold, e.g., cases where the variability of the product range enables the compensation of variabilities in the flow of resources. For more about this discussion, see Sect. 2.)

The goal of this case study hence has been twofold: first, to measure the inherent complexity in the production processes of certain industrial goods based on the analysis of production data; second, based on the complexity measurement, to propose changes to these processes that reduce the complexity while being feasible from a business and operations perspective. However, the necessary raw data from the shop floor turned out to be largely unavailable in the expected form.

In this situation, the methodology of coupling simulation and learning (Abbeel 2017) proved useful. Simulating the known physical properties of the production processes on an abstract level leads to many "observations" of the production of goods. Training an unsupervised learning algorithm like a neural autoencoder (Goodfellow et al. 2016) on this data converts the model from a physics-based simulation to a machine learning model with similar content, but different properties. The interesting property of the learned model with respect to the goal of the case study is the following: it has learned a compressed representation (Bengio et al. 2013) of the patterns inherent in the data, which is in the best case (a) able to generalize

(Kawaguchi et al. 2017) beyond the limitations and discontinuations of the abstract simulation; and (b) allows conclusions on how the original processes might be compressed (i.e., simplified) out of an analysis of its own way of compressing the learned information. Note that our two contributions—the suggestion to use simulation to overcome data scarceness, and the novel complexity measure—are independent of each other and only linked by the necessity of the case study under consideration.

The remainder of this chapter is organized as follows: Sect. 2 introduces the case study with its business background, showing the necessity and merit of a learned complexity measure. Section 3 details our methodology of linking simulation to unsupervised learning. Section 4 discusses the results of the case study before Sect. 5 concludes with several lessons learned on the problem of the unavailability of the analysis data in general.

## 2 Case Study: Complexity Management in Business Operations

The problem statement and solution approach described were applied in an industrial shop floor environment of a large international enterprise based in Switzerland. The factories are challenged with decisions about expanding the product portfolio for a higher degree of differentiation and an extended skimming of market segments, which is expected to yield higher revenues. However, it is obvious that even in the context of a modular production strategy in which new product versions are based on existing modules, increasing the product portfolio results in an increased complexity of the business operations in production, therefore resulting in increased production costs. Thus, there is a trade-off between higher revenue and higher costs. The availability of a tool to assess the complexity of a given production scheme based on measurable input data can provide a relevant support for the corresponding management decisions. Such a tool matches the definition of a so-called data product in the sense that it generates value from data for the benefit of another entity (i.e., the shop floor management) by the application of data science skills (Meierhofer and Meier 2017).

Product variety or complexity increase is often the outcome of the differentiation strategy of companies to enter market niches and to achieve higher revenues and market shares (Tang 2006). Beside the fulfillment of individual customer requirements and the outperforming of competitors (Lancaster 1990), researchers as well as practitioners in various studies reveal that an increase of product complexity does not equally lead to higher profitability and sustainable growth (Ramdas and Sawhney 2001). On the contrary, complexity is often associated with various negative effects that come attached and are built up over years (Fisher et al. 1999; Kekre and Srinivasan 1990). Several researchers claim the existence of an optimal level of product complexity that companies need to approach (Budde et al. 2015; Orfi et al.

2012; Krishnan and Gupta 2001). But the definition of the optimal level of complexity is not a trivial task because multiple factors need to be considered (Fisher et al. 1999; Budde et al. 2015). Product portfolio decisions (e.g., new product variants or new product developments) affect all steps along the value-chain, for example, development, production, and even service operations. Even minimal changes at the product architectures can have multiple impacts on the production or service side. This is also why decision-making around the product portfolio, such as decisions for new product development projects, product variants, or product architectures, is seen as one of the most critical tasks of management due to its uncertain and changing information, dynamic opportunities, and multiple and strategic considerations from different stakeholders along the value-chain (Closs et al. 2008).

Managers struggle to evaluate complexity out of a broader multifunctional perspective due to a lack of system interdependency knowledge and information asymmetries (Budde et al. 2015). This results in decisions that may be optimal for one functional perspective but not always optimal for the company along the product life cycle (Fisher and Ittner 1999; Closs et al. 2008; Lancaster 1990). Closs et al. (2008) recognized the need of metrics that measure the relational and combinatorial dimensions of complexity. These metrics should be able to predict various performance outcomes. Developing such a metric and deriving decision support from it for the case at hand was a central goal of our work.

In the given case study of the shop floor, data was available on the number of product alternatives and how they are composed as well as on the number of production steps required to produce those product types. However, within the practically given time frame of the project, it was not possible to gather the detailed data of the shop floor, for example, data about the sequence of the raw material or semifinished products across the machines or data about the load fluctuations of the individual machines. Higher effort than originally expected would have been necessary to generate all required information out of the different IT systems: the information was not directly available and not connected. Additionally, it was not possible to conduct different interviews with product managers as well as with experts from production or supply-chain departments due to organizational constraints.

Still, the project pursued the goal to make the resulting complexity of different production schemes measurable and thus to enable the assessment of different scenarios of product and production constellations. As stated in the introduction, the approach chosen here and explained in the following sections is based on training an unsupervised learning algorithm on data from simulations, which in turn are based on the scarcely available data and expert knowledge, thus transforming the physical model into a machine learning model that can provide insights into the inherent complexity on a more abstract level.

## 3 Linking Simulation and Learning

Even if there is no or insufficient data available to successfully train a machine learning model, some knowledge of the underlying nature of the system is often available from the domain experts. Here we discuss how for our case study we define a simulation model that can provide the data needed for a proof of concept of our complexity measure based on a neural autoencoder (Goodfellow et al. 2016).

### *3.1 Simulation Models Can Provide Data*

Simulations, as opposed to machine learning, are based on expert knowledge of the dynamics and rules of the complex system under analysis (Zeigler et al. 2000). Thus, in the absence of observations of the system (the desired data) we can simulate the behavior of the system by means of modeling its dynamics, running it (maybe many times), and gathering the observational data. Clearly, a simulation model needs data, too, but typically that data is of a higher level of abstraction, for example, the number of processing steps and the duration of each step for a given product. So, even if we do not have exact data for some of these values, like the durations, we can make some reasonable assumptions by talking to shop floor domain experts.

Many different simulation modeling approaches are known and the choice depends strongly on the system to be described and the knowledge we have about it (Zeigler et al. 2000). Roughly speaking, models can be characterized with respect to the following features: discrete versus continuous time/space, global versus local decisions/behavior, and deterministic versus stochastic decisions/data. Some examples are as follows:

- Physical and chemical systems are often continuous in time and space, have local forces (decisions), and are only rarely stochastic; this is why they are often described by differential equations.
- Production, supply-chain and logistics systems are discrete in time and space, decisions are often global, and they can be stochastic; thus, they are well-described by discrete-event simulations.
- Economic and sociological systems are also discrete in time and space and can be highly stochastic, however, often decisions and behavior is determined mainly locally, which is why agent-based simulation models are well-suited in this case.

With the decision for the simulation approach at hand, one can go ahead and determine the details of the model and what data is needed or needs to be generated to feed it. Validation of the simulation model is just as important as in any other simulation study. Since there is insufficient data available for direct validation, like when simulating systems that do not exist yet, one has to validate by means of consistency conditions provided by shop floor domain experts.

Finally, running the model (maybe many times) will generate the synthetic data for the subsequent machine learning step. The machine learning model is then trained on the generated data to faithfully reproduce the inputs, but with different properties than the simulation model: *our hypothesis (using the method detailed in* Sect. 3.3*) is that a successfully trained network will abstract essential features of the data used for training, and measuring the minimally required network complexity for successfully learning a given dataset would be a good measure for the complexity of the data itself.*

Once the network model is established on synthetic data, the case study that lacked data in the first place can now continue: the model trained on synthetic data can be embedded in its application and one can start testing, using, and refining it, until a freshly trained model can replace it once the real data is available.

Clearly, the fact that domain knowledge and high-level descriptions/data are needed for this approach can be seen as a drawback. On the upside, in many cases

- domain knowledge will anyway be needed for a successful data analytics project;
- higher-level descriptions/data are either already available or are not so hard to come by or estimate stochastically;
- the simulation modeling process leads to a deeper understanding of the domain and its dynamics;

which is why we argue that building a simulation model can successfully mitigate the issue of "unavailability of data" in the first place. The results obtained in that way will then have to be validated by different means, for example, through investigation by the original data owners.

## 3.2 A Concrete Example: The Job Shop Model

The goal of our case study was to measure the complexity of the manufacturing processes based on production data rather than based on business analysis. The production data desired by us would have been provided as an event log that traces the processing steps that each order undergoes on its way through the production system. Manufacturing systems like this are best modeled by discrete-event simulations that model the orders being passed from one process to the next, producing a discrete series of events in time—the exact data that we need in our case study.

In order to validate our complexity measure based on a neural autoencoder, we chose to implement a relatively simple model of a production facility, called the *job shop model* (Pinedo 2009); see the example depicted in Fig. 17.2. The job shop model describes all production steps as so-called *machines* that are visited by *jobs* that represent the orders for products to be produced. Machines can process only one job at a time. Each job has a list of *tasks*, which are combinations of machines and processing times of that job on the given machine. The tasks must be completed in the given order. Different products may visit machines in different orders and the number of tasks can vary as well. We do allow for recirculation, that is, a given job

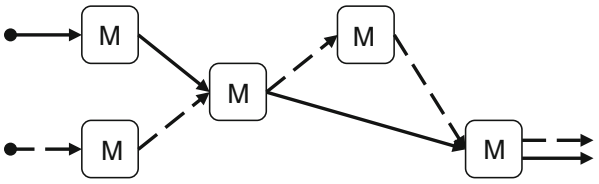


**Fig. 17.2** A job shop model with five machines, M1–M5, and two jobs, the sequences of solid and dashed arrows, respectively

may visit a given machine several times on its route through the system, and we allow a changeover time to be accounted for before a new job can be processed on a given machine. Determining the optimal sequence for a job shop is a classical NP-hard optimization problem in operations research (Pinedo 2009).

Once the sequence of the jobs to be processed and their task lists with the processing times are fixed, the model is fully deterministic. The simulation yields a log of events, each with timestamp, job ID, machine ID, and event-type, for example,

- Job entered in waitlist, job selected from waitlist
- Capacity blocked, capacity released
- Changeover started, changeover ended
- Processing started, processing ended

Thus, for given job sequences, the simulation model provides raw production data from a synthetic shop floor that can be fed into the machine learning model.

## *3.3 A Novel Neural Net-Based Complexity Measure of Industrial Processes*

In this section, we propose a novel measure to estimate complexity in production lines, based on a neural network, as well as an unsupervised approach to compute the measure. The goal is that for a given production line this complexity measure can be evaluated completely automatically without any human intervention and in (near) real time. The concept of complexity can be followed in compression theory (Henriques et al. 2013), learning theory (Zhu et al. 2009), and computational complexity theory (Park and Kremer 2015). The complexity of production lines is evaluated statically (Park and Kremer 2015) and dynamically (Fischi et al. 2015) in state-of-the-art research in order to improve manufacturing performance. Moreover, complexity can be evaluated for an entire dataset (Bousquet et al. 2004) or samples (Pimentel et al. 2014).

In our view, the complexity of a system can be quantified by how much a dataset containing an implicit full description of that system can be compressed without losing information about the system. For example, if the data describing all ongoing processes in a factory is very redundant, it can easily be compressed into a much shorter description, and the complexity of such a factory would be low. On the other

hand, for a factory where most ongoing processes are random, the description given by the data would be close to random and thus very hard to compress, and we would quantify this as a highly complex system.

In other words, our complexity measure for a system is the minimum description length or, equivalently, the maximum compression factor that can be achieved on datasets fully describing that system, without loss of information.[3] In principle, any compression algorithm could be used; however, the compression performance of those algorithms generally depends on the nature of the input data. For example, a compression algorithm that can achieve high compression factors for still images might perform quite badly on data consisting of moving images (i.e., video sequences). Since we are interested in the maximum compression rate, we need compression algorithms that are working well for the specific kind of input data we have. One way would be to hand-design good compression algorithms for our data; however, this would require obtaining a deep understanding of the underlying structures in our data by hand, which would be very labor-intensive.

For this reason, we chose to use neural networks for data compression. Unsupervised training of neural networks provides a fully automated way to extract such underlying structures from data, which are needed for good compression performance. The system is adaptive to a large degree, that is, for data with different characteristics it will automatically find the underlying structures that are better suited there. There is no need to hand-tune the compression algorithm as would be the case with classical, nonadaptive algorithms. Of course, training the network on a specific dataset requires time, but hand-tuning algorithms—in addition to time—would also require deep knowledge about the underlying data structures. Neural networks, on the other hand, once trained, can be used to *discover* such high-level structures and features in underlying data (while this promises to be a very interesting extension of our approach, this is beyond the scope of the current chapter and referred to future work).

In this chapter, we hence propose to measure the complexity of a production line using neural networks, specifically autoencoders (Goodfellow et al. 2016). The proposed measure evaluates the complexity of an entire production process. This approach is fully unsupervised and does not need any labeled data. However, a sufficient amount of data is required in order to train the autoencoder.

An autoencoder is trained to produce a replica of its input at the output layer. The structure of this type of neural network consists of a number of hidden layers connecting the input and output layer. In general, autoencoders contain a code layer as well as an encoding and a decoding function. The code is a representation of the input learned through the unsupervised training procedure. The dimensionality of the code is smaller than both input and output in undercomplete ("compressing")

[3] In principle, the compression does not need to be lossless in the strict meaning of the word. While on the noise-free simulation data used in experiments below, maximum compression while maintaining losslessness provides a natural threshold for the degree of compression in our measure, some degree of loss might even be desirable on real-world data to get rid of inherent noise from measurement errors, etc.

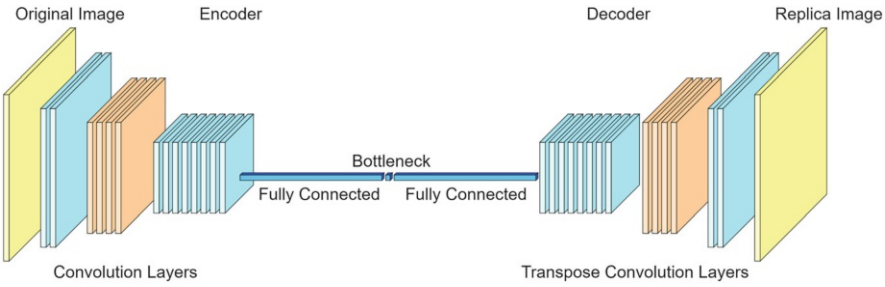


**Fig. 17.3** The structure of a deep autoencoder with encoder, decoder, and the code (bottleneck) in between. As in the picture, we also encoded our data in image format (see Sect. 4.2)

autoencoders. In this case, the code forms an information bottleneck between the encoding and decoding networks, as depicted in Fig. 17.3.

The complexity measure we propose here is the minimal bottleneck dimensionality for the autoencoder to yield lossless reconstruction. Its value does not directly represent any features of the production line process but rather reflects its overall complexity in an abstract way. The production line data from the simulation (see Sec. 3.2) is considered a source of information for lossless compression and reconstruction. Each job in the job shop is represented as a certain temporal sequence of processing steps on several machines and is encoded as a two-dimensional matrix, where one dimension is the discretized process time and the other dimension is the ID of the process machine (see Fig. 17.5). An entry in the matrix is set to one if the corresponding machine is active at that time step, and set to zero otherwise. These matrices can be interpreted as patterns or images, and the set of all patterns of all jobs occurring in a given job shop provides a representation of the complete activity of this job shop. The autoencoder tries to compress the set of all these patterns as much as possible, without loss of information. The minimum code length (i.e., the size of the smallest bottleneck layer) that can achieve this is related to the information content in the activity patterns, and is chosen as our complexity measure for this job shop.

Based on Shannon's source coding theorem (Shannon 2001), it is possible to asymptotically obtain a code rate that is arbitrarily close to the Shannon entropy (Shannon 2001) in lossless compression of a source of information. The code rate refers to the average number of bits per symbol (products in production lines) in this definition. Importantly, lossless compression of a source is not possible with a code rate below the Shannon entropy (Shannon 2001). The dynamics of the production line is initially represented in the form of images containing temporal information as well as machine identification numbers. The autoencoder subsequently performs a lossless compression of these images. Therefore, the code rate in this context corresponds to the compression ratio of images (information of production lines) to code (bottleneck of the autoencoder).

The Shannon entropy of a source of information determines the lower band of the code rate. Therefore, the minimum code rate can be used as an approximation for the Shannon entropy. Assuming a source with fixed input length in the encoder, and specifically the autoencoder, the code rate only depends on the code length or the

bottleneck dimensionality. Therefore, the Shannon entropy of the source of information (production line) is proportional to the minimum dimensionality of the bottleneck in the autoencoder. Feldman and Crutchfield (1998) explain why the Shannon entropy is a measure of statistical complexity. Recently, Batty et al. (2014) used this measure to analyze spatial information and complexity. The proposed measure of complexity in this work, minimum dimensionality of the autoencoder bottleneck (code), is directly proportional to the entropy, which is a measure of complexity. It reflects the temporal usage patterns of the machines in the production line; the more different patterns that are needed to represent the system dynamics, the more complex it is.

## 4 Experiments and Discussion

Here we provide and discuss a proof of concept for our neural-network-based complexity measure for production systems. To show its validity, we generate a series of instances of the job shop model, produce the simulation event logs, and measure the complexities of each scenario in two ways: first, using a conventional complexity measure based on business analysis (see Sect. 4.3), and second, computed with our neural-network-based measure discussed in Sect. 3.3. Since the data for the original case study was not available, we used the job shop simulation model to produce the data required for the proof of concept.

### *4.1 Scenarios*

We investigate the complexity of a series of instances of the job shop model. Starting from a simple base scenario, we vary several features in different directions, targeting different complexity drivers, namely, the number of processing steps, the number of products, the percentage of dedicated production lines, and the manufacturing stability. Introducing these variations leads to different production complexities for each scenario.

The base scenario has machines grouped in three stages that all jobs visit in the same order, reflecting the realistic situation where products typically go through stages like setup, assembly, and packaging. We generate 800 jobs that have tasks sampling all possible combinations of machines in those three stages, all with the same processing times. See Fig. 17.4 for an illustration of the base scenario with two machines per stage.

To produce different scenarios, we focus mainly on the second stage and change the number of machines available, the processing times of jobs on individual machines, the availability of machines, or the processing time depending on the choice of machine in the first stage. In addition, we enlarge the base scenario to encompass three and four machines per stage, respectively, and generate variations analogous to those described above.

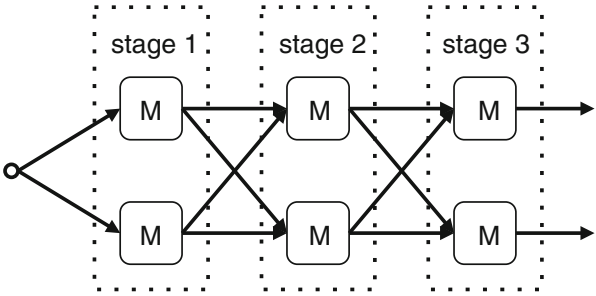


**Fig. 17.4** The base scenario with two machines per stage. Each job visits the three stages in the same order and can be assigned to either machine per stage. Here, the arrows represent all possible paths of jobs through the system

## *4.2 Data Preprocessing and Autoencoder Network Topology*

Before being fed into the autoencoder for complexity analysis, the data from the job shop has to be preprocessed. We do this in a way that allows automated integration in real factory settings in an Industry 4.0 environment later, namely, using timestamps of process steps. When jobs pass through the simulated job shop, each process step produces a timestamp when it is started and when it is stopped, together with the ID of the machine on which it is run (see Sect. 3.2).

For each job, we generate a two-dimensional matrix from this information, where one dimension is indexed by machine ID and the other by elapsed (discrete) time steps since the job started. If the machine with ID $j$ is processing the given job at time step $k$, then in the corresponding matrix the entry at position $(j, k)$ is set to +1, otherwise to $-1$. In other words, the processing of a job in the job shop can be represented as a two-dimensional pattern of black and white pixels, where white pixels indicate active machines at the corresponding time step, black pixels indicate inactive machines, and process steps are represented by horizontal white lines of different lengths (see Fig. 17.5). Each of these patterns constitutes a training pattern for the autoencoder, and each pixel position in the pattern is fed into a corresponding neuron in the input layer of the autoencoder. In our simulations, we use a maximum number of 16 machines and 61 time steps, so all our input patterns have a fixed size of $16 \times 61$ pixels. Note that not every machine or time step is used in every scenario; unused entries will simply be zero. We chose to keep the input dimensions fixed over all scenarios so that the number of weights in the neural networks would not depend on the scenario, allowing better comparability between scenarios. Thus, the input dimensions are just chosen large enough to accommodate the maximum number of machines and time steps in any of the scenarios.

While it would be possible to use a classical autoencoder (with fewer and fully connected hidden layers as the one depicted in Fig. 17.3) directly on these input data, for these fully connected networks the relatively large number of inputs ($16 \times 61 = 976$) leads to a rather large number of weights, resulting in slow learning and large training data requirements. Therefore, we decided to use a different network topology for our autoencoder: immediately after the input layer we use a stack of three convolutional layers, followed by two fully connected layers with a central hidden layer (the actual autoencoder), and finally a stack of three "transpose

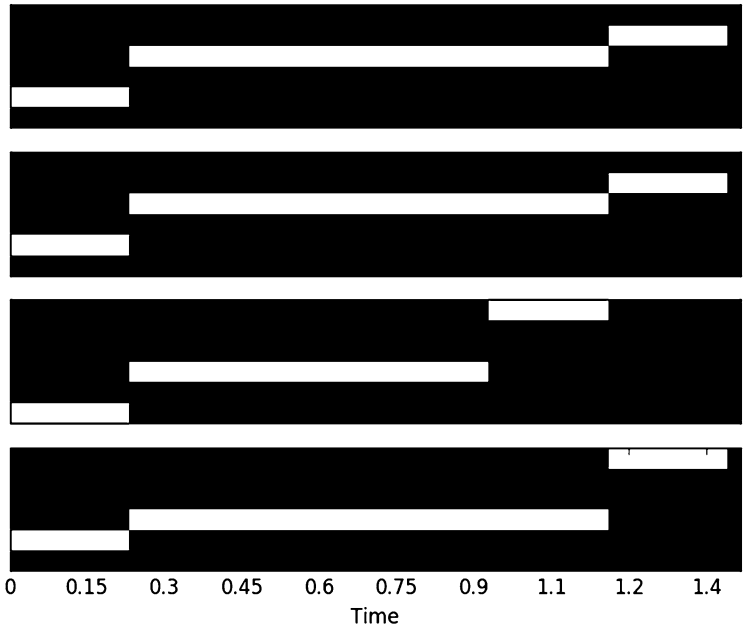


**Fig. 17.5** Example of four input patterns for the autoencoder, generated from four different jobs from the simulated job shop. In each pattern, each row of pixels corresponds to the activity of a certain machine during the processing of that job (only 6 out of 16 potentially active machines per job are shown in this illustration). The horizontal axis represents (discrete) time passed since the start of that job. White pixels denote the corresponding machine being active at the corresponding time step, black pixels represent inactive machines

convolutional" layers to revert the action of the convolutional layers (Stadelmann et al. 2018).

The convolutional layers are very good at extracting information from two-dimensional pictures with geometric features such as our horizontal process step lines, while requiring relatively few weights due to weight sharing. Furthermore, since we use a stride of 2 in each layer, also the dimensions of the input patterns are reduced accordingly. Using $3 \times 3$ filters, we compared different filter numbers and found that for 2, 4, and 8 filters in the 1st, 2nd, and 3rd convolutional layer, respectively, the network could learn to map all input patterns for all scenarios to the correct output patterns, using less than 10 neurons[4] in the central hidden layer in all cases. It should be pointed out that in spite of the larger number of layers, our network actually has much fewer weights than a traditional autoencoder: for example, for a case with 8 neurons in the bottleneck layer, the simplest traditional three-layer autoencoder would require 16,600 weights, whereas our network topology only requires 2961 weights to be learned for that case.

[4]This specific number depends on the concrete data used and can be determined experimentally for any real data.

For comparability, we decided to fix the convolutional layers at the configuration described above and only vary the number of neurons in the central hidden layer. The minimum number of hidden neurons for which the network could still learn the correct input–output mappings for all patterns (jobs) in a given scenario was then chosen as the complexity number for that scenario. In other words, the network had to map all input patterns (representing all jobs in the production line) successfully back to the *same* input patterns while passing this information through a small bottleneck layer, and the minimum size of the bottleneck layer for which this was possible was chosen as the complexity number. It should be noted that this is only a *relative* complexity measure, since changing the network configuration of the convolutional layers will affect the minimal number of hidden neurons required. In other words, how the data is preprocessed affects how easily it can be learned (Lichtensteiger and Pfeifer 2002). Here, having less filters in the convolutional layers will require more neurons in the central hidden layer for still being able to learn successfully. However, since we are not yet able to quantify this influence appropriately, for this study we decided to fix the convolutional network topology at a configuration that was shown to work well and focus only on the number of neurons in the central hidden layer for our complexity measure.

In order to verify the self-consistency of our complexity measure, we did a second full run of experiments where we used different weight initializations for the networks and added strong multiplicative random noise in the neural activities of the bottleneck layer. In addition, we varied the size of input patterns by adding different amounts of zero padding. Our first results show that in spite of these rather substantial changes to the network, the resulting complexity measures do not change significantly, indicating the robustness of our approach. With regard to computational runtime, on a desktop PC equipped with an Intel Xeon Processor E5-2620 running at 2.40 GHz and an NVIDIA Quadro M4000 GPU, learning the correct input–output mappings for all patterns (jobs) in a given scenario required around 1–5 min. When the number of neurons in the bottleneck layer was changed, the system had to learn again. Since calculating complexity required finding the minimum number of neurons in the bottleneck layer for which learning was successful, using, for example, binary search around 5–10 variations of neuron numbers were needed. Therefore, calculating our complexity measure for a given scenario took around 10–50 min on our hardware configuration.

### *4.3 Results*

To validate our neural-network-based complexity measure we compare it to a state-of-the-art conventional method (Friedli et al. 2013). It is computed as a weighted sum over contributions from the following factors (complexity drivers): number of process steps, percentage of dedicated production lines, number of changeovers, flexibility upside, and number of batches. These factors are measured for all simulated job shop scenarios and normalized to the interval between 0 and 1. Note that we

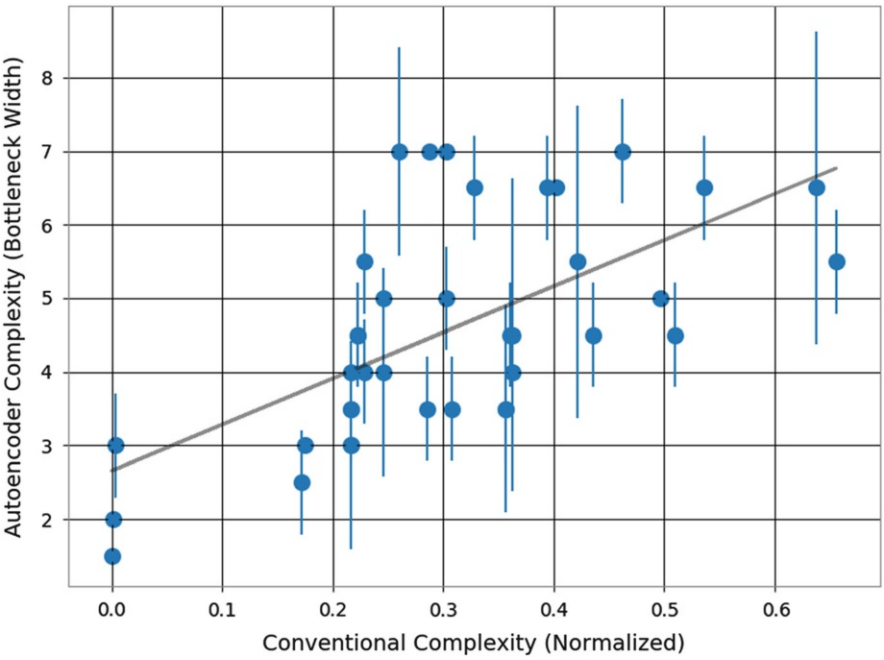


**Fig. 17.6** The complexity values for all scenarios from the autoencoder (minimal number of bottleneck neurons) is plotted against the conventional complexity normalized over all scenarios. The Pearson correlation coefficient is $p = 0.637$ ("fair correlation"). The error bars show the standard deviation of two different series of experiments (see end of Sect. 4.2)

did not consider manufacturing stability as a driver of complexity here, since the scheduling of the job shops is static and therefore the simulations are deterministic. For the weights of the individual complexity drivers we take the results from Friedli et al. (2013), and renormalized them to 1 after neglecting the weight for manufacturing stability.

Figure 17.6 shows the averages of our autoencoder-based complexity values from the two experimental runs plotted against the complexity values obtained using the conventional method. The error bars show standard deviations as conservative indicators of the variability of our approach, see discussion in the end of Sect. 4.2. The Pearson correlation coefficient is $p = 0.637$, which indicates a fair correlation. This shows that our autoencoder-complexity measures at least partly the same features as the conventional method does, rendering it a valuable tool in the analysis of production and process analysis while being determined completely in a data-driven manner. This result is to be understood as a first proof-of-concept. To improve the understanding of the relation between the two complexity measures and the dependency of the autoencoder complexity on the features of the production processes and product architectures, a complete study based on larger job shops and, preferably so, real data is needed and aimed for.

## 5 Conclusions

We claim that data analytics projects need data to be analyzed. Often taken for granted and not seriously planned as a potential showstopper, the unavailability of data of the right quality, at the right granularity, and in a reasonable project time frame may put entire projects at risk. The message is clear: gathering the right data is not to be underestimated and can make up by far the majority of the project time.

**Lesson Learned #1** For future projects, special attention needs to be paid to the measurement and gathering of the specifically required data out of the production systems.

The research conducted in this chapter showed a feasible way of how to deal with unavailable data when one is hit by it. Specifically, available high-level data can be turned into a simulation model (using extra help from domain experts) that produces finer-grained synthetic data in arbitrary quantity (but in quality bound to the explicitly modeled aspects of the simulation). This finer-grained data (independent of originating from direct measurements or simulations) can in turn be used to train a machine-learning model with intriguing properties: it inherits the properties of the simulation model while being able to generalize beyond its discontinuities. This study used state-of-the-art unsupervised learning schemas (deep convolutional compressing autoencoders) for this task.

**Lesson Learned #2** Coupling simulation and machine learning to "convert" models of the real world and thus get access to the intriguing properties of each method is a powerful tool. In the presented scenario we show how simulation can be used to provide missing input data, at least until the real data can be provided. In an age where data is considered extremely valuable, yet sometimes still scarce if too specialized, this is an important methodology in many domains from sociology to traffic, energy, and health.

We specifically introduced a novel complexity measure for industrial product architectures and process topology based on the minimum dimensionality of the bottleneck layer of our trained autoencoder. We computed this complexity measure for a range of production line scenarios, inspired by real situations in our case study. Comparing those values to the state-of-the-art complexity measures based on conventional complexity drivers suggested by business experts, we find that the two measures are fairly correlated (see Fig. 17.6), which we interpret as a proof of concept for the autoencoder approach. As opposed to the conventional measure that is based on expert knowledge and extensive human effort (qualitative interviews and subsequent work of economists), our measure has the advantage of being learned completely in an unsupervised fashion from timestamped process data alone. Note that we are not suggesting to always use this complexity measure in conjunction with a respective simulation model of the production system in question. On the contrary, the aim for further work is to establish our complexity measure by testing it in real-world situations with real shop floor data, using it as a tool to identify unwanted complexity and suggest changes in process structures and product architecture that reduce this complexity and the associated costs.

**Lesson Learned #3** The paradigm of data-driven decision support can even enter the domain of a highly qualified business consultant (that would usually estimate the classical complexity measure manually), delivering the quantitative results necessary to ponder informed management decisions.

Neither the complexity measure itself, nor the neural autoencoder architecture, or the necessary data, are highly sophisticated. They are based on available information and common-sense ideas, implemented and thoroughly verified but not much changed from the original idea. While a first prototype like this case study shows the traits of a research project, nothing hinders its direct application by engineers in business in the next scenario that is somewhat similar.

**Lesson Learned #4** It is merely the knowledge of what methods and technologies are possible and available that currently hinders the faster adoption of the data-driven paradigm in businesses.

Neither the involved simulation methods, nor the used machine learning techniques, nor the idea of bootstrapping machine learning with simulation per se are novel. Nevertheless, the data-driven complexity measure is new and arises simply as a straightforward combination of available technologies and methodologies. Innovation in this project arose from the collaboration of experts, not from individual novel developments [see also Swiss Alliance for Data-Intensive Services (2017)].

**Acknowledgments** The authors are grateful for the support by CTI grant 18993.1 PFES-ES, and for the participation of our colleagues from the ZHAW Datalab in the conducted survey.

## References

Abbeel, P. (2017). *Pieter Abbeel: Deep learning-to-learn robotic control* [video-file]. Retrieved from https://youtu.be/TERCdog1ddE

Batty, M., Morphet, R., Masucci, P., & Stanilov, K. (2014). Entropy, complexity, and spatial information. *Journal of Geographical Systems, 16*(4), 363–385.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 35*(8), 1798–1828.

Bousquet, O., Boucheron, S., & Lugosi, G. (2004). Introduction to statistical learning theory. In *Advanced lectures on machine learning* (pp. 169–207). Heidelberg: Springer.

Budde, L., Faix, A., & Friedli, T. (2015). From functional to cross-functional management of product portfolio complexity. In *Presented at the POMS 26th Annual Conference*, Washington, DC.

Closs, D. J., Jacobs, M. A., Swink, M., & Webb, G. S. (2008). Toward a theory of competencies for the management of product complexity: Six case studies. *Journal of Operations Management, 26*(5), 590–610. https://doi.org/10.1016/j.jom.2007.10.003.

Feldman, D. P., & Crutchfield, J. P. (1998). Measures of statistical complexity: Why? *Physics Letters A, 238*(4–5), 244–252.

Fischi, J., Nilchiani, R., & Wade, J. (2015). Dynamic complexity measures for use in complexity-based system design. *IEEE Systems Journal, 11*(4), 2018–2027. https://doi.org/10.1109/JSYST.2015.2468601.

Fisher, M. L., & Ittner, C. D. (1999). The impact of product variety on automobile assembly operations: Empirical evidence and simulation analysis. *Management Science, 45*(6), 771–786.

Fisher, M., Ramdas, K., & Ulrich, K. (1999). Component sharing in the management of product variety: A study of automotive braking systems. *Management Science, 45*(3), 297–315.

Fogliatto, F. S., Da Silveira, G. J., & Borenstein, D. (2012). The mass customization decade: An updated review of the literature. *International Journal of Production Economics, 138*(1), 14–25.

Friedli, T., Basu, P., Bellm, D., & Werani, J. (Eds.). (2013). *Leading pharmaceutical operational excellence: Outstanding practices and cases*. Heidelberg: Springer. https://doi.org/10.1007/978-3-642-35161-7.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Cambridge, MA: MIT Press. Retrieved December 22, 2017, from http://www.deeplearningbook.org

Henriques, T., Gonçalves, H., Antunes, L., Matias, M., Bernardes, J., & Costa-Santos, C. (2013). Entropy and compression: Two measures of complexity. *Journal of Evaluation in Clinical Practice, 19*(6), 1101–1106.

Kawaguchi, K., Kaelbling, L. P., & Bengio, Y. (2017). Generalization in deep learning. *CoRR, 1710*, 05468. Retrieved December 22, 2017, from https://arxiv.org/abs/1710.05468

Kekre, S., & Srinivasan, K. (1990). Broader product line: A necessity to achieve success? *Management Science, 36*(10), 1216–1232.

Krishnan, V., & Gupta, S. (2001). Appropriateness and impact of platform-based product development. *Management Science, 47*(1), 52–68.

Lancaster, K. (1990). The economics of product variety: A survey. *Marketing Science, 9*(3), 189–206.

Lichtensteiger, L. & Pfeifer, R. (2002). An optimal sensor morphology improves adaptability of neural network controllers. In J. R. Dorronsoro (Ed.), *Proceedings of the International Conference on Artificial Neural Networks (ICANN 2002)*, Lecture Notes in Computer Science LNCS 2415 (pp. 850–855).

Meierhofer, J. & Meier, K., (2017). From data science to value creation. In St. Za, M. Drăgoicea, & M. Cavallari (Eds.) *Exploring Services Science, 8th International Conference*, IESS 2017, Rome, Italy, May 24–26, 2017, *Proceedings* (pp. 173–181). Cham: Springer.

Orfi, N., Terpenny, J., & Sahin-Sariisik, A. (2012). Harnessing product complexity: Step 2—measuring and evaluating complexity levels. *The Engineering Economist, 57*(3), 178–191. https://doi.org/10.1080/0013791X.2012.702197.

Park, K., & Kremer, G. E. O. (2015). Assessment of static complexity in design and manufacturing of a product family and its impact on manufacturing performance. *International Journal of Production Economics, 169*, 215–232.

Pimentel, D., Nowak, R., & Balzano, L. (2014, June). On the sample complexity of subspace clustering with missing data. In *2014 IEEE Workshop on Statistical Signal Processing* (SSP) (pp. 280–283). IEEE.

Pinedo, M. L. (2009). *Planning and scheduling in manufacturing and services* (2nd ed.). Dordrecht: Springer.

Ramdas, K., & Sawhney, M. S. (2001, January 1). *A cross-functional approach to evaluating multiple line extensions for assembled products* [research-article]. Retrieved January 15, 2014, from http://pubsonline.informs.org/doi/abs/10.1287/mnsc.47.1.22.10667

Schmidhuber, J. (2008). Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. In *Workshop on anticipatory behavior in adaptive learning systems* (pp. 48–76). Heidelberg: Springer.

Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review, 5*(1), 3–55.

Stadelmann, T., Tolkachev, V., Sick, B., Stampfli, J., & Dürr, O. (2018). Beyond ImageNet - deep learning in industrial practice. In M. Braschler, T. Stadelmann, & K. Stockinger (Eds.), *Applied data science: Lessons learned for the data-driven business*. Heidelberg: Springer.

Swiss Alliance for Data-Intensive Services. (2017). Digitization & innovation through cooperation. *Glimpses from the digitization & innovation workshop at "Konferenz Digitale Schweiz"*. Retrieved April 26, 2018, from https://data-service-alliance.ch/blog/blog/digitization-innovation-through-cooperation-glimpses-from-the-digitization-innovation-workshop

Tang, C. S. (2006). Perspectives in supply chain risk management. *International Journal of Production Economics, 103*(2), 451–488. https://doi.org/10.1016/j.ijpe.2005.12.006.

Zeigler, B. P., Kim, T. G., & Praehofer, H. (2000). *Theory of modeling and simulation* (2nd ed.). Orlando, FL: Academic Press.

Zhu, X., Gibson, B. R., & Rogers, T. T. (2009). Human rademacher complexity. In *Advances in neural information processing systems* (pp. 2322–2330). Cambridge, MA: MIT Press.