Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Jan Paul Assendorp

Deep learning for anomaly detection in multivariate time series data

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jan Paul Assendorp

Deep learning for anomaly detection in multivariate time series data

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai v. Luck
Zweitgutachter: Prof. Dr. Andreas Meisel

Eingereicht am: September 1, 2017

Jan Paul Assendorp

Thema der Arbeit

Deep-Learning zur Anomalie-Erkennung in mehrdimensionalen Zeitreihendaten

Stichworte

Deep-Learning, Machine-Learning, Anomalie-Erkennung, Zeitreihen, Sensordaten Autoencoder, Generative Adversarial Network

Kurzzusammenfassung

Das Erkennen von Anomalien in Sensordaten ist ein wichtiger Anwendungsfall in der Industrie, um Fehler in maschinellen Prozessen frühzeitig erkennen zu können und potentiellen Schäden vorzubeugen. In dieser Arbeit wird ein Deep-Learning-Verfahren entwickelt, welches in mehrdimensionalen Sensordaten ungewöhnliche Muster erkennen kann. Dafür werden Echtdaten aus einer industriellen Anwendung verwendet.

Jan Paul Assendorp

Title of the paper

Deep learning for anomaly detection in multivariate time series data

Keywords

Deep Learning, Machine Learning, Anomaly Detection, Time Series Data, Sensor Data, Autoencoder, Generative Adversarial Network

Abstract

Anomaly detection is crucial for the procactive detection of fatal failures of machines in industry applications. This thesis implements a deep learning algorithm for the task of anomaly detection in multivariate sensor data. The dataset is taken from a real-world application.

Danksagung

An dieser Stelle möchte ich mich zunächst bei Prof. Kai v. Luck und Prof. Andreas Meisel für die Betreuung dieser Arbeit herzlich bedanken. Besonderer Dank gilt zudem der Firma Werum Software & Systems und damit vor allem Hendirk Bohlen, Christan Sommer und Frank Hentschel für das Ermöglichen dieser Arbeit und die ausgiebige Unterstützung. Des weiteren danke ich Prof. Ulf Brefeld für die zahlreichen Anregungen und das Feedback.

Natürlich möchte ich mich auch herzlich bei Natalia, meinen Freunden und meiner Familie bedanken, die mich in der vergangenen Zeit begleitet haben.

Contents

List of Tables

List of Figures

1 Introduction

Detecting anomalous behavior of mechanical devices is an important task to prevent failures that result in unwanted outcome or even cause damage to mechanical parts. Todays manufacturing industry intensively utilizes numerous sensors to seamlessly monitor the mechanical and electrical properties of machines. The recorded sensor data can subsequently be examined to distinguish normal from unexpected behavioral patterns.

The recent years showed a constant trend for cheaper and more capable hardware for both, storing and processing large amounts of data. As a result, companies are able to store previously not feasible datasets containing high frequent sensor data. Even with available datasets, it remains a difficult task to gain relevant insights from the data in order to implement algorithms for reliable detection of unexpected and possibly malicious behavior.

In practice, this is often accomplished by engineers with sufficient knowledge of the domain. Rules can be manually constructed according to the experts experiences with regard to constraints on the machines behavior. These rules can subsequently be incorporated into an expert system, that monitors the sensor data and rises alerts, once the data shows patterns that does not match the set of rules. Implementing these expert systems is often expensive in terms of time and the required domain knowledge.

Machine learning algorithms allow to derive knowledge from data to fit a predictive model that can further assist to make data-dependent decisions. This includes the task of anomaly detection, where e.g. a simple model based on covariance statistics can predict the probability of a certain pattern belonging to the known distribution of the sensor data from the recorded dataset. In recent literature, *deep learning* algorithms showed robust anomaly detection capabilities in complex domains, where anomalous samples can only be detected by taking into account the temporal dependencies in a multivariate sensor setting. In addition, deep learning models can learn hierarchical representations from raw input data and hence hold the potential to reduce manual feature engineering to a minimum. Given the right circumstances, deep learning models showed to be more robust as well as less expensive because manual work requiring domain knowledge can be reduced to a minimum.

This thesis explores deep learning algorithms from recent literature with the scope of building a reliable anomaly detection system for a real-world industry application, where washer-dryers are tested for durability. The available dataset contains recorded sensor data from different machines, that where tested over a fixed amount of washing cycles in order to validate the durability over the expected lifespan of the machines. For this purpose, two different approaches from recent publications have been implemented and trained on the dataset. The results are evaluated to show the potential of these approaches for the given industry application.

Following this introduction, chapter 2 specifies the objective for this thesis and subsequently discusses the motivation to apply deep learning for the task of anomaly detection. Further, section 2.3 introduces different approaches from recent deep learning literature. Chapter 3 then shows experiments for two different algorithms applied to the dataset. The results of these experiments will be discussed in chapter 3.5. Finally, chapter 4 concludes the thesis and gives a summary about future work, that should be conducted based on the results.

2 Analysis and Literature Review

Machine Learning methods have recently been very popular for pattern recognition in multivariate sensor data throughout various domains. In this thesis, machine learning will be applied for detecting anomalous patterns within sequences of multivariate sensor data from a real-world dataset. This chapter will analyze the task of anomaly detection and discuss recently applied machine learning methods which have proven to achieve good results in similar tasks.

Section 2.1 will specify the goal of this thesis. The motivation to apply deep learning to this specific use case will be given in section 2.2. Finally, section 2.3 will explore recent deep learning architectures that have been successfully applied to the task of anomaly detection or a similar task.

2.1 Anomaly Detection Objective

The objective of this thesis is to evaluate recent deep learning architectures for detecting anomalies in a specific dataset. To further specify the objective, the general task of anomaly detection will be defined in 2.1.1. The objective is then specified in 2.1.2.

2.1.1 Anomaly Detection

Anomalies in sensor data can be defined as previously unseen patterns, that do not match the expected behavior of the monitored machine. Hence, it is crucial to have a rich understanding of the behavioral characteristics of machines in order to distinguish normal from anomalous patterns in newly recorded data. The reliable detection of anomalous patterns in continuous sensor data is an important challenge in today's manufacturing industry, as previously unseen pattern can be a sign of misconfiguration, increasing mechanical wear-out or simply an unforeseen situation which can eventually lead to failures. Since many modern industry settings can rely on numerous sensors providing large streams of data from monitoring mechanical or electrical behavior of devices, there is often a high potential for the proactive detection of failures through identifying anomalous patterns.

Currently, this process is widely handled by engineers, who have advanced knowledge of the machinery in the specific domain. This knowledge enables engineers to manually detect anomalous samples within the recorded time series data based on their experience. However, as many settings record massive amounts of sensor data, the manual detection of anomalies is prone to human error. Additionally, when it comes to multivariate sensor data, this task can be immensely time-consuming, as the dataset can consist of a few hundred different sensors capturing millions of data points during the machine's life cycle. In this case, the reliable detection of anomalous samples in a timely manner can become impossible to handle manually.

A rather simple way to monitor sensor data is to define threshold values for each sensor channel and implement a system which automatically raises notifications once the specified threshold is violated. This reduces the time for manually monitoring devices but can result in many false alerts, especially in complicated domains. Even more problematic are failures which cannot be detected by thresholds but rather require to take into account the joint characteristics of multiple channels. Instead of supervising each channel for itself, a machine learning model can be trained to detect pattens in the combined sensor values. Thus, the task of detecting anomalous samples can be fully automated given an appropriate machine learning algorithm. This allows the engineers to concentrate on other important tasks, that cannot be automated easily.

Machine learning and especially available algorithms for pattern recognition and classification have proven to be very successful in many different domains. Therefore, a vast range of publications solving similar tasks with machine learning algorithms can be referred to as reference to solve a specific problem. The most common use case is to train a classifier on available examples in order to detect and distinguish different patterns within new data. This could also be applied for the detection of anomalous samples within sensor data, given the limitation that anomalous examples are available. Training a classifier requires a sufficient amount of labeled training examples. This might be applicable, once there is a known set of failures which have been extensively recorded and are therefore available for training.

In general, the task of anomaly detection is characterized by a highly unbalanced class distribution. In most cases, many instances of normal behavior are available, whereas the number of anomalous samples is very limited. As anomalous patterns are connected to machine failures, recording instances of anomalies can be very expensive. Hence, the challenge in anomaly detection lies in modeling the normal patterns and detecting previously unseen patterns, which might hint to machine failure. This can be solved e.g. by looking at the data distribution of normal data and comparing it to new examples. In many applications, this is achieved with *One-Class Support Vector Machines* Schölkopf et al. (2000), which can detect novel patterns by looking at the local density distribution of normal data.

Another approach is to predict the machines behavior by looking at the sensor data up to the current time step. However, many machines show certain behavioral characteristics constrained by an internal state that cannot be easily derived from the sensor data itself. Therefore, the machine's actions are difficult to forecast based on previous data, as the state may change unexpectedly. In these cases, it is necessary to incorporate meta information at every time step.

The challenges of anomaly detection in multivariate sensor data can be summarized as follows:

- large datasets with high frequent, real-valued sensor data
- highly multivariate data consisting of several different sensor channels
- temporal and multimodal dependencies

In recent literature, many different approaches have been successfully applied to the task of detecting anomalous patterns in sensor data. Binary classification can be seen as the most straightforward method in terms of the underlying model and the computational complexity. However, a simple classification-based model requires an adequate amount of data samples for each, the normal as well as the anomalous class. As it is often difficult to obtain a sufficient amount of anomalous data samples in real-world applications, it is often required to use a model which can be trained exclusively on normal samples and afterwards detects previously unseen patterns in new data samples. Different approaches from recent deep learning literature for unsupervised modeling of normal data will be discussed in more detail in section 2.3.

2.1.2 Specification of the Objective

The objective of this thesis is to explore deep learning methods for the reliable identification of anomalous patterns within sensor data. Subsequently, a suitable approach will be applied to a real-world dataset. The dataset is taken from an industry application, where machinery is monitored in a long-term test setting for durability and possible failure. Details about the dataset will be discussed in section 3.1.

For this thesis, anomalies are defined as unexpected patterns which were previously not observed in the sensor data, as described in 2.1.1. These anomalous patterns can be related to possible failures of the monitored machine. Recorded data is expected to consist solely of normal sequences. This implies that no failures occurred during the time the data was recorded. Otherwise, occurring anomalies are expected to be labeled accordingly. These known cases of failure can be used to validate the model. Hence, the algorithm for anomaly detection should be able to detect known anomalies as well as generalize to new and unknown anomalies. This extends the understanding of anomalies to include already known cases of failure or entirely new and previously unseen behavior that can hint to mechanical failure of the machines.

The objective focuses on using machine learning to detect anomalies. Machine learning provides tools for handling immense amounts of data with distributed algorithms. Further, machine learning allows modeling complicated behavior based on unknown underlying rules from a specific domain. In general, a complex domain makes it very hard to implement an expert system, where task-specific assumptions are made based solely on a set of manually defined rules. However, this requires extensive work by engineers with sufficient domain knowledge. In contrast, a machine learning algorithm can automatically derive the complex set of underlying rules from the data itself and encode them directly into an algorithm. Subsequently, the resulting algorithm can be used to solve the desired task. This can potentially result in a more robust process for anomaly detection, as well as require less manual work by domain experts.

However, solving the objective requires not only a suitable machine learning algorithm for pattern detection, but also a preprocessing pipeline which provides means to extract appropriate features from the given dataset in order to serve as a rich training set for machine learning algorithms. This can be seen as a more general *data mining* task, including the very specific *machine learning* objective.

In section 2.2, *machine learning* will be introduced in detail as the science of deriving knowledge from data. *Data mining* integrates machine learning algorithms into an iterative process of creating valuable insights from available data sources. Hence, data mining is understood as the task of applying machine learning algorithms to a specific data-related problem and therefore includes handling and preprocessing of the specific dataset. Figure 2.1 shows the process of data mining defined for this thesis. The presented steps have been derived from the process

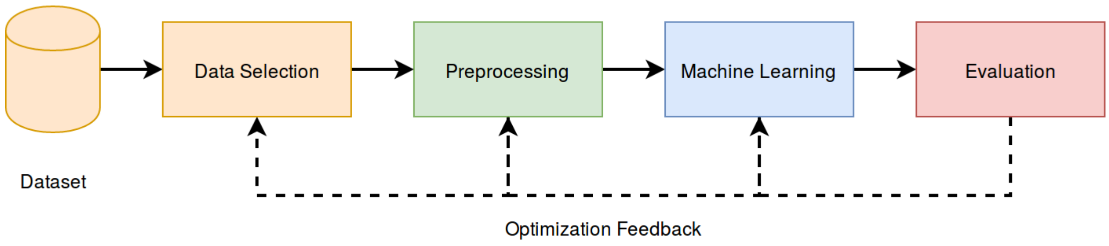


Figure 2.1: The *data mining* process used for the anomaly detection objective. The steps have been derived from the *KDD* process (Fayyad et al., 1996).

of *Knowledge Discovery in Databases* (KDD), as defined by Fayyad et al. (1996). In contrast to the KDD process, the focus here is less on creating business value that is based on new and value-adding information. Instead, the goal can be narrowed down to providing a suitable pipeline including preprocessing and a machine learning algorithm for anomaly detection for the specific industry application.

In summary, different approaches from recent publications will be considered for anomaly detection. Especially deep learning algorithms will be discussed, as deep learning provides reasonable means to handle data similar to the complexity of the given dataset in this thesis. Working on a real-world dataset introduces additional steps, such as transformation of raw data into appropriate input features for the machine learning algorithm. The goal is to extract meaningful features out of the raw dataset. This requires e.g. handling missing values in sensor channels or encoding different data types.

2.2 Machine Learning

Machine learning is the science of deriving knowledge from data and applying that knowledge to solve data-related tasks. Machine learning has been successfully applied to various objectives in a vast range of different domains. Machine learning applications can be generally divided into the following three different types of learning problems:

- Supervised machine learning
- Unsupervised machine learning
- Reinforcement learning

Supervised machine learning describes the task of learning from examples where the desired outcome is already known. For example, the classification of malicious emails can be implemented by learning a classifier on spam as well as non-spam emails. Once the classifier detects malicious emails from the given training data with a satisfying accuracy, the algorithm can be tested and validated on previously unseen examples, where the desired class is also known.

Unsupervised machine learning incorporates different tasks, where there is no given ground truth and the outcome is uncertain. This includes clustering of data sets as well as several different methods of dimensionality reduction. In these tasks of machine learning, the outcome cannot be validated by relying on previously recoded lables corresponding to the data samples.

Finally, *reinforcement learning* describes a more complex type of machine learning, where the algorithm has to react on a changing state of the given environment. The reactions are then measured by a previously defined target and thus a reward value is returned to the algorithm serving as a target for optimization.

2.2.1 Deep Learning

Deep learning describes a set of practices and algorithms for numerous architectures of deep neural networks, where the term *deep* refers to architectures consisting of multiple hidden layers. With these deep neural networks, the goal is often to derive hierarchal hidden representations of raw input data in order to solve a narrow task. As an example, in computer vision applications, *deep convolutional networks* are trained to detect different visual features from given images to categorize objects. This example shows the advantage of neural networks compared to traditional machine learning algorithms like *support vector machines* (SVM). In general, deep neural networks can learn latent features from raw data, whereas in case of traditional learning algorithms, the input features have to be carefully engineered which often requires extensive domain knowledge.

This practical advantage of deep learning algorithms offers high potential in use cases, where relevant input features cannot be manually defined due to lack of domain knowledge. In some cases, extracting features can also be too complicated to be encoded by an engineer. This can easily be applied to the task of object recognition. In this case, a human being can identify objects intuitively but cannot easily derive a complete set of rules for an algorithm to identify specific objects with invariance to scale, orientation, or the position in an image.

Simple machine learning algorithms have proven to be unsuccessful when it comes to solving tasks like object or speech recognition, which are considered as central problems in artificial intelligence (Goodfellow et al., 2016). Especially on data with a high dimensional input space, simple algorithms cannot generalize sufficiently due to the sheer amount of possible different configurations of the input data, which is often much larger than the available training samples.

Recent deep learning publications often reach state-of-the-art performance in many different tasks, which have been subject to active research within the last decades. For example, deep architectures of convolutional neural networks revolutionized the field of image recognition and ever since have been the first choice for the task of object classification with constantly achieving convenient results.

The recent achievements in deep learning applications result in major attention from media, which influences public expectations towards artificial intelligence. However, many successful applications of deep learning are limited to a very narrow task, whereas transfer of knowledge and incorporation of context remains a subject to be further explored to achieve actual progress towards general artificial intelligence.

2.2.2 Artificial Neural Networks

Artificial neural networks have been utilized for scientific applications since several decades. During the last years however, major advances in hardware for parallel computing and the availability of massive datasets have resulted in a significant increase in popularity of neural networks for many applications throughout various domains. Deep learning provides the necessary tools to effectively train deep architectures of neural networks on fast, distributed hardware (e.g. GPUs) in a timely manner. In addition, the availability of large datasets allows to fit models with millions of parameters. Given the architectural flexibility of neural networks, they can be tailored to specific needs and trained for many different applications.

In general, an artificial neural network can be seen as an algorithm for universal function approximation which can theoretically learn any continuous function[1] required by the objective to be learned on a specific dataset (Hornik, 1991). In this section, the baseline *feedforward* neural network as well as convolutional and recurrent networks will be discussed. These network architectures provide the baseline for many deep learning models and are therefore currently the most widely used architectures of neural networks.

[1]There are certain constraints on the continuous functions a neural network is able to learn which are out of scope for this overview and therefore omitted. For a complete discussion, see (Hornik, 1991).

Feedforward Neural Networks

The function modeled by a *feedforward neural network* maps a fixed size input vector $x \in \mathbb{R}^{d_x}$ to an output vector $y \in \mathbb{R}^{d_y}$, so that the learned function is defined by $f_\theta : X \to Y$, with the input space $X \subset \mathbb{R}^{d_x}$ and the output space $Y \subset \mathbb{R}^{d_y}$. During training, the scope is to find an appropriate set of parameters θ, so that the model approximates the mapping $y = f(x; \theta)$.

In general, a neural networks consists of multiple, fully connected layers of neurons, where each neuron by itself can be seen as a simple regression unit. Hence, a single neuron is defined as $o(x) = g(w^T x + b)$, where $x \in \mathbb{R}^{d_x}$ is the input vector that is multiplied by the weights $w \in \mathbb{R}^{d_x}$ and subsequently added to the *bias* term $b \in \mathbb{R}$. Afterwards, the sum serves as input for the non-linear transfer function g. Often, *sigmoid*, *tanh* or *Rectified Linear Unit* (ReLU) functions are used as non-linearity g. The latter, defined as $relu(x) = max\{0, x\}$ has been very popular recently, as rectifier units may improve the training of the model (Glorot et al., 2011) and can significantly speed up the time needed until the model converges[2].

A multi-layer neural network can be represented in the following recursive equation 2.1 for the layers $l = 1, ...L$, where L is the total number of layers in the network and the initial layer equals the input $h^{(0)} = x$, with $x \in \mathbb{R}^{d_x}$. The last layer yields the output vector $y = h^{(L)}$, where $y \in \mathbb{R}^{d_y}$.

$$h^{(l)} = g^{(l)}(h^{(l-1)} W^{(l)} + b^{(l)}) \tag{2.1}$$

In contrast to the formula for a single logistic regression unit, equation 2.1 uses a weight matrix $W \in \mathbb{R}^{d_{h^{(l-1)}} \times d_{h^{(l)}}}$ and a bias vector $b \in \mathbb{R}^{d_{h^{(l)}}}$, where the dimension $d_{h^{(l)}}$ is equivalent to the number of units in the intermediate layer l. Hence, layers are fully connected, whereas the units per layer can be computed in parallel. The non-linear function $g^{(l)}$ is then applied component-wise to calculate the activation of each unit in the layer l.

As the $sigmoid$ function saturates at 0 and 1, it can be used as activation function in the last layer of the network in order to solve classification tasks. More often, the $softmax$ function is used to output a normalized probability distribution $p(y|x)$ which satisfies $\sum_{c_i=1}^{c} p(c_i|x) = 1$ for all c classes.

Neural networks have two modes of operation. Through *forward-propagation*, the input can be processed to calculate the resulting output. However, for the neural network to approximate a function that generates the desired output, the optimal set of model parameters has to be

[2] The potential of *ReLU* activations to speed up training can be easily shown by experiments on simple classification tasks, e.g. on the MNIST dataset of handwritten digits.

estimated. This can be done through *gradient descent*. To obtain the gradients according to the defined error or *cost* function, the *back-propagation* mode of the model can be used (Rumelhart et al., 1988). For an arbitrary cost function $J(\theta)$, where θ are the parameters in the model, back-propagation aims to calculate the gradients w.r.t. to the model parameters $\nabla_\theta J(\theta)$, in order to calculate parameter updates by gradient descent. Through the chain rule of calculus, the partial derivatives w.r.t. the weight matrix W^l in layer l can be written as follows:

$$\frac{\partial J}{\partial W^l} = \frac{\partial J}{\partial h^L}\frac{\partial h^L}{\partial W^l} = \frac{\partial J}{\partial h^L}\frac{\partial h^L}{\partial h^{L-1}}\frac{\partial h^{L-1}}{\partial W^l} = ... = \frac{\partial J}{\partial h^L}\left[\prod_{k=l}^{L-1}\frac{\partial h^{k+1}}{\partial h^k}\right]\frac{\partial h^l}{\partial W^l} \tag{2.2}$$

As shown in equation 2.2, the parameter updates can be efficiently estimated by propagating the gradients backwards, starting at layer L. At each layer l, the partial derivatives w.r.t. the weight matrices W^l can by calculated by reusing the components from the previous steps. Updating the parameters according to the gradient descent procedure can be repeated until the cost function shows convergence.

Deep neural network architectures tend to overfit on the training data and therefore fail to generalize to data that has not been seen during training. This can be prevented by regularizing the network. A proven and efficient method for regularization is *dropout* (Srivastava et al., 2014), where during each training step the output of a random subset of units in the network is multiplied by 0 and thus effectively *deactivated*. This can be seen as a form of *bagging* mechanism with parameter sharing (Goodfellow et al., 2016), where ensembles of different architectures are combined to prevent overfitting. The neural network essentially learns not to rely on co-adaptions between distinct units, which perform well on the training data. This can significantly increase the generalization capabilities of neural networks in various different domains.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a special architecture of neural networks, that can effectively incorporate temporal dependencies within the input data. This can be achieved by unrolling a neural network on the temporal axis, where the network at each time step is provided with feedback connections from previous time steps. This can be implemented efficiently due to parameter sharing between the unrolled neural networks over all time steps. Additionally, RNNs are a reasonable approach for tasks that need to model sequences with different length, as the network can be dynamically unrolled according to the length of the input sequence.

A RNN with multiple layers can be formally described by extending equation 2.1 with temporal context:

$$h_t^{(l)} = g^{(l)}(h_t^{(l-1)}W^l + h_{t-1}^{(l)}U^{(l)} + b^{(l)}) \tag{2.3}$$

$$y_t = h_t^{(L)} = g^{(l)}(h_t^{(L-1)}W^{(L)} + b^{(L)}) \tag{2.4}$$

In this notation, the first layer is given by the input vector at each time step $h_t^{(0)} = x_t$. Further, the state at each layer is initialized with predefined values, so that $h_0^{(l)} = h_{init}^{(l)}$.

RNNs can be trained through gradient descent with *Backpropagation through Time* (BPTT) (Williams and Zipser, 1995). This allows a complex recurrent model to be trained end-to-end, similar to the baseline feedforward neural network architecture. Here, the intuition is that an unrolled RNN can be seen as a feedforward network with constraints on the weights, as the weight matrices in each unrolled network are shared over the time steps. Hence, the partial derivatives can be calculated for each time step and subsequently added up in order to update the weights.

Training a recurrent neural network with gradient descent requires backpropagating gradients through the entire architecture in order to calculate the partial derivative of the loss function with respect to each parameter of the model. As the chain rule is applied many times in backpropagation, the gradients flowing through the network can either become very large or very small. Due to the complex structure of RNNs, the architecture suffers from vanishing or exploding gradients during training through SGD (Hochreiter et al., 2001). In practice, large gradients can be avoided by clipping the gradient (Pascanu et al., 2013). Vanishing gradients however, remain a challenge of deep architectures and prevent the model from learning correlations between distant events. Hence, it is very difficult to model long-term dependencies within sequences with a large number of time steps.

Long Short-Term Memory Networks (LSTMs)

To solve this problem, gated RNNs have been introduced. The basic idea is to add paths through time, that cannot have exploding or vanishing gradients (Goodfellow et al., 2016). A gated architecture that was proven to be very efficient in practical applications is the *Long Short-Term Memory* (LSTM) introduced by Hochreiter and Schmidhuber (1997) and further enhanced by Gers et al. (2000). LSTM networks incorporate gating mechanisms to enable the model to decide whether to accumulate or forget certain information regarding the transferred cell state. This allows the network to operate at different timescales and therefore effectively model short as well as long-term dependencies. For instance, the model can store information on a pattern that is based on several different characteristics occurring in a relatively small time frame. Once the pattern is complete, the model can discard detailed information about the previously recorded characteristics and is therefore able to detect the next, similar type of pattern. This capability allows to efficiently solve several kinds of tasks that require sequence-modeling with temporal dependencies.

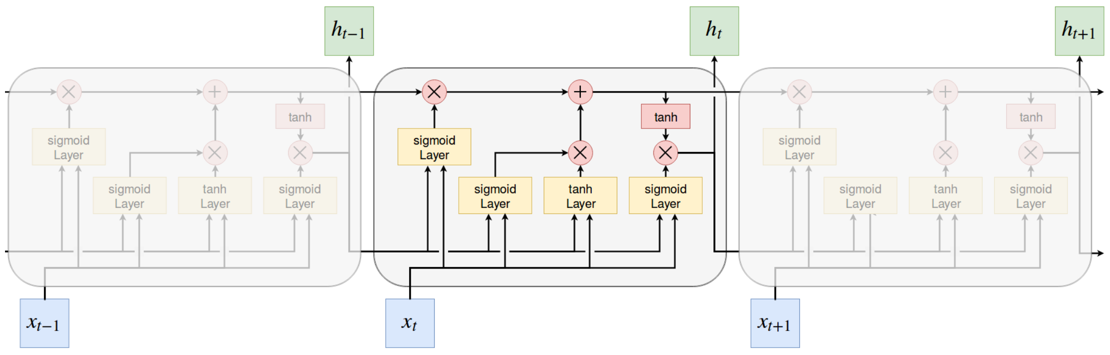


Figure 2.2: LSTM architecture with different gated connections to accumulate or forget temporal information according to the previous state as well as the current input.

Figure 2.2 shows the lstm *cells* per time step. Gating mechanisms are implemented through weighted connections with sigmoid activations, as the sigmoid function converges at 0 or 1 and can therefore be seen as a differentiable binary decision between *true* (1) and *false* (0). This allows e.g. the forget gate f to update the cell's state. As an example, the gate can force certain information in the state to be *forgotten* by setting it to zero.

Hence, adding a forget gate allows the model to discard useless information from the previous cell state by evaluating the information given by the input at the current time step t. The output of the forget gate f_t is calculated in the following manner, where W_* and U_* are weight matrices whose parameters are to be learned during the training of the model:

$$f_t = sigmoid(W_f * x_t + U_f * h_{t-1} + b_f) \tag{2.5}$$

The model can learn to accumulate certain information j_t from the current time step by taking into account the previous output. In a similar manner to the forget gate, the update gate i_t subsequently decides which information from the current time step will be added to the cell state.

$$i_t = sigmoid(W_i x_t + U_i h_{t-1} + b_i) \tag{2.6}$$

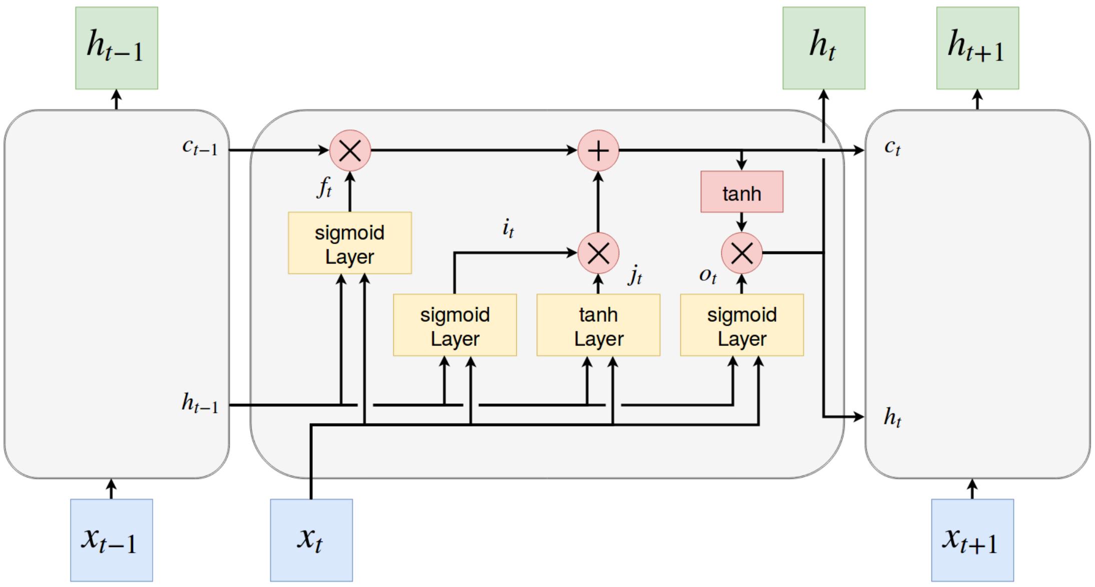


Figure 2.3: Detailed view on the gated connections in an LSTM unit at a single time step t.

$$j_t = tanh(W_j x_t + U_j h_{t-1} + b_j) \tag{2.7}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot j_t \tag{2.8}$$

Finally, the updated state c_t can be calculated out of the previous state c_{t-1}, the output of the forget gate f_t and the output of the update gate i_t as stated in equation 2.8. Here, the $\odot$ operation denotes the element-wise vector product. The output of the cell h_t at the current time step is subsequently calculated with the updated cell states c_t:

$$o_t = sigmoid(W_o x_t + U_o h_{t-1} + b_o) \tag{2.9}$$

$$h_t = o_t \odot tanh(c_t) \tag{2.10}$$

LSTMs proved to perform well in many recent publications and are rather easy to train. Therefore, LSTMs have become the baseline architecture for tasks, where sequential data with temporal information has to be processed. However, there are many extensions to this architecture, as the purpose of the individual components is disputed, and therefore, more optimal architectures may exist (Jozefowicz et al., 2015). For example, Cho et al. (2014) introduced *Gated Recurrent Units* (GRU), which have less parameters as LSTMs but show similar results in practice. Studies by Jozefowicz et al. (2015) compared different architectural modifications to LSTMs and GRUs but failed to find a model which performs consistently better than the baseline architectures.

Schuster and Paliwal (1997) used a bidirectional composition of LSTMs, where two identical layers are given the same input sequence, but one of the layers is working on the input in reversed order. Afterwards, the results of the two layers are merged, e.g. by calculating the sum of the values in each time step. This is especially useful, once the optimal output at a time step t_i requires information from time steps $t < t_i$ as well as $t > t_i$. Having access to information from the entire sequence at each time step, showed to work well in certain use cases, e.g. language modeling (Graves et al., 2013; Wu et al., 2016).

Convolutional Neural Networks

A *Convolutional neural network* (CNN) is an neural network architecture that is especially suited for 2-dimensional data structures, e.g. images. The core concept behind CNNs is to model the invariance of visual features to translation, rotation or even illumination. This allows to recognize certain objects, even once they are shifted or rotated upside down.

To achieve this, a weighted *kernel* $K \in \mathbb{R}^{d_{k,w} \times d_{k,h}}$[3] is moved over every possible position in the input image. This is computationally equivalent to a 2-dimensional *convolution* of the input image and the kernel. For an image I the convolution can be thus written as follows, where the asterisk $*$ denotes the convolution operation:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.11}$$

The resulting image S is referred to as *feature map*. The dimension of the feature map depends on the step size (*stride*) for shifting the kernel over the image. In order to retain the size of the original image, zero padding of the input image and a stride of 1 can be used. The weights of a kernel are estimated during the training of the model. Typically, a single convolutional layer contains k different kernels, that are applied to the same input image and therefore result in k different feature maps. This allows extracting feature maps, which are sensitive to certain visual features and invariant to their position in the image.

By stacking multiple convolutional layers, the model learns to extract hierarchical visual features. The first feature maps can e.g. detect edges in the original image, whereas the following feature maps detect patterns, composed of the previously extracted an more abstract features.

[3]Usually, width and height of a kernel are equal. Also, the smallest reasonable kernel would be of size 3×3, as this contains a center pixel with one pixel on every surrounding side.

Each convolutional layer is usually followed by a non-linear transfer function. In most applications, the architecture of CNNs includes *pooling* layers for downsampling of the intermediate feature maps. Alternatively, this can be achieved solely by strided convolutions as shown in the *all-convolutional* architecture proposed by Springenberg et al. (2015). A CNN for a classification task has multiple convolutional layers as feature extractors, followed by one or more dense layers for classification. Therefore, the last layer has either *sigmoid* activation in case of binary classification or a *softmax* activation for a multinomial classification task. Recently successful architectures show an increasing number of stacked convolutional layers. In general, deeper CNN architectures showed to improve the results but suffer from overfitting and vanishing gradients. The latter can be effectively prevented by incorporating *residual connections* (He et al., 2016) in order to create a passage through the architecture, where gradients can flow without vanishing. This characteristic has been broadly used in recent successful CNN architectures, e.g. in the *Inception-ResNet-v2* architecture introduced by Szegedy et al. (2017).

CNNs are very popular for pattern recognition in images but were also successfully applied to time series data (Ordóñez and Roggen, 2016; Rajpurkar et al., 2017). Due to the 2-dimensional structure of multivariate time series, research on pattern recognition in images can be partially applied to time series modeling as well. In general, sequence modeling in deep learning is often implemented using either deep convolutional networks or recurrent neural networks. However, multivariate time series do not require positional invariance of local feature on the non-time axis. Instead, the position might be crucial to the detection of patterns.

Even though RNNs are the more intuitive choice for time series, CNNs have the advantage of completely parallel computation. In contrast, RNNs need to be calculated step-wise, as each step is constrained on the previous time step.

The following section 2.3 will discuss recent publications on both sequence modeling in general and more specifically anomaly detection.

2.3 Deep Learning for Anomaly Detection

The task of anomaly detection has been subject to several recent deep learning publications. Apart from different conceptional approaches, these publications also show significant differences in architectural considerations to detect anomalies.

In general, the goal of machine learning approaches for anomaly detection is to model the distribution of normal data. This allows distinguishing anomalous patterns from what is expected based on the available normal data. For multivariate sensor data, this can be achieved e.g. by learning a multivariate gaussian distribution $\mathcal{N}(\mu, \Sigma)$ that includes covariance statistics of the sensor channels from the training dataset. The distribution of the normal data can be estimated by calculating μ and Σ based on the available data $x \in \mathbb{R}^m$ per time step, where m is given by the number of sensor channels.

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \tag{2.12}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)(x^{(i)} - \mu)^T \tag{2.13}$$

The probability of a sample per time step belonging to the normal distribution is then given by $p(x)$ in equation 2.14. In practice, known anomalies as well as normal data which was not used for training can now be utilized to estimate a threshold parameter ϵ, so that $p(x) > \epsilon$ can be used to predict anomalies in new data. This parameter can be cross-validated in order to achieve a sufficient accuracy for anomaly detection.

$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x^{(i)} - \mu)^T - \Sigma^{-1}(x^{(i)} - \mu)\right) \tag{2.14}$$

A more sophisticated approach to unsupervised anomaly detection is to use *One-Class Support Vector Machines* (OC-SVM), introduced by Schölkopf et al. (2000). OC-SVMs learn a hypercube from the distribution of the training data. This allows categorizing novel samples according to their distance from the hypercube. OC-SVMs have been broadly used for anomaly or novelty detection tasks and can therefore be seen as a good baseline method for comparison.

Simple multivariate gaussian distributions and the more complex OC-SVMs can model covariance between values in different sensor channels. However, these approaches fail to model temporal dependencies between the channel-wise values at different time steps. This can significantly reduce the potential to detect anomalies in a multimodal setting. For instance, washing programs follow a basic routine, which can be further split into subroutines. Normal patterns within a certain subroutine might be considered anomalous in a different subroutine. These temporal dependencies would have to be encoded in features which introduces additional manual effort that requires good knowledge of the data domain.

This is where recently published deep learning approaches show their full potential. In the remainder of this section, different approaches will be discussed. At first, classification models for time series data will be discussed in 2.3.1, as a method for detecting patterns connected to known failures, given the precondition that sufficient examples of these failure cases are available for training and validation.

As anomaly detection scenarios usually suffer from a lack of an sufficient amount of labeled examples to train a simple classification algorithm, the task is narrowed down to modeling normal data from the available dataset. For these tasks, forecasting models (2.3.2), autoencoder models (2.3.3) as well as generative models with adversarial training (2.3.4) will be discussed.

2.3.1 Classification Models

Classification is a straightforward approach for pattern recognition in time series data. Given a sufficient amount of anomalous samples in a dataset, the data can be labeled as belonging to either the normal or the anomalous class. In practice, anomaly detection suffers from an extremely unbalanced dataset, where only a few labeled anomalies are available. Nevertheless, this section will present recent applications of classification models for time series data. This allows discussing architectures of neural networks that can efficiently model multivariate data with temporal dependencies.

In general, the first step to classification is to identify relevant features which can be fed into a classifiers. This can be challenging in case of multivariate time series data due to its spaciotemporal nature. In practice, the multivariate sequences can also differ in length, which introduces another computational challenge. A reasonable approach to handle these differences is to use *dynamic time warping* (DTW) to calculate a measure of distance between two sequences with different length (Leng et al., 2008). This measurement of similarity can subsequently be used for a simple *nearest neighbor* approach for categorizing normal as well as anomalous sequences. However, this method does not scale well, as each classification step has to iterate over the complete set of labeled examples or make use of some sort of indexing mechanism.

A recent study by Ordóñez and Roggen (2016) utilized an neural network architecture combining convolutional and recurrent layers to identify and extract relevant features from multivariate sensor data. The intuition here is, that CNNs have proven to excel at extracting features from grid-like input structures. Further, RNNs show good results in handling temporal features. Figure 2.4 shows the *Convolutional LSTM* architecture, where the first convolutional layers extract features along the time axis and fed them into a recurrent layer. It is notable, that instead of adding a *poolig* layer after each convolution layer, which is usually used in image recognition tasks for invariance to small shifts in position and orientation, this architecture aims to preserve all sequential information from the input sequence. Finally, a dense output layer is used to predict the labels.

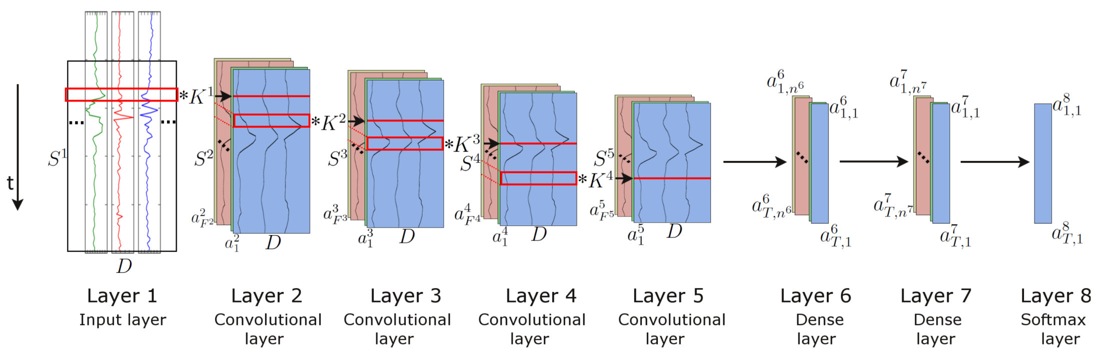


Figure 2.4: Architecture of multiple convolutional layers followed by recurrent or dense layers for classification of multivariate sensor data (Ordóñez and Roggen, 2016).

This architecture was used for human activity recognition in multivariate sensor data. A sliding window function is applied to the normalized data to extract fixed size and overlapping sequences. These sequences where labeled according to the last occurring activity in the data. This allowed for a supervised training on examples from multiple classes. Compared to traditional machine learning algorithms, the convolutional LSTM architecture achieved significantly better results, even though the classifier was trained on raw input sequences without any further feature engineering. This shows the potential of a deep neural network for the task of time series classification.

Another even more recent application for deep convolutional neural networks was shown by Rajpurkar et al. (2017). In this case, the task was to identify arrhythmias in *ECG* signals. Therefore a large dataset of signals from 30.000 patients was recorded and labeled by cardiologists. This allowed to train a classifier, consisting of 34 stacked convolutional layers. These layers were segmented into 16 blocks with residual connections with 2 convolutional layers each. The residual blocks also employed *batch normalization* (Ioffe and Szegedy, 2015) in order to make the training of such a deep architecture possible in a timely manner by allowing higher learning rates and more tolerance towards parameter initialization. The model does not predict a single label, but instead predicts one label for every second in the input sequence[4]. Hence, this can be seen as a sequence-to-sequence task, instead of a simple classification.

The architecture outperforms cardiologists in the task of arrhythmia detection and thus is a good example of how deep learning can augment medical procedures. Further, the deep CNN

[4]The input sequences in (Rajpurkar et al., 2017) contained 30 seconds of ECG signals sampled at 200 Hz. Thus the resulting input sequences have a total sequence length of 6000 data points.

presented in this paper shows reasonable architectural considerations, so the model can be applied to time series data with long sequences.

2.3.2 Forecasting Models

Classification models predict one or more nominal labels for a given input example, in this case a multidimensional sequence from several sensor channels. Instead, a machine learning model can also be trained to predict one or more continuous values, e.g. forecasting the next values for a given input sequence. Such a forecasting model can also be used for anomaly detection.

To achieve this, the model is trained solely on recorded normal data with the scope of forecasting one or more time steps, based on a fixed-length sequence of preceding steps. Once sufficiently trained, the model can be utilized to detect anomalies by comparing the forecast at each time step with the actual sequence recorded by sensors. As soon as the forecast error exceeds a predefined threshold value, a data point can be labeled as anomalous.

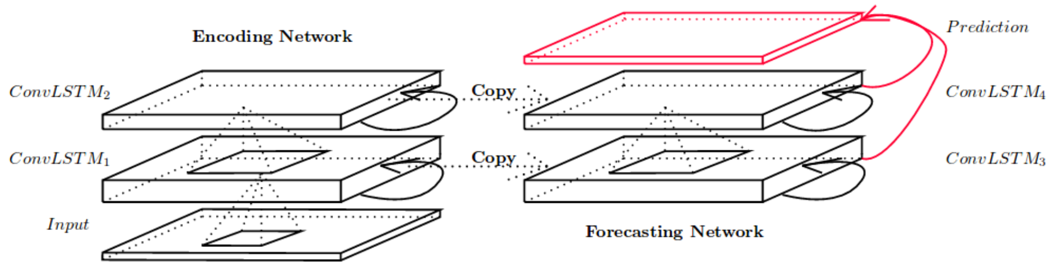


Figure 2.5: Architecture for short-term weather forecasting through stacked layers of convolutional LSTM networks in an encoder-forecasting structure (Xingjian et al., 2015).

An example from recent literature is the deep learning approach for precipitation nowcasting, proposed by Xingjian et al. (2015). The authors introduce convolutional operations in the calculations for input-to-state and state-to-state transitions of LSTM cells. This allows extending the LSTM architecture, so input, output and state are represented as 3-dimensional tensors. Precipitation nowcasting is formulated as a spaciotemporal sequence forecasting problem, which the authors approach by implementing a sequence-to-sequence model based on the introduced convolutional LSTMs. Figure 2.5 shows the architecture consisting of an encoding- and forecasting network. Each network contains multiple stacked ConvLSTMs, where the states of the forecasting network are initialized by copying the last states of the encoding network.

While this forecasting problem is significantly different from anomaly detection, it still allows drawing conclusions for sequence forecasting in general. Xingjian et al. (2015) show how to efficiently model spaciotemporal dependencies in sequences by incorporating convolutions in a LSTM encoder-forecasting architecture.

2.3.3 Autoencoder Models

The goal of autoencoder models is to learn a latent representation of the training data in a unsupervised manner. In general, autoencoders consist of an encoder and a decoder network. The encoder takes the original input and extracts a fixed-sized representation, which is usually of much smaller dimensionality than the input. The latent representation further serves as input to the decoder network, which is trained to reconstruct the original input. Hence, the autoencoder learns to extract meaningful information that sufficiently explains the characteristics of the data, which is similar to dimensionality reduction techniques like *Principal Component Analysis* (PCA).

Autoencoders for Anomaly Detection

In a similar manner to the forecasting approach in 2.3.2, *sequence-to-sequence* architectures can be utilized for anomaly detection by learning to reconstruct a given input sequence. The reconstructed sequences can subsequently be compared to the original input. The model is trained exclusively on normal data. Once the input sequence contains a patter, that is different from the normal data, the model is not able to reconstruct the sequence with equal quality compared to the reconstruction of normal data. Hence, an anomaly score can be calculated comparing the reconstruction with the original input.

However, the assumption that an autoencoder model trained on normal data is not able to sufficiently reconstruct anomalous data needs to be evaluated empirically. Even though comparable work proved that this approach can be successful in practice (Malhotra et al., 2016), this might not be true for data from a different domain.

Nevertheless, this approach is more intuitive compared to forecasting. A given input sequence is processes and an anomaly score is calculated for each point of the (multidimensional) sequence. This is comparable to the manual procedure of an engineer labeling parts of a subsequence as anomalous. It is also possible to operate in both directions on the temporal axis. In order to decide whether a single data point shows unexpected behavior, the temporal context of both, previous as well as succeeding time steps may be taken into account.

In general, autoencoders for sequences with temporal dependencies are implemented as *sequence-to-sequence* models. Sequence-to-sequence models consists of an encoder and a decoder network which can be trained end-to-end (Sutskever et al., 2014). The encoder extracts a latent representation of the input sequence and subsequently feeds it into the decoder. The decoder then outputs a target sequence based on the latent representation. Figure 2.6 shows a simple model for translating an input sentence into a target sentence.

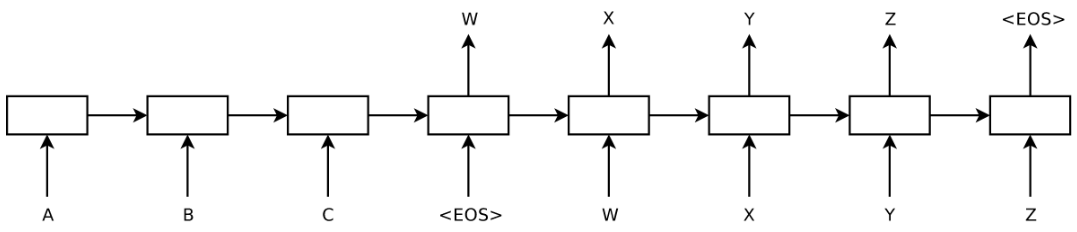


Figure 2.6: This sequence-to-sequence model takes the input sentence ABC and translates it into the target $WXYZ$ (Sutskever et al., 2014). The symbol $\langle EOS \rangle$ denotes the end of the sentence. For the end of a sentence and decoder feedback.

Sequence-to-sequence models can be implemented using RNNs for both, encoder and decoder network. In this case, the hidden state of the RNN at the last time step of the encoder network is the extracted latent representation of the input sequence. Therefore, the dimensionality of the representation is given by the number of hidden units in the RNN. Figure 2.6 also shows how to handle different lengths of the input and output sequence by incorporating an *end-of-sentence* (EOS) token. This is applicable for translation tasks but not necessary in case of autoencoders, where the input sequence and the reconstruction are of the same length. However, using an RNN for the encoder and decoder allows to dynamically adjust the network for different sequence lengths of the input during training and inference.

The sequence-to-sequence model for translation showed that multiple stacked LSTM layers outperform a single LSTM. In addition, the architecture proved to be easily extended into a deeper model by adding layers to the encoder and decoder network. Given sufficient hardware capabilities, adding more depth through stacked layers might be beneficial for domains, that require a more complex architecture to model the underlying data distributions. However, this might introduce the need for regularization to prevent overfitting or require residual connections to enable the deep architecture to learn properly.

Another finding by Sutskever et al. (2014) is that reversing the input sequence yields better results for translation tasks. This is attributed to the fact, that the paths through the model between the corresponding elements in the input and output sequence are closer to one another.

Encoder-Decoder for Anomaly Detection

Even though the previously mentioned sequence-to-sequence model focuses on a language translation task, this approach can be directly translated to anomaly detection in sensor data. Malhotra et al. (2016) used a similar model for anomaly detection in multivariate sensor data from a real-world engine dataset as well as other datasets from literature.

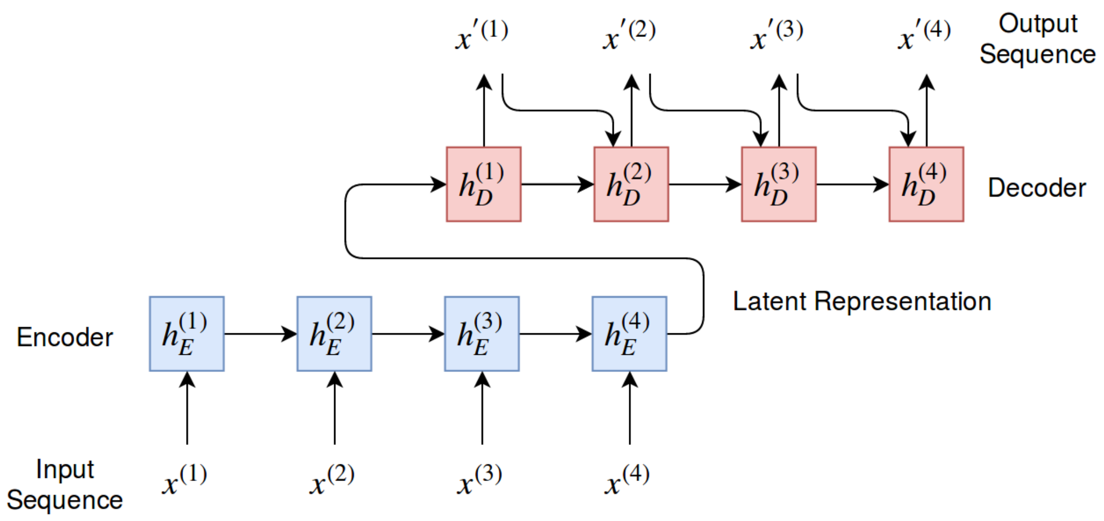


Figure 2.7: Sequence-to-sequence model consisting of a decoder on top of an encoder, where both are implemented as recurrent neural networks. The encoder takes the sequence x as input and calculates hidden states h_E for every time step. The resulting hidden representation of x is fed into the decoder. During inference, each decoder step receives the previous decoder state $h_D^{(i-1)}$ as well as the previous reconstructed output $x'^{(i-1)}$ to calculate $h_D^{(i)}$ and $x'^{(i)}$ respectively.

Malhotra et al. (2016) used a single LSTM layer for the encoder as well as the decoder network. Figure 2.7 shows the autoencoder architecture for anomaly detection. It is notable, that the previous reconstruction step is fed into the succeeding step. However, during training, the actual ground truth is fed into the decoder steps instead of the decoders predictions. This method, called *teacher forcing*, was introduced by Williams and Zipser (1989). It potentially speeds up the training as the decoder can instantly rely on its previous prediction, whereas otherwise the decoder can only start to effectively incorporate previous predictions during later stages of the training, once the predictions are reasonably accurate.

The authors applied a sliding window over the sequences to extracted fixed-length inputs for the encoder network. The autoencoder was trained on normal data only, whereas the validation for parameter tuning as well as the final testing was done on separated datasets containing normal and anomalous examples. Hence, the model was trained unsupervised to minimize the reconstruction error. Validation and testing was then done in a supervised manner to obtain the quality of the model. To achieve this, an anomaly score function was applied to the reconstructed sequences. This score was calculated based on the normal distributions of reconstruction errors on the normal data from the validation set.

Experiments with different hyper-parameters show that the autoencoder architecture may work for different sequence lengths, as the authors configured the sliding window for a length between 30 and 500 steps. In addition, the size of the hidden representation was optimized to values between 40 and 90 for the different datasets. However, the best configuration of window length and size of the hidden representation does not necessarily correlate but is rather subject to experiments on the specific dataset. Even though the autoencoder is able to handle multivariate inputs from an architectural point of view, the authors decided to reduce multivariate sensor data to a single dimension by extracting the first *principal component* through PCA.

The approach by Malhotra et al. (2016) proves, that a LSTM-based autoencoder is able to detect anomalies in sensor data from different datasets. Most importantly, the experiments on a real-world engine dataset are similar to the task of this thesis and therefore provide valuable insights. However, it has to be evaluated if reducing the multivariate sensor data through PCA is applicable for anomaly detection in this context, as the reduction potentially masks the most crucial characteristics of anomalies.

Advances from Machine Translation

In contrast to anomaly detection, machine translation is currently a more popular research topic and therefore it is subject to many recent publications. Fortunately, latest publications for machine translations are mostly based on sequence-to-sequence models (Wu et al., 2016; Kalchbrenner et al., 2016; Britz et al., 2017), which makes the advances in machine translations interesting for this thesis despite the different domain.

For instance, Wu et al. (2016) used a sequence-to-sequence model with multilayer LSTM networks for both, encoder and decoder as core algorithm for *Googles* machine translation system. The authors demonstrate, how to train and efficiently distribute a deep sequence-to-sequence architecture for a large dataset. Figure 2.8 shows the architectural layout of the encoder as well as the decoder network used for the translation task.

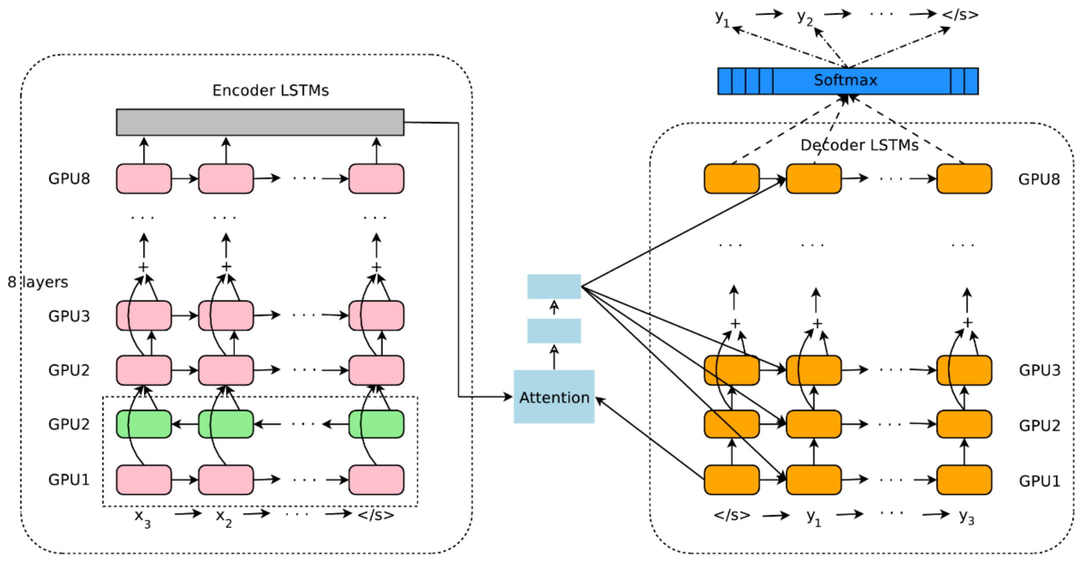


Figure 2.8: *Googles* neural machine translation architecture with deep encoder and decoder networks distributed to multiple GPUs and trained using residual connections in between the encoder and decoder layers. Further, an attention network is used between encoder and the decoder Wu et al. (2016).

To translate sentences, the containing words are mapped to an embedding space in order to gain a richer and more dense representation. Hence, the structure of the input sequence representing a sentence matches the length of the sentence and has a fixed-size embedding dimensionality per step. For variable sentence length, the encoder and decoder network can be dynamically unrolled on the temporal axis, as the LSTM cells share parameters through a specific layer. This is especially important to a translation task due to the possible differences in length between a source sentence and its translation. Given the word embeddings, the input structure is comparable to multivariate time series, which allows transferring some of the findings of this approach to the anomaly detection task of this thesis. Dynamically adjusting the network to the sequence length can also be applied for variable length sensor data. However, the reconstruction of the autoencoder will always be of the same length as the input sequence.

The encoder as well as the decoder consists of 8 LSTM layers. The fist layer of the encoder is implemented as a bidirectional RNN (Schuster and Paliwal, 1997), which scans the input sequence from both sides. Graves et al. (2013) found that a bidirectional RNN may improve the network's quality as it provides the network with context from across the entire input sequence.

From the second encoder and decoder layer onwards, residual connections (He et al., 2016) are added between the layers. These proved to enable training of very deep architectures using gradient descent, which is otherwise difficult due to the problem of vanishing gradients, described by Hochreiter et al. (2001).

The sequence-to-sequence architecture is further enhanced by an *attention* network. The attention mechanism enables the decoder to selectively focus on parts of the source sentence during translation (Luong et al., 2015). This proves to be very effective for language translation tasks and has further led to recent publications exploiting the capabilities of attention mechanisms (Vaswani et al., 2017).

Extensive experiments on similar sequence-to-sequence models for translation by Britz et al. (2017) show that good results can be achieved by a bidirectional encoder with up to 2 additional unidirectional LSTM layers, combined with a 4-layer decoder. Deeper networks with more layers needed to be extended with residual connections to be trainable. In addition, the experiments proved LSTM cells to consistently outperform GRU-cells (Cho et al., 2014).

In summary, the sequence-to-sequence model for machine translation provides a good example for architectural enhancements to LSTM-based encoder and decoder networks. It also demonstrates the potential of distributed training of a deep model on a large dataset.

In contrast to the LSTM-based translation model, other models use deep convolutional encoder and decoder networks for the same task. This includes the *ByteNet* architecture, proposed by Kalchbrenner et al. (2016).

ByteNet is based on the findings of Oord et al. (2016a), who proposed *WaveNet* for spoken language synthesis on raw wave form audio. The authors train the model to predict an audio sample based on a fixed-size set of previously recoded samples. *WaveNet* uses masked convolutions (Oord et al., 2016b) to ensure that each step in the output is only constrained by the previous steps of the input sequence. In addition, the receptive field of convolutional filters is extended by using dilated convolutions. Thus, a filter is applied to a larger area than the original filter size by skipping values in the input. By incrementing the dilation factor with every stacked convolutional layer, the receptive field of the output sequence can be extended to the entire input sequence with only a limited amount of stacked layers. The following figure 2.9 visualizes the effect of dilated convolutions with stacked layers.

The resulting architecture can process sequences in a similar fashion to RNNs but remains computable in parallel due to the masked convolutions. This potentially speeds up the computa-

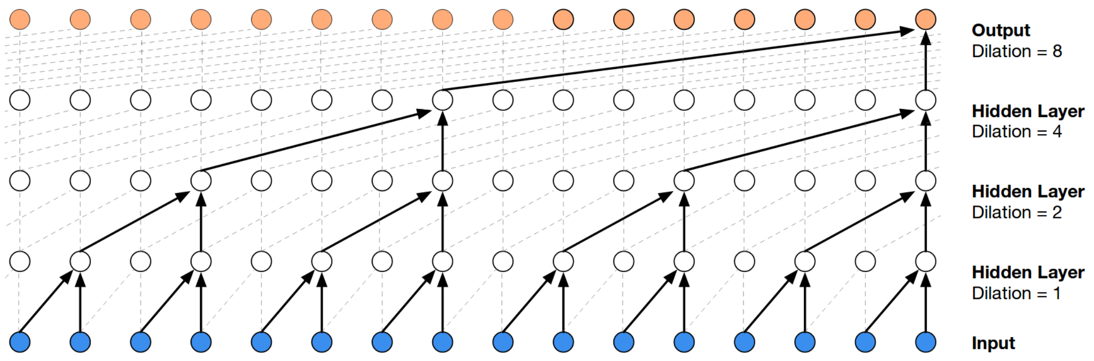


Figure 2.9: Stacked layers with dilated convolutions to increase the receptive field of each step in the output sequence in *WaveNet* (Oord et al., 2016a). Through masking out time steps in the future, each output step is only computed by the previous steps in the input sequence.

tion, which results in the capability of *ByteNet* to translate sequences with different length in linear time (Kalchbrenner et al., 2016).

In contrast to the previously mentioned approaches for sequence-to-sequence modeling, the decoder network in *ByteNet* is directly stacked on top of the representation extracted by the last encoder layer instead of a compressed representation (Malhotra et al., 2016) or an attention mechanism (Wu et al., 2016). However, for *ByteNet* Kalchbrenner et al. (2016) also experimented with replacing the convolutional encoder or decoder with an RNN respectively.

Recently, a *Wavenet*-style autoencoder was successfully utilized by Engel et al. (2017) for audio synthesis on the *NSynth*[5] dataset. This approach can be generalized for multivariate sensor data an trained on an anomaly detection task in a similar fashion to the LSTM-based autoencoder used by Malhotra et al. (2016).

Variational Autoencoders for Anomaly Detection

Variational Autoencoders (VAEs) (Kingma and Welling, 2013) are a special form of autoencoder, where the latent representation z is represented by stochastic variables. The assumption is, that the recorded data origins from a random process which is constrained by the random variable z. However, z cannot be observed directly in the data. Hence, the encoder network learns $q(z|x)$ to approximate the otherwise intractable posterior $p(z|x)$, whereas the decoder network learns

[5]Available at: https://magenta.tensorflow.org/nsynth

$p(x|z)$ and thus can be used to generate a sample based on a given z. VAEs can be trained by maximizing the variational lower bound $\mathcal{L}(q)$ for a given x (Goodfellow et al., 2016):

$$\mathcal{L}(q) = \mathbb{E}_{z\sim q(z|x)} \log p_{\text{model}}(x|z) - D_{KL}(q(z|x)||p_{\text{model}}(z)) \leq \log p_{\text{model}}(x) \tag{2.15}$$

Equation 2.15 defines the variational lower bound as the difference between the reconstruction log-likelihood and the *Kullback-Leibler* (KL) divergence of the approximate posterior distribution $q(z|x)$ and the model prior $p_{\text{model}}(z)$ Goodfellow et al. (2016). Thus, maximizing $\mathcal{L}(q)$ aims to maximize the reconstruction log-likelihood while reducing the difference between $q(z|x)$ and $p_{\text{model}}(z)$. As the encoder and the decoder are neural networks, VAEs can be optimized through *Stochastic Gradient Descent* (SGD).

Sölch et al. (2016) successfully applied a VAE consisting of *Stochastic Recurrent Networks* (STORN) (Bayer and Osendorfer, 2014) to an anomaly detection problem. The previously described LSTM-based autoencoder model presented by Malhotra et al. (2016) essentially compresses an input x into an unknown latent code. In contrast, the encoder network of the VAE maps x to a related stochastic variable z, which makes it easy to randomly sample z and generate an example x. This cannot be done with the decoder network of the LSTM-based autoencoder, as the latent code is not known.

The authors trained the VAE on multivariate sensor data from a robot arm conducting a specific task. The data contained seven sensor channels recoding the arms joint configurations. The VAE was trained solely on normal data, whereas validation and testing was done with normal and anomalous sequences that where produced by manually altering the robots behavior.

For anomaly detection, the STORN-based VAE outputs for each time step the variational lower bound and the prediction of the distribution at the next time step. These are used to calculate thresholds for anomaly detection.

Overall, Sölch et al. (2016) found that the architecture performed well in both, off-line as well as on-line anomaly detection. The latter is especially useful to detect ongoing anomalies in a timely manner. Furthermore, the VAE does not require domain knowledge and thus can be easily applied to a different data domain.

2.3.4 Generative Adversarial Networks

A *Generative Adversarial Network* (GAN) (Goodfellow et al., 2014) is a generative model that can be trained to generate samples based on observations from the training data. In contrast to other generative models, GANs contain two competing neural networks and thus introducing a practice from game theory into unsupervised learning.

The generator network G is trained to learn the function $G : Z \rightarrow X$, hence the projection from the latent space Z to the original data space X of the training data. The generator inputs $z \in Z$ are essentially uniformly distributed noise, which allows to sample from Z in order to generate a new image $G(z)$. In contrast, the discriminator network D is trained to distinguish between original samples from X and generated samples $G(z)$. The function learned is therefore $D : X \rightarrow [0, 1]$, so that the discriminator estimates the probability of the input sample belonging to the real data distribution, given by the training data. Both the generator G and the discriminator D are optimized during training through the *minimax* game $\min_G \max_D v(D, G)$, which is defined by (Goodfellow et al., 2014) as follows:

$$\min_G \max_D v(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(x)}[\log (1 - D(G(z)))] \tag{2.16}$$

During training, the generator G tries to fool the discriminator D into classifying the generated samples as real examples, whereas the discriminator tries to categorize correctly. In theory, continuous adversarial training of both networks can lead to the generation of high-quality samples, that are indistinguishable from samples drawn from the original data distribution.

GANs have recently been very popular and were therefore applied to several tasks in different domains. This includes tasks such as image generation, image-to-image translation (Kim et al., 2017; Liu et al., 2017) or the enhancement of the resolution of images (Ledig et al., 2016). Many publications also propose techniques for a more stable training of GANs (Metz et al., 2016; Arjovsky et al., 2017; Gulrajani et al., 2017). However, it still remains a difficult task to train GANs, as the generator might e.g. learn to always produce a single sample which effectively fools the discriminator.

In general, GANs hold the potential to learn the underlying features of data from an arbitrary domain in an unsupervised fashion, which is useful for a wide range of tasks. Hence, GANs can also be applied to detect anomalous patterns in multivariate data settings. To achieve this, the generator model can be trained to produce new samples which resemble normal data, seen during training. These generated samples can subsequently be compared to newly recorded sequences in order to detect previously unseen patterns.

Schlegl et al. (2017) successfully applied a GAN for the detection of anomalous visual features in medical imaging data to find indications of diseases. This allows detecting markers for diseases, which would be otherwise only detected by an image classification model trained on a sufficient amount of labeled data in a supervised manner.

The GAN is trained to generate data based on images capturing healthy anatomical characteristics. Known disease markers are used for validation of the model. For the detection of anomalous patterns in the images, the discriminator prediction is combined with a comparison to visually similar samples generated by the model.

In order to find a similar generated sample, Schlegl et al. (2017) proposed a mapping function $\mu(x) : x \rightarrow z$ from the query image x to the latent variable z. This is done by combining the discriminator loss with a residual loss which measures the visual similarity between the query image x and a generated sample $G(z)$. This allows to sample z to find a generated image $G(z_\gamma)$ that is most similar to x and is also classified by the discriminator to be from the normal data distribution.

In order to generate a similar example, the authors proposed to draw a random sample z_1 from Z and compare $G(z_1)$ with the query image x. The latent variable z is then optimized trough backpropagation of the discriminator loss and residual loss. This process is repeated for γ steps, so that the final generated sample is $G(z_\gamma)$.

In addition, Schlegl et al. (2017) used *feature matching* as an alternative to the discriminator loss, which was proposed by Salimans et al. (2016) to increase the stability of a GAN during training. Instead of training the generator on the discriminator loss, the generator is trained to generate examples that match the statistics of the normal data. This can be done by calculating the loss on the features in an intermediate layer of the discriminator. For the anomaly detection task however, this technique was not used during training, but for the discriminator loss during the process of updating z to search for a similar generated sample to the current query image.

In a similar fashion, the discriminator loss and the residual loss were used to calculate an anomaly score. The following equation 2.17 shows the score function $A(x)$, where the *residual score* $R(x)$ is given by the residual loss between x and $G(z_\gamma)$. The *discriminator score* $D(x)$ is defined by the discriminator loss using the feature matching method.

$$A(x) = (1 - \lambda) * R(x) + \lambda * D(x) \tag{2.17}$$

In addition to the identification of anomalous patterns in images through evaluating the anomaly score, the query image x can be directly compared to the most similar generated sample $G(z_\gamma)$ in order to find and interpret the anomalous regions in the image.

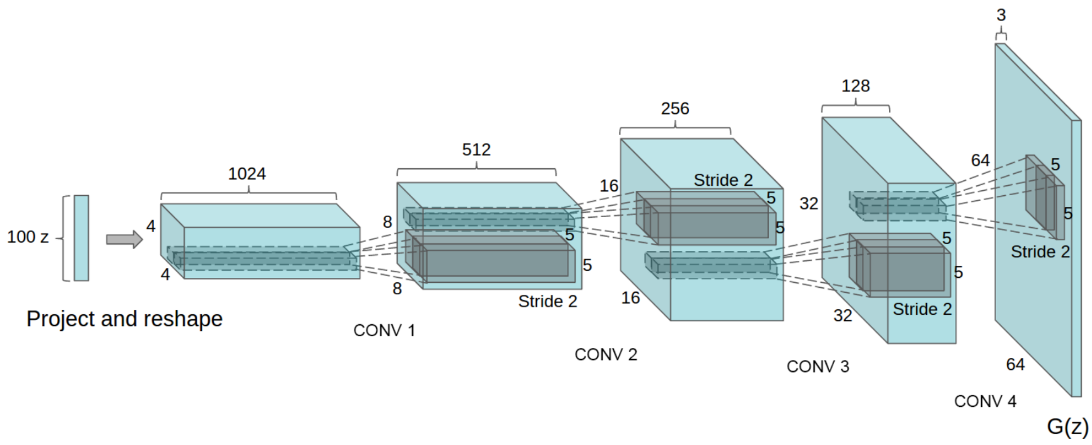


Figure 2.10: Generator network of the DC-GAN architecture proposed by Radford et al. (2016). The noise input z is projected into images, that increase in size in each convolutional layer, whereas the number of feature maps is reduced. This results in a single output image of the original shape from the training data.

Schlegl et al. (2017) utilize the *Deep Convolutional GAN* (DC-GAN) architecture (Radford et al., 2016) for the generator and discriminator model. DC-GAN incorporates important findings from recent literature on adversarial models and extensive experiments on different architectures conducted by the authors. The model does not use any spacial pooling functions but rather utilizes strided convolutions to achieve an all convolutional net (Springenberg et al., 2015). Hence, upsampling is learned by the network itself during training. Secondly, no fully connected layers are used except for the first generator and the last discriminator layer. This benefits the time it takes to train the model until it converges. In addition, batch normalization (Ioffe and Szegedy, 2015) is used in all generator and discriminator layers except for the output of the discriminator and the input of the generator. The authors also propose to use ReLU activations in the generator and LeakyReLU activations (Maas et al., 2013) in the discriminator.

Schlegl et al. (2017) found that the DC-GAN architecture was able to sufficiently learn the data distributions of healthy images from the anomaly detection task. Additionally, combining the discriminator prediction with the mapping from the image space to the latent space showed promising results for the detection of anomalous pattern.

The previously described approach to anomaly detection with GANs solely used normal data during the training. However, as the discriminator is trained in a supervised manner on real and generated examples, it is easy to incorporate labeled anomaly instances for training the discriminator model. Hence, the generator can be trained unsupervised to learn the distribution of normal data, whereas the discriminator can be trained with fake and real data including anomalies. This allows to efficiently utilize available knowledge about wanted and unwanted behavior in the domain of the specific anomaly detection problem.

This principle is comparable to the extension of OC-SVMs proposed by Görnitz et al. (2013) to train the algorithm on normal as well as anomalous examples which eventually allows for supervised training.

2.4 Conclusion

This chapter explored recent deep learning approaches to the task of anomaly detection. Malhotra et al. (2016) presented an autoencoder architecture based on LSTM networks for the application on multivariate sensor data from real-world engine data. This approach is most similar to the task of this thesis in terms of dataset and objective and can therefore be seen as a good starting point to conduct experiments. In addition, recent literature on sequence-to-sequence models present multiple architectural extensions which can potentially improve the anomaly detection capability. This includes the distributed training of very deep LSTM-based encoder-decoder architectures (e.g. GNMT) as well as utilizing convolutional networks for sequence modeling in parallel and therefore linear time (WaveNet).

The variational autoencoder presented by Sölch et al. (2016) and especially the GAN-approach to anomaly detection by Schlegl et al. (2017) offer more complex approaches but hold the potential to improve the anomaly detection capability. Especially a GAN trained for generating normal sequences can add valuable insights to the specific data domain.

The following chapter presents experiments with different approaches, that have been conducted based on the review of recent literature on similar tasks.

3 Architecture and Experiments

This chapter describes the selection and implementation of a suitable deep learning pipeline for anomaly detection that is tailored to sensor data recorded from washer-dryer machines.

First, section 3.1 explores and characterizes the real-world dataset. This includes elaborating the challenges to preprocessing of the dataset with the scope of utilizing the data for the training of a deep learning architecture for anomaly detection. The necessary preprocessing steps to achieve this are then described in more detail in section 3.2.

Based on the review of recent deep learning architectures in the previous section 2.3, an autoencoder model as well as a GAN have been implemented for experiments on this task. The architecture selection is presented in section 3.3 followed by a detailed description of the experiments in section 3.4. Finally, section 3.5 discusses results of the experiments.

3.1 Dataset

Proactive detection of possible failures in sensor data is crucial for industry applications as mechanical and electrical machines are increasingly complicated and difficult to maintain. Hence, applying recent developments in deep learning for anomaly detection to real-world sensor data is an interesting challenge. This thesis uses a real-world dataset that contains raw multivariate sensor data. The data was recorded during monitoring and testing of washer-dryer machines in the research and development phase of new machines.

The dataset consists of several sequences from long-term tests of different washer-dryer models. Table 3.1 shows the available test runs, where each run contains a long-term test of a single machine. In these long-term test scenarios, several examples of a newly designed model are monitored over the machines expected live span. Such a long-term test thus consists of thousands of repeated washing and drying cycles taking each between two and five hours. The machines tested are mostly based on different washer-dryer models, and hence show a slightly distinct behavior in an otherwise equal test setting. The available datasets were provided as they contain interesting sequences for an anomaly detection task.

The washer-dryer machines are placed on test benches for recording and storing values from several different internal sensors of the machines. The related sensor channels thus capture the machines physical behavior in terms of e.g. electrical current, water temperature or frequency of the washing drum. Apart from these real-valued sensor channels, the dataset also contains contextual status information, such as the absolute runtime of the machine or the currently selected washing program. This contextual information is either recorded directly from the machine, or else added by the test bench itself. In total, the sensor data contains 251 different channels.

Washer-dryer machines are designed to efficiently merge two devices into one. Subsequently to the washing process, the washer-dryer dries laundry automatically without requiring a secondary machine. However, this introduces an additional level of complexity to the machines. For example, the washer-dryer can experience blockages of the fan designed for the drying process. The blockage may occur due to leftover cloth particles from the preceding washing process. To prevent this, anomalous behavior of the fan should be detected in advance. Additionally, the available machines are expected to show similar characteristics regarding the drying process, as the same type of module is shared over the machines in the available datasets. Hence, this thesis focuses on detecting anomalies which hints to a possible blockage of the washer-dryers fan. However, the available dataset contains only very few test cycles containing this specific type of failure.

One of the available data channels contains a notification signal that can be used to find traces of possible failures. The notifications are raised by the test bench based on a set of rules defined by the test engineers. This can be used to label cycles as anomalous by filtering for notifications that are connected to the blockage of the fan. For this purpose, test engineers defined a subset of relevant notification codes that can be utilized for preprocessing the dataset. The errors hinting at a blockage of the drying fan are mostly connected to the fan itself. This includes an unexpectedly higher rotation speed or the fan stopping at an unusual moment during the dying process. In conclusion, failures might be detected by unusual sensor values, but also by unexpected configurations of otherwise usual values in multiple channels. Hence, covariance of the channels must be modeled by the anomaly detection system. In addition, normal configurations might be anomalous at a different point in time during the washing or drying process. This introduces the necessity to also model temporal dependencies between patterns in the sensor data.

Due to the complex nature of the washing and drying process, every cycle within the dataset is unique to some extend. This can be attributed to differences in configuration, e.g. slightly varying amounts of laundry used in the washing cycles. The internal system of the washer-dryer reacts dynamically to these configurations, which causes variations in the recorded sensor data.

In addition, different models of washer-dryer machines can show slightly varying characteristics, even though the same type of washing program is tested. Nonetheless, the same washing and drying programs show noticeable similarities and reoccurring patterns in the recorded data. Still, those patterns can be vastly different regarding the length of single parts of the washing or drying process and therefore the overall length of the cycle. A system for processing entire washing cycles or subroutines within these cycles should therefore be able to handle these differences in sequence length.

Real-world sensor data introduces the challenge of missing or inconsistent values due to the complex process of recording and storing high-frequent data. In this case, the test bench records data at a specified frequency. The data is partly produced by the test bench itself and otherwise received through an interface to the internal system of the washer-dryer machines. In case the test bench does not receive any data from the machine in time, *NaN*-values are stored instead of the actual sensor values. In the available dataset, an estimate of 1% of the recorded data is missing. Hence, an imputation of missing values is required in the preprocessing pipeline. In addition, parts of the dataset showed to be corrupted for an unknown reason. In those parts, the sensor values are distributed randomly and cannot be used. Therefore, these corrupted parts of the data have to be detected and excluded from the dataset.

In addition to the expected variation in cycle length, the sampling rate of the test bench can differ as well and might even change dynamically. This adds up to the challenge of handling the vast amount of data points per cycle, where the easily recognizable pattern span over many minutes within the cycle. In conclusion, a high resolution of the data might not be beneficial for detecting pattern which can also be recognizable at a much lower sampling rate. Recent publications described in section 2.3 mostly utilize windows over sensor sequences with less than 500 data points for pattern recognition. Thus resampling at a lower but fixed rate compared to the original data sequences will be a crucial part of preprocessing for detection of anomalies in full cycles. In contrast, applying a window function on the sensor data might not require decimation. Nonetheless, even if decimation is not required for pattern recognition, the sample rate still has to be normalized to a fixed rate.

For this thesis, several different datasets from long-term tests are available. These datasets are each recorded by test benches monitoring a unique machine from a new product line. In theory, each long-term test is conducted for a fixed amount of cycles running the same washing and drying program. However, as the tests where still in progress at the point of accessing the data, the tests each have a varying number of cycles. Also the focus of a specific drying component limits the possible amount of machines to retrieve data from. Table 3.1 lists the available long-term tests as well as the count of extracted normal and anomalous cycles.

Table 3.1: Available data sets and the containing number of cycles. The table further shows the number of extracted normal sequences as well as cycles with relevant failure cases. The remaining fraction of the cycles was discarded.

DATA SET	CYCLE COUNT	NORMAL	ANOMALIES
1	1673	275	97
2	441	416	5
3	373	354	3
4	3471	1405	75

The anomalies listed in table 3.1 have partly been labeled directly by test engineers. In addition, cycles with relevant failure notifications can be labeled as anomalous during preprocessing of the dataset. The set of possible notifications includes a few hundred different codes. From these, a subset of a few distinct notifications were selected. These selected notifications may hint to a problem related to the blockage of the dryer fan. However, without a deep knowledge of the domain, these failure notifications may also include a high fraction of false positives. This makes it difficult to use these extracted anomalies for validation of the anomaly detection model.

In summary, the described challenges of the used dataset are as follows:

- highly multivariate setting with an underlying multimodal data distribution
- varying cycle length and sampling rate
- missing and inconsistent values
- few relevant failures and irrelevant failures to discard

These challenges influence the preprocessing steps necessary to process the data with machine learning algorithms. In addition, it also constraints the deep learning architecture implemented for anomaly detection.

3.2 Data Preprocessing and Feature Selection

The characteristics of the data described in the previous section 3.1 show that extensive preprocessing is required in order to transform the raw sensor data into a form that can be processed by a deep learning algorithm. This includes the selection of relevant items from the datasets and the filtering of unusable cycles. However, the extracted features are desired to be as close to the raw sensor data as possible. This allows to reduce the amount of domain knowledge required for the anomaly detection task.

Preprocessing

From a data mining perspective, the datasets available for this thesis contains raw and unclean sensor data. To derive knowledge, the data has to be transformed and cleaned before it can serve as training data for a machine learning algorithm. Figure 3.1 visualizes the pipeline consisting of several computational steps that has been implemented to achieve this.

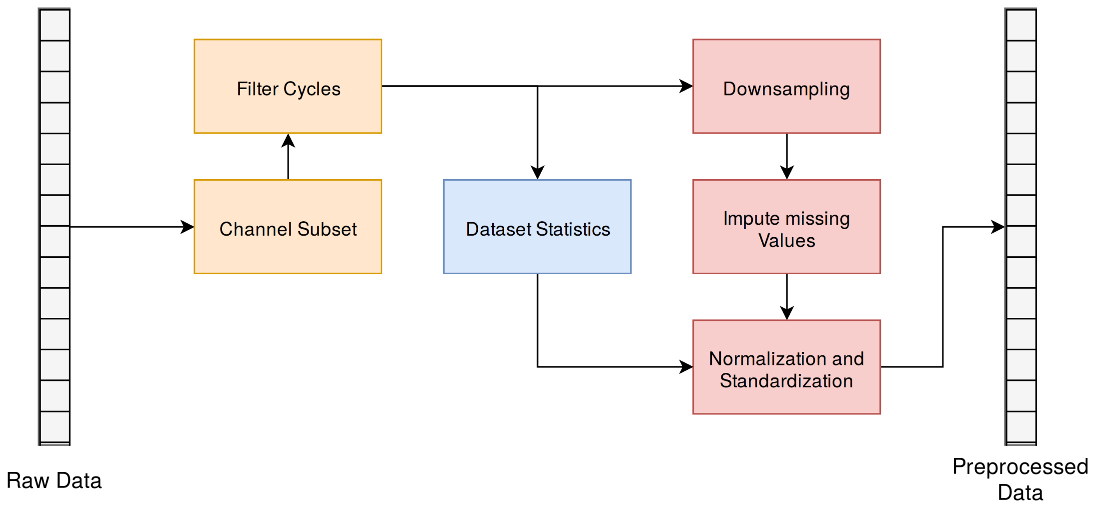


Figure 3.1: Preprocessing steps taken to transform the raw channel data from the dataset into samples that can be subsequently used as input to the anomaly detection architecture.

From the 251 different sensor channels, a subset of 35 channels can be selected. The remainder of the channels can be discarded. This is based on the experience of test engineers. According to the engineers, solely these selected channels are of interest for the detection of the anomalies regarding the drying fan. This reduces the size of the preprocessed dataset and also reduces the complexity in terms of model parameters of the processing neural network. The selected channels include sensors from the machine that capture e.g. electrical current, the rotation frequency of the washing drum or the rotation frequency of the fan used for the drying process. In addition, these selected channels include many status information. This includes e.g. the currently running program or notifications about occurring errors. In the following, these two types of selected channels are referred to as *sensor channels* and *status channels*.

Sensor channels contain actual measurements from the machine, whereas status channels contain auxiliary information, e.g. the selected washing program. The status information is partly not generated by the machine itself, but is instead provided by the testing environment, e.g. the test bench. Hence, detecting anomalous patterns can be limited to the sensor channels only. Nevertheless, status channels can give valuable meta information, such as the overall runtime of the machine which can influence the observed sensor values e.g. due to abrasion effects. Thus status channels can be provided to the machine learning algorithms as auxiliary data, which may benefit the anomaly detection quality.

Several of the available washing and drying cycles include error notifications that are not relevant for the failures regarding the drying fan. Some of these error notifications lead to an interruption of the program which is only resumed, once an engineer discarded the notification. Without a throughout evaluation of these cycles, they cannot be labeled as normal and are also not relevant as anomalous examples. Hence, such cycles cannot be used for training and are therefore removed from the dataset. This is done by filtering out all cycles whose notification channel contains codes that matches a predefined set of error notifications. However, filtering the cycles by those notification may discard false-positives as well and significantly reduces the number of cycles in the dataset.

Recent literature shows that LSTM-based recurrent neural networks can provide good results for sequence modeling with an input length of up to 500 data points (Malhotra et al., 2016). Sequences with more data points can require decimation or some form of segmentation. For instance, a window function can be applied in order to segment the sequence into smaller parts. For anomaly detection on previously recorded cycles, the sensor data can be sampled down with a high decimation factor in order to allow processing of entire cycles as a single input sequence. This can be very efficient but does not allow to detect ongoing anomalies. In contrast, a sliding window function can be applied without or with only marginal decimation of the sequences in order to detect anomalies in a timely manner.

However, the datasets may contain an alternating sampling rate. Therefore, suitable decimation methods were chosen in order to re-sample the data to a fixed rate of data points per minute. To achieve this, different methods for re-sampling have been considered. For status channels with only low-frequency changes and mostly discrete values, a simple pick of every n-th value proved to be sufficient for decimation. In contrast, sensor channels with high-frequency changes and continuous values remained difficult to efficiently decimate. *Reservoir-sampling* and decimation by calculating mean values were evaluated for every sensor channel. In addition, more complex methods for decimation have been considered, e.g. *Largest-Triangle-Three-Buckets* (LTOB) (Steinarsson, 2013), or applying a decimation filter. Finally mean calculation showed a good balance between simplicity and visually satisfying results. However, decimation of entire cycles to e.g. 10 data points per minute results in a certain amount of information loss, especially regarding extrema in the sequences.

All selected channels with continuous values have been either normalized or standardized. The sensor channels were standardized to mean $\mu = 0$ and standard deviation $\sigma = 1$. The per-channel statistics for standardization have been previously calculated over the entire dataset of normal labeled sequences using the numerically stable *Welford's method* for calculating variance in an iterative manner (Chan et al., 1983). In general, the standardization of input data to a mean of $\mu = 0$ and a standard deviation of $\sigma = 1$ has proven to speed up the training of neural networks (LeCun et al., 2012).

On average, the dataset contained 1% of missing values. Cycles with an channel containing entirely *NaN*-values have been discarded entirely. Missing values in discrete status channels have been filled by imputing the previous value if possible. For the sensor channels with continuous values, a method for imputation of a mean value between the surrounding values has been implemented. In case of entire sub-sequences missing, the values were imputed by approximating a linear function between the surrounding values.

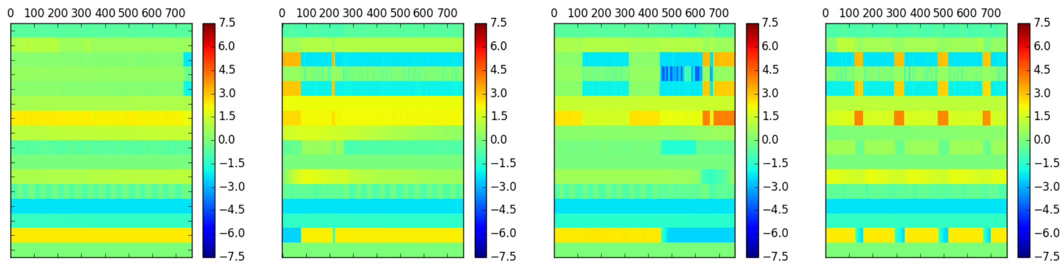
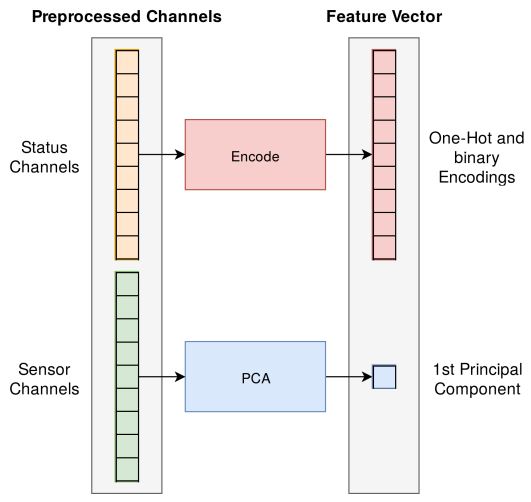


Figure 3.2: Heatmap of sensor channels that have been standardized and decimated to 100 data points per minute.

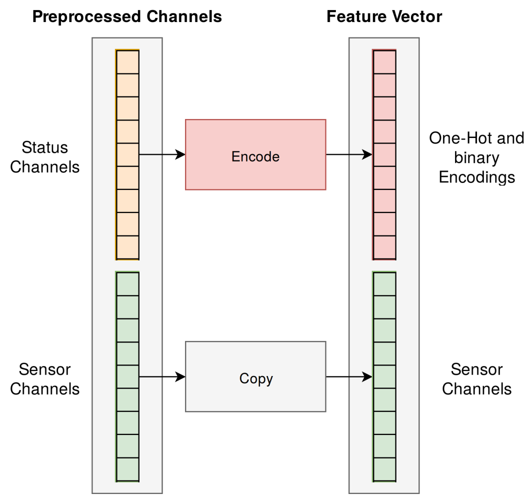
The extracted and preprocessed channels can be visualized in form of a heatmap. Once the sequences are standardized according to calculated channel statistics, the heatmap gives a good intuition of the characteristics of normal sequences. Figure 3.2 shows instances of the 15 selected sensor channels for a specific drying process in the washing cycles. The sequences have been decimated to 100 data points per minute.

Feature Selection

As the previously defined sensor channels show the actual behavior of the machines, these channels may be solely considered as input for the anomaly detection model. This implies however, that these channels contain sufficient information to allow e.g. an autoencoder model to compute a rich latent representation of the input sequences, that can be used to reconstruct the input sequence. This is not necessarily given, as the sequences from this dataset can show unpredictable behavior. This can occur once the machines internal control unit alters the expected washing procedure due to constraints, which are not easily traceable without context. Therefore, the status channels may be considered as auxiliary information to enable e.g. an autoencoder model to reconstruct even unpredictable patterns in the sequences. This can be verified by experiments with both, sensor channels and a combination of sensor and auxiliary status channels.



(a) Features per time step with PCA

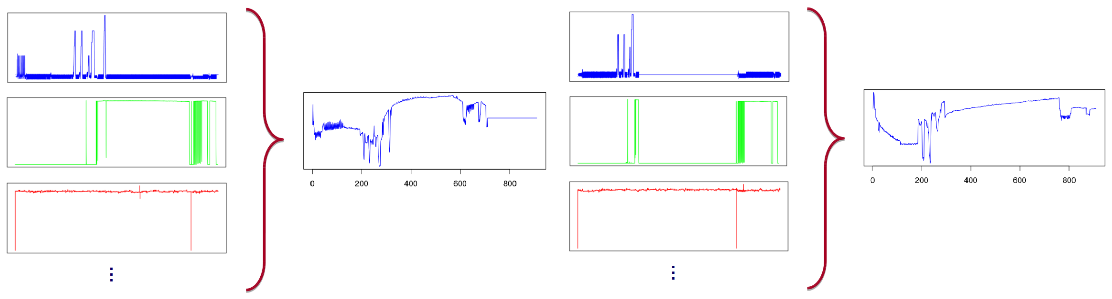


(b) Features per time step with full sensor channels

Figure 3.3: The extraction of features per time step from preprocessed sensor data. Status channels were either represented as one-hot encodings or else copied as binary features and combined into a fixed-size auxiliary vector. For parts of the experiments, the sensor channels with continues data were reduced by calculating the first principal component through PCA (a). Otherwise, the sensor channels were simply copied in order to join the auxiliary feature vector containing status information with the standardized continuous values (b).

One of the major advantages of certain deep learning architectures is the potential to directly use raw input data and take advantage of the architectures ability to automatically learn meaningful representations from the input. However, at least the status channels have to be transformed in order to serve as auxiliary input features for the anomaly detection algorithm. This includes one-hot encoding of all discrete status channels which can have more than two states. Therefore, a channel with n possible states is represented in a fixed-size binary vector $v \in \mathbb{R}^n$, so that all except one dimensions are 0. This allows all status channels to be concatenated and represented as a single fixed-size binary feature vector.

In contrast, the previously standardized sensor channels can be copied and concatenated to the feature vector. Alternatively, the sensor channels can be reduced to one dimension by calculating the first principal component through a *Principal Component Analysis* (PCA). This captures a certain amount of variance from the original sensor channels in only one dimension, and thus, significantly simplifies the complexity of the neural network for anomaly detection. Instead of computing an anomaly score based on multivariate sensor data, the problem can be reduced to a single dimension. Without the auxiliary status channels, only a scalar value has to be considered per time step. This method showed good results in a similar setting as described by Malhotra et al. (2016). Nevertheless, reducing the sensor channels will only be used as a starting point in the experiments here, as it cannot be expected that anomalous patterns are still detectable in only the first principal component. In addition, even detecting unexpected behavior in the reduced dimension does not allow retracing the origin of the anomaly in the original sensor channels.



(a) First principle component of normal data (b) First principle component of anomalous data

Figure 3.4: Calculation and extraction of the first principle component of the sensor channels through PCA. The comparison between a normal example in (a) and an anomalous example in (b) shows significant differences in the reduced data.

Figure 3.4 shows the calculated first principal component of a normal sequence in comparison to an anomalous sequence where the drying fan is blocked. For this comparison, the reduced

Table 3.2: Combinations of feature vectors for the experiments.

Feature Combination	Dimensions
First principle component	1
First principle component + auxiliary channels	53
Full sensor channels	15
Full sensor channels + auxiliary channels	67

sequences have been zero-padded to a fixed sequence length. The first principle component of normal sequences shows the same characteristics over all normal cycles from the different datasets. However, the patterns vary slightly in position and duration so that each cycle sequence is unique to some extend. In contrast, the first principle component of the labeled anomaly shows a significantly different pattern. This supports the findings of Malhotra et al. (2016). Solely considering the first principal component of the sensor channels promises to be sufficient for detection of similar anomalous pattern. However, this requires confirmation from empirical results.

PCA is a method for projection of data into a lower-dimensional representation, whereby the variance in the data is preserved. Nevertheless, the algorithm can yield different results as the resulting sequence of the first principal component can appear inverted. For simplicity, these sequences can be inverted subsequently to calculating the first principal component, so that all reduced cycles are comparable. This can also be learned directly by the neural network itself.

In summary, four different combinations of feature vectors have been considered for the experiments. The 20 sensor channels were transformed into an auxiliary feature vector with a dimension of 52. The increase of dimensionality is here attributed to the one-hot encodings of channels with categorical values, for instance the different washing or drying programs.

The standardized sensor channels where either reduced to one dimension or else copied and therefore represented 15 features per time step. Experimenting with different combinations of feature vectors allowed to evaluate the feature extraction capabilities of the neural networks implemented for anomaly detection.

3.3 Architecture Selection

In a typical anomaly detection setting, the training data contains mostly normal examples and the number of available anomalies is limited to only a handful of instances. Similar, the dataset used in this thesis does not provide a suitable amount of labeled examples that can be used as an anomaly class for training. This makes it difficult to implement a binary classification model detect anomalies, similar to the classifier used in Rajpurkar et al. (2017). Hence, a more reasonable approach is the training of a predictive model exclusively on normal data. As shown in section 2.3.2, 2.3.3 and 2.3.4, many different approaches have proven that modeling normal data to distinguish known from unknown samples can lead to good results for detecting anomalies in similar settings.

For the first experiments, an autoencoder model based on LSTM networks has been implemented in a similar fashion to the model proposed by Malhotra et al. (2016). The following section 3.3.1 describes the model and the architectural extensions for this dataset.

The experiments on this approach described in 3.4.2 found that a generative model has certain advantages over the autoencoder approach. Hence, an GAN has been implemented for generation of normal sequences. The details of the GAN-architecture are presented in section 3.3.2.

3.3.1 Autoencoder-based model

Due to the lack of anomalous samples in the given dataset, a reconstruction method has been chosen over a significantly less complex classification method. A model for reconstruction allows to train the architecture exclusively on normal data. As shown by Malhotra et al. (2016), a reconstruction model is trained with the objective to flawlessly reconstruct normal data. Once anomalous patterns occur within the sequence to be reconstructed, the model is expected to show a significant error between reconstruction and the original sequence.

Apart from the benefit to train only on normal data, a reconstruction model also allows to easily determine the exact position of anomalous patterns within the sequence. In contrast, sequence classification models as used by Ordóñez and Roggen (2016) often utilize a fixed size sliding window over the sequence and therefore categorize an entire subsequence with only one label. Rajpurkar et al. (2017) propose sequence classification in form of a sequence-to-sequence model for the annotation of multiple time steps. Similarly, a reconstruction model allows to calculate the error between the reconstructed sequence an the original input sequence at every single time step. Thus the only limit in terms of resolution for anomaly detection is the chosen decimation factor during preprocessing of the sequence and the sampling rate of the underlying sensors.

In section 2.3.3, different architectures for general sequence-to-sequence modeling have been presented. Many similarities can be seen in recent literature for machine translation, e.g. the *GNMT*-architecture (Wu et al., 2016), where many interesting concepts were combined into a sequence-to-sequence model based on multiple layers of LSTM networks. In contrast, recent literature also shows how to incorporate convolutional operations for sequence encoding and deconvolutional operations (Zeiler et al., 2010) for decoding respectively. Examples can be found in *WaveNet* (Oord et al., 2016a) or its immediate successor *ByteNet* (Kalchbrenner et al., 2016), which are briefly described in section 2.3.3. However, the most similar approach to the context of this thesis is the architecture chosen by Malhotra et al. (2016). The authors use an encoder-decoder model consisting of LSTM networks for anomaly detection in sensor data.

The crucial advantage of choosing a recurrent neural network approach over a convolutional approach lies in the challenges of the dataset at hand. In theory, recurrent networks can easily handle variable sequence length due to parameter sharing at each time step. This enables to handle the immense variations regarding the sequence length between different cycles within the dataset. Therefore, a recurrent neural network was selected as a baseline for the encoder as well as decoder architecture. However, it has to be noted that many recent approaches in sequence-to-sequence modeling rely on fully convolutional architectures that are modified in order to also work with variable-length sequences (Kalchbrenner et al., 2016; Gehring et al., 2017). The benefit of convolutional networks is that they remain computable in parallel. For this thesis, the capability to process variable length inputs is only relevant for computing an anomaly score for an entire cycle. Applying a fixed-size window function to segment the sequences that serve as input to the model renders this problem obsolete.

Similar to Malhotra et al. (2016), the first principal component has been calculated to reduce the sequence reconstruction task to one dimension only. Nevertheless, instead of only feeding the single dimension, derived from the sensor channels as input into the encoder, the status channels were also considered to enhance the input dimensions per time step. This allows the network to also incorporate context from the status channels while encoding the input. The intuition is here, that sequences can rely on unpredictable changes in status, e.g. a change of the currently running program of the machines. Providing the decoder with these contextual information is thus expected to improve the reconstruction quality.

Unrolling a RNN network on the temporal axis is limited in terms of sequence length that the network can efficiently handle. This is attributed to the vanishing gradients that may occur during backpropagating through a very long network path as described in section 2.2.1. This introduces constraints on the length of input sequences fed into the anomaly detection model. LSTMs proved to work well, even for longer sequences. However, once entire cycles are considered as input sequences, even a relatively high decimation factor results in a sequence with more than 1000 data points.

As the sequences used as input for the model are rather long compared to related approaches in recent literature, different types of recurrent networks were considered to efficiently encode long-term information from the input. Hence LSTM cells were used as the baseline for encoder and decoder layers, as they proved to be very effective even for long sequences. Another possible solution is to use the recently introduced *Phased* LSTMs (Neil et al., 2016) which are supposed to be effective for sequences with more than 1000 time steps.

In addition, a bidirectional recurrent layer has been considered as the first layer of the encoder due to the overall good results in recent machine translation models. The effectiveness of such a bidirectional layer can be explained by the cross-availability of context from the entire sequence in every time step of the following layers (Graves et al., 2013).

Other additions can also be derived from recent advances in machine translation. E.g. Wu et al. (2016) used residual connections for the sequence-to-sequence architecture with more than 4 layers stacked in either encoder or decoder. This enabled the training of very deep architectures which otherwise tend to suffer from vanishing gradients. Another interesting concept is the use of an attention network between the encoder and the decoder network. However, applying an attention mechanism over the output sequence of the encoder network would allow the network to simply copy the input sequence to achieve the prefect reconstruction. The use of attention in an autoencoder model for anomaly detection is thus not immediately apparent and has to be evaluated further.

The autoencoder network is therefore implemented as an LSTM-based encoder-decoder architecture with multiple layers. The number of layers in the encoder and decoder network are evaluated in the experiments. This includes incorporating a bidirectional LSTM as the first layer of the encoder network. Another parameter to evaluate is the number of hidden units per LSTM cell. This also influences the dimension of the latent representation between encoder and decoder network, as the last hidden state of the last encoder layer is used to initialize the decoder network.

Malhotra et al. (2016) feed the ground truth as input into the decoder network during training. The sequence is shifted, so that each decoder step is provided with the true value of the previous decoding step respectively. In reconstruction applications, the decoder is trained to reconstruct the input to the encoder. Therefore, the decoders ground truth at the time step t equals the input sequence at time step t. This procedure was introduced as *teacher forcing* (Williams and Zipser, 1989) and enables easier training of sequence-to-sequence models due to less transient errors that indirectly influence the succeeding decoding steps. Especially during the beginning of training, the model outputs more or less random values. Once the model output is used as feedback for the following decoding steps, the training will be slowed down significantly. However, the decoder network may learn to rely to heavily on the ground truth. The actual effect of teacher forcing has to be evaluated in experiments.

An extension to this is *Schedules Sampling*, introduced by Bengio et al. (2015). The decoder samples either the ground truth or the actual output of the previous decoding step based on a certain probability. The probability to sample the ground truth can be reduced incrementally during training, so that the network progressively learns to trust its own prediction, instead of relying on the ground truth. This can potentially speed up the training in the beginning without the decoder overfitting on the ground truth feedback.

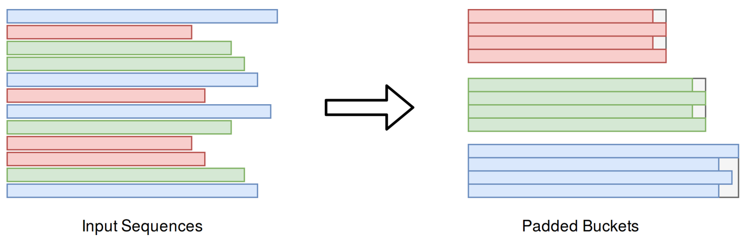


Figure 3.5: Bucketing: Sequences with different length (left) are grouped according to predefined boundaries, which decreases the amount of padding within the batches taken from single buckets.

The input sequences from the given dataset can have very different lengths, caused by slightly varying configurations at each cycle. For instance the amount of laundry filled into the machine can cause the washing cycle to have entirely different characteristics and therefore result in an alternating sequence length. Using an RNN-based architecture allows to dynamically unroll the model for each different sequence length. However, mini-batches are used to increase the efficiency by training on multiple instances at once. Therefore, the sequences within a mini-batch have to be padded to the largest length in the batch respectively.

An efficient method to handle batch-wise padding is to use *bucketing*, where the dataset is split by sequence length into subsets with predefined boundaries. Figure 3.5 demonstrates the bucketing procedure. *Bucketing* ensures that sequences are arranged in clusters with similar sequence-length. Mini-batches can be subsequently served from a single bucket, which reduces the amount of padding that is necessary within the mini-batch.

The autoencoder learns to reconstruct normal sequences. Anomalies can be detected through calculating an anomaly score based on the differences between the original an the reconstructed sequence. The function to calculate the anomaly score will be specified in section 3.4.2 along with the description of the conducted experiments.

3.3.2 GAN-based model

The second model that has been considered for anomaly detection is based on a generative model with adversarial training, as shown in section 2.3.4. The previously described autoencoder model requires a manually defined anomaly score function to evaluate the reconstruction of a processed input in order to find anomalous pattern. Even the *Variational Autoencoder* (VAE) for anomaly detection introduced by Sölch et al. (2016) requires the explicit calculation of an anomaly score per data point.

In contrast, GANs use a designated classification model (discriminator) as part of the adversarial training. Hence, there is no need to define an anomaly score function, given the assumption that the discriminator learned to classify normal and anomalous samples with sufficient accuracy. Another advantage of GANs is the possibility to sample from the latent space and generate sequences that effectively demonstrate what the model learned to be normal data. This allows to better interpret the model after training.

The model used in this thesis is inspired by research on detecting anomalous pattern in medical images presented by Schlegl et al. (2017) as described in section 2.3.4. Hence, the implemented GAN also uses deep convolutional networks for both, the generator G as well as the discriminator D. This approach is based on the *Deep Convolutional Generative Adversarial Network* (DC-GAN) architecture proposed by Radford et al. (2016), which proved to be suitable for several different unsupervised learning tasks and is therefore used in multiple successful applications that are based on GANs. This can mainly be attributed to the stability of the architecture in adversarial training, as GANs are generally hard to train due to being unstable.

Figure 3.6 shows the architectural topology of the discriminator as well as the generator model. The discriminator D consists of multiple convolutional layers and a single dense layer with sigmoid activation in the end to predict the probability $p(x)$, that an input sample x belongs to the normal class. Therefore, $p(x) = 1$ can be interpreted as a certainly normal example and $p(x) = 0$ as an example, that does not match the characteristics of the normal class and is thus likely to be an anomaly.

Both, the discriminator and the generator network consists of stacked building blocks, that are constrained by the size of the data samples.

The convolutional layers in the discriminator do not contain any pooling functions like *maxpooling* for downsampling. Instead, temporal downsampling is learned by the model itself during training through strided convolutions as proposed by Springenberg et al. (2015). However, the channel dimensions are preserved as shown by Ordóñez and Roggen (2016). In contrast, the generator uses spacial upsampling in every layer by repeating the values. In the first layer,

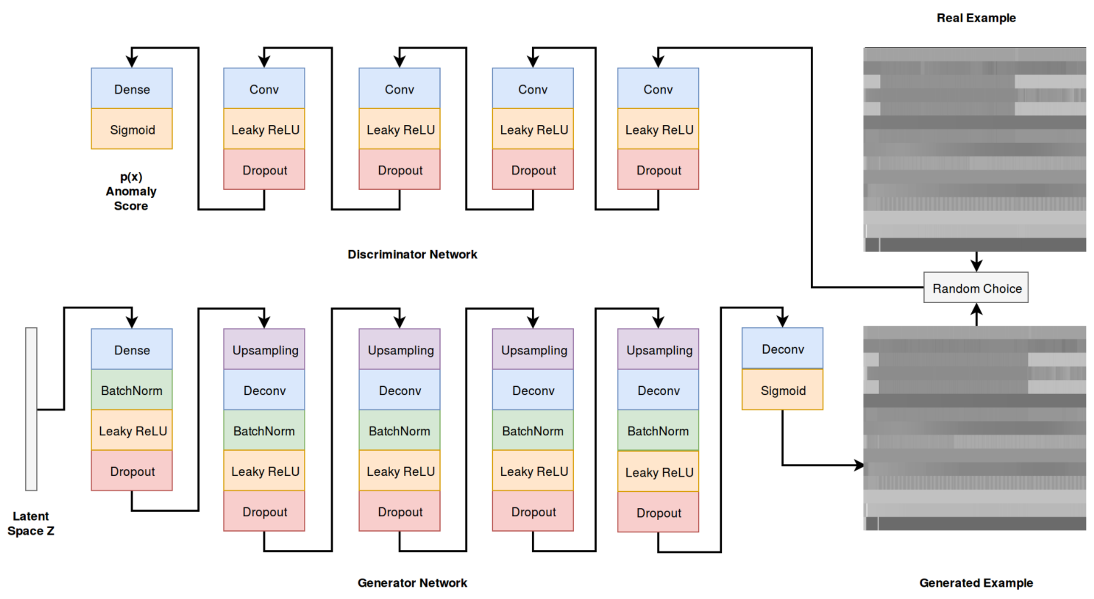


Figure 3.6: Architecture of the generative adversarial model used for anomaly detection. The generator and discriminator are jointly trained in a minimax game. The discriminator is trained by feeding generated or real examples by random choice, in order to distinguish between real and fake data samples. The generator is trained to produce examples, which fool the discriminator into classifying them as normal data.

upsampling by factor 2 is applied to the temporal as well as the spacial axis. In the following layers, upsampling by 2 is only applied to the temporal axis.

The generator network uses deconvolutional layers (Zeiler et al., 2010). These transpose the forward and the backward passes of convolutions and therefore allows computing feature maps of larger size than the input while keeping the connectivity pattern of convolutions (Dumoulin and Visin, 2016). Combined with the upsampling functions, this allows to project the latent variable z to the original size of the input sequences. Table 3.3 lists the parameters of the dense and convolutional layers in the generator and discriminator network.

Dropout is applied to all layers in the generator and discriminator except for the first discriminator layer and the last generator layer. This showed to produce better results and might also be applied during inference to provide the generator with additional noise (Isola et al., 2017).

In contrast to the original DC-GAN by Radford et al. (2016), *Batch Normalization* (Ioffe and Szegedy, 2015) is only applied to the generator network. Several projects on generative modeling with GANs found that *Batch Normalization* in the discriminator network did not improve the adversarial training. However, this has to be validated for this specific use case.

Table 3.3: Parameters of the layers in the generator (left) and the discriminator (right) network trained to generate the 15 sensor channels. For dense layers, the size denotes the number of hidden units. In case of convolutional layers, the number of filters and the filter size (in brackets) is given. The network topology is derived from DC-GAN (Radford et al., 2016).

Generator Layer	Size
Dense	50*256
Deconv 1	128 (5 × 5)
Deconv 2	64 (5 × 5)
Deconv 3	32 (5 × 5)
Deconv 4	32 (5 × 5)
Deconv 5	1 (5 × 5)

Discriminator Layer	Size
Conv 1	16 (15 × 5)
Conv 2	32 (15 × 5)
Conv 3	64 (15 × 5)
Conv 4	128 (15 × 5)
Dense	1

The latent code z is represented as either a uniformly or gaussian distributed vector of size 100. Both, the distribution and the size of z can be evaluated through experiments.

LeakyReLU (Maas et al., 2013) has been used for each layer except the last discriminator layer. This can potentially make the adversarial training of the architecture more stable by avoiding sparse gradients and was recommended for the original DC-GAN model (Radford et al., 2016).

During each training step, the discriminator and the generator are trained separately. First, the discriminator is either trained on a mini-batch of real or generated sequences by a random choice. Subsequently the generator is trained to fool the discriminator.

However, the most interesting feature of GANs is the possibility to include labeled examples that do not resemble the normal data. Hence, known anomalies or synthetically constructed failure cases can be mixed with generated samples for the training of the discriminator.

The experiments in this thesis will be limited on training the GAN to generate normal samples and validate the capability of the discriminator to distinguish previously unseen normal sequences from anomaly examples. However, Schlegl et al. (2017) found that the discriminator prediction does not suffice and also calculated an anomaly score based on generated normal samples. This procedure is described previously in section 2.3.4. Nevertheless, this will be evaluated for this use case in section 3.4.3 based on conducted experiments.

3.4 Experiments

For the task of anomaly detection in the given sensor data, two reasonable deep learning approaches have been selected. These approaches have to be evaluated through experiments on the dataset. This includes the general capability for detecting of anomalous patterns in the sensor data as well as the selection of hyper-parameters to optimize the algorithms.

Section 3.4.1 describes the experimental setup in terms of hardware, partitioning of the dataset and the evaluation of the results. This is followed by the actual experiments.

First, experiments with different variations of LSTM-based autoencoder models will be presented in section 3.4.2. Subsequently, section 3.4.3 shows the conducted experiments for utilizing a GAN architecture for the same task.

3.4.1 Experimental Setup

The encoder-decoder architecture has been implemented using *Googles* framework for distributed machine learning *TensorFlow* (Abadi et al., 2016). The training of the model has been implemented end-to-end on a GPU (Nvidia GTX 1070) with 8 GB of memory and a total of 1920 *CUDA* cores. For testing and validating the model, inference was run on a machine utilizing a 4-core CPU (Intel i7 6700k) with 32 GB of DDR4 memory. The use of a GPU for the training of the models decreased the time significantly. However, as inference steps were only taken for model validation and testing on a small subset of the data, running inference on a CPU was sufficiently fast. In addition, the *Keras* library has been used on top of *TensorFlow* for some of the experiments. This simplified especially saving and loading of trained models.

For evaluation, different metrics have been used, that were partly taken from the *scikit-learn* library for python (Pedregosa et al., 2011).

In order to evaluate the performance of the selected architectures for anomaly detection, the available data is partitioned into training, validation and test split. Training of both, the autoencoder and the GAN model is done in a semi-supervised manner. First, each model is trained unsupervised exclusively on normal data. Hence, the training split contains the majority of data which is assumed to show normal behavior. The remaining normal examples as well as the anomaly examples are evenly split into the validation and test set. Once the training is done, the validation split is used for supervised parameter tuning. Finally, the test set can be used to verify the accuracy of the selected model on previously unseen data.

Each architectural variation is evaluated on previously unseen normal as well as anomalous data. The latter is both real as well as synthetic, as the number of anomalous samples in the dataset are very few. Even though the anomaly class is augmented with synthetic samples, the normal and anomalous classes are still unbalanced.

Different types of metrics are calculated to give a good estimation of the architectures capabilities for anomaly detection. First, a normalized F_1-*score* is calculated for invariance to the skewed classes in the dataset. The F_1-score is derived from the F_β-score, which is defined as follows:

$$F_\beta = \beta * \frac{Precision * Recall}{\beta * Precision + Recall} \tag{3.1}$$

The F_1-score is therefore defined for $\beta = 1$. The function gives a measurement for the quality of a classifier by calculating a weighted fraction of recall and precision. Here, *recall* is defined by the fraction of relevant items retrieved, whereas *precision* shows the percentage of how many of the detected items are accurate. For normalization, *macro* weighting of the F_1-score has been used for all the experiments.

3.4.2 Autoencoder Experiments

The autoencoder approach to anomaly detection showed to work for a similar setting in recent literature. The first experiment focuses on an architecture that is most similar to the LSTM-based autoencoder used by Malhotra et al. (2016). Hence, the sensor channels are reduced to the first principle component through PCA. In addition, full washing cycles were used as input for the autoencoder. This makes it easy to visually interpret the resulting sequence, as the reoccurring pattern are easily recognizable. This approach is then extended to reconstruction and anomaly detection on all sensor channels for smaller input sequences.

In all experiments, the autoencoder model is trained on the training split which contains only normal sequences. As soon as the model is trained, the validation split was used to calculate a multivariate gaussian distribution $N(\mu, \Sigma)$ over the reconstruction error of all sequences from the validation data. This allows to calculate an anomaly score for every time step in a similar manner as used by Malhotra et al. (2016) for anomaly detection. Given the normal distribution, the anomaly score $a^{(i)}$ can be calculated for every time step i in the reconstructed sequence with the score function:

$$a^{(i)} = \frac{(e^{(i)} - \mu)^T}{\Sigma(e^{(i)} - \mu)} \tag{3.2}$$

The scalar anomaly score $a^{(i)}$ can be interpreted as a normalized reconstruction error, given the deviation expected for normal data samples. Therefore μ is the mean vector and Σ is the calculated covariance matrix for the multivariate reconstruction error. Finally, a threshold value ϵ can be defined empirically by determining the largest anomaly score from the validation data, so that the calculated anomaly score satisfies $a^{(*)} < \epsilon$ for all normal examples in the validation set. The threshold can also be computed through the *Youden's index*[1] based on the *Receiver Operating Characteristic* (ROC). Another approach would be to calculate the closest point on the ROC-curve to the left-upper corner.

The anomaly score is calculated for every data point in the reconstructed sequence. Hence, every data point with an anomaly score that exceeds the defined threshold can be marked as anomalous. For simplicity during validation and testing, an entire sequence can be labeled as anomalous, once a single data point in the sequence exceeds the anomaly score threshold.

Reconstruction of the first Principal Component

For the first experiments, the cycle sequences from the datasets 1, 2 and 3 are preprocessed as described in section 3.2. The cycles are decimated to 2 data points per minute. The 15 sensor channels are reduced to the first principle component. This is merged with the extracted status information in form of an auxiliary feature vector to the final feature vector with a dimension of 53 values per time step.

The available normal sequences were split into 90% training data, 5% validation data and 5% test data. From the available cycles with failures, 8 relevant cycles were selected. Due to the small amount of failure cases, model validation was done purely on previously unseen normal data. Failure cases were not needed in the validation set, as the scope of the first experiments was to find a model for basic reconstruction.

At first, a simple LSTM-autoencoder was trained to reconstruct solely the first principle component. The encoder and decoder networks are implemented to dynamically unroll the LSTM cells per layer to match the length of the input sequence. In order to efficiently normalize the length of the input sequences per batch, *bucketing* of the training dataset was used. This allowed to reduce the necessary padding for normalization of the input length. The preprocessed cycle sequences resulted in an average length of 500 data points per cycle. The first experiments are conducted in order to evaluate the number of hidden units and overall layers in the encoder and decoder network.

The encoder network was provided with solely the first principle component of the sensor channels. The decoder was then initialized with the hidden representation in form of the states of the last unit in the last encoder layer. Each decoder step was provided with the decoder prediction from the previous step.

[1]Youden's index calculation: $Sensitivity + Specificity - 1$

The model was trained using *stochastic gradient descent* (SGD) on mini-batches of the training data. Currently, there is a wide range of popular optimization algorithms based on SGD, where none of the methods stand out as being the superior choice (Schaul et al., 2014). However, according to experiments by Schaul et al. (2014), adaptive algorithms proved to be less prone to hyper-parameter tuning. Therefore, *Adam* (Kingma and Ba, 2014) is used for optimizing the weights, as the algorithm implements per-weight adaptive learning rates and *momentum*. In addition, *Adam* has been recently very popular in various deep learning applications. This makes it easier to tune the algorithms hyper-parameters based on similar experiments from recent literature.

This first model has been trained using *Adam* with a learning rate of 1×10^{-3} and a mini-batch size of 100 input sequences. These hyper-parameters have been evaluated empirically and showed promising results. For weight optimization during training, the *Mean-Squared-Error* (MSE) loss has been calculated between the input and the reconstructed sequence. The following equation shows the calculation of the MSE for a reconstruction step $\hat{y}^{(i)}$ and the actual value in at that time step $y^{(i)}$):

$$MSE = \frac{1}{m} \sum_{i} (\hat{y}^{(i)} - y^{(i)})^2 \tag{3.3}$$

The training was done for a fixed amount of *epochs*, where each *epoch* equals one iteration over the entire training set. The number of epochs is defined in an empirical manner. Alternatively, an *early stopping* mechanism can be used to terminate the training once the loss based on the test split converges.

The first experiments showed that the incorporation of a bidirectional LSTM as the first encoder layer significantly reduced the reconstruction error. This may be the result of the encoder having better access to information from across the entire input sequence at every encoding time step. Hence, multiple experiments were conducted of different architectures with bidirectional layers. The results are listed in table 3.4. For these experiments, the training loss for reconstructing normal sequences is compared to the validation loss of reconstructing previously unseen normal sequences.

Architectures with more layers and hidden units per layer seem to reduce the training loss but did not necessarily result in an improvement during validation. Instead, the training time for the same amount of epochs was increased due to more model parameters. These results may show that the amount of training data is not sufficient for generalization to the validation data.

Further, visual evaluation of the reconstruction did not show satisfactory results. To improve this, the experiments have been extended to evaluate the use of the status channels as auxiliary

Table 3.4: Comparison of reconstruction loss during training and validation of different architectures. Each model contained a bidirectional layer in the encoder network. The number of units N and layers L is shared for both encoder and decoder, whereas the bidirectional layers are excluded from the number of encoder layers. Training was done for 10000 epochs. The best results per column are highlighted in bold font.

N	L	Training Loss	Validation Loss	Training Time
128	1	7.9598 E-6	9.6011 E-4	**6H 20M**
128	2	4.9295 E-6	**8.0711 E-4**	8H 25M
128	3	2.5863 E-6	2.1564 E-3	10H 28M
128	4	3.7865 E-6	3.8911 E-3	12H 31M
256	1	5.0984 E-6	1.7331 E-3	6H 50M
256	2	2.5144 E-6	1.6736 E-3	9H 31M
256	3	**1.4710 E-6**	9.7627 E-4	12H 21M
512	1	1.0191 E-5	4.9187 E-3	12H 32M
512	2	1.6743 E-6	1.8773 E-3	18H 32M

features. Hence, the features extracted from the status channels were also used as input to the encoder model. In addition, the status information for every step were fed into the first decoder layer at the same time step. Each decoder step was therefore provided with the decoders prediction at the previous time step, as well as the actual status information at the current time step. This increased the reconstruction capability of the autoencoder models.

To reduce the training time, the autoencoder was trained using *Teacher Forcing* Williams and Zipser (1989) in the next experiments. This procedure has also been used in the LSTM-based autoencoder proposed by Malhotra et al. (2016). Hence, during training the decoder was provided with the ground truth sequence from the previous time step. During inference, the decoder output of the previous time step was fed back into the current decoding step. In theory, this speeds up the training in the beginning, where the model outputs more or less random values.

However, the experiments showed that utilizing teacher forcing lead to significantly lower reconstruction quality during inference. This can be caused by the decoder learning to heavily rely on the previous reconstruction step instead of sufficiently learning to incorporate temporal information from the networks hidden states. A solution to this problem is given by *Scheduled Sampling* (Bengio et al., 2015) as described previously in section 3.3.1. Extensive experiments were conducted to evaluate the effect of *Scheduled Sampling* to the autoencoder model.

Table 3.5 lists these experiments. For different architectures, the sampling method has been implemented using different sampling rates. The sampling rate of 0 denotes that the decoder

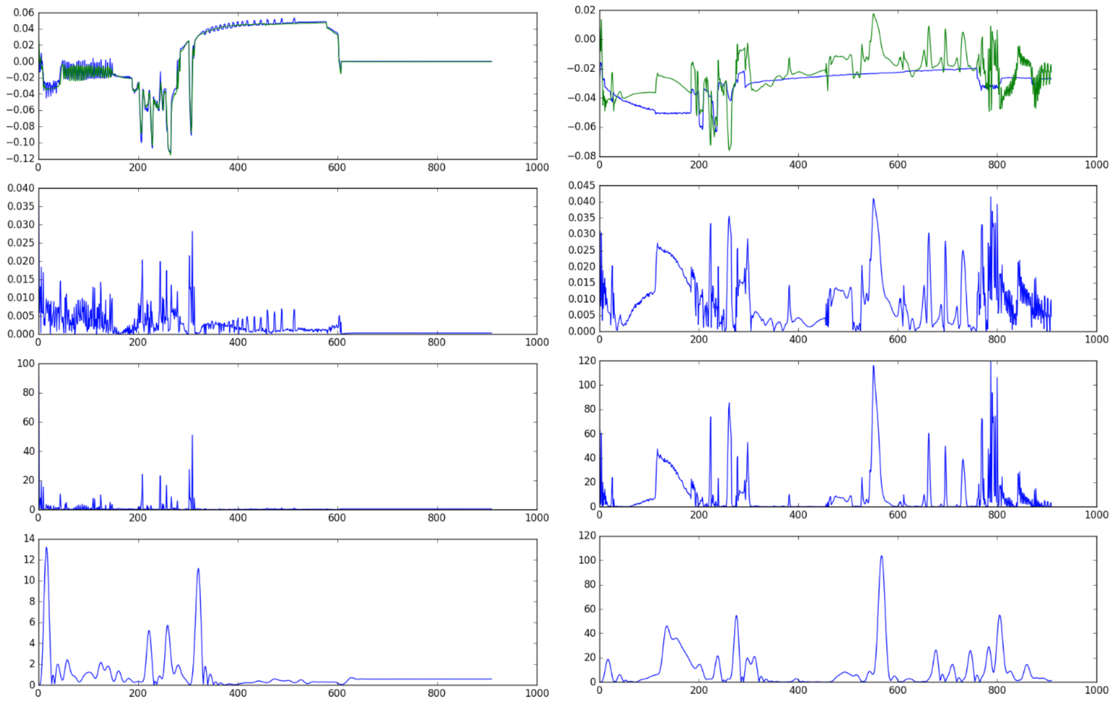
Table 3.5: Reconstruction loss of different rates for *Scheduled Sampling* in the decoder network. Each model contained a bidirectional layer in the encoder network. The number of units N and layers L is shared for both encoder and decoder, whereas the bidirectional layers are excluded from the number of encoder layers. Training was done for 10000 epochs.

N	L	Sampling Rate	Training Loss	Validation Loss
128	1	0.0	7.9598 E-6	9.6011 E-4
128	1	0.25	2.7685E-5	
128	1	0.5	2.6334E-5	
128	1	0.75	4.5693E-5	
128	1	1.0	4.0208E-4	
128	2	0.0	4.9295 E-6	8.0711 E-4
128	2	0.25	1.3210E-5	
128	2	0.5	2.2486E-5	
128	2	0.75	7.2835E-5	
128	2	1.0	3.4226E-4	
256	1	0.0	5.0984 E-6	1.7331 E-3
256	1	0.25	1.0859E-5	
256	1	0.5	2.7109E-5	
256	1	0.75	6.9570E-5	
256	1	1.0	4.8095E-4	
256	2	0.0	**2.5144 E-6**	1.6736 E-3
256	2	0.25	6.5963E-6	
256	2	0.5	3.2652E-5	
256	2	0.75	8.6253E-5	
256	2	1.0	3.8454E-4	

exclusively samples from the previous decoder steps. In contrast, with the sampling value 1, the decoder network used only the ground truth of the previous value. The sampling rate was reduced during training, so that the model could learn to use its own predictions.

The results in table 3.5 establish that the best reconstruction can be achieved once the decoder learns to use its own prediction from the previous time step. Therefore, this sampling method is discarded for the following experiments.

From these experiments, a model with a bidirectional and two additional LSTM layers in the encoder network and two LSTM layers in the decoder network has been selected. Each layer contained 128 hidden units, as these parameters showed the best results on the validation set presented in table 3.4. In contrast to the previously conducted experiments on the full cycles, this experiment was done using a window function over the cycle sequences. This allowed reducing the decimation factor and therefore maintaining more information in the sequences. The window size and step size was set to 455 data points. The decimated cycle sequences were therefore split into two parts, where the second part was zero-padded to fit the window length.



(a) Normal Sample (b) Anomaly Sample

Figure 3.7: The upper graphs shows the original sequence (blue) and the reconstruction (green). The second upper graph shows the absolute difference between original and reconstruction, whereas the second lower graph shows the calculated anomaly score for the reconstructed sequence. Finally the lower graph shows the low-filtered anomaly signal, which was used to determine anomalous patterns.

The selected model was then tested on the normal and anomalous samples from the test split. Sequences were labeled as anomalous once the anomaly score exceeded the maximum threshold value determined during the validation of the model. Figure 3.7 shows the reconstruction of a normal as well as an anomalous sequence. The sequences where processed in two parts and subsequently concatenated for visualization.

The reconstruction of the normal sequence proves that the model learned to extract relevant features from the sequences and stored them in a sufficient representation that enabled the decoder network to reproduce the input. This could be observed for the majority of the normal sequences tested, even though every normal example has unique characteristics. However, the

Table 3.6: Confusion matrix for anomaly detection using an LSTM-based autoencoder which consisted of an encoder network with one bidirectional an two standard LSTM layers and a decoder network with two LSTM layers. Each LSTM had 128 hidden units.

		PREDICTION	
		TRUE	FALSE
TRUTH	TRUE	6	2
	FALSE	5	55

calculated anomaly score shows outliers at the position of extrema in the input sequence. This might be caused by the decimation of sequences during preprocessing which adds noise to these extrema.

These outliers in the anomaly score of normal sequences can be significantly reduced by applying a low-pass filter on the score signal. For this purpose, a *Butterworth* filter of $6th$ order was used with the cutoff frequency $f_c = 10$ and a sampling frequency of $f_s = 100$. The results of the filter are presented in the lower graph of figure 3.7. The filtered anomaly score efficiently ignores outliers but rather puts an emphasis on patterns that span over multiple time steps.

The reconstruction of the anomaly example in figure supports the intuition 3.7 of the autoencoder model. The autoencoder is not able to reconstruct previously unseen structures in the input sequence. However, the reconstruction does not relate to the otherwise expected characteristics of the normal sequences. In addition, it is not possible to determine the initial time step of the anomaly, as the entire reconstruction is noisy. The interpretation of the reconstruction is therefore difficult and does not help to understand the occurring anomaly.

The anomaly score is used to label the sequences in the test set. As soon as a sequence contains a score $a^{(i)}$ that is higher than the anomaly threshold, the sequence is labeled as anomalous. The following *confusion matrix* in table 3.6 shows the results of anomaly detection on the test sequences. The model achieved an accuracy of 89.71% and a macro F_1-score of 78.59%.

The results shown in the *confusion matrix* prove, that the model is able to encode normal sequences from the dataset and reconstruct them sufficiently. The anomaly score can be used to distinguish normal data from sequences with anomalous pattern, which have not been seen during training. However, the number of tested anomalies is very limited, as only relevant failures have been considered. Further experiments with more labeled anomalies are necessary for a sufficient estimation of the detection quality.

The reconstruction does not provide reliable information about the nature of the anomaly. For instance, the anomaly cannot be traced back to the specific sensor channel it occurred in.

In addition, the reconstruction of entire cycles does not allow detecting anomalies in a timely manner. However, the model can be used for anomaly detection in already recorded cycles.

Training on Sensor Channels

For the following experiments, the autoencoder model was extended to tackle the downsides of the previously used architecture. First, the full sensor channels were fed as input to the model combined with the auxiliary status channels. Hence, the full sensor channels were also reconstructed by the autoencoder. Similar to previous experiments, the status channels were provided to the decoder network as well. This proved to yield good results in the previously conducted experiments.

In addition, the window size is reduced to span over 1 minute of sensor data. Sequences have been taken from the datasets 1, 2 and 3. Again, the normal data was split into 90% for training and 5% for validation and test split respectively. A total of 110877 samples were used for training. These samples were extracted from the dataset by applying a sliding window function with a length of 100 and a step size of 50 on the training data. This results in 1 minute of recorded sensor data per window, as the sequences are sampled down to 100 data points per minute.

In order to validate the model on more anomalous samples, synthetic anomalies were generated based on the characteristics of the labeled failure cases. Hence, a blockage of the drying fan was simulated amongst other similar scenarios. A total of 111 synthetic anomalous samples have been constructed for the test split and the validation split.

The autoencoder architecture has so far shown reasonably good results for the detection of anomalous samples on this specific dataset. However, to evaluate the quality of this approach, it is necessary to compare the results with a baseline algorithm for anomaly detection. Here, a *One-Class SVM* (OC-SVM) is used as baseline. OC-SVMs are a popular approach for the unsupervised modeling of density distribution of normal data for the detection of novel or anomalous patterns. Therefore, the algorithm is considered as baseline in several recent publications in order to compare and evaluate the quality of new approaches for anomaly detection (Thomas et al., 2016; Goix, 2016; Schlegl et al., 2017).

For this reason, an OC-SVM model with a *Radial Basis Function* (RBF) as kernel has been trained and tested on the same experimental setup as the autoencoder models. The OC-SVM was hence trained on 110877 samples. However, the OC-SVM requires a one-dimensional feature vector per example. Therefore, the extracted windows where concatenated and each of the resulting 11087700 vectors of features per time step have been fed separately for fitting the model.

Table 3.7: OC-SVM experiments on classification of anomalous samples within the sensor channels. The metrics have been calculated w.r.t. the anomaly class. A *macro* weighting scheme was used to take the imbalance of classes into account.

ν	γ	ACCURACY	PRECISION	RECALL	F1-SCORE
1E-7	6.67E-2	0.9823	0.7423	0.5664	0.6019
5E-7	6.67E-2	0.9828	0.7701	0.5666	0.6043
1E-6	6.67E-2	**0.9830**	**0.7923**	**0.5668**	**0.6060**
5E-6	6.67E-2	0.9828	0.7701	0.5666	0.6043
1E-5	6.67E-2	0.9826	0.7602	0.5665	0.6035
5E-5	6.67E-2	0.9799	0.6626	0.5696	0.5962

The OC-SVM was trained multiple times by tuning the parameter ν, which accounts for the fraction of outliers during training as well as the fraction of support vectors. After training on a specific parameter ν, the quality of the model has been calculated with different measurements on the validation data. The results are presented in table 3.7. In order to predict an anomaly score for a given test sequence, each step-wise feature vector in the sequence has been used for anomaly prediction separately. The entire sequence was then labled as anomaly, once at least one time step was detected to be anomalous by the OC-SVM. The best model was selected based on the validation results shown in table 3.7.

Extensive experiments were conducted to evaluate a good architecture for the LSTM-based autoencoder. Table 3.8 shows the training loss of several architectures that have a varying number of recurrent layers in the encoder and the decoder network. In addition, the number of hidden units per LSTM cell and the use of a bidirectional LSTM as the first encoder layer has been evaluated through experiments. All architectures have been trained for 1000 epochs over the training data. The *RMSProp* optimizer function was used with a learning rate of 1×10^{-3} (without decay) and the standard hyper-parameters $\rho = 0.9$ and $\epsilon = 1 \times 10^{-8}$. These parameters achieved satisfying results during the training of the model. The size of the mini-batch was set to 1024. This value was chosen to maximize the GPU utilization in order to speed up the training. Experiments on the mini-batch size are presented in table 3.10.

The results in table 3.8 show that deeper encoder and decoder networks as well as a bidirectional encoder reduce the reconstruction loss of normal sequences during training. Another interesting fact is that architectures with only a single encoder layer but multiple stacked decoder layers show similar reconstruction capabilities compared to overall deeper models. The training

Table 3.8: Comparison of reconstruction loss during training of different architectures. The number of hidden units N is shared for both encoder and decoder layers. Training was done for 1000 epochs with a mini-batch size of 1024.

N	$L_{Encoder}$	$L_{Decoder}$	Bidirectional	Training Loss
32	1	1		16.2896 E-3
32	2	2		12.6126 E-3
32	2 + 1	2	✓	11.8259 E-3
32	2 + 1	3	✓	11.6223 E-3
64	1	1		10.8458 E-3
64	1	2		7.2063 E-3
64	2	2		7.3527 E-3
64	2 + 1	2	✓	7.2063 E-3
64	2 + 1	3	✓	6.5211 E-3
128	1	1		6.5671 E-3
128	1	2		5.0472 E-3
128	1	3		4.2206 E-3
128	2	2		4.9640 E-3
128	2 + 1	2	✓	4.2685 E-3
128	2 + 1	3	✓	**3.5897 E-3**

of deeper architectures increased the training time constantly, so that experiments with larger networks has been out of scope on the available hardware.

Based on these findings, a LSTM-based autoencoder with 128 hidden units per LSTM cell, a bidirectional encoder with another additional stacked layer and a decoder network with at least two layers has been chosen for further experiments.

For the evaluation of the optimizer function, several experiments have been conducted for comparison of *RMSProp* and the *Adam* optimizer. Both methods are currently very popular in similar deep learning use cases. Table 3.9 lists the experiments with two different models that have been trained using either *RMSProp* or *Adam*. The hyper-parameters for training through *RMSProp* are identical to the previous experiments. For *Adam*, a learning rate of 1×10^{-3} (without decay) has been used along with the hyper-parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. This has been proposed by Kingma and Ba (2014) as a reference and produced satisfying results in this application.

The results of the experiments on the two different optimizer functions show that *Adam* outperforms *RMSProp* in terms of reconstruction loss during training and validation. In contrast, both methods show a similar training time for 1000 epochs. It can be noted, that the optimizer function does not contribute to the overall success of the model, as both optimizer functions can

Table 3.9: Comparison of reconstruction loss during training and validation of different optimizer functions. The number of hidden units N is shared for both encoder and decoder layers. Training was done for 1000 epochs with a mini-batch size of 1024.

N	$L_{Encoder}$	$L_{Decoder}$	OPTIMIZER	TRAIN LOSS	VAL LOSS	TRAINING TIME
128	2 + 1	2	RMSPROP	4.2685 E-3	1.2438 E-2	**17H 38M**
128	2 + 1	3	RMSPROP	3.5897 E-3	1.3077 E-2	24H 27M
128	2 + 1	2	ADAM	2.0208 E-3	**8.8736 E-3**	19H 10M
128	2 + 1	3	ADAM	**2.0060E-3**	9.7012 E-3	23H 04M

Table 3.10: Training of an autoencoder model with a single layer encoder and decoder network with 64 hidden units N per layer. The models were each trained for 100 epochs but with different mini-batch sizes. The final loss after 100 epochs of training and the duration of the training were recorded for comparison. The best results per column are highlighted in bold font.

N	L	BATCH SIZE	OPTIMIZER	TRAIN LOSS	TRAINING TIME
64	1	64	ADAM	**1.3262 E-2**	4H 51M
64	1	128	ADAM	1.4282 E-2	2H 25M
64	1	256	ADAM	1.3858 E-2	1H 17M
64	1	512	ADAM	1.6378 E-2	0H 43M
64	1	1024	ADAM	1.7647 E-2	**0H 34M**

be used to train the autoencoder model for anomaly detection. Nevertheless, *Adam* was selected for the following experiments.

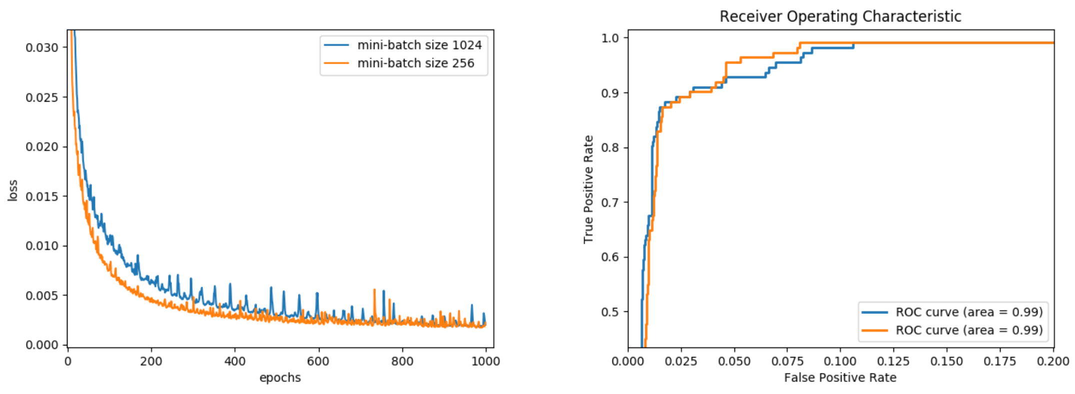
The size of the mini-batch can be increased to efficiently train on as much data as possible, while using the full capacity of e.g. GPUs for parallel computing. However, choosing a large mini-batch size can have a negative effect on training, as stochastic gradient descent exploits the noisy gradient through optimization of the cost function based on only a fraction of the training data per parameter update. Hence, experiments are conducted to evaluate the effect of the mini-batch size for the autoencoder training.

A simple autoencoder model with only a single layer in the encoder and decoder network was trained on varying mini-batch sizes. All the architectures were implemented with 64 hidden units per LSTM cell and trained through *Adam* for 100 epochs. To evaluate the effect of the mini-batch size, the final training loss and the training time are compared in table 3.10.

Table 3.11: Training of the deepest model based on previous results with different mini-batch sizes. The decoder network contains 3 LSTM layers. Both networks were trained with equal optimizer parameters.

N	L	BATCH SIZE	OPTIMIZER	TRAIN LOSS	VALIDATION LOSS	TRAINING TIME
128	3	256	ADAM	2.2234 E-3	1.2422 E-2	38H 30M
128	3	1024	ADAM	**2.0060E-3**	**9.7012 E-3**	**23H 04M**

The comparison of different batch-sizes shows that a smaller batch size improves the training loss. However, the training time is also increased significantly. To investigate this further, a deeper architecture was trained for 1000 epochs on a mini-batch size of 256 and 1024 using *Adam*. The results are presented in table 3.11. Training on a larger mini-batch size proved to significantly reduce the training time by better utilization of the GPU. In addition, the training and validation loss is improved in case of the larger mini-batch size. However, the training loss curves of both models presented in figure 3.8 shows that both models converge to a similar value. The loss curves further proves that the mini-batch size does influence the training in the beginning, as a smaller batch size shows to faster reduce the training loss. However, both models converge to a similar level during the later stages of the training.



(a) Training Loss

(b) ROC Curves

Figure 3.8: Training loss over 1000 epochs and ROC curve of the deepest autoencoder model with mini-batch sizes of 256 (orange) and 1024 (blue).

From these experiments, a promising architecture was chosen and tested on the test set with synthetic anomalies. The final architecture included a bidirectional layer in the encoder network

Table 3.12: OC-SVM in comparison to the selected autoencoder model.

MODEL	ACCURACY	PRECISION	RECALL	F1-SCORE
OC-SVM	0.9830	0.7923	0.5668	0.6060
AUTOENCODER	0.9864	0.7612	0.9121	**0.8186**

with an additional LSTM layer on top. The decoder was implemented with three stacked LSTM layers. The network was trained for 1000 epochs on a mini-batch size of 1024 examples using *Adam* with standard hyper-parameters, as described previously. Each LSTM cell in the model contained 128 hidden cells. Hence, the latent representation between encoder and decoder is a vector of size 128.

The anomaly threshold was selected during validation to maximize the macro F_1-score. With this configuration, the LSTM-autoencoder model has been compared to the baseline OC-SVM. Table 3.12 shows a comparison of different metrics calculated on the test data. The autoencoder shows a significantly higher macro F_1-score, compared to the baseline OC-SVM. Instead, the baseline model seemed to overfit on the normal data. This could be even better observed by calculating precision and recall for the anomaly class without macro weighting. However, the results do not surprise, as the autoencoder models temporal dependencies in the sequences, whereas the OC-SVM was trained to predict anomalies for single time steps.

The full potential of the autoencoder model is revealed by visualizing the reconstruction and the anomaly score. Figure 3.9 shows the reconstruction and anomaly score for a synthetic anomaly and the original sequence respectively. Similar to previous experiments, a low-pass filter was applied to the anomaly score. The autoencoder reconstruction of the normal sequence is visually identical to the original. In contrast, the reconstruction of the synthetic anomaly shows the expected pattern, learned by the model. This results in an obvious reconstruction error which yields a high anomaly score. Compared to the reconstruction of reduced sensor channels, this approach allows interpretation and a clear identification of the occurring failure.

The experiments on LSTM-based autoencoder models for anomaly detection resulted in a system that can detect anomalies in multivariate sensor data. This has been established for both, decimated sequences that contain an entires cycle as well as small windows over a short period of the cycle. Reconstruction of all the relevant sensor channels allows to interpret occurring patterns that were not seen during training.

However, further experiments on more real failure cases are needed to validate the findings of the experiments presented in this section.

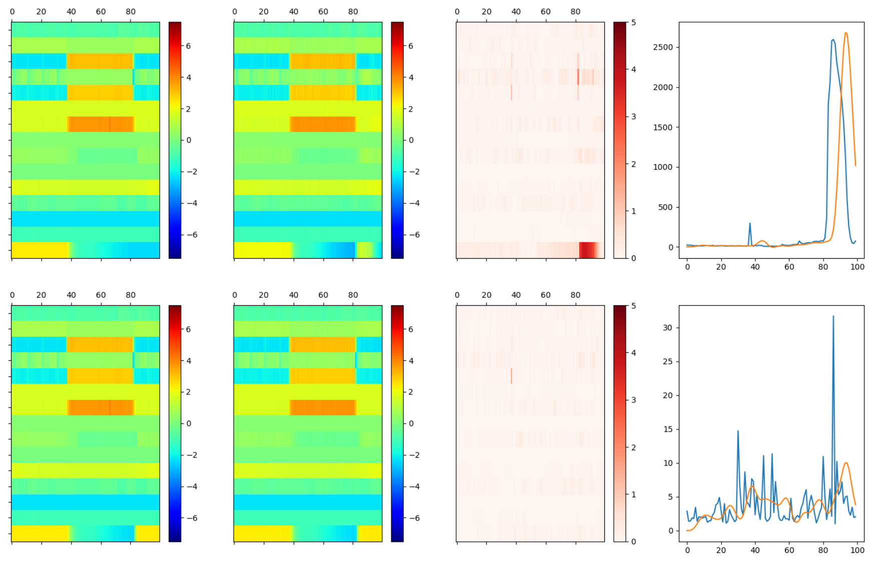


Figure 3.9: Reconstruction of a synthetic anomaly in comparison to the original sequence. The upper row shows anomaly detection in a sequence with a synthetic anomaly, whereas the original sequence is visualized in the bottom row. Each row displays the input sequence, the reconstruction, the absolute error and the anomaly score.

3.4.3 GAN Experiments

For the experiments with different GAN architectures, the available data has been preprocessed according to the defined preprocessing pipeline described in section 3.2. However, the sequences have been limited to the sensor channels only. Similar to the reconstruction of sensor channels in the previously described autoencoder experiments, the generator network of the GAN was trained to produce example of solely the relevant sensor channels. This reduces complexity by discarding the remaining channels.

Experiments where conducted on two basic experimental settings. First, a GAN was trained for modeling the first principle component of the sensor data from entire cycles. This allowed to evaluate the generative capabilities of the model, as the reduced sequences from the sensor data show distinct characteristics that can easily be recognized. Secondly, a GAN was trained to generate short windows of all relevant sensor channels. Hence, the generator network learns to model the underlying distribution of the normal data.

For the training process of the GAN, sensor sequences from datasets 1, 2 and 3 have been merged. For the first experiments, the first principle component of the sensor channels was calculated. These sequences where decimated to 5 datapoints per minute, so that the generation of an entire cycle could be learned by the generative model. For the subsequent experiments, the sensor channels were extracted without reduction. The sequences were decimated to 100 data points per minute and segmented through a sliding window function of size 786. Similar to the previous experiments, these sequences were exclusively taken from a specific dying program. This allowed including the synthetic anomalies constructed for the previous experiments.

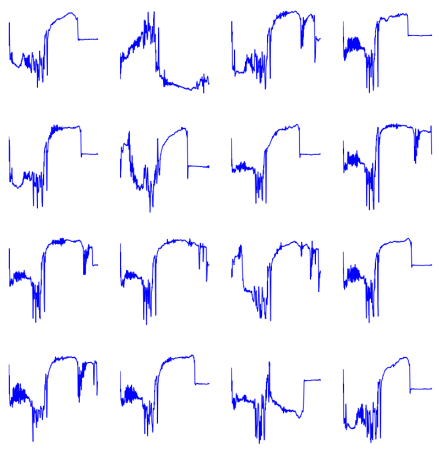
The available data has been split into training, validation and test split in the same manner as used for the experiments in section 3.4.2. The training set contained solely normal sequences. The validation and test set contain normal as well as anomalous examples.

Discriminator and generator network were trained in an adversarial fashion. For every training step, the discriminator was trained on a mini-batch of either real or generated data. This was evaluated in every training step by random choice. The real data was taken from the training set of normal sequences. After each step of discriminator training, the generator was trained to fool the discriminator. This was done by generating samples and feeding these samples into the discriminator network. The discriminator loss was subsequently used to update the generators weights.

The hyper-parameters of the GAN training were evaluated empirically. Both networks were trained using the *RMSprop* optimizer. For the discriminator training, the optimizer was used with a learning rate of 2×10^{-3} and a decay of 6×10^{-8}. In contrast the generator network was trained with a learning rate of 1×10^{-3} and a decay of 3×10^{-8}. The *LeakyReLU* activation functions were used with the parameter $\alpha = 0.2$ and *BatchNormalization* was applied with a *momentum* value of 0.9. Dropout in both networks was used with a probability of 0.4. Finally, the adversarial training was done for up to 50.000 steps and evaluated by sampling the generator network. The anomaly detection capability was then evaluated by utilizing the discriminator as a classifier for categorizing the normal an anomalous examples in the validation set.

Figure 3.10 visualizes the generated sequences after training two GAN-architectures for the reduced as well as all the sensor channels. These examples were obtained by randomly sampling the latent variable z and feeding it into the generator network.

The generated samples displayed in figure 3.10 prove that the selected GAN-architecture can be trained to generate samples of both, the reduced as well as the full sensor channels. Hence, the GAN can learn to generate multivariate sequences of arbitrary but predefined length for this specific domain.


(a) PCA Samples

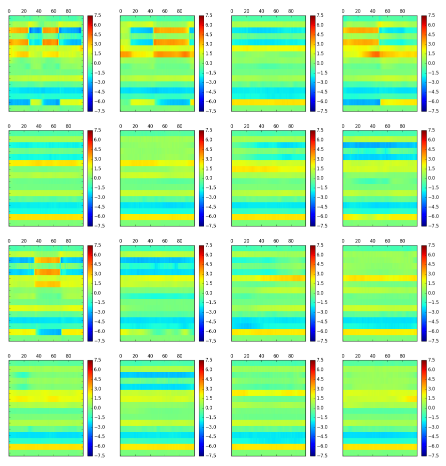

(b) Full Samples

Figure 3.10: Samples produced by the generator network through randomly sampling from the latent space.

However, experiments for both settings show that the discriminator network cannot be easily used for anomaly detection. The anomaly detection accuracy of the discriminator after adversarial training did not compete with the results of experiments on LSTM-autoencoders presented in the previous section. This can be attributed to an overfitting of the classification model on very specific characteristics of the normal data. Schlegl et al. (2017) solve this by iteratively sampling from the generator and subsequently comparing an anomalous example to the most related generated sample. Nevertheless, the experiments for this use case aimed to utilize solely the discriminator to avoid an anomaly score function that introduces additional hyper-parameters to the detection model.

Further experiments have to be conducted with possible optimizations of the discriminator network for classification of anomalies. However, these experiments are out of scope for this thesis and will therefore be considered in future work.

3.5 Discussion of Results

The experiments presented in the previous sections showed two general approaches towards anomaly detection. The autoencoder model has been used to reconstruct sequences and calculate an anomaly score based on the reconstruction error. Trained on normal sequences, the model proved to work for the detection of anomalous patterns in multivariate sensor data of varying length. In contrast, the GAN-based approach to anomaly detection learned a generative model for nomal sequences from the training data. However, classification of previously unseen anomalous examples by the discriminator network did not yield satisfying results without further modification of the architecture.

3.5.1 Autoencoder-based anomaly detection

The autoencoder models proved to sufficiently detect anomalies for the given use case. The experiments also demonstrated the benefit of certain architectural considerations or relevant findings regarding the preprocessing of the dataset.

One of the crucial findings of the experiments with autoencoder models is the benefit of incorporating status channels as auxiliary features. These were included into the input to the encoder network but also provided to the decoder network for reconstruction of reconstruction of the input. The experiments showed overall better results for reconstruction as well as anomaly detection by including the status channels.

The first experiments where conducted on an autoencoder model, which was trained on entire cycles as input sequences. The cycles where therefore sampled down to a few hundred data points. Dynamic bucketing and padding of sequences within a bucket allowed to efficiently train on sequences with varying length.

Selecting the window length over the data also defines how fast anomalies can be detected by the system. This largely depends on the specific task at hand. The most prominent task for anomaly detection is to immediately detect unusual pattern that might be connected to mechanical or electrical malfunctions. This might allow preventing fatal damage to the washer-dryer machines. In addition, the analysis of the specific failure might require the engineers to observe optical or acoustical features of the machines, that are not captures in the sensor data. This can only be done immediately during the occurrence of the failure or afterwards through e.g. recorded video footage.

An increasingly small sliding window over the sensor data, combined with a very small decimation factor or no downsampling at all allows the autoencoder model to calculate anomaly scores in a timely manner. The smallest time window used in the experiments spans one minute of sensor data with a step size of 30 seconds. Hence the anomaly score is calculated for every 30 seconds for the past one minute of data.

3.5.2 GAN-based anomaly detection

The experiments with GANs demonstrated the effectiveness of adversarial training to learn a generative model on a specific data distribution. The DC-GAN architecture was successfully implemented to train a model for the generation of normal data in the multivariate sensor setting. This can be seen as a crucial advantage compared to the autoencoder model. The trained model can be easily evaluated to obtain an understanding of what the model effectively learned as normal data. This is a very intuitive approach to anomaly detection, as the understanding of normal data can be used to compare generated sequences to potential anomalous examples.

Another interesting aspect is the discriminator network, which the GAN natively incorporates as a classification model to distinguish real from fake data. This can be translated to the problem of anomaly detection by applying the discriminator for the classification of anomalies as fake examples. In theory, this renders any form of manually designed anomaly score obsolete. Instead of predicting a scalar value for an input sequence, the discriminator could be also implemented as a sequence-to-sequence model to predict an anomaly score per time step, compared to the model proposed by Rajpurkar et al. (2017).

However, the experiments on GANs presented in the previous section found, that the discriminator cannot be used directly for anomaly detection. This has to be evaluated further in future work. The potential of GANs for anomaly detection lies in the supervised training of the discriminator on normal as well as known anomalous examples.

4 Conclusion and Future Work

This thesis applied deep learning to anomaly detection in multivariate sensor data from washer-dryer machines. Different approaches have been selected from recent publications that showed promising results on similar tasks.

Anomaly detection is defined as detecting patterns in data that have not been seen before. Hence, a deep learning algorithm can learn to model the underlying distribution of normal data in an unsupervised way. This allows predicting anomalies based on the similarity of new data to this expected distribution.

In contrast, once a sufficient amount of failures is available in the dataset and only these failures are of interest, a simple classification model can be trained for pattern recognition in the multivariate sensor data. This cannot be done for the data available in this thesis, as the amount of relevant failure cases is very limited. Hence, approaches for unsupervised learning were considered.

Finally, an LSTM-based autoencoder model was selected for anomaly detection. The model learns to reconstruct normal sequences with high accuracy. An anomaly score can be calculated based on the reconstruction error of the model. Anomalous patterns showed to result in a significantly higher anomaly score. It is therefore concluded, that the autoencoder model can be used to detect previously unseen patterns in the multivariate sensor data.

However, the reconstruction error does not necessarily allow interpreting the anomaly in terms of channel origin and the moment of occurrence. This is attributed to the model trying to make sense out of the input data. Hence, the reconstructed sequence does not relate to the expected normal behavior.

In addition, the model only allows training on normal data. In practice, many constraints on sensor data can be derived from the domain. Incorporating this knowledge into the anomaly detection would potentially improve the detection of unwanted patterns.

The experiments on autoencoder models also found that post-processing of the anomaly score is useful to filter out peaks that are related to information loss through decimation of the input sequences.

These drawbacks can be solved by using a GAN-based model for anomaly detection. The adversarial training allows implementing a predictive model, that can be used to sample normal sequences. The model therefore gives a good intuition of the expected patterns in the data. The discriminator model is trained to distinguish normal from anomalous examples in a supervised manner. This allows to easily incorporate any other labeled examples available. Known failures or synthetically generated anomalies can thus be added to the training set for the discriminator training. The discriminator model also directly predicts an anomaly score. Hence, post-processing is not needed in theory.

However, the limited experiments on a GAN-based model conducted for this use case proved that the discriminator cannot be used for anomaly detection due to overfitting on normal sequences in the adversarial training. Schlegl et al. (2017) proposed to sample similar normal examples from the generator network and calculate an anomaly score based on the differences to the actual input combined with the discriminator prediction. This approach again introduces additional complexity to the architecture. Further experiments are needed to evaluate methods to solely detect anomaly based on the discriminator prediction.

The following section 4.1 lists reasonable steps that can be done to further examine the findings of this thesis and eventually improve the anomaly detection capabilities for the washer-dryer sensor dataset. In addition, two approaches are presented for the application of the LSTM-based anomaly detection algorithm to add business value.

4.1 Future Work

The experiments show valuable insights for applying deep learning for anomaly detection on multivariate sensor data from the given dataset. However, many more technical extensions can be considered to both, the LSTM-based autoencoder as well as the GAN.

Another important step is to make the best performing model available for integration into a practical application.

Technical Improvements

The dataset for training and validating the anomaly detection model has been limited with regard to several aspects. First, the overall number of available sequences that could be used for this work has been reduced by, e.g. random inconsistencies within the data channels which rendered a large portion of the data useless. Further, the dataset has been limited in terms of labeled anomalous samples that could be used for validation of the chosen architectures. For future work the dataset can be extended to many more long-term test datasets. Training on more data can in general improve ability of the algorithm to generalize.

Given a sufficient amount of training data, deeper architectures can be trained. Adding more layers to a deep learning model can potentially improve the models ability to extract meaningful representation from complex data domains. To efficiently train a deep architecture with many more trainable parameters, requires more capable hardware. This is especially important to reduce the training time, and thus allow to conduct more experiments for hyper-parameter tuning in a timely manner.

Deep architectures may also require to incorporate residual connections between the network layers. This enables training without suffering from vanishing gradients. Good examples for deep architectures can be easily found in recent publications (Wu et al., 2016; Kalchbrenner et al., 2016; Rajpurkar et al., 2017; Gehring et al., 2017).

Normalizing the inputs of layers for each mini-batch in a deep architecture can speed up the training-time of feedforward neural networks significantly (Ioffe and Szegedy, 2015). This has been used here for the experiments with GANs. As an extension to this concept, Ba et al. (2016) introduced *Layer Normalization*, which can be applied to recurrent neural networks as well. However, the method was discarded in the autoencoder experiments here, due to the increased computational complexity without any noticeable improvement in training time. Further experiments are needed to evaluate the layer-wise normalization for this application.

Scheduled Sampling (Bengio et al., 2015) has been applied for the training of the LSTM-based autoencoder. In theory, this should allow speeding up the training of a sequence-to-sequence model by partly providing the decoder with ground truth data. However, this method did not yield good results in experiments conducted in this thesis.

Professor Forcing (Lamb et al., 2016) is an extension aiming at improving long-term sequence modeling. To achieve this, the architecture is extended by a discriminator model in a similar fashion as proposed by Goodfellow et al. (2014) for GANs. The discriminator forces the distribution of the hidden states of the decoder with applied *Teacher Forcing* to be equal to the free running decoder which does not apply the method. This can potentially improve the otherwise disappointing results from the experiments on *Teacher Forcing* conducted in this thesis.

However, the most interesting future work on the autoencoder model is the evaluation with more and different machine failures. This might require to incorporate more sensor channels or adjust the window and architecture size. Smaller windows on higher resolution sensor data might be used to detect ongoing anomalies, whereas a larger window of dynamical size can be used to detect anomalies in decimated but entire cycles. For instance, Rajpurkar et al. (2017) demonstrate efficient sequence modeling for very large input sequences.

As the experiments on GAN models did not yield satisfactory results yet, more experiments will be conducted on this otherwise promising architecture. For instance, the combined anomaly score proposed by Schlegl et al. (2017) can be implemented. However, this does not improve the drawbacks from the LSTM-autoencoder, where the anomaly score is prone to outliers in the reconstruction, and is therefore filtered by a low-pass filter. Instead, future work should focus on improving the discriminator for anomaly detection. This may be achieved without introducing complexity through additional steps, and hence eliminating the need for, e.g. a filter on top of the score function as used for the autoencoder model.

Conditional GANs can be used to generate samples that are conditioned on the available status information for a given time window. This can be seen as an image-to-image translation task, where the multivariate status information is seen as the input image and the expected normal sensor data is the output. A similar approach has been successfully applied by Isola et al. (2017) for the translation of, e.g. sketches to realistic images. This enables to train a model that learns to generate different normal data conditioned on the specific state the machine is in. Hence, different washing programs can be effectively modeled by a single architecture

Generating sequences of different length is another challenge for applying GANs to sensor data. This may be beneficial, once a generative model for entire cycle sequences is desired. It has been already shown, that LSTMs can be successfully used in a GAN-approach architecture (Im et al., 2016), hence it is reasonable to experiment with a similar approach.

A core concept behind deep learning is to automatically learn useful representations of the data at hand, without the imminent need of a domain expert manually deriving rules from experience. In this thesis however, there have been a few manually engineered steps taken in order to extract useful features from the data samples. These data-related, manually coded steps took up a majority of the work which was done to implement the anomaly detection pipeline. Hence, replacing these steps by learning methods that can be trained end-to-end with the auto-encoder network can potentially reduce the manual work, and therefore the time to transfer or optimize this framework on a similar dataset.

There are several other additions to stabilize the training of GANs. Sønderby et al. (2016) found that additional instance noise applied to generated as well as real data improved the stability of GANs during training. Decimation and up-sampling may be entirely learned by the model itself. For instance, the simple up-sampling method used in the GAN experiments can be replaced by sub-pixel convolutional layers (Shi et al., 2016). Also, experiments can be conducted on novel activation functions, e.g. the *SELU* activation function introduced by Klambauer et al. (2017).

However, the most interesting and outstanding advantage of GANs for anomaly detection is the use of labeled failure cases for training. GANs can be trained even with very few anomaly examples, as they can be mixed with generated examples during discriminator training. Still, the discriminator can be tailored to detect already known anomalies. This allows formulating the otherwise solely unsupervised learning task as semi-supervised learning.

A positive aspect about using a GAN for anomaly detection that stands out, is the ability to add all available examples to the training set for the discriminator.

Finally, future work will not be limited to LSTM-based autoencoder or GAN architectures. Recent literature shows the possibility to use solely attention mechanisms for sequence-to-sequence modeling (Vaswani et al., 2017). In addition, Kaiser et al. (2017) propose a universal model based on attention, that can be used for several different tasks. Observing new publications to the field of sequence modeling and evaluating the potential to apply new approaches will be crucial for further research on anomaly detection with deep learning.

Application in a Production System

The autoencoder approach to anomaly detection showed promising results for this specific data domain. To add business value, the autoencoder can be implemented as an automated anomaly detection system. Two different applications can be envisioned, where such a system can improve the process of long-term tests of washer-dryer machines.

- Improving the timely detection of failures
- Reducing datasets by the selection of relevant cycles

Intuitively, the autoencoder can be tailored to the detection of ongoing anomalies in the sensor data. The model enables to detect previously unseen patterns in the sensor data. Hence, it can be trained on data from several machines and several different washing or drying programs. Alternatively, the training can be specialized only on a certain machine or program. Evaluation can be done by either testing on relevant failure cases, or synthetic failures that are expected to be detected by the system. In any case, sufficient domain knowledge is required for the evaluation.

The model can be trained on the first few hundred cycles of a certain machine in order to detect anomalies in the remaining washing and drying cycles. Alternatively, the model can be pre-trained on all available data form different machines and subsequently applied to a new machine. The generalization capability over multiple machines largely depends on the variation of characteristics of these different machines. The experiments conducted in this thesis showed a sufficient generalization over different machines for the extracted channels and the type of anomalies in focus. However, further experiments have to be conducted to validate this.

The sliding window and the decimation factor can be adjusted according to the desired use case and response time. Experiments on window sizes with up to 500 data points produced good results. Even with a large window size, the step size can be reduced to allow the detection of ongoing anomalies in a timely manner. Other hyper-parameters, such as the anomaly score threshold, can be determined to maximize, e.g. recall or precision of the detection model.

The experiments proved the general ability to detect unknown patterns. Nevertheless, further experiments are necessary for hyper-parameter tuning in order to achieve the best results for the desired use case. This implies, that sufficient data is available.

The second possible application aims to reduce the amount of data that is recorded during the test processes. Long-term testing of machines produces a massive number of cycles that have to be evaluated eventually. Especially in a highly multivariate data setting with high-frequent sensor data, tools are necessary to reduce the amount of data that has to be manually evaluated by engineers.

Cycle data that contains failures is of special interest to the test engineers, as the sensor data may show why or how a failure occurred. Hence, an automated preliminary selection of the most relevant cycles may reduce the manual work of examining and identifying important sections in the data.

To achieve this, an anomaly detection system can be paced on top of a data enrichment pipeline. Figure 4.1 presents the conceptual approach of embedding the deep learning system into a pipeline for data enrichment. Sensor data is stored in an arbitrary persistence layer which may be implemented in form of a distributed storage system to efficiently handle large amounts of data.

The sensor data is stored in a suitable file format, e.g. the open *TDMS* format which allows storing meta information related to the sensor data. The data enrichment pipeline consists of multiple steps were preprocessor systems calculate metrics, statistics or transformations of the sensor data, such as *FFTs* and add these to the meta data of the data files.

Hence, the deep learning system for anomaly detection is implemented as one of these processing steps for data enrichment. Technology-dependent data conversion and preprocessing is done, before the sensor data is fed into the anomaly detection algorithm to either train the algorithm or compute an anomaly detection score. Preprocessing can be similar to the pipeline implemented in this thesis.

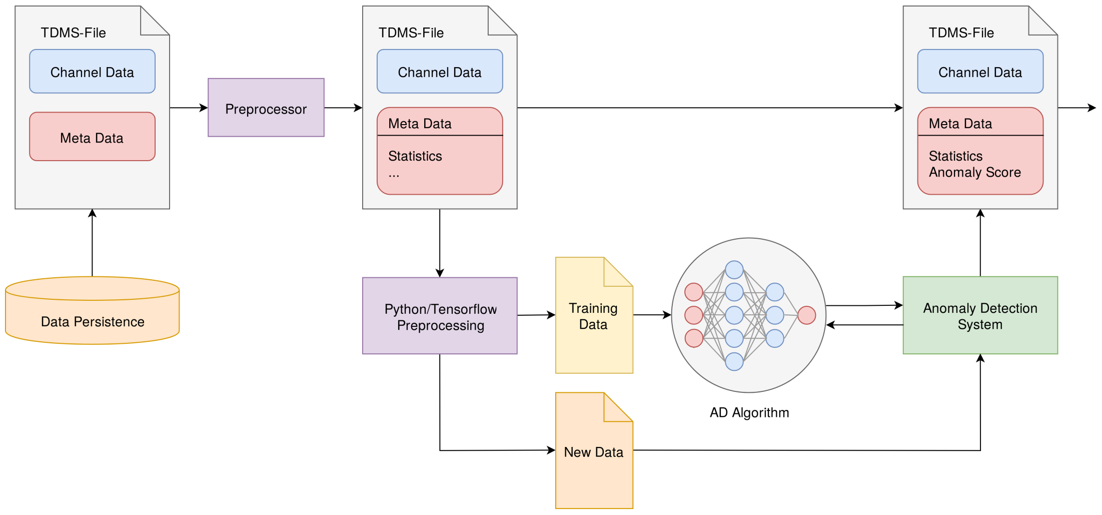


Figure 4.1: Embedding of an anomaly detection system into a data enrichment pipeline.

The anomaly detection algorithm can be tuned to predict normal cycles with high precision. Recall can be disregarded, as it is more crucial to split the processed cycles into certainly normal on the one side, and maybe normal or maybe anomalous cycles on the other side. Enriching the cycle data with such an anomaly score allows filtering out the certainly normal sequences, and thus reduce the amount of cycles that have to be evaluated by the test engineers.

These two approaches demonstrate the incorporation of an anomaly detection algorithm into the test process. This can potentially increase the capability to detect relevant pattens in the sensor data and reduce the manual work by engineers.

In general, an anomaly detection algorithm can be beneficial for several other applications. Future potential can also be seen by including such an algorithm into machines, that are stationed in the homes of consumers. The algorithm could detect unusual behavior of the machine and contact the customer service directly. The manufacturing industry could thus optimize the maintenance cycles and eventually replace crucial mechanical parts, before a fatal failure appears. This not only benefits the customer, but also reduces the service costs of the manufacturer.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA*.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bayer, J. and Osendorfer, C. (2014). Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179.

Britz, D., Goldie, A., Luong, T., and Le, Q. (2017). Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*.

Chan, T. F., Golub, G. H., and LeVeque, R. J. (1983). Algorithms for computing the sample variance: Analysis and recommendations. *The American Statistician*, 37(3):242–247.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Engel, J., Resnick, C., Roberts, A., Dieleman, S., Eck, D., Simonyan, K., and Norouzi, M. (2017). Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders. *ArXiv e-prints*.

Fayyad, U. M., Piatetsky-Shapiro, G., and Smyth, P. (1996). Advances in knowledge discovery and data mining. chapter From Data Mining to Knowledge Discovery: An Overview, pages 1–34. American Association for Artificial Intelligence, Menlo Park, CA, USA.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*.

Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural Computation, 12(10):2451-2471*.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323.

Goix, N. (2016). How to evaluate the quality of unsupervised anomaly detection algorithms? *arXiv preprint arXiv:1607.01152*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Görnitz, N., Kloft, M. M., Rieck, K., and Brefeld, U. (2013). Toward supervised anomaly detection. *Journal of Artificial Intelligence Research*.

Graves, A., Jaitly, N., and Mohamed, A.-r. (2013). Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

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

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

Im, D. J., Kim, C. D., Jiang, H., and Memisevic, R. (2016). Generating images with recurrent adversarial networks. *arXiv preprint arXiv:1602.05110*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2342–2350.

Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., and Uszkoreit, J. (2017). One model to learn them all. *arXiv preprint arXiv:1706.05137*.

Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A. v. d., Graves, A., and Kavukcuoglu, K. (2016). Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

Kim, T., Cha, M., Kim, H., Lee, J., and Kim, J. (2017). Learning to discover cross-domain relations with generative adversarial networks. *arXiv preprint arXiv:1703.05192*.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. (2017). Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*.

Lamb, A. M., GOYAL, A. G. A. P., Zhang, Y., Zhang, S., Courville, A. C., and Bengio, Y. (2016). Professor forcing: A new algorithm for training recurrent networks. In *Advances In Neural Information Processing Systems*, pages 4601–4609.

LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al. (2016). Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint arXiv:1609.04802*.

Leng, M., Chen, X., and Li, L. (2008). Variable length methods for detecting anomaly patterns in time series. In *Computational Intelligence and Design, 2008. ISCID'08. International Symposium on*, volume 2, pages 52–56. IEEE.

Liu, M.-Y., Breuel, T., and Kautz, J. (2017). Unsupervised image-to-image translation networks. *arXiv preprint arXiv:1703.00848*.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30.

Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., and Shroff, G. (2016). Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*.

Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. (2016). Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*.

Neil, D., Pfeiffer, M., and Liu, S.-C. (2016). Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*, pages 3882–3890.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016a). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. (2016b). Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*.

Ordóñez, F. J. and Roggen, D. (2016). Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1):115.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning (ICML)*, pages 1310–1318.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Rajpurkar, P., Hannun, A. Y., Haghpanahi, M., Bourn, C., and Ng, A. Y. (2017). Cardiologist-level arrhythmia detection with convolutional neural networks. *arXiv preprint arXiv:1707.01836*.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X., and Chen, X. (2016). Improved techniques for training gans. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 2234–2242. Curran Associates, Inc.

Schaul, T., Antonoglou, I., and Silver, D. (2014). Unit tests for stochastic optimization. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U., and Langs, G. (2017). Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. *arXiv preprint arXiv:1703.05921*.

Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., and Platt, J. C. (2000). Support vector method for novelty detection. In *Advances in neural information processing systems*, pages 582–588.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D., and Wang, Z. (2016). Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1874–1883.

Sölch, M., Bayer, J., Ludersdorfer, M., and van der Smagt, P. (2016). Variational inference for on-line anomaly detection in high-dimensional time series. *arXiv preprint arXiv:1602.07109*.

Sønderby, C. K., Caballero, J., Theis, L., Shi, W., and Huszár, F. (2016). Amortised map inference for image super-resolution. *arXiv preprint arXiv:1610.04490*.

Springenberg, J., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2015). Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Steinarsson, S. (2013). Downsampling time series for visual representation.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, pages 4278–4284.

Thomas, A., Clémençon, S., Feuillard, V., and Gramfort, A. (2016). Learning hyperparameters for unsupervised anomaly detection. *Anomaly Detection Workshop, ICML 2016*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280.

Williams, R. J. and Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications*, 1:433–486.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xingjian, S., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., and Woo, W.-c. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems*, pages 802–810.

Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R. (2010). Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, September 1, 2017 Jan Paul Assendorp