Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Vili Ketonen

Anomaly Detection for Injection Molding Machines Using Probabilistic Deep Learning

Master's Thesis
Espoo, December 11, 2020

Supervisor: Professor of Practice Jan Blech

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF THE MASTER'S THESIS

Author	Vili Ketonen		
Title	Anomaly Detection for Injection Molding Machines Using Probabilistic Deep Learning		
Date	December 11, 2020	**Pages**	99
Major	Machine Learning, Data Science and Artificial Intelligence	**Code**	SCI3044
Supervisor	Professor of Practice Jan Blech		

In manufacturing industries, monitoring the complicated devices often necessitates automated methods that can leverage the multivariate time series data produced by the machines. However, analyzing this data can be challenging due to varying noise levels in the data and possible nonlinear relations between the process variables, requiring appropriate tools to deal with such properties.

This thesis proposes a deep learning-based approach to detect anomalies and interpret their root causes from multivariate time series data, which can be applied in a near real-time setting. The proposed approach extends an existing model from the literature, which employs a variational autoencoder architecture and recurrent neural networks to capture both stochasticity and temporal relations of the data.

The anomaly detection and root cause interpretation performance of the proposed method is compared against five baseline algorithms previously proposed in the literature using real-world data collected from plastic injection molding machines and artificially generated multivariate time series data.

The results of this thesis show that the proposed method performs well on the evaluated multivariate time series datasets, mostly outperforming the baseline methods. Additionally, the approach had the best performance among the selected methods on providing root cause interpretation of the detected anomalies. The experiments conducted in this thesis suggest that deep learning-based algorithms are beneficial for anomaly detection in scenarios where the problem is too complicated for traditional methods, and enough training data is available. However, the amount of real-world injection molding machine data used in the experiments is relatively small, and therefore further experiments should be performed with larger datasets to obtain more generalizable results.

Keywords	anomaly detection, multivariate time series, machine learning, deep learning, variational autoencoder, recurrent neural network, injection molding
Language	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tieto-, tietoliikenne- ja informaatiotekniikan maisteriohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä	Vili Ketonen		
Työn nimi	Anomalioiden tunnistaminen ruiskuvalukoneissa todennäköisyysperusteisen syväoppimisen avulla		
Päiväys	11. joulukuuta 2020	**Sivumäärä**	99
Pääaine	Koneoppiminen, datatiede ja tekoäly	**Koodi**	SCI3044
Valvoja	Professori (Professor of Practice) Jan Blech		

Monet teollisuusautomaation laitteet synnyttävät jatkuvasti moniulotteista aikasarjadataa, jota voidaan hyödyntää näiden monimutkaisten koneiden automaattiseen monitorointiin. Tällaisen datan analysointi vaatii kuitenkin tarkoituksenmukaisia menetelmiä, jotka pystyvät käsittelemään moniulotteista dataa, jossa voi esiintyä vaihtelevia määriä kohinaa ja epälineaarisia suhteita eri prosessimuuttujien välillä.

Tämä työ esittää syväoppimiseen perustuvan menetelmän anomalioiden tunnistamiseen moniulotteisesta aikasarjadatasta, joka pystyy myös tunnistamaan havaittujen poikkeavuuksien juurisyitä, sekä analysoimaan dataa lähes reaaliaikaisesti. Menetelmä perustuu kirjallisuudessa esitettyyn menetelmään, joka käyttää variationaaliseen autoenkooderiin pohjautuvaa arkkitehtuuria, sekä takaisinkytkettyjä neuroverkkoja, joiden avulla menetelmä pystyy mallintamaan datassa esiintyvää satunnaisuutta sekä ajallisia suhteita.

Työssä esitellyn menetelmän suorituskykyä havaita anomalioita ja tunnistaa niiden juurisyitä vertailtiin viiteen erilaiseen kirjallisuudessa aikaisemmin esitettyyn menetelmään käyttäen muoviruiskuvalukoneista kerättyä dataa, sekä keinotekoisesti luotua moniulotteista aikasarjadataa.

Työn tulokset osoittavat, että esitelty menetelmä tunnistaa anomalioita hyvin testatuilla datakokoelmilla, suoriutuen useimmiten paremmin kuin vertaillut menetelmät. Tämän lisäksi työssä esitetty menetelmä oli vertailluista menetelmistä paras arvioivaan havaittujen anomalioiden juurisyitä. Työssä suoritetut kokeet osoittavat, että syväoppimiseen perustavat algoritmit ovat hyödyllisiä anomalioiden tunnistamisessa, kun ongelma on liian monimutkainen perinteisille menetelmille ja koulutusdataa on saatavilla tarpeeksi. Työssä käytetyn ruiskuvalukonedatan määrä on kuitenkin suhteellisen pieni, minkä vuoksi lisätutkimuksia tulisi suorittaa suuremmilla datamäärillä, jotta saataisiin yleistettävämpiä tuloksia.

Asiasanat	anomalian tunnistaminen, moniulotteinen aikasarja, koneoppiminen, syväoppiminen, variationaalinen autoenkooderi, takaisinkytketty neuroverkko, ruiskuvalu
Kieli	Englanti

Acknowledgements

I wish to thank my supervisor Jan Blech from Aalto University who provided me valuable guidance and feedback, thorough the thesis process. I want to also acknowledge the colleagues from ABB Wiring Accessories Oy for their collaboration and for providing me an interesting set of real-world data to examine in this thesis. It was also nice to visit the Porvoo factory and see the robots and machines operating in person. Finally, I would also like to thank my family and friends who have supported me during my academic journey.

Espoo, December 11, 2020

Vili Ketonen

Abbreviations and Acronyms

AE	Autoencoder
AI	Artificial intelligence
AIC	Akaike information criterion
ANN	Artificial neural network
AR	Autoregression
ARIMA	Autoregressive integrated moving average
BP	Backpropagation
BiRNN	Bidirectional recurrent neural network
BPTT	Backpropagation through time
CNN	Convolutional neural network
DL	Deep learning
DNN	Deep neural network
ELBO	Evidence lower bound
EVD	Extreme value distribution
EVT	Extreme value theory
FN	False negative
FP	False positive
GAN	Generative adversarial network
GPD	Generalized Pareto distribution
GRU	Gated recurrent unit
HS-trees	Half-space trees
i.i.d.	independent and identically distributed
IF	Isolation Forest
IIoT	Industrial internet of things
IMM	Injection molding machine
IoT	Internet of things
KL divergence	Kullback–Leibler divergence
kNN	k-nearest-neighbors
KPI	Key performance indicator
LODA	Lightweight on-line detector of anomalies

LOF	Local outlier factor
LSTM	Long short-term memory
MC	Monte Carlo
MCUSUM	Multivariate cumulative sum
MEWMA	Multivariate exponentially weighted moving average
ML	Machine learning
MLE	Maximum likelihood estimation
OC-SVM	One-class support vector machine
OPC-UA	Open Platform Communications Unified Architecture
PCA	Principal component analysis
POT	Peaks-Over-Threshold
RBF	Radial basis function
RCR	Root cause recall
ReLU	Rectified linear unit
RNN	Recurrent neural network
SGD	Stochastic gradient descent
SPC	Statistical process control
SSM	State-space model
SVM	Support vector machine
TBPTT	Truncated backpropagation through time
TN	True negative
TP	True positive
UCL	Upper control limit
VAE	Variational autoencoder
VAR	Vector autoregression
VARMA	Vector autoregressive moving average

Contents

Chapter 1

Introduction

With the rapid development of big data technologies and the industrial internet of things (IIoT), time series data is continuously produced in large amounts. While the data collected from these devices can be a valuable asset, efficient data analysis necessitates automated methods that require minimal human assistance. For instance, in industrial automation, modern manufacturing processes often involve complicated machines that are challenging to monitor manually. In such a context, automated data monitoring tools are essential, as they allow fast detection of possible anomalies in the complex processes. Consequently, early detection of irregularities allows to minimize downtimes and to schedule predictive maintenance actions.

Most of the produced real-world data is unlabeled. In the anomaly detection context, this usually means that there are no prior examples of anomalies but only the raw data. Therefore, supervised anomaly detection methods cannot often be used, where models could be trained with examples of anomalous data. Thus, unsupervised (and semi-supervised) methods are preferred because they require no examples of anomalies but can detect outliers in the unlabeled data by finding samples that deviate from the rest of the data.

The research conducted in this thesis is motivated by an anomaly detection application in plastic injection molding machines (IMMs) at ABB Oy Wiring Accessories. These machines carry multiple sensors, which produce multivariate time series data, consisting of several performance metrics, during the cyclic injection molding process. Monitoring the key performance indicators (KPIs) of each cycle allows for detecting potential problems in the machines for quality monitoring and predictive maintenance purposes. A commonly used strategy to detect anomalies in these machines is using simple pre-defined thresholds. When a measured value exceeds the threshold, the machine shuts down and raises the alarm. While the limit-based approaches can detect certain irregularities, they require in-depth domain

knowledge and manual refinement when the machine operating conditions change. Moreover, these approaches cannot detect anomalies that occur within the pre-defined thresholds. Hence, more sophisticated methods are in demand. The use of advanced techniques can be particularly beneficial to monitor the multivariate data produced by IMMs, where the different process variables are known to exhibit complex nonlinear correlations [18].

When monitoring a complicated process that is observed using multiple KPIs, it is often not enough to monitor each process metric separately since the anomalous behavior might only be detectable by analyzing the relationship between multiple metrics at the same time. Moreover, the sensor readings may be very noisy, making the analysis of a single metric less reliable. Therefore, this thesis focuses on multivariate anomaly detection, where the anomalies are identified directly from the multivariate observations of the time series data. However, merely detecting an anomaly in a multivariate time series is usually insufficient because it is equally important to know the root cause of the anomaly. Hence, estimating the individual features contributing to the detected anomaly is essential, as it can aid the human operators in solving the physical root cause of the detected anomaly.

The constantly evolving field of deep learning (DL), a subset of machine learning, allows building models that can directly learn from data without extensive preprocessing of the data, reducing the need for in-depth domain knowledge in the application field. When applied to anomaly detection, DL-based approaches bring various advantages. These methods work well with multivariate and high-dimensional data, eliminating or reducing the need to perform feature selection or building separate models for each input variable. Additionally, these approaches can model non-linear relationships between multiple variables within the data, which can be essential to build a robust anomaly detection model for complex data.

While many DL-based methods have been proposed in the literature, recent approaches have not been tested on injection molding data. Moreover, many prior studies have only tested their algorithms on a private dataset or compared their method only to other DL-based models. Therefore, DL-based methods need to be studied further using new datasets from different application domains, such as injection molding, and examining whether DL-based models bring advantages over traditional approaches.

1.1 Research Objectives

This thesis proposes a DL-based anomaly detection approach to detect anomalies from the multivariate time series data collected from the injection mold-

ing machines. Although this work focuses on injection molding machine data, the proposed approach should be generic and applicable to other multivariate time series data problems. Since previous studies have introduced various DL-based anomaly detection methods, the proposed approach should utilize concepts from the recent studies and attempt to improve an existing method as a research contribution. The proposed model should be unsupervised (or semi-supervised), thus not requiring examples of anomalies during training. In addition, the approach should be robust to noise and model the temporal dependencies in the data. Furthermore, the model should be suitable for a real-time setting. Hence, it should make the predictions fast enough, and it can only use a limited window of historical observations to predict the anomalousness of the observation at the current time step. Additionally, when detecting an anomaly, the proposed method should identify the individual variables of the multivariate observation that contribute most of the detected anomaly to aid in discovering the root cause of the anomaly. Finally, to evaluate the proposed method's anomaly detection and anomaly interpretation performance, the approach should be compared against various baseline methods from the literature using the collected injection molding machine data and artificially generated multivariate time series data.

1.2 Structure of the Thesis

The rest of this thesis is organized as follows. Chapter 2 provides a theoretical background on various concepts and algorithms used in the methods proposed in this thesis. After providing the essential background, Chapter 3 surveys different anomaly detection techniques for multivariate time series data and reviews studies related to anomaly detection for injection molding machines. Then, Chapter 4 presents the datasets used in the experiments of this thesis. Following that, Chapter 5 describes the proposed methods for the studied research problem. Chapter 6 explains the software used to implement the selected methods, their hyperparameter configuration, and the hardware used in the experiments. Chapter 7 evaluates the performance of the proposed approach by comparing its anomaly detection and root cause interpretation results to the selected baseline methods. Finally, Chapter 8 discusses the findings of this thesis, identified limitations and potential future directions, and then concludes the work by providing a summary of the contents discussed in this thesis.

Chapter 2

Background

This chapter provides a background of relevant concepts and algorithms used in this thesis. The chapter begins by defining the term time series. Afterward, the essence of machine learning is described, followed by an explanation of various deep learning concepts and other algorithms that the study uses in the proposed approach. Finally, the chapter provides an overview of anomaly detection in general, focusing on time series data.

2.1 Time Series

A time series is a sequence of observations ordered in time, where the data points are usually collected at regular, even-spaced intervals [77]. In general, time series data can be separated into univariate and multivariate time series. In a univariate time series, only a single variable varies over time. A simple example of a univariate time series is a sensor measuring the room temperate every second. A multivariate time series, on the other hand, is composed of multiple time-dependent variables. Besides being dependent on time, each variable may depend on the other variables with varying degrees of (possibly non-linear) correlation. From another perspective, a multivariate time series consists of multiple univariate time series, which are correlated to varying degrees. A simple example of a multivariate time series is a tri-axial accelerometer producing three-dimensional observations every second for each axis (x, y, z) [12]. Another example of a multivariate time series is KPI data measured from an IMM, which this thesis uses in the experiments. In such data, the measured variables may exhibit complex temporal correlations within each other. Having given a general description of a time series, we now provide a more formal definition:

Definition 2.1.1 (Time series)**.** A time series $X = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$ is a se-

quence of N observations, where each observation $\mathbf{x}_t \in \mathbb{R}^d$ is a d-dimensional vector at time $t \leq N$: $\mathbf{x}_t = (x_t^1, x_t^2, \ldots, x_t^d)$, and $X \in \mathbb{R}^{N \times d}$. Usually, the observations are collected at equal-spaced, discrete time intervals. The sequence is called a univariate time series when $d = 1$ and a multivariate time series when $d > 1$.

In general, a single univariate time series of a variable x can be decomposed into three main components: trend, cycles, and seasonality [22]. Figure 2.1 illustrates these components. The trend is a systematic linear or nonlinear component of the variable x that changes over time but does not repeat within the observed time frame. The cyclical component presents the gradual, long-term, potentially irregular rises and falls of the variable x. Besides the trend and cyclical components, the time series data may contain a seasonal component that presents the regular, repetitive variations of the variable x, such as a reoccurring daily or weekly pattern. In addition to these three main components, every time series contains a random error (noise) component, which presents a random increase or decrease of the variable x for a certain period.

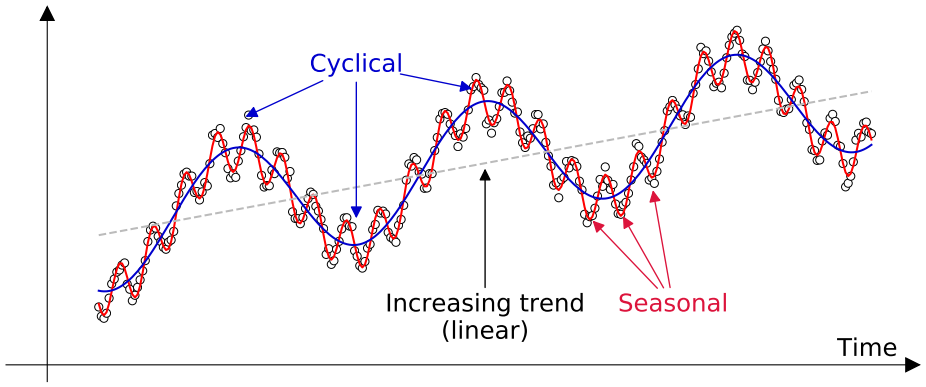


Figure 2.1: Examples of typical time series patterns in a univariate time series. The observations are marked using outlined circles, which display the random noise present in the time series.

A time series is called stationary when its statistical properties do not vary over time, i.e., the mean, variance, and autocorrelation structure of the series are independent of time. The stationarity is an important property since many classical statistical algorithms for time series analysis work only with stationary data. However, time series data is often non-stationary in real-world applications, and therefore many classical algorithms require the time series data to be first transformed into a stationary form. [11]

2.2 Machine Learning

Machine learning (ML) is a subset of artificial intelligence (AI) consisting of algorithms that improve automatically through experience (i.e., data) without being explicitly programmed for the given task [25, p. 2]. There are several different ML modes, yet most algorithms can be divided into supervised and unsupervised learning [25, p. 102].

Supervised learning algorithms learn from a dataset where the data points consist of different features associated with a label or target. These methods are called supervised because they need an external teacher to provide a label y_i for each data point $\mathbf{x}_i$, which the algorithm uses to learn the mapping $f(\mathbf{x}) = \mathbf{y}$. In other words, given a collection of observed data points $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ that is associated with a value or vector of labels $\mathbf{y}$, these algorithms learn to predict $\mathbf{y}$ from $\mathbf{x}$, often by estimating $p(\mathbf{y}|\mathbf{x})$ [25, p. 103].

In contrast, unsupervised learning algorithms learn from a dataset containing data points with many features without any associated label. Thus, these algorithms must independently learn (without supervision) useful properties from the structure of the dataset alone as compared to the supervised learning algorithms. [25, p. 103]

2.3 Deep Learning

Deep learning (DL) is a subclass of machine learning methods based on artificial neural networks (ANNs). The major advantage of DL over classical ML methods lies in its ability to automatically extract features of the data (known as representation learning), requiring no expensive domain expertise to perform feature engineering [14, 25]. As the name suggests, ANNs are vaguely inspired by biological neural networks [25, 66]. ANNs consist of artificial neurons, which are also called nodes, and connections between these nodes called weights that are adjusted during the learning process.

An ANN has one or more hidden layers, each of which consists of several neurons. When the ANN consists of multiple hidden layers, it is called a deep neural network (DNN). Each neuron in each network layer receives the output of each neuron of the previous layer as an input, where each neuron holds a separate weight for each input it receives. These weights are adjusted during the training of the network. The inputs and weights that each neuron receives are combined into a weighted sum, which then passes through an

activation function that determines the neuron's final output. Formally,

$$y = g(\sum_{i=1}^{d} x_i w_i + b) = g(\mathbf{w}^T \mathbf{x} + b)$$

where y is the output of the neuron, $g(.)$ is the activation function, $\mathbf{w}$ is the learnable weight vector, b is the learnable bias term (used to adjust the output of the activation function), $\mathbf{x} \in \mathbb{R}^d$ is the input vector, and d is the number of input dimensions.

To learn complex patterns present in the dataset, neural networks must approximate nonlinear mappings between the inputs and outputs. ANNs can do this using activation functions, which introduce non-linearities in the network. There are many different activation functions for this purpose, such as Rectified Linear Unit (ReLU) $\max(0, x)$, sigmoid $(1 + e^{-x})^{-1}$, and hyperbolic tangent $\tanh(x)$.

Most deep learning algorithms learn using an optimization algorithm called stochastic gradient descent (SGD) [25]. The optimization procedure requires choosing an objective function (also known as a loss function) for measuring the discrepancy (error) between the target output y and the computed output $\hat{y}$, which is to be minimized (or maximized) to optimize the network parameters. The gradients for the weight adjustments of the multilayer network are computed using an algorithm called backpropagation (BP) [25, p. 200]. The BP procedure allows us to compute the gradients of an objective function for each weight and bias in a multilayer ANN by simply using the chain rule for the derivatives. The backpropagation algorithm is repeatedly applied to propagate gradients through all layers of the network, starting from the last layer (the output of the network) all the way to the first layer (where the input data is fed to the network). Formally, the equation for updating a single parameter in the network using gradient descent can be expressed as

$$\theta(t+1) = \theta(t) - \eta \frac{\partial C}{\partial \theta}$$

where θ is a single parameter of the network, η is the learning rate, and C is the cost (or loss) function. The partial derivative in the equation above is calculated using the chain rule.

2.3.1 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a class of ANNs for processing sequential data with temporal dependencies [25]. Thus, they are naturally suited for handling time series data where the data points present temporal

relationships. In an RNN, each hidden cell receives the output from the previous cell and an external input. The output of the cell is then passed to the next cell similarly. Using the previous hidden state allows the network to retain a memory of the past decisions, enabling the network to find correlations between events separated by multiple steps. Although various forms of RNNs exist, they all follow the same idea of using recurrent connections between hidden cells. In the simplest form, an RNN can be expressed using the following set of equations:

$$\mathbf{h}_t = \sigma_h(\mathbf{W}_x\mathbf{x}_t + \mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{b}_h)$$
$$\mathbf{y}_t = \sigma_y(\mathbf{W}_y\mathbf{h}_t + \mathbf{b}_y)$$

where $\mathbf{h}_t$ is the hidden state, σ_h and σ_y are the activation functions, $\mathbf{W}_x$, $\mathbf{W}_h$, $\mathbf{W}_y$ are the weight matrices, $\mathbf{b}_h$ and $\mathbf{b}_y$ are the bias vectors, $\mathbf{x}_t$ is the current input vector, $\mathbf{h}_{t-1}$ is the previous state, and $\mathbf{y}_t$ is the output vector. Figure 2.2 presents a visualization of this structure.

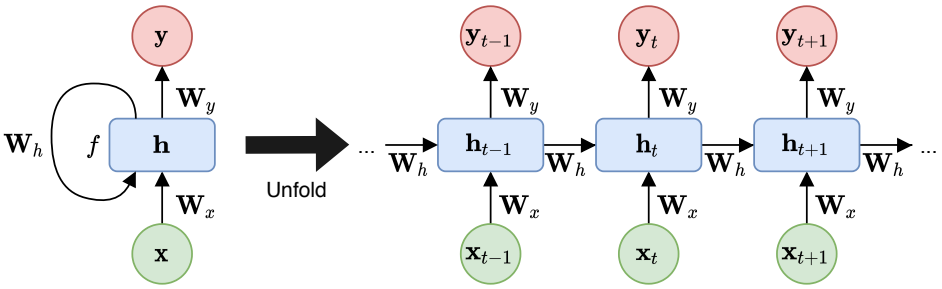


Figure 2.2: An unfolded basic recurrent neural network.

Training of RNNs relies on an extension of the backpropagation algorithm called backpropagation through time (BPTT). Simply put, BPTT unrolls the RNN and propagates the error backward over the whole input sequence, one step at a time. Then, the weights of the network are updated with the accumulated gradients. One issue of BPTT is that the training of an RNN becomes slow with very long input sequences because the forward/backward cost per parameter must be performed over a large number of time steps. To counter this problem, a variation of BPTT called truncated backpropagation through time (TBPTT) was introduced, which approximates the full BPTT, and is most commonly used in practice. The disadvantage of this algorithm is that the network cannot learn as long dependencies as the full BPTT since the gradient can only flow back until it is truncated.

Plain RNNs can only use the past information of the sequence up to the current state. However, in some applications, predicting from the current

time step could also benefit from knowing the sequence's future values. For this purpose, bidirectional recurrent neural networks (BiRNNs) [63] were invented. BiRNNs use two separate RNNs to utilize the whole input sequence information, where one processes the sequence in a forward direction and the other backward. With this structure, there are two hidden states ($\overrightarrow{\mathbf{h}}_t$ and $\overleftarrow{\mathbf{h}}_t$) produced from each time step, which summarize the information of the past and the future of the sequence at the current time step. The outputs of BiRNN are the combination of the forward and hidden states at each time step. The states can be combined in different ways, such as concatenating or summing the terms together. BiRNNs are trained similar to plain RNNs as the two RNNs do not interact.

While, in theory, RNNs are powerful models for sequential data modeling, they are challenging to train in practice. The main reasons for this are the vanishing gradient and exploding gradient problems. [54] Both problems are caused by gradient-based learning and backpropagation, where the partial derivatives are computed by traversing the network from the last layer towards the first layer. By applying the chain rule, the network layers undergo continuous matrix multiplications to compute their gradients towards the final layer. In a neural network with n hidden layers, n gradients are multiplied together. If these gradients are large, the gradients will exponentially increase when propagating the gradients towards the initial layer of the model, causing the exploding gradient problem. The vanishing gradient is the opposite problem where the gradients will decrease exponentially when the gradients consist of small numbers when propagating the gradients through the model.

2.3.2 Long Short-Term Memory

A long short-term memory (LSTM) is a type of RNN that was designed to solve the vanishing gradient issue present in the classical RNNs, making the LSTM be able to learn long-term dependencies of over 1000 time steps [30]. The LSTM overcomes the vanishing gradient problem using an explicit memory (a cell state) and special gating units that regulate the information flow passing through the cell memory, imposing a constant error flow through the cell. There are several variants of LSTM with minor architectural differences, often having no significant performance advantages over each other in practice [26]. We describe the commonly used classic LSTM variant proposed by Gers et al. [23], who added a forget gate into the LSTM architecture. The LSTM unit, presented in Figure 2.3, is composed of a cell, an input gate, an output gate, and a forget gate.

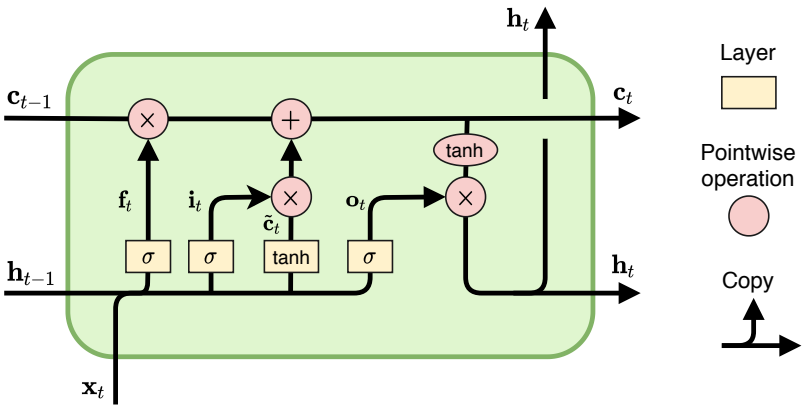


Figure 2.3: Basic architecture of an LSTM cell.

The first gate of the LSTM unit, called the forget gate, decides what information to drop from the cell state. The layer takes previous hidden state $\mathbf{h}_{t-1}$ and current input $\mathbf{x}_t$ and outputs a number between 0 and 1 through a sigmoid activation function σ, where 0 implies completely forgetting and 1 means completely keeping the information.

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f)$$

The next step, consisting of two parts, decides what new information to store in the cell state. First, a sigmoid layer called the input gate determines which values need to be updated. Then, a tanh layer chooses the candidate values $\tilde{\mathbf{c}}_t$ that could be added to the cell state.

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i)$$
$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c)$$

The new cell state is then obtained by multiplying the old state $\mathbf{c}_{t-1}$ by $\mathbf{f}_t$, thereby forgetting the information the forget gate decided to drop earlier. Then, the new candidate values $\tilde{\mathbf{c}}_t$ scaled by the input gate $\mathbf{i}_t$ are added where the input gate decided how much to update each cell state value.

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$$

The final step is to decide what to output based on the cell state. First, a sigmoid layer called the output gate decides which parts of the cell state go to output. Next, the cell state passes through a tanh function (pushing

the values between -1 and 1), multiplied by the output of the sigmoid gate to only output the decided parts, yielding the final output of the LSTM cell.

$$\mathbf{o}_t = \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{b}_o)$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

2.3.3 Gated Recurrent Unit

A gated recurrent unit (GRU) [16] is another variant of RNNs, which can be considered a lightweight version of LSTM. Several studies have demonstrated that GRU generally performs as well as LSTM [10, 17]. Besides, GRU is faster to train and better suited for smaller datasets than LSTM due to its simpler architecture and fewer parameters [10, 17, 67]. Whereas LSTM uses three different gates, GRU has reduced these into only two: the reset gate and the update gate. Furthermore, GRU has no separate cell state, and it uses only the hidden state $\mathbf{h}$ to transfer information.

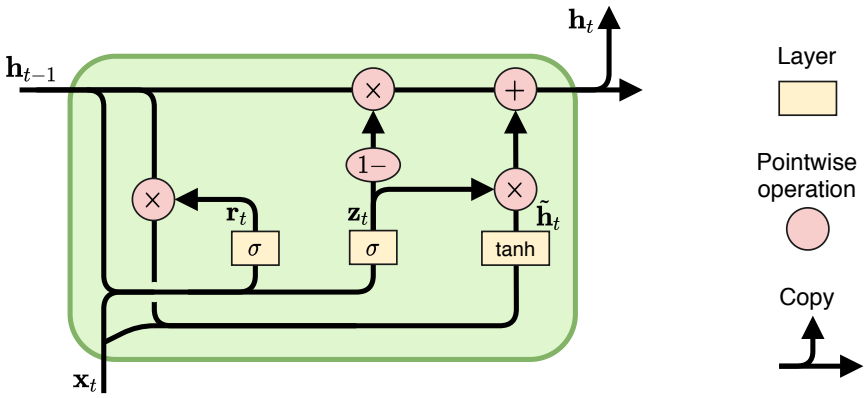


Figure 2.4: Basic architecture of a GRU cell.

The update gate of GRU functions similarly to the forget and input gate of an LSTM, deciding what information to discard and what information to add. The reset gate acts similarly to the forget gate of LSTM by deciding how much old information to forget. The simplified structure, and the similarity between LSTM and GRU, are also visible in the equations of GRU:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{b}_z)$$
$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r)$$
$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h\mathbf{x}_t + \mathbf{U}_h(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h)$$
$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$

where $\mathbf{W}$ and $\mathbf{U}$ are the weights to be learned, $\mathbf{h}_t$ is the hidden state, and $\mathbf{x}_t$ is the input. This structure is visualized in Figure 2.4.

2.3.4 Autoencoders

An autoencoder (AE) is an unsupervised (also referred to as self-supervised) ANN that learns to efficiently compress and encode input data into a reduced latent presentation, and then reconstruct the original input using the compressed presentation. AEs can be considered a generalized version of Principal Component Analysis (PCA). Unlike PCA that can only perform linear dimensionality reduction, AEs can learn complex, nonlinear relationships in the data. For instance, an autoencoder with a single linear layer is nearly equivalent to PCA [12].

As illustrated in Figure 2.5, an AE consists of an encoder, a bottleneck layer, and a decoder. The encoder $g(.)$, parametrized by ϕ, first maps the input data $\mathbf{x}$ into a lower dimensional latent presentation (also referred to as a code) $\mathbf{z}$, where the size of the bottleneck layer determines the latent space dimensionality. Then, the decoder $f(.)$, parameterized by θ maps the lower dimensional signal back to the original form, which in this case is the reconstructed input $\mathbf{x}' = f_\theta(g_\phi(\mathbf{x}))$.

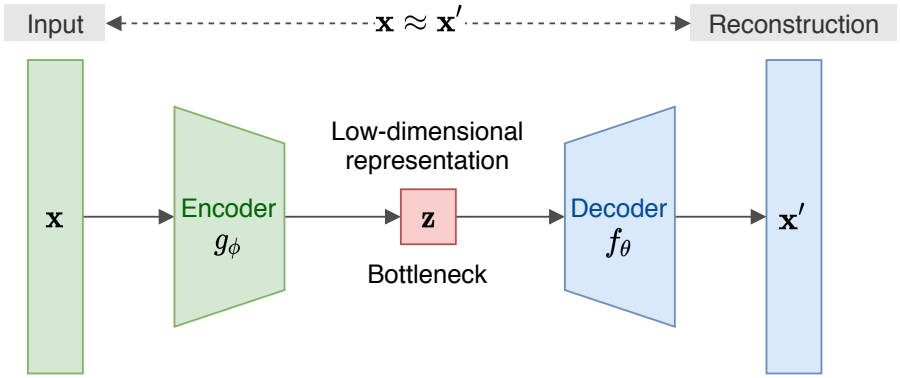


Figure 2.5: Illustration of autoencoder model architecture. Adapted from Weng [73].

The parameters of the model (θ, ϕ) are learned simultaneously during the training, where the model attempts to reconstruct the original input, $\mathbf{x} \approx f_\theta(g_\phi(\mathbf{x}))$. The difference between the input and the reconstruction is called reconstruction error, which the AE attempts to minimize as its objective function during the training. Various metrics can be used to measure the

reconstruction loss depending on the application, such as the mean squared error (MSE):

$$L_{\mathrm{AE}}(\theta, \phi, \mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}_i - f_\theta(g_\phi(\mathbf{x}_i)))^2 \tag{2.1}$$

The bottleneck layer should generally have smaller dimensionality than the input data to prevent the AE from simply learning an identity function, where it merely copies its inputs to the output without learning any meaningful representation of the data. Alternative measures are also possible, such as introducing some form of regularization, usually by incorporating an additional loss term in the objective function.

The exact architecture of the encoder and decoder parts varies by the use case. For instance, both encoder and decoder could contain a simple feedforward network, LSTM layers, or convolutional neural network (CNN) layers.

2.3.5 Variational Autoencoders

Variational autoencoder (VAE) [35, 36] is a deep generative model that combines Bayesian graphical models with deep neural networks. The main advantage of VAEs over AEs is that they can learn more precise latent presentations of the data through probabilistic modeling, which can make them generalize better than AEs [5].

Variational Inference

To understand the underlying idea of VAEs, we now describe the problem scenario from a probabilistic perspective. Let us consider some dataset $\mathbb{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ consisting of N independent and identically distributed (i.i.d.) samples that are generated by a random process involving an unobserved continuous latent variable $\mathbf{z}$. The process has two steps. First, a value $\mathbf{z}_i$ is drawn from some prior distribution $p_{\theta^*}(\mathbf{z})$, followed by drawing a value $\mathbf{x}_i$ from some conditional distribution $p_{\theta^*}(\mathbf{x}|\mathbf{z})$. The true parameters θ^* and the values of the latent variables $\mathbf{z}_i$ are unknown to us. In this framework, the inference problem refers to computing the posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$, i.e., inferring the values of $\mathbf{z}$ given the observed data $\mathbf{x}$. This posterior can be written as:

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}, \mathbf{z})}{p_\theta(\mathbf{x})} = \frac{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})}{\int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z}} \tag{2.2}$$

Unfortunately, the integral in the denominator in Equation 2.2 for computing the marginal likelihood $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{x}) d\mathbf{z}$ is typically intractable by

analytical methods and has no closed-form solution. These intractabilities appear quite commonly due to complicated likelihood functions, $p_\theta(\mathbf{x}|\mathbf{z})$, such as in VAEs, where the likelihood function can be expressed by a neural network with one or more nonlinear hidden layers [35]. Therefore, this true posterior distribution needs to be approximated. In the context of VAEs, this problem is solved using variational inference.

Variational inference aims to approximate the intractable true posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ by a variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$ parameterized by ϕ. The idea is that the variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$ belongs to a family of parametrised family of distributions of simpler form than the true posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$. The goal is to find the best approximation of the target distribution among the chosen family, i.e., finding the most similar distribution to the target distribution. This approximation can be found by measuring the Kullback-Leibler divergence (KL divergence) between the variational approximation and the true posterior distribution, which measures the dissimilarity between the two distributions. Given two continuous distributions, q and p, their KL divergence is given by:

$$D_{KL}(q \mid\mid p) = \int_z q(z) \log \frac{q(z)}{p(z)} dz$$

Since our goal is to find the variational parameters ϕ that minimize the divergence between the variational approximation $q_\phi(\mathbf{z}|\mathbf{x})$ and the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$, finding the optimal approximate posterior can be formalized by the following optimization problem:

$$q^*_\phi(\mathbf{z}|\mathbf{x}) = \underset{\phi}{\operatorname{argmin}}\, D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \mid\mid p_\theta(\mathbf{z}|\mathbf{x}))$$

Architecture Overview

When viewing the VAE from an AE perspective, the unobserved latent variables $\mathbf{z}$ can be considered the code of an AE. Further, $q_\phi(\mathbf{z}|\mathbf{x})$ is commonly referred to as a probablistic encoder since given a data point $\mathbf{x}$, it outputs a distribution over the values of latent variables $\mathbf{z}$ that can characterize the data point. Similarly, $p_\theta(\mathbf{x}|\mathbf{z})$ is called a probabilistic decoder since it outputs a disribution over the possible values of $\mathbf{x}$ given a latent variable $\mathbf{z}$. Figure 2.6 illustrates the general structure of a VAE modelled with multivariate Gaussian distributions.

Similar to AEs, the encoder and decoder networks of the model can be any type of neural network depending on the application. For sequential data, which is the focus of this thesis, the networks commonly use RNNs or CNNs.

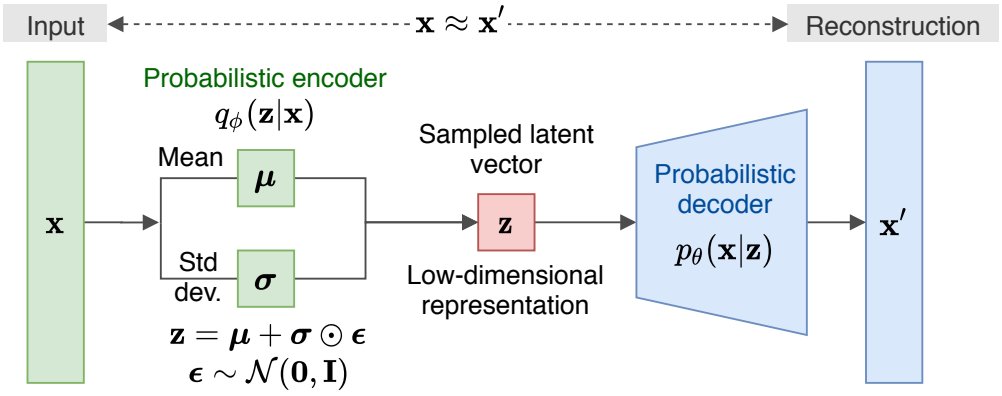


Figure 2.6: A variational autoencoder with multivariate Gaussian prior and posterior. Adapted from Weng [73].

Training Objective

As the optimization objective in a VAE, we want to maximize the log marginal likelihood $\log p_\theta(\mathbf{x})$ of the data under the model. For any choice of variational approximation $q_\phi(\mathbf{z}|\mathbf{x})$, the following holds:

$$\begin{aligned}
\log p_\theta(\mathbf{x}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x})] && (2.3)\\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right] && (2.4)\\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right] && (2.5)\\
&= \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]\right]}_{\mathcal{L}_{\theta,\phi}(\mathbf{x})} + \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right]}_{D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))} && (2.6)
\end{aligned}$$

The second term in Equation 2.6 is the KL divergence of the approximate posterior from the true posterior, which is non-negative, and zero if and only if $q_\phi(\mathbf{z}|\mathbf{x})$ equals the true posterior distribution. The first term is called the variational lower bound, also known as the evidence lower bound (ELBO), on the marginal likelihood of the data [36]. Since the KL divergence is always non-negative, the ELBO is a lower bound of the log-likelihood of the data:

$$\begin{aligned}
\mathcal{L}_{\theta,\phi}(\mathbf{x}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right]\\
&= \log p_\theta(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x}))\\
&\leq \log p_\theta(\mathbf{x})
\end{aligned}$$

By maximizing the ELBO $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ with regards to the parameters θ and ϕ, we can simultaneously maximize the marginal data likelihood $\log p_\theta(\mathbf{x})$ and minimize the KL divergence, making the approximation $q_\phi(\mathbf{z}|\mathbf{x})$ from the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$ become better. As the final loss function of the VAE, we negate the ELBO since our goal is to minimize the loss during model training. Now, given a sample $\mathbf{x}_i$, the objective to be minimized has the form

$$L_{\text{VAE}}(\theta, \phi, \mathbf{x}_i) = -\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}_i|\mathbf{z})\right] + D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}_i) \mid\mid p_\theta(\mathbf{z})).$$

The expectation in the loss term can be calculated with Monte Carlo (MC) sampling, where a single sample is usually sufficient when using large enough mini-batches in training [35]. The KL divergence between $q_\phi(\mathbf{z}|\mathbf{x}_i)$ and $p_\theta(\mathbf{z})$ can be computed analytically when the distributions have the same parameter family and a closed-form presentation exists. When no analytical form exists, the KL divergence term can be computed using MC sampling similarly to the likelihood term. With the ELBO as the objective function of the VAE, we can jointly optimize all parameters (ϕ and θ) of the model using SGD.

Reparameterization Trick

To train the network using backpropagation with SGD, the model needs to be differentiable with regard to its learnable parameters, which requires deterministic model parameters. However, the computational graph of a VAE contains a stochastic node due to the sampling of the latent variable $\mathbf{z}$ from the approximate posterior. Fortunately, this problem is easily solvable using a reparametrization trick proposed by Kingma et al. [35], which allows differentiating the objective function with regard to both ϕ and θ through a change of variables.
The trick, illustrated in Figure 2.7, consists of expressing the random variable $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$ as a differentiable (and invertible) transformation $\mathbf{z} = g(\phi, \mathbf{x}, \boldsymbol{\epsilon})$ of an auxiliary random variable $\boldsymbol{\epsilon}$ from a fixed distribution which is independent of $\mathbf{x}$ or ϕ. For instance, let the variational approximate posterior be a multivariate Gaussian with a diagonal covariance structure:

$$\log q_\phi(\mathbf{z}|\mathbf{x}_i) = \log \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2 \mathbf{I})$$

where the mean and standard deviation, $\boldsymbol{\mu}_i$ and $\boldsymbol{\sigma}_i^2$, are the outputs of the probabilistic encoder of the VAE. Now, we can sample from the posterior $\mathbf{z}_i \sim q_\phi(\mathbf{z}|\mathbf{x}_i)$ simply using $\mathbf{z}_i = g(\phi, \mathbf{x}_i, \boldsymbol{\epsilon}) = \boldsymbol{\mu}_i + \boldsymbol{\sigma}_i \odot \boldsymbol{\epsilon}_i$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

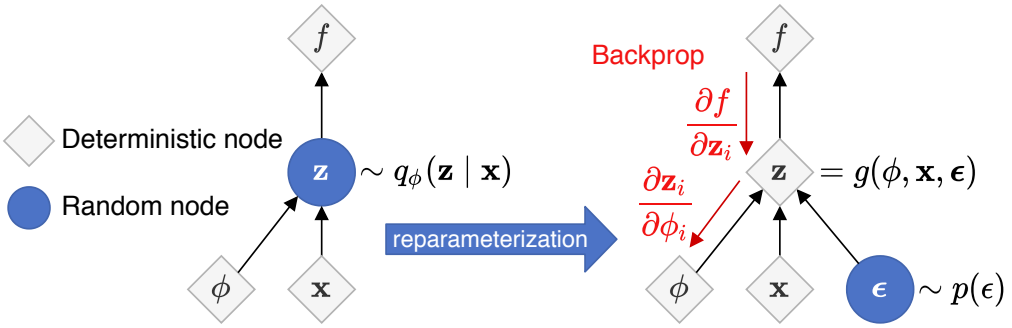


Figure 2.7: Illustration of the reparameterization trick. Adapted from Kingma and Welling [36].

2.3.6 Normalizing Flows

Gaussian distributions are often used as the distribution for the latent variables in VAEs [38], which can be too simple since the true probability distribution of the data could be far from Gaussian. Normalizing flows [59] are used to transform simple probability densities, such as Gaussians, into more expressive distributions, making them useful in applications such as generative models, reinforcement learning, and variational inference [38].

A normalizing flow transforms a simple probability distribution, such as Gaussian, into a more complex distribution by applying a series of invertible and differentiable mappings [38]. Let $\mathbf{z} \in \mathbb{R}^d$ be a random variable and $f : \mathbb{R}^d \mapsto \mathbb{R}^d$ an invertible smooth mapping. When using this mapping to transform a random variable $\mathbf{z} \sim q(\mathbf{z})$, the resulting random variable $\mathbf{z}' = f(\mathbf{z})$ has the following probability distribution:

$$q(\mathbf{z}') = q(\mathbf{z}) \left| \det \frac{\partial f^{-1}}{\partial \mathbf{z}'} \right| = q(\mathbf{z}) \left| \det \frac{\partial f}{\partial \mathbf{z}} \right|^{-1}$$

By applying a series of simple mappings following the equation above, we can construct arbitrarily complex densities. The resulting density $q_K(\mathbf{z})$ is obtained by sequentially transforming a random variable $\mathbf{z}_0$ with distribution q_0 through a chain of K transformations f_k:

$$\mathbf{z}_K = f_K \circ \cdots \circ f_1(\mathbf{z}_0), \quad \mathbf{z}_0 \sim q_0(\mathbf{z}_0)$$

$$\log q_K(\mathbf{z}_K) = \log q_0(\mathbf{z}_0) \sum_{k=1}^{K} \log \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right|,$$

where the first equation is a shorthand for the formula $f_K(f_{K-1}(\ldots f_1(\mathbf{z}_0)))$.

A wide range of normalizing flows has been proposed [38]. One of the simplest flows, named planar flow [59], is defined as follows,

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^T\mathbf{z} + b),$$

where $\mathbf{u}, \mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$ are the trainable parameters, and $h : \mathbb{R} \to \mathbb{R}$ is an element-wise non-linearity. The Jacobian determinant for the transformation is

$$\left|\det \frac{\partial f}{\partial \mathbf{z}}\right| = \left|1 + \mathbf{u}^T\psi(\mathbf{z})\right|.$$

2.4 Linear Gaussian State-Space Model

Linear state-space models (SSMs) [37, 50] are commonly used in time-series analysis and modeling of dynamic systems. These models, described by a linear dynamical model, assume that the observations are generated linearly from hidden states with some additional random variation. Formally, the models assume that the d-dimensional observations $(\mathbf{y}_1, \ldots, \mathbf{y}_N)$ are generated from m-dimensional latent states $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ following a first-order Gaussian Markov process:

$$\begin{aligned}
\mathbf{x}_t &= \mathbf{T}\mathbf{x}_{t-1} + \mathbf{w}_t \\
\mathbf{y}_t &= \mathbf{O}\mathbf{x}_t + \boldsymbol{\epsilon}_t \\
\mathbf{x}_0 &\sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)
\end{aligned}$$

where $\mathbf{T} \in \mathbb{R}^{m\times m}$ and $\mathbf{O} \in \mathbb{R}^{d\times d}$ are the transition and observation matrices, $\mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{W})$ and $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{E})$ are the transition and observation noises, $\mathbf{x}_0$ is assumed to follow normal distribution with some mean $\boldsymbol{\mu}_0 \in \mathbb{R}^m$ and covariance matrix $\boldsymbol{\Sigma}_0 \in \mathbb{R}^{m\times m}$. The dimensionality of the latent space, m, is usually assumed to be much smaller than the dimensionality of the observations, d, to efficiently model the dependencies between the high dimensional observations. [44]

The inference of linear Gaussian SSMs can be performed using the Kalman filter algorithm, which provides an exact Bayesian solution for the model [50, p. 632]. In particular, when the initial state $\mathbf{x}_0$ is Gaussian, then all subsequent states of the model will also be Gaussian. This property allows the exact computation of the posterior of the latent states given the observations. Additionally, the algorithm allows computing the marginal likelihood of the observed sequence and performing posterior predictive density computation for the observations, which is useful, for instance, for time series forecasting.

For further details of state-space models and Kalman filtering, we refer the reader to chapter 18 of Murphy's book [50].

2.5 Extreme Value Theory

Extreme Value Theory (EVT) [6] is a branch of statistics studying extreme values in probability distributions. The key result from EVT suggests that the extreme values (the tail) of any distribution follow a Generalized Pareto distribution (GPD). This result is particularly useful in anomaly detection because it allows developing parametric distribution-based anomaly detection methods using prediction errors of a model, without making critical assumptions of the unknown distribution of the data [19, 65].

The principle of the EVT can be explained as follows. Let X be a random variable and $F(x) = P(X \leq x)$ be its cumulative distribution function. Let us denote the tail of the distribution by $\tilde{F}(x) = P(X > x)$. According to EVT, under a weak condition, the extreme events follow the same distribution, regardless of the underlying data distribution [65]. This distribution is known as the extreme value distribution (EVD):

$$G_\gamma : x \mapsto \exp(-(1+\gamma x)^{-\frac{1}{\gamma}}), \quad \gamma \in \mathbb{R}, \quad 1+\gamma x > 0$$

where γ denotes the extreme value index of the distribution, which describes the tail heaviness of the underlying probability distribution. After fitting an EVD to the tail of unknown distribution, we can analyze the probability of the extreme events in the data [65].

The Peaks-Over-Threshold (POT) is a technique within EVT that allows us to model the distribution tails of a set of data generated from an unknown distribution, enabling estimating the probability of rare events beyond the largest observed values [58]. The technique relies on the second theorem in EVT (Pickands–Balkema–de Haan theorem), which is defined as

$$\bar{F}_t(x) = P(X - t > x \mid X > t) \sim (1 + \frac{\gamma x}{\sigma(t)})^{-\frac{1}{\gamma}}.$$

The theorem suggests that the excess over a learning threshold t, denoted as $X - t$, is likely to follow a GPD with parameters γ, σ. The POT method applies this theorem by first selecting those instances of the initial observations that exceed a certain learning threshold t. A GPD can then be used to approximate the probability distribution of the selected observations under extreme value conditions. The GPD parameters (γ, σ) can be estimated by applying parametric statistical methods to those observations. The most commonly used method for this purpose is the maximum likelihood estimation (MLE). [58]

2.6 Anomaly Detection

Anomaly detection refers to the problem of identifying the patterns in data that do not follow the expected behavior [13]. Finding these anomalous events or patterns in the data has been an actively studied research area for a long time in the data mining field, and it remains one of the most critical tasks in data analysis [10]. With the recent technological advances, the amount of data produced by modern industrial devices is usually high-dimensional and high-volume, exceeding the human capabilities to monitor it manually. Hence, the demand for automated data analysis tools is high.

Anomaly detection has a wide range of application domains. These include fraud detection, intrusion detection, fault diagnosis in industry, healthcare, network monitoring, smart devices, smart cities, and the internet of things (IoT) devices.

2.6.1 Defining Anomalies

The term anomaly has no single agreed definition due to the diverse domains of anomaly detection. Anomalies are also alternatively referred to as outliers, abnormalities, discordants, deviants, exceptions, peculiarities, or contaminants in different application domains [10, 13]. Although anomalies have no single, universally agreed definition, some commonly accepted characterizations exist. For instance, Chandola et al. [13] use the following definition: "Anomalies are patterns in data that do not conform to a well defined notion of normal behavior.". Another widely recognized definition was given by Hawkins [29]: "An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism". Following the main aspects from the previous definitions and considering the context of this thesis, we define anomalies (and outliers) as follows:

Definition 2.6.1 (Anomaly)**.** An anomaly is an unexpected deviation in an observation or sequence of observations from the expected behavior in a univariate time series or in the correlation between multiple variables of a multivariate time series.

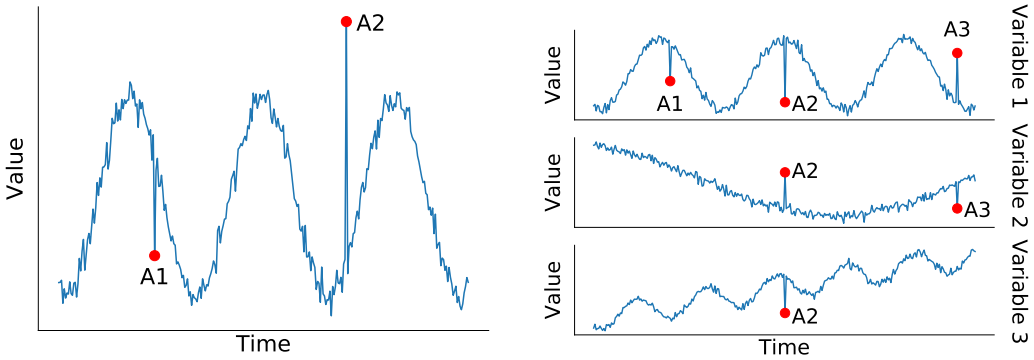
A closely related term to anomalies is noise, which is a different concept than an anomaly. Noise represents the unavoidable random variation present in the data, which has two primary sources: errors caused by the measurement tools and errors arisen during processing or gathering the data. Since noise is meaningless information in the data, it is considered a data attribute that is not of interest to data analysts. [10, 13, 80]

2.6.2 Types of Anomalies

To be able to identify anomalies, it is essential to acknowledge the various types of anomalies that may be present in the data. Chandola et al. [13] categorize anomalies into the following groups:

- **Point anomalies** When a data point differs significantly from the rest of the data, it is considered a point anomaly. Point anomaly is the simplest type of anomaly in the anomaly detection field and the easiest to detect since no contextual information is needed to identify the anomaly. For example, in a time series data of temperatures measured in Finland over a year, a temperature of 40 °C can be considered anomalous at any time of the year. Figure 2.8a provides an example of a (global) point anomaly (A2) in a univariate time series.

- **Contextual anomalies** When a data point is considered anomalous only in a specific context, it forms a contextual anomaly. For instance, in a time series data of temperatures measured in Finland over a year, observing a temperature of 20 °C would be normal in summer. However, such a temperature measured in winter would be considered an anomaly. Figure 2.8 presents examples of univariate (A1 in Figure 2.8a and A1 in Figure 2.8b) and multivariate (A2 and A3 in Figure 2.8b) contextual anomalies in time series data.



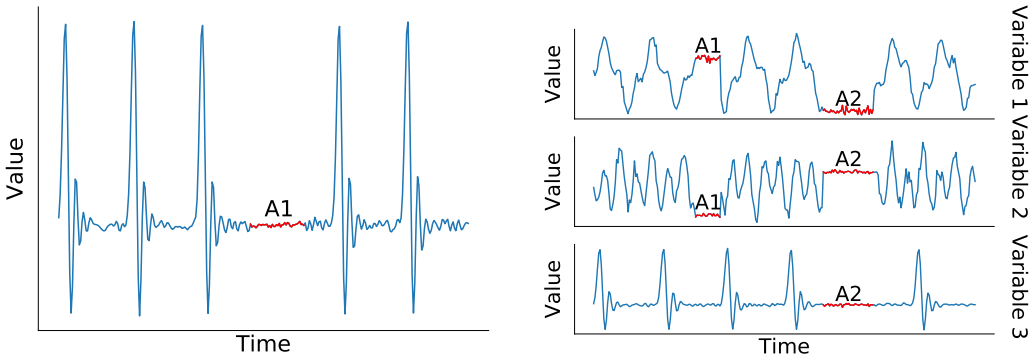
(a) Univariate time series.

(b) Multivariate time series.

Figure 2.8: Point and contextual anomalies in time series data.

- **Collective anomalies** When a group of related data is considered anomalous, while the individual data points may not be considered anomalous on their own, the group constitutes a collective anomaly. Collective anomalies can be local or global, and they may affect a single

variable or multiple variables in multivariate data [9]. For example, in time series data, this type of anomaly could be an event that persists for an unusually long time. These anomalies can only be detected in data where the data points are related to each other, such as sequential, spatial, or graph data. Figure 2.9 provides examples of univariate (A1 in Figure 2.8a) and multivariate (A1 and A2 in Figure 2.8b) collective anomalies in time series data.



(a) Univariate time series.

(b) Multivariate time series.

Figure 2.9: Collective anomalies in time series data.

Since this thesis focuses on anomaly detection in multivariate time series data, contextual and collective anomaly types are particularly important to consider because the data points could have strong temporal correlations. Additionally, the multivariate aspect of the time series makes the anomaly detection problem more challenging than in a univariate context since the anomalies occurring in such data can be more complicated. In the simplest form, similar to the univariate case, anomalies in a multivariate time series may refer to unusually high or low values or unexpected subsequences occurring in one or more of the time-dependent variables [15]. However, multivariate anomalies may also correspond to an unexpected change in the (possibly non-linear) relationship between two or more variables [15].

Understanding the possible types of anomalies in the data is essential when choosing an appropriate anomaly detection method. For instance, if the algorithm can only process univariate time series data, it cannot identify multivariate anomalies that exhibit correlations between multiple univariate time series [9]. Finally, if the anomaly detection method neglects the temporal relations of the time series data, it can only detect non-contextual, (global) point anomalies [9].

2.6.3 Types of Anomaly Detection

Many anomaly detection methods require training using a dataset with labeled data samples indicating whether the data point is anomalous or normal. Based on the extent of the use of labeled data, anomaly detection approaches can be separated into three different types: supervised, semi-supervised, and unsupervised anomaly detection [13].

Supervised anomaly detection methods use labeled training data, which contains examples of both normal and abnormal samples. This approach comes with several disadvantages. First, obtaining anomalous examples is often challenging since anomalous instances are far fewer than normal instances in many real applications, which causes additional challenges due to the imbalanced class distribution [27]. Second, labeling the data is usually performed manually by human experts with domain knowledge, which is expensive and time-consuming [27]. Besides, supervised methods can only detect known types of anomalies that have been present in the training set that the models learned from, thus requiring a significant effort to keep the models up-to-date as types of anomalies are discovered [13]. Due to these challenges, supervised anomaly detection methods have limited use in many practical applications, hence semi-supervised and unsupervised approaches are preferred.

Semi-supervised anomaly detection techniques assume that the training data contains only normal instances of the data. Since these models require no examples of anomalous instances, they are suitable for a broader range of applications than supervised anomaly detection approaches. Furthermore, in practice, most semi-supervised anomaly detection techniques can be used in unsupervised mode by assuming that the training data contains only a few anomalous samples and that the model is robust enough to the presence of these anomalies during training [13]. The usual approach in semi-supervised techniques is to build a model using the data representing the normal behavior of the data, and then using the trained model to discover anomalies in unseen test data. A downside of the semi-supervised methods is that they require enough samples of normal data, where the training set should sufficiently present the underlying distribution of the data. Otherwise, these methods may yield a high false alarm rate when they classify previously unseen (yet normal) data instances as anomalous.

Unsupervised anomaly detection methods do not require separate training data but can be directly applied to unseen data. Therefore, these techniques are the most widely applicable to various anomaly detection domains. These methods work under the assumption that there are far more normal than anomalous instances present in the data [13]. When this assumption does not

hold, these approaches are likely to produce a high rate of false positives [13].

2.6.4 Output of an Anomaly Detection Algorithm

One of the core parts of any anomaly detection algorithm is the way they report the identified anomalies. In general, there are two possible output types [13, 24]. The first type of output is a direct binary label that indicates whether a data instance is anomalous or normal. The second possibility is an anomaly score that can be used to weigh the severity of an anomaly. The score can be a distance, a confidence level, or some similar metric indicating the degree of abnormality of the data point. After computing the anomaly score for a data point, it can be used to classify the point as normal or abnormal by introducing a threshold for the anomaly score.

Determining a suitable threshold for labeling the inputs as anomalous or normal is not always straightforward when using an algorithm that outputs an anomaly score. One way to determine the optimal threshold is by employing validation data with labeled anomalous points and segments and using it to determine the threshold giving the best detection performance. However, often there are not enough labeled anomalous data available. Hence, some other means are usually needed to determine a suitable threshold automatically [32].

In general, an anomaly score is preferred over a direct label since the anomaly score enables to rank different anomalies and prioritizes the most severe anomalies. Furthermore, the number of reported anomalies can be easily reduced by adjusting the threshold, allowing the operator to control the balance of false positives and false negatives depending on the application domain. A direct label is often used in supervised anomaly detection algorithms, whereas unsupervised and semi-supervised algorithms usually output a score.

2.6.5 Concept Drift

In many real-world situations, the environment generating the data is dynamic and non-stationary. Hence, the behavior considered normal in such an environment is expected to change over time. This phenomenon is commonly known as concept drift [4, 21], and it presents an additional challenge in the anomaly detection field. For instance, in an industrial environment, configuration changes, maintenance work, machine updates, and raw material quality fluctuations are common occurrences. These events might alter the system's normal behavior, requiring the applied anomaly detection method to adapt to such changes. Another challenge the concept drifts cause for

the anomaly detection methods is distinguishing the true concept drifts from anomalies and noise [21], which can be difficult in noisy environments and in situations where the presence of concept drifts, or their characteristics, is unknown.

Some of the possible ways to adapt to concept drifts include training the anomaly detection models either continually or retraining using recent batches of data [21]. Choosing a proper way to handle concept drifts depends on the problem domain. In environments with rapid changes occurring continuously, only methods that can learn continually are applicable. On the other hand, when the concept drifts are less common or occur gradually, methods relying on batch retraining may be applicable. Hence, it is essential first to understand the dynamics of the data and the environment, and only then choose a suitable approach to the problem.

2.6.6 Anomaly Detection in Streaming Data

Streaming applications form additional challenges and constraints for anomaly detection. These applications involve monitoring a continuous sequence of data in real-time, which often arrives at high volume and high frequency, requiring to process the data stream efficiently [4]. Since the stream is potentially infinite, any algorithm that attempts to store the entire stream in memory will run out of space [68]. Another challenge in streaming applications occurs from the uncertainties in the input sources, such as delays or measurement errors in the sensors, making the isolation of true anomalies from random noise more difficult [69].

The problem of concept drift (see Section 2.6.5) is often present in streaming applications, which requires finding suitable ways for adapting the models to the potential changes of the data distribution. Depending on the dynamics of the data stream, the predictive models can be trained in several different learning modes: using offline learning, incremental learning, or online learning [21].

Offline learning is the traditional way of training models in the data mining field. In this setting, the model is trained offline using a set of previously collected historical data. Afterward, the trained model can be used for online prediction of new unseen, continuous stream of data. This type of learning is only suitable for static environments where the underlying data distribution does not change over time. Additionally, the method is also applicable in environments where the drifts in the data distribution occur slowly, allowing for retraining the model occasionally from scratch using a newer set of historical data.

Incremental learning mode updates the current model using the most

recent data. In contrast to offline learning mode, where the model has to be trained from scratch every time, this mode updates the model incrementally using each new sample or a small batch of recent data. Hence, in incremental algorithms, the learning is not required to operate in an online (real-time) fashion but incrementally, for example, after observing a specific number of new samples. Incremental algorithms may also have access to the previous examples or summary of the examples seen so far, allowing several passes over the selected set of previously obtained samples [21]. Since this learning mode keeps updating the model with new data, algorithms using this approach can adapt to gradual concept drifts occurring in the data distribution.

Online learning mode is the more restricted version of incremental learning, where the current model is updated continuously with each new sample at a time. Thus, these algorithms must be able to update their state faster than the rate of samples arriving. The algorithms relying on this learning mode can adapt faster to concept drifts in the data distribution compared to incremental learning due to the continual learning from each new sample. Table 2.1 summarizes the differences between online learning and offline learning approaches and between online and offline anomaly detection setups.

	Offline detection	Online detection
Offline learning	The data can be stored and analyzed entirely offline. No real-time analysis is required.	Anomalies must be detected in real-time, but data distribution is relatively static, allowing offline learning.
Online learning	Data distribution of the collected data varies over the observations.	Real-time prediction is required and data distribution may have sudden or gradual concept drifts.

Table 2.1: Comparison of offline and online learning modes for anomaly detection.

The unique properties of streaming applications make anomaly detection over data streams a challenging task, where no perfect solution exists. Complementing the ideas presented by Ahmad et al. [4], this study represents the following constraints for an ideal real-time anomaly detection algorithm in streaming data:

1. Predictions must be performed in an online fashion, i.e., the model must

determine the observation $\mathbf{x}_t$ as normal or anomalous before receiving the subsequent sample $\mathbf{x}_{t+1}$.

2. The model must learn the underlying data distribution in online, incremental, or offline mode depending on the dynamics of the data stream, such as the frequency of concept drifts.

3. The model must perform the anomaly detection in an unsupervised fashion, i.e., without data labels or manually tweaking the parameters.

4. The model should adapt to the possible concept drifts in the underlying statistics of the data stream.

5. The model should detect anomalies as early as possible.

6. The model should minimize both false positives and false negatives.

It is important to note that while the above constraints describe the ideal properties that an anomaly detection algorithm for streaming data should possess, often in practice, some of these constraints may be relaxed, depending on the actual application and problem domain.

Chapter 3

Related Work

This chapter reviews prior work related to the topic of this thesis. The first section of this chapter provides a short survey of relevant anomaly detection algorithms applicable to anomaly detection in multivariate time series data. The second part reviews studies that have proposed methods to detect anomalies in injection molding machine data.

3.1 Anomaly Detection in Multivariate Time Series Data

A wide range of unsupervised anomaly detection methods have been proposed for time series data [4, 9, 10, 13]. Following recent surveys on anomaly detection techniques in time series [10, 20, 31], this study separates these methods into three different categories:

- **Statistical methods** assume that a specific statistical model generates the data.
- **Classical machine learning methods** learn information directly from the data without assuming any specific generative model for the data.
- **Deep learning methods** are a subset of machine learning methods that use artificial neural networks to learn from the data.

In the next three sections, we review some relevant approaches within these categories that could apply to the multivariate time series anomaly detection problem considered in this thesis.

3.1.1 Statistical Methods

Statistical models were the earliest algorithms used for anomaly detection [31, 47]. These models assume that the normal instances of data exist in high probability regions of a stochastic model, whereas anomalies exist in the low probability regions [13]. Typically, these methods fit a statistical model (usually for normal behavior) on previously collected data, which can then perform statistical inference on new unseen data, such as data arriving in streaming (online) fashion. When an observed sample has a low probability of being generated by the fitted model, it is declared an anomaly [13].

Simple univariate statistical techniques are often employed in streaming anomaly detection settings due to their low computational overhead. Examples of such algorithms include moving window statistics (e.g., mean, standard deviation, kurtosis), exponential smoothing such as Holt-Winters, and Grubb's test [4]. While these methods are beneficial in streaming applications with extremely high volume and velocity of data, most of them are limited to finding spatial anomalies in univariate time series data [4]. To apply such algorithms to multivariate data, one needs to monitor each variable of the multivariate separately, leading to poor performance in a multivariate setting.

A widely used statistical approach in manufacturing industries called statistical process control (SPC) leverages statistical methods to control and monitor processes in an online setting [47, p. 121]. Generally, SPC consists of two phases. First, a model is built offline using previously collected data, which is also used to find the control limits of normal behavior. Then, in the second phase, the model can be used for monitoring the process in real-time. For multivariate data, commonly used SPC algorithms include multivariate exponentially weighted moving average (MEWMA), multivariate cumulative sum (MCUSUM), and Hotelling's T^2 statistic. As these methods assume data to be i.i.d., they should generally not be directly applied to autocorrelated data, such as time-series data. Hence, some additional algorithm needs to be usually applied to make the data i.i.d. before applying SPC methods [43]. Another downside of the usual multivariate SPC methods is that they do not allow easy interpretation of the detected anomalies [43].

Prior studies [46] have also successfully used statistical linear time series analysis methods, such as autoregression (AR), autoregressive integrated moving average (ARIMA), vector autoregression (VAR), and vector autoregressive moving average (VARMA) in anomaly detection on time series data. VAR and VARMA are multivariate extensions to the univariate models, which can model linear relationships between multiple time series. These methods can be used for anomaly detection by forecasting the signal and

measuring the error between the observed and predicted value. The models are usually fitted offline with training data, along with parameter tuning based on the data. One disadvantage of these models is that they are generally not very robust to noise, as they usually assume Gaussian distribution on the error term. Additionally, these models require the time series to be stationary and therefore need methods such as differencing to transform the time series into stationary form.

Statistical techniques have several disadvantages. Since these algorithms assume a specific statistical model, they work well only when parametric assumptions closely describe the real distribution. Making such assumptions is particularly challenging in environments where the statistical properties of the environments change over time. Additionally, many of these techniques can be applied only for univariate time series data. While different techniques have developed to overcome this limitation, the algorithms still suffer from the increased dimensionality of the data [13, 31]. Moreover, while the dimensionality of the data may be reduced with methods, such as PCA, they usually come with a cost of lower interpretability of the detected anomalies.

3.1.2 Classical Machine Learning Methods

Anomaly detection methods based on classical ML algorithms do not assume any specific generative model for the data but act more like a "black box" that learns directly from the data. This thesis divides the classical ML-based anomaly detection methods into the following seven categories:

1. **Density-based methods** assign an anomaly score for each point based on the density of the local neighborhood, given some predefined constraints. One of the popular approaches in this category is the Local Outlier Factor (LOF) algorithm, which has also been adapted to detect outliers in data streams [57, 61].

2. **Distance-based methods** detect anomalies based on the distance of each point to their nearest neighbors. A typical algorithm in this category is the k-nearest-neighbors (kNN). In a streaming setting, the algorithm finds the k-nearest-neighbors in a sliding window for each incoming data point. The anomaly score is then calculated as the average distance to all the k neighbors. Since the data points are considered i.i.d., additional feature engineering is required to detect temporal anomalies. Since distance-based methods rely on distances between points, the algorithms are susceptible to the curse of dimensionality [39].

3. **Support vector machine-based methods** learn the boundaries of the points presenting the normal data and classifies any points outside this region as anomalies. The core algorithm of this class is the one-class support vector machine (OC-SVM) [45], which has also been adapted to work in an online manner where the model is sequentially updated while also taking temporal correlations of the data into account [79].

4. **Clustering-based methods** cluster the data points and then classifies those points as anomalous that are far away from other clusters or form abnormal clusters. For time series data, recent work by Wang et al. [72] proposed a clustering algorithm for streaming data that cuts the time series into non-overlapping windows and then clusters the subsequences within the window to detect anomalies. Their approach works with multivariate data and allows anomaly interpretation and detection of anomalous subsequences.

5. **Tree-based methods** build regression or classification trees and base the anomaly score on how many times the data points have to be subdivided in the tree before the target data point is isolated from the others. An example of a tree-based algorithm is Isolation Forest (IF) [42], which assumes that anomalies are the minority and have different values from observations considered normal. Under this assumption, anomalies are more likely to be isolated from other observations when a database of points is randomly partitioned. Streaming half-space-trees (HS-trees) [68] is an online variant of IF, which is incrementally trained. The algorithm works best when anomalies are spread out in the data stream but underperforms in situations where anomalies appear together in a time window.

6. **Projection-based methods**, such as PCA, can be used to project the data into a lower-dimensional space, allowing the use of algorithms that work well with low dimensional data like many univariate methods. The downside of projection-based approaches is that anomalies that are detectable in the original space might appear similar to the normal data in the projected space. These methods can also be used directly for anomaly detection by measuring the projected data points' reconstruction error when the projected points are transformed back into the original space.

7. **Ensemble-based methods** combine the results of multiple algorithms to achieve better results than any of the algorithms alone. An example

of such an algorithm is the Lightweight on-line detector of anomalies (LODA) [56], which uses an ensemble of simple one-dimensional histograms for anomaly detection. The algorithm supports real-time incremental learning, anomaly interpretation and has been shown to work well even with high-dimensional data [56].

The main disadvantage of classical ML-methods is that they often require additional feature engineering, which might require domain knowledge. For instance, most of these methods assume the data to be i.i.d., neglecting the temporal context of the observations. Hence, creating additional features that capture the temporal context of the data is generally needed for time series data. A commonly used alternative approach is to apply these methods to small windows of time series data at a time, where the temporal aspect might be less relevant [28].

3.1.3 Deep Learning Methods

Due to the high dimensionality, complex temporal dependencies, non-linear relations, and stochasticity often present in real-world multivariate time series data, numerous studies have proposed DL-based anomaly detection approaches, which have demonstrated promising performance on such challenging data. Most DL-based methods require an initial phase of offline training with previously collected data (usually consisting of normal instances). After the training, most of these methods can be applied for online anomaly detection for new unseen data.

DL-based anomaly detection models often operate in a semi-supervised setting, where they are trained with normal data instances, after which they fail to reconstruct anomalous samples well. Hence, a core assumption when using these models is that the training data sufficiently covers the underlying true data distribution of normal samples. When this assumption is not met, the models can still learn to model the training data well. However, they will not generalize well to new unseen data and may flag unseen normal data as anomalies.

Prior studies have proposed various DL-based architectures as part of their time series anomaly detection approach. Next, we review the most common DL-techniques used in the literature and present some selected works employing each technique:

1. **Recurrent neural networks (RNNs)**, and their advanced variants (LSTMs and GRUs), are commonly used building blocks in DL-based anomaly detection methods for time series data due to their natural

ability to process sequential information. Usually, purely RNN-based anomaly detection models use a window of previously observed data points to predict the values at the next time stamp and then utilize the error between the predicted and observed values as the anomaly score. For instance, Hundman et al. [32] used a prediction error-based LSTM model on spacecraft telemetry data to detect point and contextual anomalies from each separate channel of a multivariate time series data. However, since their approach uses a separate model for each signal, the method cannot address correlations between different variables of the multivariate time series, making the model unable to detect multivariate anomalies.

2. **Convolutional neural networks (CNNs)** have also been used to detect anomalies from multivariate time series data. The advantages of using CNNs over RNNs are that they use fewer parameters than RNNs due to parameter sharing in the convolutional filters. Additionally, computations of CNNs are faster than RNNs, since they can be processed in parallel, while RNNs need sequential processing. An example of a CNN-based anomaly detection method is the model called DeepAnT proposed by Munir et al. [49], which is suitable for both univariate and multivariate time series data. The authors state that compared to an RNN-based approach, the model requires fewer data to train. However, their method can only detect anomalies but not interpret the root causes. Zhang et al. [78] proposed an anomaly detection model called MSCRED combining autoencoder architecture, LSTMs, and CNNs. Their method supports detecting anomalies from multivariate time series data and interpreting the root causes of the detected anomalies.

3. **Generative adversarial networks (GANs)**, consisting of a generator and discriminator networks, have been applied to detect anomalies by training the generator to reconstruct normal data instances and the discriminator to distinguish normal and anomalous data instances. Li et al. [40] proposed MAD-GAN, a model using LSTMs and GAN-based architecture. However, the model can only provide a binary label (normal or anomalous) for the data points, but it does not provide an anomaly interpretation. Additionally, the authors mention that their model is prone to the typical challenges present in GANs, where the model training can be unstable [40]. Khoshnevisan and Fan [33] extended the CNN-LSTM-based MSCRED [78] model by using its architecture in a GAN-based model named RSM-GAN. Their study showed

that RSM-GAN is better than MSCRED at detecting anomalies from multivariate time series data that contains seasonal variations and when the training data is contaminated with a small number of anomalies.

4. **Autoencoders (AEs)** are used for anomaly detection by training the model on normal data instances (a semi-supervised setting), which usually constitute the majority of the data in anomaly detection tasks. The trained model then fails to reconstruct unseen anomalous data samples, producing a large reconstruction error [12]. An example of such an AE-based anomaly detection model for multivariate time series data is the LSTM-based AE proposed by Malhotra et al. [48] that learns to reconstruct the normal patterns of the time series data and uses the reconstruction error of the input series to detect anomalies.

5. **Variational autoencoders (VAEs)** are used similarly for anomaly detection tasks as AEs. After training the model with normal data, anomalous instances have a lower reconstruction probability than normal instances. VAEs have been shown to outperform AEs on anomaly detection tasks, particularly on noisy data [67]. Park et al. [53] introduced an LSTM-based VAE that reconstructs the multivariate signals and reports an anomaly when an anomaly score based on the reconstruction likelihood is higher than a state-based dynamic threshold. Su et al. [67] proposed a model called OmniAnomaly that uses GRU cells and a VAE architecture for each time step of the input sequence to model the temporal dependencies and stochasticity of the multivariate time series. Additionally, they used several other techniques, such as normalizing flows and linear gaussian SSM as a prior for the latent variable distribution, to improve their model's performance.

The main disadvantages of DL-based anomaly detection methods are their need for a large amount of training data and the long training time of the models. In general, these algorithms need several epochs of training over a complete dataset, which makes most of them unsuitable for online learning setup [8]. Nevertheless, the DL-models are usually applicable to real-time anomaly detection in relatively stationary streaming data environments (see Section 2.6.6) since the inference time of the models is usually fast. Additionally, if the concept drifts occur gradually in such environments, the models can be retrained periodically on demand.

3.2 Anomaly Detection for Injection Molding Machines

Anomaly detection in the data produced by IMMs has been an attractive research topic since the injection molding process is widely employed in the plastic industry. Moreover, since both the IMMs and the available anomaly detection techniques keep developing, the subject remains relevant.

In an early work by Costa et al. [18], the authors demonstrated that ANNs could be successfully used to monitor IMMs' process KPIs. They used a simple RNN model with a single hidden layer with ten units to predict the injection time and material cushion size at the next time step using the cycle time, dosage time, injection time, and cushion size values from the previous time step.

Ribeiro [60] applied support vector machines (SVMs) to predict the quality properties of the injection molded parts using six different process measurements of an IMM as an input. The inputs were mapped to one of six possible fault categories, which presented different defects of the molded parts. The author also compared the performance of the SVM to a radial basis function (RBF) neural network, concluding that the SVM had better accuracy and better generalization to the problem. A disadvantage of Ribeiro's method is that the model uses supervised learning. It is costly and labor-intensive to continuously monitor the produced parts' quality properties to obtain training data. Moreover, the training data might become obsolete when the IMM's parameters are altered, which is sometimes necessary due to fluctuations in the raw material properties or when a machine part gets replaced. Such an action could shift the measured values of the process, requiring to restart the training data collection from the beginning.

Woll et al. [74] demonstrated that ANNs could be applied to monitor the quality of injection-molded products. Similar to Ribeiro's work, they trained the model to map the data patterns produced during each injection molding cycle to part quality. Their study consisted of two parts. First, they showed how a set of aggregated values collected from the process could predict the produced part's weight as a quality metric. Second, they demonstrated that the continuous measurements from a cavity pressure sensor from each cycle could predict the part length as a quality metric. The study showed that ANNs were superior in predicting part quality compared to traditional SPC techniques, which have been widely used in the injection molding process industry [74]. The method has similar disadvantages to Ribeiro's work since this approach also uses supervised learning.

Nagorny et al. [51] tested both CNN and LSTM architectures to predict

the quality of the injection molded parts using thermographic images of the molded parts and raw sensor signals. Their study showed that neural networks gave better prediction scores than classical regression algorithms. They measured the continuous sensor data of each signal at 100 Herz from each injection molding cycle lasting 30 seconds, producing 3000 samples per cycle. Afterward, they manually measured the produced part's width as a quality characteristic, which they then trained the models to predict using regression. Like the two studies reviewed earlier, their method employs supervised learning, requiring manual labor to map the input data to the produced part quality properties. Moreover, varying noise levels in the sensor readings and concept drifts due to external factors can make it challenging to obtain such mapping that would not become obsolete after some time.

Schiffers et al. [62] used an ensemble of DBSCAN algorithm to detect anomalous injection molding cycles in an unsupervised fashion. First, they explained how to select meaningful injection process features for their anomaly detection task. After identifying the useful features, they performed feature engineering to generate the final features they supplied to their algorithm. Additionally, to make the anomaly reporting of their method more specific, they grouped the final features into five separate feature sets based on the IMM's physical component groups. They also introduced a method to identify the individual features that can explain each detected anomaly's root cause based on the cluster ensembles. When clustering the final feature vectors, they considered data points in large clusters as normal but in small clusters as anomalous. Their approach showed promising results in identifying anomalous cycles. Since their method uses unsupervised learning, it requires less manual labor than the three previously investigated studies. However, their method requires manual feature engineering, which often requires assistance from the domain experts, requiring an additional manual step before applying the method.

Belkadi and Billaudel [7] developed an adaptive fault detection method to monitor the quality of parts produced by an injection molding machine by temporally segmenting the machine's continuous metrics according to the trends and shape of each metric over a specified time interval. They calculated signatures representing a model of each metric and stored them in a scalable database. They considered a part defective when the similarity value between the signatures generated during the operation and the reference metric was over an acceptable, adaptively determined threshold.

Chapter 4

Datasets

This chapter presents the datasets used in the experiments of this thesis, and highlights the main aspects of the data that are important to consider when determining a suitable anomaly detection approach.

4.1 Injection Molding Machine Data

The data analyzed in this thesis is provided by ABB Oy Wiring Accessories, a business unit within the ABB group, located in Porvoo, Finland. The production facility in Porvoo is highly automated, with industrial robots handling the assembly from the start to the product packaging. The factory produces various plastic-based electrical installation products, such as switches, sockets, and wiring accessories, using injection molding.

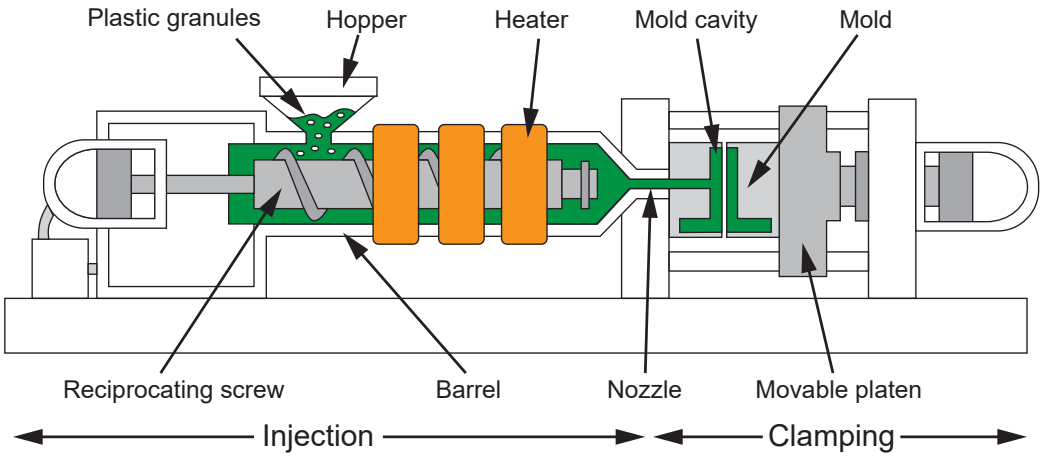


Figure 4.1: An injection molding machine.

The core parts of an IMM are visualized in Figure 4.1. In principle, the general injection molding process is relatively simple. A thermoplastic raw material, usually in the form of granules, is continuously fed through a feed hopper into a heated barrel with a reciprocating screw. The screw delivers the melted raw material forward towards the front of the screw, where a specific volume (a shot) of the material is injected into a cold mold using high pressure, filling the mold cavity. After the material has cooled down enough to solidify, the mold opens, and an array of pins are driven forward to extract the produced part. Then, the mold closes again, and the procedure is repeated, forming a cyclic process.

4.1.1 Data Origins

The production cell focused in this study (depicted in Figure 4.2) comprises two IMMs, named M1 and M2, and various robots handling moving the parts and packaging the final products. The cell produces AP9 and AP10 junction boxes (displayed in Figure 4.3) that consist of a box and a lid, made by M2 and M1, respectively. The outputted product type depends on the mold placed inside each machine and the product-specific machine configuration. Since the IMMs produce the actual products, they constitute the core part of the cell. Concurrently, they are the most complex part of the unit since the injection molding process contains several uncontrollable factors, which affect the stability of the process and the resulting part quality, such as the raw material quality variations and environmental factors.

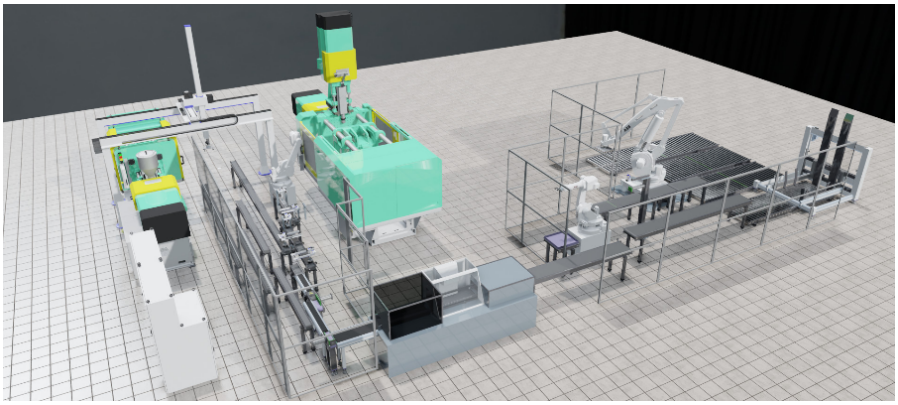


Figure 4.2: A 3D model of the highly automated ABB production cell that produces AP9 and AP10 junction boxes. The two injection molding machines (M1 is on the left, and M2 is on the right) are colored turquoise. Image by Ilpo Tanskanen, ABB Oy Wiring Accessories, Porvoo.

(a) ABB AP9 junction box. [3]

(b) ABB AP10 junction box. [2]

Figure 4.3: Output products of the production cell.

The model type of M1 is an Arburg Allrounder 520 A, which is a relatively standard IMM. The M2 is an Arburg Allrounder 720 A Multi-component with two separate injection units because the lids are composed of two different types of plastic. The IMMs have multiple sensors that collect process metrics from each phase of the injection molding cycle, producing multivariate time series data for monitoring the machine state. The data is accessible in real-time through the Open Platform Communications Unified Architecture (OPC-UA) server embedded in the machines. Figure 4.4 visualizes our proposed data monitoring pipeline for future production use. The proposed pipeline shows in high-level detail how the data could be transferred from the machines to the anomaly detection unit in real-time, located in a cloud platform outside the factory network (on demand).

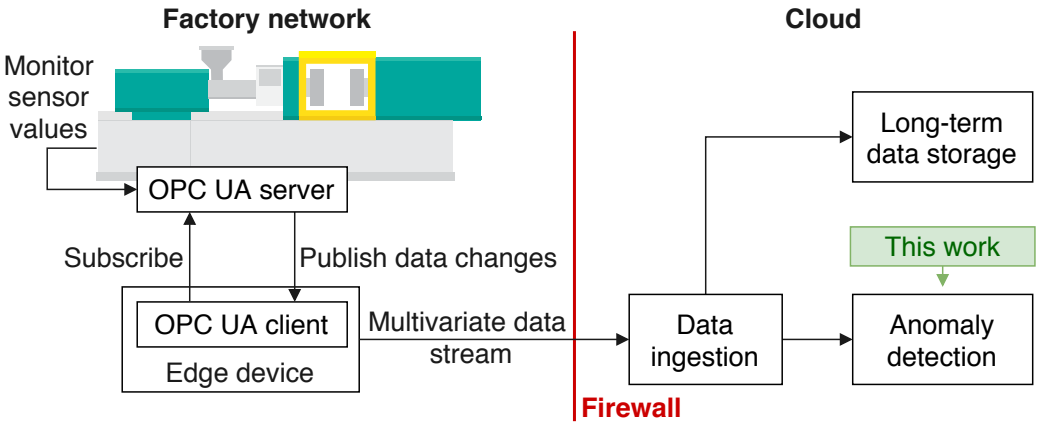


Figure 4.4: Proposed data monitoring pipeline for future production use.

Both machines provide read access to various KPIs measured from each injection molding cycle. The machine reports both continuous metrics, such as pressure curves, and aggregated metrics from the cycle, such as the cycle duration, injection time, and peak injection pressure. In this thesis, we focus on the aggregated KPIs from the cycles, i.e., we collect a set of KPIs from each cycle. This decision is constrained by the current data collection configuration set by the operators, where only aggregated IMM metrics are collected. On the other hand, this choice reflects the domain experts' interest in whether the aggregated metrics can be used for quality monitoring and predictive maintenance purposes.

4.1.2 Dataset Description

The injection molding data was collected from the two IMMs over nine days of a continuous production run, during which the production cell was manufacturing AP9 products with white color material. The data collected from each machine contains several aggregated KPIs, selected by the domain experts, measured from each injection molding cycle. Since the measurements were collected once per cycle, the data collection interval corresponds to the cycle time, approximately 16 seconds. However, there are irregularly longer delays between some cycles in the data due to occasional manual machine halts. For this thesis, we, regardless, consider that the consecutive sequence of cycles form a continuous time series, even if there is a larger than usual break between the cycles. This decision simplifies the experimental setup when we use the data for training and evaluating the selected models' anomaly detection performance.

We purposely focus on IMM data that originates from a single mold (same product) and a single color for the experiments of this thesis. We performed this selection because each mold has specific parameter settings that directly affect the machines' sensor readings. Similarly, different coloring choices of the same product affect the sensor readings since each color material requires its specific configuration. For instance, a particular color raw material might require a different cooling time compared to another material. Moreover, since the machine operators have only recently started the data collection from the machines, there is no enough data available to cover different scenarios simultaneously for our experiments. Hence, we focus on data originating from this combination of a mold and color for both IMMs.

Table 4.1 displays the statistical properties of the datasets collected from M1 and M2. Both datasets contain approximately the same number of samples (around 26 000) since M1 and M2 are jointly producing parts for a single product. The small difference in the number of samples between the two

datasets occurs because M2 had more calibration shots performed during the data collection than M1. The table shows that the M1 dataset contains 21 different aggregated KPIs (features) collected from the M1, and the M2 dataset contains 30 different KPIs obtained from M2. A full list of the features and their description is presented in Appendix A.

Dataset	Number of features	Dataset size (# cycles)	Anomaly ratio (labeled anomalies) (%)
M1	21	26 110	9.67
M2	30	26 468	12.82

Table 4.1: Dataset information.

We have labeled the anomalies in the datasets with assistance from the domain experts who provided specific rules to identify the anomalies. The majority of the anomalies are irregularities caused by the operators switching the machine's operating mode temporarily from automatic to manual mode, which commonly induces unusual behavior in particular KPIs. For example, if the machine operator notices that a part got stuck in the packaging line, the machine would be stopped, causing such an anomaly. The datasets contain both global point anomalies, which occur only within a single cycle, and anomalies spanning over multiple consecutive cycles, forming continuous anomalous regions. An example of a continuous anomaly would be a temperature in the barrel rising abnormally high and then slowly decreasing back to the normal level.

For the evaluation purposes of this thesis, we use the last 30% of the collected injection molding datasets to test the anomaly detection performance of the selected methods (data preprocessing steps are described later in Section 5.4). Therefore, we conducted the labeling of the anomalies in this part of the datasets carefully by hand to avoid inducing bias that could affect the evaluation of different anomaly detection methods. Additionally, since different algorithms might detect a continuous anomaly in a slightly different position, we labeled these continuous anomalous regions large enough to acknowledge this matter. Note that since this thesis focuses on unsupervised (and semi-supervised) anomaly detection, we do not use these labeled anomalies to train the models but to identify the regions of normal operation data that we provide to the models during training (a semi-supervised setup). Additionally, we use these labeled anomalies to evaluate the anomaly detection performance of the models.

Figure 4.5 presents a section of the multivariate time series data from the M1 dataset of seven selected KPIs with manually labeled anomalies. The

first occurring anomaly, A1, spans over multiple consecutive cycles, affects multiple KPIs, and is easily visible. This anomaly is an example of a deviation occurring when the machine operating mode is switched from automatic to manual and then back to automatic. The second anomaly, A2, is a smaller peak detectable in cycle time and three other metrics. The anomaly A4 is similar to A2 but is also visible in the screw peak injection pressure metric. The anomalies A3 and A5 are individual abnormal peaks occurring in the backpressure metric within a single cycle.

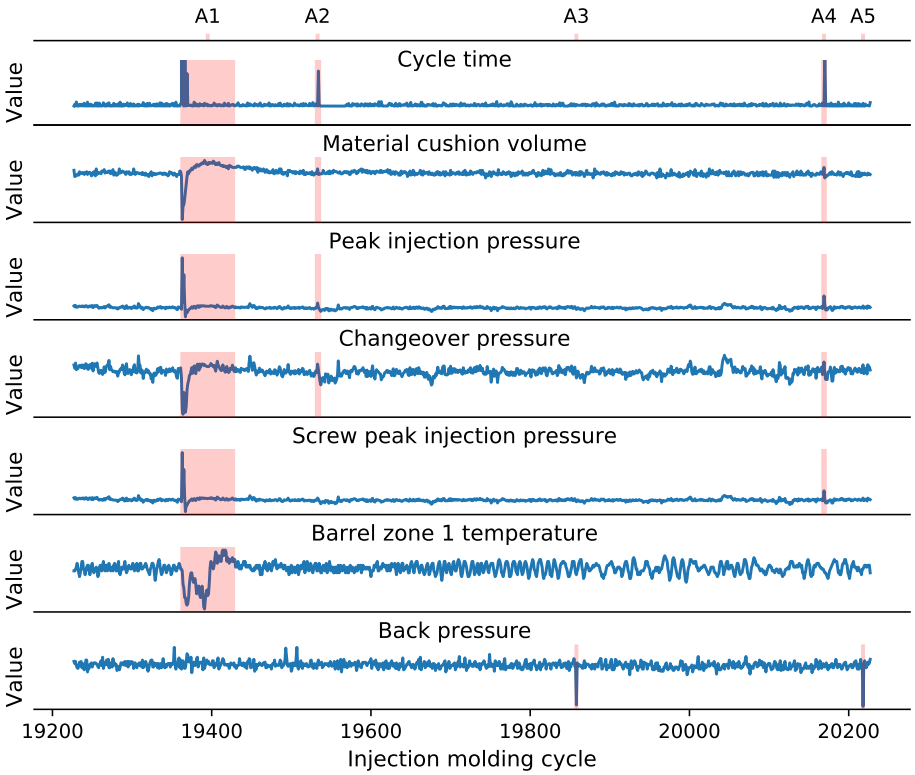


Figure 4.5: A section of multivariate time series data from the M1 dataset of seven selected metrics. The labeled anomalies (A1-A5) are highlighted in red.

Figure 4.6 displays seven selected metrics from the M2 dataset from a section of the multivariate time series data. The first detail we can see from the figure is that each visualized anomaly causes a detectable pattern in multiple metrics at the same time. The anomaly A2 differs from the other displayed anomalies (A1 and A3-A5) as it does not cause a high peak in the cycle time metric.

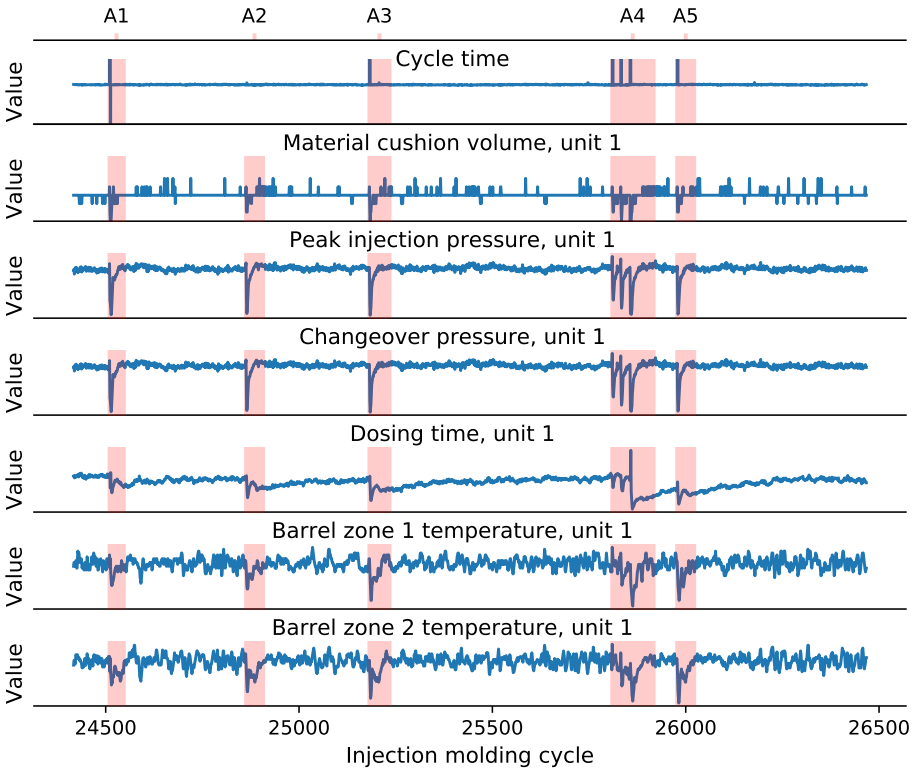


Figure 4.6: A section of multivariate time series data from the M2 dataset of seven selected metrics. The labeled anomalies (A1-A5) are highlighted in red.

Both Figure 4.5 and Figure 4.6 display important aspects of the collected multivariate time series data. First, we can see that the univariate time series components of the multivariate injection molding data are relatively stationary. This observation is expected because the collected data originates from a single mold and single color raw material, and thus, the machine's parameter settings are fixed to a certain value, which the measured KPIs should follow. However, there are varying levels of noise present in all the metrics. Some metrics, such as cycle time, stay relatively static, but metrics such as the changeover pressure and the barrel temperatures have larger fluctuations. The visualized data highlights the benefits of multivariate anomaly detection over univariate methods. First, many of the anomalies in the data affect multiple metrics at the same time. Second, since the measured values contain varying noise levels, detecting anomalies using a single metric could be difficult because the noise may obscure the anomalies. However, when analyzing

multiple metrics simultaneously, the occurrence of an anomaly becomes more evident, highlighting the benefits of multivariate anomaly detection.

4.2 Artificial Data

In addition to the real-world injection molding data, we generate artificial multivariate time series data to assess how well the evaluated anomaly detection methods perform on simulated data with temporal structure and artificially generated anomalies. Additionally, since the anomalies in the collected real-world data are manually labeled, the labeling is subjective and prone to bias. Hence, the generated artificial data also acts as a more objective test set in our experiments.

The generated artificial dataset contains $n = 80\,000$ samples with 20 dimensions. We use the first half of the dataset for training and the second half for evaluating the selected anomaly detection methods. We generate each univariate component of the multivariate time series using the equation

$$S(t) = \sin[(t - t_0)/F] + 0.3 \times \epsilon_t$$

where $t_0 \in [10, 100]$ is a randomly chosen shift in phase, $\epsilon_t \sim \mathcal{N}(0, 1)$ is random Gaussian noise, and $F \in [10, 40]$ controls the periodicity of the signal. We randomly inject 40 artificial anomalies in the second half (testing set) of the generated data. Each generated anomaly is randomly chosen to be either of contextual or collective type. Each contextual anomaly has a duration (in time steps) randomly selected from range $[1, 20]$, and the anomaly is a peak where intensity is selected from the range $[1.5, 2.5]$, which is added or subtracted to the value of each point within the range of the anomaly. Each collective anomaly has a duration in the range $[80, 160]$, and each point within the range is set to have the value equal to the value from the first point in the range, leading to all the values of the anomaly having the same constant value. Each generated anomaly has one to three root causes, i.e., each anomaly can appear in up to three dimensions. The labeling of these anomalies is adjusted so that the end range of each anomaly is moved 50 time steps further from the end of the actual anomaly to allow more flexible detection for the models for our evaluation purposes (we present the evaluation metrics in Section 5.5).

Figure 4.7 displays a section of the generated artificial data used in the experiments. There are five artificially generated anomalies A1-A5 visible in this section. The anomalies A1-A4 present contextual anomalies, and A5 is a collective anomaly. The figure also shows the duration (range) of

each anomaly, where the end of the range is adjusted according to procedure described earlier.

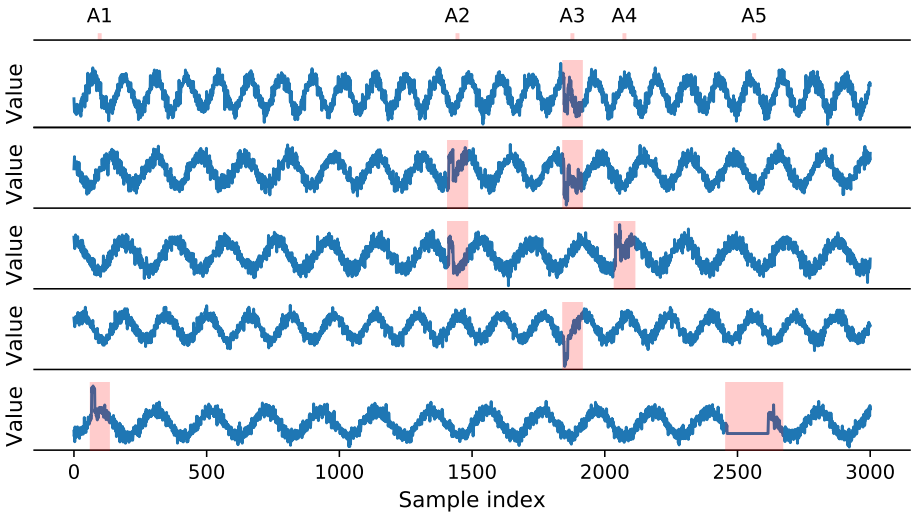


Figure 4.7: A section of the artificial multivariate time series data of five selected feature dimensions. The labeled anomalies (A1-A5) are highlighted in red.

Chapter 5

Methods

This chapter presents the proposed method for the multivariate time series anomaly detection problem considered in this thesis. After introducing the proposed method, the chapter presents five baseline algorithms from the literature to assess the proposed model's anomaly detection performance. Finally, the chapter describes the preprocessing steps performed on the data and the selected evaluation metrics.

5.1 OmniAnomaly

Section 3.1 presented a wide range of potential methods for anomaly detection in multivariate time series data. Based on the requirements of the problem under study, we chose the OmniAnomaly model proposed by Su et al. [67] for further investigation since the authors state that the model has the following features, which satisfy the requirements imposed by the research objectives described in Section 1.1:

1. The model needs only normal data for training (semi-supervised).
2. The model considers temporal relations in the multivariate data.
3. The (trained) model can detect anomalies in an online setting.
4. The model is robust to noise (use of VAE architecture).
5. The model can interpret the root causes of the detected anomalies.

This section presents the original OmniAnomaly model in detail. After presenting the model, we discuss some of its drawbacks and present our proposed improvements in Section 5.2.

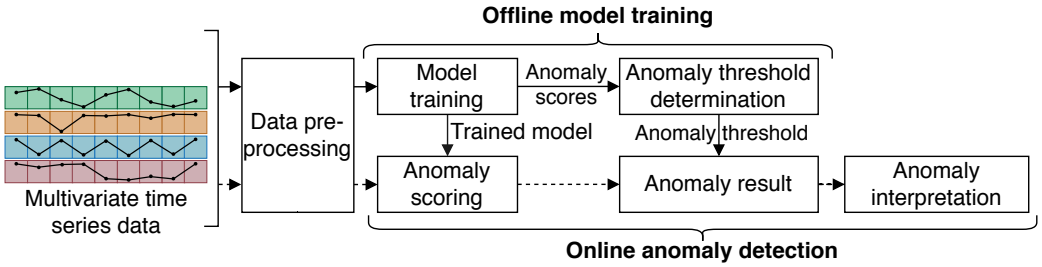


Figure 5.1: Overall structure of OmniAnomaly. Adapted from Su et al. [67].

Figure 5.1 displays the overall structure of OmniAnomaly's anomaly detection pipeline, consisting of two phases: offline training and online anomaly detection. In both phases, the input data is first normalized and then divided into sequences of length $T+1$ using a sliding window where $T+1$ is the size of the window.

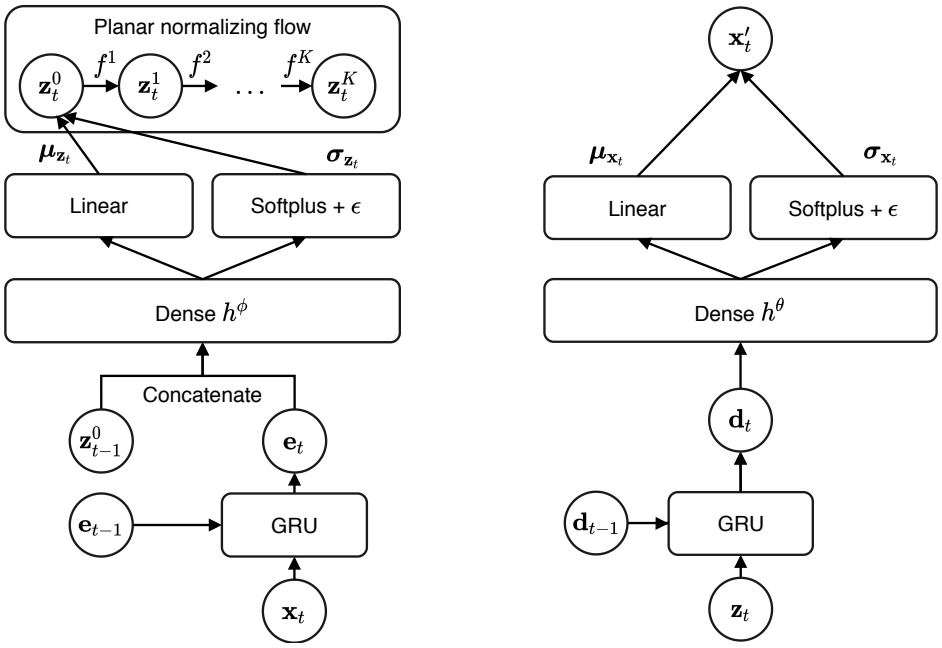
After preprocessing the training data, it is used to train the model offline. During the offline training phase, the model learns to capture the normal patterns of the multivariate time series data. After the training, the trained model computes anomaly scores for each training observation, which are used to automatically determine a suitable anomaly threshold using the POT method from EVT (described later in Section 5.1.4). The offline training step can be executed on-demand, such as daily, weekly, or monthly, depending on the characteristics of the data, taking into account possible concept drifts.

The online anomaly detection phase uses the trained model obtained during the offline training phase. The model can be used for online anomaly detection as follows. To compute an anomaly score for a new observation $\mathbf{x}_t$, the model receives the preprocessed observation $\mathbf{x}_t$ and T preceding observations before $\mathbf{x}_t$, which are used as contextual information to determine whether the latest observation is anomalous. If the computed anomaly score exceeds the anomaly threshold determined in the offline phase, the observation $\mathbf{x}_t$ is declared anomalous. Further details of the online anomaly detection phase are discussed in Section 5.1.3. Suppose $\mathbf{x}_t$ is identified as anomalous. In that case, the model allows explaining the anomaly by computing the reconstruction probability (negative log-likelihood) of each (feature) dimension of $\mathbf{x}_t$ and then ranking these scores (described in detail in Section 5.1.5).

5.1.1 Model Architecture

OmniAnomaly uses a VAE architecture where both encoder and decoder networks include a GRU layer to capture the temporal dependencies of the

observations. Additionally, the model uses several other techniques to capture the temporal dependencies within the sampled latent variables and to learn better latent presentations of the input data.



(a) Encoder (variational) net $q_\phi(\mathbf{z}|\mathbf{x})$ (b) Decoder (generative) net $p_\theta(\mathbf{x}|\mathbf{z})$

Figure 5.2: Architecture of OmniAnomaly at a time t. Adapted from Su et al. [67].

Input

At a time t, as an input, the model receives a sequence of observations $\mathbf{x} = (\mathbf{x}_{t-T}, \mathbf{x}_{t-T+1}, \ldots, \mathbf{x}_t)$, i.e., $\mathbf{x}_t$, and the T preceding observations of $\mathbf{x}_t$, which we denote by $\mathbf{x}_{t-T:t}$. In the following sections, we explain the model architecture based on an observation $\mathbf{x}_t$ at a time step t of the input sequence to simplify the notations, similar to the original work.

Encoder

The encoder part of the model is visualized in Figure 5.2a. At a time t, the GRU layer receives an observation $\mathbf{x}_t$ and the previous hidden state $\mathbf{e}_{t-1}$ to output a new hidden state $\mathbf{e}_t$. Then, $\mathbf{e}_t$ is concatenated with the latent presentation $\mathbf{z}_{t-1}^0$ sampled at the previous time step. The concatenated output is then passed to a dense layer h^ϕ with ReLU activation. The concatenation

is performed to allow the network to capture information from the previously sampled stochastic latent variables. Next, the output from the dense layer is passed to two separate dense layers to generate mean $\boldsymbol{\mu}_{\mathbf{z}_t}$ and standard deviation $\boldsymbol{\sigma}_{\mathbf{z}_t}$. This part of the encoder can be formulated as

$$\begin{aligned}\boldsymbol{\mu}_{\mathbf{z}_t} &= \mathbf{w}^{\boldsymbol{\mu}_\mathbf{z}} h^{\phi}([\mathbf{z}^0_{t-1}, \mathbf{e}_t]) + \mathbf{b}^{\boldsymbol{\mu}_\mathbf{z}} \\ \boldsymbol{\sigma}_{\mathbf{z}_t} &= \text{softplus}(\mathbf{w}^{\boldsymbol{\sigma}_\mathbf{z}} h^{\phi}([\mathbf{z}^0_{t-1}, \mathbf{e}_t]) + \mathbf{b}^{\boldsymbol{\sigma}_\mathbf{z}}) + \boldsymbol{\epsilon}^{\boldsymbol{\sigma}_\mathbf{z}}\end{aligned}$$

where $[\mathbf{z}^0_{t-1}, \mathbf{e}_t]$ is the concatenation of $\mathbf{z}^0_{t-1}$ and $\mathbf{e}_t$, $\text{softplus}(a) = \log(\exp(a)+1)$, and $\boldsymbol{\epsilon}^{\boldsymbol{\sigma}_\mathbf{z}}$ is a small positive number. Next, the latent presentation $\mathbf{z}^0_t$ is sampled from $q_0(\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}_{\mathbf{z}_t}, \boldsymbol{\sigma}^2_{\mathbf{z}_t}\mathbf{I})$ using the reparameterization trick:

$$\mathbf{z}^0_t = \boldsymbol{\mu}_{\mathbf{z}_t} + \boldsymbol{\epsilon}_t \odot \boldsymbol{\sigma}_{\mathbf{z}_t} \qquad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Finally, to learn a complex, non-Gaussian variational distribution $q_{\boldsymbol{\phi}}(\mathbf{z}_t|\mathbf{x}_t)$, the model uses K planar normalizing flows f (see Section 2.3.6) to transform the $\mathbf{z}^0_t$ from distribution q_0 into a more complex distribution q_K:

$$\mathbf{z}_t = \mathbf{z}^K_t = f^K(f^{K-1}(...f^1(\mathbf{z}^0_t)))$$

$$\log q_K(\mathbf{z}^K_t) = \log q_0(\mathbf{z}^0_t) \sum_{k=1}^{K} \log \left| \det \frac{\partial f_k}{\partial \mathbf{z}^{k-1}_t} \right|$$

Prior

As the prior $p_\theta(\mathbf{z}_t)$, OmniAnomaly uses linear gaussian SSM (see Section 2.4) to introduce a temporal structure between the latent variables:

$$p_\theta(\mathbf{z}_t) = p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}) = \mathbf{O}_\theta(\mathbf{T}_\theta \mathbf{z}_{t-1} + \mathbf{v}_t) + \boldsymbol{\epsilon}_t$$

where $\mathbf{T}_\theta$ and $\mathbf{O}_\theta$ are the transition and observation matrices, and $\mathbf{v}_t$ and $\boldsymbol{\epsilon}_t$ are the transition and observation noises. The purpose of introducing a temporal structure into the prior distribution is that it could help the latent variables better capture the temporal relations of the data.

Decoder

The decoder part of the model attempts to reconstruct $\mathbf{x}_t$ using the encoded latent presentation $\mathbf{z}_t = \mathbf{z}^K_t$. First, at a time t, the GRU layer receives a sampled latent variable $\mathbf{z}_t$ outputted by the encoder and the previous hidden state $\mathbf{d}_{t-1}$, which then outputs the next hidden state $\mathbf{d}_t$. The output of the GRU, $\mathbf{d}_t$, is then passed through a dense layer h^θ with ReLU activation.

Then, the output of the dense layer is given to two separate dense layers to generate mean $\boldsymbol{\mu}_{\mathbf{x}_t}$ and standard deviation $\boldsymbol{\sigma}_{\mathbf{x}_t}$:

$$\boldsymbol{\mu}_{\mathbf{x}_t} = \mathbf{w}^{\boldsymbol{\mu}_\mathbf{x}} h^\theta(\mathbf{d}_t) + \mathbf{b}^{\boldsymbol{\mu}_\mathbf{x}}$$
$$\boldsymbol{\sigma}_{\mathbf{x}_t} = \text{softplus}(\mathbf{w}^{\boldsymbol{\sigma}_\mathbf{x}} h^\theta(\mathbf{d}_t) + \mathbf{b}^{\boldsymbol{\sigma}_\mathbf{x}}) + \boldsymbol{\epsilon}^{\boldsymbol{\sigma}_\mathbf{x}}$$

The final decoding distribution $p_\theta(\mathbf{x}_t|\mathbf{z}_t)$ is presented by a multivariate Gaussian with a diagonal covariance matrix, $\mathcal{N}(\boldsymbol{\mu}_{\mathbf{x}_t}, \boldsymbol{\sigma}^2_{\mathbf{x}_t}\mathbf{I})$. This distribution is a core part of the model because it is used to compute the anomaly score for the observation $\mathbf{x}_t$ by computing the negative log-likelihood $-\log p_\theta(\mathbf{x}_t|\mathbf{z}_t)$. If the observation at time t is anomalous, the reconstruction probability of the sample $\mathbf{x}_t$ would be low, and consequently, the anomaly score high. Computing the anomaly score is further discussed in Section 5.1.3, and using the score for anomaly interpretation is described in Section 5.1.5.

5.1.2 Model Training

The parameters of the network are learned similar to a standard VAE, where the model is trained by optimizing the negative ELBO, similarly as described in Section 2.3.5. Each training sample of OmniAnomaly is a sequence $\mathbf{x}_{t-T:t}$ of length $T+1$. The objective function to be minimized can be formulated as

$$L(\mathbf{x}_{t-T:t}) = \frac{1}{L}\sum_{l=1}^{L} \underbrace{-\log p_\theta(\mathbf{x}_{t-T:t}|\mathbf{z}^l_{t-T:t})}_{\text{Reconstruction error}} + \underbrace{\log q_\phi(\mathbf{z}^l_{t-T:t}|\mathbf{x}_{t-T:t}) - \log p_\theta(\mathbf{z}^l_{t-T:t})}_{\text{KL divergence}}$$

where L is the number of MC samples drawn for each latent variable at each time step. The first term (reconstruction error) in the objection function above corresponds to the negative log-likelihood $-\log p_\theta(\mathbf{x}_{t-T:t}|\mathbf{z}_{t-T:t}) = -\sum_{i=t-T}^{t} \log p_\theta(\mathbf{x}_i|\mathbf{z}_{t-T:i})$ where $p_\theta(\mathbf{x}_i|\mathbf{z}_{t-T:i}) = \mathcal{N}(\boldsymbol{\mu}_{\mathbf{x}_i}, \boldsymbol{\sigma}^2_{\mathbf{x}_i}\mathbf{I})$. The difference of the two last terms is the KL divergence, which act as a regularization to the loss. The first term in the KL divergence is the variational approximation of the true posterior distribution of the latent variables: $\log q_\phi(\mathbf{z}_{t-T:t}|\mathbf{x}_{t-T:t}) = \sum_{i=t-T}^{t} \log(q_\phi(\mathbf{z}_i|\mathbf{z}_{i-1}, \mathbf{x}_{t-T:i}))$. The last term is the prior: $\log p_\theta(\mathbf{z}^l_{t-T:t}) = \sum_{i=t-T}^{t} \log p_\theta(\mathbf{z}_i|\mathbf{z}_{i-1})$.

5.1.3 Online Anomaly Detection

The trained model can be used for online anomaly detection as follows. First, note that the input of OmniAnomaly is always a sequence of length $T+1$.

Hence, to compute the anomalousness of a new observation $\mathbf{x}_t$ at a time step t in a data stream, the model receives the sequence $\mathbf{x}_{t-T:t}$ as an input, i.e., $\mathbf{x}_t$ and the T preceding observations before $\mathbf{x}_t$. To compute the anomaly score of $\mathbf{x}_t$, the model utilizes the decoding distribution $p_\theta(\mathbf{x}_t|\mathbf{z}_{t-T:t})$, which allows us to compute the likelihood of the original input $\mathbf{x}_t$ given the compressed latent presentation $\mathbf{z}_{t-T:t}$. The idea is that if $\mathbf{x}_t$ resembles normal data that the model has encountered during training, the likelihood of the input observation would be high because the latent presentation represents the input well. On the other hand, if $\mathbf{x}_t$ is anomalous and different from the normal data, the likelihood of the anomalous input given the compressed latent presentation would be low since the latent presentation cannot present the anomalous input very well. Note that we use negative reconstruction log-likelihood as the final anomaly score of the model, i.e., $S_t = -\log p_\theta(\mathbf{x}_t|\mathbf{z}_{t-T:t})$, which is contrary to the original paper [67], where the authors directly use the log-likelihood for scoring. Using the negative log-likelihood as an anomaly score makes the model's output consistent with other anomaly detection models used in this thesis, where a higher anomaly score indicates a more anomalous sample.

5.1.4 Automatic Threshold Selection

Choosing a suitable threshold for the anomaly scores is a fundamental part of an anomaly detection method. In OmniAnomaly, the authors proposed using the POT technique from EVT (see Section 2.5) to select the threshold offline using the training data. Note that since we use the negative log-likelihood as the anomaly score of the model, the formulation presented here is changed accordingly. First, during the offline training phase, the trained model computes an anomaly score for each training observation. These computed scores form a univariate time series $(S_1, S_2, \ldots, S_{N'})$ where N' is the number of training samples. Then, to obtain a suitable anomaly threshold, the model applies the POT method on the sequence of anomaly scores. As described in Section 2.5, the second theorem of EVT can be written as

$$\bar{F}_t(s) = P(S - t > s \mid S > t) \sim (1 + \frac{\gamma s}{\sigma(t)})^{-\frac{1}{\gamma}}$$

where t is the initial threshold, γ and $\sigma(t)$ are shape and scale parameters of GPD, S is a value in the anomaly score series $(S_1, S_2, \ldots, S_{N'})$. The parameters $\hat{\gamma}$ and $\hat{\sigma}$ are estimated using MLE. After obtaining the parameters of the GDP using MLE, the final threshold is then computed as:

$$t_F \simeq t + \frac{\hat{\sigma}}{\hat{\gamma}}((\frac{qn}{N_t})^{-\hat{\gamma}} - 1)$$

where q is the desired probability to observe $S > t$, n is the total number of observations, and N_t is the number of S_i where $S_i > t$. The POT method has two hyperparameters that need to be adjusted empirically to find a suitable anomaly threshold for the model: the desired probability q and the initial threshold t. Siffer et al. [65] recommend choosing q in the range $[10^{-3}, 10^{-5}]$ and initial t as the 98% quantile of the data.

5.1.5 Anomaly Interpretation

OmniAnomaly outputs the anomaly score as negative log-likelihood of the data, i.e., $-p_\theta(\mathbf{x}_t|\mathbf{z}_{t-T:t}) \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{x}_t}, \boldsymbol{\sigma}^2_{\mathbf{x}_t}\mathbf{I})$. After detecting an anomaly, the model allows for interpretation of the root causes of the anomaly, i.e., the individual features (dimensions) of the multivariate observation that contribute to the anomaly. Since the anomaly score $-p_\theta(\mathbf{x}_t|\mathbf{z}_{t-T:t})$ is a multivariate Gaussian distribution with a diagonal covariance matrix, it can be factorized as a product of univariate Gaussians of each dimension, or as a sum of each dimension when using the log-likelihood:

$$-\log p_\theta(\mathbf{x}_t|\mathbf{z}_{t-T:t}) = -\sum_{i=1}^{D} \log p_\theta(x_t^i \mid \mathbf{z}_{t-T:t})$$

where D is the dimensionality of an observation $\mathbf{x}_t$. Thus, the anomaly score of an observation $\mathbf{x}_t$ can be expressed as $S_t = -\sum_{i=1}^{D} S_t^i$, where $S_t^i = -\log p_\theta(x_t^i|\mathbf{z}_{t-T:t})$ denotes the anomaly score of x_t^i. Using this formulation, after detecting an observation $\mathbf{x}_t$ as anomalous, we can interpret its root causes by estimating the reconstruction probability separately for each dimension of $\mathbf{x}_t$. The resulting scores S_t^i $(1 \leq i \leq D)$ are sorted in descending order, forming a list AS_t. The higher the rank that an individual dimension x_t^i has in AS_t, the more it contributes to the detected anomaly $\mathbf{x}_t$. Thus, human operators can directly use the ordered list AS_t for anomaly interpretation. The idea is that the ranked features should provide enough clues for the operators to understand the physical root cause of the detected anomaly.

5.2 Improving OmniAnomaly

This section describes some shortcomings we discovered in OmniAnomaly and presents our suggestions to improve it.

5.2.1 Issues Identified in OmniAnomaly

The first issue of OmniAnomaly that we encountered is that the GitHub implementation [1] of the model provided by the authors does not fully follow the architecture described in the original paper [67]. For instance, the architecture presented in the paper uses a dense layer with ReLU activation in both encoder and decoder networks, but the GitHub implementation uses linear layers instead. Additionally, the implementation of the recurrent latent variable distribution in the encoder part of the model seems to miscalculate the samples' log probability. This error is visible in the GitHub implementation, where the hidden state of the encoder GRU cell is concatenated with the sampled latent variable of the same time step as the hidden state instead of using the latent sample from the previous time step, violating the structure described in the paper. These issues challenge the overall quality of the GitHub implementation.

We performed extensive tests with OmniAnomaly using the artificial data presented in Section 4.2 applying both a model following the original paper's architecture and the GitHub implementation from the original authors. Besides, we tested the model with various configurations, such as disabling normalizing flows and using simple multivariate Gaussian as the prior distribution for the latent variables instead of using the linear Gaussian SSM. We also tested the model without using the latent variable concatenation procedure in the encoder network and using bidirectional GRU layers. However, none of the model variations we tested could learn the autocorrelation structure of the normal data well.

During our experiments, OmniAnomaly was able to reconstruct only a subset of the univariate components of the multivariate time series depending on the chosen size of the latent dimensionality. However, even when increasing the latent dimensionality close to the input dimensionality, the results were unsatisfactory, as the reconstruction of normal data was poor. Figure 5.3 displays an example of the poor reconstruction of the artificial data from one of our experiments using a configuration similar to the one described in the original paper. These challenges prevented us from using OmniAnomaly for the actual anomaly detection experiments considered in this thesis. Therefore, we concluded that we needed to propose adjustments to the model architecture to improve its performance, which we present in the next section.

The final drawback of OmniAnomaly that we mention is that it assumes the training data to be anomaly-free. However, as described in Section 4.1.2,

[1] `OmniAnomaly` codebase: `https://github.com/NetManAIOps/OmniAnomaly/`. Using the version of the commit bd5262e on 28.5.2020 (latest available as of 7.12.2020).

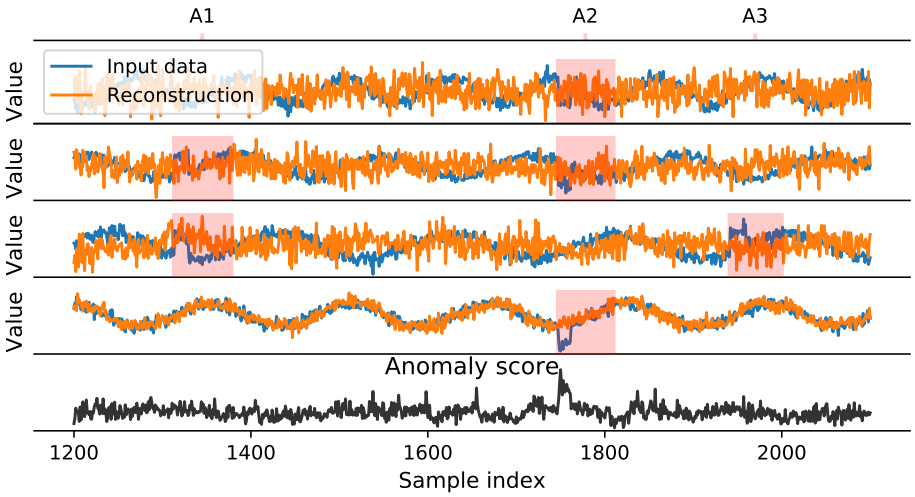


Figure 5.3: An example of a poor reconstruction of normal data and the anomaly scores computed by OmniAnomaly for a section of the artificial data of four chosen dimensions (out of 20). Labeled (true) anomalies are highlighted in red.

the injection molding datasets used in this study contain anomalies relatively often in the collected data. The lack of long continuous anomaly free (normal) data segments makes it challenging to use the collected data to train the model without performing some additional steps that address the anomalies in the training sets. If we supplied the raw data to train the model, the model could also learn to reconstruct anomalous data patterns well, resulting in poor anomaly detection performance. Therefore, we also address this issue in the next section with our proposal.

5.2.2 Proposed Improvements

Due to the issues outlined in the previous section, we propose several improvements to OmniAnomaly using ideas from related studies that demonstrated to improve the anomaly detection performance of the model in our experiments.

Using Both Static and Dynamic Latent Presentation

The main challenge we encountered with OmniAnomaly was that the model failed to reconstruct the normal patterns of the artificial data well in our experiments, leading to poor anomaly detection performance. Hence, we

investigated different architectural variations to improve the model based on the recent techniques proposed in the literature.

Inspired by the disentangled sequential autoencoder proposed by Li and Mandt [41] we propose to modify the OmniAnomaly model architecture to learn a disentangled latent representation of the input sequence, consisting of a static component capturing global features of the sequence and a dynamic component capturing time-dependent features of the sequence. The idea is that having two different latent variable representations of the input sequence should help the model to capture aspects of the sequence better since OmniAnomaly only uses time-varying latent representations of the sequence.

Like OmniAnomaly, the input of the model at each time step t is a sequence $\mathbf{x}_{t-T:t}$. The proposed model architecture uses the following probabilistic generative model:

$$p_\theta(\mathbf{x}_{t-T:t}, \mathbf{z}_{t-T:t}, \boldsymbol{f}) = p_\theta(\boldsymbol{f}) \prod_{i=t-T}^{t} p_\theta(\mathbf{z}_i) p(\mathbf{x}_i | \mathbf{z}_i, \boldsymbol{f}),$$

where $\boldsymbol{f}$ and $\mathbf{z}_i$ are the latent variables, where $\boldsymbol{f}$ captures global features of the input sequence, while $\mathbf{z}_i$ models time-dependent features. The generation of observation $\mathbf{x}_i$ depends on both latent variables $\mathbf{z}_i$ and $\boldsymbol{f}$.

The encoder part of the model uses the following factorization of the variational distribution q:

$$q_\phi(\mathbf{z}_{t-T:t}, \boldsymbol{f} | \mathbf{x}_{t-T:t}) = q_\phi(\boldsymbol{f} | \mathbf{x}_{t-T:t}) \prod_{i=t-T}^{t} q_\phi(\mathbf{z}_i | \mathbf{x}_i)$$

where we assume that the input sequence's global features are independent of the time-varying features of the sequence.

We discuss the final model architecture in detail in Section 5.2.3. In the next section, we represent the modified objective function that can be used to train this generative model.

Modifying Objective Function to Ignore Labeled Anomalies

As mentioned in Section 4.1.2, the collected injection molding datasets contain anomalies throughout the data, meaning that there are anomalies in the training sets. However, the model should not learn any of these known anomalous patterns. While one could replace each anomalous pattern in the training data with the corresponding "normal pattern", this could lead to suboptimal performance if the generated normal pattern does not fully represent the expected behavior of the data [76]. Therefore, inspired by Xu et

al. [76] we propose to modify the objective function of the model to ignore the labeled anomalies in the training set by supplying the model labels of the known anomalies during training. If no labeled anomalies exist in the training set, the model can be trained in an unsupervised fashion, assuming that most of the data consists of normal samples.

The new objective function of the model is

$$L(\mathbf{x_{t-T:t}}) = \frac{1}{L}\sum_{l=1}^{L}\Bigg[\underbrace{-\sum_{i=t-T}^{t}\left[\alpha_i \log p_\theta(\mathbf{x}_i|\mathbf{z}_i^l, \boldsymbol{f})\right]}_{\text{Reconstruction error}} + \underbrace{\log q_\phi(\mathbf{z}_{t-T:t}^l|\mathbf{x}_{t-T:t}) - \beta \log p_\theta(\mathbf{z}_{t-T:t}^l)}_{\text{KL divergence (dynamic feature encoder)}} + \underbrace{\log q_\phi(\boldsymbol{f}^l|\mathbf{x}_{t-T:t}) - \beta \log p_\theta(\boldsymbol{f}^l)}_{\text{KL divergence (static feature encoder)}}\Bigg]$$

where L is the number of MC samples for dynamic latent variables $\mathbf{z}_t$ and the static latent presentation $\boldsymbol{f}$. We use $\alpha_i = 1$ to indicate a normal observation in a sequence and $\alpha_i = 0$ to express an anomaly that should be ignored. Therefore, when $\alpha_i = 0$, the contribution of $\log p_\theta(\mathbf{x}_i|\mathbf{z}_{t-T:i}^l, \boldsymbol{f})$ is excluded from the loss calculation. The term $\beta = (\sum_{i=t-T}^{t} \alpha_i)/T$ acts as a scaling factor, which reduces the contribution of the dynamic prior $\log p_\theta(\mathbf{z}_{t-T:t}^l)$ and the static prior $\log p_\theta(\boldsymbol{f}^l)$ according to the ratio of normal points in the input sequence $\mathbf{x}_{t-T:t}$. The contributions of the variational distributions q are not modified, because unlike the priors that are part of the generative model (i.e., they contribute to modeling the normal patterns of the data), the variational distributions express a mapping from the input to the latent space, without considering normal patterns [76]. This modified objective function allows the proposed model to reconstruct the normal patterns of the input sequence $\mathbf{x}_{t-T:t}$, even if some points in the sequence are anomalous.

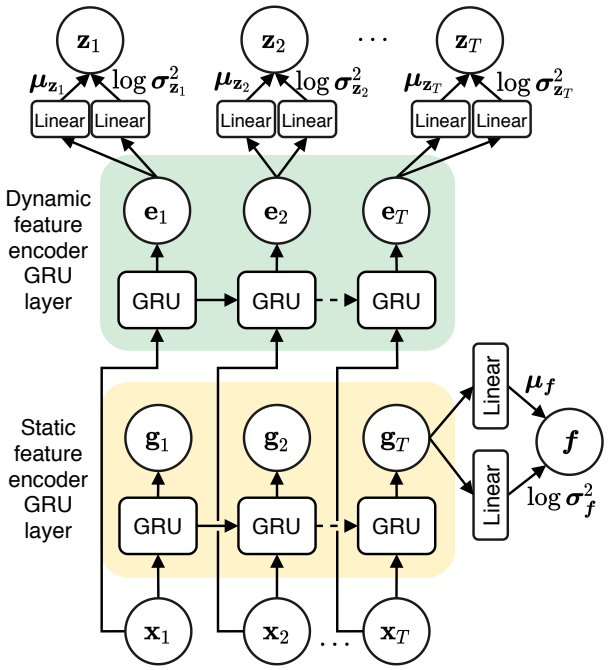
Using Logarithms When Computing Distribution Parameters

OmniAnomaly uses layers that directly output standard deviation in encoder ($\boldsymbol{\sigma}_{\mathbf{z}_t}$) and decoder ($\boldsymbol{\sigma}_{\mathbf{x}_t}$) networks through a layer with a softplus activation function and a small epsilon. However, since the standard deviation values are often very small, this forces the optimization with very small numbers, which could cause numerical instabilities during gradient calculation. Therefore, we propose to replace these layers with linear layers that output $\log \boldsymbol{\sigma}_{\mathbf{z}_t}^2$ and $\log \boldsymbol{\sigma}_{\mathbf{x}_t}^2$ instead. Outputting the logarithm of variance forces the network

layer to have an output range of natural numbers instead of small positive numbers, and therefore computations with log and exp are numerically stable operations for computers.

5.2.3 Final Model Architecture

This section describes the architecture of our model in detail. Note that in our experiments, using normalizing flows, using linear Gaussian SSM as a prior, or using the latent variable concatenation in the encoder, did not improve the model's anomaly detection performance. Therefore, we discarded these techniques in the final proposed model architecture. Our model uses simple multivariate Gaussian priors for both the static feature encoder and the dynamic feature encoder. We refer to the proposed model as DSVAE-AD (disentangled sequential VAE-based anomaly detector). Note also that apart from the different architecture, the proposed model operates similarly to OmniAnomaly. Hence, it shares the same offline training and online anomaly detection pipeline, similar anomaly scoring technique, and threshold selection approach. Figure 5.4 visualizes our proposed model architecture.



(a) Encoder network architecture for input sequence $\mathbf{x}_{1:T}$.

(b) Decoder network architecture at a time step t.

Figure 5.4: Architecture of the proposed model, DSVAE-AD.

Dynamic Feature Encoder

At a time t, the dynamic feature encoder GRU layer receives an observation $\mathbf{x}_t$ and the previous hidden state $\mathbf{e}_{t-1}$ to output a new hidden state $\mathbf{e}_t$. The output from this layer is used to generate mean $\boldsymbol{\mu}_{\mathbf{z}_t}$ and log variance $\log \boldsymbol{\sigma}^2_{\mathbf{z}_t}$. This part of the encoder can be formulated as

$$\boldsymbol{\mu}_{\mathbf{z}_t} = \mathbf{w}^{\boldsymbol{\mu}_\mathbf{z}}\mathbf{e}_t + \mathbf{b}^{\boldsymbol{\mu}_\mathbf{z}}$$
$$\log \boldsymbol{\sigma}^2_{\mathbf{z}_t} = \mathbf{w}^{\boldsymbol{\sigma}^2_\mathbf{z}}\mathbf{e}_t + \mathbf{b}^{\boldsymbol{\sigma}^2_\mathbf{z}}$$

Finally, the latent presentation $\mathbf{z}_t$ is sampled from $q_\phi(\mathbf{z}_t|\mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}_{\mathbf{z}_t}, \boldsymbol{\sigma}^2_{\mathbf{z}_t}\mathbf{I})$ using the reparameterization trick:

$$\mathbf{z}_t = \boldsymbol{\mu}_{\mathbf{z}_t} + \boldsymbol{\epsilon}_t \odot \exp(\frac{1}{2}\log \boldsymbol{\sigma}^2_{\mathbf{z}_t}) \qquad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Static Feature Encoder

The static feature encoder receives the input sequence $\mathbf{x}_{t-T:t}$, which is given to a GRU layer. The final GRU cell in the layer outputs the final hidden state $\mathbf{g}_T$ that encodes a presentation of the whole input sequence. This final hidden state $\mathbf{g}_T$ is then given to two separate linear layers to generate mean $\boldsymbol{\mu}_{\boldsymbol{f}}$ and log variance $\log \boldsymbol{\sigma}^2_{\boldsymbol{f}}$. This part of the encoder can be formulated as

$$\boldsymbol{\mu}_{\boldsymbol{f}} = \mathbf{w}^{\boldsymbol{\mu}_{\boldsymbol{f}}}\mathbf{g}_T + \mathbf{b}^{\boldsymbol{\mu}_{\boldsymbol{f}}}$$
$$\log \boldsymbol{\sigma}^2_{\boldsymbol{f}} = \mathbf{w}^{\boldsymbol{\sigma}^2_{\boldsymbol{f}}}\mathbf{g}_T + \mathbf{b}^{\boldsymbol{\sigma}^2_{\boldsymbol{f}}}$$

Finally, the static latent presentation $\boldsymbol{f}$ of the input sequence is sampled from $q_\phi(\boldsymbol{f}|\mathbf{x}_{t-T:t}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{f}}, \boldsymbol{\sigma}^2_{\boldsymbol{f}}\mathbf{I})$ using the reparameterization trick:

$$\boldsymbol{f} = \boldsymbol{\mu}_{\boldsymbol{f}} + \boldsymbol{\epsilon} \odot \exp(\frac{1}{2}\log \boldsymbol{\sigma}^2_{\boldsymbol{f}}) \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Decoder

At each time step, the decoder network attempts to reconstruct $\mathbf{x}_t$ using the encoded dynamic latent presentation $\mathbf{z}_t$ and the static latent vector of the sequence, $\boldsymbol{f}$. At a time step t, the decoder network receives a sampled latent variable $\mathbf{z}_t$ and $\boldsymbol{f}$, which are concatenated. Then, the GRU layer receives the concatenated inputs and the previous hidden state $\mathbf{d}_{t-1}$, which then outputs the next hidden state $\mathbf{d}_t$. The output of the GRU, $\mathbf{d}_t$, is then given to separate linear layers to generate mean $\boldsymbol{\mu}_{\mathbf{x}_t}$ and log variance $\log \boldsymbol{\sigma}^2_{\mathbf{x}_t}$:

$$\boldsymbol{\mu}_{\mathbf{x}_t} = \mathbf{w}^{\boldsymbol{\mu}_\mathbf{x}}\mathbf{d}_t + \mathbf{b}^{\boldsymbol{\mu}_\mathbf{x}}$$
$$\log \boldsymbol{\sigma}^2_{\mathbf{x}_t} = \mathbf{w}^{\boldsymbol{\sigma}^2_\mathbf{x}}\mathbf{d}_t + \mathbf{b}^{\boldsymbol{\sigma}^2_\mathbf{x}}$$

Like OmniAnomaly, the final decoding distribution $p_\theta(\mathbf{x}_t|\mathbf{z}_t, \boldsymbol{f})$ is a multivariate Gaussian with a diagonal covariance matrix, $\mathcal{N}(\boldsymbol{\mu}_{\mathbf{x}_t}, \boldsymbol{\sigma}^2_{\mathbf{x}_t}\mathbf{I})$, that is used to compute anomaly scores for each observation $\mathbf{x}_t$ and interpret the root causes of the detected anomalies.

5.3 Baseline Methods

To evaluate the proposed model, we compare its performance to five baseline models from the literature. The baseline models include simple statistical algorithms, classical machine learning algorithms, and a DL-based method. This section provides a brief description of each baseline method.

5.3.1 Multivariate Exponentially Weighted Moving Average

As the first baseline method, we use the MEWMA control chart [43], a commonly applied multivariate method in statistical process control (SPC) [46]. MEWMA assumes the monitored data, $\mathbf{x}_i \in \mathbb{R}^d$, $i = 1, 2, \ldots,$ to be independent multivariate normal vectors, making it underperform when the data contains serial correlation. Despite this, we use this simple method for comparison to see how neglecting the autocorrelation structure of the data affects the anomaly detection performance in our experiments. Another disadvantage of MEWMA is that the chart does not provide an easy way to interpret the detected multivariate anomalies, i.e., identifying the individual features that can explain the detected anomaly [43].

Given multivariate observations $\mathbf{x}_1, \mathbf{x}_2, \ldots,$ the MEWMA chart is based on the vectors

$$\mathbf{z}_i = \mathbf{r}\mathbf{x}_i + (\mathbf{1} - \mathbf{r})\mathbf{z}_{i-1}$$

where $i = 1, 2, \ldots,$ $\mathbf{z}_0 = \mathbf{0}$ and $\mathbf{r} = diag(r_1, r_2, \ldots, r_d)$, $0 < r_j \leq 1$, $j = 1, 2, \ldots, d$. We consider the case where there is no a priori reason to weight the past observations of the d variables differently, i.e., $r_1 = r_2 = \cdots = r_d = r$, allowing to express the MEWMA vectors as

$$\mathbf{z}_i = r\mathbf{x}_i + (1 - r)\mathbf{z}_{i-1}.$$

To detect anomalies, MEWMA chart gives an out-of-control-signal when $\mathbf{z}_i^T \boldsymbol{\Sigma}_{\mathbf{z}_i}^{-1} \mathbf{z}_i > h$ where $\boldsymbol{\Sigma}_{\mathbf{z}_i}$ is the covariance matrix of $\mathbf{z}_i$ and h is the threshold used to achieve the specified in-control average run length (ARL), where the run length is defined as the number of samples before the chart produces

the out-of-control-signal [46]. When $r_1 = r_2 = \cdots = r_d = r$, $\mathbf{\Sigma}_{\mathbf{z}_i}$ can be calculated as

$$\mathbf{\Sigma}_{\mathbf{z}_i} = \frac{r}{2-r}[1-(1-r)^{2i}]\mathbf{\Sigma}$$

where $\mathbf{\Sigma}$ is estimated from the historical training data of normal process behavior [43].

5.3.2 Vector Autoregression and Hotelling's T-Squared

As the second baseline method, we use vector autoregression (VAR) to model the serial correlation in the data, and Hotelling's T^2 statistic to detect anomalies in residuals between the predicted $\hat{\mathbf{x}}_t$ and observed $\mathbf{x}_t$ values of the data, i.e., $\hat{\mathbf{e}}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$, following the method proposed by Pan and Jarrett [52]. We refer to this method as VAR-T^2 in the later parts of this thesis. Note that similar to MEWMA, this model can only perform anomaly detection but cannot interpret the root causes of the anomalies since it is not achievable with the plain Hotelling's T^2 chart (see Section 3.1.1).

During the offline phase, we fit the VAR model using historical training data consisting of normal samples. Then, to use the model for anomaly detection, we apply Hotelling's T^2 statistic on the residuals $\hat{\mathbf{e}}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$ between the predicted and observed values of the data as follows,

$$T_t^2 = \hat{\mathbf{e}}_t^T \mathbf{\Sigma}_{\hat{\mathbf{e}}}^{-1} \hat{\mathbf{e}}_t$$

where $\mathbf{\Sigma}_{\hat{\mathbf{e}}}$ is the covariance matrix of $\hat{\mathbf{e}}$, estimated from the historical training data. Anomaly threshold is the upper control limit (UCL),

$$\text{UCL} = \frac{p(n+1)(n-1)}{n(n-p)} F_{\alpha,p,n-p}$$

where F represents an F-distribution, α is the chosen confidence interval, p is the number of variables (i.e., the dimensionality of the data), and n is the number of samples in the training data.

5.3.3 One-Class Support Vector Machine

A one-class support vector machine (OC-SVM) is an unsupervised ML algorithm trained only with normal data instances. The model learns the boundaries of the normal data points in the training set, and after training, it can classify any points that lie outside the learned boundary as outliers. Since OC-SVM assumes the data to be i.i.d., it ignores the temporal structure of the time series data. Hence, additional feature engineering is usually required

to take the temporal aspect of the data into account, which may require extensive domain knowledge. However, for this thesis, we do not perform any additional feature engineering to the data for this anomaly detection method, as we want to compare whether ignoring the temporal dependencies in the data affects the anomaly detection performance in our datasets.

5.3.4 Lightweight On-Line Detector of Anomalies

The lightweight on-line detector of anomalies (LODA) is an ensemble-based unsupervised anomaly detection method proposed by Pevnỳ [56]. LODA uses one-dimensional histograms that are constructed with sparse random projection vectors of the input data. Using these random spare projections, LODA can use simple one-dimensional histograms as weak estimators, resulting in an anomaly detection model that can process large datasets relatively fast. Additionally, the sparse projections allow the model to evaluate observations with missing values and use such observations during training.

LODA uses the averaged negative log probability computed from each histogram on the projected input data as the anomaly score. Additionally, LODA supports anomaly root cause interpretation, where it uses the sparse projections to compute each feature's contribution to the detected anomaly using a one-tailed two-sample t-test. This test statistic measures the difference in the anomaly score of an input sample between using only those sparse projections where a specific feature is used and those projections where that feature is not used. When the measured difference is high, the particular feature has a high impact on the detected anomaly.

LODA also supports an online learning setup by updating the histograms on each newly arriving sample. When using this learning mode, no offline training is required, and the model can adapt to possible concept drifts in the data. Pevnỳ suggests two alternative ways to update the histograms. In the first approach, the existing histograms are updated using each new sample, where the histograms always contain the T latest observations where T denotes the size of the chosen window. When a sample is removed from the window, its contribution is also removed from the histograms. The second approach uses a two-window approach, where the first window is used to assign anomaly scores for the newly arriving samples, and the samples are added to the histograms of the second window. Every time the second window becomes full, it replaces the first window, and the second window is reset.

Like the OC-SVM, LODA also considers the data as i.i.d., ignoring the temporal structure of the data. However, when using the online learning setup, where the model uses a window of the latest observations to build the

histograms, the model can adapt to the concept drifts in the data stream and may also be able to capture some temporal anomalies in the data. Therefore, in the experiments, we test LODA with both offline training and online learning setups.

5.3.5 LSTM Encoder-Decoder

As a baseline DL-based method, we use the LSTM encoder-decoder anomaly detection model proposed by Malhotra et al., called EncDec-AD [48]. The model uses an autoencoder architecture, where both encoder and decoder contain a single LSTM layer. Figure 5.5 displays the architecture of the model. In our work, we supply the model as an input a sliding window of length T at each new time step, similar to our proposed model. Hence, the input of the model is a sequence $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$ of length T. The encoder LSTM learns a compressed fixed-length presentation of the input sequence, expressed in the final hidden state of the LSTM, $\mathbf{e}_T$. This encoded presentation is provided as the initial state of the decoder LSTM layer. The decoder then reconstructs the sequence in reverse order using the current hidden state and the value it predicted at the previous time step. At each time step, the output of the decoder LSTM is given to a linear layer, which outputs the reconstruction of the sequence at that time step. This predicted value is then supplied to the LSTM at the next time step as an input, along with the new hidden state of the cell. The same operation is performed for each time step. As a final step, the reconstructed sequence is reversed since the decoder outputs the sequence in reverse order.

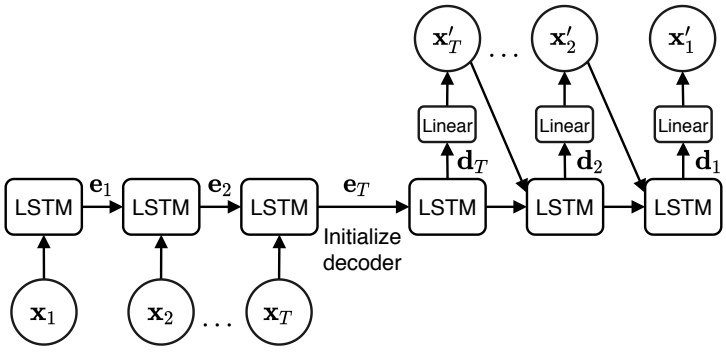


Figure 5.5: LSTM encoder-decoder architecture.

The training objective of the model is the MSE between the input sequence and the reconstruction: $\sum_{X \in s_N} \sum_{i=1}^{T} \|\mathbf{x}_i - \mathbf{x}'_i\|^2$ where T is length of the input sequence, and s_N is the set of normal training sequences.

To calculate anomaly scores, the model requires an offline phase and a validation set. The model uses the validation set to calculate each point's reconstruction error, $\mathbf{e}_i = |\mathbf{x}_i - \mathbf{x}'_i|$. The error vectors calculated for each point in the validation set are then used to estimate the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ of a multivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ using MLE. After the training, the model can be used online as follows. To compute the anomaly score a_i for a new observation $\mathbf{x}_i$, the model uses the equation: $a_i = (\mathbf{e}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{e}_i - \boldsymbol{\mu})$. Since we use a sliding window approach during anomaly prediction, we compute the anomaly score for the last observation in the window, where the rest of the observations in the sequence provide the temporal context for the model, similar to our proposed method.

5.4 Data Preprocessing

The injection molding datasets contain a small number of missing values in a few measured KPIs in some injection molding cycles. We replace these missing entries using linear interpolation. Then, we normalize the data using min-max scaling ($x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$) as follows. First, we obtain the minimum and maximum value of each feature using the training data of normal operation where known anomalies are removed. Using the obtained min-max values, we normalize the complete training set and the testing set so that each feature value within the normal range is between 0 and 1.

Since the proposed method and the baseline methods (except LODA that uses online learning) require data for training, we split the injection molding datasets into a training set and testing set. The first 70% of the data is used for training, and the last 30% is used for evaluation purposes. The resulting sizes of these datasets are displayed in Table 5.1.

Dataset	Number of features	Training set (70%)		Testing set (30%)	
		Total samples	Anomaly ratio (%)	Total samples	Anomaly ratio (%)
M1	21	18 277	5.56	7833	19.26
M2	30	18 527	8.68	7941	22.47

Table 5.1: Injection molding datasets split into training and testing sets.

5.5 Evaluation Metrics

While the classical anomaly detection field primarily focuses on point-based anomalies, anomalies in time series data usually occur over a period of time. Therefore, the evaluation of anomaly detection performance on time series data must be carefully conducted to avoid mispresenting the performance of the evaluated methods, which unfortunately happens relatively often in many published studies [70, 75]. Inspired by Hundman et al. [32], we use the following rules to evaluate the anomaly detection performance of each selected method on our time series datasets:

1. A **true positive** (TP) is recorded if an anomaly is detected within any portion of a true labeled anomalous sequence. Only one true positive is recorded even if multiple correctly recorded anomalies occur within the same labeled sequence.

2. A **false positive** (FP) is recorded for each detected anomaly that is detected outside any portion of any true labeled anomalous sequence.

3. A **false negative** (FN) is recorded if no anomalies are detected within any portion of a true labeled anomalous sequence. Only one false negative is recorded for each undetected labeled sequence.

Additionally, we ignore the first T points, where T refers to the size of the largest window among the evaluated window-based anomaly detection methods, in the performance evaluation to make the comparison between the evaluated algorithms fair since window-based anomaly detection algorithms require the initial observations for context and provide no predictions for those points. Such a procedure is justified since the algorithms evaluated in this thesis are meant to be used for real-time anomaly detection in streaming data, and therefore we can assume there are always previous observations available for context for the window-based algorithms.

Due to the large imbalance of anomalous and normal data instances, classical performance metrics, such as accuracy, cannot be used to evaluate the anomaly detection performance. Hence, to evaluate the performance of the implemented models, we use the $\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$ metric, where $\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}$, and $\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$, which is commonly used in the field of anomaly detection [40, 67]. The F1 score is the harmonic mean between precision and recall, which weighs precision and recall evenly. However, it is important to note that the F1 score is not always the best metric to evaluate the performance of an anomaly detection algorithm in real-world applications since the importance between precision and recall is often application-specific.

For instance, having a high recall when detecting cyberattacks is more important than a high precision because missing a real cyberattack needs to be avoided. Nevertheless, we use the F1 metric in this thesis, as we assume that precision and recall are equally important. Finally, note that additional factors could also be evaluated, such as the anomaly detection delay of each method. Nevertheless, we use the F1 score metric for this thesis since this metric alone allows us to measure the evaluated methods' anomaly detection performance.

Calculating the number of TPs, FPs, and FNs requires choosing a proper threshold for the anomaly scores outputted by each anomaly detection method. However, choosing the anomaly threshold is usually application-specific, making this selection subjective. Therefore, during the evaluation, we use the best possible F1 score for each method using a grid search to find the anomaly threshold for each algorithm that yields the best F1 score on the testing set.

To assess anomaly root cause interpretation performance of the selected methods, we use the hitrate metric suggested by Su et al. [67]. Note that we use a slightly different formulation for the metric for clearer notation, but the formulas still work equally. Also, note that in this context, root causes refer to the individual features of a multivariate anomalous observation that contribute to the anomaly. We define the anomaly root cause recall (RCR) as

$$\mathrm{RCR}(p, A) = \frac{N_{\mathrm{pred}}(p, A)}{N_{\mathrm{true}}(A)},$$

where $N_{\mathrm{true}}(A)$ denotes the number of true root causes of the labeled anomaly A, and $N_{\mathrm{pred}}(p, A)$ denotes the number of overlapping root causes (feature dimensions) between the true root causes of the anomaly A and the top $\frac{p}{100} \times N_{\mathrm{true}}(A)$ predicted root causes of the anomaly, ranked in descending order by the predicted feature importance. In our experiments, we use $p = 100\%$. During the evaluation, we calculate the average RCR score of each method by calculating the RCR score for each TP predicted by the method, i.e., each anomaly that the algorithm has correctly detected. When the detected anomaly is a continuous segment, we use the feature scores from that point of the segment where the anomaly detection algorithm has reported the highest anomaly score.

Chapter 6

Implementation

This chapter describes the software used to implement the proposed method and the baselines, the hardware we used in the experiments, and each model's hyperparameter configuration.

6.1 Software and Hardware

All the algorithms implemented in this thesis use Python (version 3.7.6) programming language. For the baseline statistical VAR-T^2 model, we use the implementation from Statsmodels [64] library for the VAR algorithm. The OC-SVM uses the implementation from Scikit-learn (Sklearn) library [55]. For LODA, we use the code from anlearn [1] library. We extended this LODA implementation to also support online learning by implementing the algorithm described in the original paper [56]. The deep learning models use Tensorflow [1] (version 2.3), where we leverage the built-in high-level Keras utilities whenever possible to simplify the model implementation and related procedures, such the model training. Additionally, we use Tensorflow Probability (version 0.11.1) for the deep learning models' probabilistic layers. The library includes probability distributions (many with built-in support for reparameterization trick), bijectors (for the normalizing flows), and probabilistic layers to simplify model architecture implementation. For the automatic anomaly threshold selection of DSVAE-AD, we use the POT implementation available on GitHub [2].

We trained the deep learning models on a single machine with a GPU and

[1] `anlearn` codebase: `https://github.com/gaussalgo/anlearn`. Using the version of the commit 35ee0e6 on 7.9.2020 (latest available as of 7.12.2020).

[2] `SPOT` codebase: `https://github.com/Amossys-team/SPOT`. Using the version of the commit a7a499c on 16.8.2017 (latest available as of 7.12.2020).

used the fast CuDNN implementation of LSTM and GRU layers in Tensorflow. The other baseline methods utilize CPU only in their implementations. Table 6.1 describes the hardware of the machine used in the experiments of this thesis.

Component	Description
CPU	Intel(R) Core(TM) i9-9880H CPU @ 2.30Ghz
RAM	32 GB
GPU	Nvidia Quadro RTX 4000, 8GB GDDR6 memory

Table 6.1: Hardware specification.

6.2 Hyperparameters

This section describes how the hyperparameters of each model were selected and adjusted.

6.2.1 Proposed Method

The proposed model, DSVAE-AD, is trained in a semi-supervised setting, where the model is trained with training data consisting of normal instances. Therefore, since the training sets of the injection molding datasets contain anomalies, we make the model ignore them in its objective function during training by supplying the model the labels of the known anomalous injection molding cycles in the data, as explained in the definition of the objective function in Section 5.2.2.

We train the model using mini-batch SGD with Adam [34] optimizer with the learning rate set to 0.001. We set the mini-batch size as 128 for the injection molding datasets and 256 for the artificial data during training. All model weights are initialized using Glorot uniform initializer, which is the default weight initializer of Tensorflow. The sliding window size (length of input data sequences) is set to 100. Note that we did not notice the KL term collapsing problem during training, and therefore we did not apply procedures, such as KL term annealing, in the loss calculation. We use the last 30% of the training data as a validation set to prevent the model from overfitting. We run the training for 200 epochs and use an early stopping strategy to stop the training when the loss on the validation set does not decrease for ten consecutive epochs. During training, we use a single MC sample when sampling each latent variable. When the model is used to

compute the anomaly scores for the observations in the test set, we use $L = 64$ MC samples to make the predicted reconstruction probabilities less noisy. Each GRU layer in the model has 128 hidden units. We clip the gradients by norm with 10.0 as the limit to prevent exploding gradient problems.

Table 6.2 displays the hyperparameter settings that are specific to each dataset. Two primary hyperparameters that need to be adjusted for each dataset are the dimensionality of the dynamic encoder's latent variables and the dimensionality of the static encoder's latent variable. Having too small dimensionality for the latent variables could make the model underfit the normal data, hence having a bad reconstruction of normal data patterns. On the other hand, having too high latent variable dimensionality would make the model reconstruct both normal and anomalous data well, making the model perform poorly for anomaly detection. Automatically choosing these parameters is not trivial [67], and therefore the optimal values for these parameters must be searched empirically using the available training data and evaluating the normal data reconstruction quality.

Dataset	Mini-batch size	Latent dim. dynamic	Latent dim. static	Total trainable params	Training time per epoch (s)
M1	128	3	30	192 492	7
M2	128	5	30	203 010	7
Artificial	256	3	80	223 566	5

Table 6.2: Hyperparameter settings of DSVAE-AD specific to each evaluated dataset.

As imposed by the research objectives in Section 1.1, the proposed method should be able to detect anomalies in real-time, requiring the model to perform predictions fast enough for each new observation in a data stream. In the datasets evaluated in our experiments, DSVAE-AD takes approximately 130ms to calculate the anomaly score for a single observation (single sliding window sequence), making the model applicable for a near real-time setting. We expect that further improvements to the inference speed are possible because the machine we used in the experiments had an old version of the GPU driver installed to allow us to utilize the fast CuDNN implementations of LSTM and GRU layers due to a driver bug in the newer driver versions specific to our hardware environment.

6.2.2 Baseline Methods

MEWMA is fitted using the normal data of each injection molding training set by discarding the labeled anomalies, i.e., the algorithm calculates the mean and covariance using the normal historical data. In the experiments, we test different weighting factors between the range $[0, 1]$ and choose the one giving the best performance using the previously defined evaluation criteria.

VAR-T^2 is fitted with training data where the labeled anomalous samples are ignored during training by replacing them with null values. The VAR model's optimal number of lags is determined using the Akaike information criterion (AIC), which is a commonly used model selection method in statistics [71].

Similar to the two previous baseline methods, OC-SVM is fitted using only normal samples in the training data, i.e., all labeled anomalies are removed. We use the default parameters of the OC-SVM implementation in Sklearn, i.e., a radial basis function (RBF) kernel, nu = 0.5, and gamma = $\frac{1}{D \times Var(X)}$.

For LODA, we perform our experiments using both offline training and online learning setups. For each experiment, we test different histogram counts in the range $[100, 500]$, the number of bins in the range $[10, 100]$, and report the configuration that yields the best F1 score on each evaluated dataset. In the online learning mode, we choose the window size in range $[50, 500]$ and similarly report the results of the best performing model configuration.

For EncDec-AD, we use the same window size as for the DSVAE-AD, 100. Similar to DSVAE-AD, we train this model using Adam optimizer with the learning rate set to 0.001. We set the number of hidden units to be the same in both encoder and decoder networks. We test both 128 and 256 hidden units and choose the one yielding a better F1 score in the experiments for each evaluated dataset. We test the model using both LSTM and GRU layers in the encoder and decoder networks and choose the configuration giving the better F1 score. Note that we also tested using bidirectional RNN layers, but they did not improve the model performance. We train EncDec-AD using only normal sequences of the training data. Hence, when using the injection molding data, which contains anomalies in the training data, any sliding window containing anomalies in the training set is ignored. We also tested replacing the anomalies in the training data with linear interpolation, but this yielded worse results on both injection molding datasets.

Chapter 7

Evaluation

This chapter presents the results of the proposed method and the baseline methods applied to the injection molding data and the artificial data using the evaluation criteria presented in Section 5.5.

7.1 Injection Molding Machine Data

Table 7.1 presents the precision, recall, and best F1 score of the proposed method, DSVAE-AD, and each baseline on the M1 and M2 datasets. The F1 score displayed for each method is selected using the anomaly threshold yielding the highest F1 score for each method on the testing set data, following the evaluation criteria described in Section 5.5. We can see that most of the evaluated methods achieved a relatively good F1 score in both injection molding datasets. This outcome occurs because most of the labeled anomalies in the collected datasets are (global) point anomalies that are relatively trivial to detect since they can be determined anomalous without contextual information. Nevertheless, there are still differences in the F1 scores achieved by the evaluated methods on these datasets, which we discuss next.

In the M1 dataset, we can see that the proposed method, DSVAE-AD, achieved a higher F1 score than the baselines, indicating that it can detect some less apparent anomalies that the baseline methods cannot detect as accurately. Among the baseline methods, the LODA model trained offline, and OC-SVM yielded the best performance, further confirming that most of the anomalies in the collected data are detectable without considering temporal aspects of the data. The presence of a high number of point anomalies is also inferable from the high F1 score achieved by MEWMA, which is the simplest baseline algorithm we used that also ignores the temporal structure in the data. The baseline deep learning method, EncDec-AD, does not perform as

well as DSVAE-AD and the previously discussed baselines. A possible reason for its lower performance is that the model is not robust enough to the noise present in the data due to it using an AE architecture instead of VAE that generally handles stochasticity better by using probability distributions to model the data. Additionally, the number of training samples given to EncDec-AD was relatively small since we ignored any sliding window that contained anomalies, likely making the model underfit to the normal data. The table also shows that the LODA using online learning had worse performance than LODA trained offline, indicating that the window-based online model updating does not help to detect additional anomalies in this data. The worst performing model on the M1 dataset by the F1 metric is VAR-T^2. Its low F1 score occurs due to the model outputting high anomaly scores for some normal observations even after the labeled anomalous sequences have ended, producing many false positives. Such behavior would be disadvantageous in a real-world setting since the model would declare normal observations as anomalous after the actual anomaly has already ended, lowering the human operators' trust towards the method's predictions. The likely reason why VAR-T^2 outputs such anomaly scores is that the data is more complicated and less predictable than VAR can model with its statistical assumptions of the data.

	M1 dataset			M2 dataset		
Method	Precision	Recall	$F1_{best}$	Precision	Recall	$F1_{best}$
MEWMA	0.9818	0.8852	0.9310	0.9516	0.8939	0.9219
VAR-T^2	0.7925	0.6885	0.7368	0.9434	0.7576	0.8403
OC-SVM	0.9821	0.9016	0.9402	1.0000	0.8182	0.9000
LODA (offline)	1.0000	0.9344	0.9661	0.9677	0.9091	**0.9375**
LODA (online)	0.9615	0.8197	0.8850	0.9516	0.8939	0.9219
EncDec-AD	1.0000	0.8033	0.8909	0.9483	0.8333	0.8871
DSVAE-AD	1.0000	0.9672	**0.9833**	1.0000	0.8333	0.9091

Table 7.1: Performance metrics of each method evaluated on the injection molding datasets.

When inspecting each evaluated method's best F1 scores on the M2 dataset, we can see that many methods achieve a lower score than in the M1 dataset. The main reason for the lower F1 scores on this dataset is that a concept drift occurred in the M2 testing set data within some KPIs, making it more difficult for the models to detect contextual anomalies using a single anomaly threshold applied in the experiments. In other words, be-

cause we calculate the F1 score based on a single anomaly threshold yielding the best F1 score on the testing data, detecting contextual anomalies in the data points before this particular concept drift becomes difficult because the anomaly scores of the anomalous observations in this part of the dataset may be lower than the anomaly scores of the normal data points after the concept drift. Figure 7.1 visualizes the concept drift, which occurs approximately after the 3200th injection molding cycle in the testing set, in metrics such as the dosing time in the injection unit 2, and mold protection time. The figure displays how this concept drift causes the anomaly scores outputted by DSVAE-AD and LODA (offline) to remain at a higher level than before the concept drift for all later data points, both normal and anomalous. Although the figure displays the anomaly scores only for these two methods, the concept drift also affects most of the other baselines' performance. While the concept drift affects most of the methods, we can see that the LODA model trained online achieves a relatively good F1 score on the M2 data, showing the benefits of online learning in the presence of concept drifts and non-stationary data.

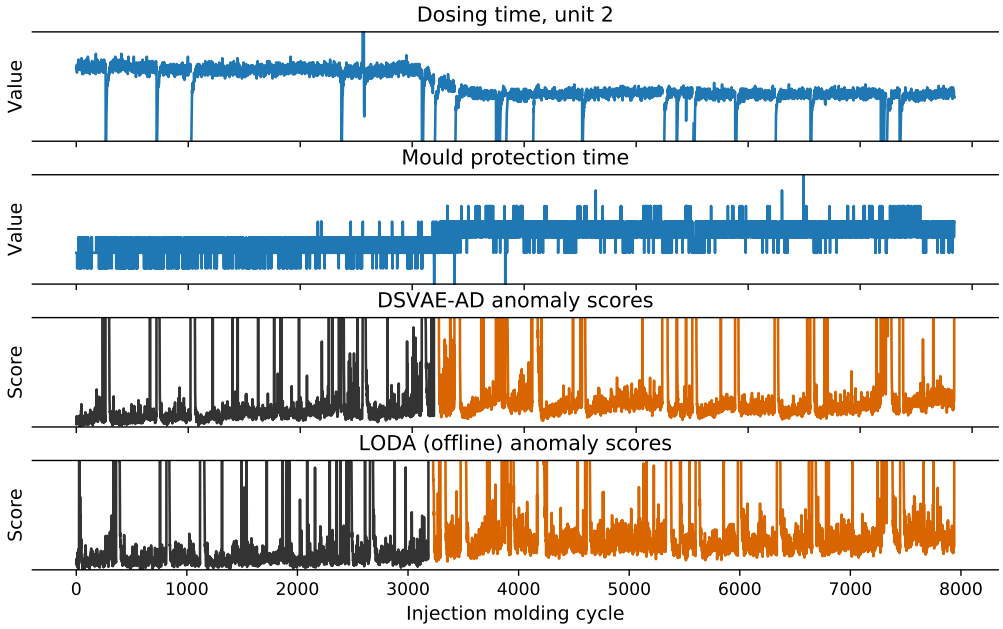


Figure 7.1: The effect of concept drift that occurs in certain metrics of the M2 testing set data on the anomaly scores outputted by DSVAE-AD and LODA trained offline. The anomaly scores computed after the concept drift are highlighted in dark orange color.

Table 7.2 displays the anomaly root cause interpretation results of the evaluated methods on the injection molding datasets. Since most of the methods cannot provide root cause interpretation, the table only shows the results of the proposed method, DSVAE-AD, and LODA, which can interpret the detected anomalies' root causes. The table shows that DSVAE-AD achieves a higher RCR score than LODA (both offline and online learning setups) on both datasets. Moreover, the relatively high RCR scores achieved by DSVAE-AD suggest that the proposed method can sufficiently interpret the individual KPIs contributing to the detected anomalies in the injection molding datasets.

	Root cause recall (RCR)	
Method	M1 dataset	M2 dataset
LODA (offline)	0.7676	0.8974
LODA (online)	0.8062	0.8924
DSVAE-AD	**0.8498**	**0.9088**

Table 7.2: Root cause interpretation results on the injection molding datasets.

7.2 Artificial Data

Table 7.3 displays the results of the evaluated methods applied to the artificial data. MEWMA has the lowest F1 score on this dataset, which confirms that the method should not be used for data with strong temporal dependencies because it assumes the data points to be independent. Like MEWMA, the baseline classical machine learning methods (OC-SVM and LODA), which consider the data points as i.i.d., have poor anomaly detection performance because of the autocorrelation between the data points. We can see that LODA trained offline has better performance than the LODA using an online learning setup, indicating that LODA cannot capture the temporal anomalies well in the artificial data even when the model uses an online learning setup.

Based on the F1 scores, the best-performing methods for anomaly detection on the artificial data are DSVAE-AD, EncDec-AD, and VAR-T^2, all capable of modeling the temporal structure of the data. The relatively good performance of DSVAE-AD on the artificial data and the injection molding data demonstrates that the proposed method applies to different multivariate time series data problems. VAR-T^2 performs well with the artificial data

Method	Precision	Recall	$F1_{best}$	Root cause recall (RCR)
MEWMA	0.5333	0.2051	0.2963	-
VAR-T^2	0.9062	0.7436	0.8169	-
OC-SVM	0.7500	0.3077	0.4364	-
LODA (offline)	0.7407	0.5128	0.6061	0.5417
LODA (online)	0.7368	0.3590	0.4828	0.6548
EncDec-AD	1.0000	0.6667	0.8000	-
DSVAE-AD	0.8611	0.7949	**0.8267**	**0.9516**

Table 7.3: Performance of the evaluated methods on the artificial data.

because the statistical properties of the artificial data match assumptions made by the VAR model, i.e., the noise in the data is Gaussian, and the time series data is stationary. However, we expect the performance of VAR-T^2 to decrease with more complex data, where the statistical assumptions made by the model do not hold, as occurred in the experiments conducted with the injection molding datasets, where the VAR-T^2 had the lowest F1 score among the evaluated methods.

On the artificial dataset, we also tested selecting the anomaly threshold of DSVAE-AD with the POT method used in OmniAnomaly [67] to demonstrate how to determine a suitable anomaly threshold for the data practically. As described in Section 5.1.4, the hyperparameters of POT have to be chosen empirically by examining the anomaly scores computed on the training set and possibly using a validation set with some known anomalies to adjust the hyperparameters so that the final threshold can isolate the anomalies in the validation set. Using the POT method, DSVAE-AD achieves an F1 score of 0.8219 by using the hyperparameters $q = 5 \times 10^{-5}$ and $t = 98\%$ and the anomaly scores computed on the training set, which demonstrates that this technique could be potentially be used to determine a suitable anomaly threshold for the model.

Similar to the anomaly root cause interpretation results on the injection molding datasets, DSVAE-AD outperforms LODA (both offline and online learning setups) on the anomaly RCR performance metric (0.9516) on the artificial data. Since each artificially generated anomaly can appear in a maximum of three out of the 20 dimensions (see Section 4.2 for properties of the artificial data), blindly guessing the root causes of an anomaly would yield an RCR score of 0.15 on average, indicating that the DSVAE-AD can perform anomaly root cause interpretation on the detected anomalies with high recall.

Figure 7.2 displays an example reconstruction and the anomaly scores

produced by DSVAE-AD for a section of the generated artificial data, demonstrating how the model performs anomaly detection in practice. We can see from the figure that DSVAE-AD fails to reconstruct the anomalous patterns in the data since those observations differ from the normal sine-wave patterns that the model learned from the samples in the training data. As discussed in Section 5.1.3, the underlying reason why the model fails to reconstruct these anomalies is that when the anomalous input sequence is expressed as a compressed latent presentation, the presentation cannot capture the anomalous patterns of the data, but its normal aspects. Therefore, the model can only reconstruct the expected normal pattern of the anomalous input sequence. Consequently, the model produces a lower (reconstruction) likelihood for the anomalous observations that the latent representations cannot express well, and thus a higher anomaly score.

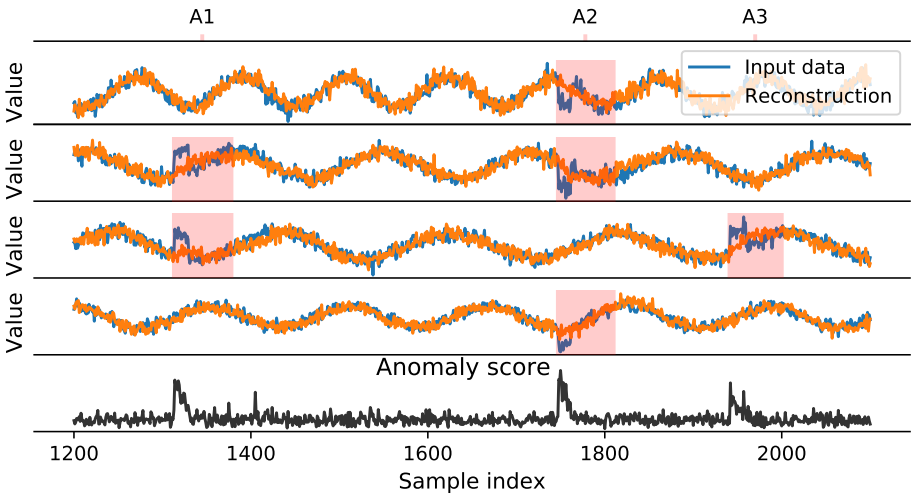


Figure 7.2: An example of the reconstruction and the anomaly scores from a section of the artificial data predicted by DSVAE-AD. Labeled (true) anomalies are highlighted in red.

Chapter 8

Conclusions

This chapter presents the final remarks of this thesis. In Section 8.1, we discuss this work's results, the limitations of the work, and suggestions for future work. Finally, we conclude this thesis with Section 8.2, which summarizes the contents described in this thesis.

8.1 Discussion

The results obtained in this thesis show that the proposed DL-based method can be used to detect anomalies in the multivariate time series data generated by the IMMs. Similarly, the results show that the proposed method can also be applied to other multivariate time series data problems, as we demonstrated with the generated artificial data. Additionally, the results demonstrated that the proposed method could interpret the detected anomalies' root causes with a relatively good performance. The experiments also demonstrated that the model could be applied in a near real-time setting for streaming data, as the proposed method could compute the anomaly score in approximately 130 ms for a new observation with the used hardware.

Our proposed method, DSVAE-AD, extends the OmniAnomaly model [67], where we introduced several changes to the original model to achieve a better anomaly detection performance in our experiments. While the performance of DSVAE-AD showed promising results in the performed experiments, there remain multiple ways to improve the model further. For example, while we could not find the benefits of using normalizing flows or complex prior distributions, including these techniques in the model should be further tested with new, larger datasets. Furthermore, introducing an attention mechanism could be explored to improve the model's latent presentations. With the use of attention, the model could better localize the most relevant parts of the

input sequence, leading to better reconstructions of normal data patterns that could improve anomaly detection performance.

A considerable limitation of the current work is the relatively small size of the real-world injection molding datasets used to evaluate the algorithms, making the experimental results obtained on these datasets less reliable and less generalizable. Besides, the DL-based algorithms' full capacity is only obtainable with large enough datasets since more data would make these algorithms generalize better to the data. Moreover, as the data was collected only for a short duration (nine days), it cannot capture the whole range of IMMs' normal behavior. The insufficient dataset size was also the main factor contributing to the concept drift occurring in the M2 testing set data, which degraded the anomaly detection performance of most evaluated models. In other words, since the M2 training set contained no similar examples as the observations in the testing set after the concept drift, the models trained in offline mode considered the data points after the concept drift less normal.

Another limitation occurring from the small size of the injection molding datasets is that most of the labeled anomalies in these datasets correspond to (global) point anomalies, which are relatively trivial to detect with many algorithms, as the results from our experiments demonstrated. Furthermore, most anomalies in these datasets had occurred due to operation mode switches in the IMMs, which are known in advance, as they are performed manually. Thus, detecting these anomalies does not bring much value to the machine operators. However, by collecting more comprehensive datasets, some rarer anomalies could appear in the data that could be non-trivial and more challenging to detect. For instance, the data collected over a long time could include anomalies that are critical to detect, such as an injection mold cavity getting blocked or some machine part starting to fail, which would affect the quality of the output products.

The proposed DL-based method is not limited to only anomaly detection on injection molding machines, but it is applicable for any multivariate time series data problem. Therefore, we demonstrated the applicability of the proposed approach to artificially generated data. However, the generated artificial time series data is relatively simple, and therefore the results on this dataset cannot demonstrate the proposed approach's full capability. Therefore, future work should try applying the proposed method to other, more complex, multivariate time series data problems also from different application domains.

The underlying assumption of the proposed method is that the training data adequately describes the normal variations of the data, and any future observation that deviates from the learned normal is declared anomalous. However, the training data could become obsolete due to actions, such as the

machine going through maintenance or when the operators perform other configuration changes. Thus, when concept drift occurs in the machine's normal data distribution, the model needs to be re-trained. To avoid completely re-training the model, future work could also explore adaptive training techniques for the proposed method, where the model is incrementally trained with new data to make it adapt to concept drifts. The incremental training could be performed, for example, by storing the new observations in a buffer, and when the buffer gets full, the model receives the new batch of data for training. Such an approach could be feasible to be performed online for the injection molding data since the training time of a single epoch of the model was considerably less than the duration of a single injection molding cycle in our experiments.

Concept drifts occurring in the data also make it challenging to choose a suitable anomaly threshold if the underlying data distribution keeps changing. Consequently, the POT approach proposed by Su et al. [67] that yields a fixed anomaly threshold could be insufficient in practical applications. Therefore, future work could test different dynamic anomaly threshold techniques for our model, such as the method proposed by Park et al. [53].

This thesis focused on anomaly detection in IMMs using the aggregated process data collected from each injection molding cycle. While the aggregated metrics from the cycles can be used to detect certain types of anomalies in the injection molding process, they might be incapable of expressing certain anomalies occurring inside a cycle. For instance, an anomaly could only be identifiable by looking at a continuous injection pressure curve of a cycle but not be visible in an aggregated metric, such as the peak injection pressure. Therefore, future work should try applying the proposed method on the continuous metrics of the IMM data, such as the pressure curves of each cycle, to assess whether such metrics can offer a better ability to detect anomalies occurring in the machines. Besides, the temporal aspect of the data would be essential to consider in the continuous metrics, which the proposed model should be able to model well.

Anomaly detection is the first step to an automated quality monitoring system. After finding a suitable algorithm that can accurately detect anomalies and identify the individual metrics contributing most to the detected multivariate anomaly, the next step could be embedding domain knowledge to perform domain-specific fault diagnoses. For instance, after detecting an anomaly and determining the KPIs contributing to it, the system could suggest the most likely problem in the machine that has caused the anomaly. The suggestion could rely on a pre-defined set of rules that map the collection of the identified root cause KPIs of the anomaly to domain-specific root causes. However, such an approach would require more domain knowledge of

IMM specific problems, and some issues might be highly machine-dependent. Nonetheless, this approach could still be valuable to identify the most commonly occurring problems in IMMs.

As a final remark, it is crucial to understand the data's characteristics before choosing a suitable anomaly detection method. Besides, one should always start by applying the simplest algorithms to their anomaly detection problem before considering more complex approaches. Simultaneously, one should weigh each method's strengths and weaknesses. For instance, DL-based methods usually require significantly more training data than many simpler algorithms, preventing using such approaches before acquiring more data.

8.2 Summary

This thesis proposed a DL-based anomaly detection method called DSVAE-AD to detect anomalies from multivariate time series data and interpret their root causes. This thesis started by providing the reader with a theoretical background of the concepts and algorithms related to the proposed approach. Afterward, we presented a survey of various anomaly detection methods for multivariate time series data and reviewed studies related to anomaly detection for IMMs. We then presented the real-world IMM datasets and the artificial data that we use in this thesis's experiments. Following that, we introduced the proposed method, DSVAE-AD, which extends an existing approach from the literature based on the shortcomings we detected in the original model during our experiments. To evaluate the proposed method's performance and to assess whether the DL-based approach has advantages over traditional methods, we compared DSVAE-AD's anomaly detection performance to five baseline methods selected from the literature. The results of this thesis show that DSVAE-AD could successfully detect most of the anomalies in the collected multivariate IMM datasets and the generated artificial data and had the best anomaly detection performance in most of the experiments. Additionally, the method had the best performance among the evaluated methods in interpreting the detected anomalies' root causes. The experiments of this study demonstrate that DL-based anomaly detection methods can be beneficial when the data is too complicated for traditional approaches, and enough data for training is available. However, the size of the collected injection molding datasets used in this thesis is relatively small, limiting the generalizability of the obtained results. Therefore, future work should collect a more extensive set of IMM data from the machines to evaluate whether the results would be similar to the outcomes of our work. Future

studies should also try applying the proposed method to other multivariate time series problems and compare our approach to other DL-based anomaly detection models.

Bibliography

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] ABB. ABB AP10. `https://new.abb.com/products/2TKA140002G1/ap10-junction-box-ip65`. Accessed 24.8.2020.

[3] ABB. ABB AP9. `https://new.abb.com/products/2TKA140012G1/ap9-junction-box-ip65`. Accessed 24.8.2020.

[4] Ahmad, S., Lavin, A., Purdy, S., and Agha, Z. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing 262* (2017), 134–147.

[5] An, J., and Cho, S. Variational autoencoder based anomaly detection using reconstruction probability. Tech. rep., SNU Data Mining Center, 2015. `http://dm.snu.ac.kr/static/docs/TR/SNUDM-TR-2015-03.pdf`.

[6] Beirlant, J., Goegebeur, Y., Segers, J., and Teugels, J. L. *Statistics of extremes: theory and applications*. John Wiley & Sons, 2006.

[7] Belkadi, A., and Billaudel, P. Real-time adaptive fault detection and isolation on a high-speed multi-injection molding machine. In *2019 4th Conference on Control and Fault Tolerant Systems (SysTol)* (2019), IEEE, pp. 62–67.

[8] Besedin, A., Blanchart, P., Crucianu, M., and Ferecatu, M. Evolutive deep models for online learning on data streams with no storage. In *ECML/PKDD 2017 Workshop on Large-scale Learning from Data Streams in Evolving Environments* (Skopje, Macedonia, Sept. 2017).

[9] Blázquez-García, A., Conde, A., Mori, U., and Lozano, J. A. A review on outlier/anomaly detection in time series data. *arXiv preprint arXiv:2002.04236* (2020).

[10] Braei, M., and Wagner, S. Anomaly detection in univariate time-series: A survey on the state-of-the-art. *arXiv preprint arXiv:2004.00433* (2020).

[11] Brockwell, P. J., and Davis, R. A. *Introduction to Time Series and Forecasting*. Springer, 2016.

[12] Chalapathy, R., and Chawla, S. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407* (2019).

[13] Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection: A survey. *ACM computing surveys (CSUR) 41*, 3 (2009), 1–58.

[14] Chauhan, N. K., and Singh, K. A review on conventional machine learning vs deep learning. In *2018 International Conference on Computing, Power and Communication Technologies (GUCON)* (2018), IEEE, pp. 347–352.

[15] Cheng, H., Tan, P.-N., Potter, C., and Klooster, S. Detection and characterization of anomalies in multivariate time series. In *Proceedings of the 2009 SIAM international conference on data mining* (2009), SIAM, pp. 413–424.

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

[17] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[18] Costa, N., Cunha, A., and Ribeiro, B. Monitoring an industrial plastic injection moulding machine using neural networks. In *Artificial Neural Nets and Genetic Algorithms* (1999), Springer, pp. 89–94.

[19] Davis, N., Raina, G., and Jagannathan, K. LSTM-based anomaly detection: Detection rules from extreme value theory. In *EPIA Conference on Artificial Intelligence* (2019), Springer, pp. 572–583.

[20] Davis, N., Raina, G., and Jagannathan, K. A framework for end-to-end deep learning-based anomaly detection in transportation networks. *Transportation Research Interdisciplinary Perspectives 5* (2020), 100112.

[21] Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. A survey on concept drift adaptation. *ACM computing surveys (CSUR) 46*, 4 (2014), 1–37.

[22] Gerbing, D. Time series components. *School of Business Administration, Portland State University* (2016).

[23] Gers, F. A., Schmidhuber, J., and Cummins, F. Learning to forget: continual prediction with LSTM. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)* (1999), vol. 2, IET, pp. 850–855 vol.2.

[24] Goldstein, M., and Uchida, S. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLOS ONE 11*, 4 (04 2016), 1–31.

[25] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[26] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems 28*, 10 (2016), 2222–2232.

[27] Guo, Y., Liao, W., Wang, Q., Yu, L., Ji, T., and Li, P. Multidimensional time series anomaly detection: A GRU-based gaussian mixture variational autoencoder approach. In *Asian Conference on Machine Learning* (2018), pp. 97–112.

[28] Gupta, M., Gao, J., Aggarwal, C. C., and Han, J. Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and data Engineering 26*, 9 (2013), 2250–2267.

[29] Hawkins, D. M. *Identification of outliers*. Chapman and Hall, London and New York, 1980.

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

[31] Hodge, V., and Austin, J. A survey of outlier detection methodologies. *Artificial intelligence review 22*, 2 (2004), 85–126.

[32] Hundman, K., Constantinou, V., Laporte, C., Colwell, I., and Soderstrom, T. Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2018), pp. 387–395.

[33] Khoshnevisan, F., and Fan, Z. RSM-GAN: A convolutional recurrent gan for anomaly detection in contaminated seasonal multivariate time series. *arXiv preprint arXiv:1911.07104* (2019).

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

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

[36] Kingma, D. P., and Welling, M. An introduction to variational autoencoders. *CoRR abs/1906.02691* (2019).

[37] Kitagawa, G., and Gersch, W. Linear gaussian state space modeling. In *Smoothness priors analysis of time series*. Springer, 1996, pp. 55–65.

[38] Kobyzev, I., Prince, S., and Brubaker, M. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), 1–1.

[39] Kouiroukidis, N., and Evangelidis, G. The effects of dimensionality curse in high dimensional knn search. In *2011 15th Panhellenic Conference on Informatics* (2011), IEEE, pp. 41–45.

[40] Li, D., Chen, D., Jin, B., Shi, L., Goh, J., and Ng, S.-K. Madgan: Multivariate anomaly detection for time series data with generative adversarial networks. In *International Conference on Artificial Neural Networks* (2019), Springer, pp. 703–716.

[41] Li, Y., and Mandt, S. Disentangled sequential autoencoder. *arXiv preprint arXiv:1803.02991* (2018).

[42] Liu, F. T., Ting, K. M., and Zhou, Z.-H. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining* (2008), IEEE, pp. 413–422.

[43] Lowry, C. A., Woodall, W. H., Champ, C. W., and Rigdon, S. E. A multivariate exponentially weighted moving average control chart. *Technometrics 34*, 1 (1992), 46–53.

[44] Luttinen, J., Raiko, T., and Ilin, A. Linear state-space model with time-varying dynamics. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2014), Springer, pp. 338–353.

[45] Ma, J., and Perkins, S. Time-series novelty detection using one-class support vector machines. In *Proceedings of the International Joint Conference on Neural Networks, 2003.* (2003), vol. 3, pp. 1741–1745 vol.3.

[46] Mahmoud, M. A., and Maravelakis, P. E. The performance of the MEWMA control chart when parameters are estimated. *Communications in Statistics—Simulation and Computation 39*, 9 (2010), 1803–1817.

[47] Maimon, O., and Rokach, L. *Data Mining and Knowledge Discovery Handbook*. Springer, 2005.

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

[49] Munir, M., Siddiqui, S. A., Dengel, A., and Ahmed, S. DeepAnT: A deep learning approach for unsupervised anomaly detection in time series. *IEEE Access 7* (2018), 1991–2005.

[50] Murphy, K. P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[51] Nagorny, P., Pillet, M., Pairel, E., Le Goff, R., Loureaux, J., Wali, M., and Kiener, P. Quality prediction in injection molding. In *2017 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)* (2017), IEEE, pp. 141–146.

[52] Pan, X., and Jarrett, J. Using vector autoregressive residuals to monitor multivariate processes in the presence of serial correlation. *International Journal of Production Economics 106*, 1 (2007), 204–216.

[53] Park, D., Hoshi, Y., and Kemp, C. C. A multimodal anomaly detector for robot-assisted feeding using an LSTM-based variational autoencoder. *IEEE Robotics and Automation Letters 3*, 3 (2018), 1544–1551.

[54] Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International conference on machine learning* (2013), pp. 1310–1318.

[55] 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. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[56] Pevnỳ, T. Loda: Lightweight on-line detector of anomalies. *Machine Learning 102*, 2 (2016), 275–304.

[57] Pokrajac, D., Lazarevic, A., and Latecki, L. J. Incremental local outlier detection for data streams. In *2007 IEEE symposium on computational intelligence and data mining* (2007), IEEE, pp. 504–515.

[58] Rébillat, M., Hmad, O., Kadri, F., and Mechbal, N. Peaks over threshold–based detector design for structural health monitoring: Application to aerospace structures. *Structural Health Monitoring 17*, 1 (2018), 91–107.

[59] Rezende, D. J., and Mohamed, S. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770* (2015).

[60] Ribeiro, B. Support vector machines for quality monitoring in a plastic injection molding process. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 35*, 3 (2005), 401–410.

[61] Salehi, M., Leckie, C., Bezdek, J. C., Vaithianathan, T., and Zhang, X. Fast memory efficient local outlier detection in data streams. *IEEE Transactions on Knowledge and Data Engineering 28*, 12 (2016), 3246–3260.

[62] Schiffers, R., Morik, K., Struchtrup, A. S., Honysz, P.-J., and Johannes, W. Anomaly detection in injection molding process data based on unsupervised learning. *Journal of Plastics Technology 14*, 5 (2018), 301–347.

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

[64] Seabold, S., and Perktold, J. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference* (2010).

[65] Siffer, A., Fouque, P.-A., Termier, A., and Largouet, C. Anomaly detection in streams with extreme value theory. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017), pp. 1067–1075.

[66] Singh, Y., and Chauhan, A. S. Neural networks in data mining. *Journal of Theoretical & Applied Information Technology 5*, 1 (2009).

[67] Su, Y., Zhao, Y., Niu, C., Liu, R., Sun, W., and Pei, D. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019), pp. 2828–2837.

[68] Tan, S. C., Ting, K. M., and Liu, T. F. Fast anomaly detection for streaming data. In *Twenty-Second International Joint Conference on Artificial Intelligence* (2011).

[69] Tatbul, N. Streaming data integration: Challenges and opportunities. In *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)* (2010), IEEE, pp. 155–158.

[70] Tatbul, N., Lee, T. J., Zdonik, S., Alam, M., and Gottschlich, J. Precision and recall for time series. In *Advances in Neural Information Processing Systems* (2018), pp. 1920–1930.

[71] Wagenmakers, E.-J., and Farrell, S. AIC model selection using Akaike weights. *Psychonomic bulletin & review 11*, 1 (2004), 192–196.

[72] Wang, X., Lin, J., Patel, N., and Braun, M. Exact variable-length anomaly detection algorithm for univariate and multivariate time series. *Data Mining and Knowledge Discovery 32*, 6 (2018), 1806–1844.

[73] Weng, L. From autoencoder to beta-VAE, 2018. `https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html`. Accessed 12.11.2020.

[74] Woll, S. L. B., Cooper, D. J., and Souder, B. V. Online pattern-based part quality monitoring of the injection molding process. *Polymer Engineering & Science 36*, 11 (1996), 1477–1488.

[75] Wu, R., and Keogh, E. J. Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress. *arXiv preprint arXiv:2009.13807* (2020).

[76] Xu, H., Chen, W., Zhao, N., Li, Z., Bu, J., Li, Z., Liu, Y., Zhao, Y., Pei, D., Feng, Y., et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference* (2018), pp. 187–196.

[77] Yu, Y., Zhu, Y., Li, S., and Wan, D. Time series outlier detection based on sliding window prediction. *Mathematical problems in Engineering 2014* (2014).

[78] Zhang, C., Song, D., Chen, Y., Feng, X., Lumezanu, C., Cheng, W., Ni, J., Zong, B., Chen, H., and Chawla, N. V. A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33, pp. 1409–1416.

[79] Zhang, Y., Meratnia, N., and Havinga, P. Adaptive and online one-class support vector machine-based outlier detection techniques for wireless sensor networks. In *2009 international conference on advanced information networking and applications workshops* (2009), IEEE, pp. 990–995.

[80] Zhu, X., and Wu, X. Class noise vs. attribute noise: A quantitative study. *Artificial intelligence review 22*, 3 (2004), 177–210.

Appendix A

Data attributes

Name	Unit
Cycle time	s
Mold protection time	s
Robot gripping time	s
Material cushion volume	cm^3
Injection time	s
Dosing time	s
Peak injection pressure	bar
Changeover pressure	bar
Screw peak injection pressure	bar
Screw injection speed	cm^3/s
Back pressure	bar
Pressure integral	bar×s
Feeding zone temperature	°C
Barrel zone 1 temperature	°C
Barrel zone 2 temperature	°C
Barrel zone 3 temperature	°C
Barrel zone 4 temperature	°C
Barrel zone 5 temperature	°C
Barrel zone 6 temperature	°C
Barrel zone 7 temperature	°C
Barrel zone 8 temperature	°C

Table A.1: Data attributes (21 in total) in the M1 dataset. The values were measured once per injection molding cycle.

Name	Unit
Cycle time	s
Mold protection time	s
Robot gripping time	s
Clamping force	kN
Unit 1	
Material cushion volume	cm^3
Injection time	s
Dosing time	s
Peak injection pressure	bar
Back pressure	bar
Changeover pressure	bar
Feeding zone temperature	°C
Barrel zone temperatures - 8 measurement points	°C
Unit 2	
Material cushion volume	cm^3
Injection time	s
Dosing time	s
Peak injection pressure	bar
Back pressure	bar
Feeding zone temperature	°C
Barrel zone temperatures - 5 measurement points	°C

Table A.2: Data attributes (30 in total) in the M2 dataset. The values were measured once per injection molding cycle.