Unsupervised representation learning for time series anomaly detection and beyond

Ferreira Correia; L.

Chapter 2

Background and Theory

To aid the reader in understanding all concepts discussed in this thesis, this chapter provides an introduction to the principles required. First, Chapter 2.1 offers a brief overview on time series theory, highlighting some key properties of time series data and how to interpret them. Then, Chapter 2.2 presents the required foundations on anomaly detection as well as a novel taxonomy. Chapters 2.3–2.5 introduce a number of time series modelling constructs which play a foundational role later on in Chapter 5, while the concepts presented in Chapters 2.6 and 2.7 become especially relevant in Chapter 6.

2.1 Time Series Theory

Time series are signals that represent a property or feature of a dynamic system as a function of time, usually sampled at a fixed rate. Unlike image or text data, which humans learn to process from a young age, time series data is less intuitive and requires corresponding domain knowledge to be interpreted. An arbitrary time series $\mathcal{S}$ can be univariate, i.e. $\mathcal{S} \in \mathbb{R}^T$, or multivariate, i.e. $\mathcal{S} \in \mathbb{R}^{T \times d}$, where T refers to the number of discrete time steps and d to the number of features in the time series. More specifically, univariate time series can only possess a temporal correlation, i.e. along the time axis, whereas multivariate time series can also contain correlation along the feature axis.

Time series can be decomposed into different components. The first one is noise, also referred to residuals, in which observations are independent of one another, identically distributed and have a mean of zero [11]. Another component is trend, which is

present when a time series has a non-zero mean. If the mean of the time series is zero, it is referred to as a stationary time series [11, 12]. The last component is seasonality, which is present when certain patterns occur at arbitrary frequencies [11].

Time series can be analysed using different methods in the time domain and in the frequency domain.

One time domain method is the autocorrelation analysis. Calculating the autocorrelation can be a helpful tool to understand how time steps are correlated with one another at a given lag. To calculate the autocorrelation ρ_h at a given lag h, the autocovariance γ_h needs to be obtained first, which is shown in Equation 2.1 [12].

$$\gamma_h = \mathbb{E}\left[(\mathcal{S}_{t+h} - \bar{\mathcal{S}})(\mathcal{S}_t - \bar{\mathcal{S}})\right] \tag{2.1}$$

Here, $\mathcal{S}$ denotes an arbitrary univariate time series such that $\mathcal{S} \in \mathbb{R}^T$, and $\bar{\mathcal{S}}$ denotes the mean of the time series $\mathcal{S}$. As is evident, for $h = 0$, the autocovariance is simply the variance. With the autocovariance established, the autocorrelation is defined as the autocovariance normalised by the variance, as shown in Equation 2.2 [12].

$$\rho_h = \frac{\gamma_h}{\gamma_0} \tag{2.2}$$

Time series which feature a high autocorrelation for a large number of lags tend to feature slower dynamics than time series featuring high autocorrelation for a more limited number of lags.

In contrast, a method in the frequency domain is a Fourier analysis. The Fourier representation of a signal decomposes it into its different sinusoidal components, revealing the different frequencies present in the time series [12]. For digital and therefore discretely sampled signals, a discrete Fourier transform F needs to be applied, which is mathematically described in Equation 2.3 [12].

$$F_k = \frac{1}{\sqrt{T}} \sum_{t=1}^{T} \mathcal{S}_t e^{-i2\pi t \frac{k}{T}} \tag{2.3}$$

Here, F_k denotes the presence of the frequency k in time series $\mathcal{S}$.

2.2 Anomaly Detection

To understand anomaly detection problems, one must become familiar with their multi-faceted nature. To provide the reader with an illustration of the relationship be-

tween terms introduced in the taxonomy used, an overview is represented graphically in Figure 2.1.

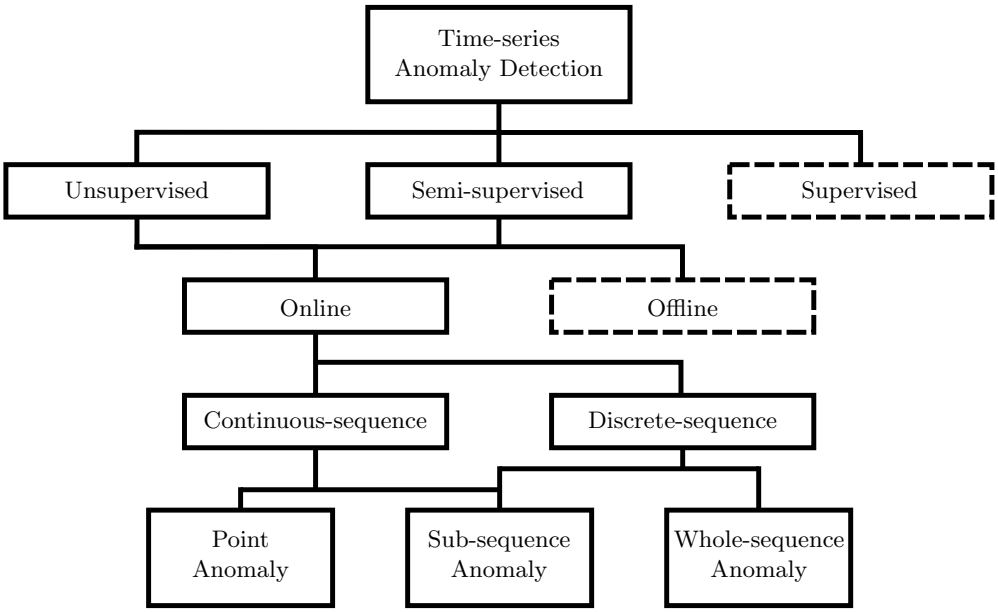

Figure 2.1: A visual representation of taxonomy used in this thesis.

2.2.1 Anomaly Types

Anomalies in time series can be assumed to have different shapes and forms. A commonly used and accepted definition [13] reads as follows:

"An observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism."

The *mechanism* can be a change in the system, like a broken component inside the system under investigation. Over the years, several taxonomies are proposed to better classify different anomalies. The taxonomy here follows the one suggested by Blásquez-García et al. [14], where anomalies are classified into point outliers, sub-sequence outliers (also known as collective anomalies in the literature) and outlier time series. While they acknowledge that the terms *outlier* and *anomaly* are widely used to refer to the same concept, Blásquez-García et al. [14] note that outliers are mostly used in cases of unwanted patterns whereas anomalies in cases of interest. In this work, these

three types will be from now on referred to as *point anomalies*, *sub-sequence anomalies* and *whole-sequence anomalies*, respectively.

Consider a testing subset $\mathcal{D}^{\text{test}}$ in a data set $\mathcal{D}$ containing N sequences, such that $\mathcal{D}^{\text{test}} = \{\mathcal{S}_1, ..., \mathcal{S}_n, ..., \mathcal{S}_N\}$, where $\mathcal{S}_n \in \mathbb{R}^{T_n \times d_{\mathcal{D}}}$. In a given multivariate sequence $\mathcal{S}_n$, also consider an anomaly event $\mathcal{A} \in \mathbb{R}^{S \times d_{\mathcal{A}}}$ of length S that is detected in $d_{\mathcal{A}}$ channels, where $d_{\mathcal{A}} \leq d_{\mathcal{D}}$.

A *point anomaly* is defined as a value at a time step that does not conform to the typical behaviour of a system, or, in other words, $S = 1$ for $\mathcal{A}$. An example of a point anomaly is illustrated in Figure 2.2. While more easily detectable than the other

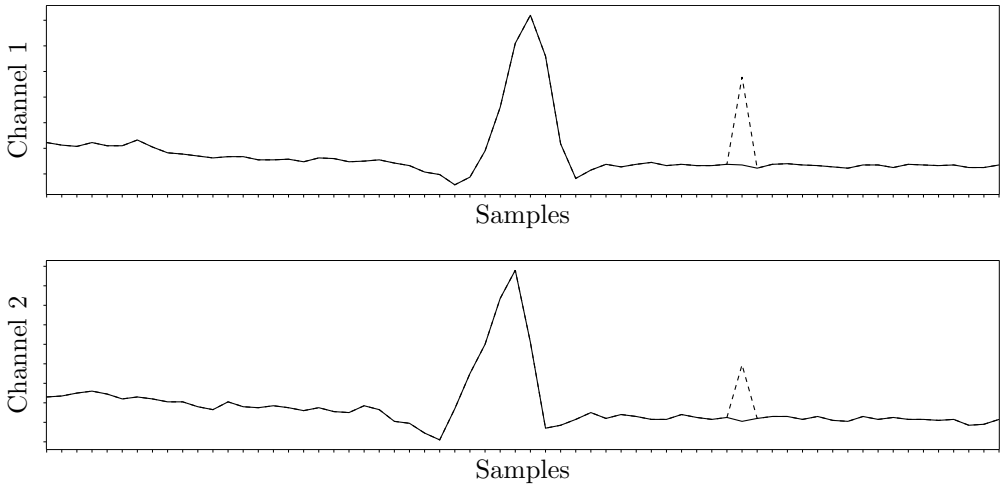

Figure 2.2: Example of a *point anomaly* in both channels. The continuous line represents the ground-truth time series and the dashed line the anomaly. The ground-truth time series shown is a heartbeat from the MIT-BIH Arrhythmia Database [1].

types, these anomalies are rare events in the real world, as systems affected cannot usually return to a nominal state before the next time step arrives unless the sampling rate is very low. Throughout this work, *nominal* is used as a synonym for *normal* or *anomaly-free* to avoid confusion with normal/Gaussian distributions.

Sub-sequence anomalies are defined by a series of anomalous time steps, i.e. a sub-sequence within a time series that does not reflect the nominal behaviour of a system, or, in other words, $1 < S < T_n$ for $\mathcal{A}$. In a real-world system, this type of anomaly may occur when a component in the said system runs at reduced functionality or fails but manages to recover to a nominal state after a period of time. An example of a sub-sequence anomaly is illustrated in Figure 2.3.

A *whole-sequence anomaly* can be seen as a long sub-sequence that has the same

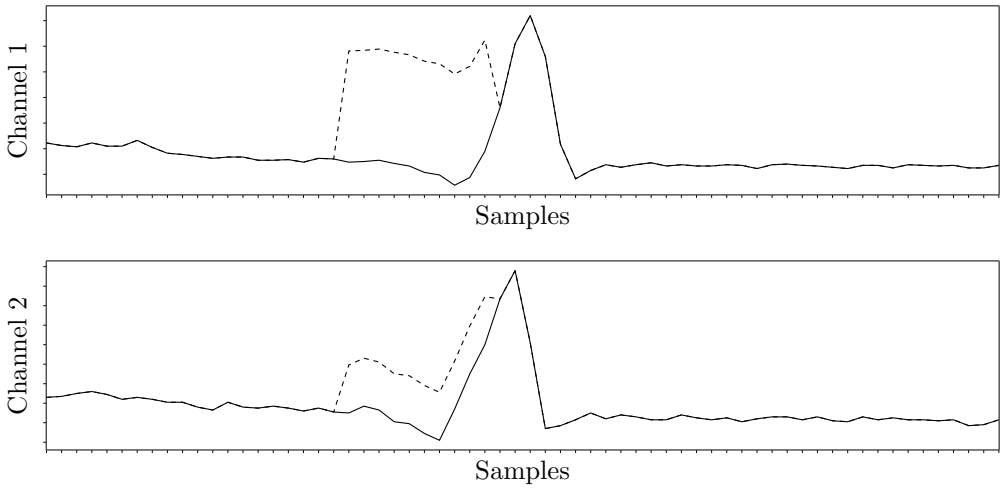

Figure 2.3: Example of a *sub-sequence anomaly* in both channels. The continuous line represents the ground-truth time series and the dashed line the anomaly. The ground-truth time series shown is a heartbeat from the MIT-BIH Arrhythmia Database [1].

length as the entire sequence, or, in other words, $S = T_n$ for $\mathcal{A}$. This type of anomaly can occur when an initial parameter or state deviates from the norm, leading to all observations in the sequence being anomalous as well. An example of a whole-sequence anomaly is illustrated in Figure 2.4.

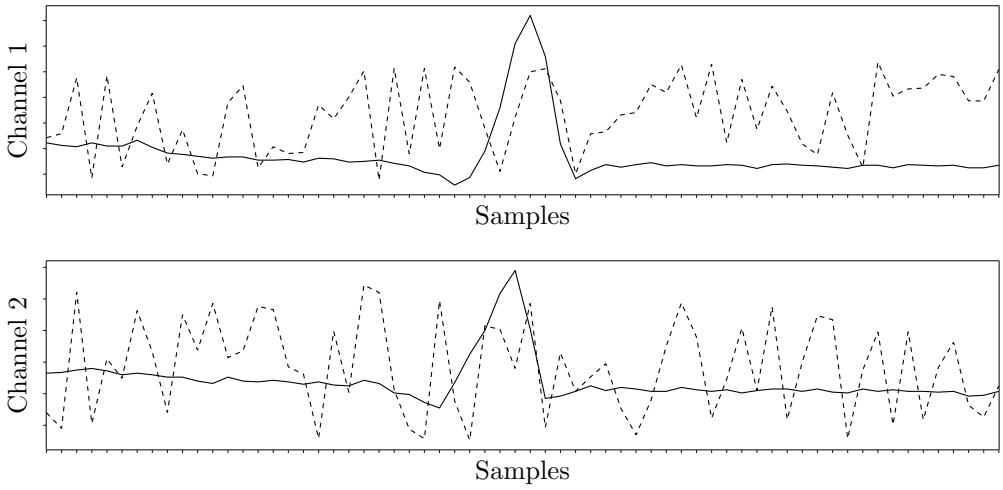

Figure 2.4: Example of a *whole-sequence anomaly* in both channels. The continuous line represents the ground-truth time series and the dashed line the anomaly. The ground-truth time series shown is a heartbeat from the MIT-BIH Arrhythmia Database [1].

2.2.2 Concept of Supervision

Analogous to types of learning, there is supervised, semi-supervised and unsupervised anomaly detection. *Supervised* anomaly detection is essentially imbalanced binary time series classification and is only rarely found in literature. This is most likely because, in real-world use cases, possible anomaly types are rarely known a priori. In addition to that, labelling data is expensive, which is why unsupervised and semi-supervised anomaly detection are more relevant in both literature and the real world. *Unsupervised* anomaly detection is independent of any labels, i.e. any available data for model training may contain both anomalous and nominal time series, and it is not known which is which [15]. In contrast to that, *semi-supervised* anomaly detection can be considered a more relaxed setting, where anomalous time series are absent from the training subset [15]. In the real world, this still requires some labelling to ensure an entirely nominal training subset, which is not always given. Some literature diverges from this taxonomy, agreeing with the supervised and semi-supervised definitions but defining unsupervised anomaly detection differently. According to Schmidl et al. [16], unsupervised anomaly detection involves detecting anomalous behaviour without a training procedure. This definition implies that learning nominal behaviour from *mostly* nominal data is not possible, which is investigated in Chapter 5.3.

2.2.3 Continuous- and Discrete-sequence Problems

Detecting anomalous behaviour in time series is referred to as *time series anomaly detection*, which can be split into two main areas: continuous- and discrete-sequence, where the former is the most common type present in public datasets. It is defined as detecting anomalies in a process that runs for a continuous time period without breaks. Typically, the test subset $\mathcal{D}^{\text{test}}$ in the dataset $\mathcal{D}$ in a continuous-sequence problem consists of a singular multivariate time series composed of multiple nominal and anomalous sub-sequences, i.e. $\mathcal{D}^{\text{test}} = \{\mathcal{S}_1\}$. Discrete-sequence anomaly detection, in contrast, is defined as detecting anomalies in N chunks of processes that happen independently of each other, such as automotive test benches, where several tests may occur sequentially but are not temporally contiguous and hence provide a time series for each test, i.e. $\mathcal{D}^{\text{test}} = \{\mathcal{S}_1, ..., \mathcal{S}_n, ..., \mathcal{S}_N\}$. Here, the testees, i.e. the test subjects, are not monitored over a continuous period of time, but are instead monitored solely during each process chunk. Automotive testing is not the only use for discrete-sequence anomaly detection, however. Another discrete-sequence problem could also include the analysis of the flight behaviour of an aeroplane, where the time while it is

docked is irrelevant and may not be recorded. Therefore, datasets for discrete-sequence anomaly detection consist of several nominal and anomalous time series, where a given anomalous time series may be entirely anomalous, in case of a whole-sequence anomaly, or only partly, in case of a point or sub-sequence anomaly.

2.2.4 Online Anomaly Detection

Detecting anomalous behaviour in a timely manner is also advantageous, since the source of anomalous behaviour may bring about damage to said system. Such problems require a time series to be evaluated as it is being recorded, not just once it is finished; such problems are referred to as *online* time series anomaly detection. Using model-based approaches generally involves two processes: training and inference. Training is the process of automatically adjusting model parameters by means of optimisation using training data, whereas inference refers to the application, where model parameters are no longer adjusted. Online approaches are defined as models that perform inference in an online manner and, optionally, are trained in an online manner. Online training, therefore, refers to the process of adjusting model parameters as training data is streamed. Unlike offline training, which is done once in most cases, online training runs continuously and for an undefined amount of time. Online inference refers to the ability of an approach to detect anomalous behaviour in time steps as soon as they are observed, rather than waiting for the time series to have finished streaming and only then evaluating the entire sequence, i.e. offline. This is especially useful in real-world use cases where timely detection is important. A more strict version of online anomaly detection is real-time anomaly detection, where the current time step is evaluated before the next one arrives.

2.3 Long Short-term Memory Cells

Introduced in 1997 by Hochreiter and Schmidhuber [17], long short-term memory (LSTM) cells were developed to prevent gradients from exploding or disappearing when propagating backwards in time during training, a problem associated with regular recurrent reural retworks (RNN) cells. This architecture still suffered from a problem, where cell states may increase uncontrollably and hence an evolution featuring so-called forget gates was introduced by Gers et al. [18] to avoid saturation of the output activation function. The LSTM cell consists of four gates, which compute the current cell state c_t and the hidden state h_t at time step t. The hidden state is used as the

output of the cell to other downstream layers, whereas the cell state is used as the memory of the cell, which includes relevant information from previous time steps and enables LSTM cells to represent temporal dependencies. Other architectures, like the gated recurrent unit (GRU) [19] combine the hidden and cell states into one state, due to their overlapping functionality. Consider an arbitrary multivariate time series $\mathcal{S} \in \mathbb{R}^{T \times d_{\mathcal{D}}}$ and an LSTM layer it is fed into. The hidden and cell states are computed by feeding the input at the current time step $\mathcal{S}_t$ into the forget gate, the cell candidate gate, the input gate and the output gate. The inner connections between the different gates inside an LSTM cell is graphically depicted in Figure 2.5. These gates are

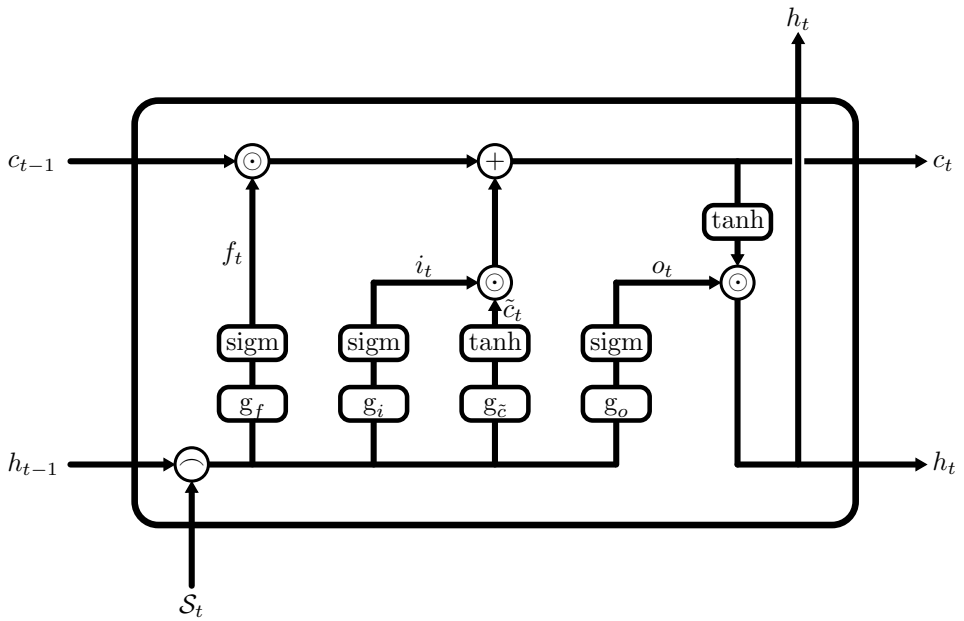

Figure 2.5: Illustration of the topology of a long short-term memory cell, with $\mathcal{S}_t$, c_t and h_t denoting the current input time step, cell state and hidden state at time step t, respectively. Moreover, $\odot$ denotes the Hadamard product and $\frown$ denotes the concatenation of two vectors.

parametrised by three sets: input weights W, the recurrent weights R and the input bias b, as shown in Equation 2.4.

$$W = \begin{bmatrix} W_i \\ W_f \\ W_{\tilde{c}} \\ W_o \end{bmatrix} \qquad R = \begin{bmatrix} R_i \\ R_f \\ R_{\tilde{c}} \\ R_o \end{bmatrix} \qquad b = \begin{bmatrix} b_i \\ b_f \\ b_{\tilde{c}} \\ b_o \end{bmatrix} \tag{2.4}$$

The forget gate controls the level of reset of the previous cell state c_{t-1}. It outputs forget gate mask f_t denoting how open the gate is to passing information and is modelled using Equation 2.5.

$$f_t = \sigma(\mathrm{g}_f) = \sigma(W_f \cdot \mathcal{S}_t + R_f \cdot h_{t-1} + b_f) \tag{2.5}$$

The resulting forget gate output f_t then multiplied by the cell state from the previous time step c_{t-1} to mask information deemed unimportant.

The cell state candidate $\tilde{c}_t$ contains new potential information that will be added and is computed as shown in Equation 2.6.

$$\tilde{c}_t = \tanh(\mathrm{g}_{\tilde{c}}) = \tanh(W_{\tilde{c}} \cdot \mathcal{S}_t + R_{\tilde{c}} \cdot h_{t-1} + b_{\tilde{c}}) \tag{2.6}$$

The input gate controls the level of update of the cell state candidate $\tilde{c}_t$ and protects the memory contents from perturbation by irrelevant inputs and forward-flowing activation [17, 18]. Its output is the input gate mask i_t which is calculated using Equation 2.7.

$$i_t = \sigma(\mathrm{g}_i) = \sigma(W_i \cdot \mathcal{S}_t + R_i \cdot h_{t-1} + b_i) \tag{2.7}$$

The product of the cell candidate $\tilde{c}_t$ and the input gate mask i_t is then added to the masked previous cell state and becomes the current cell state c_t. Hence, the current cell state c_t can be expressed using Equation 2.8.

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

Here, $\odot$ designates the Hadamard Product.

The output gate fulfils a similar role to the forget gate. The output of this gate is used to protect the cell from backward flowing error [18]. This can be represented by Equation 2.9.

$$o_t = \sigma(\mathrm{g}_o) = \sigma(W_o \cdot \mathcal{S}_t + R_o \cdot h_{t-1} + b_o) \tag{2.9}$$

The current cell state c_t is then normalised by a hyperbolic tangent function and then element-wise multiplied with the output gate mask o_t to retain the information needed. The product is the current hidden state h_t, Equation 2.10.

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

Evidently, if the LSTM layer discussed above is placed after another LSTM layer, its input is no longer the current time step $\mathcal{S}_t$ of the input time series, but rather the current hidden state h_t of the previous layer.

An LSTM layer in its "unrolled" state can be thought of a chain of such LSTM cells, all identically parameterised, which pass the computed cell- and hidden states to one another to model temporal dependencies. Another way to think about an LSTM layer is in its "rolled" state as a single LSTM cell, where its outputs are fed back as its inputs.

Bidirectional long short-term memory (BiLSTM) layers are the composition of two parallel LSTM layers, each modelling time in a different direction, whose hidden states are concatenated before being passed to the following layer.

2.4 Variational Autoencoders

The variational autoencoder (VAE) [20, 21] is a generative model that structurally resembles an autoencoder, but is theoretically derived from variational Bayesian statistics. In contrast to the regular deterministic autoencoder, the VAE uses the evidence lower bound (ELBO), which is a lower bound approximation of the so-called log evidence $\log p_\theta(\mathbf{X})$, as its objective function. The ELBO, Equation 2.11, can be expressed as the reconstruction log-likelihood and the negative Kullback-Leibler divergence D_{KL} between the approximate posterior $q_\phi(\mathbf{Z}|\mathbf{X})$ and the prior $p_\theta(\mathbf{Z})$, which is typically assumed to be a Gaussian distribution [22]. The training data does not have to follow a Gaussian distribution, since the latent distribution is a nonlinear mapping of the training data.

$$\mathcal{L}_{\theta,\phi}(\mathbf{X}) = \mathbb{E}_{\mathbf{Z}\sim q_\phi(\mathbf{Z}|\mathbf{X})}\left[\log p_\theta(\mathbf{X}|\mathbf{Z})\right] - D_{\mathrm{KL}}(q_\phi(\mathbf{Z}|\mathbf{X})||p_\theta(\mathbf{Z})) \tag{2.11}$$

Here, $\mathbf{Z} \in \mathbb{R}^{w\times d_\mathbf{Z}}$ is the sampled latent matrix, $\mathbf{X} \in \mathbb{R}^{w\times d_\mathcal{D}}$ denotes the input window and w represents the window length, whereas $d_\mathcal{D}$ and $d_\mathbf{Z}$ refer to the dataset and latent matrix dimensionality, respectively. Gradient-based optimisation minimises an objective function and the goal is the maximisation of the ELBO, and hence the final loss function is defined as the negative of Equation 2.11, shown in Equation 2.12.

$$\mathcal{L}_{\mathrm{VAE}} = -\mathcal{L}_{\theta,\phi}(\mathbf{X}) \tag{2.12}$$

Finally, to enable the backpropagation through the otherwise intractable gradient

of the ELBO, the *reparameterisation trick* [20] is applied, shown in Equation 2.13. This is one of the reasons for the use of a Gaussian distribution as the latent distribution, although any distribution type from the "location-scale" type can be used [20], like a Laplace distribution.

$$\mathbf{Z} = \boldsymbol{\mu}_{\mathbf{Z}} + \epsilon \cdot \boldsymbol{\sigma}_{\mathbf{Z}} \tag{2.13}$$

Note that ϵ is simply sampled from a standard Gaussian distribution, such that $\epsilon \sim \mathcal{N}(0, 1)$ and $(\boldsymbol{\mu}_{\mathbf{Z}}, \log \boldsymbol{\sigma}^2_{\mathbf{Z}}) = q_\phi(\mathbf{X})$.

2.5 Multi-head Attention

To simplify the explanation of multi-head attention (MA) as employed in this work, multi-head self-attention (MS) will be described instead, with the small difference between MA and MS being pointed out at the end. MS consists of two different concepts: self-attention and its multi-head extension. Self-attention is nothing more than scaled dot-product attention [23] where the key, query, and value are the same. The scaled dot-product attention score is the softmax [24] of the product between query matrix $\mathbf{Q}$ and key matrix $\mathbf{K}$, which is scaled by $\sqrt{d_{\mathbf{K}}}$. The product between the attention score and the value matrix $\mathbf{V}$ yields the context matrix $\mathbf{C}$, as shown in Equation 2.14.

$$\mathbf{C} = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_{\mathbf{K}}}}\right)\mathbf{V} \tag{2.14}$$

Compared to recurrent or convolutional layers, self-attention offers a variety of benefits, such as the reduction of computational complexity, as well as an increased amount of operations that can be parallelised [23]. Moreover, self-attention inherits an advantage over Bahdanau-style attention [25] from the underlying scaled dot-product attention mechanism: it can run efficiently in matrix multiplication manner [23].

Multi-head self-attention then allows the attention model to attend to different representation subspaces [23], in addition to learning useful projections, rather than being a stateless transformation [26]. This is achieved using weight matrices $\mathbf{W}_i^Q$, $\mathbf{W}_i^K$, $\mathbf{W}_i^V$, which contain trainable parameters and are unique for each head i, as shown in Equation 2.15.

$$\mathbf{Q}_i = \mathbf{Q}\mathbf{W}_i^Q \qquad \mathbf{K}_i = \mathbf{K}\mathbf{W}_i^K \qquad \mathbf{V}_i = \mathbf{V}\mathbf{W}_i^V \tag{2.15}$$

Once the query, key and value matrices are linearly transformed via the weight ma-

trices, the context matrix $\mathbf{C}_i$ for each head i is computed using Equation 2.16.

$$\mathbf{C}_i = \text{Softmax}\left(\frac{\mathbf{Q}_i\mathbf{K}_i^T}{\sqrt{d_\mathbf{K}}}\right)\mathbf{V}_i \tag{2.16}$$

Then, for h heads, the different context matrices are concatenated and linearly transformed again via the weight matrix $\mathbf{W}^O$, resulting in the multi-head context matrix $\mathbf{C} \in \mathbb{R}^{w \times d_\mathbf{z}}$, Equation 2.17.

$$\mathbf{C} = [\mathbf{C}_1, ..., \mathbf{C}_h]\mathbf{W}^O \tag{2.17}$$

The underlying mechanism of MA is identical to MS, with the only difference being that $\mathbf{K} = \mathbf{Q} \neq \mathbf{V}$. Essentially, MA finds which time steps correlate most with each other inside a given input window and weighs the time steps in context matrix $\mathbf{C}$ accordingly. The benefit of this alteration is investigated in Chapter 5.3.

2.6 Active Learning

The oldest literature in the field of active learning deals mostly with data augmentation for supervised classification algorithms, though lately, it has also found application in anomaly detection. In the context of model-based anomaly detection, active learning typically involves two components in a loop, as shown in Figure 2.6.

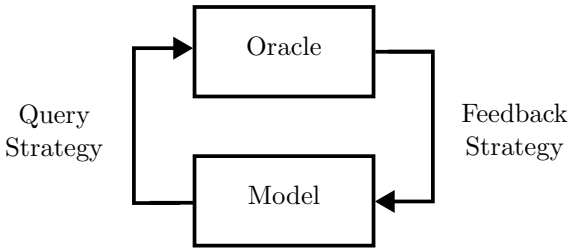

Figure 2.6: Interaction between the oracle and a model-based anomaly detector.

The first component is the oracle, also referred to as a user, human annotator, or domain expert, whose task is to label previously unlabelled samples as nominal or anomalous, so they can be used for other tasks. The second component is the anomaly detection model, which is trained in an unsupervised manner, i.e. on unlabelled data, or in a semi-supervised manner, i.e. on nominal-only data. The samples labelled by the oracle are used to calibrate the model to improve detection performance. The oracle and the model interact with each other using the query strategy (QS) and the

feedback strategy (FS).

The QS is defined as the procedure used to choose the samples to be sent to the oracle [27]. The goal of the QS is to select the samples from a candidate pool that provides the most useful information for the FS. Labelling is expensive, and hence the QS is often constrained by a parameter called the query budget, which represents the number of samples an oracle is willing to label at a time.

The FS is defined as the method of applying the knowledge in form of the labelled samples. In the case of data augmentation supervised classification algorithms, the FS is straight-forward, however, in the unsupervised anomaly detection, it plays a larger role due to the inherent incompatibility of unsupervised methods with labelled data.

2.7 Dynamic Time Warping

Originally proposed by Vintsyuk [28], Sakoe and Chiba [29], dynamic time warping (DTW) is used to calculate the optimal alignment of two time series that may be of different lengths and out of sync w.r.t. each other.

To calculate the DTW distance δ, consider two univariate sequences $\mathbf{x}$ and $\mathbf{y}$, where $\mathbf{x} \in \mathbb{R}^{T_\mathbf{x}}$, $\mathbf{y} \in \mathbb{R}^{T_\mathbf{y}}$, and u, v represent arbitrary indices within the respective sequences, such that $\mathbf{x} = [x_1, ..., x_u, ..., x_{T_\mathbf{x}}]$ and $\mathbf{y} = [y_1, ..., y_v, ..., y_{T_\mathbf{y}}]$. The local cost matrix $\mathbf{G}$ represents the cost of aligning each time step in $\mathbf{x}$ with each time step in $\mathbf{y}$. The implementation of DTW used in this work [30] uses the Euclidean norm for the computation of the local cost. Hence, the local cost between $\mathbf{x}$ at index u and $\mathbf{y}$ at index v is as shown in Equation 2.18.

$$\mathbf{G}_{u,v} = (x_u - y_v)^2 \tag{2.18}$$

The optimal path through $\mathbf{G}$ represents the path where the cumulative cost is lowest and can be found using dynamic programming. In the cumulative cost matrix $\mathbf{D}$, each index denotes the cumulative cost of arriving at the respective index; it is calculated as shown in Equation 2.19.

$$\mathbf{D}_{u,v} = \mathbf{G}_{u,v} + \min \begin{cases} \mathbf{D}_{u-1,1} & \text{for } u > 1 \\ \mathbf{D}_{1,v-1} & \text{for } v > 1 \\ \mathbf{D}_{u-1,v-1} & \text{for } u, v > 1 \end{cases} \tag{2.19}$$

Finally, the DTW distance δ equals the square root of the cumulative cost of the

optimal path, as shown in Equation 2.20.

$$\delta = \sqrt{\mathbf{D}_{T_{\mathbf{x}},T_{\mathbf{y}}}} \tag{2.20}$$