POLITECNICO DI TORINO

Master's Degree
in Mathematical Engineering

Master's Thesis

Anomaly Detection in Multivariate Time Series using Graph Convolutional Networks

Supervisor
prof. Francesco Vaccarino

Candidate
Giulio Nenna

Academic Year 2023-2024

Summary

In the era of data-driven applications, time series data is prolific, particularly in domains involving sensor-monitored systems. Such systems often produce a stream of multivariate time series data that can be leveraged to predict malfunctions and failures or, more broadly, anomalous events. Prediction of anomalies in time series data is a challenge that has been extensively pursued by researches, as failure forecast could save crucial resources in most applications. In this thesis, classical approaches to this problem will be discussed and a new approach that leverages graph convolutional neural networks will be presented. Such approach enables a much more in-depth analysis of multivariate time series since it allows to detect not only temporal correlations but also correlations between features. Moreover, cutting edge methodologies such as temporal convolutions and variational autoencoders will be exploited to build a powerful anomaly detection pipeline that is able to rival state-of-the art algorithms.

Contents

List of Tables

List of Figures

Chapter 1

Anomaly Detection and Time Series

1.1 An introduction to Time series

In science and engineering, there are numerous cases where one deals with data collected systematically over time. In general, we refer to *Time Series* whenever we encounter data observed at different time points. The process of collecting data over time, also known as *sampling*, can be done at regular or irregular time intervals, and the collected data can be multidimensional or unidimensional numerical values.

One of the most prominent examples of time series is the vast amount of data related to the stock market. It is possible to consider the price of a financial instrument (e.g., stock of a publicly traded company, see Fig. 1.1) as a univariate time series, i.e., a set of real data observed orderly over time. If we also consider data related to a subset of the market (e.g., the price of a set of stocks), then we are dealing with a set of time series that are not necessarily independent, and we refer to them as multivariate time series.

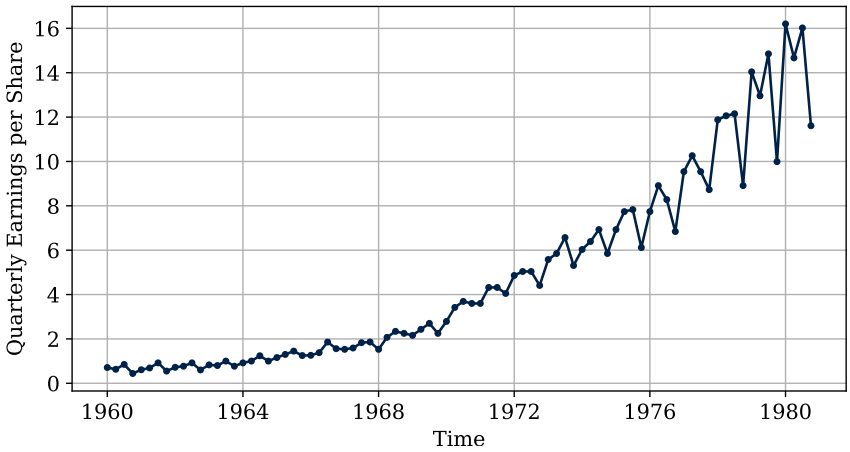


Figure 1.1: Johnson & Johnson Quarterly Earnings Per Share, 1960-1980

The main goal of time series analysis is to develop mathematical models that provide plausible descriptions of sampled data. For this reason, it is necessary to build a statistical framework to define a general model useful for describing time series effectively. One of the most commonly used statistical models for describing a time series is that of *Stochastic Processes*. To define them, it is necessary to introduce some basic probabilistic concepts.

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be such that:

- Ω represents the set of events.
- $\mathcal{F}$ represents the σ-algebra defined there.
- $\mathbb{P}$ represents the probability measure defined on Ω and $\mathcal{F}$.

Also, let $(E, \mathcal{E})$ be a measurable space, also known as the **State Space**, where $\mathcal{E}$ represents the σ-algebra defined on E.

Furthermore, let I be the **Set of Indices**, which, in real-world applications, represents the space in which times exist, and can therefore be discrete or continuous.

Definition 1. *A **Stochastic Process** $(X_t)_{t\in I}$ is defined as a family of random variables such that, for a fixed t, the following holds:*

$$\begin{aligned} X_t :\Omega &\longrightarrow E \\ \omega &\longrightarrow X_t(\omega) \end{aligned}$$

In other words, $X_t^{-1}(B) \in \mathcal{F}$, $\forall B \in \mathcal{E}$.

Therefore, a time series is nothing more than a stochastic process, i.e., a set of random variables ordered in time, and the observed data are nothing but the realization of a stochastic process. The set of indices I can be discrete or continuous, finite or infinite, and in the case of time series, it represents the sampling instants.

Theoretically, most processes described by a time series can be observed at any continuous point in time, and conceptually it would be more correct to treat them as continuous stochastic processes (i.e., with a continuous set of indices). The theoretical framework that supports stochastic processes allows for the treatment of continuous-time processes. However, for the purposes of this thesis, discrete-time processes will be of interest, which, in applications, always have a finite time: $\{X_t\}_{t\in I} = \{X_1, X_2, \dots X_n\}$.

Example 1 (White Noise)**.** One of the simplest models is that of white noise. Simply put, it is assumed that all observations are identically distributed and independent with a centered normal distribution:

$$X_t = W_t$$

$$W_t \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2).$$

In particular, if $X = \{X_t\}_{t\in I}$, then X is a multivariate normal with a diagonal variance-covariance matrix. Therefore, all observations are independent of each other, and there is no correlation between past and future.

$$X \sim \mathcal{N}(0, \sigma^2 I)$$

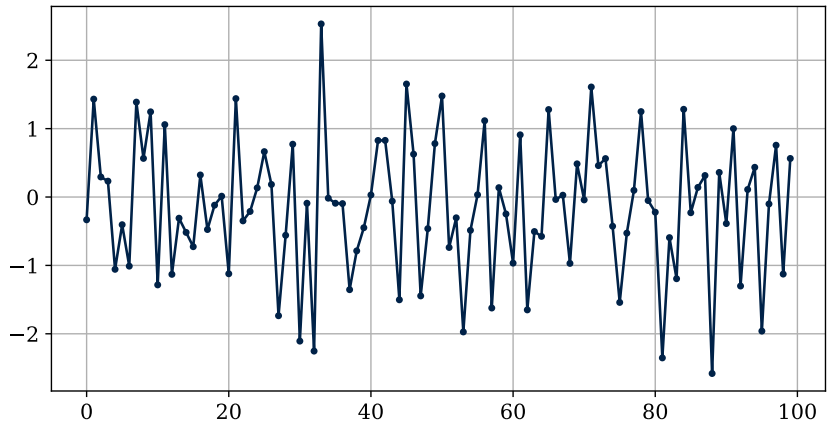


Figure 1.2: Realization of white noise with $\mu = 0$ and $\sigma^2 = 1$ (100 observations)

1.1.1 Stationarity and Autocorrelation

A complete description of a time series, observed as a collection of n random variables, is provided by the multivariate cumulative distribution function of the process:

$$F(x_1, x_2, \dots x_n) = \mathbb{P}(X_1 \leq x_1, X_2 \leq x_2, \dots X_n \leq x_n)$$

The functional form of the cumulative density is often unknown except in the case where the process is jointly normal. It is therefore necessary to make assumptions about the processes under consideration to obtain useful information for estimating F.

Definition 2 (Strong Stationarity)**.** *A process $\{X_t\}_{t \in I}$ is said to be **strongly stationary** if, for every $n \in \mathbb{N}$, for every n-tuple $(t_1, t_2, \dots t_n) \in I^n$ and for every choice of h such that $t_i + h \in I$, the following holds:*

$$P(X_{t_1} \leq x_1 \dots X_{t_n} \leq x_n) = P(X_{t_1+h} \leq x_1 \dots X_{t_n+h} \leq x_n). \tag{1.1}$$

Therefore, a process is strongly stationary if the joint cumulative density function for subsets of variables is equal to the cumulative density of the same variables shifted by a window h. From 1.1, with $n = 1$, it follows:

$$\mathbb{P}(X_t \leq c) = \mathbb{P}(X_{t+h} \leq c) \qquad \forall h \text{ s.t. } t + h \in I \tag{1.2}$$

This means that the same cumulative distribution must hold for every point in time. An example of a strictly stationary process is given by white noise 1, where it can be observed that this stationarity condition is very restrictive and limited

to an uninformative set of models. Therefore, a variant of strong stationarity is introduced, capable of encompassing more informative models:

Definition 3 (Weak Stationarity)**.** *A process $\{X_t\}_{t\in I}$ is said to be* ***weakly stationary of order*** *$s \in \mathbb{N}$ if, for every $n < s$ and every n-tuple $(t_1, t_2, \dots t_n) \in I^n$ and every choice of h such that $t_i + h \in I$, the following holds:*

$$\mathbb{E}(X_{t_1} X_{t_2} \dots X_{t_n}) = \mathbb{E}(X_{t_{1+h}} X_{t_{2+h}} \dots X_{t_{n+h}}) < \infty \tag{1.3}$$

In particular, in the case of a weakly stationary process of order 2, the following holds:

$$\mathbb{E}(X_t) = \mathbb{E}(X_{t+l}) \tag{1.4}$$
$$\mathbb{E}(X_t X_{t+l}) = \mathbb{E}(X_{t+h} X_{t+l+h}) \tag{1.5}$$

assuming that t, $t + l$, and $t + l + h$ are all in I. Therefore, from 1.4, it follows that the expected value is constant, and from 1.5, it follows that the covariance depends on the temporal **lag** that separates two observations. In fact:

$$\mathrm{Cov}(X_{t_i}, X_{t_i+l}) = \mathbb{E}(X_{t_i} X_{t_i+l}) - \mathbb{E}(X_{t_i})\mathbb{E}(X_{t_i+l}) = \gamma(l) \tag{1.6}$$

Therefore, we define $\gamma(\cdot)$ as the **autocovariance** function, and from 1.6, it follows that $\gamma(0) = \mathrm{Var}(X_{t_i})$, while $\gamma(l) = \gamma(-l)$. At this point, it is possible to define an important measure regarding time series:

Definition 4 (Autocorrelation)**.** *Let $\{X_t\}_{t\in I}$ be a weakly stationary time series of order $s \geq 2$. Then it is possible to define the* ***autocorrelation function*** *(ACF) $\rho(\cdot)$ as:*

$$\rho(l) = \frac{Cov(X_{t_i}, X_{t_i+l})}{Var(X_{t_i})} = \frac{\gamma(l)}{\gamma(0)}. \tag{1.7}$$

Observation 1. The autocorrelation function is defined assuming weak stationarity of the process. It is also possible to define autocorrelation and correlation for non-stationary processes, but it is necessary to indicate their dependence on time since it is no longer possible to assume the result obtained in 1.5:

$$\gamma_{t_i}(l) = \mathrm{Cov}(X_{t_i}, X_{t_i+l}) \tag{1.8}$$
$$\rho_{t_i}(l) = \frac{\mathrm{Cov}(X_{t_i}, X_{t_i+l})}{\sqrt{\mathrm{Var}(X_{t_i})\mathrm{Var}(X_{t_i+l})}} \tag{1.9}$$

Given the definitions of these two quantities, it remains to introduce their respective estimators. In the case of non-stationarity, an estimate of the autocorrelation at different time instants should be calculated using multiple realizations of the same time series $\{X_t\}_{t\in I}$. In most cases, this is practically impossible since there is often only one realization of the time series under consideration. In particular, let $x = (x_1, x_2, \dots x_n)$ be a realization of the process $\{X_t\}_{t\in I}$. Then, assuming at least weak stationarity of order 2, it is possible to estimate the autocovariance and autocorrelation functions through:

$$\hat{\gamma}(l) = \frac{1}{n}\sum_{i=l+1}^{n}(x_i - \bar{x})(x_{i-l} - \bar{x}) \tag{1.10}$$

$$\hat{\rho}(l) = \frac{\sum_{i=l+1}^{n}(x_i - \bar{x})(x_{i-l} - \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{1.11}$$

The calculation of autocorrelation can be visualized in a plot called a "sample autocorrelation function" that shows the autocorrelation value varying with the lag. An example of a sample autocorrelation function can be seen in figure 1.3, where the autocorrelation of the white noise introduced in example 1 is calculated. In particular, it can be noted that the first value is equal to 1, while all others are very close to 0. This is significant in that each observation of white noise has no correlation with the past or future.

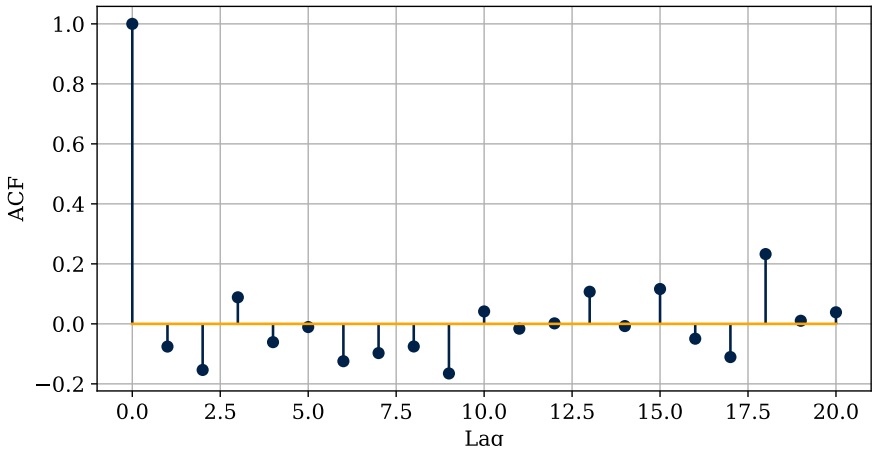


Figure 1.3: Sample autocorrelation function of white noise with $\mu = 0$ and $\sigma^2 = 1$

Observation 2. The sample ACF can also be calculated for non-stationary time series; however, in this case, the estimate loses its meaning. In the case of non-stationary processes, what is actually calculated is the autocorrelation of the process only after being purged of any trends that make it non-stationary, as we will see later.

Example 2 (Air Passengers)**.** Let's analyze a time series related to the monthly number of passengers of an airline (Fig. 1.4).

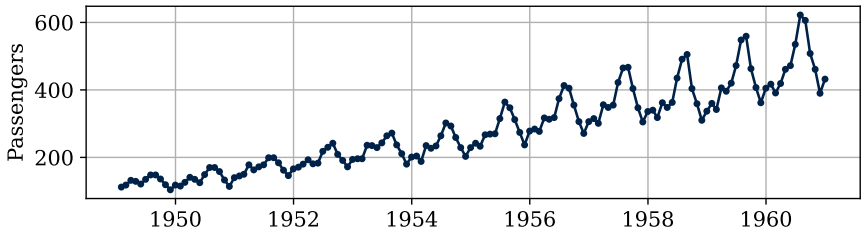


Figure 1.4: Monthly airline passengers

As visually evident, there is a poorly defined increasing trend over the years, which indicates that the series is **not** stationary, and the calculation of autocorrelation would be uninformative. It is possible to eliminate this trend component from the time series to obtain only the seasonal component through the *STL decomposition* (Fig. 1.5).

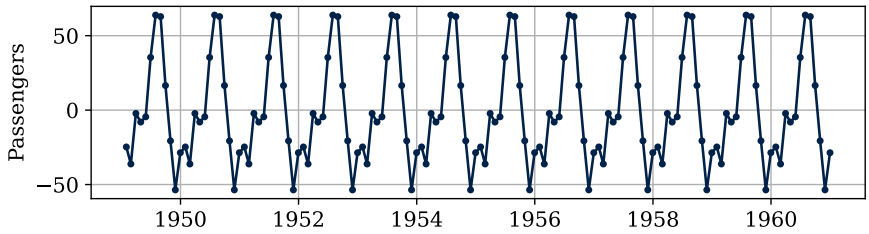


Figure 1.5: Monthly airline passengers (seasonal component)

At this point, having obtained a trend-free process, it is appropriate to calculate the sample ACF, obtaining the result in Fig. 1.6, from which it is possible to notice a positive correlation between observations with a lag $l = 12$, i.e., one year apart.

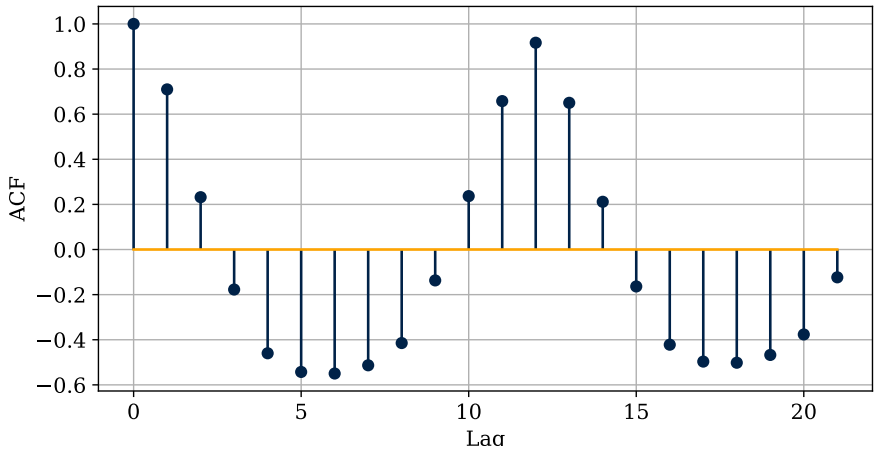


Figure 1.6: ACF of the seasonal component of the "Air Passengers" series

So far, the correlation analysis has focused on the study of an isolated time series. However, it is of common interest when one wants to infer information regarding multiple time series jointly. It might be necessary to discover if there are correlations between pairs of time series to detect their dependence.

Definition 5 (Joint Stationarity)**.** *Two time series* $\{X_t\}_{t\in I}$ *and* $\{Y_t\}_{t\in I}$ *are called* ***jointly stationary*** *if both are stationary, and if the* ***joint covariance function****:*

$$\gamma_{XY}(h) = Cov(X_{t+h}, Y_t) = \mathbb{E}[(X_{t+h} - \mu_X)(Y_{t+h} - \mu_Y)] \tag{1.12}$$

is a function that depends only on the lag h*.*

Therefore, we can define:

Definition 6. *The* ***Joint Correlation Function*** *(CCF, Cross-Covariance Function) of two jointly stationary time series* $\{X_t\}_{t\in I}$ *and* $\{Y_t\}_{t\in I}$ *is defined as:*

$$\rho_{XY}(h) = \frac{\gamma_{XY}(h)}{\sqrt{\gamma_X(0)\gamma_Y(0)}} \tag{1.13}$$

Similarly to what was done before, it is possible to define their respective estimators of joint covariance and correlation as:

$$\hat{\gamma}_{XY}(h) = \frac{1}{n}\sum_{i=1}^{n-h}(x_{i+h} - \bar{x})(y_{i+h} - \bar{y}) \tag{1.14}$$

$$\hat{\rho}_{XY}(h) = \frac{\hat{\gamma}_{XY}(h)}{\sqrt{\hat{\gamma}_X(0)\hat{\gamma}_Y(0)}} \tag{1.15}$$

1.1.2 Models for Time Series

Autoregressive Model (AR)

Definition 7. *An autoregressive model of order 1 AR(1) is a time series* $\{X_t\}_{t\in I}$ *of the form:*

$$X_t = \alpha X_{t-1} + \omega_t \tag{1.16}$$

with $|\alpha| < 1$ *and* $\omega_t \sim \mathcal{N}(0, \sigma^2)$.

By iteratively developing Eq. 1.16, assuming that I is of infinite dimension, we obtain:

$$X_t = \alpha^k \omega_{t-k} + \cdots = \sum_{i=0}^{\infty} \alpha^i \omega_{t-i} \tag{1.17}$$

It can be noticed that an autoregressive model is given by the infinite sum of i.i.d. normal random variables. Furthermore, it is possible to calculate autocorrelation by noting that:

$$\mathrm{Cov}(X_t, X_{t+k}) = \frac{\alpha^k \sigma^2}{1-\alpha^2} = \gamma(k). \tag{1.18}$$

As shown in 1.18, autocovariance does not depend on time; therefore, the process is weakly stationary of order 2 or higher. Moreover, since the sum of normal random variables is still normal, we obtain:

$$X_t \sim \mathcal{N}\left(0, \frac{\sigma^2}{1-\alpha^2}\right) \qquad X_t | X_{t-1} \sim \mathcal{N}(\alpha X_{i-1}, \sigma^2) \tag{1.19}$$

To conclude, autocorrelation is calculated using 1.7, resulting in:

$$\rho(k) = \alpha^k \tag{1.20}$$

Observation 3. The model just defined is a **zero-mean autoregressive** model. In case one wants to switch to an AR(1) model with mean μ, a variable change $Y_t = X_t - \mu$ is sufficient.

Definition 8. *An autoregressive model of order p AR(p) is a time series* $\{X_t\}_{t\in I}$ *of the form:*

$$X_t = \alpha_1 X_{t-1} + \alpha_2 X_{t-2} + \cdots + \alpha_p X_{t-p} + \omega_t \tag{1.21}$$

with $\omega_t \sim \mathcal{N}(0, \sigma^2)$.

If an operator called *Backshift Operator* B is introduced such that $B^k x_t = x_{t-k}$, then 1.21 can be rewritten as:

$$\Theta_p(B) X_t = \omega_t \tag{1.22}$$

$$X_t = \Theta_p^{-1}(B)\omega_t \tag{1.23}$$

Thus, the *backshift polynomial* $\Theta_p(B)$ is introduced as in 1.22, and considering that B is a linear operator, it is possible to explicitly express $\Theta_p^{-1}(B)$ by obtaining from 1.23:

$$X_t = \left(1 + \sum_{i=1}^{\infty} c_i B^i\right) \omega_t \tag{1.24}$$

Observation 4. The series $\sum_{i=1}^{\infty} c_i B^i$ converges if and only if the polynomial $\Theta_p(B)$ (intended as a polynomial of B) has all its roots with an absolute value strictly less than 1.

Moving Average Model (MA)

Definition 9. *A* ***Moving Average*** *model of order 1 MA(1) is a time series* $\{X_t\}_{t\in I}$ *of the form:*

$$X_t = \omega_t + \beta\omega_{t-1} = (1 - \beta B)\omega_t \tag{1.25}$$

where $\omega_t \sim \mathcal{N}(0, \sigma^2)$.

From the properties of expected value and variance, it is straightforward to derive that:

$$\mathbb{E}(X_t) = 0$$

$$\gamma(k) = \begin{cases} \sigma^2(1+\beta^2) & \text{if } k = 0 \\ \beta\sigma^2 & \text{if } k = 1 \\ 0 & \text{otherwise} \end{cases}$$

Observation 5. The MA(1) model can be expressed as an AR model of infinite order. Indeed, from 1.25:

$$(1+\beta B)^{-1}X_t = \omega_t$$
$$\left(1+\sum_{i=1}^{\infty} c_i B^i\right) X_t = \omega_t$$
$$X_t = \sum_{i=1}^{\infty} c_i B^i X_t + \omega_t$$

Definition 10. *A Moving Average model of order q MA(q) is a time series* $\{X_t\}_{t\in I}$ *of the form:*

$$X_t = \omega_t + \beta_1\omega_{t-1} + \cdots + \beta_q\omega_{t-q} \tag{1.26}$$

where $\omega_t \sim \mathcal{N}(0,\sigma^2)$.

Similarly to the autoregressive model, 1.26 can be expressed as:

$$X_t = \sum_{i=0}^{q} \beta_i\omega_{t-i} = \Phi_q(B)\omega_t \tag{1.27}$$

with $\beta_0 = 1$. Using 1.27, we obtain:

$$\mathbb{E}[X_t] = 0 \tag{1.28}$$

$$\gamma(k) = \begin{cases} 0 & \text{if } k > q \\ \sigma^2 \sum_{i=0}^{q} \beta_i^2 & \text{if } k = 0 \\ \sigma^2 \sum_{i=0}^{q-k} \beta_j\beta_{j+k} & \text{if } k \in [1,\dots q] \end{cases} \tag{1.29}$$

ARMA Model

Definition 11. *An Autoregressive Moving Average model of order (p, q) ARMA(p, q) is a time series $\{X_t\}_{t\in I}$ of the form:*

$$X_t = \underbrace{\alpha_1 X_{t-1} + \cdots + \alpha_p X_{t-p}}_{AR(p)\ part} + \underbrace{\omega_t + \beta_1 \omega_{t-1} + \cdots + \beta_q \omega_{t-q}}_{MA(q)\ part} \tag{1.30}$$

which, in compact form, can be written as:

$$\Theta_p(B) X_t = \Phi_q(B) \omega_t \tag{1.31}$$

Integrated Models

The differencing of a process is the process of order 1 differences:

$$X_t^* = X_t - X_{t-1}$$
$$X_t^* = (1 - B) X_t$$

Definition 12. *A process is said to be **Integrated of order d** I(d) if differencing it d times produces white noise:*

$$(1 - B)^d X_t = \omega_t \tag{1.32}$$

Observation 6. Differencing d times does **not** mean considering differences of order d; indeed:

$$X_t^{**} = X_t^* - X_{t-1}^* \tag{1.33}$$

The integration of a process is a useful operation in the case of non-stationary processes that exhibit trends. For example, consider a process:

$$X_t = a + bt + \omega_t$$

which is clearly a non-stationary process with a linear trend over time. Differentiating it, we get:

$$X_t^* = b + \omega_t - \omega_{t-1} = b + (1 - B)\omega_t$$

X_t^* is therefore a stationary MA(1) process. Integration allows us to study evidently non-stationary processes using stationary models that are easier to handle.

ARIMA Models

Combining what has been expressed so far, it is possible to define an important family of processes that represent the starting point for time series modeling.

Definition 13. *An Autoregressive Integrated Moving Average model ARIMA (p,q,d) is a time series $X_{tt\in I}$ of the form:*

$$\Theta_p(B)(1-B)^d X_t = \Phi_q(B)\omega_t \tag{1.34}$$

ARIMA models can be further expanded in order to account for seasonality in data (SARIMA models). These models are the foundational bricks for modeling time series and, as we will see in the next chapter, can be used for anomaly detection in time series data.

1.2 An Overview of Anomaly Detection

In today's complex and dynamic digital landscape, the rapid growth of data has become both a boon and a challenge for various industries. Among the vast sea of information, identifying irregularities or anomalies is crucial for ensuring the integrity, security, and efficiency of systems and processes. Anomaly detection, a subset of machine learning and data analytics, has emerged as a powerful tool to address this need.

Anomaly detection refers to the process of identifying patterns or instances that deviate significantly from the norm within a dataset. These anomalies can manifest in various forms, such as unusual behavior, unexpected events, or outliers that diverge from the expected patterns. The objective of anomaly detection is to discern these deviations and flag them for further investigation.

But how can we define an anomaly? Turns out that even coming out with a precise description of anomalies is not an easy task. A broad and not very descriptive definition for an anomaly can be the following:

Definition 14. *An Anomaly (or outlier) is a substantial variation from the norm Mehrotra et al. [2017]*

The fact that no precise definiton of an anomaly can be formulated suggests that a precise definition cannot exist unless the application field is specified. Still, some common traits of anomalies across all applications can be outlined.

First of all, anomaly detection approaches are based on models and prediction from past data. Many scientific and engineering fields are based on the assumption that processes exist in nature that follow certain rules, resulting in the state of a system. We then formulate hypoteses about the process from data that describe the normal behavior of a system. The primary assumption of normal behavior is stationarity, i.e. the underlying processes, that led to the generation of the data, are believed not to have changed significantly. Variations from the norm may occur in the processes, hence system may also exist in abnormal states. The task of anomaly detection is to discover such variations from the norm.

At first glance, one could think that the problem we are facing is a classification problem, i.e. separating data into two classes: anomalous and non-anomalous. This classification task could then be tackled with well known machine learning classification algorithms from simple support vector machines and decision tree to heavy-duty deep learning architectures. The fact that anomalies are variations from normal behavior is the very reason why the classification approach is often unsuccessful: anomalies are, for their nature, much rarer than non-anomalies, hence there is a strong imbalance between the two classes in any dataset and that is the main reason why classification algorithm may not work. Furthermore, we could see much variance among anomalies themselves, making it difficult to even put them under the same class.

1.2.1 Performance metrics

Similarly to classification tasks, to evaluate the performance of any anomaly detection algorithm we use metrics such as *Precision, Recall, F1 score* and *AUC-ROC score*.

Precision and Recall

Let $\hat{C}$ the random variable that represents the prediction of the anomaly detection algorithm and let C be the random variable that represents the actual class of the observed datapoint x. Let P be the *positive class* meaning the class representing anomalies (this is a class and not a random variable, both C and $\hat{C}$ could take P as a possible value). Moreover, let:

True positive number (tp): the number of datapoints that are actually anomalies and have correctly been marked by the model as anomalies ;

False positive number (fp): the number of datapoints that are not anomalies but have been mistakenly marked by the model as anomalies ;

False negative number (fn): the number of datapoints that are actually anomalies but have been mistakenly marked by the model as non-anomalies

Definition 15 (Precision and Recall)**.** *Let $\mathcal{D}$ be the dataset, then Precision (Pr) and Recall(Rc) can be defined and approximated as:*

$$Pr = \mathbb{P}(C = P|\hat{C} = P) = \frac{\mathbb{P}(C = P, \hat{C} = P)}{\mathbb{P}(\hat{C} = P)} \simeq \frac{\frac{tp}{|\mathcal{D}|}}{\frac{tp+fp}{|\mathcal{D}|}} = \frac{tp}{tp + fp} \tag{1.35}$$

$$Rc = \mathbb{P}(\hat{C} = P|C = P) = \frac{\mathbb{P}(\hat{C} = P, C = P)}{\mathbb{P}(C = P)} \simeq \frac{\frac{tp}{|\mathcal{D}|}}{\frac{tp+fn}{|\mathcal{D}|}} = \frac{tp}{tp + fn} \tag{1.36}$$

As we can notice in 1.35, Precision (Pr) is the probability that a datapoint is an anomaly, knowing that the model has classified it as an anomaly. Conversely, Recall (Rc), is the probability that a datapoint has been (correctly) classified as an anomaly, knowing that is an anomaly Roelleke [2013].

F_1 Score

When we are dealing with imbalanced classes both Precision and Recall become valuable metrics since:

Precision measures the accuracy of positive prediction and in imbalanced classes, high precision is important because it indicates that when the classifier predicts the positive class, it is likely correct.

Recall measures the ability of the classifier to capture all relevant instances of the positive class. In imbalanced classes, high recall is important because it ensures that the classifier doesn't miss many instances of the minority class.

The F_1 *Score* combines precision and recall using their harmonic mean, providing a single metric that balances these two aspects. This is useful since F_1 score will be high when both Precision and Recall are high, preventing situations where Precision dominates recall or vice-versa.

Definition 16 (F Score)**.** *F_1 score can be defined as the harmonic mean between the Precision and Recall metrics.*

$$F_1 = \frac{2}{\frac{1}{Rc} + \frac{1}{Pr}} = 2\frac{Pr \cdot Rc}{Pr + Rc} \tag{1.37}$$

Moreover 1.37 can be seen as a particular case of a more general form. In fact,

we can define the F_β score as:

$$F_\beta = (1+\beta^2)\frac{Pr \cdot Rc}{\beta^2 Pr + Rc} \tag{1.38}$$

Hence F_1 is the case where $\beta = 1$.

AUC-ROC Score

AUC-ROC (Area Under the Receiver Operating Characteristic Curve) is another performance metric used in binary classification problems to evaluate the ability of the model to discriminate between anomalies and non anomalies. In order to compute the Receiver Operating Characteristic Curve we first need to define two quantities: True Positive Rate and and False Positive Rate.

True Positive Rate is just an alias for Recall in the field of information retrieval. It can in fact be defined as in 1.36 and it represents the Probability of the model of classifying correctly an anomaly knowing that the observed datapoint is an anomaly.

On the other hand, *False Positive Rate* is the probability that the model classifies a given datapoint as anomalous knowing that said datapoint is non-anomalous. Similarly to 1.36, we can define the False Positive Rate as:

$$FPR = \mathbb{P}(\hat{C} = P | C = N) = \frac{\mathbb{P}(\hat{C} = P, C = N)}{\mathbb{P}(C = N)} \simeq \frac{fp}{fp + tn} \tag{1.39}$$

where tn is the *True Negative Number* meaning the number of non-anomalies that are correctly classified as such.

As we will see in the following chapters, anomaly detection algorithms work by learning an *Anomaly Score*. This means that the output of the algorithm is not just a 0-1 classification, instead is a score that reflects "how much a point can be considered an anomaly". This means that in order to obtain a classifier a threshold must be set on the anomaly score, above witch a point is considered anomalous.

With this knowledge on hand we can finally define the Receiver Operating Characteristic Curve as the curve obtained plotting the False Positive Rate against the True Positive Rate of the model at various threshold settings. A visual example can be seen in Figure 1.7 where the better the classifier is, the more the curve is squashed towards the $(0, 1)$ point, that represent the perfect classifier in the ROC space.

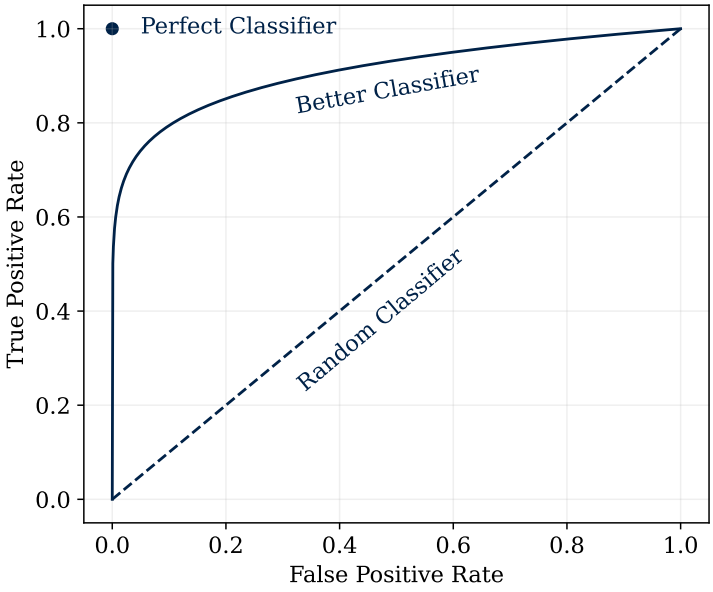


Figure 1.7: Graphical example of AUC-ROCs of multiple models

Finally, the AUC-ROC, as the name suggests, is the Area under the Receiver Operating Curve and is simply computed by integrating the ROC along the False Positive Rate axis. AUC-ROC is a score in $[0, 1]$ with 1 being the perfect classifier.

> **Observation 7.** The AUC-ROC score is useful since it decouples the value of the threshold from the quality of the predictions. This means that AUC-ROC is a score that classifies a model regardless of the threshold setting, that can be addressed *after* the model is trained and can be application dependent.

1.2.2 General Approaches to Anomaly Detection

In this section we will outline general concepts and approaches to anomaly detection. For simplicity data $\mathcal{D}$ will be considered to exist in multidimensional continuous space $\mathbb{R}^n$, but the same concepts can be applied to data that lives in discrete spaces.

Distance Based Anomaly Detection

One of the most intuitive ways to identify whether a data point is anomalous or not is by evaluating its distance to other points in the dataset. Assuming that a measure of distance exists in the space where data lives, we can then assume that

a point is anomalous if its distance to other points is "high enough". In order to classify whether a point is an anomaly or not, we need both an **anomaly measure** $\alpha(\cdot)$ and a **threshold** θ such that if a point $p \in \mathcal{D}$ is classified as an anomaly when $\alpha(p) > \theta$.

In order to define an anomaly measure we first need to define a distance measure on the space of the data. One of the most common distance measures used for data in $\mathbb{R}^N$ is the l^p distance with $p = 1$ or $p = 2$ (Euclidean Distance).

$$d_{l^p}(p, q) = \left(\sum_{i=0}^{n} |p_i - q_i|^p \right)^{\frac{1}{p}} \tag{1.40}$$

A more statistically refined version of the euclidean distance can be found in the Mahalanobis distance. Given a probability distribution Q on $\mathbb{R}^N$ with mean $\mu = (\mu_1, \mu_2, \ldots \mu_N)^T$ and a positive-definite covariance matrix S, then the Mahalanobis distance between two points $p, q \in \mathbb{R}^N$ with respect to Q is:

$$d_M(p, q; Q) = \sqrt{(p - q)^T S^{-1} (p - q)} \tag{1.41}$$

In practice S is the sample covariance matrix measuring correlations between dimensions for all points in the dataset $\mathcal{D}$. This measure of distance is used instead of the euclidean distance since dimensions are normalized by the covariance matrix and takes into account the correlations between different variables in multivariate data.

Once a distance measure has been defined, an anomaly score can be computed based on various criterions. Some examples are:

- Distance to all points: $\alpha(p) = \sum_{q \in \mathbb{D}} d(p, q)$
- Distance to nearest neighbor: $\alpha(p) = \min_{q \in \mathcal{D}, q \neq p} d(p, q)$
- Average distance to k nearest neighbors: $\alpha(p) = \frac{1}{k} \sum_{q \in \mathcal{N}_k(p)}^{k} d(p, q)$ where $\mathcal{N}_k(p)$ is the set of the k nearest neighbors to p.
- Median distance to k nearest neighbors.

Clustering Based Anomaly Detection

Clustering algorithms make the foundations of modern machine learning. They are widely used in a myriad of applications, one above all: classification. Since

Anomaly Detection can be thought as a particular kind of classification task, various clustering techniques can then be used in order to distinguish anomalies from normal data.

Given a set of points $\{p_1, p_2, ...p_n\} \in \mathcal{D}$, clustering algorithms generally assign each point (or most of the points) to a cluster C_i, $i \in \{1, \dots K\}$ where K can be a pre-defined hyperparameter or not, meaning that the number of clusters can be a-priori defined based on the clustering algorithm used. The assignment of each point is based on a distance measure or a similarity measure that must be chosen according to the space in which the data lives in. Each cluster C_i generated can be represented by their center points called *centroids* $c_i = \sum_{p \in C_i} p_j / |C_i|$.

One of the most common clustering algorithm is the *k-means clustering algorithm*, which minimizes the quantity:

$$\sum_{p \in \mathcal{D}} \sum_{c \in C} \|p_i - c_i\|^2 \tag{1.42}$$

that represent the sum of all the euclidean distances between all points and all centroids, given a fixed number k of clusters. A variation, the *Fuzzy k-means algorithm*, introduces the concept of closeness to centroids to the minimization function, arguing that points that are closer to a centroid should be given greater weightage in the minimization process. Hence, if $\mu_{i,j} \in [0, 1]$ represents the degree to witch p_i belongs to cluster C_j then the quantity to minimize becomes:

$$\sum_{p \in \mathcal{D}} \sum_{c \in C} \mu_{i,j} \|p_i - c_i\|^2 \tag{1.43}$$

Other clustering approaches are density based such as the DBSCAN clustering algorithm what, conversely to nearest-neighbor clustering, does not require a predefined number of clusters and will generate as many clusters as needed based on other hyperparameters that refer to the sensitivity of the search to density. Since this algorithm is density based then datasets with discrete or categorical data are not usable unless other density concept are used.

Assuming that clusters have been identified, several approaches are available to determine whether a point is anomalous or not. One of the simplest classification criterion is checking whether a point has been assigned to a cluster or not, assuming that the clustering algorithm used allows no cluster assignment for some of the points.
This approach cannot be used if all points are assigned to a cluster, in this case

another parameter can be used to assess the anomaly measure: proximity to other points. We can in fact compute an anomaly measure as:

$$\alpha(p) = \min_j d(p, c_j) \tag{1.44}$$

which is simply the distance of the point to the nearest centroid. It's safe to say that if this value exceeds a threshold θ then the point can be considered an anomaly. This approach works well if the number of clusters k is correctly chosen: if k is to high then the clustering "overfits" and anomalies themselves could become a cluster, if k is too low the opposite occurs and normal points could be considered anomalies.

1.3 Anomaly Detection in time series

The general approaches for anomaly detection can be applied to time series data. Anomaly detection for time series data is a research area that is thriving since time series data, characterized by its sequential and temporal nature, is ubiquitous in various domains such as finance, healthcare, manufacturing, and telecommunications.

Anomaly detection in time series data is a crucial aspect of data analysis aimed at identifying patterns that deviate significantly from the expected norm. These anomalies can manifest as sudden spikes, dips, or irregularities in the data, often indicative of abnormalities, errors, or even emerging trends. The ability to detect anomalies promptly is vital for mitigating risks, ensuring data integrity, and enhancing overall system reliability.

Several factors contribute to the complexity of anomaly detection in time series data. These include seasonality, trends, and various external factors that can influence the underlying patterns. Traditional statistical methods may fall short in capturing the intricate dynamics of time series data, leading to the growing reliance on advanced techniques, machine learning algorithms, and artificial intelligence.

In this context, modern anomaly detection approaches leverage a spectrum of methodologies, including supervised and unsupervised machine learning, deep learning, and ensemble methods. These techniques allow for the automated identification of anomalies by learning from historical data, adapting to evolving patterns, and continuously improving the detection process.

Since detecting abnormalities in time series data can be very task-specific a taxonomy of the methods that can be used will now follow Blázquez-García et al.

[2020]. Anomaly detection in time series data can be classified mainly based on two factors: type of input data and outlier type.

Input data types can be *univariate* time series 1 or *multivariate time series* 22. One can notice that univariate models can also deal with multivariate time series data as long as each feature dimension is treated as a univariate time series. This implies that correlation between dimension is not accounted hence the method is not able to detect correlation anomalies between features.

A detection method can also be defined with respect to the outlier type that it can detect. Possible type of outliers can be point outliers, subsequence outliers and time series outliers.

1.3.1 Point Outliers

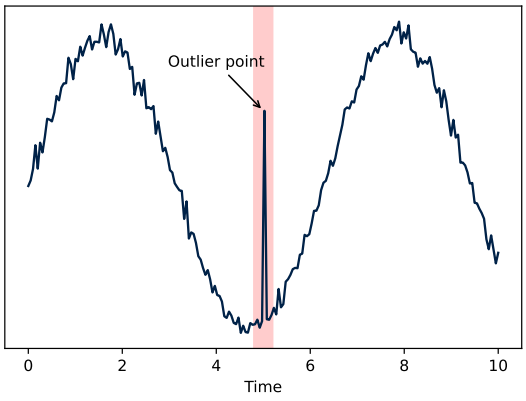
A point outlier is a datum that behaves unusually in a specific time instant when compared either to the other values in the time series (global outlier) or to its neighboring points (local outlier). Moreover point outliers can be univariate or multivariate depending on whether they affect one or more time-dependant variables respectively (figure 1.8).

The most intuitive definition for *point outlier* is a point in time that significantly deviates from its expected value. When a univariate time series is under scrutiny, then each point x_t with $t \in \{1, 2, ...T\}$ can be defined as an outlier if the difference from its expected value $\hat{x}_t$ exceeds a given threshold τ

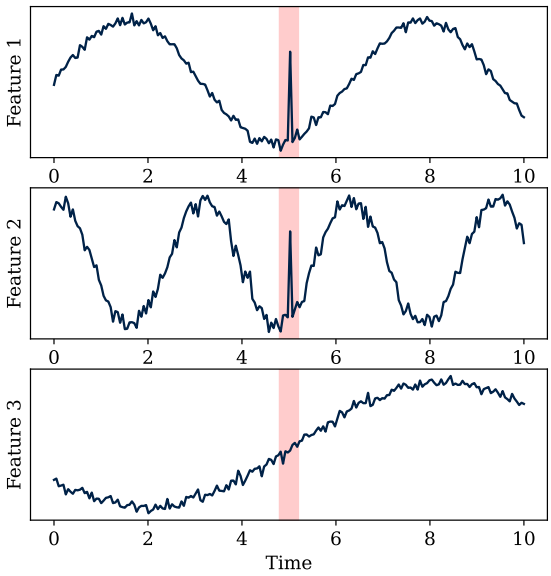
$$|x_t - \hat{x_t}| > \tau \tag{1.45}$$

This concept may seem pretty simple, but it covers the difficult task of estimating observations from time series data which constitutes its whole research area. Note that we used the term *estimating* and not *predicting* since $\hat{x}_t$ can be estimated using only previous observations or neighbor observations including future observations. This distinction is crucial since methods that only rely on past data to estimate new observation can be defined as *online algorithms* meaning that they can be deployed in real-time application where future data is not available. On the other hand, methods that leverage future data cannot be deployed in real-time for obvious reasons, unless a certain delay is allowed.

Outlier detection that make use of point estimation are based on a model that is adequately fitted to the data in order to infer the behavior of the stochastic process underlying the data. For this reason these approaches are also called *model*



(a) A point outlier in univariate data



(b) A point outlier in multivariate data

Figure 1.8: An example of point outliers

based and they differentiate one from another based on the model used to fit the data and the methodologies used for estimation. One example of model that can be used to estimate data is the ARIMA model seen in 13 from which one can estimate future data and compute its variance to evaluate whether an observation is anomalous or not.

Other point outlier detection algorithm can also be *density based* or *histogramming based*. The first category relies on the concept that x_t can be defined as an outlier if less than τ observations lie within a distance R from x_t, given a distance measure $d(\cdot, \cdot)$

$$x_t \text{ outlier} \iff |\{x \in X | d(x, x_t) \leq R\}| < \tau \tag{1.46}$$

The second category is based on the concept of histogramming time series data and the fact that a point can be considered anomalous if their removal from the time series results in a histogram representation with lower error than the original.

When dealing with multivariate time series data for point outlier detection various techniques can be adopted to overcome the dimensionality of the data. First of all, multivariate data can be treated as multiple univariate time series hence univariate

techniques can be adopted in an ensamble fashion to perform anomaly detection. This approach unfortunately disregards any dependencies that may exist between the variables.

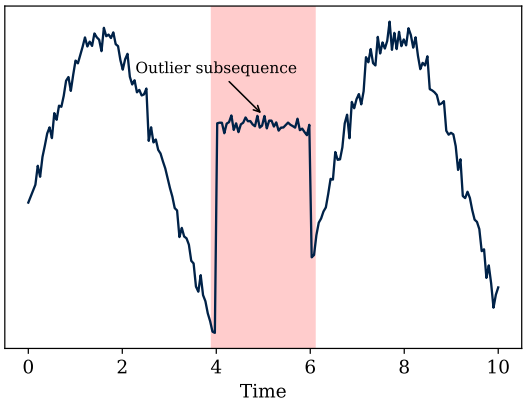
Another approach to leverage well established univariate point outlier detection algorithms with multivariate data is to use techniques to aggregate multiple features into independent time series. This means to apply a preprocessing method to the multivariate time series to find a new set of uncorrelated variables where univariate techniques can be applied. This falls under the umbrella of *dimensionality reduction*.
In contrast to this, multivariate data can be analyzed using natively multivariate techniques. As shown for univariate techniques, the most important category of multivariate anomaly detection algorithm are model based. This means that a model is fitted on multivariate time series data and is used to infer estimations on observations that will be leveraged to classify whether if an observation is an anomaly or not. Once an estimation $\hat{x}_t$ has been computed, a simple anomaly measure based on distance can be used as a decision rule:

$$\|x_t - \hat{x}_t\| > \tau \tag{1.47}$$

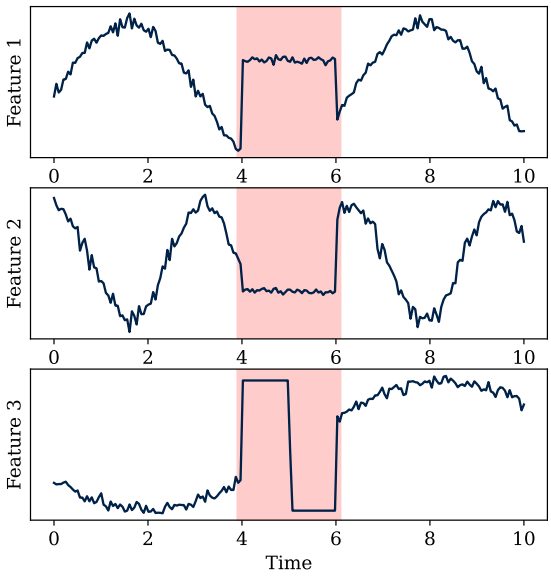
The main approaches used to experiment with anomaly detection in this thesis are model based point outlier detection algorithms and will be discussed later.

1.3.2 Subsequence outliers

A subsequence outlier is a set of consecutive points in time whose joint behavior is unusual, although each observation individually is not necessarily a point outlier. Similarly to point outliers, they can be both defined in univariate and multivariate time series (figure 1.9)



(a) A subsequence outlier in univariate dat



(b) A subsequence outlier in multivariate data

Figure 1.9: An example of subsequence outliers

One of the most successful techniques used for subsequence outliers detection is the *Discord detection approach*, first developed in Keogh et al. [2005].

A discord detection algorithm: HOT-SAX

Keogh et al. [2005] addresses the problem of searching for discords within a time series. The main goal is to find the subsequence of a time series that is the "least similar" to the others; this subsequence is informally called a "discord." This approach to anomaly detection in time series is particularly interesting as it requires only one parameter to consider, which is the length of the subsequences to examine.

Definition 17 (Subsequence)**.** *Let $X = \{X_t\}_{t \in T}$ be a discrete time series of length m (i.e., $T = \{1, 2, \dots m\}$). Then, a* ***subsequence*** *of X of length n starting from p is a time series $X'(n, p)$ defined as:*

$$X' = \{X_t\}_{t \in T'} \quad \textit{where} \quad T' = \{p, p+1, \dots p+n-1\} \tag{1.48}$$

In less formal terms, a subsequence $X'(n, p)$ is a "window" of length n extracted from X starting at time p.

Observation 8 (Sliding Window)**.** Given a time series X of length m, all subsequences of length n can be extracted using a "sliding window." Since the length of subsequences is a globally fixed parameter n, it will be omitted from the notation, considering only the starting point p: $X'(p)$.

Definition 18 (Non-Self Match)**.** *Given a time series X, let $X'(p)$ and $X'(q)$ be two subsequences of length n. They are called* ***Non-Self-Matching*** *if $|p-q| \geq n$, meaning the two subsequences do not overlap.*

Definition 18 is justified by the fact that, if the goal is to compare subsequences with each other, it makes little sense to compare two overlapping subsequences as they would exhibit a high similarity simply due to their proximity.

Definition 19 (Discords)**.** *Given a time series X, the subsequence $X'(l)$ is a* ***Discord*** *of X if $X'(l)$ has the largest distance from its nearest non-self match compared to all other subsequences. Formally, it holds:*

$$\min(Dist(X'(l), M_l)) > \min(Dist(X'(p), M_p)) \quad \forall p \in \{1, \dots m-n-1\} \quad (1.49)$$

where M_l and M_p represent the sets of non-self-matching subsequences with $X'(l)$ and $X'(p)$, respectively.

Observation 9. The distance defined can be any type of distance between time series. The required properties for the distance function are typical, particularly that it takes positive values and is symmetric. Also, before invoking the concept of distance between time series, it is necessary to **normalize** all the time series being compared to have zero mean and unit variance. For the purposes of this work, comparing time series with different offsets and amplitudes would make little sense.

Starting from Definition 19, it is intuitive to imagine a brute-force algorithm that returns the discord within a time series.

Algorithm 1 Brute force Discord Discovery.

Require: $n \in \mathbb{N}_+$, $X = \{X_t\}_{t \in T}$

```
max_dist ← 0
loc ← NaN
for p ∈ {1, 2, ... |X|−n + 1} do                          ▷ Outer loop
    nn_dist ← ∞
    for q ∈ {1, ... |X|−n + 1} do                          ▷ Inner loop
        if |p − q|≥ n then                                 ▷ Non-self match check
            if Dist(X′(p), X′(q)) < nn_dist then
                nn_dist ← Dist(X′(p), X′(q))
            end if
        end if
    end for
    if nn_dist > max_dist then
        max_dist ← nn_dist
        loc ← p
    end if
end for
```

Algorithm 1 has a computational complexity of $\mathcal{O}(|X|^2)$, making it unscalable for moderately large datasets. However, two fundamental observations can significantly improve its performance:

1. In the inner loop, it is not necessary to check all possible distances: if $X'(p)$ has at least one neighbor at a distance less than `max_dist`, then $X'(p)$ is definitely not the discord.

2. The order of checking subsequences in both the outer and inner loops influences the algorithm's performance.

The first improvement is easy to achieve: as soon as a distance smaller than `max_dist` is encountered in the inner loop, move directly to the next iteration of the outer loop. The second improvement is more complex to implement: a perfect ordering would make the algorithm much more efficient, but the ordering itself could be computationally expensive. Therefore, an heuristic is needed to approximate the perfect ordering in an economical way.

The HOT-SAX Algorithm

The ordering heuristic is based on a representation of time series developed in Lin et al. [2003] called SAX (Symbolic Aggregate ApproXimation).

A m-dimensional time series $X = \{X_1, \dots X_m\}$ can be projected onto a w-dimensional space ($w < m$) through:

$$\bar{X}_i = \frac{w}{m} \sum_{j=\frac{m}{w}(i-1)+1}^{\frac{m}{w}i} X_j \quad i \in \{1, \dots w\} \tag{1.50}$$

In practice, the original series is partitioned into w windows, and the average value is assigned to each window.

Now, let $k \in \mathbb{N}$, and define $z = \left(z_{\frac{1}{k}}, z_{\frac{2}{k}}, \dots z_{\frac{k-1}{k}}\right)$ as a vector of quantiles of $\mathcal{N}(0,1)$. Thus, z partitions $\mathbb{R}$ into k equiprobable intervals under a standard normal distribution. If each segment is associated with a "symbol" α_i with $i \in \{1, 2, \dots k\}$, we can define a symbolic representation of the time series X as follows:

Definition 20. *A series X can be defined as a **word** $\hat{X} = \{\hat{X}_1, \hat{X}_2, ...\hat{X}_w\}$ where:*

$$\hat{X}_i = \alpha_j \iff \bar{X}_i \in \left(z_{\frac{j-1}{k}}, z_{\frac{j}{k}}\right) \tag{1.51}$$

where $z_0 = -\infty$ and $z_1 = +\infty$

Definition 21. *Let X be a time series of length m, and let $X'(p)$ be its subsequences of length n with $p \in \{1, \dots m - n + 1\}$. Let $\hat{X}'(p)$ be the words of length w generated from $X'(p)$. Then, the SAX representation of X is a vector of words consisting of $m - n + 1$ words of length w generated from each subsequence of X:*

$$X^{SAX} = \{\hat{X}'(1), \dots \hat{X}'(m - n + 1)\} \tag{1.52}$$

Now we can define a sorting heuristic for the outer loop. Let $C_X = (c_1, c_2, \dots c_{m-n+1})$ be a vector where c_i is the count of occurrences of the word $\hat{X}'(i)$ within the series X^{SAX}. The first subsequences to examine in the outer loop are those with the lowest possible number of occurrences. For example, a partial ordering of the subsequence vector might have at the beginning all subsequences that have an occurrence count of 1. The intuition behind this heuristic is based on the fact that it is more likely to find discords among rarer subsequences (i.e., those with fewer occurrences).

Regarding the heuristic for the inner loop, the goal is, given a subsequence $X'(p)$ under examination, to find its nearest neighbor at a smaller distance as soon as possible in the loop. Starting from $X'(p)$, an approximation of this desired ordering is obtained by visiting first all subsequences whose word representation is the same as $X'(p)$, hoping to find more similar subsequences.

To make these heuristics efficient, multi-dimensional arrays, trees, and linked lists data structures are used. For implementation details, refer to Keogh et al. [2005].

1.3.3 Outlier Time series

When the input data is multivariate then entire time series can also be outliers when the behavior of the series significantly differs from the rest. (Figure 1.10)

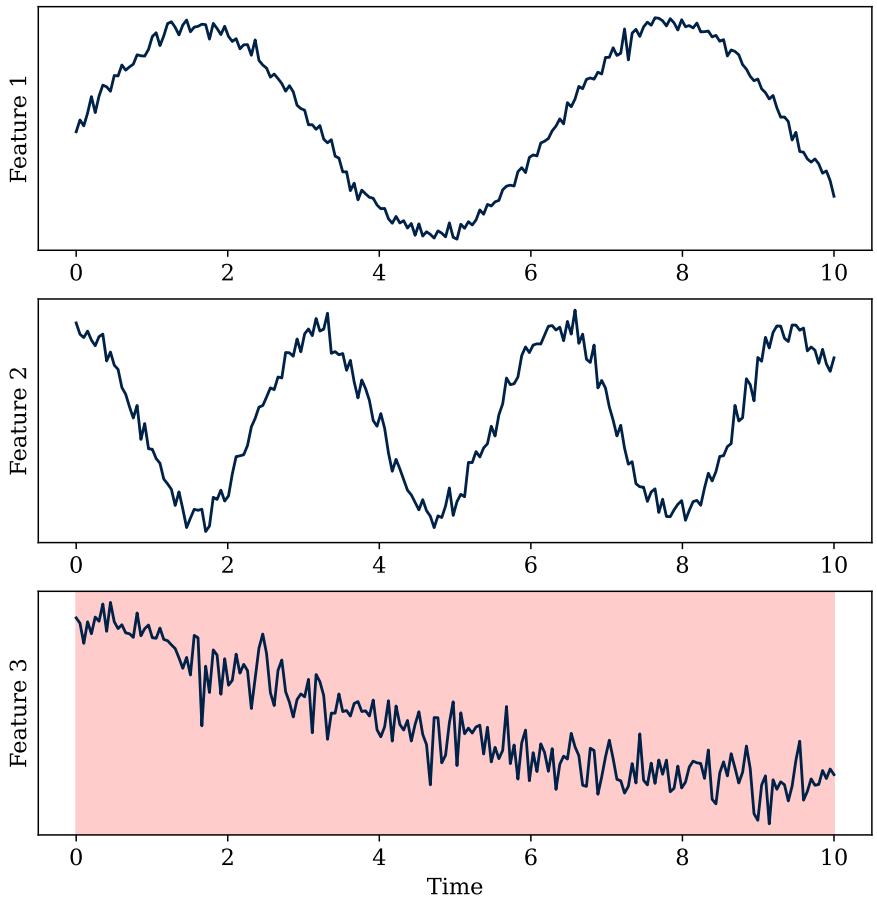


Figure 1.10: An example of a time series that can be considered an outlier (in red)

Chapter 2

Anomaly Detection Methodologies

In this chapter we will introduce the anomaly detection methodologies that have been experimented with in this thesis. In particular a class of methods known as Graph Neural Networks (GNN) has been implemented as the foundation of the full pipeline that will be shown at the end.

Graph neural networks have shown to be very effective in tasks where modeling multidimensional dependencies is crucial for the task at hand. In our case, we will show how the concept of *Graph Convolution* is used to model both temporal and feature correlation in a multivariate time-series prediction problem. The aim is to build a robust prediction model for multivariate time-series that will be used to assess an anomaly score of each observation.

To further boost the performance of the main GNN pipeline, a *Temporal Convolutional Network* will be deployed. Temporal convolution is a common practice in the Deep Learning realm and in this setting will be used as a *translation layer* that aims to give to the input data a common representation across features. This is needed since sensor data from real-world applications is very heterogeneous and naive normalization techniques are not enough to guarantee an accurate input representation of the data.

It is now necessary to expand the definition 1 of stochastic processes to introduce the so-called multivariate stochastic processes, necessary for modeling multivariate time series where, for each time instant, as many observations are recorded as there are data sources (e.g., sensors simultaneously recording different measurements).

Definition 22. *A **Multivariate Stochastic Process** $(X_t)_{t\in I}$ is defined as a family of multivariate random variables such that, for fixed t, the following holds:*

$$\begin{aligned} X_t :\Omega &\longrightarrow E \\ \omega &\longrightarrow X_t(\omega) \end{aligned}$$

That is, $X_t^{-1}(B) \in \mathcal{F}$, $\forall B \in \mathcal{E}$. In particular, X_t, for $t \in I$, are multivariate random variables, where $X_t = (X_{t,1}, X_{t,2}, \ldots, X_{t,d})$.

Operationally, a multivariate time series with real values is represented by a matrix $X \in \mathbb{R}^{n\times d}$, where n is the length of the time series, and d is the number of features. What has been said so far about anomaly detection in univariate time series can be directly applied to multivariate time series.

2.1 The MTAD-GAT Methodology for Anomaly Detection

In Zhao et al. [2020], the problem of anomaly detection for multivariate time series is addressed through convolutional neural networks. In particular, the concept of *graph convolution* or graph convolution is used to model the interdependence of observations both along the temporal and feature dimensions.

Let $X \in \mathbb{R}^{N\times d}$ be the matrix containing the multivariate time series under consideration, where N is the total length of the time series, and d is the number of features. As is common to all deep learning algorithms used in anomaly detection, K instances of smaller multivariate time series windows are generated through a sliding window procedure. These windows, denoted as $x \in \mathbb{R}^{n\times d}$, are used as input to the model. In this work, the model's performance is studied based on the implementation by ML4ITS [2023]. The original model was subsequently modified to improve its performance on the tested datasets.

A summary diagram of the original MTAD-GAT model is presented in Fig. 2.1 and analyzed in detail in the following paragraphs.

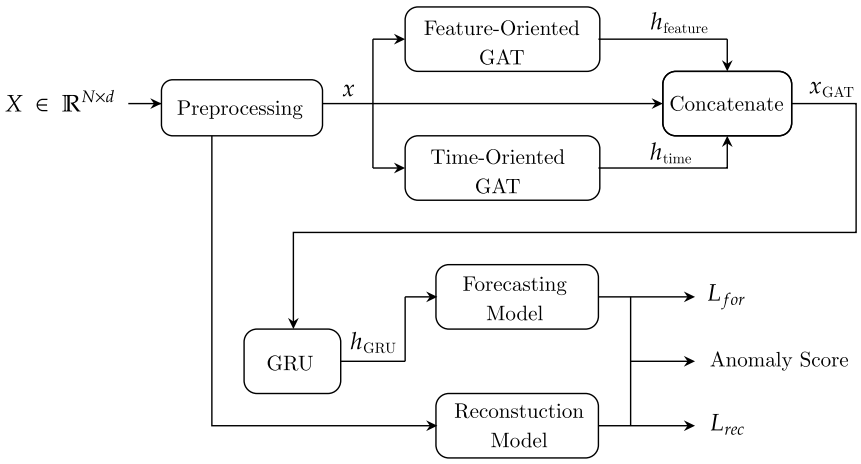


Figure 2.1: Descriptive diagram of the MTAD-GAT architecture

2.1.1 Preprocessing

Firstly, the data matrix X is column-normalized. This process can be done either through the classic `MinMaxScaler` 2.1 and `StandardScaler` 2.2 implemented in `scikit-learn`, or by experimenting with the use of `QuantileTransformer` 2.3.

$$\tilde{X}_{:,i} = \frac{X_{:,i} - \min\left(X_{:,i}^{\text{train}}\right)}{\max\left(X_{:,i}^{\text{train}}\right) - \min\left(X_{:,i}^{\text{train}}\right)} \quad i \in \{1, 2 \dots d\} \tag{2.1}$$

$$\tilde{X}_{:,i} = \frac{X_{:,i} - \mu_i}{\sigma_i} \quad i \in \{1, 2 \dots d\} \tag{2.2}$$

$$\tilde{X}_{i,j} = F_g^{-1}(q_{i,j}) \quad i, j \in \{1, 2 \dots d\} \tag{2.3}$$

Here, $X_{:,i}$ represents the i-th column of the matrix X, X^{train} is the portion of the data used for training, μ_i and σ_i are the sample mean and variance of $X_{:,i}$, and $q_{i,j}$ is the sample quantile for the observation $X_{i,j}$ relative to the column $X_{:,i}$. Finally, F_g is the cumulative distribution function of a distribution g, chosen between uniform and standard normal.

Observation 10. All three scaling methodologies serve to make the features comparable by performing a transformation that brings the data into the same range of values. Unlike `MinMaxScaler` and `StandardScaler`, `QuantileTransformer` is a less commonly used scaler. The reasons for the possible use of `QuantileTransformer` can be found in the nature of the data used within the algorithm. Often, data

from sensors have both high definition and a very large range of possible values (see Figure 2.2, an excerpt from the SWaT dataset Goh et al. [2017]).

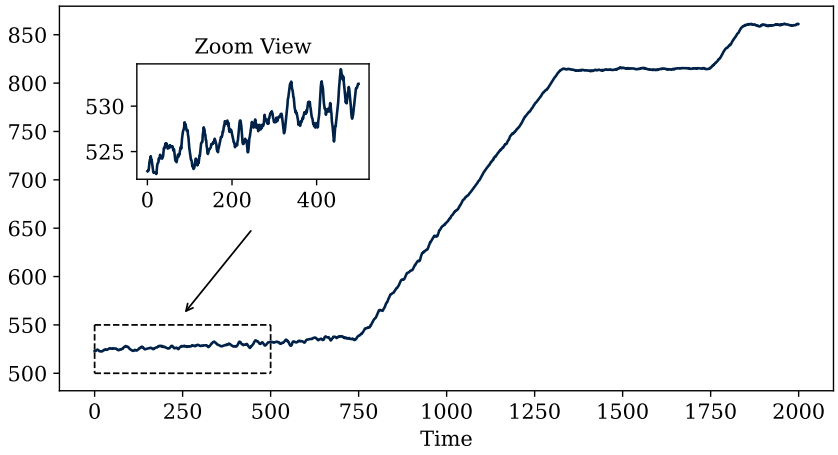


Figure 2.2: Zoom view of one of the features in the SWaT dataset Goh et al. [2017]

The `QuantileTransformer` mitigates this effect by distributing the data according to a distribution g, chosen between a uniform $U(0,1)$ and a standard normal $\mathcal{N}(0,1)$ (Fig. 2.3). The choice of the scaler is dictated by the model's performance, and it is important to consider that the `QuantileTransformer`, although mitigating the above-mentioned issues, naturally alters the distribution of the data.

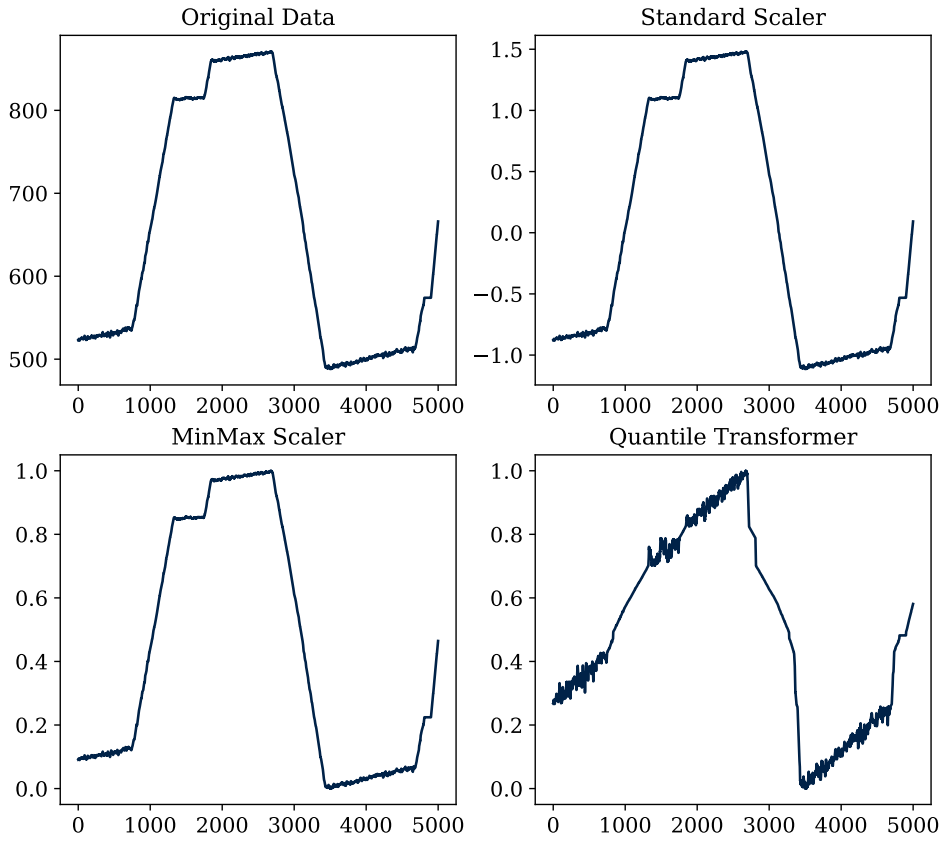


Figure 2.3: Effect of various scalers on raw data. For `QuantileTransformer`, $g = \mathcal{N}(0, 1)$

In addition to scaling, another preliminary data cleaning operation known as *Spectral Residual* (SR) Ren et al. [2019] is performed. SR is a univariate algorithm that cleans a time series based on spectral analysis of the data. Let $x = X_{:,i}$ be the column to which SR is applied; then, define:

$$\mathcal{F}(x) = (r_1 e^{\theta_1 i}, r_2 e^{\theta_2 i}, \ldots, r_n e^{\theta_n i}) \tag{2.4}$$
$$\theta = (\theta_1, \theta_2, \ldots, \theta_n) \tag{2.5}$$
$$l = (\log(r_1), \log(r_2) \ldots, \log(r_n)) \tag{2.6}$$
$$l_{\text{smooth}} = l * h_q \tag{2.7}$$
$$\delta = l - l_{\text{smooth}} \tag{2.8}$$
$$\text{SR} = |\mathcal{F}^{-1}(e^{\delta + i\theta})| \tag{2.9}$$

Here, $\mathcal{F}(x)$ is the discrete Fourier transform of the vector x, $h_q = \mathbb{1}/q \in \mathbb{R}^q$ with q fixed is used to calculate the moving average of l through convolution $*$. Ideally, SR defines a map of *salience* of the data x, where a high value of $SR(j)$ indicates that $x(j)$ is an anomaly in the frequency domain. Therefore, all points that exceed a certain threshold τ are replaced with the mode of x.

Once preprocessing is complete, windows are generated through a simple sliding window. Thus, each instance of data passed to the model is a window $x \in \mathbb{R}^{n \times d}$ extracted from the matrix X. Is worth noting that sliding windows are generated in this case with a stride number of 1 meaning that each window overlaps the next by $n - 1$ samples as shown in figure 2.4.

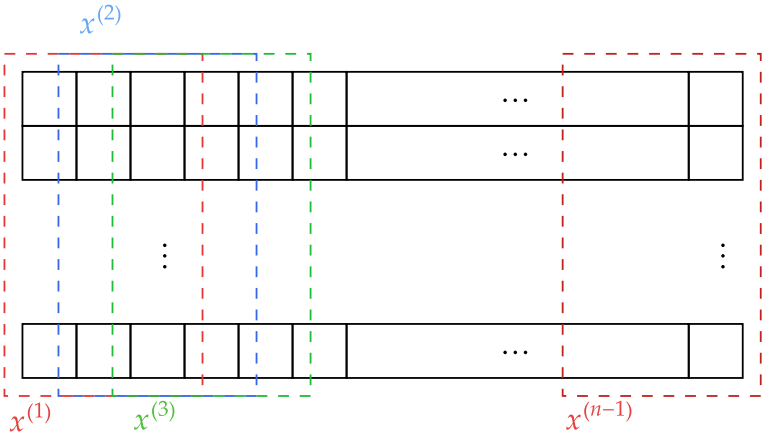


Figure 2.4: An example of a sliding window extraction with stride 1 and $n = 3$

2.1.2 Graph Attention

Graph convolution, also known as *Graph Attention* (GAT), is the core of the MTAD-GAT methodology. The graph concept is used to capture correlations both temporally and across the features of the time series. In Deep Learning a graph is often used to point out a structure that has a set of feature vectors that are connected to each other by edges. With respect to the mathematical concept of graph $\mathcal{G} = (V, E)$ where V is the set of vertices and E is the set of edges, in deep learning vertices are vectors that have a predefined length and a weight matrix A determines the weight of the connections between each vertex.

In its most general form, given $(v_1, v_2, \dots v_K)$ feature vectors $v_j \in \mathbb{R}^m$ related to each of the K nodes in a graph G, the output representation for each node is expressed as:

$$h_j = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} v_j\right) \tag{2.10}$$

Here, σ represents the sigmoid function[1], and α_{ij} represents the attention score measuring the contribution of node j to node i. Only the adjacent nodes to v_i, denoted as $\mathcal{N}_i$, participate in the calculation, and α_{ij} is calculated as follows:

$$e_{ij} = \text{LeakyReLU}(w^T \cdot (v_i \oplus v_j)) \tag{2.11}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{l \in \mathcal{N}_i} \exp(e_{il})} \tag{2.12}$$

Here, $\oplus$ represents the concatenation operation, and $w \in \mathbb{R}^{2m}$ is a column vector of learnable parameters. The LeakyReLU function is a non-linear activation function.

Following the proposed scheme in Figure 2.1, each instance x of the temporal window produced after preprocessing is passed through two different graph attention layers:

1. **Feature-oriented Graph Attention Layer**: Each node of the graph consists of $v_i = \{x_{t,i} | t \in [0, n)\}$, representing a specific feature. Each edge of the graph represents the correlation between features. There are a total of d nodes, each of size n, so the output of the feature-oriented GAT is a matrix of size $d \times n$.

[1]The sigmoid function in Deep Learning is a function that introduces nonlinearity in the data and "normalizes" data inside the $[0, 1]$ interval. A common sigmoid function to use is the logistic function: $\sigma(x) = \frac{1}{1+e^{-x}}$

2. **Time-oriented Graph Attention Layer**: Each node of the graph consists of $v_i = \{x_{i,j} | j \in [0, d]\}$, representing a specific timestamp. Each edge of the graph represents the correlation between timestamps. This concept is similar to that of a Transformer Vaswani et al. [2023], where a similar attention concept is applied along the temporal dimension. There are n nodes of size d, so the output of the time-oriented GAT is a matrix of size $n \times d$.

Once passed through the two GAT layers, the output is concatenated with the input along the feature dimension, resulting in a matrix $x_{\text{GAT}} \in \mathbb{R}^{n \times 3d}$.

2.1.3 Gated Recurrent Unit

Following the two Graph Attention layers, the output x_{GAT} is passed to a Recurrent Neural Network (RNN). In particular, the model uses a Gated Recurrent Unit (GRU) Chung et al. [2014]. In detail, the main concept behind a GRU is its ability to accept sequential input, updating a *hidden state* h_t over time, preserving a "memory" of the sequence. Given a sequence of input $x = (x_1, x_2, ...x_T)$, the hidden state h_t is calculated as:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{2.13}$$

Here, $\odot$ denotes element-wise multiplication, z_t is called the *update gate*, and $\tilde{h}_t$ is the candidate hidden state. Basically the current hidden state is a convex combination of the last hidden state and a new candidate hidden state. In particular:

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \tag{2.14}$$

$$\tilde{h}_t = \tanh(W x_t + U(r_t \odot h_{t-1})) \tag{2.15}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{2.16}$$

Here, $\sigma(\cdot)$ and $\tanh(\cdot)$ are element-wise operations. All learnable parameters are assumed to have the appropriate dimensions to accept input elements $x_t \in \mathbb{R}^{3d}$ and return, after processing x_t for $t \in \{1, ...n\}$, a final hidden state $h_{\text{GRU}} \in \mathbb{R}^k$. Ideally, h_{GRU} contains the "memory" of the time window and act as sort of a embedding vector of the time series. This information is passed through the two final modules: the *Forecasting model* and *Recostruction Model*. They both take part in the calculation of the final loss.

2.1.4 Forecasting Model

The Forecasting model is the module that, given the input of the last hidden state of the GRU h_{GRU}, returns a value $\hat{x} \in \mathbb{R}^d$ representing the prediction for the timestamp succeeding the last timestamp in the input time window to the model $x = (x_1, \ldots, x_n)$.

The module consists of a stack of three fully connected layers with dimension d_2, as described in Figure 2.5.

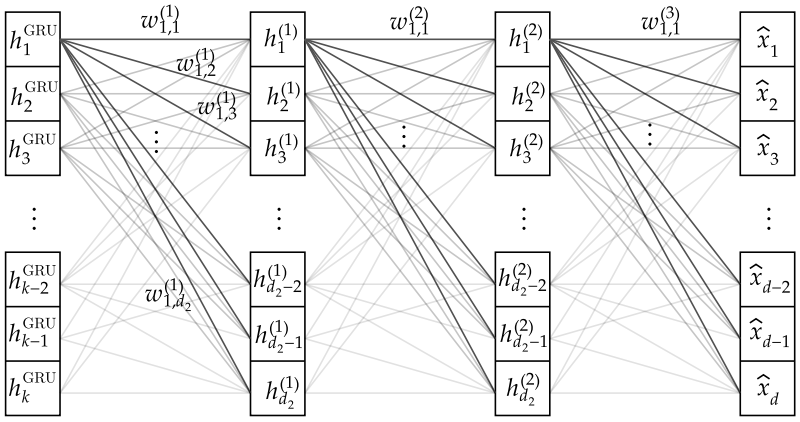


Figure 2.5: Descriptive diagram of the fully connected layer within the Forecasting Model

The output $\hat{x}$ of the module is then used for calculating the loss formulated as the Root Mean Square Error (RMSE) of the prediction with respect to the actual data:

$$L_{\text{for}} = \sqrt{(x_n - \hat{x})^T (x_n - \hat{x})} \tag{2.17}$$

2.1.5 Reconstruction Model

The goal of the Reconstruction Model is to learn a marginal distribution of the data through a latent representation z. The objective is, given a representation $z \in \mathbb{R}^{d_3}$ of x in a latent space, to compute $p_\theta(x|z)$. To achieve this, the method outlined in Kingma and Welling [2022] is employed the *Variational Autoencoder*, leveraging a probabilistically justified latent representation.

A simplified yet effective summary of a Variational Autoencoder is the following: a variational autoencoder can be defined as being an autoencoder whose training is regularised to avoid overfitting and ensure that the latent space has good properties that enable generative process.

Let's start by first defining what an autoencoder is. Given a data point $x \in \mathbb{R}^d$ we can define an **encoder** $e(\cdot) \in E$ as a function that returns a representation of x in a latent space $\mathbb{R}^k$ that is dimensionally smaller than $\mathbb{R}^d$. Hence an encoder is a function that performs a reduction in the dimensionality of the data. On the contrary, a **decoder** $d(\cdot) \in D$ is a function that, given an input from the latent space $z \in \mathbb{R}^k$ returns a point in the starting space $\mathbb{R}^d$. An **Autoencoder** is a encoder-decoder pair such that the loss of information is minimised:

$$(e^*, d^*) = \underset{(e,d)\in E\times D}{\arg\min}\ \epsilon(x, d(e(x))) \tag{2.18}$$

In classical machine learning, *Principal Component Analysis* is for all intents and purposes an autoencoder since its main objective is to find a suitable subspace $S \subset V$ such that the projection onto S from V minimize the loss of information. In a deep learning setting, the simplest form of autoencoder is a stack of two fully connected layers such that the dimension of the hidden layer is smaller than the starting dimension (figure 2.6):

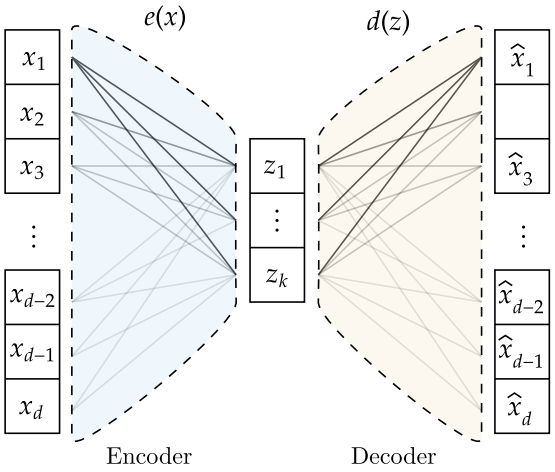


Figure 2.6: Descriptive diagram of a simple autoencoder

A simple autoencoder could easily be trained to minimize the reconstruction loss, such autoencoder would be very effective if the only purpose of our training

is to reduce the dimensionality of the data while retaining most of the information without any loss during the encoding-decoding pipeline. However this approach could easily lead to overfitting and, most importantly, to severe irregularity of the latent space preventing us from using it as a sample space for generative purposes. What would be preferable instead is a *regularized* latent space that is, to some extent, even interpretable.

For example imagine we are trying to fit an autoencoder on images of faces. It would be of great use if each face is encoded in two interpretable parameters (smile value and pose value) instead of two meaningless parameters. This is useful since once the autoencoder is trained we obtain both an interpretable encoding and a decoder that, given two interpretable parameters, is able to generate new data according to the given specifications.

This is solved in the variational autoencoder approach by encoding each input not as point in a latent space but instead as a *probability distribution* (as is in its parameters) from which we could then sample from and obtain a latent sample that can be then feeded through the decoder function.

In this particular setting the encoder returns the mean and the variance μ_x, σ_x of a multivariate normal distribution from which we then sample $z \sim \mathcal{N}(\mu_x, \sigma_x I)$ that we use as input of the decoding space:

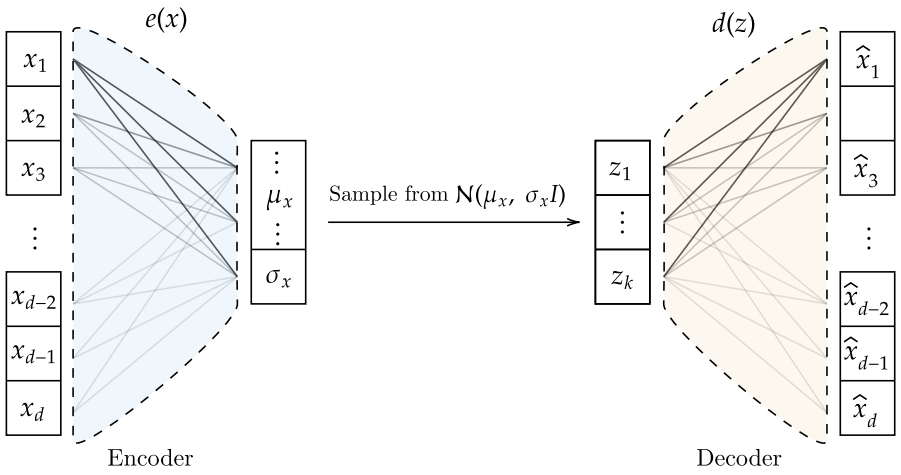


Figure 2.7: Descriptive (simplified) diagram of the Variational Autoencoder

To enforce the above mentioned regularity of the latent space, a regularization

term is used. In practice the regularisation is done by enforcing distribution to be close to a standar normal distribution. This way, we require the covariance matrices to be close to the identity, preventing punctual distributions, and the mean to be close to 0, preventing encoded distribution to be too far apart from each others.

We will now provide a mathematical framework to justify the use of the variational autoencoder and its implemantation in this work.

Given a sequence of input $x = \{x^{(i)}\}_{i=1}^{n}$, it is assumed to be a stochastic process involving an unobserved continuous random variable z (the latent variable). Specifically:

- $z^{(i)}$ is generated from $p_{\theta^*}(z)$, the prior distribution on the latent variable.
- $x^{(i)}$ is generated from $p_{\theta^*}(x|z)$, the likelihood of x given the observation of the latent variable z.

The aim is to maximize the likelihood of the data:

$$p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz. \tag{2.19}$$

The problem is that 2.19 is intractable because we can't reliably evaluate the integral for all possible values of z. Let's now show a workaround that will enable us to maximize 2.19.

Let $q_\theta(z|x)$ be the approximation of $p_\theta(z|x)$, i.e., the true posterior distribution of z. In terms of deep learning architecture, $q_\theta(z|x)$ is an Encoder: given x, it produces a distribution over possible values of z from which x could have been generated. On the other hand, $p_\theta(x|z)$ is a Decoder since, given the encoded z, it computes a distribution of x.

Let the marginal likelihood computed as:

$$\log p_\theta(x^{(1)}, x^{(2)} \dots x^{(n)}) = \sum_{i=1}^{N} \log p_\theta(x^{(i)}) \tag{2.20}$$

The main objective is to maximize 2.20. To do so we leverage the definition of the *Kullback-Leibler Divergence* Kullback and Leibler [1951].

Let:

$$D_{\mathrm{KL}}(q_\phi(z|x^{(i)})\|p_\theta(z|x^{(i)})) := \sum_z q_\phi(z|x^{(i)}) \log \frac{q_\phi(z|x^{(i)})}{p_\theta(z|x^{(i)})} = \tag{2.21}$$

$$\sum_z q_\phi(z|x^{(i)}) \left(\log \frac{q_\phi(z|x^{(i)})}{p_\theta(z, x^{(i)})} + \log p_\theta(x^{(i)}) \right) = \tag{2.22}$$

$$\sum_z q_\phi(z|x^{(i)})(\log q_\phi(z|x^{(i)}) - \log p_\theta(z, x^{(i)})) + \log p_\theta(x^{(i)}) \underbrace{\sum_z q_\phi(z|x^{(i)})}_{=1} \tag{2.23}$$

from which:

$$D_{\mathrm{KL}}(q_\phi(z|x^{(i)})\|p_\theta(z|x^{(i)})) = \underbrace{-\mathbb{E}_{q_\phi(z|x^{(i)})}[-\log q(z|x^{(i)}) + \log p_\theta(z, x^{(i)})]}_{\mathcal{L}(\theta,\phi;x^{(i)})} + \log p(x^{(i)}) \tag{2.24}$$

hence:

$$\log p_\theta(x^{(i)}) = D_{\mathrm{KL}}(q_\phi(z|x^{(i)})\|p_\theta(z|x^{(i)})) + \mathcal{L}(\theta, \phi; x^{(i)}). \tag{2.25}$$

Since $\log p_\theta(x^{(i)})$ does not depend on q, maximizing $\mathcal{L}(\theta, \phi; x^{(i)})$ is the same as minimizing D_{KL}. To effectively maximize $\mathcal{L}(\theta, \phi, x^{(i)})$ a *reparametrization trick* is used as follows: let $g_\phi(\cdot, \cdot)$ a differenciable transformation (e.g. MLP) then the latent variable is assumed to be a reparametrization of a noise variable ϵ:

$$\tilde{z} = g_\phi(\epsilon, x) \qquad \text{with} \qquad \epsilon \sim p(\epsilon). \tag{2.26}$$

If $p(\epsilon)$ is accurately choosen then, using Monte-Carlo estimation, we can compute:

$$\mathbb{E}_{q_\phi(z|x^{(i)})}[f(z)] = \mathbb{E}_{q_\phi(z|x^{(i)})}[f(g_\phi(\epsilon, x^{(i)}))] \simeq \frac{1}{L} \sum_{l=1}^{L} f(g_\phi(\epsilon^{(l)}, x^{(i)})) \tag{2.27}$$

with $\epsilon^{(l)} \sim p(\epsilon)$ and L is the number of samples used for Monte-Carlo estimation. Referring to 2.24 we obtain:

$$\tilde{\mathcal{L}}^A(\theta, \phi; x^{(i)}) = \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(x^{(i)}, z^{(i,l)}) - \log q_\phi(z^{(i,l)}|x^{(i)}) \tag{2.28}$$

where $z^{(i,l)} = g_\phi(\epsilon^{(l)}, x^{(i)})$. Hence if p_θ and q_ϕ are known distribuition then it's possible to calculate $\nabla_{\theta,\phi}\tilde{\mathcal{L}}^A(\theta, \phi; x^{(i)})$ in order to maximize $\tilde{\mathcal{L}}^A$.

In this particular case we use the *Variational Auto-Encoder* and set the prior

distribution of the latent variable as $p_\theta(z) = \mathcal{N}(z; 0, I)$. We also assume that $p_\theta(x|z)$ is a multivariate normal distribution whose parameters are the output of a MLP with z as input. Let:

$$\log q_\phi(z|x^{(i)}) = \log \mathcal{N}(z; \mu^{(i)}, \sigma^{2(i)} I) \tag{2.29}$$

where $\mu^{(i)}$ and $\sigma^{2(i)}$ are the output of a MLP with $x^{(i)}$ and ϕ as inputs. Then if we define:

$$z^{(i,l)} = g_\phi(x^{(i)}, \epsilon^{(l)}) = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)} \qquad \epsilon^{(l)} \sim \mathcal{N}(0, I) \tag{2.30}$$

we obtain:

$$\mathcal{L}(\theta, \phi; x^{(i)}) \simeq \frac{1}{2} \sum_{j=1}^{J} (1 + log(\sigma_j^{(i)})^2 - \mu_j^{(i)2} - \sigma_j^{(i)2}) + \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(x^{(i)}|z^{(i,l)}) \tag{2.31}$$

which is the loss of our reconstruction module used for training the model.

2.2 Temporal Convolution Networks (TCN)

In 2.1 we have seen how the concept of graph convolution can be applied to time series modeling both feature-wise and temporal-wise correlations. Although this method can be very effective to our means, in this section we will discuss how temporal convolutions can be added to the architecture to improve the overall performance of the anomaly detection algorithm.

Temporal convolutions for time series modeling first appeared in Bai et al. [2018] where, similarly to the autoencoder concept shown in 2.1.5, dilated convolutions are used to encode-decode the original time series in order to obtain a richer representation. The concept of dilated convolution is used for anomaly detection in Thill [2022] with good quality results. In this work dilated temporal convolutions will be implemented in the architecture seen in 2.1 trying to improve performance.

2.2.1 Classical Convolution

Convolution, in the machine learning context, operates as a transformative operation that captures spatial hierarchies and dependencies within data. Stemming from its roots in signal processing and linear algebra, convolution first appeared in machine learning application in Lecun et al. [1998] giving birth to Convolutional Neural Networks (CNNs), a branch of deep learning that saw an exponential growth in research due to the efficacy of convolution in image-based tasks.
To comprehend the essence of convolution in machine learning, it is imperative

to first grasp its mathematical underpinnings. Convolutional layers in neural networks leverage convolutional kernels, small filters that slide over input data to extract local patterns and features. The mathematical convolution operation between these filters and input data is pivotal in learning hierarchical representations, allowing the network to discern intricate structures and patterns in the data.

Definition 23 (Convolution in finite space)**.** *Let $x \in \mathbb{R}^T$ (input signal) and $h \in \mathbb{R}^k$ (convolution kernel) be two real-valued vectors. Then the* ***convolution operation*** *$(*)$ between x and h results in a vector (signal) y defined as follows:*

$$y_n = (x * h)_n = \sum_{i=0}^{k-1} h_i x_{n-i} \tag{2.32}$$

The output vector y has dimension $T - k + 1$, hence, to obtain an output vector that has the same dimension of the input vector, the *zero-padding* technique[2] is used accordingly. The convolution operation can be thought of as sliding a window of lenght k, which contains the filter weights h, over the input sequence x and computing a weighted average of x with the weights h in each time step.

Often times input signals such as images or multivariate time series are represented as multivariate signals: $x = (x_0, x_1, \dots x_{T-1})$ with $x_i \in \mathbb{R}^d$ where d is often referred to the number of *channels* of the signal. In this case the kernel h should also be multivariate with the same number of channels of the input signal.

Definition 24 (Multivariate convolution)**.** *Let $x = (x_0, x_1, \dots x_{T-1})$ with $x_i \in \mathbb{R}^d$ be the input signal and $h = (h_1, h_2, \dots h_k)$ with $h_i \in \mathbb{R}^d$ then the convolution operation results in a vector defined as:*

$$y_n = (x * h)_n = \sum_{i=0}^{k} h_i^T x_{n-i} \tag{2.33}$$

Multivariate convolution results in a single-channel output that can be seen as the sum of the convolutions across all input channels:

$$(x * h)_n = \sum_{i=0}^{k} h_i^T x_{n-i} = \sum_{i=0}^{k-1} \sum_{j=0}^{d} h_{i,j} x_{n-i,j} = \sum_{j=0}^{d} \sum_{i=0}^{k-1} h_{i,j} x_{n-i,j} = \sum_{j=0}^{d} (h_{:,j} * x_{:,j})_n \tag{2.34}$$

[2]The Zero Padding technique consists in adding zeros to the beginning and the end of the input vector in order to obtain an output vector of the desired dimension

2.2.2 Dilated convolutions

When dealing with convolutions of signals one trick we can use is to perform an *oversampling* of the convolution kernel h. Oversampling h by q is as simple as adding zeros between each element:

$$h^{(q)} = (\underbrace{h_0, 0, 0, \dots 0}_{q \text{ times}}, h_1, 0, 0 \dots 0, h_2, 0, 0, \dots, 0 \underbrace{h_k, 0, 0, \dots 0}_{q \text{ times}}). \tag{2.35}$$

The resulting convolution operation between the signal x and the oversampled kernel h results in:

$$(x * h^{(q)})_n = \sum_{j=0}^{d} \sum_{i=0}^{qk-1} h_{i,j}^{(q)} x_{n-i,j} = \sum_{j=0}^{d} \sum_{i=0}^{k-1} h_{i,j} x_{n-qi,j}. \tag{2.36}$$

Hence the following definition:

Definition 25 (Dilated Convolution)**.** *Let $x = (x_0, x_1, \dots x_{T-1})$ with $x_i \in \mathbb{R}^d$ be the input signal and $h = (h_1, h_2, \dots h_k)$ with $h_i \in \mathbb{R}^d$. Then the dilated convolution with dilation factor q can be defined as:*

$$y_n = (x *_q h)_n = \sum_{i=0}^{k-1} h_i^T x_{n-qi} \tag{2.37}$$

In practice, what we are doing in 2.37 is "skipping" elements of the input signal x during the convolution operation. In deep learning applications this allows the convolution operation to effectively capture information from a larger receptive field while using a smaller filter. This is particularly beneficial for tasks that require modeling long-range dependencies in the input data.

From a signal processing standpoint, oversampling the convolution kernel effectively "sharpens" its frequency response making it more sensitive to small changes in the frequency:

$$H^{(q)}(e^{-j\omega}) = \sum_{n=0}^{k-1} h_n^{(q)} e^{-j\omega n} = \sum_{n=0}^{k-1} h_n e^{-nq\omega} = H(e^{-jq\omega}). \tag{2.38}$$

2.2.3 Dilated Convolutional Layer and the TCN block

In Deep learning application a convolutional layer is typically comprised of many discrete filters, and the individual outputs are stacked into a so-called feature map. If a signal x of length T is passed through a convolutional layer with n_{filters} filters, the resulting feature map has the dimension $T \times n_{\text{filters}}$. The weights h of

each filter are learnable parameters, trained using the back-propagation algorithm.

The main idea behind the Temporal Convolutional Network (TCN) block is to build a stack of dilated convolutional layers, where the dilation rate increases with each added layer. A common choice is to start with a dilation rate of $q = 1$ for the first layer of the network and double q with every new layer. This way the receptive field of the model increases exponencially, as shown in figure 2.8.

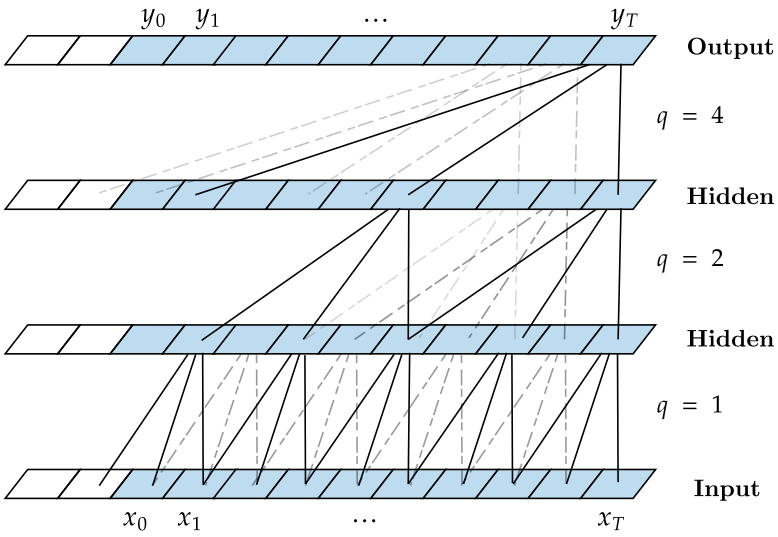


Figure 2.8: A simplified view of a dilated convolution layer

More implementation details are in Bai et al. [2018], for our purpose we consider a TCN block as a block that takes as input a signal x of dimensions $n \times T$ and outputs a signal with the same dimensions after performing a bunch of dilated convolutions. The idea behind this particular usage of the TCN block is that using dilated convolutions we can obtain a more informative representation of the time series given in input. The intuition behind this is the fact that each dimension of the time series has to be processed with different level of detail since often, in real data, signals among dimensions can have very different spectral characteristics. Hence, with the use of dilated temporal convolution we obtain a cohesive and uniform representation across all features.

The TCN block has been implemented in our architecture both in input and output of the model as shown in figure 2.9

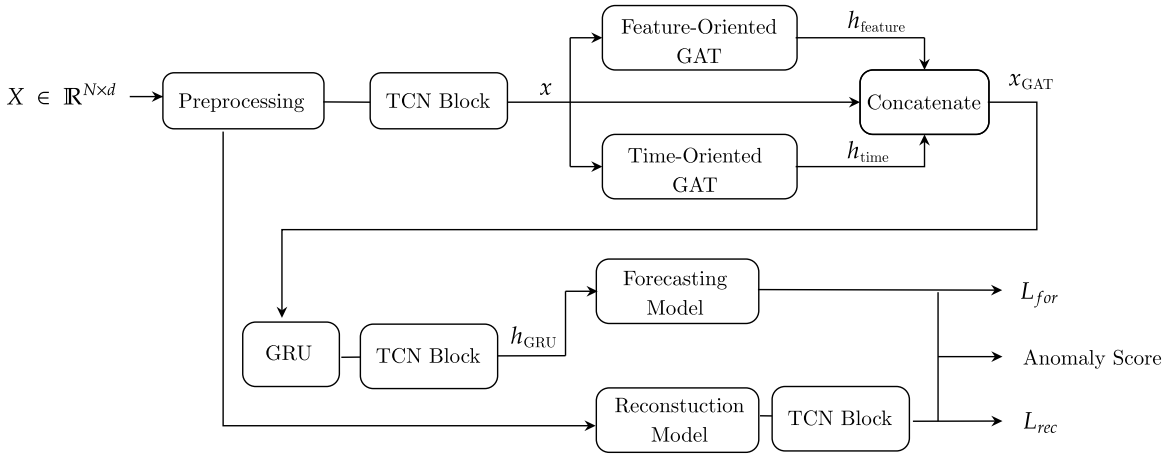


Figure 2.9: Descriptive diagram of the final MTAD-GAT architecture

Chapter 3

Experimental methodology

In this chapter we will explore the implementation aspects of this work. We will give a brief overview of the datasets used for training and explore the technical details behind the implementation of the methods shown in the previous chapter. Code snippets will be presented in this chapter but they have to be considered as a *proof of concept* because the code presented has been trimmed in order to improve readability. Finally, experimental results will be presented evaluating the performance of the methods used and possible use cases that can be deployed.

3.1 Datasets used

Experimenting with anomaly detection algorithms exposes one key problem: in order to evaluate the performance of any given algorithm, data must be "labeled" meaning that, if we are dealing with time series, each timestamp must be labeled as anomalous or not in order to check if the model performed well.

The most important thing to keep in mind however is that labeled data is only needed during the evaluation of the model and not during its training. This is because training is conducted in an "unsupervised" manner: loss in the models studied in this thesis is never computed using labels, instead only using intrinsic information from within the model as seen in the previous chapter.

Time series data that is both good quality and labeled is very difficult to obtain: data must be labeled in a supervised manner meaning that some timestamps have to be labeled as "anomalies" which is inherently difficult since anomalies are events that one does rarely see.

Hence most of the available datasets used for Time Series Anomaly Detection

are synthetic or are gathered from sensors that monitor a system that is artificially attacked to generate anomalies.

In the following sections we will introduce all the datasets used for our analysis.

3.1.1 The SWaT Dataset

The Secure Water Treatment (SWaT) is a water treatment testbed for research in cybersecurity. SWaT consists of a modern six-stage process. The process begins by taking in raw water, adding necessary chemicals to it, filtering it via an Ultrafiltration (UF) system, de-chlorinating it using UV lamps, and then feeding it to a Reverse Osmosis (RO) system. A picture of the actual water treatment testbed is depicted in figure 3.1 A multivariate time series dataset is obtained from the

Figure 3.1: A picture of the SWaT testbed

testbed by collecting data from the various sensors that monitor all the different components of the dataset. In table 3.1 we list the first 10 features of the sensor, for more information and the full list of the 51 sensors refer to Goh et al. [2017] Data has been sampled with a frequency of one second for 11 days of continuous operation. For the first 7 days the system is monitored under normal conditions and during the remaining 4 days the testbed a series of cyber attacks have been conducted to the testbed that ultimately resulted in anomalies in sensor data. Data is provided with a list of all attacks and is conveniently labeled. After some preprocessing data is feeded to the model as a matrix $X \in \mathbb{R}^{T \times N}$ where T is the number of timestamps and N is the number of features. Labels are represented

No.	Name	Type	Description
1	FIT-101	Sensor	Flow meter; Measures inflow into raw water tank
2	LIT-101	Sensor	Level Transmitter; Raw water tank level
3	MV-101	Actuator	Motorized valve; Controls water flow to the raw water tank
4	P-101	Actuator	Pump; Pumps water from raw water tank to second stage
5	P-102 (backup)	Actuator	Pump; Pumps water from raw water tank to second stage
6	AIT-201	Sensor	Conductivity analyser; Measures NaCl level
7	AIT-202	Sensor	pH analyser; Measures HCl level
8	AIT-203	Sensor	ORP analyser; Measures NaOCl level
9	FIT-201	Sensor	Flow Transmitter; Control dosing pumps
10	MV-201	Actuator	Motorized valve; Controls water flow to the UF feed water tank

Table 3.1: List of the first 10 features of the SWaT Dataset

as a binary vector $y \in \mathbb{R}^T$ where 0 corresponds to normal behavior while 1 corresponds to anomalous behavior.

A graphical example of the SWaT dataset is depicted in figure 3.2.

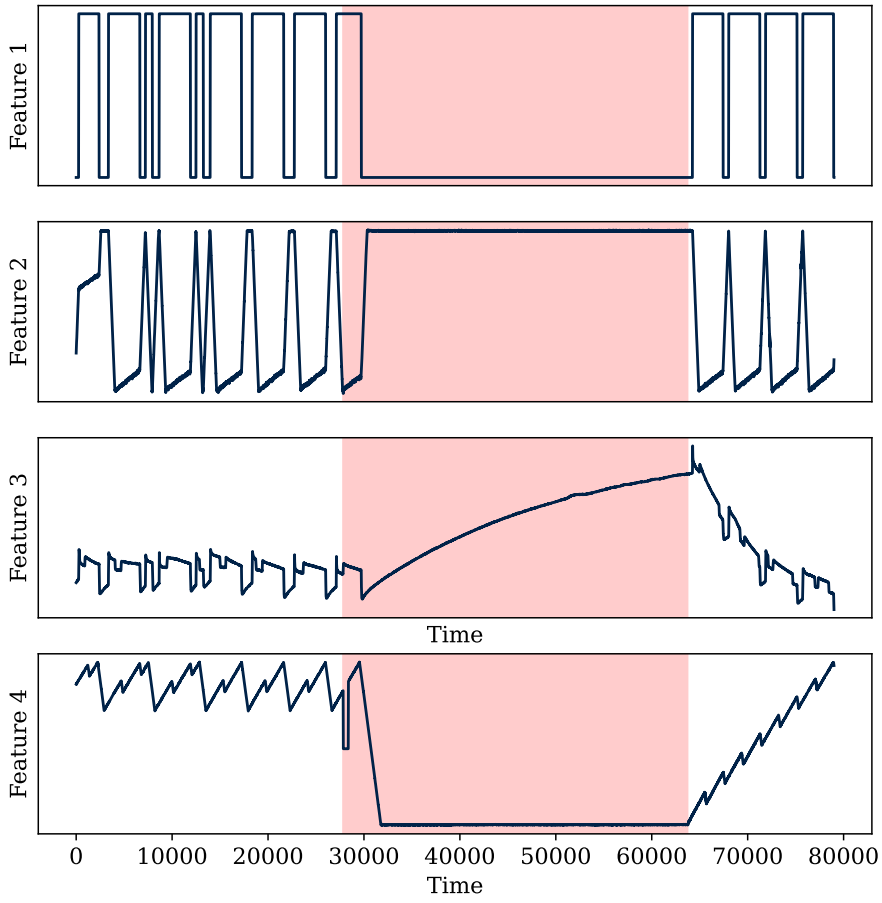


Figure 3.2: A portion of the SWaT where a labeled anomaly is depicted as the portion in red

3.1.2 The WADI Dataset

WADI is a natural extension of SWaT, comprising two elevated reservoir tanks, six consumer tanks, two raw water tanks and a return tank. It also comes equipped with chemical dosing systems, booster pumps and valves, instrumentation and analysers. WADI takes in a portion of SWaT's reverse osmosis permeate and raw water, thus forming a complete and realistic water treatment, storage and distribution network.

WADI has the capabilities to simulate the effects of physical attacks such as water leakage and malicious chemical injections. Together with SWaT, WADI provides opportunities for researchers to work on a full spectrum of possible cyber and physical attacks on a water treatment and distribution plant.

WADI has more features than the SWaT dataset ($\sim$ 130 features) and is composed of 14 days of continuos normal operation and 2 days where attacks were conducted.

Figure 3.3 shows a portion of the dataset with an highlited anomaly in red.

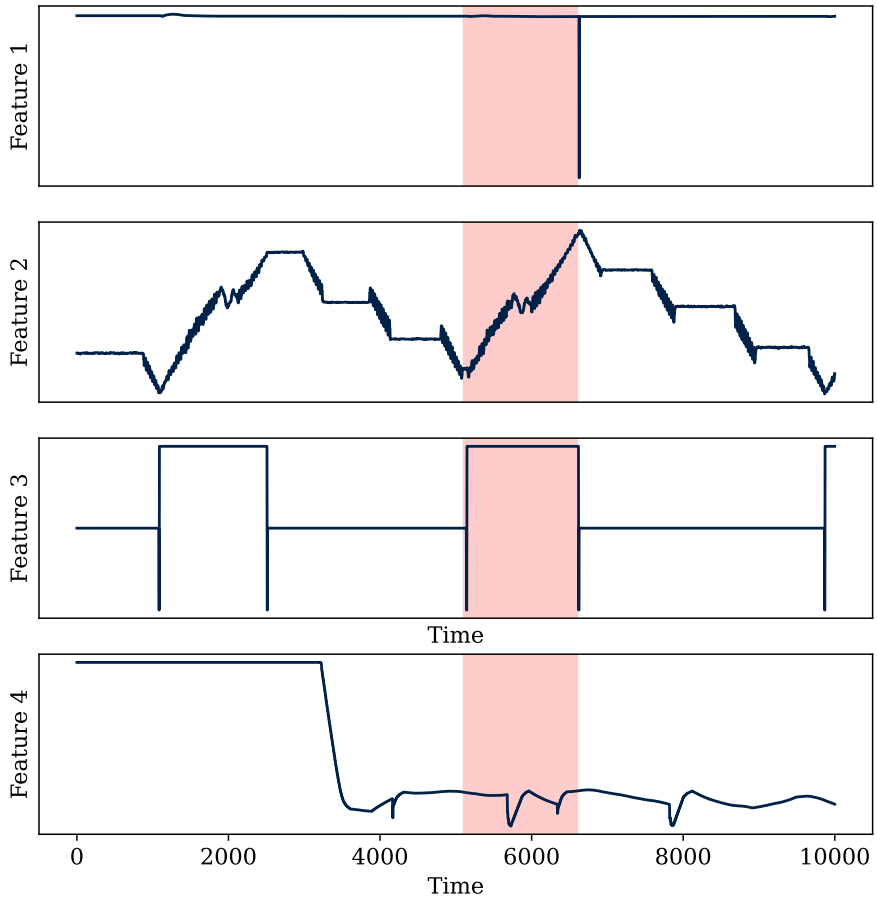


Figure 3.3: A portion of the WADI where a labeled anomaly is depicted as the portion in red

As is the case with the SWaT dataset, WADI dataset comes with a list of attacks that have been conducted on the system and each timestamp is labeled (anomaly or not).

3.1.3 Human Activity Dataset

The Smartphone-Based Recognition of Human Activities and Postural Transitions DataSet (ACT in short) Reyes-Ortiz et al. [2014] is a collection of signal recordings from smartphone sensors. The dataset contains readings from various sensors and each timestamp is labeled with the corresponding position (e.g. sitting position, standing position).

The experiments were carried out with a group of 30 volunteers within an age bracket of 19-48 years. They performed a protocol of activities composed of six basic activities: three static postures (standing, sitting, lying) and three dynamic activities (walking, walking downstairs and walking upstairs). The experiment also included postural transitions that occurred between the static postures. These are: stand-to-sit, sit-to-stand, sit-to-lie, lie-to-sit, stand-to-lie, and lie-to-stand. All the participants were wearing a smartphone (Samsung Galaxy S II) on the waist during the experiment execution.

The sensor signals (accelerometer and gyroscope) were pre-processed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/window). The sensor acceleration signal, which has gravitational and body motion components, was separated using a Butterworth low-pass filter into body acceleration and gravity. The gravitational force is assumed to have only low frequency components, therefore a filter with 0.3 Hz cutoff frequency was used. From each window, a vector of 561 features was obtained by calculating variables from the time and frequency domain.

An anomaly detection algorithm can be deployed in this context to detect the transitions from one position to another. The transitioning labels (sit-to-stand, stand-to-sit...) can be labeled as anomalies and the algorithm can be evaluated on its ability to detect transitions.

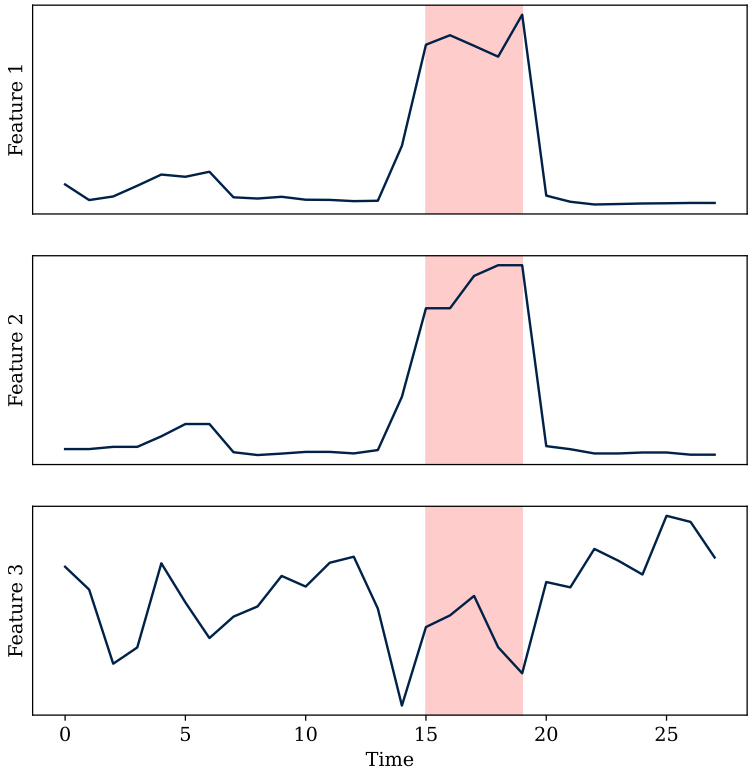


Figure 3.4: A portion of the ACT dataset where a labeled anomaly is depicted as the portion in red

3.1.4 Metro Dataset

The MetroPT Dataset Veloso et al. [2022] is the result of a Predictive Maintenance project with the metro transportation service in Porto, Portugal. The objective of the dataset is to help build anomaly detection methods for failure prediction.

The dataset is composed of 15 features collected from a system of sensors installed on an operating train. It contains data from January to June 2022 and the train performs on average 26 trips per day.

Failures are manually labeled, an example of a labeled failure on the system is depicted in Figure 3.5.

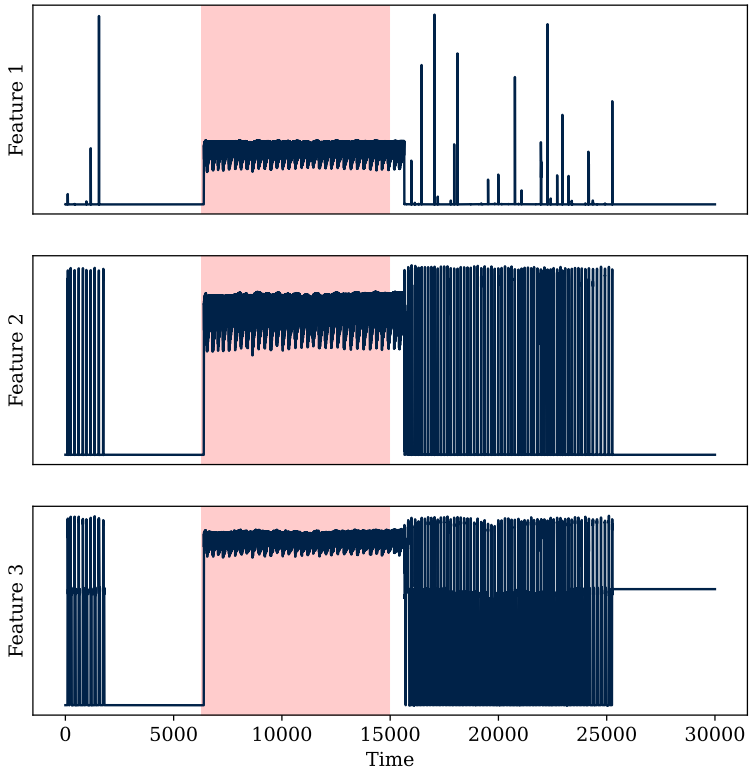


Figure 3.5: A portion of the METRO dataset where a labeled anomaly is depicted as the portion in red

3.1.5 Automotive Dataset

During the development of this thesis it has been possible to experiment with data from a reputable automotive company. Two datasets were provided to experiment with. The first is composed of aggregated data from 7 vehicles gathered every 15 minutes while the second is composed of data sampled every second.

Nature of the data is unsupervised, meaning that no labels were provided with this data and the aim of the analysis is to gain insights about faults on vehicles that have been claimed during operations.

The full datasets are composed of many features > 1000 but, due both to computational feasibility and interpretability, only a subset of feature has been selected

for the analysis.

3.2 Technical implementation, the PyTorch library

The implementation of the deep learning algorithms seen in 2.1 has been carried out in Python code, leveraging the famous PyTorch library when necessary.

In order to easily manage file sizes of $\sim 500Mb$ per dataset the preprocessing is executed *offline* meaning that a script `preprocessing.py` has been created to take care of all the preprocessing procedures specific for each dataset. The preprocessed datasets are then stored in a `.pkl` file and later used for training.

During preprocessing we've carried out experiments regarding dataset size and sample rate. All datasets used have been sampled with a frequency of $1s$ hence in order to reduce the dimension of the dataset and ease the computation on some datasets we have tried *downsampling* the data picking observations with a frequency of $2s$ or $3s$. This drastically reduces the dimension of the dataset and in some cases lead to the same accuracy results. Moreover, data that has a strong periodicity can be truncated without loss of accuracy.

Preprocessing also accounted for scaling the datasets as seen in 2.1.1 and handling missing values. Missing values handling varies from dataset to dataset and will be explored in the following section. Optionally the dataset can be cleaned of spikes and simple univariate outliers using the spectral residual algorithm seen in 2.1.1.

3.2.1 The Optimization algorithm: ADAM

Training has been carried out using the typical framework of deep learning models. Each weight in the model is considered as a trainable parameter so let θ be the vector containing all the model parameters. The objective of the training is tying to minimize a loss function $f(\theta)$ by adjusting the parameters accordingly.

The most common algorithm used to achieve the minimization of the loss function is the ADAM algorithm Kingma and Ba [2017].

Evaluating the loss function $f(\theta)$ of the model on a given dataset would mean

to evaluate the model on all the datapoints of such dataset. This would mean to perform a *forward pass* of all the data for each evaluation and most of the times this procedure is computationally unfeasible. Hence we assume that $f(\theta)$ is a random variable and its realizations $f_1(\theta), f_2(\theta), \dots f_T(\theta)$ are computed by evaluating the loss only on a small portion of the dataset called *batch*. The goal of the algorithm then becomes to minimize the expected value of such random variable $\mathbb{E}[f(\theta)]$. Assuming that $f(\theta)$ is differentiable w.r.t. θ then $g_t = \nabla_\theta f_t(\theta)$. The most simple form of a stochastic descent algorithm would be the following: given an initialized parameter vector θ_0, then the update rule for θ_t at the t-th training step would be the following:

$$\theta_t = \theta_{t-1} - \eta \nabla_\theta f_t(\theta_{t-1}) \tag{3.1}$$

Where η represents the *stepsize* of the update rule. If $f(\cdot)$ is a convex function then the update rule 3.1 guarantees the convergence of θ to a value that minimizes $f(\cdot)$.

The ADAM algorithm extends the idea of the simple stochastic gradient descent update rule by updating the exponential moving averages of the gradient (m_t) and the squared gradient (v_t) where hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages. From this concept the name ADAM: Adaptive Moment estimation.

A pseudo-code version of the ADAM algorithm is depicted in algorithm 2. Proof of convergence and implementation details are out of the scope of this thesis and can be found in Kingma and Ba [2017].

The code snippet 3.1 shows how training is performed using the PyTorch library. The function `fit` has a `torch.model` defined inside and cycles through epochs. In the context of deep learning, an epoch refers to one complete pass through the entire training dataset during the training phase of a neural network. In other words, it's a single iteration over the entire dataset to update the weights of the neural network.

During an epoch batches of sliding windows are extracted from the dataset using a `Data Loader` and for each batch the losses are computed. Then a `backward` propagation is computed and an optimization step is performed, as is the case with every deep learning model.

Algorithm 2 The ADAM algorithm.

Require: α ▷ Step size
Require: $\beta_1, \beta_2 \in [0, 1)$ ▷ Exponential decay rates for the moment estimates
Require: $f(\theta)$ ▷ Stochastic objective function with parameters θ
Require: θ_0 ▷ Initial parameter vector

$m_0 \leftarrow 0$
$v_0 \leftarrow 0$
$t \leftarrow 0$
while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ ▷ Get gradients w.r.t. stochastic objective at timestep t
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ ▷ Update biased first moment estimate
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ ▷ Update biased second raw moment estimate
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ ▷ Compute bias-corrected first moment estimate
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ ▷ Compute bias-corrected second raw moment estimate
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ ▷ Update Parameters
end while
return θ_t

```
def fit(self, train_loader, val_loader=None):

        for epoch in range(self.n_epochs):
            self.model.train()
            forecast_b_losses = []
            recon_b_losses = []

            for x, y in train_loader:#for each sliding window
                x = x.to(self.device)
                y = y.to(self.device)
                self.optimizer.zero_grad()

                preds, recons, mean, log_var = self.model(x) #
    outputs the predictions and the reconstructions

                forecast_loss = torch.sqrt(self.forecast_criterion
    (y, preds)) #loss from prediction
                recon_loss = torch.sqrt(self.recon_criterion(x,
    recons)) #loss from reconstruction
                loss = forecast_loss + recon_loss

                loss.backward() #computes the backward step
                self.optimizer.step() #compute the optimizer
    update step

                forecast_b_losses.append(forecast_loss.item())
                recon_b_losses.append(recon_loss.item())

            # Evaluate on validation set
            forecast_val_loss, recon_val_loss, total_val_loss = "
    NA", "NA", "NA"
            if val_loader is not None:
                forecast_val_loss, recon_val_loss, total_val_loss
    = self.evaluate(val_loader)
                self.losses["val_forecast"].append(
    forecast_val_loss)
                self.losses["val_recon"].append(recon_val_loss)
                self.losses["val_total"].append(total_val_loss)

                if total_val_loss <= self.losses["val_total"][-1]:
                    self.save(f"model.pt")
        if val_loader is None:
            self.save(f"model.pt")
```

Listing 3.1: Training Framework of the model

3.3 Threshold setting methods

As discussed in the previous chapters, the output of anomaly detection algorithms is mainly an anomaly score for each timestamp encountered. This means that the focus is not on directly classifying whether a sampled record is an anomaly or not, instead providing a score that assesses how much the said sample is anomalous with respect to the ones used in training data. This is why an important anomaly measure used for evaluating our methods is the AUC-ROC score seen in 1.2.1: it decouples the classification task from the anomaly measure, providing a score that is not dependent on the threshold setting.

While this is extremely convenient from a method evaluation standpoint, a threshold on the anomaly score is still needed to deploy the model in a real setting: a limit on the anomaly score must be set to decide when to signal that the system monitored is showing anomalous behavior.

To this end, various methods can be deployed. The following sections will include a brief overview of the methods studied in this thesis.

3.3.1 Peaks-Over-Threshold (POT) approach

The Peaks-over-Threshold Siffer et al. [2017] approach for finding a threshold to use in the anomaly score evaluations leverages Extreme Value Theory. For the full details refer to Siffer et al. [2017].

Let X be a random variable, we denote as z_q its quantile at level $1-q$ meaning that z_q is the smallest value s.t. $\mathbb{P}(X \leq z_q) \geq 1-q$. An important result from extreme value theory comes from Balkema and de Haan [1974], and it states that, under general assumptions, there exist γ, $\sigma \in \mathbb{R}$ such that:

$$\bar{F}_t(x) = \mathbb{P}(X - t > x | X > t) \sim_{t \to \tau} \left(1 + \frac{\gamma}{\sigma} x\right)^{-\frac{1}{\gamma}} \tag{3.2}$$

Where t is a set threshold, τ is the upper limit of the cumulative density function $F(x)$. In 3.2 $\bar{F}_t(x)$ is the probability of X exceeding t by x given that X is already exceeding t. The aim of the method is to get estimates $\hat{\gamma}$ and $\hat{\sigma}$ since from 3.2 it can be computed:

$$z_q \simeq t + \frac{\hat{\sigma}}{\hat{\gamma}} \left(\left(\frac{qn}{N_t} \right)^{-\hat{\gamma}} - 1 \right) \tag{3.3}$$

where q is the desired probability, n the total number of observation and N_t is the number of *peaks* i.e. the number of X_i s.t. $X_i > t$.

Estimates are computed through the maximum likelihood estimation of 3.2, having to maximize:

$$\log \mathcal{L}(\gamma, \sigma) = -N_t \log \sigma - \left(1 + \frac{1}{\gamma}\right) \sum_{i=1}^{N_t} \log \left(1 + \frac{\gamma}{\sigma} Y_i\right) \tag{3.4}$$

Where $Y_i > 0$ are the excesses of X_i over t. The optimization is done numerically.

The key takeaway of this method is the fact that we are able to compute a quantile z_q of the observed random variable X **without making assumption on its distribution**, and this is possible thanks to Extreme value theory.

Operationally the quantile value z_q is initialized with the first n observation of the data. Then, for each exceedance of X over t, $\hat{\gamma}$ and $\hat{\sigma}$ can be adjusted accordingly as depicted in algorithm 3:

Algorithm 3 The SPOT algorithm (Streaming POT).

Require: $(X_i)_{i>0}, n \in \mathbb{N}, q \in (0, 1)$

$\mathbf{A} \leftarrow \emptyset$ ▷ The set of anomalies
$t \leftarrow$ `SetInitialThreshold`$(X_1, \dots X_n)$
$\hat{\gamma}, \hat{\sigma} \leftarrow$ `MLE`$(X_1, ...X_n)$ ▷ Maximum likelihood estimation
$z_q \leftarrow$ `ComputeQuantile`$(q, n, N_t, t, \hat{\gamma}, \hat{\sigma})$
$k \leftarrow n$
for $i > n$ **do**
    if $X_i > z_q$ **then**
        Add (i, X_i) in $\mathbf{A}$
    else if $X_i > t$ **then**
        $N_t \leftarrow N_t + 1$
        $k \leftarrow k + 1$
        $\hat{\gamma}, \hat{\sigma} \leftarrow$ `MLE`$(X_1, ...X_{i+n})$
        $z_q \leftarrow$ `ComputeQuantile`$(q, n, N_t, t, \hat{\gamma}, \hat{\sigma})$
    else
        $k \leftarrow k + 1$
    end if
end for

The implementation of this algorithm sets the initial threshold t as an *High enough quantile* meaning an empirical quantile with a q level that is lower than the POT q argument.

3.3.2 Epsilon approach

The Epsilon method Hundman et al. [2018] is an alternative approach to finding the best threshold for the anomaly score. Let $\hat{y}^{(t)}$ be the prediction/reconstruction output of the anomaly detection algorithm, let $e^{(t)}$ be the corresponding error (evaluated in l^2 norm in this thesis). Each $e^{(t)}$ is appended to a vector of errors $\mathbf{e}$ that is smoothed using exponentially-weighted average ($\mathbf{e_s}$). The problem of finding the best threshold ϵ can be formulated as the following optimization problem:

$$\max_{z \in [l_z, u_z]} \quad \frac{\frac{\Delta\mu(\mathbf{e}_s)}{\mu(\mathbf{e}_s)} + \frac{\Delta\sigma(\mathbf{e}_s)}{\sigma(\mathbf{e}_s)}}{|\mathbf{e}_a| + |\mathbf{E}_{\text{seq}}|^2} \tag{3.5}$$

$$\text{s.t.} \quad \epsilon = \mu(\mathbf{e}_s) + z\sigma(\mathbf{e}_s) \tag{3.6}$$

$$\Delta\mu(\mathbf{e}_s) = \mu(\mathbf{e}_s) - \mu(\{e_s \in \mathbf{e_s} | e_s < \epsilon\}) \tag{3.7}$$

$$\Delta\sigma(\mathbf{e}_s) = \sigma(\mathbf{e}_s) - \sigma(\{e_s \in \mathbf{e_s} | e_s < \epsilon\}) \tag{3.8}$$

$$\mathbf{e_a} = \{e_s \in \mathbf{e}_s | e_s > \epsilon\} \tag{3.9}$$

$$\mathbf{E}_{\text{seq}} = \text{continuous sequences of } e_a \in \mathbf{e_a} \tag{3.10}$$

In much simpler terms, the problem 3.5 aims to find a threshold ϵ defined as 3.6 that, if all values above are removed, would cause the greatest decrease in the mean 3.7 and the standard deviation 3.8 of the smoothed errors $\mathbf{e_s}$. We also want to penalize for having larger number of anomalous values 3.9 and sequences 3.10 to prevent overly greedy behavior.

Since values of z ranging from 2 to 10 were found to be optimal for most of the use cases, the optimization can be easily brute-forced without requiring any particular optimization solving strategy.

3.3.3 Best-F_1 search approach

The Best-F_1 score search approach is a much more naive method compared to the others. This method is a brute force approach for finding a threshold that produces the best F_1 result in the test dataset. Hence in can only be used in a supervised context and it's a valuable metric since it shows the capability of the anomaly detection algorithm when the optimal threshold is computed.

3.4 Experimental results

Experimental results will be evaluated in this section. As previously discussed, the result of the anomaly detection pipeline is an anomaly score for each encountered

timestamp. Anomalies are typically classified by inferring them from the anomaly score and this is necessary when deploying the anomaly detector. For our evaluation purposes we will try to decouple the threshold selection problem from the model evaluation since it can be very application dependant.

Our method produces a loss that is the combination of the loss from the reconstruction model and the forecast model. Such loss is computed on a per-feature basis and can be used as an anomaly score. At our disposal we both have the "global anomaly score" (the mean of the anomaly score from each feature) and the anomaly score from each feature. Threshold setting algorithms previously seen work on the global anomaly score but it is also interesting to analyze individual anomaly score for each feature.

3.4.1 PCA analysis of the anomaly score

At our disposal for analysis from the anomaly detection algorithm we have an anomaly matrix $A \in \mathbb{R}^{N_t \times N_f}$ where N_t is the length of the test multivariate time-series and N_f is the number of features of the time-series. Hence $A_{i,j}$ is the anomaly score of the i-th timestamp for the j-th feature. From the anomaly matrix A we can compute the global anomaly vector $\mathbf{a}$:

$$\mathbf{a} \in \mathbb{R}^{N_t} \qquad \mathbf{a}_i = \sum_{j=0}^{N_f} A_{i,j} \quad \text{for } i \in [0, 1, ...N_t]. \tag{3.11}$$

Evaluating the mean of the anomaly score from each feature is the standard procedure for anomaly detection algorithms. The problem with the mean along the features is the fact that some information can be lost when the number of features is high enough, even if the aim of the algorithm is to condense all the information in one anomaly score measure.

What has been observed through experimentation in this thesis is that applying a feature reduction algorithm to the anomaly score matrix and analyzing the reduced set of features can give some insights about anomalous behavior of data. In particular Principal Component Analysis has been applied to the anomaly matrix to reduce the number of anomaly features to ~ 5 and a moving average smoothing has been applied in order to point out spikes in the anomaly scores from various features.

Moreover, Principal Component Analysis can be used to infer the loadings of the

original features from each principal component, providing us with information about what features concurred to spikes in anomaly scores.

Observation 11. PCA stands for *Principal Component Analysis* and is a technique used to reduce the dimensionality of the data (number of columns) while retaining as much information as possible.

Let $X \in \mathbb{R}^{n \times p}$ be our feature matrix and let

$$\Sigma = \frac{1}{N-1} \bar{X}^T \bar{X} \in \mathbb{R}^{p \times p} \tag{3.12}$$

Be the *Covariance matrix* of X, where $\bar{X}$ is the *centered* data matrix i.e. $\bar{X} = [x_1 - \bar{x_1}, \dots x_p - \bar{x_p}]$ (all the columns of X are centered w.r.t. their sample mean).
Since the covariance matrix Σ of our centered data is a symmetric semi-definite positive matrix, we can compute its **eigendecomposition**:

$$\Sigma = V \Lambda V^T \tag{3.13}$$

with:

$$V = [v_1, v_2, \dots, v_p] \in \mathbb{R}^{p \times p}, \qquad \Lambda = \begin{bmatrix} \lambda_1 & \dots & 0 \\ \vdots & \ddots & 0 \\ 0 & \dots & \lambda_n \end{bmatrix}, \qquad \lambda_1 \geq \lambda_2 \geq \dots \geq 0. \tag{3.14}$$

Then it can be shown that **the columns of V describe the directions of greater variance of data**, depending on the eigenvalues magnitude. This means that V is a new basis for $\mathbb{R}^p$ and can be seen as a **rotation matrix**. Let $x^{(i)} \in \mathbb{R}^p$, then $y^{(i)} = V^T x^{(i)}$ is the same vector but represented with basis V. By this token we could compute the rotation of each point in our original data X obtaining the matrix $Y = \bar{X}V \in \mathbb{R}^{n \times p}$. The resulting covariance matrix Σ_Y would be:

$$\Sigma_Y = \frac{1}{N-1} Y^T Y = V^T \bar{X}^T \bar{X} V = V^T \Sigma V = \Lambda \tag{3.15}$$

This means that data written w.r.t. the new basis V has a diagonal covariance matrix Λ and the variance of Y w.r.t. the axis v_i is λ_i. We call $v_1, \dots, v_p$ **principal components** and λ_i is the **Explained variance** of the principal component v_i.

Since eigenvalues are computed in descreasing order, the first components of the data written in the basis V are the one with the most explained variance. Hence, truncating the rotated data matrix to the first k columns (Y_k), causes a loss of explained variance that is quantified by the remaining $\lambda_{k+1} \dots \lambda_p$ eigenvalues. The **percentage of explained variance** by the first k components is hence computed as:

$$\texttt{ev\_ratio} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{p} \lambda_i}. \tag{3.16}$$

3.4.2 Results on supervised datasets

Quantitative and qualitative results will be shown in this section for each of the supervised datasets. Supervised datasets are the ones that are provided with timestamps labels that indicates whether a timestamp is anomalous or not hence providing a way to objectively measure the performance of an algorithm.

The SWAT dataset

The SWAT dataset, when provided by the authors, is already split in train set and test set. The train set is composed of $\sim 500K$ observation of 51 features while the test set is composed of $\sim 450K$ observations with the same number of features. Data is sampled with a $1s$ frequency and empirical results suggest that performing a downsampling of one every 10 observations not only left the results unchanged but sped the training process (as expected) by 10 times. Results for the SWAT dataset are the following:

F1 score	Precision	Recall	AUC-ROC	Dim. Red.	TCN
0.91	0.97	0.85	0.90	Yes	Yes
0.89	0.99	0.81	0.91	No	Yes
0.89	0.95	0.84	0.89	No	No

Table 3.2: Experimental Results for the SWAT dataset.

The WADI dataset

Like the SWAT dataset, the WADI dataset is provided already split in training set and test set from the authors. The length of the time series provided are comparable to the ones seen for the SWAT dataset. Unlike the SWAT dataset,

the number of features is much higher. We do have in fact 123 features, hence the dataset is much bigger than SWAT. A downsampling has been performed to keep dimensions under control.

F1 score	Precision	Recall	AUC-ROC	Dim. Red.	TCN
0.83	0.80	0.90	0.73	Yes	Yes
0.86	0.74	0.99	0.75	Yes	No
-	-	-	-	No	No

Table 3.3: Experimental Results for the Wadi dataset.

Results in table 3.3 are not shown for the run with no dimensionality reduction since the high dimensionality of the dataset requires some sort of dimensionality reduction to complete training in a reasonable time.

The METRO dataset

The metro dataset is provided with a range of dates during which attacks were conducted on the system. The dataset has been split such that all attacks are featured in the test dataset resulting in a 60/40 split (approximately). In this case the number of features is much smaller than the other datasets with 15 features. A downsampling of one every 5 observation has been performed to keep the length of the time series under control ($\sim 170K$).

F1 score	Precision	Recall	AUC-ROC	Dim. Red.	TCN
0.77	0.70	0.86	0.86	Yes	Yes
0.87	0.77	0.99	0.93	No	Yes
0.56	0.38	0.99	0.81	No	No

Table 3.4: Experimental Results for the METRO dataset.

The ACT dataset

The ACT dataset is provided with the greatest number of features among all datasets exhamined. In particular 561 are available for this dataset. Training the network on the full 561 features is prohibitive both from a training time and memory point of view. Hence depiceted results are shown with the dimensionality reduction in place.

F1 score	Precision	Recall	AUC-ROC	Dim. Red.	TCN
0.91	0.85	0.99	0.99	Yes	Yes
0.91	0.86	0.98	0.99	Yes	No
-	-	-	-	No	No

Table 3.5: Experimental Results for the ACT dataset.

3.4.3 Automotive Dataset

As presented in 3.1.5 two datasets for the analysis were provided by the company. The first dataset is composed of data sampled every 15 minutes. For this reason the length of the time series is limited ($\sim$ 1600 samples) and it is very difficult to find correlations along the time axis since aggregation techniques have been performed. This led to inconclusive results since the model is not able to fit on such high level aggregated data.

On the contrary, the second dataset is composed of readings from vehicle sensors sampled at a frequency of one second. This is more than enough to properly fit the model and train a network that is able to recognize patterns across multivariate time series data. Since data is unlabeled it's not possible to objectively score a performance of the model. Moreover, the dataset has been provided with the aim to gain insights about some specific mechanical parts of some vehicles and this can only be assessed by qualitatively analyze data with the aid of skilled technicians.

What can be shown is how well test data is reconstructed by the model and the efficacy of the PCA method of analyzing the anomaly score in finding "visual anomalies" in the data. As it is clearly visible in Figure 3.9 the model is able to fit data under normal behavior and when anomalies arise, it causes some of the principal components of the anomaly score to spike. Investigating the loadings of the spiked principal component is then possible to pinpoint what features causes the spike to further investigate the anomaly.

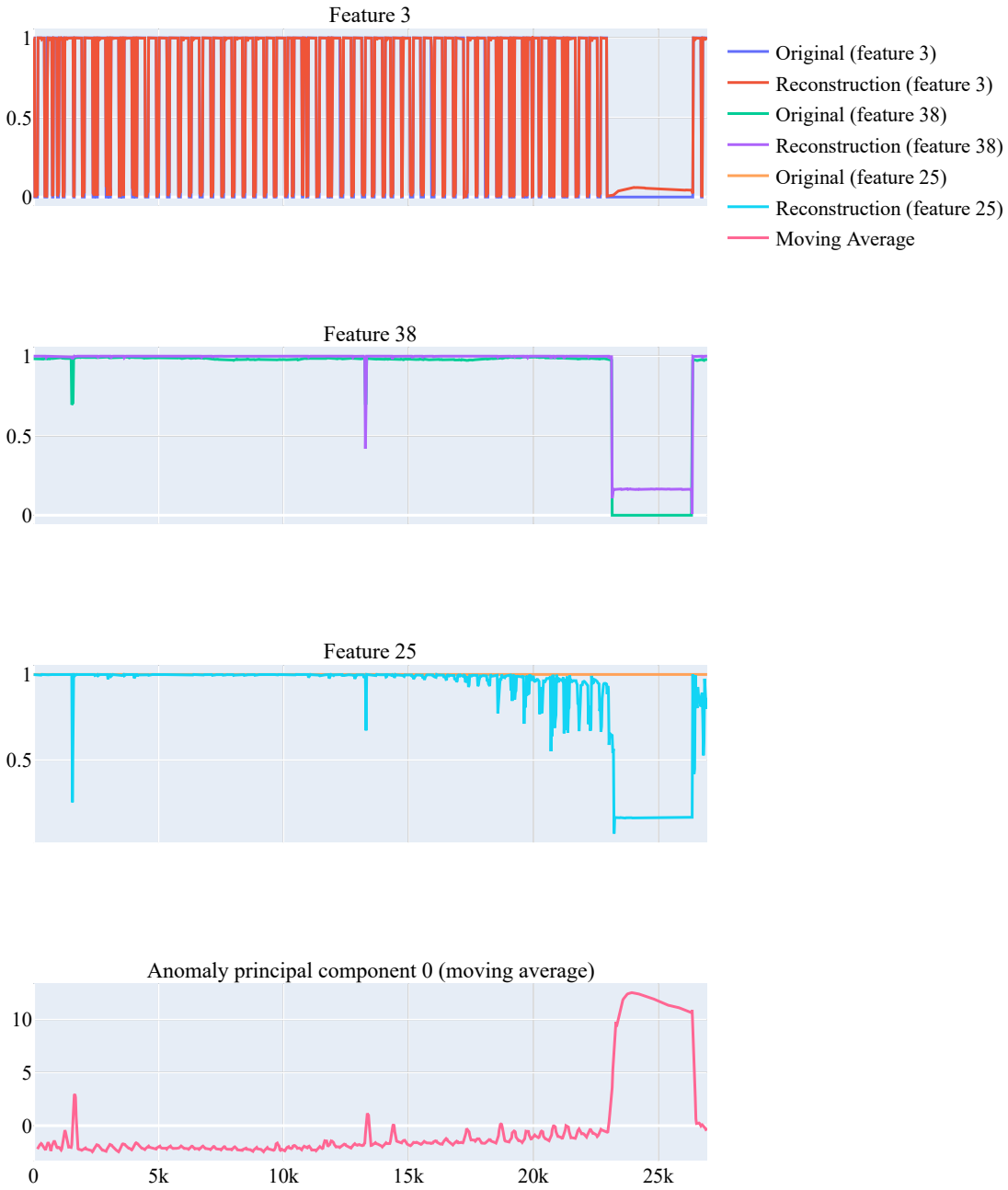


Figure 3.6: A visual plot of the reconstruction performed by the model and the first principal component of the anomaly score for the SWAT dataset.

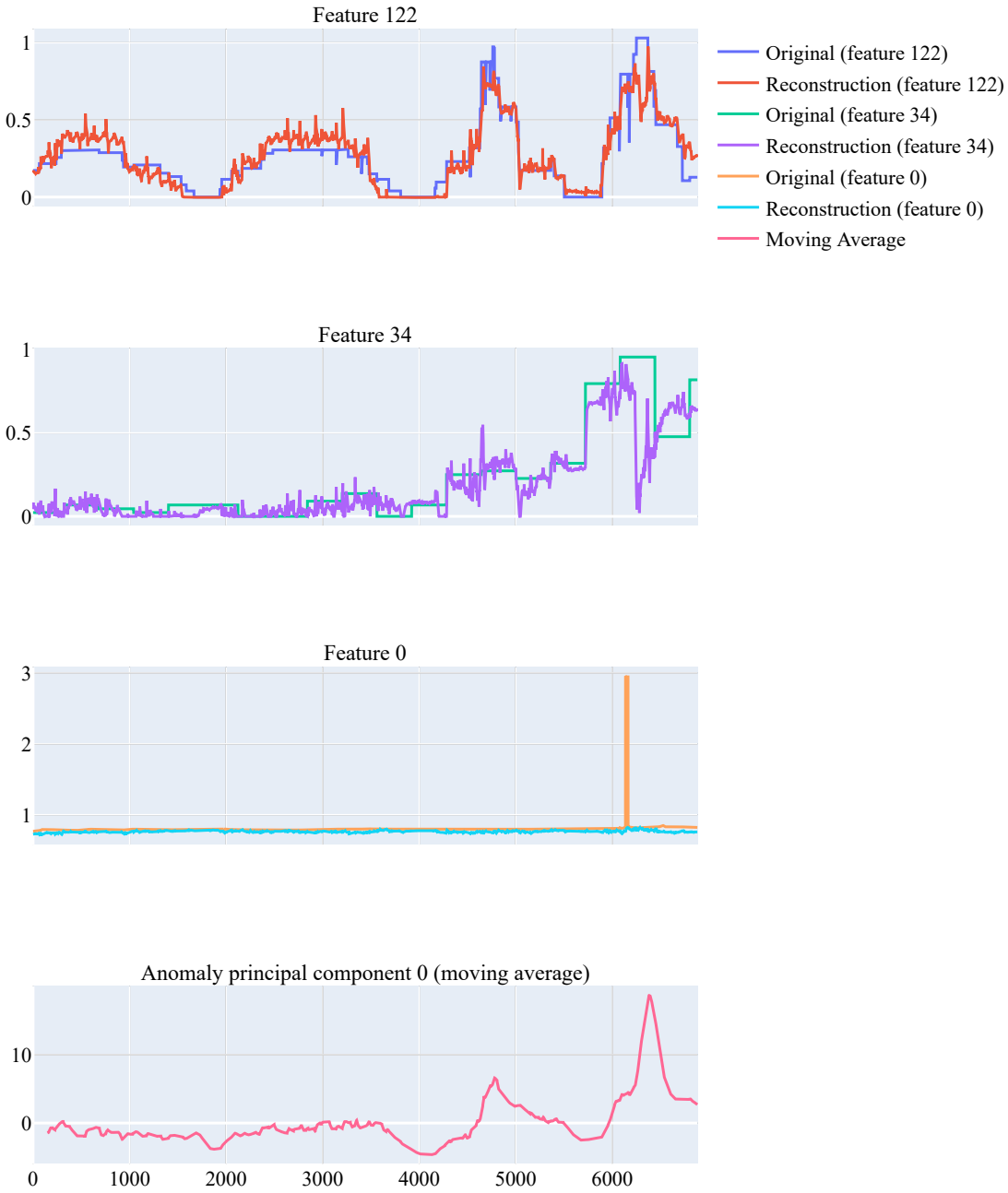


Figure 3.7: A visual plot of the reconstruction performed by the model and the first principal component of the anomaly score for the WADI dataset.

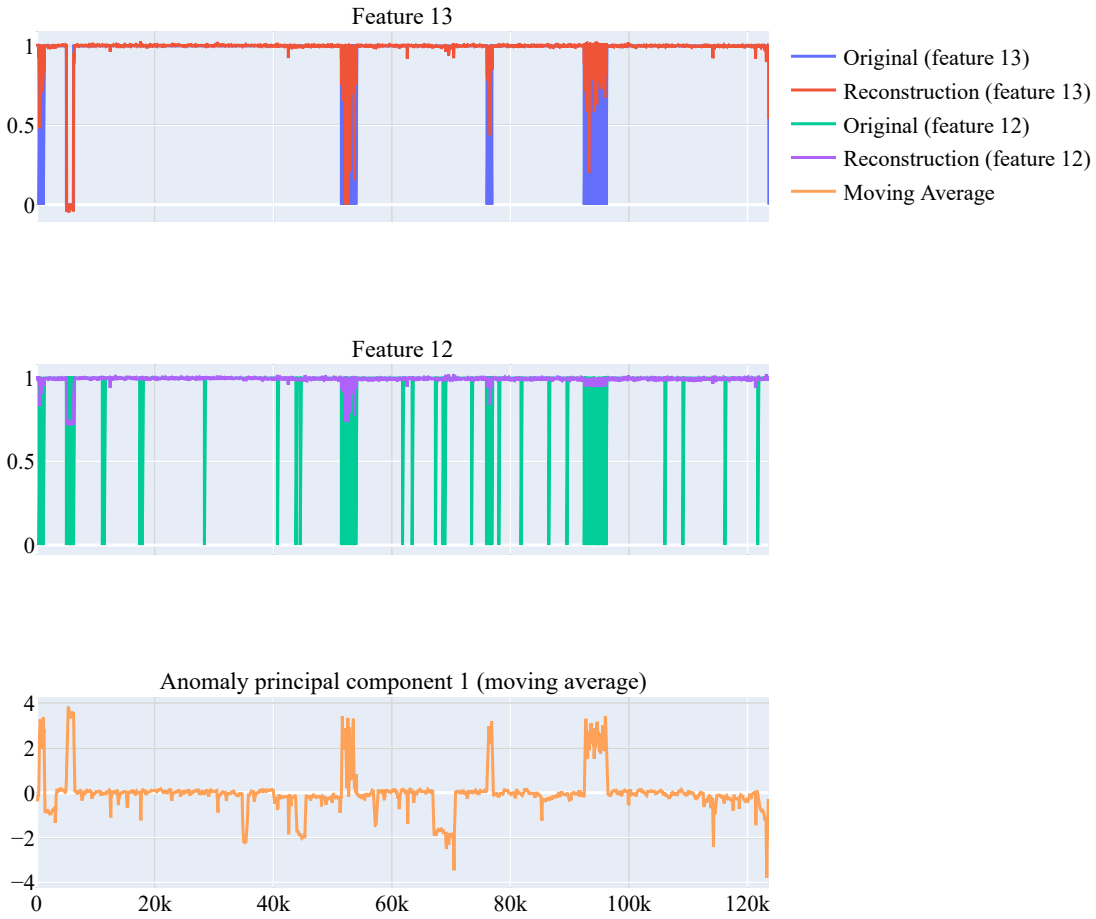


Figure 3.8: A visual plot of the reconstruction performed by the model and the first principal component of the anomaly score for the ACT dataset.

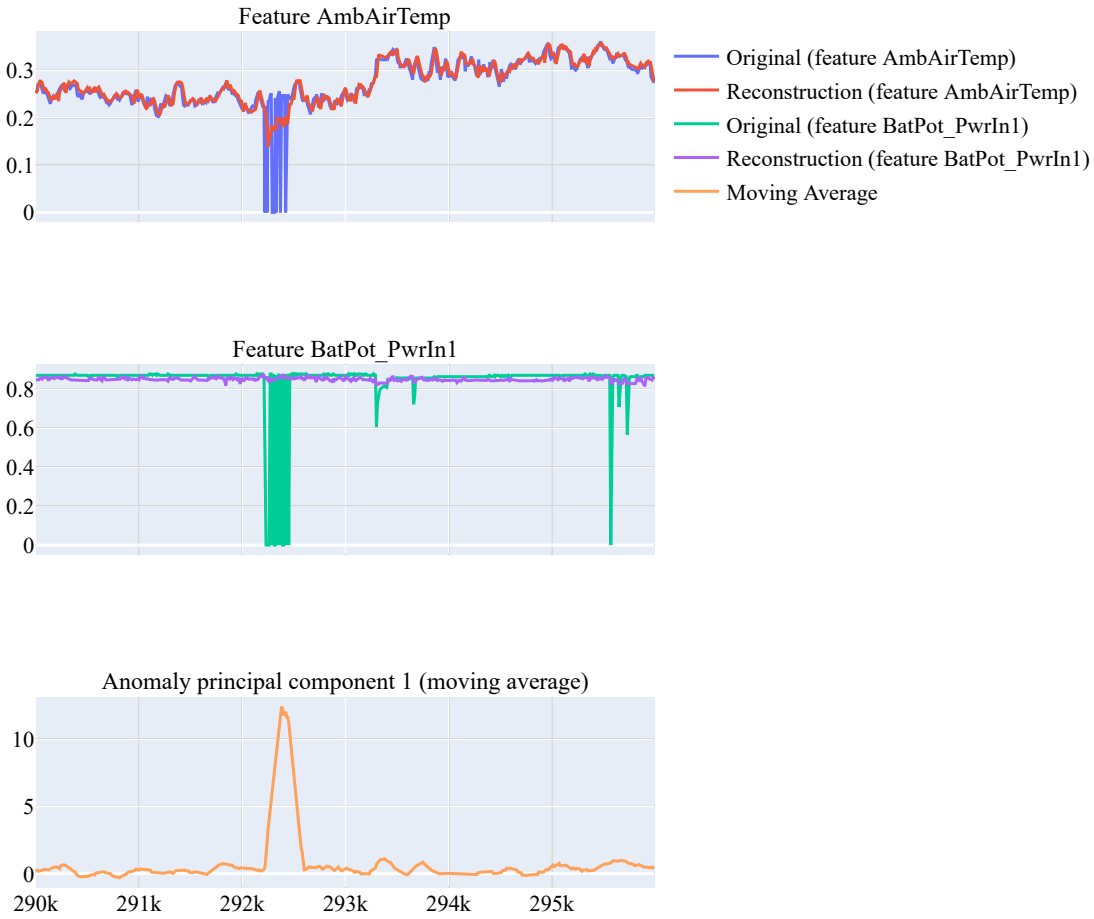


Figure 3.9: A visual plot of the reconstruction performed by the model and the first principal component of the anomaly score for the Automotive dataset.

Chapter 4

Conclusion

In this thesis, we have delved into the intricate realm of anomaly detection within multivariate time series. Our focus revolves around the analysis of time series derived from sensor data, which often exhibits inherent noise and heterogeneity.

The challenge inherent in anomaly detection lies in precisely defining what constitutes an anomaly. Given that anomalies are infrequent occurrences, inherently eluding observation in training data, conventional model training becomes an impractical endeavor. In response to this predicament, our approach in this study gravitates towards the utilization of model-based anomaly detection methods. This methodology entails identifying anomalies when observed data diverges from predicted data. The predictions, in turn, emanate from a model fine-tuned with normal behavioral data. This strategic shift enables a more effective and practical means of detecting anomalies in complex and dynamic multivariate time series. In Chapter 1 we delved into the details of time series, time series modeling, and how models can be used to detect anomalies in time series data listing some model examples.

In Chapter 2, we introduced a cutting-edge methodology centered around graph neural networks for modeling time series. Leveraging Graph Convolution Layers, this approach proves instrumental in capturing correlations within multivariate time series, encompassing both feature interdependencies and historical observations. The synergy of Graph Convolution is harnessed alongside two pivotal components: a fully connected predictor and a variational autoencoder, employed for forecasting future observations.

The incorporation of variational autoencoders stands out for their remarkable efficacy in mitigating overfitting concerns, concurrently demonstrating exceptional accuracy in predictions. This prowess is attributed to the inherent regularization

techniques embedded in their core formulation, making them a robust choice for enhancing predictive performance within the intricate landscape of multivariate time series analysis.

In tandem with graph convolutions and the variational autoencoder, we integrated a powerful methodology known as Temporal Convolution to tackle the inherent heterogeneity within the data. Temporal Convolution networks execute dilated convolutions along the temporal axis for each feature within the time series. This strategic implementation is designed to yield a cohesive and uniform representation across all features, both in the input and output of the model. By employing dilated convolutions, we aim to enhance the model's ability to discern temporal patterns and nuances, fostering a more comprehensive and effective analysis of the diverse data characteristics present in the multivariate time series.

These methodologies have been implemented utilizing the robust PyTorch library. The models underwent training on both publicly accessible labeled benchmark datasets and a proprietary dataset tailored to a specific use case, generously provided by a reputable automotive company. Accuracy assessments are exclusively feasible on supervised benchmark datasets, where the model has exhibited exceptional prowess in discerning anomalies within multivariate time series data.

For the automotive use case, a dedicated visualization tool was crafted to enhance the interpretability of the model output. A PCA-based approach was applied to anomaly score metrics, streamlining the identification of features manifesting the most anomalies. This innovative technique not only automates the process but also furnishes a clear visualization of anomaly peaks, facilitating the visual pinpointing of anomalies.

Looking ahead, future endeavors will hone in on improving the model's interpretability. Acknowledging the inherent interpretational challenges of deep learning models, a prospective study may delve into leveraging the adjacency matrices generated in the graph convolution layers to unravel dependencies between features. Additionally, the automation of the PCA-based approach for anomaly peak detection aims to transform the algorithm from a diagnostic tool into a production-ready, real-time anomaly detector, ushering in heightened efficiency and practical utility.

Bibliography

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, 2018.

A. A. Balkema and L. de Haan. Residual life time at great age. *The Annals of Probability*, 2(5):792–804, 1974. ISSN 00911798. URL http://www.jstor.org/stable/2959306.

Ane Blázquez-García, Angel Conde, Usue Mori, and José Antonio Lozano. A review on outlier/anomaly detection in time series data. *CoRR*, abs/2002.04236, 2020. URL https://arxiv.org/abs/2002.04236.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

Jonathan Goh, Sridhar Adepu, Khurum Nazir Junejo, and Aditya Mathur. A dataset to support research in the design of secure water treatment systems. In Grigore Havarneanu, Roberto Setola, Hypatia Nassopoulos, and Stephen Wolthusen, editors, *Critical Information Infrastructures Security*, pages 88–99, Cham, 2017. Springer International Publishing. ISBN 978-3-319-71368-7.

Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Söderström. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. *CoRR*, abs/1802.04431, 2018. URL http://arxiv.org/abs/1802.04431.

Eamonn Keogh, Jessica Lin, and Ada Fu. Hot sax: Efficiently finding the most unusual time series subsequence. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, page 226–233, USA, 2005. IEEE Computer Society. ISBN 0769522785. doi: 10.1109/ICDM.2005.79. URL https://doi.org/10.1109/ICDM.2005.79.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951. doi: 10.1214/aoms/1177729694. URL https://doi.org/10.1214/aoms/1177729694.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, DMKD '03, page 2–11, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 9781450374224. doi: 10.1145/882082.882086. URL https://doi.org/10.1145/882082.882086.

Kishan G. Mehrotra, Chilukuri K. Mohan, and HuaMing Huang. *Anomaly Detection Principles and Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2017. ISBN 3319675249.

ML4ITS. mtad-gat-pytorch, 2023. URL https://github.com/ML4ITS/mtad-gat-pytorch.

Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. Time-series anomaly detection service at microsoft. *CoRR*, abs/1906.03821, 2019. URL http://arxiv.org/abs/1906.03821.

Jorge-Luis Reyes-Ortiz, Luca Oneto, Alessandro Ghio, Albert Samá, Davide Anguita, and Xavier Parra. Human activity recognition on smartphones with awareness of basic activities and postural transitions. In Stefan Wermter, Cornelius Weber, Włodzisław Duch, Timo Honkela, Petia Koprinkova-Hristova, Sven Magg, Günther Palm, and Alessandro E. P. Villa, editors, *Artificial Neural Networks and Machine Learning – ICANN 2014*, pages 177–184, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11179-7.

Thomas Roelleke. *Information Retrieval Models: Foundations and Relationships*. Morgan & Claypool Publishers, 1st edition, 2013. ISBN 1627050787.

Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouet. Anomaly detection in streams with extreme value theory. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 1067–1075, New York, NY, USA, 2017. Association for

Computing Machinery. ISBN 9781450348874. doi: 10.1145/3097983.3098144. URL https://doi.org/10.1145/3097983.3098144.

M. Thill. *Machine Learning and Deep Learning Approaches for Multivariate Time Series Prediction and Anomaly Detection*. Leiden University, 2022. URL https://books.google.it/books?id=Uvj7zgEACAAJ.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

Bruno Veloso, João Gama, Rita P. Ribeiro, and Pedro M. Pereira. A benchmark dataset for predictive maintenance, 2022.

Hang Zhao, Yujing Wang, Juanyong Duan, Congrui Huang, Defu Cao, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. Multivariate time-series anomaly detection via graph attention network. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 841–850, 2020. doi: 10.1109/ICDM50108.2020.00093.