Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Axel Ø. Harstad & William E. G. Kvaale

Spatio-Temporal Graph Attention Network for Anomaly Detection in the Telco Domain

Master's thesis in Computer Science
Supervisor: Massimiliano Ruocco
Co-supervisor: Sara Malacarne & Claudio Gallicchio
June 2021

NTNU
Norwegian University of Science and Technology

Axel Ø. Harstad & William E. G. Kvaale

Spatio-Temporal Graph Attention Network for Anomaly Detection in the Telco Domain

Master's thesis in Computer Science
Supervisor: Massimiliano Ruocco
Co-supervisor: Sara Malacarne & Claudio Gallicchio
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

In the following pages lies our master's thesis on how the recent advances in deep learning architectures, namely graph neural networks, can perform unsupervised anomaly detection in the Telecommunications (telco) domain. This work is motivated by the need for efficient and accurate anomaly detection in the telco domain, where Key Performance Indicators (KPIs) of base stations are continuously being monitored. Furthermore, network infrastructures are constantly being upgraded, 5G is on its way, and there is an exponential increase of devices and antennas. Thus, it is impractical to achieve robust and dependable anomaly detection results without relying on data-driven models to automate this task. Also, the numerous KPIs constitute multivariate time-series with complex patterns and dependencies that the anomaly detection system should learn to leverage.

In order to address this, we build a complete framework for unsupervised anomaly detection, where we investigate the use of Graph Attention Network (GAT) [43], in combination with a forecasting-based and a reconstruction-based model, in addition to a non-parametric thresholding method. Firstly, we verify our framework on three commonly-used benchmark datasets within anomaly detection. Secondly, we conduct extensive studies where we 1) analyze GAT's potential for capturing and exploiting the complex relationships of multivariate time-series in both spatial and temporal dimensions, 2) investigate the impact of combining a forecasting-based and a reconstruction-based model, and 3) verify the effectiveness of the non-parametric thresholding method. Lastly, we employ and evaluate our framework on real-world telco data provided by Telenor.

Sammendrag

På de neste sidene er masteroppgaven vår om hvordan de nylige fremskrittene i dyplæringsarkitekturer, nemlig grafnevrale nettverk, kan utføre feildeteksjon uten tilsyn, i telekommunikasjonsdomenet (telco). Dette arbeidet er motivert av behovet for effektiv og nøyaktig deteksjon av avvik i telco-domenet, hvor viktige ytelsesindikatorer (KPI-er) for basestasjoner kontinuerlig overvåkes. Nettverksinfrastruktur oppgraderes fortløpende, 5G er på vei, og det er en eksponentiell økning av enheter og antenner. Dermed er det svært utfordrende å oppnå robust og pålitelig anomalideteksjon uten å benytte seg av datadrevne modeller for å automatisere denne oppgaven. Dessuten utgjør de mange KPI-ene multivariate tidsserier med komplekse korrelasjoner og avhengigheter som feildeteksjonssystemet bør lære å dra nytte av.

For å adressere dette bygger vi et komplett rammeverk for feildeteksjon uten tilsyn, der vi undersøker bruken av grafoppmerksomhetsnettverk (GAT) [43], i kombinasjon med en prognosebasert og en rekonstruksjonsbasert modell, i tillegg til en ikke-parametrisk terskelmetode. For det første verifiserer vi rammeverket vårt på tre anerkjente referansedatasett innen feildeteksjon. For det andre gjennomfører vi omfattende studier der vi 1) analyserer GAT sitt potensial for å oppfatte og utnytte komplekse korrelasjoner i multivariate tidsserier i både rom- og tidsdimensjoner, 2) undersøker effekten av å kombinere en prognosebasert og en rekonstruksjonsbasert modell, og 3) verifiserer effektiviteten av den ikke-parametriske terskelmetoden. Til slutt benytter og evaluerer vi rammeverket vårt på virkelig data levert av Telenor.

Preface

This master thesis is part of an MSc. in computer science at the Norwegian University of Science and Technology and is written in collaboration with the Norwegian Open AI Lab and Telenor. We would like to thank our supervisor Massimiliano Ruocco for his exceptional support and discussions throughout this project. Furthermore, we would like to extend our gratitude towards our co-supervisors, Sara Malacarne and Claudio Gallicchio, for sharing their expertise in the field of telecommunications, multivariate time-series analysis and graph neural networks.

Axel Ø. Harstad & William E. G. Kvaale

Trondheim, June 14, 2021

Contents

List of Figures

List of Tables

Acronyms

ANN Artificial Neural Network. 6, 8, 9, 10

BPTT Backpropagation Through Time. 9, 10

DNN Deep Neural Network. 9, 26, 27, 32, 33

GAT Graph Attention Network. 2, 18, 19, 22, 23, 42

GCN Graph Convolutional Network. 16, 17, 19

GNN Graph Neural Network. 2, 13, 14, 15, 16, 18

GRU Gated Recurrent Unit. 2, 11, 12

KPI Key Performance Indicator. 1, 2, 4, 35, 38

LSTM Long Short-Term Memory. 2, 10, 11, 12

MSL Mars Science Laboratory Dataset. 35, 36, 41, 42, 45, 47, 48, 49, 50, 51, 65, 67

MTAD Multivariate Time-series Anomaly Detection. 2, 4

RNN Recurrent Neural Network. 7, 8, 9, 10, 11

SMAP Soil Moisture Active Passive Dataset. 35, 36, 41, 42, 45, 47, 48, 49, 50, 51, 65, 67

SMD Server Machine Dataset. 35, 36, 41, 42, 45, 47, 48, 49, 50, 51, 65, 67

STGAT Spatio-Temporal Graph Attention Network. 3, 65

STGNN Spatio-Temporal Graph Neural Network. 2

TDBU Telenor Denmark Business Unit. 4, 37, 41, 43, 45, 47, 51, 52, 58, 61, 66, 67

telco Telecommunications. 1, 3, 4, 35, 43, 61, 65

Chapter 1

Introduction

In this introductory chapter, we start by explain the motivation for this thesis in section 1.1, and define our goals and research questions in section 1.2. In section 1.3 we outline the main contributions of this thesis, and section 1.4 describes the structure of the thesis.

1.1 Motivation

In the domain of Telecommunications (telco), Key Performance Indicators (KPIs) of base stations are continuously recorded, measuring quantities such as the number of voice-related attempts, the number of data-related failures, and the number of distinct users that receives error messages. In other words, the KPIs reflect the health status of the system. Any failure could potentially impact an immense amount of users, making efficient and accurate anomaly detection vital to alert for potential incidents in time. Throughout this thesis, we denote KPIs, counters, and features interchangeably.

Time-series anomaly detection has been an important research topic for decades, and has a wide range of applications in industry [48]. *Univariate* time-series anomaly detection methods consider individual streams of values (e.g. a single KPI), and can be used to detect anomalies for a single metric. However, monitoring each KPI separately is generally not sufficient to determine if the system as a whole is running normally. The numerous KPIs constitute a *multivariate* time-series, where the KPIs are expected to correlate, both in space and time.

Consequently, it is essential to take the correlations between different KPIs into consideration in a multivariate time-series anomaly detection system.

Due to the inherent lack of labeled anomalies in historical data, and the diverse nature of anomalies, it is unpractical for domain experts to manually label instances as anomalous or non-anomalous. Thus, anomaly detection is typically treated as an *unsupervised* learning problem, where the model must learn to detect anomalies without the help of labeled examples. Until present, there has been significant progress in the study of Multivariate Time-series Anomaly Detection (MTAD). Several classical methods have been developed, such as linear-based models [38], distance-based models [2], and density-based model [4]. However, such techniques model relationship between features in relatively simple ways, e.g. only capturing linear relationship, which is insufficient for the complex, nonlinear characteristics that often are present in real-world data.

More recently, deep learning-based approaches have showed vast improvements in anomaly detection. For instance, reconstruction-based approaches [26], [29], [40], [23] which uses the reconstruction error to assess the anomaly likelihood, and forecasting-based approaches (e.g. via Long Short-Term Memory Networks (LSTMs) or Gated Recurrent Unit Networks (GRUs)) [7], [25], [50], [17], [15] which uses the prediction error, have showed promising performance for multivariate anomaly detection.

Parallel to the progress of MTAD methods, the use of Graph Neural Networks (GNNs) has proven to be a highly successful way for dealing with graph-structured data. A Spatio-Temporal Graph Neural Network (STGNN) is a subgroup of GNNs that works on graphs where the node features changes dynamically over time, and thus aims to learn spatial dependency and temporal dependency at the same time. For example, a traffic network consisting of speed sensors placed along roads can be represented using a graph in which the speed sensors constitutes the nodes and the roads constitute the edges. As the traffic condition of one road may depend on its adjacent roads' conditions, it is necessary to consider spatial dependency when performing traffic speed forecasting [46]. As STGNNs operate on graph data, it typically requires a pre-defined graph structure, specifying the edge connections of the graph. However, Graph Attention Networks (GATs) [43] provides a way of modelling data as a graph without requiring a graph structure. GATs generalize the attention mechanism that has been successfully used in many sequence-based tasks such as machine translation [3], [12], [42], and machine reading [9], and adopts it to be applied on graphs. Recently, GATs have also been applied on time-series data [48], [15], thus constituting a form for spatio-temporal graph neural network that does not require an underlying graph structure.

1.2 Goals and Research Questions

The broader, overall goal that we want our work to contribute towards can be formulated as the following:

Goal *Develop a complete, accurate and robust framework for unsupervised anomaly detection in the telecommunications domain.*

We are provided with real-world industry data of Key Performance Indicators (KPIs) that monitors the behaviour of multiple base stations, collected over a one-year period. A detailed explanation of the KPIs is given in section 5.1. Motivated by its recent success, we aim to study whether Graph Attention Networks (GATs) can be used to model the complex, highly non-linear relationships expected to be present among the KPIs, with the underlying purpose of serving as a part of a complete, accurate and robust framework for unsupervised anomaly detection. Specifically, we formulate our main objective of this thesis through two research questions.

Research question 1 *How can Graph Attention Networks be utilized to model the inter-feature (spatial) relationships between the KPIs?*

In general, GATs are used to model the relationship between features by representing the features as nodes in a graph and their inter-dependencies as the edges. In this way, GATs provides a way to explicitly capture the (feature-wise) correlations between KPIs. However, in addition to being correlated to each other, the different KPIs are expected to contain time-wise dependencies. As the KPIs rely on the user traffic of the telco network, temporal information should be taken into consideration. Thus follows our next research question:

Research question 2 *Can Graph Attention Networks be used to capture and exploit temporal dependencies of the KPIs?*

As already stated, GATs are typically used to model feature-wise correlations rather than time-wise. However, recent work by Zhao et al. [2020] show that GAT can successfully be used for the purpose of learning temporal dependencies. Motivated by this, we aim to study whether GATs additionally can be employed to capture and exploit any temporal dependencies present in the KPIs.

In order to address our research questions, we ensemble a deep learning model that combines a feature-wise (spatial) GAT layer and a time-wise (temporal) GAT layer, together constituting a Spatio-Temporal Graph Attention Network (STGAT).

1.3 Contributions

The main contributions of our work can be summarized as follows:

i We propose a PyTorch [31] implementation of a Spatio-Temporal Graph Attention Network for unsupervised multivariate time-series anomaly detection[1]. The GAT layers are strongly influenced by Microsoft's MTAD-GAT [48].

ii We conduct an ablation study using benchmark datasets within multivariate time-series anomaly detection, demonstrating the impact of core components in our network.

iii We perform extensive preprocessing and feature engineering of a non-public KPI dataset of the telco domain, provided by Telenor Denmark Business Unit (TDBU).

iv We perform comprehensive experiments on the TDBU dataset, visualizing and discussing the results.

v We analyze the learned attention matrices of the GAT layers, in addition to exploring a novel modification of the attention mechanisms (recently proposed in [5]).

vi We investigate the use of transfer learning to achieve a generalized model, capable of working on KPI data of multiple base stations.

1.4 Project Structure

The rest of this thesis is divided into six parts. In Chapter 2, we provide an introduction to the background and theory used in this master thesis. Chapter 3 gives an overview of the State Of The Art and related work for MTAD methods. In Chapter 4, we describe our proposed framework. The experimental setup is outlined in Chapter 5. In Chapter 6, we present and discuss the results. Finally, we summarize our work and give a few suggestions for future work in Chapter 7.

[1] https://github.com/ML4ITS/mtad-gat-pytorch

Chapter 2

Background and Theory

In this chapter, we present an introduction to the main theoretical background needed in order to understand the unsupervised anomaly detection framework proposed in this thesis.

2.1 Anomaly Detection

Anomaly detection aims to discover unexpected events or rare items in data [32]. It has been an active research area for several decades, and being able to accurately detect anomalies constitute a key component of many real-world applications. In most scenarios, the anomalies to be detected do not come from single measurements, but from systems with multiple sensors, generating multivariate time-series. Multivariate time-series anomaly detection entails challenges, as the context of features must be taken into account.

There exist different kinds of anomalies. A *point* anomaly is a data point that is anomalous w.r.t. the majority of other individual data points. A point anomaly can be a *content* anomaly or a *contextual* anomaly. Content anomalies can be defined as abnormal instances in data with respect to the implicit data alone [14], e.g. an abnormal spike in data-related attempts of a base station, independent from external reasons. *Contextual* anomalies are instances that are considered anomalous when viewed in a specific context [28]. We define two subgroups of contextual anomalies, namely *feature-wise* contextual anomalies and *time-wise* contextual anomalies. Feature-wise contextual anomalies refer to data points that

are only considered anomalous when viewed in the context other features. For instance, a sudden increase in voice-calls via a base station does not necessarily represent a failure in the system. It must be viewed in context of the other features, thus making it feature-wise contextual. On the other hand, time-wise contextual anomalies refer to point anomalies that are considered anomalous only within a certain period of time, otherwise not.

2.2 Deep Learning

> Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction
>
> *LeCun et al. [2015]*

The reader is assumed to have a basic understanding of computer science and machine learning concepts. In this thesis, we will concentrate on deep learning, which is a subfield of machine learning. As the introductory citation from LeCun et al. [2015] states, the *deep* part of deep learning comes from the stacking of multiple layers, thus, making the neural networks deep. Deep learning has in recent years been the key ingredient of improvements experienced in domains such as visual object recognition, object detection, speech recognition and protein folding [36].

For decades, conventional machine learning approaches were restricted in its capacity to deal with data in its raw natural form. Prior to the breakthroughs of deep learning, conventional machine learning approaches required feature extractors designed by domain experts in order to convert the raw data into a suitable internal representation that the machine learning system could use as input. Deep learning methods are representational methods that enable machines to learn directly from raw data.

2.2.1 Activation Functions

A key component of deep learning is the use of non-linear modules that transform the data. An Artificial Neural Network (ANN) can obtain the ability to represent highly complex functions by utilizing a sufficient amount of non-linear transformations. These non-linear modules are called activation functions.

The need for non-linearity in ANNs may be attributed to the fact that very few phenomena in this world are constrained to linearity. We need to enable the network to approximate functions that do not follow non-linearity. If we were to

use a linear activation function, such as

$$y = ax + b, \tag{2.1}$$

we would lose the effect of stacking several layers, since stacking multiple linear layers would be equivalent to having one single layer with linear activation.

The sigmoidal, or sigmoid, is an activation function that one can easily recognize as non-linear, as seen in Equation 2.2. Sigmoid is a clear candidate when the target is to classify a probability as an output. It is used extensively in binary classification problems. One can take advantage of the Softmax function for multiclass classification, a more generalized logistic activation function.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

The most commonly-used activation function in recent deep learning has been the Rectified Linear Unit (ReLU), seen in Equation 2.3. ReLU delivers improved gradient propagation, decreasing the likelihood of vanishing gradient problems when compared to other activation functions, such as the sigmoidal. It reduces training time, as it yields a sparse activation, due to all negative values are set to zero. A disadvantage of ReLU is that, as it is bound to the range $[0, inf]$, it can blow up activations.

$$f(x) = max(0, x) \tag{2.3}$$

The sparsity leveraged by ReLU is not only beneficial, but it is also the root of a problem known as *the dying ReLU problem*. The gradient of 0 is 0; hence, neurons receiving negative values will not recover and get stuck at 0 - and effectively die. This phenomenon can arise if the neurons have not been initialized adequately or the data is not properly normalized.

As a result of this problem, Maas et al. [2013] introduced Leaky ReLU. It allows for a slight but significant leak of information for negative values. Leaky ReLU is mathematically described as:

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.01x & \text{otherwise} \end{cases} \tag{2.4}$$

2.3 Recurrent Neural Networks

The main idea behind a Recurrent Neural Network (RNN) [21] is to tackle sequential data. In a conventional fully-connected feed-forward neural network,

one assumes that all inputs to the model are independent. This assumption is not applicable for sequential data, such as text or time-series data from sensors. The temporal dynamics that connect the data are more valuable to the network than the spatial information in each input.

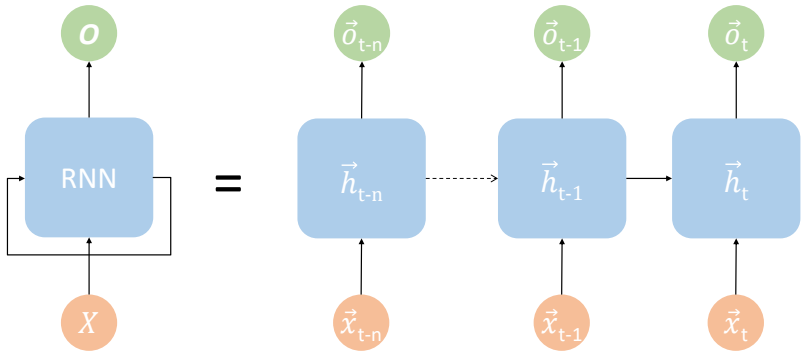


Figure 2.1: Visualization of a rolled and unrolled RNN.

What makes an RNN *recurrent* is that it processes sequential data in a *loop*, which allows the network to persist the state between inputs. The network's hidden layers will inherit the *hidden state*, or working memory, from the previous iteration. An RNN can be seen as a chain of identical ANNs, sharing the same set of weights, non-linear activation functions, and hyperparameters. This way of conceptualizing an RNN is known as *unrolled*, as opposed to the *rolled* version as seen on the left in Figure 2.1.

Specifically, an RNN achieves its recurrent effect by letting the hidden state $\vec{h}_t \in \mathbb{R}^u$ at timestamp t be computed using the current input $\vec{x}_t \in \mathbb{R}^d$ and the previous hidden-state $\vec{h}_{t-1}$, where u is the number of hidden units and d is the number of input features. Furthermore, an output $\vec{o}_t \in \mathbb{R}^c$ is computed using $\vec{h}_t$, where c is the desired dimension of the output. Thus the total input is defined as $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, all hidden states are defined as $\boldsymbol{H} \in \mathbb{R}^{n \times u}$, and the total output is defined as $\boldsymbol{O} \in \mathbb{R}^{n \times c}$.

RNNs contains an additional weight matrix per layer, when compared to a feed-forward neural network, as illustrated in Figure 2.2. This additional hidden-state-

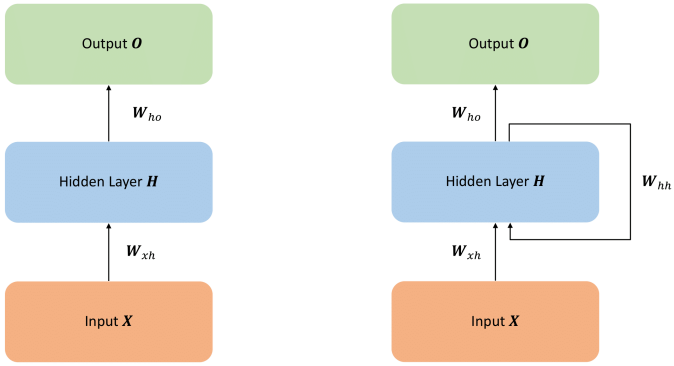


Figure 2.2: Illustration of difference between Feed Forward ANNs and RNNs.

to-hidden-state matrix is defined as $\boldsymbol{W}_{hh} \in \mathbb{R}^{u \times u}$. The weight matrix between the input layer and hidden layer is defined as $\boldsymbol{W}_{xh} \in \mathbb{R}^{d \times u}$, and for the output layer we define $\boldsymbol{W}_{ho} \in \mathbb{R}^{u \times c}$. All the information in the network gets passed through activation functions, such as *tanh* or *sigmoid*, denoted as σ.

Combining all these notations, we have that the hidden state at timestamp t is computed as follows:

$$\vec{h}_t = \sigma_h(\vec{x}_t \boldsymbol{W}_{xh} + \vec{h}_{t-1} \boldsymbol{W}_{hh}), \tag{2.5}$$

and the output is computed as

$$\vec{o}_t = \sigma_o(\vec{h}_t \boldsymbol{W}_{ho}) \tag{2.6}$$

We omitted the use of bias term for the ease of notation.

2.3.1 Backpropagation Through Time (BPTT)

A fundamental component of RNN is Backpropagation Through Time (BPTT) [44]. As conventional ANNs utilize backpropagation with Gradient Descent to obtain the optimal weights, RNNs have an additional layer of complexity due to their cyclic nature. We define a loss function between the network output $\boldsymbol{O}$ and true target values $\boldsymbol{Y}$ as:

$$\mathcal{L}(\boldsymbol{O}, \boldsymbol{Y}) = \sum_{t=1}^{T} \mathcal{L}(\vec{o}_t, \vec{y}_t) \tag{2.7}$$

In simplified terms, the BPTT enables the unrolled chain of ANNs to compute the loss and error gradients across each ANN, backward through the chain, yielding a *rolled up* RNN with updated weights.

2.3.2 Vanishing and Exploding Gradients

There are two widely known issues with training regular RNNs, namely the *vanishing* and the *exploding* gradient problem [30]. When a RNN is trained on long sequences, the gradients will propagated through a long path of timesteps. Vanishing gradients may occur when the norm of gradients is less than one, making the gradients to exponentially decrease towards zero, as they propagate backwards through time. On the other hand, if the norm is greater than one, the gradients will exponentially increase, eventually causing the gradients to eventually. These two problems make it difficult for standard RNNs to learn long-term dependencies.

2.3.3 Long Short-Term Memory (LSTM)

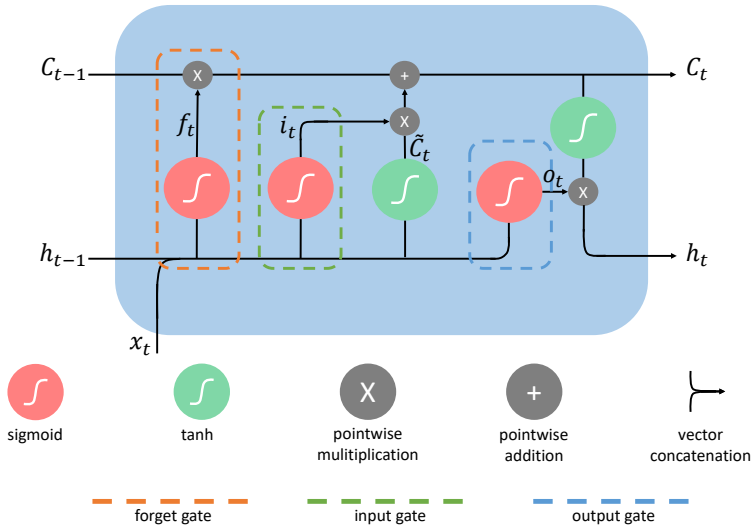


Figure 2.3: Illustration of an LSTM cell.

Since standard RNNs have limitations in retaining long-term dependencies, due to the challenges with exploding and vanishing gradients, other variants of RNNs have been developed. Long Short-Term Memory Networks (LSTMs) [16] alleviate the problems with retaining past data in the working memory. Hence, LSTM is an

algorithm that was developed in order to mitigate the challenges with short-term memory in RNNs. An illustration of an LSTM cell is shown in Figure 2.3.

The difference from a standard RNN is the use of gates. The gates are constructed to optionally pass information through the network. Traditionally, an LSTM contains three gates; input, forget and output gate.

The forget gate is responsible for choosing what information it should remove (*forget*), or retain (*remember*). It consists of one sigmoid function, which acts as a filter.

$$\vec{f_t} = \sigma(\boldsymbol{W}_f[\vec{h}_{t-1} \oplus \vec{x}_t] + \vec{b}_f), \tag{2.8}$$

where $\oplus$ is the concatenation operation.

Furthermore, the input gate decides which information to add to the cell state. It utilizes a sigmoid activation function in conjunction with a tanh function which creates a new set of candidate values $\tilde{\boldsymbol{C}}_t$.

$$\begin{aligned} \vec{i_t} &= \sigma(\boldsymbol{W}_i[\vec{h}_{t-1} \oplus \vec{x}_t] + \vec{b}_i) \\ \tilde{\boldsymbol{C}}_t &= tanh(\boldsymbol{W}_C[\vec{h}_{t-1} \oplus \vec{x}_t] + \vec{b}_C) \end{aligned} \tag{2.9}$$

The attentive reader might notice that both the input gate and forget gate consists of a sigmoid activation with the same input, $\vec{h}_{t-1}$, and $\vec{x}_t$. The reason for this is that the gates have distinct roles. Pointwise operations are applied to the vectors that are allowed through the input and forget gates in the following manner:

$$\boldsymbol{C}_t = \vec{f_t} * \boldsymbol{C}_{t-1} + \vec{i_t} * \tilde{\boldsymbol{C}}_t \tag{2.10}$$

The last gate is the output gate. Its goal is to select the valuable information from the current cell state C_t, and finally, output $\vec{h}_t$.

$$\begin{aligned} \vec{o}_t &= \sigma(\boldsymbol{W}_o[\vec{h}_{t-1} \oplus \vec{x}_t] + \vec{b}_o) \\ \vec{h}_t &= \vec{o}_t * tanh(\boldsymbol{C}_t) \end{aligned} \tag{2.11}$$

2.3.4 Gated Recurrent Unit (GRU)

The GRU, as illustrated in Figure 2.4, was proposed by Cho et al. [2014]. Their contribution was to combine the forget and update gate into one single update gate. The update gate controls how information from previous hidden states will influence the current hidden state. In addition, it has a reset gate, which "effectively allows the hidden state to drop any information that is found to be irrelevant later in the future, thus, allowing a more compact representation." (Cho, et al. (2014), p. 3)[10]

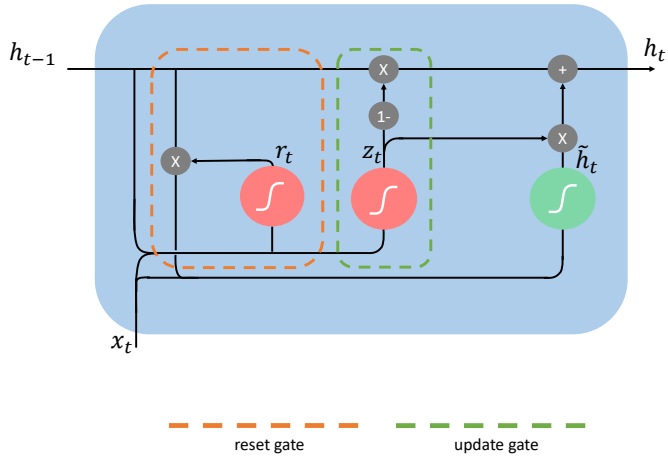


Figure 2.4: Illustration of a cell.

GRU is popularly known to have fewer parameters and is more efficient than LSTM. Furthermore, GRUs do not need to maintain any internal cell state, thus outputting the hidden state directly. Specifically, the gates and the output of a GRU cell is defined as follows:

$$\begin{aligned} \vec{z}_t &= \sigma(\boldsymbol{W}_z[\vec{h}_{t-1} \oplus \vec{x}_t] + \vec{b}_z) \\ \vec{r}_t &= \sigma(\boldsymbol{W}_r[\vec{h}_{t-1} \oplus \vec{x}_t] + \vec{b}_r) \\ \tilde{\vec{h}}_t &= tanh(\boldsymbol{W}_c[\vec{h}_{t-1} \oplus \vec{x}_t] + \vec{b}_c \\ \vec{h}_t &= (1 - \vec{z}_t) * \vec{h}_{t-1} + \vec{z}_t * \tilde{\vec{h}}_t \end{aligned} \tag{2.12}$$

By inspecting Equation 2.12, we observe that the relevance of the previous hidden state, $\vec{h}_{t-1}$, is determined by the values from the update gate, $\vec{z}_t$. If $\vec{z}_t$ approaches 0, it will result in $\vec{h}_{t-1}$ losing its influence, and that the current hidden state will rely more on the candidate hidden state, $\tilde{\vec{h}}_t$.

2.4 Graph Neural Networks

2.4.1 Motivation

Deep learning has revolutionized many machine learning tasks in recent years, ranging from image classification and video processing to speech recognition and natural language understanding [46]. In most of these tasks, the data is typically represented in the Euclidean space. For example, an image can be represented as a 2D grid, and text as a 1D sequence. While deep learning effectively operates on regular Euclidean data, there is an increasing number of applications where data is generated from non-Euclidean domains and represented as graphs with complex relationships between objects [46]. Due to the non-Euclidean structure of graphs, existing deep learning methods cannot be directly applied to graph data. Recently, there is increasing interest in extending deep learning approaches for graph data, which has lead to the emergence of Graph Neural Networks (GNNs), which operate directly on graph data.

2.4.2 Definitions

Graphs are a kind of data structure which models a set of objects (vertices or nodes) and their relationships (edges). Below, we provide formal definitions of graph-related concepts.

Definition 2.4.1 (Graph) *A graph is formulated as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of edges. Let n denote the number of nodes in the graph.*

Definition 2.4.2 (Node Neighbourhood) *Let $v_i \in \mathcal{V}$ denote a node and $e_{ji} = (v_j, v_i) \in \mathcal{E}$ denote an edge pointing from v_j to v_i. The neighbourhood of node v_i is defined as $\mathcal{N}_i = \{v_j \in V \mid e_{ji} \in \mathcal{E}\}$.*

Definition 2.4.3 (Node Features) *A graph may have node features $\boldsymbol{X}$ where $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ is a node feature matrix with $\vec{x_i} \in \mathbb{R}^d$ representing the feature vector of node v_i.*

Definition 2.4.4 (Adjacency Matrix) *The adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is a $n \times n$ holds information about all edges, with $A_{ij} = 1$ if $(v_i, v_j) \in \mathcal{E}$ and $A_{ij} = 0$ if $(v_i, v_j) \notin \mathcal{E}$.*

Definition 2.4.5 (Degree Matrix) *The degree matrix $\boldsymbol{D} \in \mathbb{R}^{n \times n}$ is a diagonal matrix with $D_{ij} = |\mathcal{N}_i|$ if $i = j$, and $D_{ij} = 0$ if $i \neq j$.*

2.4.3 Message Passing

The concept of GNNs was first proposed in [35], that extends existing neural network methods for processing the data represented in graph domains. In general, each node in a graph is naturally defined by its features and the features of related nodes. Motivated by this, the target for GNNs is to learn a node embedding $\vec{h_i} \in \mathbb{R}^d$ which captures the complete information relevant for representing node v_i. The defining feature of any GNN is that is uses a form of *message passing*, in which vector messages are exchanged between nodes in the graph and updated using neural networks.

During each message-passing iteration in a GNN, the embedding $\vec{h_i}$ corresponding to each node $v_i \in \mathcal{V}$ is updated according to information aggregated from v_i's neighbourhood $\mathcal{N}_i$, see Figure 2.5. This message-passing update can be expressed as follows:

$$\begin{aligned} \vec{h}_i^{(k+1)} &= \text{UPDATE}^{(k)}\left(\vec{h}_i^{(k)}, \text{AGGREGATE}^{(k)}(\{\vec{h}_j^{(k)}, \forall j \in \mathcal{N}_i\})\right) \\ &= \text{UPDATE}^{(k)}\left(\vec{h}_i^{(k)}, \mathbf{m}_{\mathcal{N}_i}^{(k)}\right), \end{aligned} \tag{2.13}$$

where UPDATE and AGGREGATE are arbitrary differentiable functions (i.e. neural networks) and $\mathbf{m}_{\mathcal{N}_i}$ is the "message" that is aggregated from v_i's graph neighbourhood $\mathcal{N}_i$. Subscripts are used to denote different nodes, while superscripts to denote different iterations of message passing.

At each iteration of the GNN, the AGGREGATE function takes as input the set of embeddings of the nodes in v_i's neighbourhood and generates a message $\mathbf{m}_{\mathcal{N}_i}$. The update function UPDATE then combines the message with the previous embedding $\vec{h_i}$ of node v_i to generate the updated embedding. The initial embeddings at $k = 0$ are set to the input features for all nodes, i.e. $\vec{h}_i^{(0)} = \vec{x}_i$.

k-hop Neighbourhood

After performing a single aggregation ($k = 1$), a node's embedding goes from being its original feature vector, $\vec{x_i}$, to an combination of the original features of itself and its neighbours. As these iterations progress each node embedding contains more and more information from further reaches of the graph. At $k = 1$ every node embedding contains information from its 1-hop neighbourhood, i.e. the features of its immediate neighbours, which can be reached by a path of length 1 in the graph. After the second iteration ($k = 2$), every embedding has information from its 2-hop neighbourhood, etc. In general, after k iterations every node embedding contains information about its k-hop neighbourhood.

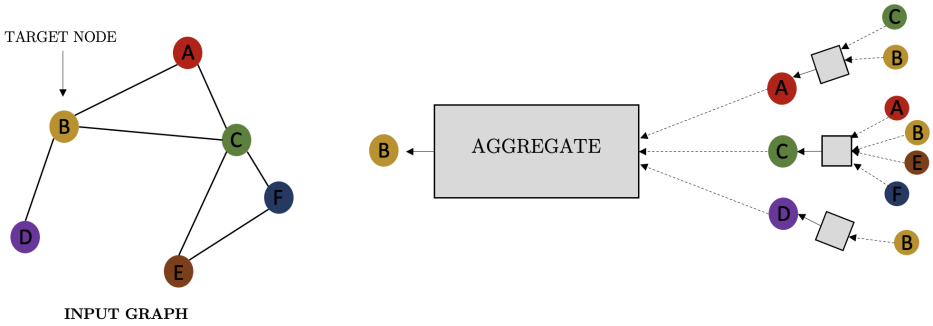


Figure 2.5: Illustration of message passing. The model aggregates messages from B's neighbours (A, C, and D). Simultaneously, A, C and D are updated based on the aggregation of their neighbours, respectively. At the next iteration, the messages aggregated from B's neighbours are followingly based on information aggregated from A, C, and D's neighbourhoods. This way, the receptive field of each node's embedding grows with the number of iterations.

2.4.4 The Basic GNN

As stated earlier, the UPDATE and AGGREGATE functions from Equation 2.13 are typically implemented as neural networks. Using an simplified version of the original model proposed in [35] we have a basic GNN model where the message passing is defined as:

$$\vec{h}_i^{(k)} = \sigma\left(\boldsymbol{W}_{self}^{(k)}\vec{h}_i^{(k-1)} + \boldsymbol{W}_{neigh}^{(k)}\sum_{j\in\mathcal{N}_i}\vec{h}_j^{(k-1)}\right), \tag{2.14}$$

where $\boldsymbol{W}_{self}^{(k)}$ and $\boldsymbol{W}_{neigh}^{(k)}$ are trainable weight matrices and σ is an non-linear activation function (e.g. tanh or ReLU).

2.4.5 Message Passing with Self-loops

It is common to add self-loops to the input graph and thus excluding the need for the update function in the message passing. In this case, the message passing can be simply defined as:

$$\vec{h}_i^{(k)} = \text{AGGREGATE}\left(\{\vec{h}_j^{(k-1)}, \forall j \in \mathcal{N}_i \cup \{i\}\}\right), \tag{2.15}$$

where the aggregation is now taken over the set $\mathcal{N}_i \cup \{i\}$, i.e. the node's neighbours as well as the node itself. The specifics of the aggregation function in Equation 2.15 can then be defined as:

$$\vec{h}_i^{(k)} = \sigma\left(\boldsymbol{W}^{(k)} \sum_{j \in \mathcal{N}_i \cup \{i\}} \vec{h}_j^{(k-1)}\right), \tag{2.16}$$

The consequence of adding self-loops to the graph is that the weight matrices $\boldsymbol{W}_{self}$ and $\boldsymbol{W}_{neigh}$ have shared parameters. It simplifies the message passing and one can this way argue that it reduces the risk of overfitting, but it also limits the expressive power of the GNN, as the information from a node's neighbours is not distinguished from the information from the node itself.

2.4.6 Neighbourhood Normalization

The aggregation operation from Equation 2.14 and Equation 2.16 simply takes the sum of the neighbour embeddings. In cases where nodes have very different degrees (different number of neighbours) this may be problematic. Consider the case where node v_i has 100x higher degree than node v_j. Then it is reasonable to expect that the output from the aggregation will be much larger for v_i than for v_j. This significant difference in magnitude may lead to numerical instabilities and optimization problems. One simple solution to this problem is to normalize the aggregation based upon the degrees of the nodes involved. If we denote the aggregation part of Equation 2.16 as $\mathbf{m}_{\mathcal{N}_i}$, the normalized aggregation becomes:

$$\mathbf{m}_{\mathcal{N}_i} = \frac{1}{|N_i|} \sum_{j \in \mathcal{N}_i} \vec{h}_j \tag{2.17}$$

.

However, [20] proposes a modified normalization factor:

$$\mathbf{m}_{\mathcal{N}_i} = \sum_{j \in \mathcal{N}_i} \frac{\vec{h}_j}{\sqrt{|N_i||N_j|}}, \tag{2.18}$$

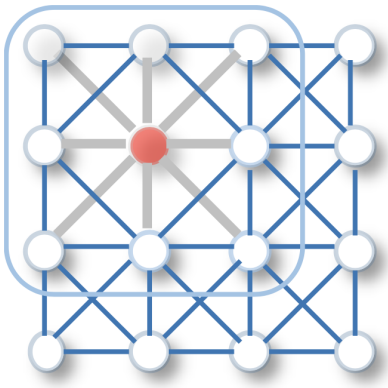
with the intuition that messages coming from high-degree neighbours should be down-weighted, since they may be less precise and informative.

2.4.7 Graph Convolutional Networks (GCN)

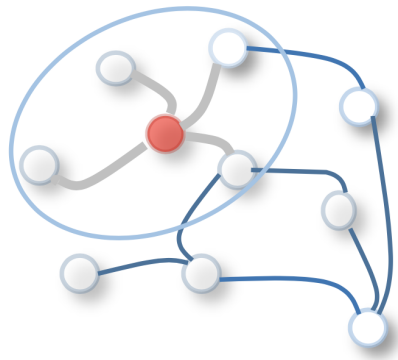
One of the most popular graph neural network models is the Graph Convolutional Network (GCN), first outlined in [20].

GCNs generalize the operation of *convolution* from grid data in the Euclidean space to graph data in the non-Euclidean space. As illustrated in Figure 2.6, an image is represented as a 2D grid and can be considered a special case of graphs, with each pixel representing a node which is connected to its nearby pixels. Similarly to the 2D convolution, a graph convolution is typically performed by aggregating the features of a node's neighbours. Thus, the graph convolution is a form of message passing. Specifically, the standard GCN model from [20] utilizes the Kipf normalization from Equation 2.18 as well as the self-loop update approach from Equation 2.16, and thus the graph convolution (message passing function) is defined as:

$$\vec{h}_i^{(k)} = \sigma\left(\boldsymbol{W}^{(k)} \sum_{j \in \mathcal{N}_i \cup \{i\}} \frac{\vec{h}_j^{(k-1)}}{\sqrt{|N_i||N_j|}}\right). \tag{2.19}$$



(a) 2D convolution. Each pixel in an image is viewed as a node, with neighbours determined by the filter size (here: 3x3). The 2D convolution takes the weighted average of pixel values within the filter. Note that the neighbours of a node are positionally ordered and have a fixed size.



(b) Graph convolution. To get a hidden representation of the red node, the graph convolution operator aggregates the node features of the red node along with its neighbours, typically via a summation or average. Contrary to the image case, the neighbours of a node are unordered and may vary in size.

Figure 2.6: 2D Convolution vs. Graph Convolution (figure adapted from [46])

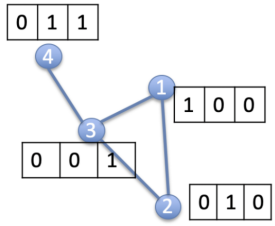
Graph Convolution using the Adjacency Matrix

So far, we have defined the message-passing functions at the node level. However, in practice, message passing can often be performed simultaneously for all nodes. In that case, we should define the message passing at a graph-level. The adjacency matrix contains information about every node's neighbourhood, i.e. all the edges in the graph. Let $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ denote the adjacency matrix with added self-loops. The degree matrix $\boldsymbol{D}$ contains the degree of each node. Let $\tilde{\boldsymbol{D}}$ denote the degree matrix of $\tilde{\boldsymbol{A}}$. Furthermore, let $\boldsymbol{H} = \{\vec{h}_0, \vec{h}_1, ..., \vec{h}_n\} \in \mathbb{R}^{n \times d}$ denote the matrix of node embeddings.

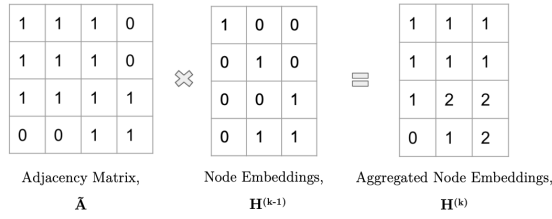
Then, we can write the graph-level version of Equation 2.19 as:

$$\boldsymbol{H}^{(k)} = \sigma\left(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{(k-1)}\boldsymbol{W}^{(k-1)},\right) \tag{2.20}$$

where $\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$ achieves the Kipf normalization, while $\tilde{\boldsymbol{A}}$ together with $\boldsymbol{H}^{(k-1)}$ performs the neighbourhood aggregation. A simplified version of Equation 2.20 is depicted in Figure 2.7, which illustrates how the adjacency matrix is used to perform neighbour aggregation for all nodes simultaneously.



(a) Example graph with four nodes, each with a 3-dimensional embedding vector $\vec{h}_i^{(k-1)}$.



(b) The adjacency matrix (including self-loops) specifies all edges present in the graph, and the embedding matrix contains all node embeddings. The updated (aggregated) node embeddings are then given by the matrix multiplication between the adjacency matrix and the embedding matrix.

Figure 2.7: Simplified Graph convolution using matrix multiplications

2.4.8 Graph Attention Networks (GAT)

Attention mechanisms has been successfully used in many sequence-based tasks [3], [12]. Veličković et al. [2018] propose Graph Attention Networks (GATs) which incorporates the attention mechanism into the aggregation step of the GNN. The core idea is to assign an attention weight to each neighbour, which

is used to weight this neighbour's influence during the aggregation. As opposed to GCNs, GATs allows the assigning of different importances to nodes of a same neighbourhood, enabling a leap in model capacity [43]. The message passing function is defined as:

$$\vec{h}_i' = \sigma\left(\sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{ij} \boldsymbol{W} \vec{h}_j\right), \tag{2.21}$$

where $\vec{h_i'}$ is the updated embedding of node v_i, $\vec{h}_j$ is the current embedding of node v_j, $\boldsymbol{W}$ is a weight matrix of shared linear transformation which is applied to every node, and α_{ij} denotes the attention that node v_i should attend to node v_j during the aggregation. The attention mechanisms are illustrated in Figure 2.8.

To get the attention weight α_{ij}, an attention *score* is first computed as

$$e_{ij} = a(\boldsymbol{W}\vec{h}_i, \boldsymbol{W}\vec{h}_j), \tag{2.22}$$

where a is a single-layer feedforward neural network, parametrized by a weight vector $\vec{a} \in \mathbb{R}^{2d}$. In essence, a works by multiplying $\vec{a}^T$ with the concatenation of the transformed node embeddings of node v_i and v_j, and applies a nonlinearity function (LeakyReLU in [43]) to output a score that indicates the *importance* of v_j's features to v_i, i.e.

$$e_{ij} = \text{LeakyReLU}\left(\vec{a}^T[\boldsymbol{W}\vec{h}_i \oplus \boldsymbol{W}\vec{h}_j]\right), \tag{2.23}$$

where $.^T$ represent transposition and $\oplus$ is the concatenation operation.

Note that this attention score may be computed between every pair of nodes, dropping all structural information. However, the graph structure is commonly injected by performing *masked attention*, i.e. e_{ij} is only computed for nodes $j \in \mathcal{N}_i \cup \{i\}$. In practice, this masking can be achieved by multiplying with the adjacency matrix. These attention scores are then normalized using the softmax function:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i \cup \{i\}} \exp(e_{ik})}. \tag{2.24}$$

Fully expanded out, the attention weights is then defined as:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{a}^T[\boldsymbol{W}\vec{h}_i \oplus \boldsymbol{W}\vec{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}_i \cup \{i\}} \exp\left(\text{LeakyReLU}\left(\vec{a}^T[\boldsymbol{W}\vec{h}_i \oplus \boldsymbol{W}\vec{h}_k]\right)\right)}. \tag{2.25}$$

Note that because the attention weights are normalized, i.e.

$$\sum_{k \in \mathcal{N}_i \cup \{i\}} \alpha_{ik} = 1,$$

the aggregation in Equation 2.21 will followingly be normalized across the neighbouring nodes, so there is no need for an additional normalization step.

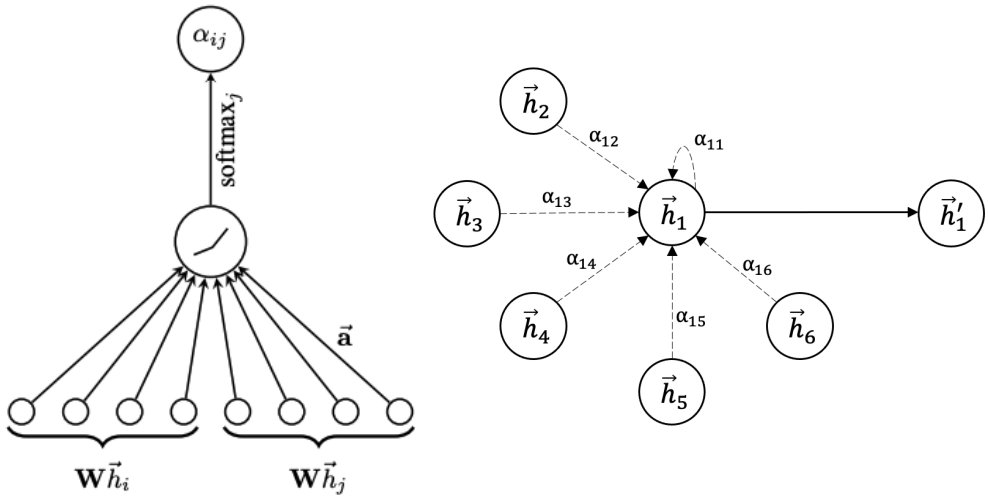


Figure 2.8:
Left: The attention mechanism $a(\boldsymbol{W}\vec{h}_i, \boldsymbol{W}\vec{h}_j)$ computes attention weights α_{ij} for all neighbours of v_i (figure adapted from [43]). **Right:** The embeddings are aggregated using the attention weights in order to obtain the updated embedding $\vec{h'}_1$.

Chapter 3

State Of The Art

Time-series anomaly detection is a well-researched domain and is yet an active field of research as it has many applications on an industrial scale. One perspective of categorizing the methods under this umbrella is supervised versus unsupervised learning methods. Supervised learning methods require data with labels in order for the model to learn. In the real world, time-series data are collected at an incredible speed which makes the task of labeling anomalies challenging, and expensive. Therefore, unsupervised learning methods compose an essential toolbox to tackle multivariate time-series analysis [18]. There exists a few other perspectives to categorize anomaly detection methods, such as univariate versus multivariate, and forecasting-based vs reconstruction-based solutions. In this chapter, we summarize important works about multivariate time-series anomaly detection, from the perspective of the latter.

3.1 Forecasting-based Models

In general, forecasting-based models tries to predict future values of a time-series, using a window of predecessing values. Then, anomalies are predicted based on the forecasting errors.

Chauhan et al. [7] employs deep LSTM networks in order to develop a forecasting-based model for healthy ECG (electrocardiogram) signals. LSTM-AD [25] also uses LSTM-layers, but predicts multiple steps ahead, instead of just one. The prediction errors are modeled as a multivariate Gaussian distribution, which is used

to assess the likelihood of the observed value. DAGMM [50] focuses on anomaly detection of multivariate data without any temporal dependencies, where the input of DAGMM is just a single entity observation (all features but for one timestamp at the time), instead of a sequence of inputs.

Hundman et al. from NASA JPL proposes LSTM-NDT [17], an unsupervised and non-parametric dynamic thresholding method for fitting a threshold on the forecasting error, and uses a simple LSTM network to forecast values. In addition, they propose a false-positive mitigation strategy which is a key element in developing systems for real-life applications.

He et. al. [15] argue that existing methods are not explicitly capturing relationships between features, and propose to use a GAT layer for this purpose. Additionally, they use multiscale convolutions to capture temporal patterns of various ranges.

3.2 Reconstruction-based Models

In general, reconstruction-based models learn a representation of the *normal behavior* for the entire time-series input sequence, by reconstructing the original input from some latent representation with reduced dimension. The reconstruction error is then used to predict anomalies.

LSTM-ED [26] propose an Encoder-Decoder scheme to learn the representation over normal time-series. LSTM-VAE [29] suggests to use an LSTM-based Variational Auto-Encoder (VAE) as the reconstruction model. It models the underlying distribution of the multivariate time-series and reconstructs the data with an expected distribution. OmniAnomaly [40] argues that the stochastic nature of multivariate time-series often causes deterministic reconstruction methods to be mislead by unpredictable (random) patterns. Therefore, they propose a stochastic recurrent neural network, in which robust representations are learned using stochastic variable connection and normalizing flows. Generative Adversarial Network (GANs) have been widely used in multivariate time-series anomaly detection, where it learns to reconstruct the input from a latent representation, through a minimax two-player game [13]. MAD-GAN [23] is an approach utilizing GANs, where the generator and discriminator is modeled using LSTM networks.

3.3 Combination-based Models

Both forecasting-based and reconstruction-based models have shown success in the task of multivariate time-series anomaly detection. Zhao et. al. [48] argue that forecasting-based and reconstruction-based models have shown their superiority in some specific situations, and that they are complementary to each other. Therefore, they propose to make the model both forecast next values in addition to reconstruct the input sequence of values. Moreover, they leverage two parallel GAT layers to explicitly capture correlations among both features and timestamps.

Chapter 4

Methodology

In this chapter, we explain each part of our framework in detail. We begin by mathematically formulate the problem we seek to address, in section 4.1. In section 4.2 we give an overview of the complete framework, while section 4.3 explains the DNN model and its components, and section 4.4 describes the inner workings of the Threshold Selection Model.

4.1 Problem Formulation

The overall problem we seek to address is that of multivariate time-series anomaly detection, which can be formulated as follows:

Problem Formulation: An input of multivariate time-series anomaly detection is denoted by $\boldsymbol{X} = \{\vec{x}_1, \vec{x}_2, ..., \vec{x}_n\} \in \mathbb{R}^{n \times k}$, where n is the number of timestamps, and k is the number of features in the input. $\vec{x}_i \in \mathbb{R}^k$ is the vector containing values for all k variables at time i. As time-series can be very long, we use a sliding window to generate fixed-length inputs of length n: $\{\vec{x}_{t-n}, \vec{x}_{t-n+1}, ..., \vec{x}_{t-1}\}$. The target is to produce an output vector $y \in \mathbb{R}^n$, where $y_i \in \{0, 1\}$ denotes whether x_i is an anomaly.

4.2 Framework Overview

We address the problem of multivariate time-series anomaly detection by combining a DNN model with a threshold selection model (TS model). In this section we describe the architecture of our proposed framework. The DNN model is heavily inspired by the recent works of Zhao et. al. [48], which introduced a novel Graph Attention model named *MTAD-GAT* for solving the problem of multivariate time-series anomaly detection. However, as there is no public code available for *MTAD-GAT*, the implementation is done solely by us.

The overall architecture of our framework is shown in Figure 4.1. It consists of a deep neural network which takes the multivariate time-series data as input and outputs anomaly scores, indicating its belief in different timestamps being anomalous. These anomaly scores are fed to a threshold selection model, which fits a threshold to the distribution of anomaly scores, and uses this threshold to label timestamps as anomalous or non-anomalous.

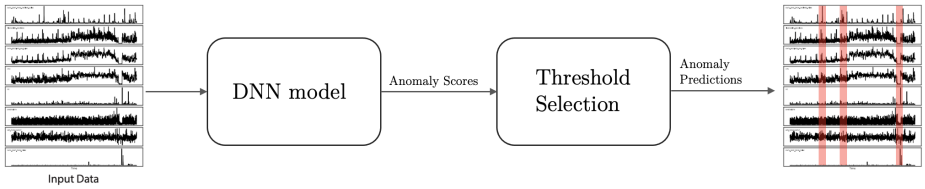


Figure 4.1: Overall architecture of our framework for multivariate time-series anomaly detection.

4.3 DNN Model

The architecture of the DNN model is depicted in Figure 4.2. In summary, the DNN model takes a window of multivariate time-series data as input, $\boldsymbol{X} = \{\vec{x}_{t-n}, ..., \vec{x}_{t-1}\} \in \mathbb{R}^{n \times k}$ ($k = 8$ in Figure 4.2), and tries to both forecast values for time t, as well as reconstruct values for the complete input sequence. The predictions are compared to the real values of the time-series, yielding two separate losses, and the model is optimized in an end-to-end fashion using a joint objective function which combines the two losses. At inference, the same function is used to output the anomaly scores that are passed to the threshold selection model. This way, we have that the DNN model is separate from the threshold selection model, and they do not interact with each other when the DNN model is trained.

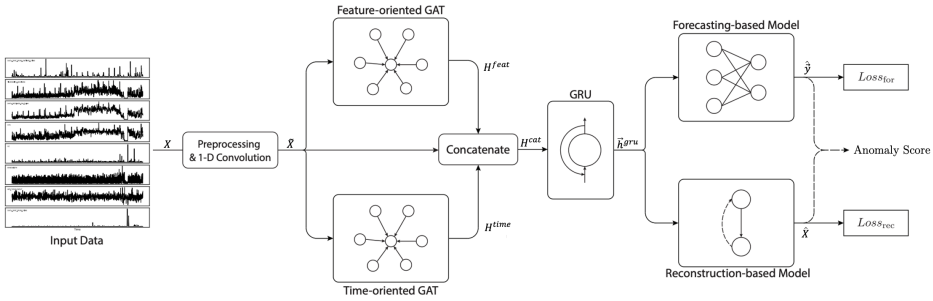


Figure 4.2: The architecture of the DNN model in our framework. Figure partially adapted from [48].

The DNN model is composed of the following modules (in order):

1) We preprocess the raw input data and apply a 1-D convolution in the temporal dimension in order to smooth the data.
2) The output of the 1-D convolution module are processed by two parallel graph attention modules, one feature-oriented and one time-oriented, in order to capture dependencies among features and timestamps, respectively.
3) We concatenate the output from the 1-D convolution module and the two GAT modules, and feed them to a Gated Recurrent Unit, to capture longer sequential patterns.
4) The output from the GRU are fed into a forecasting model and a reconstruction model, to get a prediction for the next timestamp, as well as a reconstruction of the input sequence.

Table 4.1 summarizes the notations we will use throughout this section.

4.3.1 1-D Temporal Convolution

A 1-D convolution with kernel size d_{conv} is applied to the preprocessed input data. The convolution works along the temporal direction, each feature constituting a channel. As demonstrated in previous work [34], the convolution operation is good at local feature engineering within a sliding window, and has been shown to alleviate possible noise effects of the original input data [15]. We denote the result of the 1-D convolution as $\tilde{\boldsymbol{X}} \in \mathbb{R}^{n \times k}$ (having the same dimension as $\boldsymbol{X}$).

4.3.2 Feature-oriented Graph Attention Layer

The graph attention (GAT) layers are the core designs of the DNN model. The goal of the feature-oriented GAT layer is to capture correlations existing across

$\boldsymbol{X}$	an instance of multivariate time-series input
n	the length of $\boldsymbol{X}$ in a pre-defined sliding window
k	the number of features in $\boldsymbol{X}$
$\tilde{\boldsymbol{X}}$	input after preprocessing and 1-D Convolution
$\vec{v}_i$	input node representation for a GAT layer
$\vec{h}_i$	output node representation for a GAT layer
$\boldsymbol{H}^{feat}$	total output for the feature-oriented GAT layer
$\boldsymbol{H}^{time}$	total output for the time-oriented GAT layer
$\boldsymbol{H}^{cat}$	concatenation of $\tilde{\boldsymbol{X}}$, $\boldsymbol{H}^{feat}$ and $\boldsymbol{H}^{feat}$
$\vec{h}^{gru}$	last hidden state of the GRU layer
d_{conv}	kernel size of the 1-D Convolution
d_{gru}	hidden dimension of the GRU layer
d_{for}	hidden dimension of the fully-connected layer in the Forecasting Model
d_{rec}	hidden dimension of the GRU in the Reconstruction Model
$\hat{\vec{y}}$	one-step ahead forecasts from the Forecasting Model
$\hat{\boldsymbol{X}}$	window reconstructions from the Reconstruction Model
$\vec{s}$	anomaly scores outputted from the DNN model
γ	hyper-parameter to combine forecasting and reconstruction loss into the anomaly score
ϵ	threshold fitted by the Threshold Selection Model

Table 4.1: Notations

the different features of the multivariate time-series. The feature-oriented GAT layer treats the multivariate time-series as a complete graph, where each node represents a feature, and each edge denotes the relationship between two corresponding features. That is, in our input data $\boldsymbol{X} \in \mathbb{R}^{n \times k}$, we view each column (feature) as a separate node. This way, each node is represented by a sequential vector $\vec{v}_i = \vec{x}_{:,i} = \{x_{1,i}, x_{2,i}, ..., x_{n,i}\}$ of length n, where $x_{t,i}$ is the value of feature i at time t.

As described in subsection 2.4.8, a GAT layer explicitly models the relationship between every pair of nodes by learning attention scores that are used to weight the neighbours' influence during the aggregation. This way, the relationships between different features of the multivariate time-series can be carefully captured. Also, GAT layers do not rely on a pre-defined graph structure, i.e. an adjacency matrix. This is a convenient property, as many multivariate time-series does not have a natural graph structure defined. However, if present, the graph structure can easily be injected into the GAT layer.

Specifically, the output for each node is calculated by the feature-oriented GAT layer as follows:

$$\vec{h}_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \vec{v}_j\right), \tag{4.1}$$

where $\vec{h}_i \in \mathbb{R}^n$ denotes the output representation of node $\vec{v}_i$, α_{ij} is the attention

score between node i and node j, $\mathcal{N}_i$ is the neighbouring nodes of node i, and σ represents the sigmoid activation function. Note that $\mathcal{N}_i$ will be the set of all nodes if no graph structure is defined.

We denote the output representation for each node of the feature-oriented GAT layer as $\vec{h}_i^{feat} \in \mathbb{R}^n$, and there is a total of k nodes. Thus, the total output from this layer is a matrix $\boldsymbol{H}^{feat} = \{\vec{h}_1^{feat}, ..., \vec{h}_k^{feat}\} \in \mathbb{R}^{k \times n}$.

Figure 4.3 illustrates how the feature-oriented GAT layer creates the graph from the input data, and how the output representation is computed for a single node.

4.3.3 Time-oriented Graph Attention Layer

The goal of the time-oriented GAT layer is to capture dependencies between different timestamps (as opposed to between features). In order to utilize the GAT layer for this purpose, the time-oriented GAT layer views the input data as a complete graph in which each node represents the values for all features at a specific timestamp. Concretely, in the input data $\boldsymbol{X} \in \mathbb{R}^{n \times k}$, we view each row (timestamp) as a separate node, and all other timestamps in the sliding window as neighbours. This way, each node is represented by the sequential vector $\vec{v}_t = \vec{x}_{t,:} = \{x_{t,1}, x_{t,2}, ..., x_{t,k}\}$. As opposed to in the feature-oriented GAT layer, each node now has length k, and there are n nodes in total. Otherwise, the output representation of each node is computed in the same way as in Equation 4.1.

We denote the output representation for each node of the time-oriented GAT layer as $\vec{h}_i^{time} \in \mathbb{R}^k$, and there is a total of n nodes. Thus, the total output from this layer is a matrix $\boldsymbol{H}^{time} = \{\vec{h}_1^{time}, ..., \vec{h}_n^{time}\} \in \mathbb{R}^{n \times k}$.

Figure 4.4 shows how the time-oriented GAT layer creates the graph from the input data, and how the output representation is computed for a single node. Note the differences from Figure 4.3.

4.3.4 GRU for Long-Term Temporal Dependencies

We concatenate the output of the 1-D convolution ($\tilde{\boldsymbol{X}}$), the feature-oriented GAT ($\boldsymbol{H}^{feat}$) and the time-oriented GAT ($\boldsymbol{H}^{time}$) along the feature dimension, in order to make a matrix of shape $n \times 3k$, which we denote as $\boldsymbol{H}^{cat}$. The concatenation is then fed to a single-layered GRU with d_{gru} hidden state size, for capturing sequential patterns. $\boldsymbol{H}^{feat}$ has augmented the representation of each feature by combining the information of correlated features, while $\boldsymbol{H}^{time}$ combines the information from correlated timestamps. On the other hand, $\tilde{\boldsymbol{X}}$ contains a smoothed

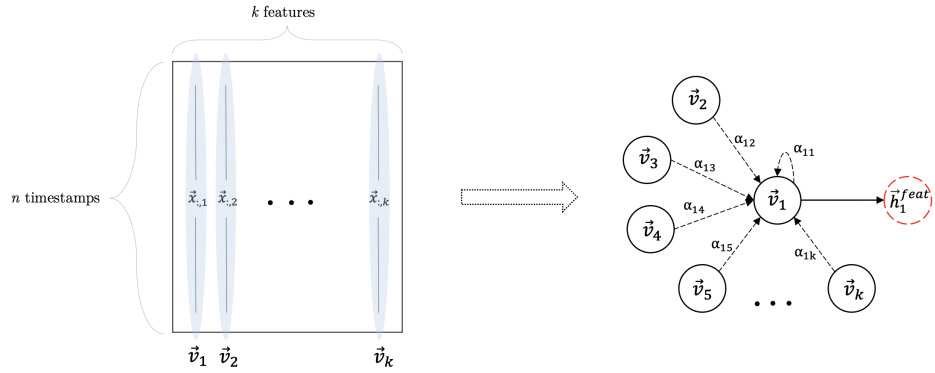


Figure 4.3: Illustration of how the feature-oriented GAT layer creates a graph where each node represents *one* feature across n timestamps, and how the attention mechanism is used to compute the output $\vec{h}^{feat}$ for a specific node.

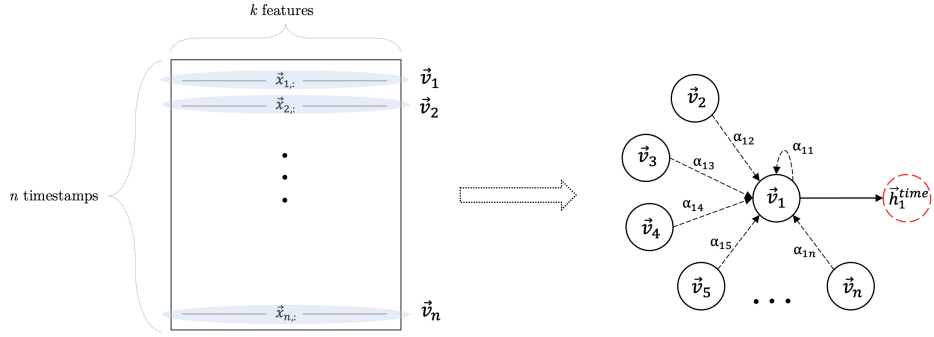


Figure 4.4: Illustration of how the time-oriented GAT layer creates a graph where each node represents *one* timestamp across k features, and how the attention mechanism is used to compute the output $\vec{h}^{time}$ for a specific node.

version of the original input. Intuitively, these three sources add different but informative information. This enriches the information available at each timestamp, thus helping the GRU to better capture temporal dependencies. Recall from subsection 2.3.4 that a GRU outputs the hidden state corresponding to each input timestamp that it receives. Thus, we extract the *last* hidden state as the final output of the GRU layer, and denote it as $\vec{h}^{gru} \in \mathbb{R}^{d_{gru}}$.

4.3.5 Forecasting Model

The output from the GRU layer is flattened and fed to a forecasting model that predicts the value for each feature at the next timestamp. Specifically, the forecasting model is a 3-layered fully-connected neural network, with hidden dimensions d_{for}. Intuitively, the last hidden state of the GRU has received information from all prior timestamps within the sliding window, and should thus contain useful information to forecast the next value. We denote the output from the forecasting model as $\hat{\vec{y}} \in \mathbb{R}^k$, containing the one-step ahead prediction for each of the k features.

4.3.6 Reconstruction Model

The reconstruction model takes the output from the GRU layer ($\vec{h}^{gru}$) and tries to reconstruct the complete input sequence. As seen from the reconstruction model's perspective, one can view the previous parts of the model as an encoder, and the reconstruction model as the decoder. From this point of view, the path going from input data and through the reconstruction model constitutes an auto-encoder, where the input data $\boldsymbol{X}$ has been encoded into latent variables of dimension d_{gru} (the output from the GRU layer), and then decoded by the reconstruction model. Contrary to [48], which uses an Variational Auto-Encoder, our reconstruction model consists of a single-layered GRU with hidden dimension d_{rec}. As the GRU expects sequential data, $\vec{h}^{gru}$ is copied n times (length of input data), and then fed to the GRU. This also ensures that each individual timestamp reconstruction within the window is dependent on the reconstruction of previous timestamps, making them more consistent. We denote the output from the reconstruction model as $\hat{\boldsymbol{X}} \in \mathbb{R}^{n \times k}$, containing reconstructions for each k feature for each n timestamp.

4.3.7 Joint Optimization & Anomaly Score

Given the current window input from $t = 0$ to $t = n$, the loss function for the forecasting model is the Root Mean Squared Error (RMSE) between the predicted and the actual values for $t = n + 1$:

$$Loss_{for} = \sqrt{\frac{1}{k}\sum_{i=1}^{k}(y_i - \hat{y}_i)^2}, \tag{4.2}$$

where $\hat{y}_i$ is the predicted value for the i^{th} feature, and y_i is the actual value, i.e. $y_i = x_{n+1,i}$.

Similarly, for the reconstruction model, the loss function is the RMSE between

the reconstructed values and the actual values for the complete input window:

$$Loss_{rec} = \sqrt{\frac{1}{nk}\sum_{t=1}^{n}\sum_{i=1}^{k}(x_{t,i} - \hat{x}_{t,i})^2}. \tag{4.3}$$

The DNN model is trained end-to-end using a joint optimization target that combines the forecasting and reconstruction loss:

$$Loss = Loss_{for} + Loss_{rec} \tag{4.4}$$

At inference, i.e. after the DNN model is fully trained and applied to new data, we record the loss, and calculate an anomaly score for each feature using a weighted sum of the forecasting and reconstruction loss. Concretely, given the forecasted values of a timestamp, $\vec{y}_t$, and the reconstructed value for the same timestamp, $\hat{\vec{x}}_t$, we define the anomaly score for feature i as the following:

$$s_i = \frac{(y_{t,i} - \hat{y}_{t,i})^2 + \gamma(x_{t,i} - \hat{x}_{t,i})^2}{1+\gamma}, \tag{4.5}$$

where γ is a hyper-parameter to combine the forecasting error and the reconstruction error. This way, we get an individual anomaly score for each feature, at every timestamp. In cases where we are interested in detecting anomalies at entity-level, i.e. across all features, we simply take the mean of all feature anomaly scores at a specific timestamp:

$$s_{entity} = \frac{1}{k}\sum_{i=1}^{k} s_i \tag{4.6}$$

In turn, the anomaly scores are fed to the Thresholding Selection Model, which decides a threshold and label a timestamp-feature combination (or only timestamp if at entity-level) as anomalous if its corresponding anomaly score exceeds the threshold.

4.4 Threshold Selection Model

The Threshold Selection Model is the second core component of our framework (see Figure 4.1). In summary, it receives the anomaly scores from the DNN model, fits a threshold, and predicts anomalies. Following good machine learning (ML) practice, we use anomaly scores from the training set to *decide* the threshold, and then *apply* that threshold to the anomaly scores of the test set. As we already

have all the data available (train and test), it is practically possible to include the test anomaly scores when fitting the threshold. However, from a real-case perspective, where one get inputs of the multivariate time-series in real-time, this would not be possible. Thus, in order to get a realistic evaluation of our framework's performance, we keep to the good ML practice.

There exist several approaches that could be employed to set the threshold. A common approach is to make Gaussian assumptions about the distribution of errors (e.g. [1] and [37]). However, if the parametric assumptions are violated, this approach may become problematic. As stated in "Research Issue 6" for outlier detection in data streams by Sadik and Gruenwald [33], no distribution assumptions should be made on the data. Therefore, we apply a method that satisfies this criteria, namely the non-parametric approach proposed in [17].

4.4.1 Non-Parametric Thresholding

The main motivation behind this method is that it achieves high performance with low overhead, and without statistical assumption regarding the distribution of errors. Given the anomaly scores $\vec{s}$ (anomaly scores for a relatively large number of timestamps) provided by the DNN model, it searches through a range of values and selects a threshold ϵ that is determined by:

$$\epsilon = \underset{\epsilon}{\text{argmax}} \frac{1}{|\vec{s}_a|} \left(\frac{\Delta\mu(\vec{s})}{\mu(\vec{s})} + \frac{\Delta\sigma(\vec{s})}{\sigma(\vec{s})} \right), \tag{4.7}$$

where

$$\begin{aligned} &(*) : \Delta\mu(\vec{s}) = \mu(\vec{s}) - \mu\left(\{s \in \vec{s} \mid s < \epsilon\}\right), \\ &(**) : \Delta\sigma(\vec{s}) = \sigma(\vec{s}) - \sigma\left(\{s \in \vec{s} \mid s < \epsilon\}\right), \\ &(***) : \vec{s}_a = \{s \in \vec{s} \mid s > \epsilon\}. \end{aligned} \tag{4.8}$$

$(*)$ is the mean of the anomaly scores that are smaller than ϵ, $(**)$ is the standard deviation of the anomaly scores that are smaller than ϵ, and $(***)$ are the anomaly scores greater than ϵ.

In terms, a threshold ϵ is found such that if all values above are removed, would cause the greatest percent decrease in the mean and standard deviation when compared to the mean and standard deviation of all values. The method also penalizes thresholds that produces a large set of anomalous vales ($|\vec{s}_a|$), to prevent overly greedy behavior [17].

The candidate thresholds are selected from the set

$$\{\mu(\vec{s}) + z\sigma(\vec{s}) \mid z \in Z\}, \tag{4.9}$$

where Z is an ordered set of positive values, representing the number of standard deviations above $\mu(\vec{s})$.

Once a threshold is fitted using the train anomaly scores, the same threshold is applied to the anomaly scores of the test set, and anomalies are predicted. Specifically, given m test anomaly scores, $\vec{s} = \{s_1, s_2, ..., s_m\}$, belonging to m timestamps, and the threshold ϵ, the Threshold Selection Model will output a vector $\vec{y} \in \mathbb{R}^m$, where

$$y_i = \begin{cases} 1, & s_i \geq \epsilon \\ 0, & s_i < \epsilon. \end{cases}, \tag{4.10}$$

Figure 4.5 shows an example output of the Threshold Selection Model.

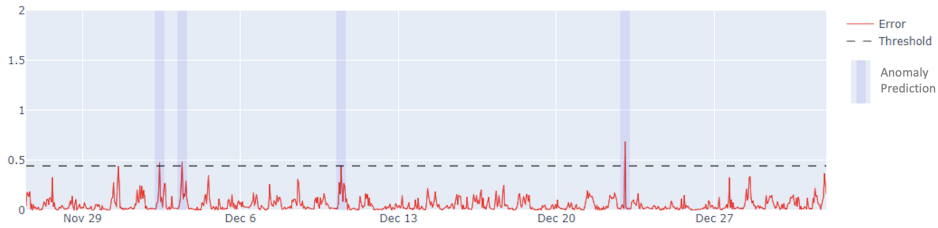


Figure 4.5: Example output of the Threshold Selection Model. On four occasions does the anomaly score exceed the threshold and anomalies predicted.

Chapter 5

Experimental Setup

This chapter covers the experiments we perform. Section 5.1 describes the multivariate time-series datasets we use, section 5.2 explains the different experiments in detail, and section 5.3 describes what metrics we use to evaluate the performance of our framework.

5.1 Datasets

Our main objective is to apply our framework on real-world KPI data of the telco domain, provided by Telenor. However, in order to verify the effectiveness of our framework, in addition to analyze its different components, we use three commonly-used benchmark datasets within anomaly detection for multivariate time-series, and compare results with SOTA methods. These datasets contains test sets where anomalies have been labeled by domain experts, and can therefore be used to get a numerical evaluation of our framework. Section 5.1.1 describes the benchmark datasets, while subsection 5.1.2 presents the dataset provided by Telenor, as well as describing the preprocessing and feature engineering we perform in order to make it applicable for our framework.

5.1.1 MSL, SMAP & SMD

The three benchmark datasets we use are the Soil Moisture Active Passive Dataset (SMAP), the Mars Science Laboratory Dataset (MSL), and the Server Machine Dataset (SMD), summarized in Table 5.1.

SMAP and MSL are two public spacecraft datasets from NASA's Jet Propulsion Laboratory [27]. The SMAP dataset originates from one of the first Earth observation satellites by NASA. The satellite uses an advanced radiometer in order to peer through clouds, vegetation and other features on Earth's surface. Thus, it can monitor water and energy fluxes, which leads to improved flood predictions and monitoring of droughts. The MSL dataset contains data from the Curiosity Rover Environmental Monitoring Station, collected during its adventures on Mars. Both MSL and SMAP consists of a single telemetry feature in addition to command information (information regarding the module to which a command was issued, whether it was sent or received, etc) that has been one-hot encoded and slotted into each time step [17].

The SMD dataset is a 5-week-long dataset from a large Internet company, collected and made public by Su et al. [2019]. SMD consists of three groups of entities, each with 28 different machines. It is concretized by the aforementioned authors that each of these machines should be trained and tested individually. Each machine consists of 38 continuous features representing different server-related counters, such as CPU load, network usage, memory usage, etc.

Dataset	**MSL**	**SMAP**	**SMD**
No. of attributes	55	25	38
Training subset size	58 317	135 183	708 405
Testing subset size	73 729	427 617	708 420
Anomaly rate %	10.72	13.13	4.16
Data description	Telemetry data: computational, radiation, temperature, power, activities, etc.		Server data: CPU load, network usage, memory usage, etc.

Table 5.1: Benchmark Datasets Information.

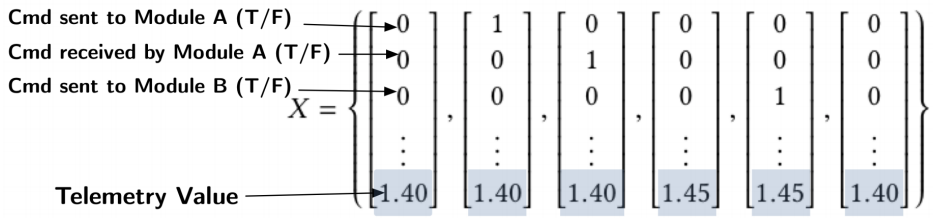


Figure 5.1: Example data from the MSL and SMAP datasets, where each telemetry value has corresponding one-hot encoded command information. Illustration adapted from [17].

5.1.2 Telenor Dataset

Overview

The dataset is provided by *TDBU*, and consist of data coming from the **Radio Access Network** (RAN). RAN is the part of a telecom system that connects individual devices (phones, computers, etc) to the **Core Network**. The Core Network is the central elements of the telecom system which provides services to customers connected to the network, such as routing of calls and downloading/uploading data. The data includes different types of RAN which operate on different technologies, namely GRAN/GERAN for 2G technology, UTRAN for 3G technology and EUTRAN for 4G technology.

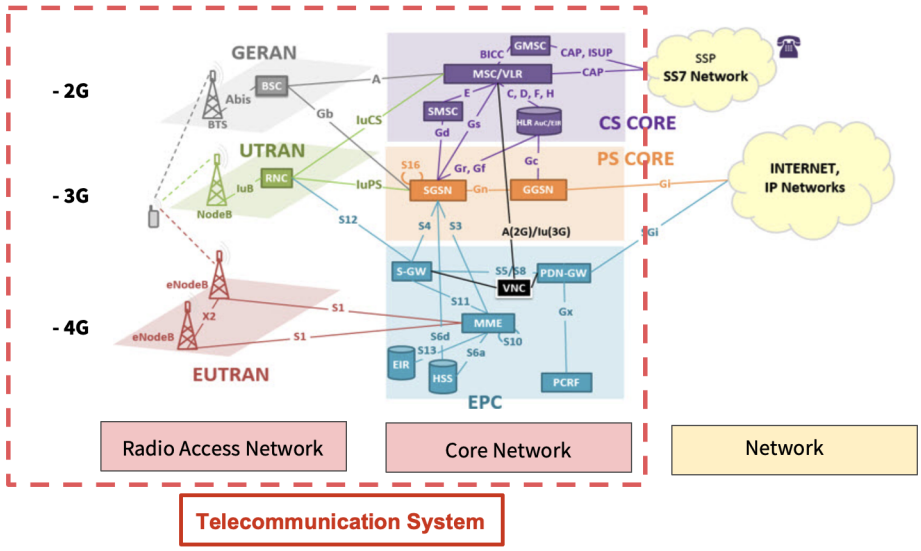


Figure 5.2: Illustration of the different parts in a telecommunication system.

The data is produced by multiple base stations, or *sites* (we will refer to them as such), that are spread across a given geographical area. A site is divided into three sectors, each covering a 120°area. A sector has multiple *cells*, and cells run on different frequencies. Cells of lower frequencies are part of the so-called *coverage layer*, which has longer range and covers larger areas, but provides worse signals. Cells of higher frequencies are part of the *capacity layer*, which is used by users that are closer to the site, thus giving a better signal. Depending on the demand, cells of different frequencies can be activated (turned on) or deactivated (turned off) within a sector. Figure 5.3 shows a high-level view of the different

parts of a site.

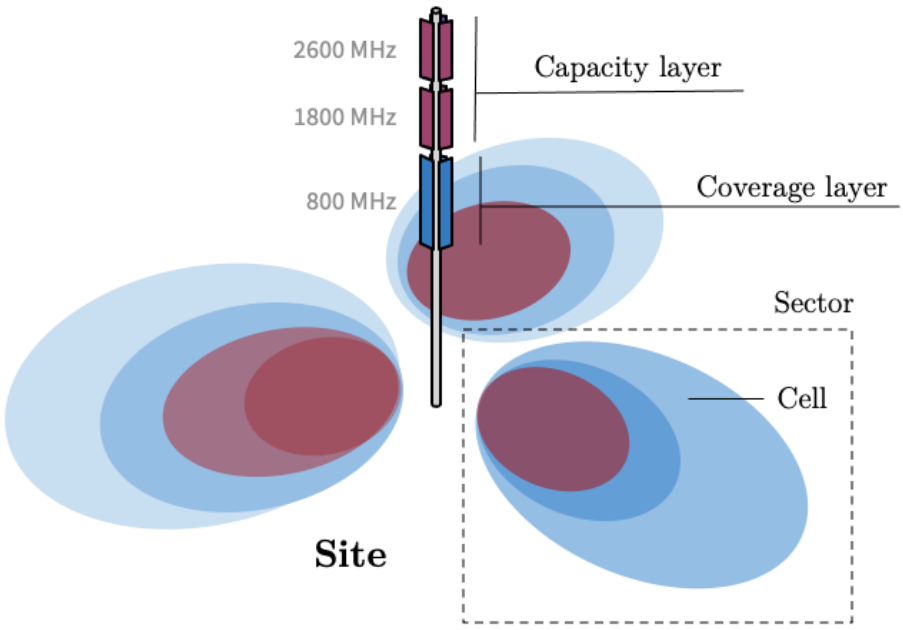


Figure 5.3: A site contains 3 sectors, each of which can have multiple cells. The coverage layer (cells of lower frequencies) cover larger areas but have lower quality signal, while the capacity layer (cells of higher frequencies) have a higher quality signal but smaller range.

Concretely, our data is collected from multiple sites located in Copenhagen, Denmark. It consists of numerous counters, representing different Key Performance Indicators (KPIs). The raw data is recorded at cell-level, meaning that each instance (row) in the data corresponds to a particular cell. In reality, whenever a cell is used (e.g. to connect a call), information regarding the event is recorded using the KPI counters. However, in our data, the time resolution is hourly, meaning that all recorded values for a counter within an hour are aggregated into one number. Each row is marked with a timestamp, which specifies the date and hour of the row.

We choose a subset of the KPI values to serve as the features of our data. These features were picked based on discussion with telco domain experts, filtering out the ones not relevant for our purpose of detecting anomalies. Table 5.2 provides a description of each feature.

Name	Description
avail_period_duration	Minutes that cell was available
unavail_unplan_nom	Minutes that cell was unplanned unavailable
unavail_unplan_denom	Defined as *avail_period_duration* − *unavail_plan_denom*
mcdr_denom	Total number of voice-related attempts
msdr_denom	Total number of data-related attempts
msdr_nom_s	Number of data-related setup failures
msdr_nom_d	Number of data-related drops
ho_denom	Total number of handovers
ho_nom	Number of handover failures
impact_nom	Number of distinct users with error message

Table 5.2: Description of the features in the Telenor dataset.

Feature Engineering

The data goes through an extensive pipeline of preprocessing and feature engineering, where we obtain multivariate time-series of the format $X \in \mathbb{R}^{n \times k}$, ready to be input to our framework. Here, we briefly describe the main parts of this pipeline.

Imputing Missing Values

We follow a "hierarchical" approach to impute missing values for a cell. The main motivation behind this approach is that the cells are distributed across a relatively large geographical area. Cells of different sites do not necessarily behave similarly, as the user traffic will vary depending on the location of the site. Also, as values are varying with time, one should use values of the same (or nearby) timestamp to impute values. Therefore, we first group all cells such that each group contains rows (cell data) of the same timestamp, site and sector, i.e. each group contains cells that are identical with respect to both time and location. Missing values are imputed using only values from rows within the same group. Concretely, the median is used to impute missing values for columns with discrete values, while the average is used for continuous columns. However, as there may be only one row in such a group, we group the rows again, this time based on only timestamp and site (one step down in the hierarchy) and perform imputations again. We repeat this process, each time making the groups more general, until there are no missing values left.

Aggregation of Cells

As the data contains cells from sectors belonging to many different sites, there are initially multiple rows with the same timestamp. A possible way of converting

it to a valid multivariate time-series, in which each timestamp is unique, is to isolate each individual cell. However, as cells may be activated and deactivated based on the demand, there will often be a substantial amount of timestamps for which a cell has no values. To overcome this problem, we aggregate all cells within the same sector. This converts the data to be on sector-level, where each row represents the average values for all cells in a sector, for a specific timestamp.

Concatenation of Sectors

Sectors of the same site share the same geographical location and are likely to share similar user traffic patterns. Therefore, we combine each of the three sectors within a site by concatenating their features (columns) for each timestamp, as illustrated in Figure 5.4. Each of the original KPIs is now present three times for every row, each corresponding to one of the three sectors.

Now, a row represents all counters of the three sectors of a site, at a particular timestamp. We isolate each site, such that each site constitutes its own multivariate time-series of the format $\boldsymbol{X} \in \mathbb{R}^{N \times k}$. Specifically, $N = 10200$ (each site has data from a time span of slightly more than a year), $k = 30$ (3×10), and a total of 34 sites. For each site, we use the first 90% as training data, and the last (most recent) 10% as test set.

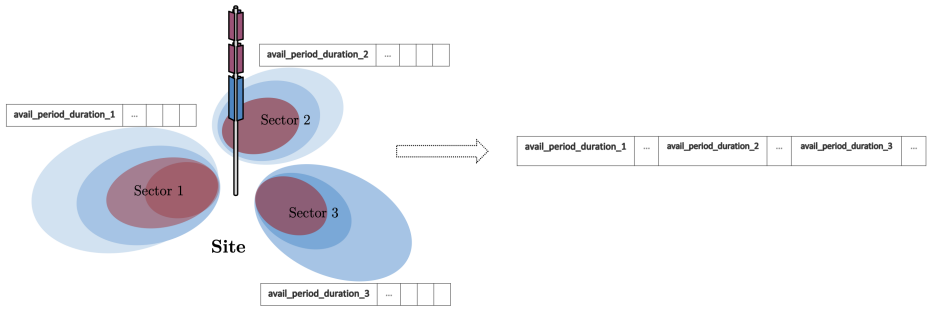


Figure 5.4: Illustration of how the values of each sector is concatenated column-wise.

5.1.3 Feature-wise Normalization

Different features will typically have values on different scales (e.g. one sensor might have values in the range [0.0, 0.5], while another in [100, 200]. To improve robustness of our framework, we perform normalization individually for each feature of the multivariate time-series. This data normalization is applied to

both training and testing set, for all datasets. Specifically, the values for each feature i is normalized using the minimum and maximum values from the training data:

$$\tilde{x}_i = \frac{x_i - \min(x_{i,\text{train}})}{\max(x_{i,\text{train}}) - \min(x_{i,\text{train}})}, \tag{5.1}$$

where $\min(x_{i,\text{train}})$ and $\max(x_{i,\text{train}})$ is the minimum and maximum value, respectively, for feature i in the training set.

5.2 Experimental Plan

In this section, we describe the experiments we perform, both on the benchmarking datasets, and on the TDBU dataset.

5.2.1 Experiments on the Benchmark Datasets

Recall from subsection 5.1.1 that MSL and SMAP only has one continuous feature, while the rest are 0s and 1s. Therefore, we make the DNN model take all features as input, but only forecast and reconstruct the one continuous feature. For SMD, on the other hand, all features are continuous, so the model outputs predictions for all.

We use the same model configuration for the three benchmarking datasets. The sliding window size is set to $n = 150$ for MSL and SMAP, and $n = 100$ for SMD. The kernel size of the 1-D convolution is $d_{conv} = 7$, while the hidden dimension of the GRU layer (d_{gru}), the fully-connected layers in the Forecasting Model (d_{for}), and the Reconstruction Model (d_{rec}), are set to 150. γ, which combines the forecasting loss and the reconstruction loss, is set to 1.0. As mentioned, MSL, SMAP and SMD has pre-defined training and test sets (see Table 5.1). We set aside 10% of the training data as a validation set, which is used to set the parameters above. The labeled ground-truth anomalies in the test set are at entity-level, meaning that a complete timestamp is either labeled as anomalous or non-anomalous. Therefore, we use anomaly scores at entity-level, as shown in Equation 4.6.

We use the Adam optimizer [19] with a learning rate of 0.001 to train our model for 50 epochs on MSL and SMAP, and 10 epochs for each subpart of SMD. All configurations are summarized in Table 5.3. Each run is repeated 10 times, and the average performance (Precision, Recall, F1) is reported along with the standard deviation (section 6.1).

	MSL	SMAP	SMD
n	150	150	100
d_{conv}	7	7	7
d_{gru}	150	150	150
d_{for}	150	150	150
d_{rec}	150	150	150
γ	1.0	1.0	1.0
learning rate	0.001	0.001	0.001
epochs	50	50	10

Table 5.3: Training configuration for the benchmark datasets.

Ablation Study

In order to analyze the effectiveness and necessity of the core components of the DNN model, we conduct an ablation study where different components are removed. Concretely, we run experiments on the benchmark datasets, with four different compositions of the DNN model:

(i) *w/o feature*: Removing the feature-oriented GAT layer

(ii) *w/o time*: Removing the time-oriented GAT layer

(iii) *w/o forecasting*: Removing the Forecasting Model

(iv) *w/o reconstruction*: Removing the Reconstruction Model

To examine the influence of the two GAT layers, we disable the feature-oriented GAT layer and the time-oriented GAT layer one at the time, denoted as *w/o feature* and *w/o time*, respectively. In addition, to analyze the effect of combing a forecasting-based model and a reconstruction-based model, we remove the Forecasting Model (denoted *w/o forecasting*) and then the Reconstruction Model (*w/o reconstruction*), making the model output only reconstructions and only forecasted values, respectively.

We use the same model parameters and training configurations as previously (Table 5.3). Each composition of the DNN model is run ten times on MSL and SMAP, and three times on SMD (SMD is considerably more expensive), and the average performance along with standard deviation is reported.

Evaluating the Threshold Selection Model

In order to evaluate the Threshold Selection Model, we compare it to a brute-force thresholding method that searches across multiple thresholds, and chooses

the one that yields the highest F1 score. As the number of candidate thresholds grows, this method will approach the theoretical upper limit for the best possible performance, given the output from the DNN model. In our experiments, we choose the number of candidate thresholds such that it balances the accuracy with the computational complexity of the search.

5.2.2 Experiments on the Telenor Dataset

For all our experiments on the TDBU dataset, we use a train/test split of 0.9/0.1, making the training data consist of values from the 10^{th} of November 2019 to 25^{th} of November 2020, while the test data is from the November 26^{th} 2020 to December 31^{st} 2020. We use 10% of the training data as validation set, which we used to find optimal model and training configuration. Specifically, we use a window size of $n = 168$ (corresponding to one week), while d_{conv}, d_{gru}, d_{for}, d_{rec} and γ are set to the same as for the benchmarking datasets (see Table 5.3). We train for 50 epochs with a learning rate of 0.001.

Main Experiments

Our main experiments consists of employing our proposed framework on each of the 34 sites present in the TDBU data. Recall from subsection 5.1.2 that we column-wise concatenate the features from all three sectors within a site, on the assumption that sectors within the same site will have similar behaviour as they share the same geographical location. Every site is trained individually, and anomalies are predicted on their respective test sets. For a real-life anomaly detection system in the telco domain it is useful that anomalies are predicted at feature-level (in contrast to entity-level), as this provides important insight for anomaly diagnosis. Therefore, we output the anomaly scores *per feature* (Equation 4.5), and make the Threshold Selection Model predict anomalies for each individual feature.

Transfer Learning

The concept of transferring knowledge originates from educational psychology, and psychologist Charles M. Judd's *theory of generalization* states that learning to transfer is the result of generalization of experience [49] [6]. Transfer learning has become an important tool in machine learning [41], where a model developed for a task is reused as the starting point for a model on a second task. We consider there to be several incentives for investigating the use of transfer learning in our scenario. Firstly, as each site is an entity of the same type, it is reasonable to believe that knowledge gained from training on one site could be beneficial when training and employing our framework on another site. Secondly, each site has

limited data (a one-year period with hourly resolution), making transfer learning a way of preventing any potential problems related to insufficient training data. Lastly, the exists a very high number of sites, making it unpractical having to train and employ a separate framework on each and every one.

In order to address this, we perform a transfer learning-inspired experiment. We use a form of transfer learning that deviates slightly from the regular. In regular transfer learning, a model that has been trained on one task is trained on some other task, using the already-trained model as starting point, and then applied to unseen data from the newer task. In this way, the model is transferring knowledge gained from the previous task, but is *fine-tuned* on the present task before it is employed. In our scenario, if we were to perform transfer learning in this manner, we would have to fine-tuned the model on each site, essentially losing the point of not having to train a separate model for each site. Instead, we train the same model on each of the 34 sites, one after the other, and only then, making it predict anomalies for each site individually. Effectively, this is equivalent to concatenating the training data from all sites into one large training set, but keeping all tests the same as previously. If such a model is successful, it represents a *generalized* model, capable of being employed to numerous sites without having to be customized to each individual site.

Site-based versus Sector-based

As explained in subsection 5.1.2, we column-wise concatenate all three sectors within a site, with the intuition that sectors of the same site will correlate. To evaluate this argument, we additionally employ our framework on individual sectors (sector-based), and compare results against those from the main experiments (site-based). That is, for a site A that consists of three sectors $\{A_1, A_2, A_3\}$, we train individually on A_1, A_2 and A_3, and compare the results with the result of A from the site-based model.

Intuitively, we expect that if there exists correlation in traffic patterns amongst the sectors, the model can benefit from forecasting and reconstructing all of them, making the site-based more accurate than the sector-based. Contrarily, if sectors' patterns strongly deviate from each other, we hypothesis that the site-based model may be mislead, making the sector-based model a better choice.

GATv2: Dynamic Attention

Recently, Brody et al. [2021] proved that the original GAT (proposed by Veličković et al. [2018]) can only compute a restricted kind of attention (which they refer to as *static*) where the ranking of attended nodes is unconditioned on the query node. That is, the ranking of attention weights is global for all

nodes in the graph, a property which the authors claim to severely hinders the expressiveness of the GAT. In order to address this, they introduce a simple fix by modifying the order of operations, and propose GATv2, a *dynamic* attention variant that is strictly more expressive that GAT.

Brody et al. [2021] state that the main problem in the standard GAT scoring function (Equation 2.23) is that the learned layers $\boldsymbol{W}$ and $\vec{a}$ are applied consecutively, and thus can be collapsed into a single linear layer. Instead, they modify the original GAT attention scoring function by applying the $\vec{a}$ layer after the non-linearity and the $\boldsymbol{W}$ layer after the concatenation, effectively applying an MLP to compute the score for each node pair:

$$\begin{aligned} \text{GAT (Veličković et al. [2018])}: \quad & e_{ij} = \text{LeakyReLU}\left(\vec{a}^T[\boldsymbol{W}\vec{h}_i \oplus \boldsymbol{W}\vec{h}_j]\right) \\ \text{GATv2 (Brody et al. [2021])}: \quad & e_{ij} = \vec{a}^T\text{LeakyReLU}\left(\boldsymbol{W}[\vec{h}_i \oplus \vec{h}_j]\right) \end{aligned} \tag{5.2}$$

Brody et al. [2021] perform an extensive evaluation and show that GATv2 outperforms GAT across a wide variety of benchmark datasets. Motivated by their success, we incorporate GATv2 into the feature-oriented GAT layer, and perform experiments on the TDBU dataset.

5.3 Evaluation Details

The TDBU dataset contains no labeled anomalies, so we will evaluate our results using visualizations and discussions with domain experts. The benchmark datasets (MSL, SMAP and SMD) have ground-truth labels for their test set. Thus, we use precision, recall and F1-score to evaluate our framework on these datasets, defined as:

$$Precision = \frac{TP}{TP + FP}, \qquad Recall = \frac{TP}{TP + FN}, \qquad F_1 = \frac{2 \times Precision}{Precision + Recall},$$

where TP denotes *true positives*, FP denotes *false positives*, and FN denotes *false negatives*.

It is often the case that anomalies occur in the form of contiguous segments. In real-life, it is usually acceptable to trigger an alert for any point within the anomaly segment. Therefore, we follow the strategy proposed in [47], which seems to have become the convention when evaluating anomaly detection methods [40] [32] [48] [15]. If any point in a ground-truth anomaly segment is detected, we mark the segment as correctly classified, and all points in this segment are treated as if they was detected correctly. Meanwhile, any point outside the anomaly segment is treated as normal. This evaluation strategy is illustrated in Figure 5.5.

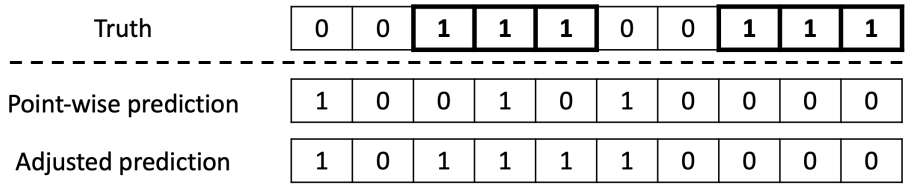


Figure 5.5: Illustration of the evaluation strategy. For 10 contiguous points in a time-series, we have three rows; first row defines the ground truth, the second is the point-wise anomaly predictions, and the last row indicates the adjusted predictions according to the evaluation strategy.

Chapter 6

Results and Discussion

In this chapter we present and discuss the results on the experiments we perform. Section 6.1 compares the performance of our framework on the benchmarking datasets, to that of existing SOTA methods, and demonstrate the impact of key components in our framework through an ablation study. In section 6.2 we present and discuss the results from the experiments run on the TDBU dataset.

6.1 Results on Benchmark Datasets

6.1.1 Comparison with SOTA

Table 6.1 displays the performance of our framework on the benchmark datasets, averaged over 10 runs. The other SOTA methods included are LSTM-NDT [17], LSTM-VAE [29], DAGMM [50], OmniAnomaly [40], MTAD-TF [15] and MTAD-GAT [48], all described in chapter 3.

Compared to the best SOTA methods, our framework performs moderately worse on SMAP and SMD, and achieves the best F1 score on MSL with a 4% improvement over the best SOTA performance. Our framework performs reasonably well on all three datasets, which indicates that it has satisfactory generalization capabilities across different scenarios. In addition, Table 6.2 shows the F1 score with standard deviation of the 10 runs included. The standard deviations are overall small, telling us that our framework is stable and not too sensitive to differences in initialization or randomness in the training phase.

	MSL			SMAP			SMD		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
LSTM-NDT	0.5944	0.5374	0.5640	0.8965	0.8846	0.8905	0.5684	0.6438	0.6037
LSTM-VAE	0.5257	0.9546	0.6780	0.8551	0.6366	0.7298	0.7922	0.7075	0.7842
DAGMM	0.5412	**0.9934**	0.7007	0.5845	0.9058	0.7105	0.5835	0.9042	0.7093
OmniAnomaly	0.8867	0.9117	0.8989	0.7416	**0.9776**	0.8434	0.8334	**0.9449**	0.8857
MTAD-GAT	0.8754	0.9440	0.9084	0.8906	0.9123	**0.9013**	-	-	-
MTAD-TF	0.9043	0.8988	0.9015	0.9779	0.8192	0.8916	**0.9045**	0.9048	**0.8940**
Ours	**0.9478**	0.9352	**0.9410**	**0.9891**	0.7738	0.8679	0.8957	0.8702	0.8538

Table 6.1: Performance of our framework on the benchmark datasets, compared to SOTA methods.

In summary, these results verifies the effectiveness of our framework, which was the objective of our experiments on the benchmark datasets.

6.1.2 Ablation Study

The results of the ablation study (described in subsection 5.2.1) are shown in Table 6.3. For each composition of the DNN model, we report the average performance and standard deviation across multiple runs (10 for MSL and SMAP, 3 for SMD), and compare it to the original performance of the complete DNN model.

The impact of the feature-oriented GAT layer

The model denoted *w/o feature* represents the DNN model where the feature-oriented GAT layer is removed. Interestingly, we observe that the average F1 score actually increased by 0.8% on MSL and only decreased by 1.3% on SMAP. However, on SMD, we notice a significant decrease of 10.9%, from 0.8538 to 0.7447. We suspect that the removal of the feature-oriented GAT layer has impacted the datasets differently because of the fundamental dissimilarity between the SMD dataset and the two other datasets. Recall from subsection 5.1.1 that MSL and SMAP contains mostly one-hot encoded data. In our model, each entry in the one-hot encoding is then treated as a separate feature, for which the feature-oriented GAT layer attempts to find correlations between. Seeing as most features are discrete values of 0 or 1, there might be a lack of correlations. In SMD, however, all features are continuous streams of values, creating a larger potential for feature-wise dependencies to capture and exploit. Following this line of argumentation, it appears likely that the feature-oriented GAT layer could

Dataset	Metric	
MSL	Precision	0.9478 ± 0.0172
	Recall	0.9352 ± 0.0282
	F1	0.9410 ± 0.0078
SMAP	Precision	0.9891 ± 0.0072
	Recall	0.7738 ± 0.0311
	F1	0.8679 ± 0.0175
SMD	Precision	0.8957 ± 0.0106
	Recall	0.8702 ± 0.0207
	F1	0.8540 ± 0.0209

Table 6.2: Results on the benchmark datasets, with standard deviations included.

	MSL	SMAP	SMD
Complete DNN Model	0.9410 ±0.0078	0.8679 ±0.0175	**0.8540** ±0.0209
w/o time	0.9430 ±0.0201	0.8654 ±0.0206	0.7587 ±0.0146
w/o feature	**0.9486** ±0.0078	0.8549 ±0.0294	0.7447 ±0.0166
w/o forecasting	0.9330 ±0.0106	**0.8934** ±0.0103	0.7838 ±0.0623
w/o reconstruction	0.9274 ±0.0102	0.8287 ±0.0393	0.7913 ±0.0045

Table 6.3: Ablation study results showing F1 score ± stdev.

be highly beneficial for data where inter-feature relationships exists, but that it could add unnecessary complexity and even be damaging if the data is lacking correlations between its features.

The impact of the time-oriented GAT layer

The model denoted *w/o time* represents the DNN model where the time-oriented GAT layer is removed. The results show negligible effects on MSL and SMAP, where the average F1 score changes by only $+0.2\%$ and -0.3%, respectively, while there is a considerable decrease of 9.5% on SMD. As with the feature-oriented GAT layer, we think it is reasonable to believe that this is caused by the nature of the datasets. Continuous values are more likely to have temporal patterns that the time-oriented GAT layer can learn and use to its advantage, making it beneficial for SMD, but not necessarily for MSL or SMAP.

The impact of the Forecasting Model

The model denoted *w/o forecasting* represents the DNN model where the forecasting is removed, thus giving insight into the impact of the Forecasting Model. We see only a slight decrease of 0.8% in average F1 score on MSL, from 0.9410 to 0.9329, and that the average F1 score actually increases with 2.6% for SMAP, from 0.8679 to 0.8934. For SMD, however, we see a decrease of 7%, from 0.8538 to 0.7838. Once again, we see that the impact is very different for SMD compared to MSL and SMAP. The reason for this is not obvious, but we hypothesize that it can be because MSL and SMAP only has one single feature that the DNN model predicts values for, making the impact of having *both* forecasting and reconstruction less significant. For SMD, on the other hand, the DNN model predicts all 38 input features, and it is more likely that the Forecasting Model and the Reconstruction Model work in a complementary way. In addition, when the Forecasting Model tries to predict the values of the next timestamp, it is reasonable to believe that it will emphasize information from certain timestamps more than others, for instance from the most recent ones, or timestamps that are otherwise related. If values from those timestamps happen to be noisy, the Forecasting Model might become deluded, causing it to predict inaccurately. For MSL and SMAP, where only one feature is predicted, noisy values will be more damaging, as the error for that feature make up the entire anomaly score. For SMD, however, the error of several feature predictions are combined, making each individual feature less important.

The impact of the Reconstruction Model

The model denoted *w/o reconstruction* represents the DNN model where the reconstruction is removed, thus giving insight into the impact of the Reconstruction Model. For MSL and SMAP we see a decrease of 1.4% and 3.9% in average F1 score, respectively, while for SMD we observe an 6.2% decrease. The fact that the model consistently perform worse on all three datasets imply that the Reconstruction Model is an essential part of the DNN model. In contrast to the Forecasting Model, which predicts one step ahead, the Reconstruction Model tries to learn the distribution of entire time-series windows, and is therefore expected to be less sensitive to occasional noise and perturbation. When the Reconstruction Model is removed, all predictions are made by the Forecasting Model, which, as discussed earlier, is more easily mislead by noisy values.

Conclusion of the Ablation Study

Our ablation study reveals interesting and somewhat surprising results. We see that the usefulness of the two GAT layers is highly dependent on the data and

the nature of its features. Additionally, the Reconstruction Model seems to be generally of higher importance than the Forecasting Model, possibly because the former is more robust to noise and perturbation. For MSL and SMAP, the impact of each core component is less signficant, and both actually achieved a better performance for a composition of the DNN model where one of the core components were removed. However, for SMD, we observe that the DNN model performs considerably worse when any of the core components are removed. The TDBU dataset shares much more resemblance to SMD than MSL and SMAP, as it consists of many continuous streams of values that are expected to have both feature-wise correlations and temporal dependencies. Therefore, we use the full implementation of our framework when running experiments on the TDBU data.

6.1.3 Evaluation of the Threshold Selection Model

Table 6.4 shows comparison between the Threshold Selection Model and the brute-force method that searches through a large number of possible thresholds (denoted $F1_{best}$). As the brute-force method picks the threshold with the highest F1 score, this provides an approximation of the upper performance limit, given the output from the DNN model. As we can see, the Threshold Selection Model performs extremely well on MSL, with a F1 score only slightly lower than $F1_{best}$, but lies 7.5% and 9.9% below $F1_{best}$ on SMAP and SMD, respectively. Keep in mind that the brute-force method checks the F1 score for every threshold that it attempts, something that will never be possible in a real-case scenario. Therefore, despite the difference in performance for SMAP and SMD, we would say that the Threshold Selection Model performs reasonably well. Additionally, the consistently high F1 scores of the brute-force method on all three datasets verifies that the DNN model is performing very well. However, the difference in performance between the Threshold Selection Model and the the Brute-Force Search demonstrates that the former has room for improvement, and demonstrate that the thresholding method is a crucial part of an unsupervised anomaly detection system.

	MSL	SMAP	SMD
$F1$ using Threshold Selection Model	$0.9410_{\pm 0.0078}$	$0.8679_{\pm 0.0175}$	$0.8540_{\pm 0.0209}$
$F1_{best}$ using Brute-Force Search	$0.9542_{\pm 0.0036}$	$0.9428_{\pm 0.0201}$	$0.9530_{\pm 0.0032}$

Table 6.4: Comparison of Threshold Selection Model versus Brute-Force Threshold Search. F1 score ± stdev showed.

6.2 Results on the Telenor Dataset

In this section we present, visualize and discuss the results from the experiments performed on the TDBU dataset. As the data is not labeled with ground-truth anomalies, we evaluate our framework mainly through visual inspection, in addition to feedback from domain experts. Section 6.2.1 covers the results from our main experiment. Section 6.2.2 visualizes the learned attention matrices of the two GAT layers. In subsection 6.2.3 we analyse the effect of transfer learning, while in subsection 6.2.4 we examine the consequence of combining sectors of the same site.

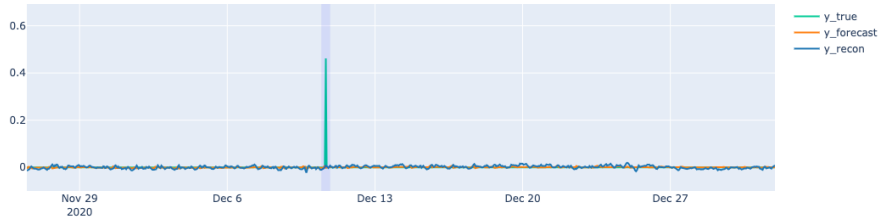
6.2.1 Main Experiments results

As explained in subsection 5.2.2, our main experiment consists of employing our framework on multiple sites. The DNN model forecasts and reconstructs all input features, and outputs anomaly scores per feature. In turn, these anomaly scores are fed to the Threshold Selection Model, that predicts anomalies for each individual feature.

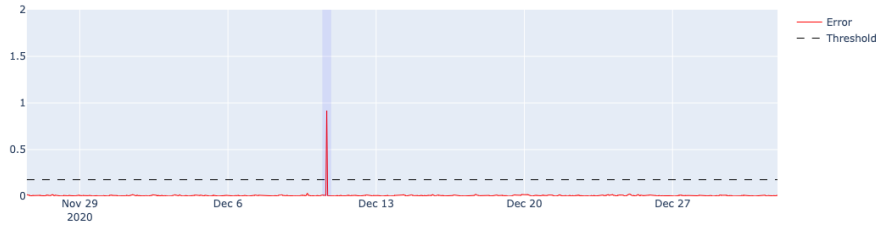
Sudden spikes

Some features in the TDBU dataset represent counters that remains constant for the vast majority of the time. If these features deviates from their normal value, it will be detected by our framework, and likely be marked as an anomaly. Figure 6.1 illustrates such a scenario, for the feature *unavail_unplan_nom*, which represents the average number of minutes that the cells in a sector is unavailable during the hour. The topmost figure shows the forecasted values from the Forecasting Model (denoted *y_forecast*) and the reconstructed values from the Reconstruction Model (denoted *y_recon*), along with the actual values of the feature (denoted *y_true*), while the figure in the middle shows the corresponding anomaly scores (denoted *Error*). We visualize the anomalies detected by our framework by shading a small area around it. The lower figure shows the anomaly scores of *unavail_unplan_nom* on the train set. These are the anomaly scores that are input to the Threshold Selection Model, which fits a threshold to capture the peaks, and then applies the fitted threshold to the anomaly scores of the test set.

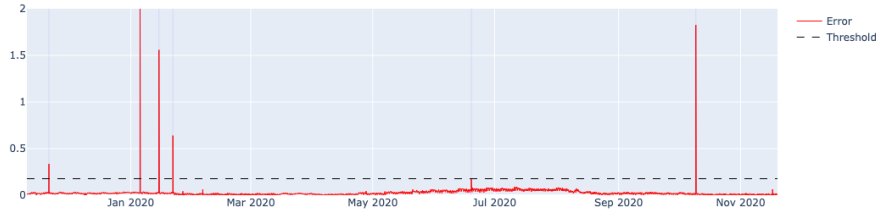
Such cases may also arise for features that are not necessarily constant most of the time, but keep within a limited range of values for a period of time. In general, sudden spikes are likely to be detected by our framework. Figure A.2 illustrates a few other cases where one or more sudden spikes are marked as anomalies.



(a) The sudden spike is easily detected by our framework.



(b) All values that exceeds the threshold found by the Threshold Selection Model are marked as anomalies.



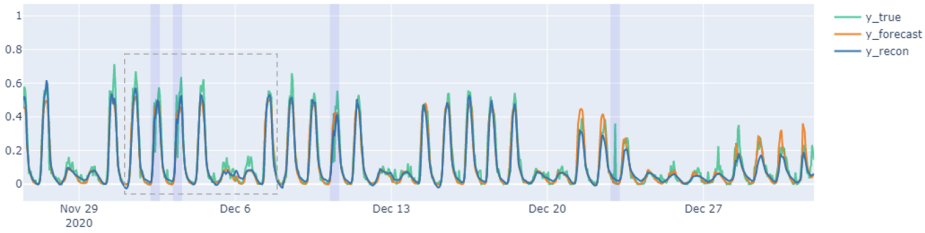
(c) The anomaly scores of the train set are used to set the threshold, which in turn are applied to the test anomaly scores.

Figure 6.1: *unavail_unplan_nom*.

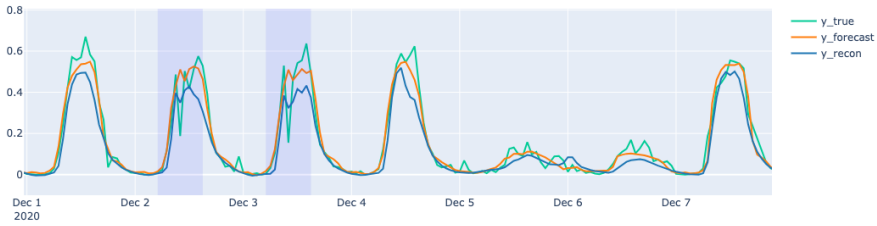
Anomalies in Complex Temporal Patterns

Sudden spikes that deviate substantially from the majority of values constitute a relatively easy scenario, for which far less sophisticated methods than ours are expected to manage. Rather, detecting anomalies in features that express more complex patterns are of higher interest, but consequently more difficult. Figure 6.2a is an illustrative example, showing the feature *mcdr_denom*, that represents the total number of voice-related attempts. We notice that there is a

clear weekly trend (five high tops corresponding to each of the weekdays, followed by two minor ones for the weekend), and that our framework has learned these temporal patterns very accurately. At first sight, it might seem like our framework is predicting the leftmost anomalies because of high peaks, something that in this case would be a false alarm. However, when looking closer (Figure 6.2b), wee see that the anomalies are actually caused by a sudden drop in the actual value. We see the same drop for the anomaly in the middle. For the rightmost anomaly, we observe a clear spike that occurs at an unusual time, and the framework rightfully triggers an alert. This spike actually corresponds to midnight on Christmas Eve, which explains the sudden increase in voice-related attempts. In addition, we observe that the traffic following Christmas Eve deviates from the normal weekly pattern, but that our model is able to dynamically adapt and continue to predict accurately.



(a) The temporal patterns are accurately modeled by our framework. On four occasions does it trigger an anomaly alert.

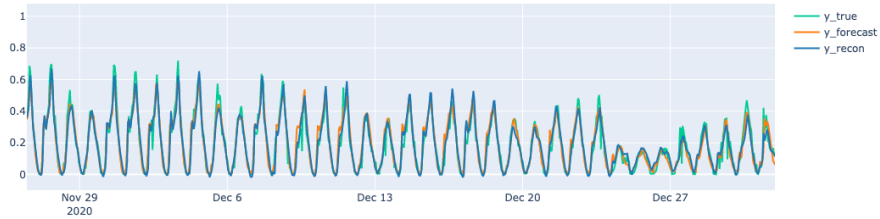


(b) Inspection of the the marked area in Figure 6.2a reveal that these anomalies are caused by an abnormal drop.

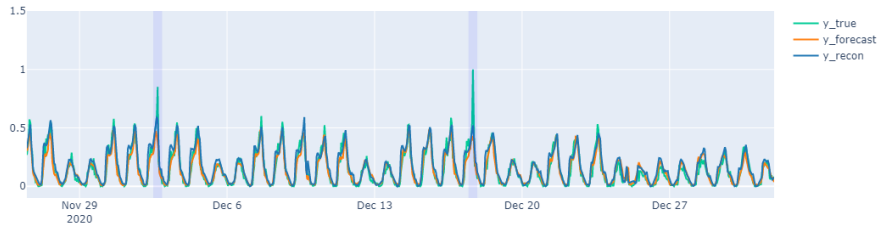
Figure 6.2: Illustrating the complex but clear temporal patterns present in *mcdr_denom* for a particular sector.

It is the time-oriented GAT and the GRU layer that is responsible for learning time-wise relationships and exploiting temporal patterns to make better forecastings and reconstructions. We note that our model, in general, is doing a good job

at forecasting and reconstructing values that follow complicated temporal patterns, indicating that the time-oriented GAT and the GRU layer are successfull in their task. Figure 6.3 shows a few other examples where our framework learns complex temporal patterns very precisely.



(a) *ho_denom*. No anomalies predicted by our framework.



(b) *mcdr_denom*. Two abnormally high spikes make our framework trigger anomalies.

Figure 6.3: Examples of features for which our model accurately learns complex temporal patterns.

Feature-wise Contextual Anomalies

As we have seen, the anomalies our framework detects are caused by abnormal values in the data itself. However, these anomalies are not necessarily considered anomalous from a business-perspective. For example, the spike from Figure 6.1 represents a sudden increase in the average number of minutes for which cells in a sector are unplanned unavailable. From the business-perspective, this may be acceptable and not even a cause of concern, *unless* certain other counters show abnormal behaviour as well. Thus, to decide whether a feature actually is an anomaly of actual interest, one often needs to take the other features into account, i.e. one must consider the *feature-wise context* of an anomaly. In order to address this, we additionally make our framework extract anomalies that are aligned across multiple features, illustrated in Figure 6.4. In this example, we

see that *unavail_unplan_nom* spikes for all three sectors (denoted using *_1*, *_2*, *_3*), something that alone would not be considered an anomaly. However, as our framework simultaneously detects anomalies for *msdr_denom*, measuring the number of data-related attempts, and *impact_nom*, measuring the number of users that received an error message, this should be treated as a true anomaly. Thus, this is an example of feature-wise contextual anomaly. More examples of such anomalies can be found in Appendix A.

Failure Cases

Our framework is not perfect, and there will be true anomalies that it fails to detect. One class of anomalies that we found our framework struggling to detect is *time-wise contextual* anomalies. These are anomalies that are considered anomalous within a local time window, but otherwise not. The Threshold Selection Model fits the threshold to the anomaly scores of the training set, and applies that same threshold to the entire test set. This enables scenarios in which the threshold is set too high, creating the possibility for true negatives. An example of this for *msdr_nom_d*, representing the number of data-related drops, is shown in Figure A.3. Because of a few very large spikes in the training set, the Threshold Selection Model fits a threshold that is higher than desired, causing it to fail to detect the second spike in the test set. Compared to the entire training set, this spike is small. However, with respect to the more recent context, it represents an abnormal value that should have been detected, thus constituting a time-wise contextual anomaly.

6.2.2 Visualization of the Attention Matrices

To further analyze the impact of the feature-oriented GAT layer and the time-oriented GAT layer, we extract their attention matrices and visualize the attention weights.

Temporal Attention

Recall that the attention matrix of the time-oriented GAT layer is an $n \times n$ matrix (n is the size of the input window), where each row represents how much the values at a timestamp should attend to the values of any other timestamp (including itself). Figure 6.5 shows the attention weights for the *last* timestamp (the bottommost row), when the time-oriented GAT layer was fed with a 1-week input window (168 hours, same as we used in training). For each of the 168 input timestamps, there is a corresponding attention weight, which represent how much attendance should be given *by* the last timestamp *to* a particular other timestamp. The rightmost attention weights correspond to the most recent

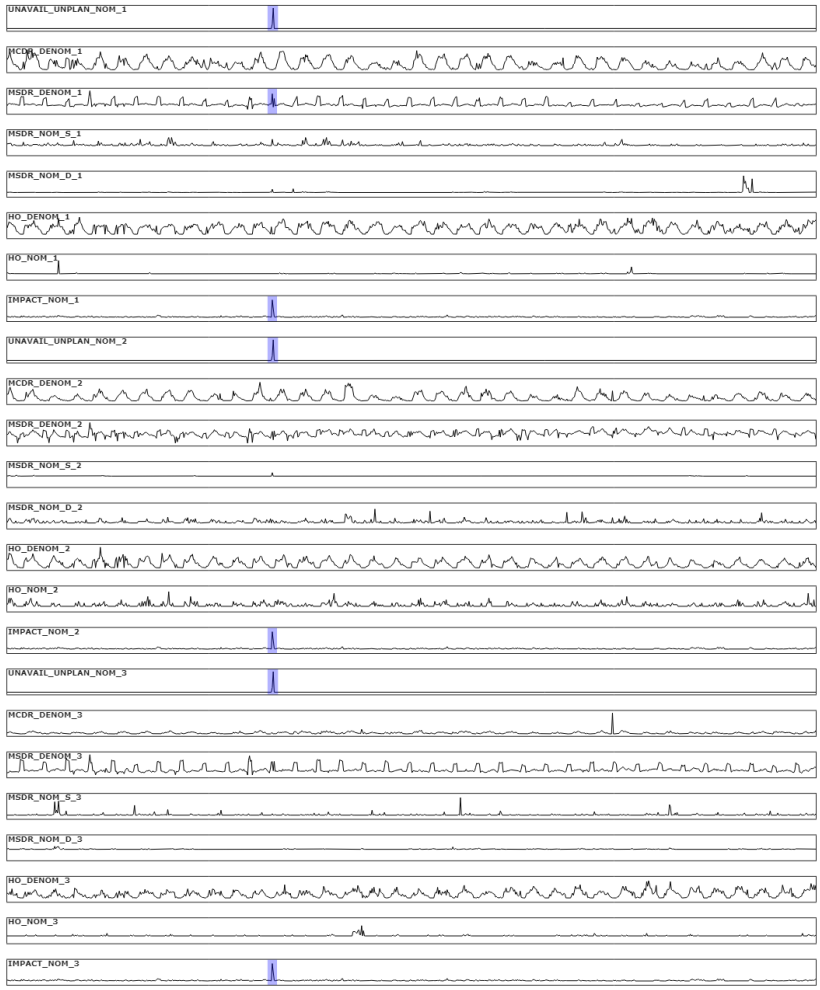


Figure 6.4: Example of a feature-wise contextual anomaly, where anomalies are aligned across multiple features. Individually they are not considered an anomaly, but together they represent anomalous behavior.

timestamps, while the leftmost correspond to the least recent. This timestamp correspond to a Thursday at noon. We note that the attention weights are large for the most recent timestamps, and then spikes in daily pattern, corresponding

to 24, 48, etc., hours before the current timestamp. Thus, it seems like the time-oriented GAT layer is capturing daily patterns and has learned to attend to timestamps that are similar in terms of the time of the day. Also, what is perhaps even more impressing, is that the attention weights are lower for the forth and fifth day prior to the current timestamp, which we know correspond to the weekend (Saturday and Sunday). The lower part of Figure 6.5 shows *mcdr_denom* for the same input window, illustrating that the daily patterns of the weekend deviates from that of the weekdays. This way, we see that the model has learned to *not* emphasize weekends when the current timestamp is at weekday. In summary, these observations leads us to believe that the time-oriented GAT layer can be highly useful for learning and exploiting temporal dependencies.

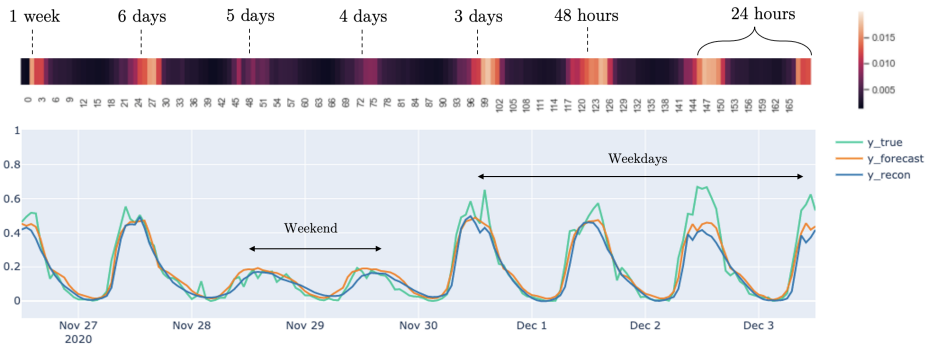


Figure 6.5: **Top:** Illustration of attention weights for the *last* timestamp of the attention matrix from the time-oriented GAT layer, when the model is fed with a 1-week input window (168 hours). Lighter color denotes higher attention weight. For each of the 168 input timestamps, there is a corresponding attention weight, which represent how much attendance should be given *by* the last timestamp *to* a particular other timestamp. The rightmost attention weights correspond to the most recent timestamps, while the leftmost correspond to the least recent. **Bottom:** The values of *mcdr_denom* in the same input window. The last timestamp of the input window corresponds to a Thursday at noon. We see that the time-oriented GAT layer attends the most recent timestamps, in addition to timestamps of previous weekdays at the same time of the day, but also that it has learned not to emphasize weekends, as weekends do not share the daily patterns of weekdays.

Feature-wise Attention

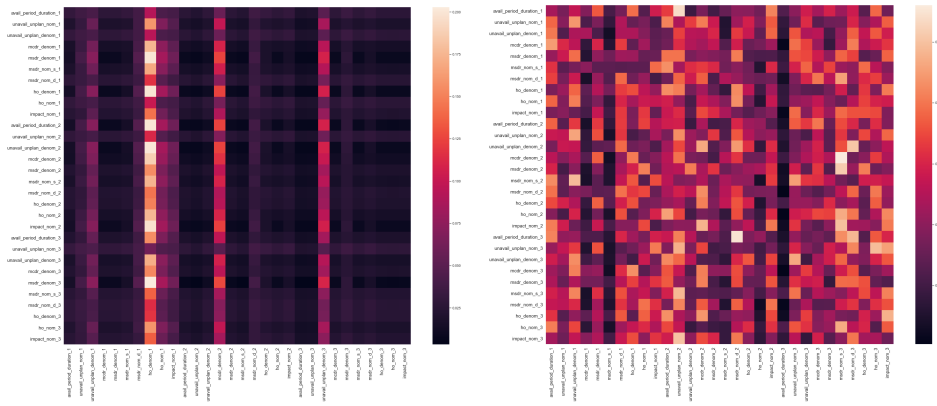
As described in subsection 5.2.2, we perform experiments on the TDBU dataset using both the original GAT attention scoring function (which we here denote

as *static*) and the GATv2 version proposed by [5] (denoted *dynamic*). In order to evaluate the two methods, we extract the complete attention matrix of the feature-oriented GAT layer for both methods, and compare them. This is illustrated in Figure 6.6, where Figure 6.6a and Figure 6.6b shows the attention matrix computed using GAT and GATv2, respectively. Recall that the attention matrix is a $k \times k$ matrix (k is the number of features), where row i represents how much feature i attends to each of the k features (including itself).

By inspecting the static attention matrix in Figure 6.6a, we observe what the attention weights have the same ordering in every row. For example, we see that every feature (row) attends the most to *ho_denom_1*, second most to *mcdr_denom_2*, and third most *unavail_unplan_denom_3*. Common to Brody et al. [2021], we consider this property of the original GAT to be undesirable, as every feature is prohibited against having an individual ordering of attention weights. Instead, the expressive power is restricted to the "sharpness" of the distribution of the produced attention, i.e. each feature can distribute the attention differently to different features, but must keep the same ordering. However, for the dynamic attention in Figure 6.6b, we see that this restriction is not present, yielding richer attention weights. Using the dynamic attention, each feature can attend as it like to any other feature, and need not to have the same ordering. By comparison of the two attention matrices, we conclude that GATv2 constitute a more powerful graph attention mechanism than the original GAT.

An interesting remark is that the diagonal of the attention matrix, which represent how much each feature attends to itself, does not tend to be higher than the rest. Intuitively, this seems unreasonable, as one would expect that a feature's own values should be important for forecasting and reconstructing that feature. However, recall from subsection 4.3.4, that in addition to the output from the two GAT layers, the GRU layer receives the features (after the 1-D convolution) directly. Thus, the GRU layer always receives the (almost) unprocessed features, a fact that might enable the feature-oriented GAT layer to focus on learning attention weights for which the "self-to-self"-attention weight does not need to be emphasized.

Figure 6.7 shows one row of the dynamic attention matrix, namely *unavail_unplan_nom_3*, representing the minutes of which sector three is down (unavailable) for a particular site. From the telco-perspective, *unavail_unplan_nom_3* will typically be correlated with different type of failures, which is reflected in high attention scores for *msdr_nom_d_3*, and *ho_nom_3*, representing the number of data drops and handover failures in sector three, respectively. Additionally, it is also shows high attendance to *impact_nom_3*, representing the number of users that receives error messages, and *unavail_unplan_nom_2*, which represents the same as *unavail_unplan_nom_3*, but for sector two. However, other attention



(a) Static Attention (Original GAT)

(b) Dynamic Attention (GATv2)

Figure 6.6: Comparison of the attention matrix from the feature-oriented GAT layer, between original GAT and GATv2. Each row represents how much attendance is given *by* the source-feature corresponding to the row, *to* every other feature (including itself). With original GAT, the ordering of the attention attention weights is global, unconditioned on the source-feature. GATv2 does not have this restriction, yielding more expressive attention weights.

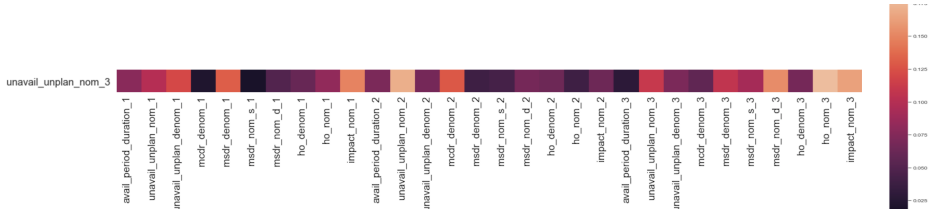


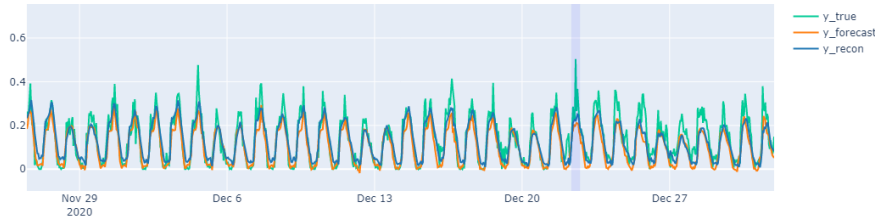
Figure 6.7: Illustration of attention weights for *unavail_unplan_nom_3* (representing the minutes of which sector three is down (unavailable) for a particular site), extracted from the feature-oriented GAT layer when it was fed with a particular input window. Lighter colors indicate higher attention weights.

weights are less intuitive, e.g. that there is relatively high attention to the general data traffic in sector one (*msdr_denom_1*), but not to the unavailability of sector one (*unavail_unplan_nom_1*). By inspecting other examples, we find that it will often be difficult to draw meaningful insight from certain attention weights, which reminds us that one should be careful when attributing human intuition to

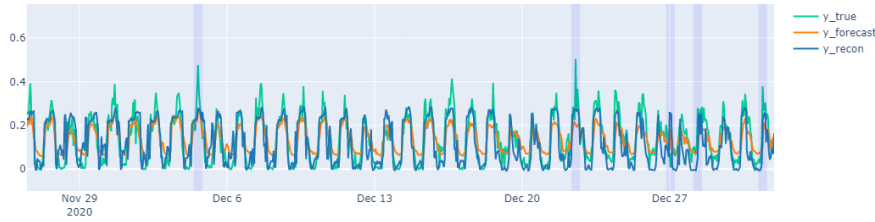
the mechanics of neural networks. Neural networks are extremely complex and there is a reason for why they are commonly referred to as "black boxes"; it is difficult to understand why they act the way they do.

6.2.3 Transfer Learning results

Recall from subsection 5.2.2 that the utilization of transfer learning on the TDBU data was motivated by several factors, namely 1) that combining knowledge from multiple sites could be beneficial for the model, 2) that it provides significantly more training data, and 3) that it removes the unpractical requirement of having to train a separate model for each site. We evaluate its effectiveness by visually comparing results of the transfer-learned model against the the normal site-based model. Despite some cases for which the transfer-learned model performed better, we found it to provide less accurate reconstruction and forecastings for the majority of the sites. Figure 6.8 reflects the typical difference in performance present for most sites. Through discussions with telco domain experts, we know that there can be large variation in traffic patterns between different sites. As the transfer-learning model tries to generalize across all sites, it is enforced to learn more universal dependencies that applies to all sites. Thus, if data from different sites vary too much, this constitutes a problem for the transfer-learning model, where it ends up not being able to accurately predict each individual one. A natural approach for overcoming this problem would be to cluster sites based on their similarity in traffic patterns, and then apply the transfer-learning model to sites within the same group. Specifically, this could be achieved by utilizing clustering algorithms or representational learning techniques [45].



(a) Non-transfer learning.

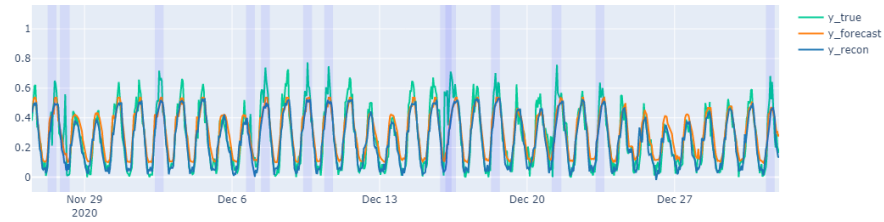


(b) Transfer learning.

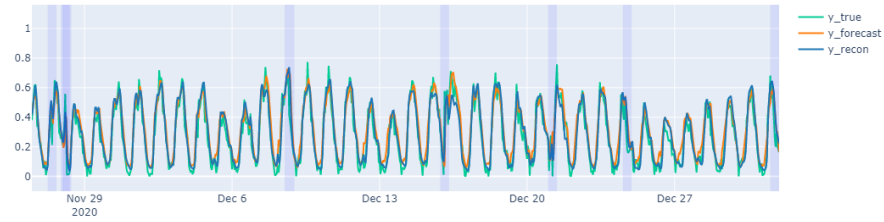
Figure 6.8: *ho_denom*. Comparison of transfer learning versus a conventional site-based trained model.

6.2.4 Site-based versus Sector-based results

Recall that the purpose of running sector-based experiments was to test our assumption that feeding the model with data from all three sectors within a site (in contrast to each sector separately) would provide it with richer information, enabling it to learn any correlations potentially present between sectors. As stated in section 5.2.2, we expected the site-based model to perform better for sites with sectors of similar traffic patterns, but that the sector-based model might be advantageous for sites where such similarities are missing. In general, we found the results to meet our expectations. Figure 6.9 illustrates one example in which the sector-based seems to be performing better. This corresponding site has two of three sectors pointing towards a populated part of the city, while the last is directed towards a road. By visual inspection, we observe that the site-based model has a worse fit, causing it to detect several false positives.



(a) Site-based.

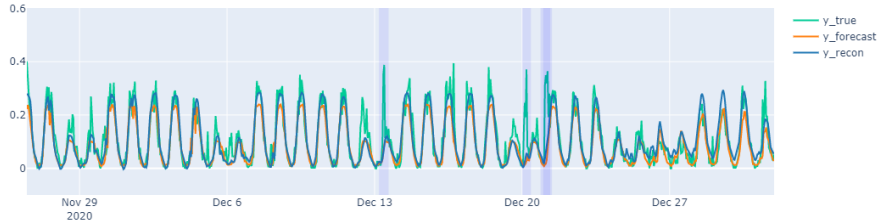


(b) Sector-based.

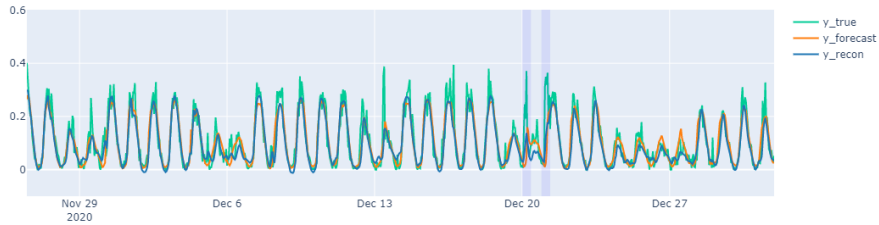
Figure 6.9: *mcdr_denom*. Comparison of a site-based model trained on all three sectors and a sector-based model.

In Figure 6.10, we can observe a site-based model which has benefited from learning from all sectors simultaneously. The site which it has trained on, has three sectors with highly comparable traffic patterns. The sector-based model has

only trained on one sector, and thus it has not benefited from learning the trend from its neighboring sectors. We can observe that it is attempting to reconstruct and forecast an anomalous segment, whilst the site-based has drawn advantages from learning the trend across sectors. This is better visualized in Figure A.1 where all three features, from each sector respectively, is compared in a site- versus sector-based fashion.



(a) Site-based.



(b) Sector-based.

Figure 6.10: *mcdr_denom*. Comparison of a site-based model trained on all three sectors and a sector-based model.

These results suggest that combining the sectors of the same site is not necessarily beneficial, especially if they are not expected to share similar traffic patterns. As with the transfer learning model, there is an argument to be made for clustering sectors based on similarity in traffic pattern, and combining sectors within each group, instead of naively combing those that belong to the same site.

Chapter 7

Conclusion

In this concluding chapter, we provide a short summary of the work that has been done in this master thesis, as well as giving a few suggestion for future work. The summary of our findings related to the research questions stated in section 1.2 are addressed in subsection 7.1.1 and subsection 7.1.2, respectively.

7.1 Summary

In this master's thesis, we have developed and assessed a Spatio-Temporal Graph Attention Network (STGAT) for unsupervised anomaly detection in the telco domain. We verified the effectiveness of our framework by evaluating it on three benchmark datasets, in which our framework achieved the best F_1 score on the MSL dataset, with a 4% improvement over the best state-of-the-art performance. By comparison with a brute-force threshold search, we saw that the Threshold Selection Model performed decent compared to the approximate upper-limit, but the difference were significant enough to make us realize that a proper threshold method is of upmost importance in any unsupervised anomaly detection system. Additionally, we performed an extensive ablation study with the purpose of demonstrating the impact of the core components of our model, i.e. the time-oriented GAT layer, the feature-oriented GAT layer, the Forecasting Model, and the Reconstruction Model. In general, for MSL and SMAP, each core component had little to no impact on the model's performance. However, for SMD, that consists of multiple continuous features (in contrast to MSL and SMAP), we saw a significant decrease in performance when any of the core components

were removed.

Further, we employed our framework on the TDBU dataset, a real-world industry dataset of Key Performance Indicators (KPIs). Our main experiments consisted of training and testing our framework separately on multiple sites, where data from each sector had been column-wise concatenated. Through visualizations and discussions with domain experts, we found our framework to be highly successful in detecting a wide variety of anomalies, from easily-detected sudden spikes to anomalies hidden in complex patterns. Also, as our framework detects anomalies at feature-level, it can output feature-wise contextual anomalies, where anomalies of multiple features are occurring at the same time. However, we also showed that, because of the way the Threshold Selection Model works, our model may struggle to detect time-wise contextual anomalies, i.e. anomalies that are considered anomalous only within a short window of time.

Besides, we investigated the use of transfer learning with the purpose of obtaining a generalized model, capable of working across multiple sites. The experiments showed that the transfer learning model struggled to forecast and reconstruct accurately, probably because of the fact that sites can have quite different traffic patterns, making it difficult to generalize well across sites. In addition, we tested our initial assumption that feeding the model with data from all three sectors within a site would yield better results than using each sector separately. The results confirmed our suspicion of that this assumption would only hold for sites where the sectors has similar traffic patterns.

The overall goal that we wanted our thesis to contribute towards was to develop a complete, accurate and robust framework for unsupervised anomaly detection in the telecommunications domain. The aforementioned work of this thesis showed that our proposed framework, for the most part, is 1) complete, because it is able to take raw data as input, and output detected anomalies for each input feature in real-time, 2) accurate, as it is able to accurately detect anomalies of multiple sorts, and 3) robust, as it performs well across multiple sites, in addition to being on par with SOTA methods on several benchmark datasets. However, we experienced that our framework is not flawless. For example, our framework struggled to detect time-wise contextual anomalies, and the Threshold Selection Model performed considerably worse than the approximate upper-limit performance. In our seek to achieve the overall goal, we aimed to study how Graph Attention Networks (GATs) could be used to model the complex, highly non-linear relationships expected to be present among the KPIs. We divided our aim of study into two research questions, which we will address next.

7.1.1 The first aim of the study

The first aim of the study (Research question 1) was to study how Graph Attention Networks (GATs) can be used to model inter-feature (spatial) relationships between the KPIs. In our model, the feature-oriented GAT layer is responsible for capturing the spatial dependencies. We evaluated the feature-oriented GAT in primarily two ways. Firstly, through our ablation study, we saw that our framework's performance on SMD significantly decreased when removing the feature-oriented GAT layer. Recall that we focus on SMD, rather than MSL and SMAP, because the TDBU dataset is fundamentally different from the latter ones. This result indicate that the feature-oriented GAT could be highly beneficial for data where inter-feature relationships exists. Secondly, we analyzed the learned attention matrix of the feature-oriented GAT layer, both using the original attention mechanism of GAT [43], and the newly proposed modifications of GATv2 [5]. We observed that the original GAT was only able to compute a restricted kind of attention where the ranking of attended nodes is global across all nodes. Using GATv2, however, this limitation is removed, making it strictly more expressive than the original GAT. By inspecting individual rows of the attention matrix, we observed that feature-oriented GAT layer had learned relationships that seem intuitive from a telco-domain perspective. However, we also found several examples that lacked such intuitive characteristics. In summary, we found the feature-oriented GAT layer to be beneficial for modeling dependencies between KPIs, but that it can be difficult to draw meaningful insights from the learned attention matrix.

7.1.2 The second aim of the study

The second aim of the study (Research question 2) was to study whether Graph Attention Networks (GATs) additionally could be utilized to capture and exploit temporal dependencies of the KPIs. In our model, the time-oriented GAT layer is responsible for capturing the temporal dependencies. We evaluated the time-oriented GAT layer in the same way as we evaluated the feature-oriented GAT layer. In our ablation study, we saw that the time-oriented GAT layer had a large positive impact on the performance of our framework on SMD. Additionally, when analyzing its learned attention matrix, we found that it had learned to capture complex temporal patterns, which also seemed to be intuitive from the telco-domain perspective. In summary, we found the time-oriented GAT layer to be successful in its task of capturing and exploiting temporal dependencies present in the KPIs. However, we would like to analyse its effect even more. As stated earlier, deep neural networks typically model temporal dependencies through a recurrent neural network, such as GRU or LSTM, and the role of the GRU layer in our model is in fact to capture long temporal patterns. Therefore, it would be

interesting to see how our model would perform if the GRU layer was removed completely, effectively leaving more of the temporal dependencies to be modeled by the time-oriented GAT layer.

7.2 Future Work

Anomaly detection remains an active field of research, and we expect to witness several breakthroughs in the time to come. We consider there to be several interesting approaches that could serve as a natural progression of this work. Firstly, the use of other thresholding methods could be explored. For example, *Peaks-Over-Threshold* (POT)[39] is a method originating from Extreme Value Theory (EVL) that has become popular in recent research [48] [15] [40]. Secondly, exploring other implementations of the Reconstruction Model could be considered. We implemented it as a single-layered GRU, but recent research has shown success using other architectures, e.g. Variatonal Auto-Encoders (VAEs) [48] and Generative Adversarial Networks (GANs) [23]. Thirdly, GATs constitute a relatively new research area, and is constantly evolving. For example, we recognized in [5] that the original GAT has limitations which restricts its expressive power. Thus, the mechanisms of graph attention could be further explored.

Alternatively, other deep-learning attention approaches could be investigated. In recent years, the Transformer [42] is an attention mechanism that has emerged as the preferable choice when faced with the challenge of sequence modeling and forecasting, due to its capability of capturing long-term contextual information as well as its parallel efficiency. Chen et al. [2021] proposed **GTA**; **G**raph Learning With **T**ransformer for **A**nomaly Detection, which automatically learns a graph structure followed by graph convolutions, and utilizes a Transformer-based architecture for modeling the temporal dependencies.

When it comes to the telco-domain, we believe one should explore other ways of making the model capturing spatial relationships between sectors and sites. So far, we have incorporated geographically spatial information by concatenating KPIs of sectors within the same site, and then making the model learn attention weights between all these KPIs. As suggested previously, instead of combining sectors of the same site, one could consider clustering sectors based on their similarity in traffic patterns, using clustering algorithms or representational learning techniques. This could also make transfer learning more effective, possibly improving the the performance of a generalized model that works across multiple sites. However, the model would still be restricted to only learning correlations between sectors in the same cluster as the model computes attention weights between every pair of KPIs, and the dimension of the attention mechanism grows by the number of KPIs for each sector included, this is a rather computationally

expensive approach. Additionally, if including numerous sectors (e.g. 5 or 10), it might become implausible to model the correlation between every pair of KPIs. A way to address this would be to model the graph at a higher level, similar to what was proposed by Cirstea et al. [2021]. In our case, this could be to e.g. model each sector as a node in the graph. Compared to the methodology of this thesis, each node would go from representing an individual KPI of a sector, to representing a complete sector with all its KPIs serving as the features of the node. In this way, the attention weights are computed between sectors, rather than between KPIs, which would enable the model itself to learn which sectors that are correlated.

Bibliography

[1] S. Ahmad, Alexander Lavin, S. Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.

[2] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. volume 2431, pages 15–26, 08 2002. ISBN 978-3-540-44037-6. doi: 10.1007/3-540-45681-3_2.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2015.

[4] Markus Breunig, Hans-Peter Kriegel, Raymond Ng, and Joerg Sander. Lof: Identifying density-based local outliers. volume 29, pages 93–104, 06 2000. doi: 10.1145/342009.335388.

[5] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks?, 2021.

[6] Robert B. Burns and Clifford B. Dobson. *Transfer of learning (training)*, pages 345–357. Springer Netherlands, Dordrecht, 1984. ISBN 978-94-011-6279-1. doi: 10.1007/978-94-011-6279-1_9. URL `https://doi.org/10.1007/978-94-011-6279-1_9`.

[7] Sucheta Chauhan and Lovekesh Vig. Anomaly detection in ecg time signals via deep long short-term memory networks. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–7, 2015. doi: 10.1109/DSAA.2015.7344872.

[8] Zekai Chen, Dingshuo Chen, Zixuan Yuan, Xiuzhen Cheng, and Xiao Zhang. Learning graph structures with transformer for multivariate time series anomaly detection in iot, 2021.

[9] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1053. URL https://www.aclweb.org/anthology/D16-1053.

[10] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL http://arxiv.org/abs/1406.1078.

[11] Razvan-Gabriel Cirstea, Chenjuan Guo, and Bin Yang. Graph attention recurrent neural networks for correlated time series forecasting – full version, 2021.

[12] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. Convolutional sequence to sequence learning. 05 2017.

[13] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[14] Michael A. Hayes and Miriam A.M. Capretz. Contextual anomaly detection in big sensor data. In *2014 IEEE International Congress on Big Data*, pages 64–71, 2014. doi: 10.1109/BigData.Congress.2014.19.

[15] Q. He, Y. Zheng, C.L. Zhang, and H. Wang. Mtad-tf: Multivariate time series anomaly detection using the combination of temporal pattern and feature pattern. *Complexity*, 2020:1–9, 10 2020. doi: 10.1155/2020/8846608.

[16] Sepp Hochreiter and JÃ¼rgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[17] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Söderström. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*. doi: 10.1145/3219819.3219845. URL http://dx.doi.org/10.1145/3219819.3219845.

[18] Dino Ienco and Roberto Interdonato. Deep multivariate time series embedding clustering via attentive-gated autoencoder. In Hady W. Lauw, Raymond Chi-Wing Wong, Alexandros Ntoulas, Ee-Peng Lim, See-Kiong Ng, and Sinno Jialin Pan, editors, *Advances in Knowledge Discovery and Data Mining*, pages 318–329, Cham, 2020. Springer International Publishing.

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

[20] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, 2017. URL `https://openreview.net/forum?id=SJU4ayYgl`.

[21] John Kolen and Stefan Kremer. *A Field Guide to Dynamical Recurrent Neural Networks*. 01 2001. ISBN 978-0-7803-5369-5.

[22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi: 10.1038/nature14539. URL `https://doi.org/10.1038/nature14539`.

[23] Dan Li, Dacheng Chen, Lei Shi, Baihong Jin, Jonathan Goh, and See-Kiong Ng. Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks, 2019.

[24] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[25] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. 04 2015.

[26] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection, 2016.

[27] Peggy O'Neill, Dara Entekhabi, Eni Njoku, and Kent Kellogg. The nasa soil moisture active passive (smap) mission: Overview. In *2010 IEEE International Geoscience and Remote Sensing Symposium*, pages 3236–3239, 2010. doi: 10.1109/IGARSS.2010.5652291.

[28] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton van den Hengel. Deep learning for anomaly detection: A review. *CoRR*, abs/2007.02500, 2020. URL `https://arxiv.org/abs/2007.02500`.

[29] Daehyung Park, Yuuna Hoshi, and Charles C. Kemp. A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder, 2017.

[30] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2013.

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[32] 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. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, Jul 2019. doi: 10.1145/3292500.3330680. URL `http://dx.doi.org/10.1145/3292500.3330680`.

[33] Md. Shiblee Sadik and Le Gruenwald. Research issues in outlier detection for data streams. *ACM SIGKDD Explorations Newsletter*, 15:33–40, 03 2014. doi: 10.1145/2594473.2594479.

[34] C. D. Santos and M. Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *COLING*, 2014.

[35] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

[36] Andrew W. Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander W. R. Nelson, Alex Bridgland, Hugo Penedones, Stig Petersen, Karen Simonyan, Steve Crossan, Pushmeet Kohli, David T. Jones, David Silver, Koray Kavukcuoglu, and Demis Hassabis. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, Jan 2020. ISSN 1476-4687. doi: 10.1038/s41586-019-1923-7. URL `https://doi.org/10.1038/s41586-019-1923-7`.

[37] Dominique T. Shipmon, Jason M. Gurevitch, Paolo M. Piselli, and Stephen T. Edwards. Time series anomaly detection; detection of anomalous drops with limited features and sparse examples in noisy highly periodic data, 2017.

[38] M. Shyu, S. Chen, Kanoksri Sarinnapakorn, and L. Chang. A novel anomaly detection scheme based on principal component classifier. 2003.

[39] 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, pages 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`.

[40] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '19, pages 2828–2837, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330672. URL `https://doi.org/10.1145/3292500.3330672`.

[41] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning, 2018.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[43] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[44] Paul Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78:1550 – 1560, 11 1990. doi: 10.1109/5.58337.

[45] Lirong Wu, Zicheng Liu, Zelin Zang, Jun Xia, Siyuan Li, and Stan. Z Li. Deep clustering and representation learning with geometric structure preservation, 2021.

[46] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021. doi: 10.1109/TNNLS.2020.2978386.

[47] Haowen Xu, Yang Feng, Jie Chen, Zhaogang Wang, Honglin Qiao, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, and et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW â18*, 2018. doi: 10.1145/3178876.3185996. URL `http://dx.doi.org/10.1145/3178876.3185996`.

[48] 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, 2020.

[49] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020.

[50] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BJJLHbb0-`.

Appendix A

Figures, Tables and Listings

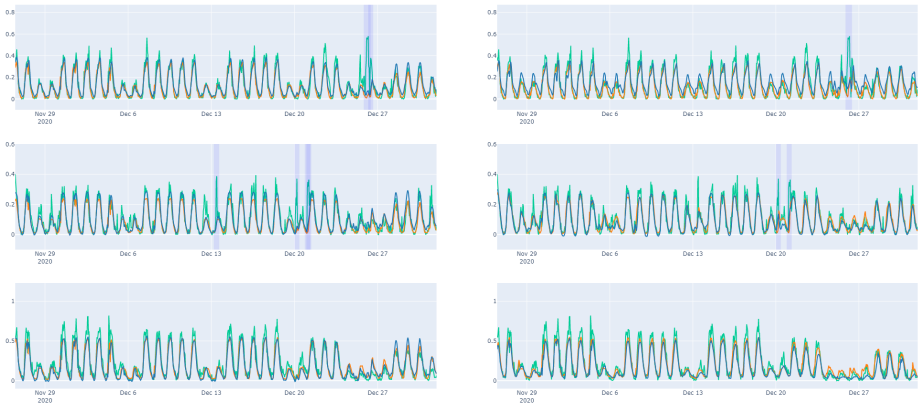
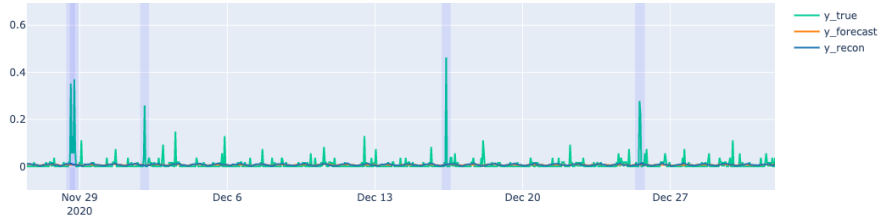
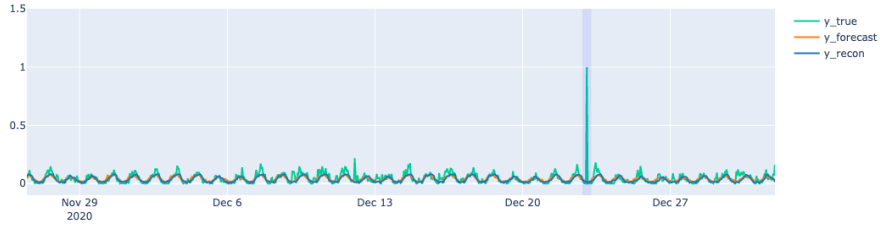


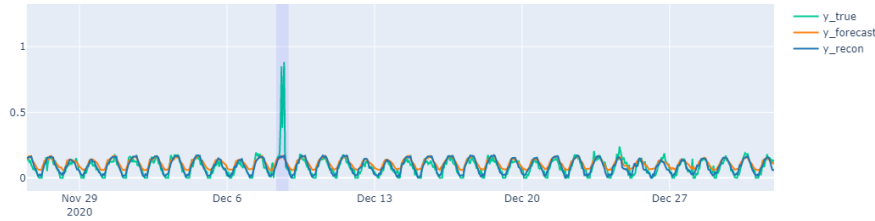
Figure A.1: Sector-wise comparison between a site-based and sector-based model. The row number indicates sector number. **Left:** Site-based model results. **Right:** Sector-based model results. *Green* is ground truth, *blue* is reconstruction and *orange* is forecasting.



(a) *msdr_nom_s*

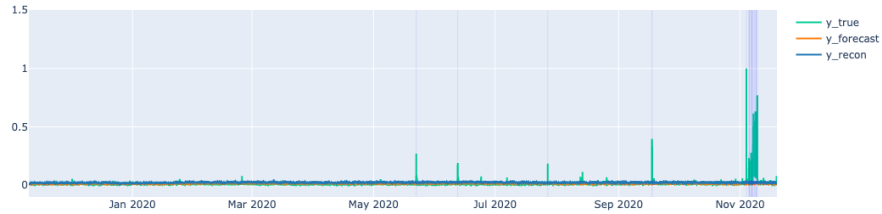


(b) *mcdr_denom*

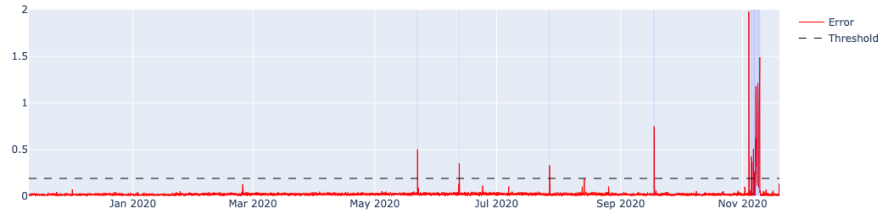


(c) *ho_denom*

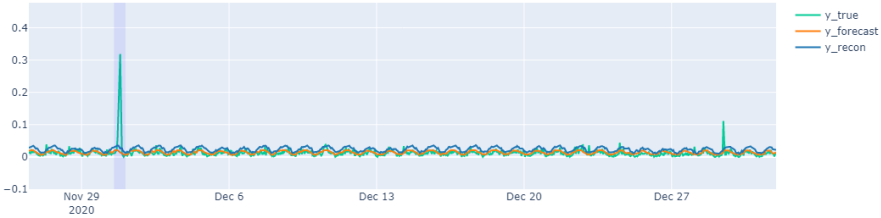
Figure A.2: Examples of sudden spikes that our framework detects as anomalies.



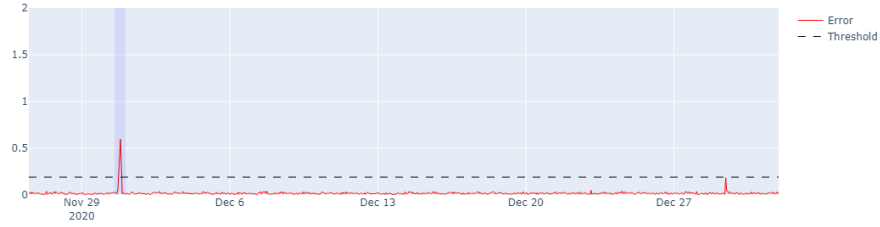
(a) Forecasts and reconstructions for the training set.



(b) Train anomaly scores. The large values of the rightmost anomaly scores causes the Threshold Selection Model to select a high threshold.

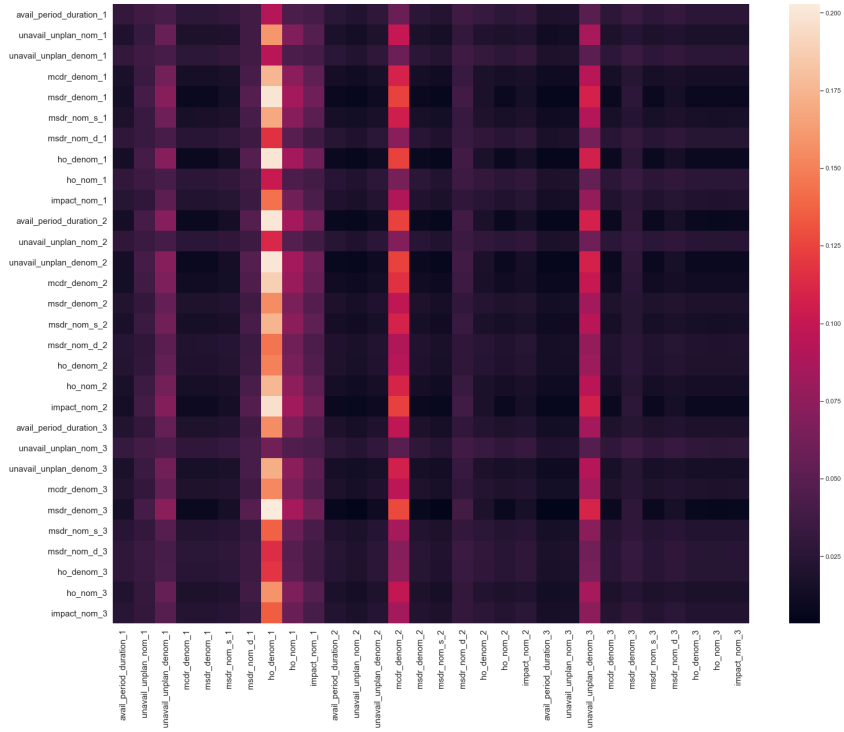


(c) Forecasts and reconstructions for the test set. The smaller second spike is not detected.

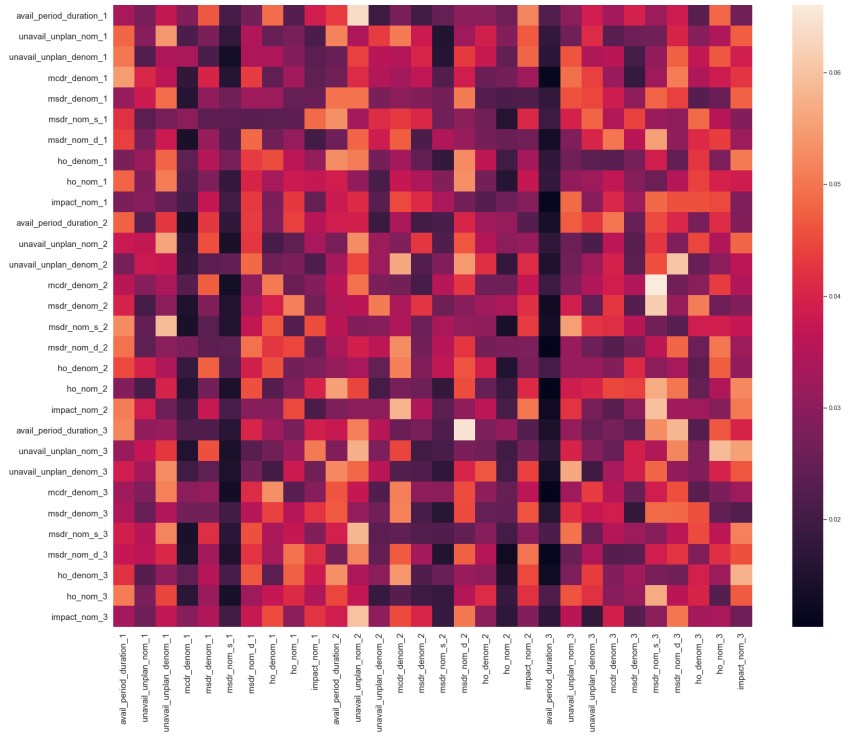


(d) Test anomaly scores. From the anomaly scores we see that the second spike is just below the threshold.

Figure A.3: A failure case for our framework where the second spike in the test set is not marked as an anomaly.



(a) Static Attention (Original GAT)



(b) Dynamic Attention (GATv2)

Figure A.4: Comparison of the attention matrix from the feature-oriented GAT layer, between original GAT and GATv2.

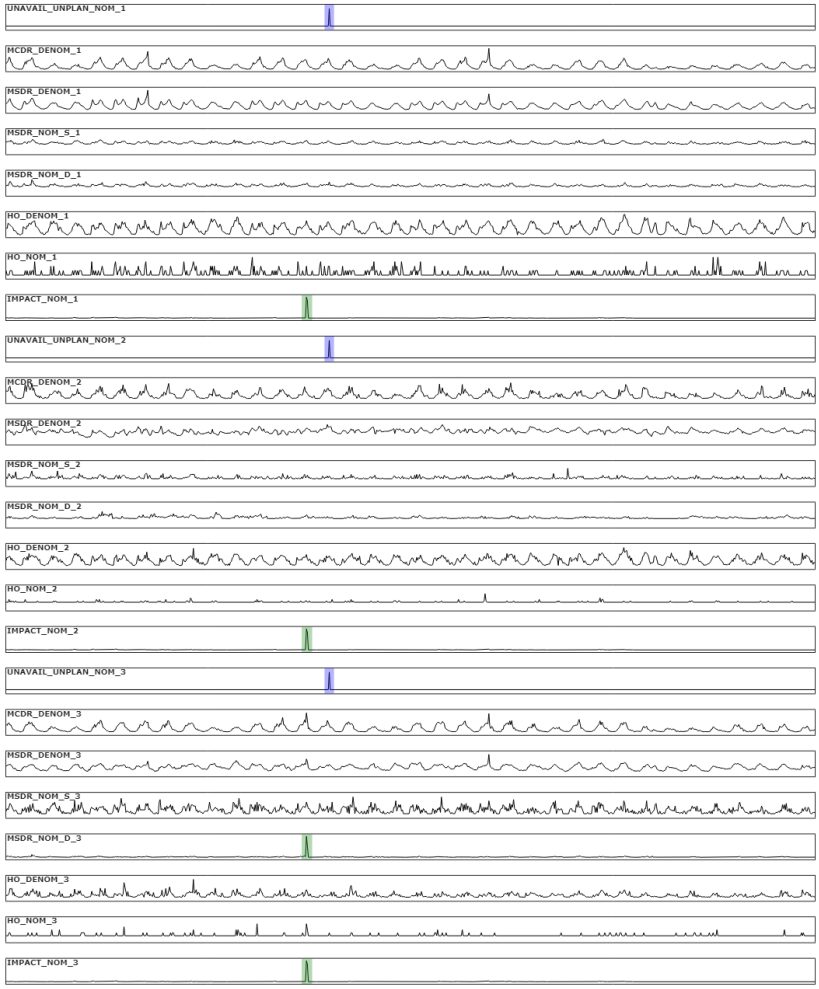


Figure A.5: Example of a feature-wise contextual anomaly, where anomalies are aligned across multiple features. Individually they are not considered an anomaly, but together they represent anomalous behavior.

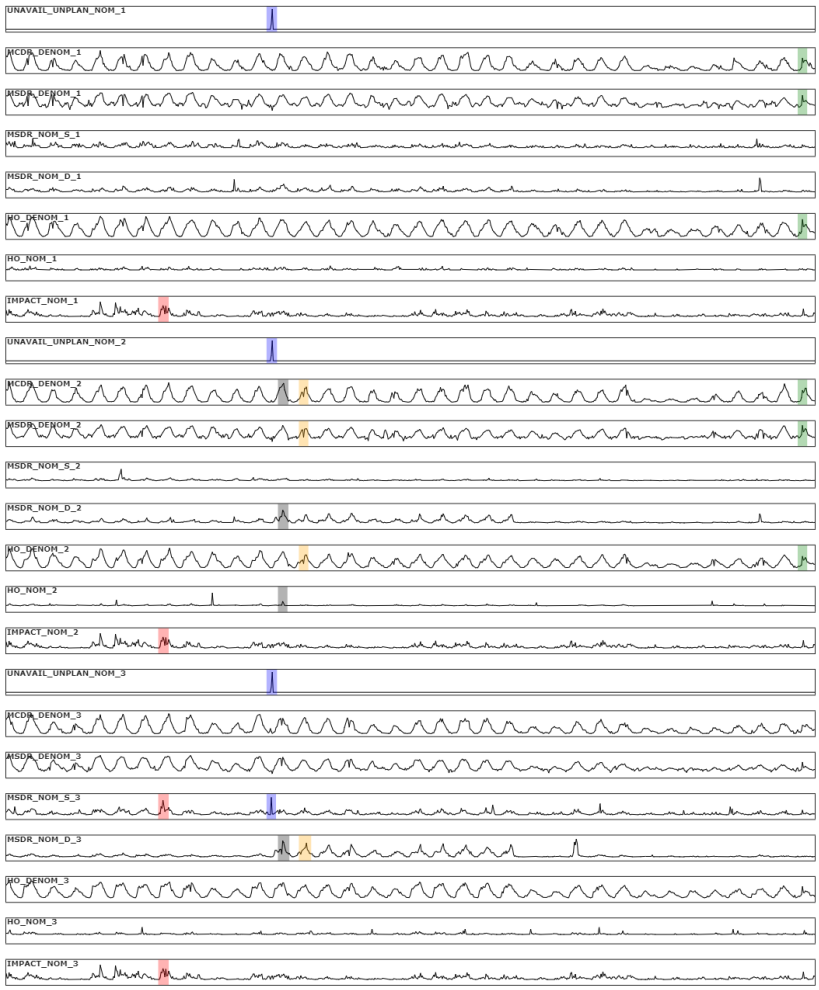
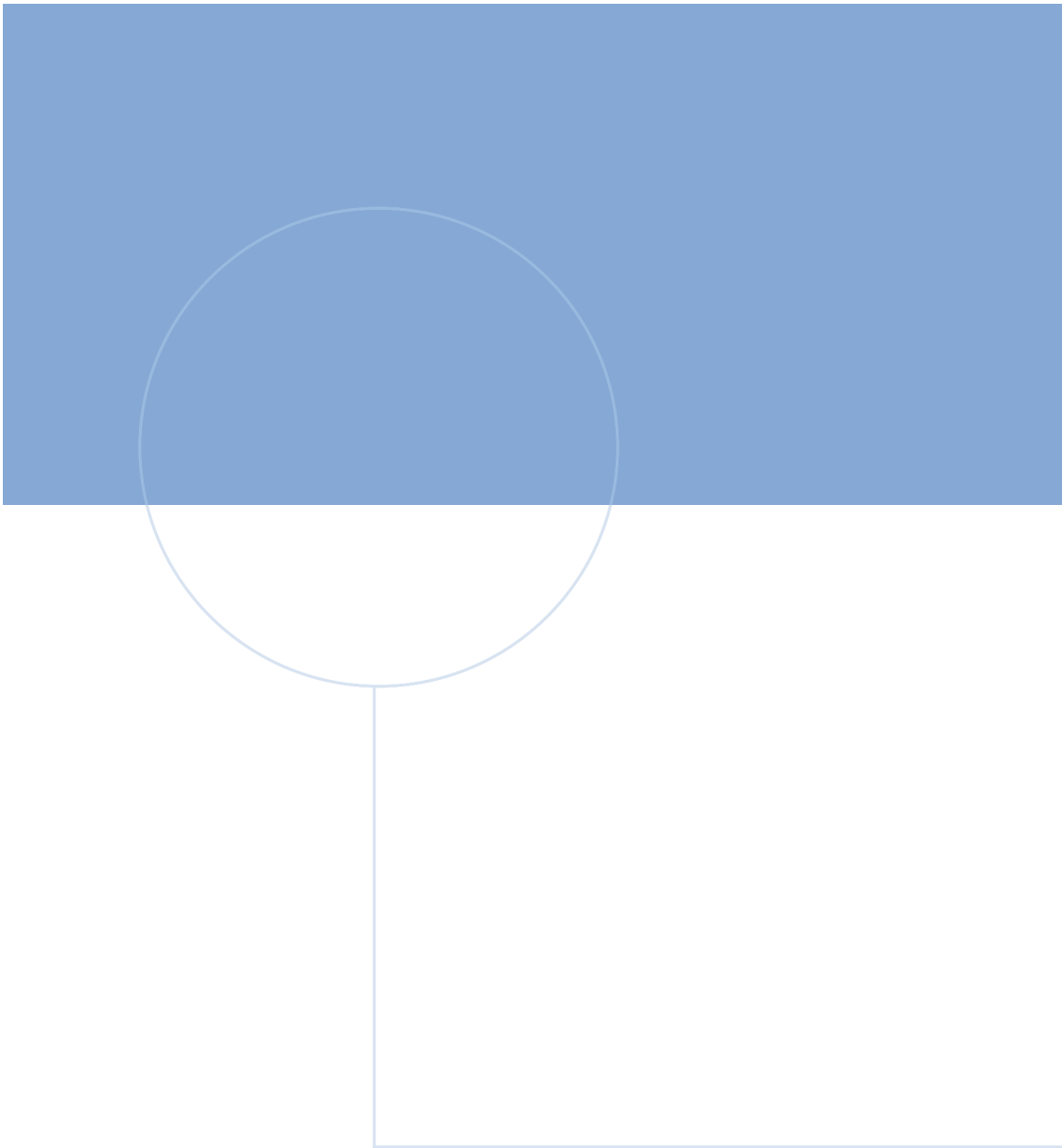


Figure A.6: Another example of feature-wise contextual anomalies detected by our framework.



NTNU

Norwegian University of
Science and Technology