Technische Universität Berlin
Electrical Engineering and Computer Science
Department of Telecommunication Systems

Autoencoder-Based Anomaly Detection for Combined Sewage Overflow Monitoring

Kyra Kerz
Matriculation number: 329289

Supervisor: Prof. Dr. habil. Odej Kao
Prof. Dr. rer. nat. Volker Markl
Advisor: M. Sc. Felix Lorenz

Berlin, August 16, 2023

Bachelor's thesis statement of originality

I hereby declare that the thesis

Autoencoder-Based Anomaly Detection for Combined Sewage Overflow Monitoring

submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.

The independent and unaided completion of the thesis is affirmed by affidavit:

Berlin, August 16, 2023

Name

Abstract

Combined Sewer Overflow (CSO) events are a major reason for water pollution in open water bodies. Each time the sewer system is loaded with more storm- and wastewater than the connected sewer treatment plant can handle, the overload of storm- and wastewater is pumped into adjacent water bodies [39]. Therefore it is necessary to detect possible incoming CSO events, in order to prevent an overload of the sewage system and be able to initiate counter measures before pumping additional wastewater into nature. To address this problem, I propose to leverage anomaly detection methods.
In this thesis a Long Short-Term Memory Autoencoder (LSTM-AE) for anomaly detection for time series, which is a combination of an Autoencoder with LSTM-Units, will be presented. Measurements of a set of sensors installed inside the sewer systems of Berlin were used to develop a forecasting model, that is able to classify the data into anomalous and non-anomalous events. The Autoencoder is used to handle the binary classification part of the task. It calculates a feature vector of the data set, which encapsulates the underlying structure of the data set. The LSTM Units are ensuring, that the temporal dependencies of the time series are captured in the feature vector. The feature vector will be used to produce a forecast, that is going to be compared to the ground truth in order to evaluate the LSTM-AE. It is shown, that the data set was not linear separable, therefore a capable ML method to address this like LSTM-AE for anomaly detection is needed. The area under the receiver operating characteristic curve (AUC) rates the proposed model with a score of 0.91 as a very effective classifier. For the classification a threshold needed to be chosen, based on the parameters of the confusion matrix, which lead to a F1-score of 0.754. Thus it is shown, that using a LSTM-AE for anomaly detection can successfully be leveraged for the task of detecting CSO events before they happen.

Zusammenfassung

Combined Sewer Overflow (CSO) Vorkommnisse sind ein Hauptgrund für die Wasserverschmutzung in offenen Gewässern. Jedes Mal, wenn das Kanalsystem mit mehr Sturm- und Abwasser belastet wird, als die angeschlossene Kläranlage bewältigen kann, wird die Überlastung von Sturm- und Abwasser in angrenzende Gewässer [39] gepumpt. Daher ist es notwendig, mögliche eintretende CSO-Ereignisse zu erkennen, um eine Überlastung des Kanalnetzes zu verhindern und Gegenmaßnahmen einleiten zu können, bevor zusätzliches Abwasser in die Natur gepumpt wird. Um dieses Problem anzugehen, schlage ich vor, Anomalie-Erkennungsmethoden einzusetzen.
In dieser Arbeit wird ein Long Short-Term Memory Autoencoder (LSTM-AE) zur Anomalie-Erkennung für Zeitreihen vorgestellt, der eine Kombination aus einem Autoencoder mit LSTM-Einheiten darstellt. Anhand von Messungen einer Reihe von Sensoren, die in den Berliner Kanalisationssystemen installiert sind, wurde ein Prognosemodell entwickelt, das in der Lage ist, die Daten in anomale und nicht-anomale Ereignisse zu klassifizieren. Der Autoencoder wird verwendet, um den binären Klassifikationsteil der Aufgabe zu bearbeiten. Er berechnet einen Merkmalsvektor des Datensatzes, der die zugrunde liegende Struktur des Datensatzes kapselt. Die LSTM-Einheiten sorgen dafür, dass die zeitlichen Abhängigkeiten der Zeitreihe im Merkmalsvektor erfasst werden. Der Merkmalsvektor wird verwendet, um eine Prognose erstellen, die mit der Grundwahrheit verglichen wird, um die LSTM-AE zu bewerten. Es wird gezeigt, dass der Datensatz nicht linear trennbar war, daher wird ein geeignetes ML-Verfahren wie das LSTM-AE zur Anomalieerkennung benötigt. Die Fläche unter der Receiver Operating Characteristic Curve (AUC) bewertet das vorgeschlagene Modell mit einem Score von 0,91 als einen sehr effektiven Klassifikator. Für die Klassifikation musste ein Schwellenwert gewählt werden, der auf den Parametern der Verwirrungsmatrix basiert, die zu einem F1-Score von 0,754 führen. Damit wird gezeigt, dass LSTM-AE zur Anomalieerkennung erfolgreich für die Aufgabe genutzt werden kann, CSO-Ereignisse zu erkennen, bevor sie eintreten.

Contents

List of Figures

List of Tables

1 Introduction

Water pollution is a huge problem in urban environments. Major contributors are waste water accumulations of cities, that are pumped into nearby water bodies. In order to prevent this, water network monitoring is needed. Waternetworks can span thousands of kilometers of pipelines beneath cities. This makes monitoring of those networks a difficult and complex task. The sewers are getting equipped with sensor networks in order to monitor the waterflow and waterlevels inside [39]. The collected data than can be used to gain knowledge of the systems behaviour and might even be useful to predict future states of the system. More specifically one application is the monitoring of combined sewage overflow. The purpose of combined sewer systems is collecting industrial wastewater, rainwater runoff and domestic sewage in a shared system. This wastewater is than transported to the sewage treatment plant, where it is treated and later on discharged into an adjacent water body. The combined sewage system is designed in a way, that it is able to handle the daily sewage from domestic, industrial and commercial sources. However in case of heavy rain fall, the capacities of the sewage treatment plants can exceed their limits and the wastewater of the combined sewers are released into adjacent rivers and lakes. Such events are an cause for water pollution problems [51]. Therefore it would be beneficial to tackle this problem with the help of the collected data of the sensor network. This is were anomaly detection comes into play. The sheer volume of data makes it impossible to tag anomalies manually, especially for multivariate data sets. For anomaly detection numerous machine learning algorithms have been proposed. Therefore it is important to emphasize the main criteria of the anomaly detection task for the given system. For this work the measured data of installed sensors inside the sewage system of Berlin is analyzed. The purpose of the monitoring of the water levels is to develop an early warning system to predict CSO events and based on that to initiate counter measures in order to prevent pumping of waste water into nature. The focus of this thesis is developing a forecast model for anomalous events, in this case CSO events. Anomaly detection is a broad field in machine learning (ML). There are different approaches in order to do anomaly detection e.g. support vector machines (SVM), hidden Markov models (HMM) or clustering methods. Developing such a model is heavily reliant on the structure of the given data. Thus the main question is, if it is possible to find a suitable architecture, which is able to extract the important features of the data in order to detect anomalies. In this case the data has several important attributes to consider. The model must be able to handle temporal dependencies, multivariate time series and non linear separability of the data. For this purpose the long short-term memory autoencoder (LSTM-AE) is proposed in order to tackle the task of anomaly detection for CSO events, which is able to handle the attributes and underlying structure of the collected data and is able to produce a forecast.

1.1 Contributions and structure of this work

This thesis is structured as follows. At first the related work section is covered in 1.2, where the state of the art methods for anomaly detection, time series analysis and anomaly detection for time series analysis are presented. Afterwards the theoretical background of this thesis is discussed in section 2. This includes an overview about anomaly detection and time series analysis, as well as types of neural networks that are relevant for the proposed architecture of this thesis like autoencoder (AE), recurrent neural network (RNN) and long short-term memory (LSTM). Following the actual methodology and architecture will be explained in section 3. In this part the main contributions are explained, which are depicted in figure 1.1.

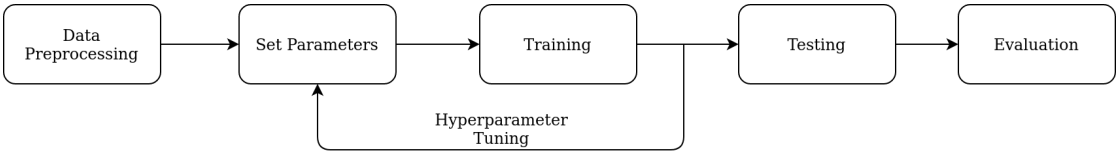

Figure 1.1: Process chain

At first an overview of the used data set and its preprocessing methods, that are required in order to match demanded input shape of the LSTM-AE, are be given. The next step is the labelling and its used metrics. This is required as the LSTM-AE is trained on non anomalous data. Then the data is the split into training, validation and holdout data set. After that the actual model architecture of the LSTM-AE and its implementation details are described as well as the initial setup and parameters for the training process. Furthermore the used methods and tools for hyperparameter tuning are presented. Another topic of this chapter is the description of the metrics of the confusion matrix, that are used for the evaluation of the results. Finally the training schedule is discussed and the trained model is applied on the test set in order to produce a forecast. In section 4 the results of the chosen model for the validation set and the unseen holdout data set are presented. The forecast it compared to the ground truth data set and an error density function is calculated. The values of the error density function are plugged in the evaluation methods, that are explained in section 3. With the parameters of the confusion matrix a threshold is chosen in order to detect anomalous water flow and therefore CSO events. Another purpose of the confusion matrix is the construction of the ROC-curve and calculation of the AUC-value in order to evaluate the performance and separability capability of the model. Finally in section 5 a conclusion about this approach in regards to anomaly detection for CSO events is critically assessed and improvements open for future work are stated.

1.2 Related Work

Anomaly detection in time series is a broad field, with many different approaches. Common used algorithms can be subdivided in the following categories: Statistical methods,

probabilistic methods, proximity based methods, clustering based methods and prediction based methods [12].
Statistical Models are mathematical models, that are using past measurements in order to approximate the behaviour of a monitored component. Whenever a new data point is measured, it will be compared to the statistical model. If the result doesn't match with the model, it is considered as anomaly. In [54] the authors put forward a traffic anomaly detection algorithm for wireless sensor networks (WSN), which uses an autoregressive integrated moving average (ARIMA) model. ARIMA makes traffic prediction in order to find anomalies in a WSN, which is used because of the non-stationary properties of the traffic and limited computing and energy capacity of the sensors. They systematically analyze the characteristics of WSN traffic and the causes of WSN abnormal traffic. Another ARIMA based model is presented by Ayesha et al. [47]. They are using ARIMA for forecasting wastewater inflow to sewage treatment plants and detecting anomalous water levels based on R-square value and normalized Bayesian information criterion. A downside of ARIMA is that you have to tune your model parameters each time you want to predict a new time series [43].
Probabilistic methods can be parametric or non-parametric. It's parametric, if the measured data points are following a well known distribution. In order to detect an anomaly, the data points will be classified with respect to the probabilistic model while determine the probabilities of the measurements. In [33] the authors make use of the Hidden Markov Model (HMM). They use it to characterize and detect changes in the probabilistic pattern sequence of data coming from the network. Relationships between data streams are modelled through sequences of linear dynamic time-invariant models whose trained coefficients are used to feed a HMM. When the pattern structure of incoming data cannot be explained by the trained HMM, a change is detected. But this actually assumes we know the number of states beforehand and HMMs struggle with underlying non linearity inside the data [28].
Proximity-based models are calculating the distances between measurements. Anomalous are those data points, which have a higher distance to correct measurements. A subset of those algorithms are clustering models, where the measurements are used to create clusters. New data points are labelled as anomalous, if they are far from their clusters centroid or are matched to small and isolated clusters. This means data doesn't have to be prelabelled. In [3] the authors propose a density-based spatial clustering of applications with noise (DBSCAN) as model in order to detect outliers in temperature data. Another clustering based model is proposed by Li et al. [23]. They are comparing the unsupervised methods of K-means clustering and spectral based clustering for the prediction of water levels inside sewer systems and detecting anomalies in case of stormwater flooding. However clustering approaches struggle with high dimensional data and with varying cluster sizes, making it unreliable to find outliers that are sorted into a cluster.
Prediction-based models use data points to train a model. This model is then used to predict the value of the next incoming measurements. In this case a measurement is labelled as anomaly, if the actual measurement is too different from the predicted data

point. [53] proposed a one class support vector machine (SVM) as model. The SVM is used to learn the feature vector of network traffic. This is based on entropy theory and is able to detect anomalous traffic behavior by treating it like a classification problem. Unfortunately SVM tend to work not very well for large data sets [40].
Other models of this category are neural networks. Zhang et al. are comparing several kinds of neural networks, that are able to handle temporal dependencies and multivariate time series like SVM, long short-term memory (LSTM), gated recurrent units (GRU) and recurrent neural networks (RNN), for forecasting CSO events [55]. The results are showing, tht LSTM and GRU are especially suitable to capture those attributes and are outperforming SVM. A further variation of the LSTM network is proposed by [26]. They are using a combination of an autoencoder and LSTM cells in order to forecast production of steel casting processes. This is done in an unsupervised manner to obtain the features and therefore domain knowledge isn't necessary to produce the forecast. The forecast is then used in order to detect anomalous behaviour with the help of the extracted features for classification, while also taking time dependencies into account.

2 Background

In this chapter the theoretical background of this thesis is explained. At first anomalies and anomaly detection are defined in section 2.1. After that the definition and structure of time series analysis are discussed in section 2.2, as well as how time series are playing a role in anomaly detection. In section 2.3 autoencoders are explained, which are the foundation for the proposed machine learning architecture that is used in this work. Temporal dependencies are a huge factor, that normal neural networks can't handle very well. This is where recurrent neural networks (RNN) are coming into play, which are explained in section 2.4. Those are able to handle short term dependencies, but not long term dependencies. Therefore an enhanced variant, the long short-term memory cell (LSTM), is presented in section 2.5 as core element of the model architecture. Finally everything comes together and the proposed model architecture, the LSTM-AE, is introduced. This model is able to handle temporal dependencies, as well as being able to be used as semi-supervised approach for anomaly detection.

2.1 Anomaly detection

Anomalies or outliers are definied by [14] as followed:

> "*An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism.*"

Therefore anomaly detection is the process of finding patterns in data that do not conform to expected behavior [4]. This means most of the time anomalies only occur very rarely in the data. Their features differ from the normal instances significantly. There are three kinds of anomalies explained in [14]:

- Point Anomalies: A data point which as single instance is observed against other data points as anomaly is called a point anomaly.
- Collective Anomalies: A set of data points is called collective anomaly, if the set is anomalous against other sets of data points. On the other hand each data point for itself is not anomalous.
- Contextual Anomalies: A single or several data points which are anomalous in some defined context, but not otherwise, are called contextual anomaly. The concept of context is imbued in the structure of the data set. This means it has to be considered as part for the problem formulation.

The task of anomaly detection can be quite challenging. It's not always clear which regions are to be defined as normal. There are several problems while identifying anomalies. First the boundaries between normal data and anomalies is not always unambiguous, especially if noise is prevalent. Furthermore getting enough data to train on normal data and validate on both normal and anomalous data can be difficult. Other obstacles can be changing environments, additive outliers and time dependencies like temporal changes, level shifts or seasonal level shifts. Adversaries are trying to disguise their frauds and are trying to make them seen as normal as possible.
As seen in the related work section 1.2 there is no best model to detect anomalies. A model that is strong in detecting punctual anomalies, wont have the same success detecting contextual or collective anomalies and vice versa. As mentioned in section 1 this thesis will focus on ML techniques for anomaly detection. Those are highly dependent on the data and its structure. In general methods that are used for anomaly detection are divided in following categories:

- Supervised Anomaly Detection: This is the simplest case, but also the rarest and most unrealistic. In this scenario, there is a lot of data for each individual anomaly and there is also a large amount of data without anomaly. This case is simple because it is a classification problem with two or more classes, depending on the number of anomalies considered. Proven methods can be applied to this classification problem. However, obtaining so much high quality data is rare. But it is difficult to collect data for all conceivable anomalies. For this reason, this scenario is rarely used.

- Semi-supervised Anomaly Detection:

  In this case, the data is only available of the normal state. There is therefore no data on the anomalies. The difficulty here is to create a complete model that represents all normal data. It is important to ensure that this model does not become too general and that anomalies are not considered normal variables.

- Unsupervised Anomaly Detection:

  The last scenario does not require marked data. So there is an huge amount of data without the information whether the data is anomalous or if the data is non anomalous. Therefore this is the most common scenario. To identify the anomalies, the algorithm basically assumes that there are far more non anomalous values than anomalies.

In order to determine the quality of the anomaly detection method, Lavin and Ahmand proposed the Numenta Anomaly Benchmark (NAB) as benchmark for time series anomaly detection [21]. Basically, an anomaly detection algorithm should either label each time point with anomaly or not anomaly, or forecast a signal for some point and test if this point value varies from the forecasted enough to deem it as an anomaly. This is a problem of using forecasting techniques for anomaly detection. It should be aimed to

capture trends/seasonality in data with avoiding overfitting. If only the previous values of the time series are used to predict its future values, it is called univariate time series forecasting. If the predictors of several time series are used to forecast, it is called multi variate time series forecasting [17].

2.2 Time series

Time series are important because there are so many prediction problems that involve a time component. This makes the search for anomalies even more difficult to handle. A time series is a set of data with mostly equally spaced points with a timestamp ordered in time. This means the data is time discrete [25]. In time series data, an anomaly is any unexpected change in a pattern in one or more of the data points explained by timely properties [48]. There are two major use cases of time series - time series analysis and time series forecasting. Time series analysis is used to develop models that best capture the underlying structure of a time series. This is a form of descriptive modelling [42]. Time series forecasting on the other hand is studying time series in order to predict the future behaviour of the variables based on past values of the time series. Because the order of the data is important, each model will have time as an independent variable. This is a form of predictive modelling [1]. Again this can be done while considering univariate or multivariate time series. Analog to subsection 2.1, univariate time series are depending on one variable while multivariate time series are depending on more than one variable [49]. A time series can be dissected in four different component trend, seasonality, cyclicity and variation [36]:

- Trend : The long-term general change in the level of the data with a duration of longer than a year.
- Seasonality: Regular wavelike fluctuations of constant length, repeating themselves within a period of no longer than a year. Reasons can have its origin in cultural behaviour like vacation times or natural conditions like weather fluctuations that are representative of the season.
- Cyclicity: Wavelike movements or regular fluctuations around the long-term trend, lasting longer than a year.
- Variation: The random variations of the data comprise the deviations of the observed time series from the underlying pattern for instance noise.

These components are used to decompose the time series signal:

$$\text{Additve: } Y = S + T + C + V$$

$$\text{Multiplicative: } Y = S * T * C * V$$

In some time series, the amplitude of both the seasonal and irregular variations do not change as the level of the trend rises or falls. In such cases, an additive model is appropriate. In many time series, the amplitude of both the seasonal and irregular variations

increase as the level of the trend rises. In this situation, a multiplicative model is usually appropriate. Stationarity is an important characteristic of time series. A time series is said to be stationary if its statistical properties do not change over time. In other words, it has constant mean and variance, and covariance is independent of time.

In order to develop a sufficient time series forecasting model, some issues need to be considered. Time series analysis is the preparatory step before developing a forecast of the series. The more information about the data is known, the better the forecast. Furthermore underlying assumptions about the data like distributions or external factors that may influence the trend have to be detected . Especially while applying ML methods, there are some problems to keep in mind. Hyperparameter tuning of the model isn't easily applied. For instance when using cross-validation, the data set is randomly divided in training and validation set. However, the data set can't be randomly divided, because it has a temporal dependencies. Sometimes for time series modeling, earlier data is used as the training set, while newer data is used as the validation set [1]. In addition, when forecasting new values, the model itself is based on the entire data set, not just the training partition. This is because the data most relevant to a forecast are the observations that happened most recently.
Other problems are periods of missing observations or fluctuations of the data. This can be solved with methods like backward fill, interpolation, mean of nearest neighbors or mean of seasonal counterparts [9].
Finally, many predictive modeling problems involving sequences require a prediction that itself is also a sequence. These are called sequence-to-sequence, or seq2seq, prediction problems. This means that the chosen model needs to be adjusted in order to be able to output a sequence and not just a probability or classification value. Another challenge with sequence data is that the temporal ordering of the observations can make it challenging to extract features suitable for use as input for supervised learning models, often requiring deep expertise in the domain or in the field of signal processing [10]. And finally and above else the model must be able to capture the time dependencies of the data set.

2.3 Autoencoders

An autoencoder is feed forward neural networks. It is a an unsupervised learning algorithm, which means it is using unlabeled data. One of its main use cases is data compression, denoising of data and especially dimensionality reduction.
There are already prominent tools for dimensionality reduction like principal component analysis (PCA). In contrast to PCA autoencoders are not just able to detect linear dependencies, but are also able to formulate non linear functions. In fact an autoencoder without non linear activation functions works exactly like PCA. Another field of application for autoencoders is anomaly detection, because of its property to extract features of the given input data in an unsupervised manner. To be more precise they are self-supervised because they generate their own labels from the training data. They are most

of the time used for one versus many classification. Therefore autoencoders are able to detect anomalies due to training and comparing input to output. Is the reconstruction error greater than a certain threshold it will be considered an anomaly.
Usually autoencoders are restricted in ways that allow them to copy only approximately. Because the model is forced to prioritize which aspects of the input should be copied, it often learns useful properties of the data. The objective of the autoencoder is to minimize reconstruction error between the input and output. This helps autoencoders to learn important features present in the data.
In order to learn the most salient features of the training data, they learn the hidden representation h_t. Figure 2.3 shows the architecture of a basic autoencoder. It consists of three main parts: The encoder, decoder and the bottleneck as fully connected layers.

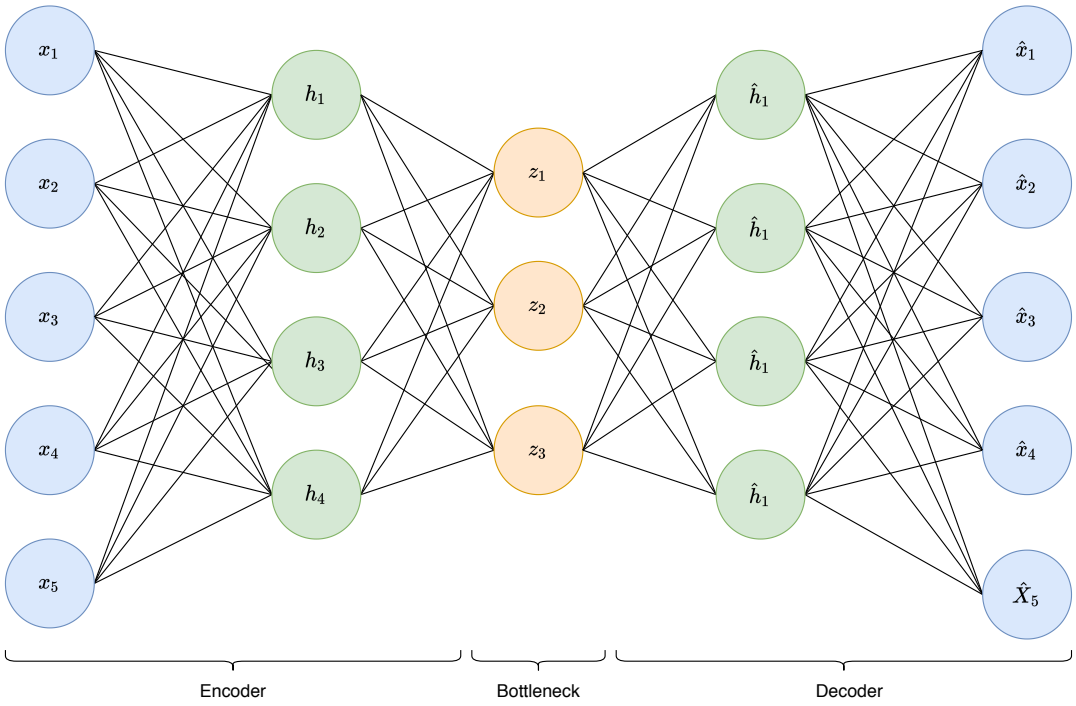

Figure 2.1: Autoencoder architecture

The layers of the encoder and the layers of the decoder have the same dimension with a smaller bottleneck layer in between. The encoder encodes the input to a feature vector. The decoder decodes this feature vector in order to get a reconstruction of the original input as output. Because of the shallow network universal approximation theorem, a neural network needs just one layer to be able to construct every continuous function [24]. However it is not easy to determine how many nodes are needed and more layers lead to faster training times. Furthermore being able to construct a good representation of a data set, doesn't mean it has a good ability to learn. In fact deeper networks tend ti have better generalization performances than shallow networks, because they are able to

learn more of the intermediate features [50]. Furthermore as more layers lead to smaller number of nodes in the middle layer, it will result in more compression [46].
The encoder function learns the a hidden representation h_t, which is the output of the encoder part. The function has the form $h = f(x) = s(Wx + b)$. Here s stands for the activation function, which is non linear like a tanh or sigmoid. W is the weight matrix with the dimension $h \cdot x$ and b is the bias [38].
In order to prevent h_t to just learn the identity function, the hidden layer has to have less neurons than the input and output layer. This way the autoencoder is able to detect the salient features of the input data. Because of this this kind of autoencoder is called undercomplete. The decoder function g maps the hidden representation back to a reconstruction $y = g(x) = s(W'h + b_y)$, which is the ouput.
Autoencoder are trained by finding the parameters W, W', b_h, b_y, which are minimizing the reconstruction error of the training set. In case of linear functions a typical loss function is the mean squared error. The cross entropy function is used, if the activation function is the sigmoid and if the input is binary [18]. In both cases the gradient is just x. That means we can use regular backpropagation.
In order to increase the effectiveness of the encoding function to represent the important features, there are different kinds of autoencoders. Regularized/sparse autoencoders have a constraint in their loss function. This means there is a reguralization term added in the loss function. This will make the autoencoder learn a sparse representation of data. There are other variations like denoising autoencoders, variational autoencoders, contracting autoencoders, which have the goal to learn a better latent space representation [8]. Unfortunately autoencoders have some downsides when it comes to feature extraction. For instance there can be gaps in the latent space. This is equivalent to having a lack of data in a supervised learning problem. The next one is the problem of separability in the latent space. And the last one is the separability of the spaces, which means classification of features is not always unique [46]. Furthermore since it can learn features specific for the given training data, it can not be used for other types of inputs. Finally is to mention that they are lossy. The output of the autoencoder will not be exactly the same as the input, it will be a close but degraded representation. On top of that vanilla autoencoder are not able to handle temporal dependencies [38].
An autoencoder has 4 hyperparameters that need to be considered for training. As mentioned before the number of layers is variable. The autoencoder can be as deep as we like. In the figure 2.3 we have 2 layers in both the encoder and decoder. The number of nodes in the hidden layers which are decreasing with each subsequent layer of the encoder and increases back in the decoder. The number of nodes in the bottleneck is the third parameter, whereas a smaller number leads to more compression, but a to small number might cut out important features. And finally the loss function, which is used to determine how well the output of the autoencoder is compared to the ground truth.

2.4 Recurrent Neural Networks

In order to display temporal dependencies in sequential data, ordinary feed forward networks (FFN) aren't sufficient. This is the task of recurrent neural networks (RNN), which have an internal state to memorise temporal behaviour [41]. This means they remember information of previous inputs. Time series data based tasks like speech or handwriting recognition is made applicable by RNN. This is possible, because in comparison to FFN, RNN possesses loops. Additionally to forward connections, the neurons are possessing feedback connections to the previous layer or to themselves. There are different kinds of feedback connections, which are partitioned as follows. The first is the direct feedback. In this case the output of a neuron is used as another input and connection to itself. The second is the indirect feedback where the neuron is connect to the neurons of the previous layer. The third is the lateral feedback where each neuron is connected to all other neurons of the same layer. The fourth is the full feedback where each neuron is connected to every other neuron. Figure 2.2 shows the architecture of a RNN. The variable x describes the input neuron and o the output neuron. h is the hidden layer that acts as the memory of the network. It is calculated on the current input and the previous time step of the hidden state. The output layer of the RNN is described by o. The connections between input layer and hidden layer, hidden layer to itself and hidden layer to output layer are respectively parameterized by the weight matrices U, V and W. This means the hidden state and the output are calculated as follows [27]:

$$h_t = f_h(Ux_t + Wh_{t-1} + b_h)$$
$$o_t = f_o(Vh_t + b_o)$$

In this case b_s is the bias and f_s a non-linear transformation function like tanh or softmax with $s \in \{h, o\}$.
The right side shows an unfolded version of a RNN into a full network which takes all the time steps from x_{t-1} to x_{t+1} of the sequence into account.

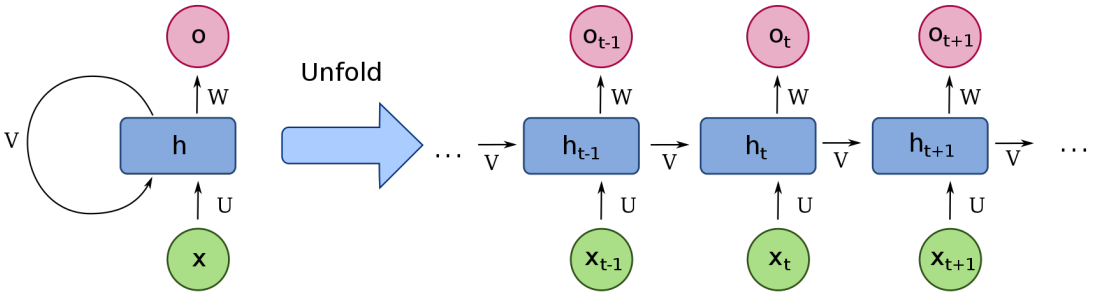

Figure 2.2: Unfolded RNN [1]

Similar to normal neural networks the parameters are trained with the help of back propagation through time (BPTT). As a result the loss will be minimized by the gradient descent method. Thereby the derivative of each activation function will be calculated iteratively. Because of that in each step the loss will be multiplied with a scaling factor for each layer in the network. If the value of this factor is between zero and one, the loss will become smaller and smaller and vanish eventually. Therefore this will lead to ineffective calculations of the weights. On the other hand if the values of the factor are bigger than one, gradient would eventually explode [37].
This means that the recalculation of the parameters of the hidden layers near the input layer have less impact on the output. Furthermore information that was passed early on in the sequence will be forgotten faster than information that will be passed at the end of the sequence, because of the back propagation. Therefore an improved version of the RNN is needed to prevent the problem of the vanishing/exploding gradient [15].

2.5 LSTM

Long Short Term Memory networks (LSTM) is a direct feedback based recurrent neural network, that is tackeling the problem of the vanishing gradient. The most basic version was proposed by [15]. It is able to store it's hidden state throughout time due to a memory cell in the hidden layer, an input gate and an output gate. Additionaly [13] introduced the forget gate and shortly later the peephole connections into the architecture [11]. This means that the forget gate, the input gate and output gate are connected to the present cell state C_{t-1}, while the output is connected to the updated cell state C_t. LSTM are based on a direct feedback. It possesses the capability to add or remove information to the cell state due to the functionality of the gates. This means that LSTM are like RNN chained one after another, but in contrast they are able to regulate the information that will pass on to the next state. Figure 2.3 shows the structure of an LSTM module.

[1] `https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg`

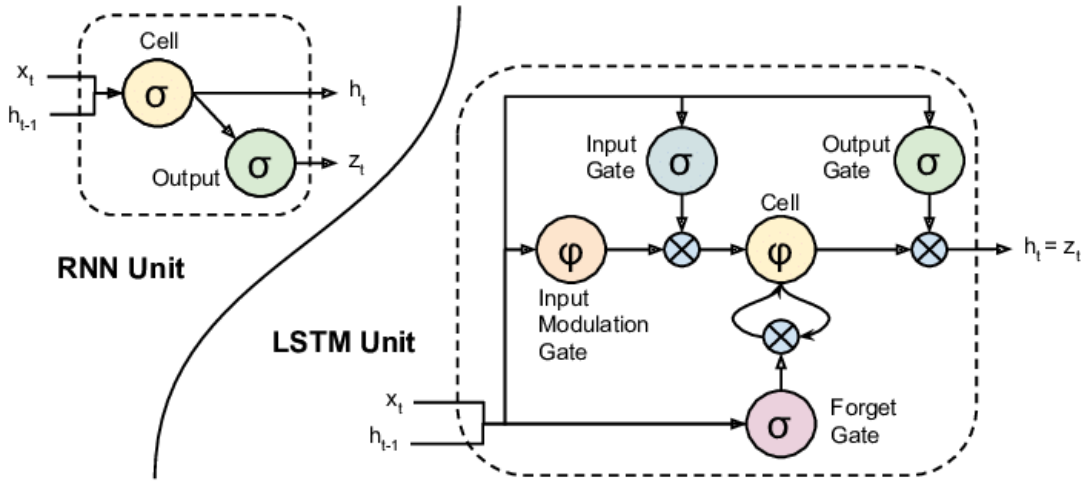

Figure 2.3: RNN Unit and LSTM Unit [7]

x_t describes the input, h_{t-1} the previous hidden state and h_t the new hidden state. The hidden state is holding the information of earlier transmitted inputs and is used for predictions. On entering the cell, the sum of h_t and x_t named s_t is calculated. This result is used as input for the gates and impacts their behaviour. Furthermore it also used to update c_t.
[13] describes the components of the cell as follows with W_h being the weight matrix and b_h the bias vector with $h \in \{o, i, f, C\}$.
The forget gate regulates what information gets discarded from the cell state. Even though the basic form of the LSTM can prevent the vanishing gradient problem like normal RNN it is still possible that the LSTM can suffer from the exploding gradient.That's why the forget gate was added. There s_t is passed through a sigmoid function, which returns values between zero and one. Is the result close to one c_{t-1} will be kept, is it close to zero it will be deleted.

$$f_t = \sigma(W_f \cdot (s_t, h_{t-1}) + b_f)$$

The purpose of the input gate is updating the cell state. Again s_t is pushed into a sigmoid function. Values near one decides which values are being updated, while values near zero are being discarded. This is multiplied with the result of s_t and shoved into a tanh function.The output of the tanh function lies between 1 and -1. This will help to regulate the network to prevent exploding values.

$$i_t = \sigma(W_i \cdot (s_t, h_{t-1}) + b_i)$$

$$\hat{C}_t = tanh(W_C \cdot s_t + b_C)$$

In the next step the old cell State c_{t-1} is updated to C_t. To this end c_{t-1} is multiplied with f_t to forget the information that was rendered uselsess. The result is added with the convolution of i_t and C_t, which decides how much the update is scale by s_t.

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \hat{C}_t$$

As depicted in figure 2.3 the cell state has a connection back to the forget gate. This is the key to preventing the vanishing gradient. This loop is called constant error carousel (CEC) and thus activity circulates in the CEC as long as the forget gate remains open. Just as the input gate learns what to store in the memory block, the forget gate learns for how long to retain the information, and once it is outdated to erase it by resetting the cell state to zero. This prevents the cell state from growing to infinity and allows the memory cell to store new incoming data, without overwriting the information from previous cells. The output gate will model the next hidden state h_t based on the cell state C_t. To prevent the cell state from blowing up, it will be transformed by a *tanh* function. This result is multiplied with the output of the *sigmoid*, which transformed s_t. This leads to the new hidden state h_t, which contains information on previous inputs.

$$o_t = \sigma(W_o \cdot (s_t, h_{t-1}) + b_o)$$

$$h_t = o_t \cdot tanh(C_t)$$

Both the new cell state c_t and the new hidden state h_t is carried over to the next time step. The last components are the peephole connections. When the output gate is closed, none of the gates have access to the CECs they control. The resulting lack of essential information may interfere network performance. The peephole connections allow the other gates to take a look at the cell state and therefore the gates gain access to the CEC [11].

$$f_t = \sigma(W_f \cdot (s_t, C_{t-1}, h_{t-1}) + b_f)$$

$$i_t = \sigma(W_i \cdot (s_t, C_{t-1}, h_{t-1}) + b_i)$$

$$o_t = \sigma(W_o \cdot (s_t, C_t, h_{t-1}) + b_o)$$

Therefore the ability to handle vanishing and exploding gradients enabled by the CEC, makes LSTM a very strong candidate for handling time dependent data. It is able to handle short term, as well as long term dependencies.

2.6 LSTM Autoencoder

Sequence prediction is not an easy task. The main reason is, because the input length can vary. This is especially troublesome for neural networks, because they are designed to process inputs with fixed length [34]. Also temporal ordering of the observations can make it challenging to extract features as input for supervised models, because good domain knowledge is needed. A LSTM autoencoder (LSTM-AE) is a variant of an autoencoder, that is able to process time sequences. It can handle univariate or multivariate time-series data. Like regular autoencoders it consists of an encoder and a decoder, but LSTM units are used for the layers. Because of this LSTM-AE can learn a compressed representation of sequence data and have been used on video, text, audio, and time series sequence data [32]. Many predictive modeling problems involving

sequences require a prediction that itself is also a sequence. These are called sequence-to-sequence models.

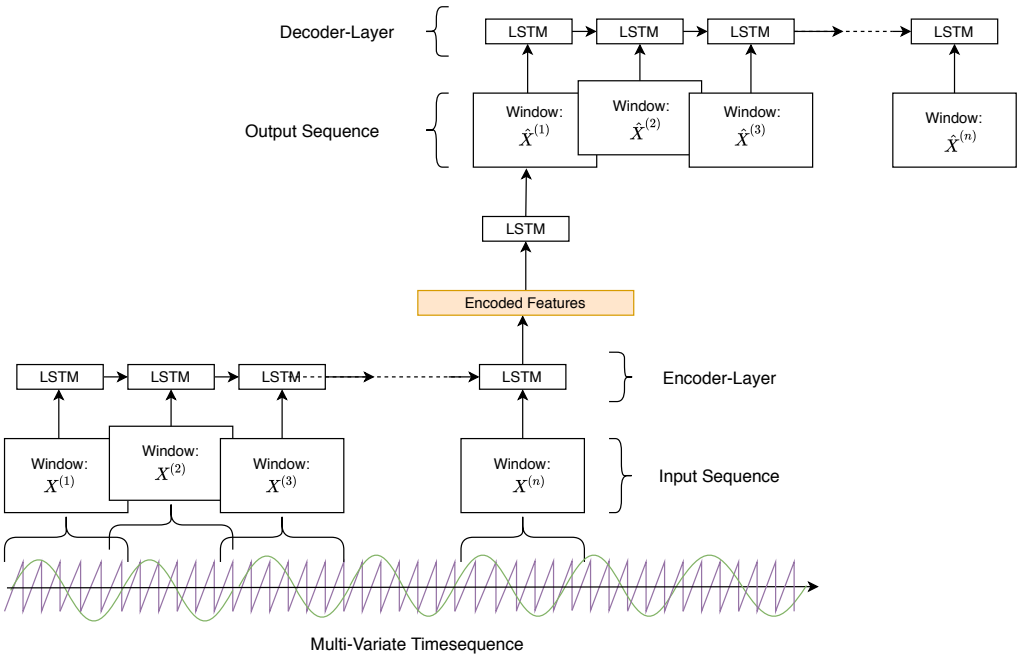

Figure 2.4: Principle of a LSTM-AE based on [34]

Figure 2.4 shows the process chain of a LSTM-AE. In this architecture, an encoder LSTM model reads the input sequence step-by-step. This means the input has to be split into sliding windows with a chosen length. After reading the entire input sequence, the hidden state/output of this model represents an internal learned representation of the entire input sequence as a fixed-length vector. This vector is then provided as an input to the decoder model that interprets it as each step in the output sequence is generated. The output sequence is again partitioned into windows. This is the main difference compared to an normal LSTM model, which doesn't have an encoded representation of the features [35]. For Anomaly detection the LSTM-AE will be trained on normal data. For the predicted sequence the error will be measured in order to explore anomalous data.

3 Case Study

In this chapter the used methodology and case study is presented. The first section 3.1 covers processing and cleaning up the data set. The training of the LSTM-AE requires the data to be non anomalous. Therefore the labelling of the measurements is explained in detail in section 3.2. After that the features of the data set are reorganized in order to fit as input for the LSTM-AE. For that purpose a windowing algorithm is used and described in section 3.3. Furthermore section 3.3 is also covering the implemented architecture of the LSTM-AE itself. In order to achieve the best outcome, hyperparameter tuning is explained in section 3.4. After that the evaluation methods are analyzed in detail in section 3.5. Finally the training schedule and initialization of the parameters are outlined in section 3.6.

3.1 Data set and preprocessing

In this chapter the given data set is elucidated and it is explained how the data is embedded in the model. The data set is provided by the Berliner Wasserbetriebe (BWB). Inside the sewage network of Berlin sensors are installed, in order to measure the water-level of different types of canals. These are the combined sewer overflow (CSO), level measurement (LM) and the flow rate (FR). On top of that three rain-measurements (RM) were made by measuring stations which are located around Wilmersdorf. The measurements were taken in a period from the 1th of May until the 30th of October in the years 2016 and 2017. The resolution of the records is following a five minutes interval. This means, that in each year, 183 days with 288 measurements each, were recorded. But it is still possible, that the sensor readings are corrupted, because of noise or other kind of disturbances. These measurements are indicated with not a number (NaN) in the data set. In order to interpolate the missing measurements, the NaN will be replaced with the arithmetic mean of the data points before and after the missing values.

- Combined sewer overflow (CSO): There are three different time series given for the CSO and the unit is $\frac{m^3}{s}$. The three different time series are capturing the possible overflows in the sewer leading to the sewage treatment plants in Stahnsdorf, Ruhleben and Wassmannsdorf. For further analysis the used CSO will be the combined sum of the measurements of all three time series. The CSO events are the main objective of the analysis of this thesis. It is explored, under which conditions unusual overflows are occurring.

- Level measurement (LM): The LM is measured at three stations in Wilmersdorf. The measurements are described in meters above standard zero (meter above normal null - mNN). LM is the main feature of the anomaly detection, because it correlates the most with the development of the CSO after the precipitation. The correlation coefficient between CSO and LM is 0.37, whereas the correlation coefficient between CSO and FR is just 0.17. The highest correlation has in fact rain and CSO with a value of 0.52, which makes sense, because the canalisation is supposed to be able to handle the every day water usage of the city.

- Flow rate (FR): FR is measured at three locations. At the sewers leading to the treatment plants in Ruhleben, Stahnsdorf and Wassmannsdorf. The unit is given as $\frac{l}{s}$. For further analysis, the FR will be used as combined sum of the measurements of all three stations. The FR is used as the second feature but it correlates not as strong compared to LM with the CSO. However, there is still a relevant influence for the anomaly detection.

- Rain measurements (RM): The precipitation is measured at the same three locations as well and is given in millimeter (mm). This one is summed up over all 183 observed days. Therefore at first the difference has to be calculated in order to get the precipitation amount of each day for every five minutes. At days with a high amount of precipitation there will definitely be CSO events. Therefore it would be sensible to analyse under which circumstances CSO events occur, that are not tied to a huge amount of precipitation.

After parsing the data, it will be stored in arrays. The measurements are plotted in order to receive a rough overview of the structure and patterns of the data. Based on the plots, measurements matrices were generated for each of the four types that contain the data points. Those matrices have the dimension 183x288. This means, that for each of the 183 days, there are 288 measurements in their respective units available, as the time resolution is 5 minutes. The partitioning in days is chosen, because a periodic pattern of the data structure emerged, while taking a look at the plots. Figure 3.1 shows an exemplary instance of a day based on the measurements. The red line indicates the CSO, the blue line the precipitation and the green the LM. In this case there is no overflow event as there is no rain.
It can be observed, that in the morning between 5am and 8am the LM and FR are increased. This is a periodic pattern of each day, which may correlate to the pattern of living of the inhabitants of that district. Figure 3.2 is showing an example day for a CSO event. It can be seen that the level of the LM reaches a peak around 10am due to the periodic repetitions of each day. After a short time a short downpour is setting in around 12am with max value of 3.1 mm. This lets the level of the LM increase steeply up to 33 mNN. This is were the CSO event happens. The combined sewage was already filled with the wastewater of the residents activities. Adding the volume of the heavy rain fall on top leads to an overflow and the activation of the sewage pump to redirect the wastewater into adjacent water bodies with up to 12.5 $\frac{m^2}{s}$.

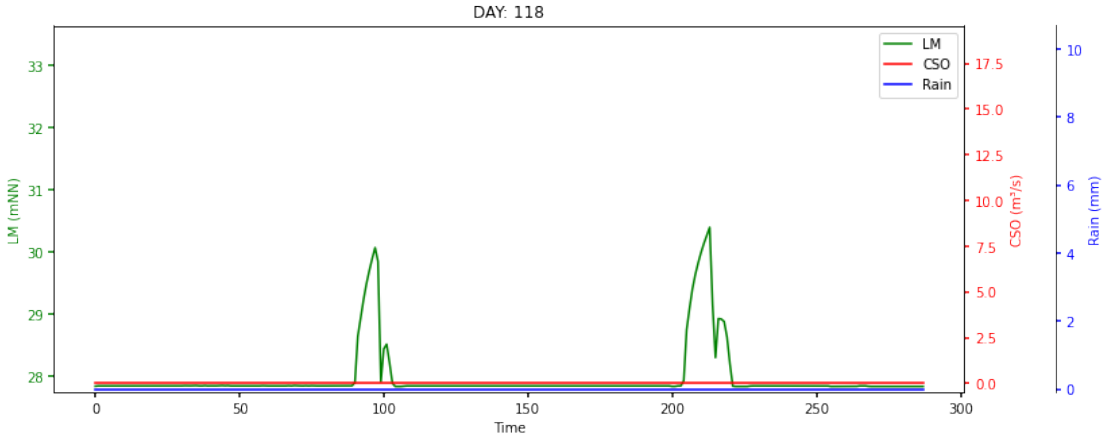

Figure 3.1: Day without CSO overflow and with the periodically recurring patterns of the inflow chamber level recognizable from the plots

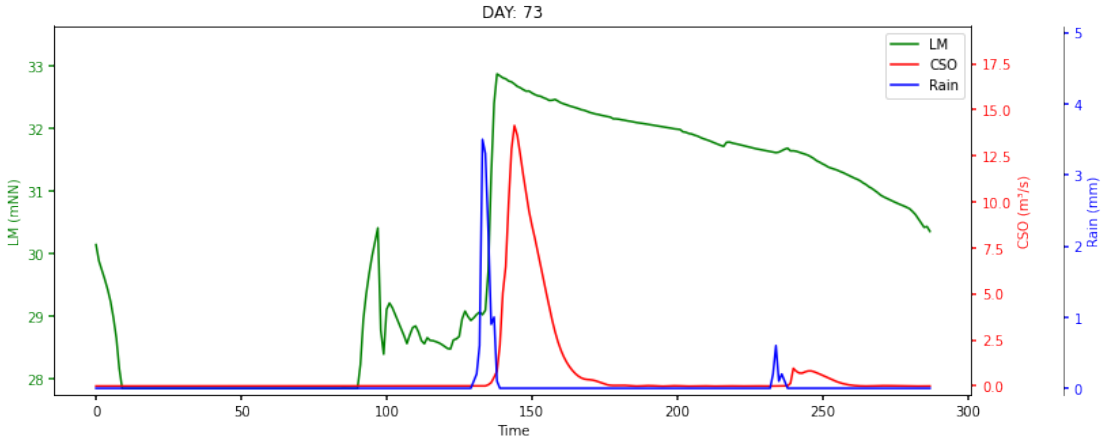

Figure 3.2: Day with CSO Event

Visualising the CSO Events with plots is useful, because the data set needs to be split manually in non-anomalous and anomalous data points, which will be discussed in the next section.

3.2 Labelling

LSTM-AE are part of semi-supervised classification approaches. This means they have to be trained on non-anomalous data. Therefore the measurements of data sets have to be labelled as normal or anomalous. In this case every time the CSO is reaching a certain threshold, it is labelled as anomaly.
The data has to be labelled by hand and therefore the preprocessed data is plotted in

order to find a suitable threshold. In figure 3.1 the progression of the CSO is shown in red and the the LM is shown in blue over the span of one day. In this example the CSO reaches peak values up to 25 $\frac{m^2}{s}$. In the left figure the CSO reaches a value of 5 $\frac{m^2}{s}$. The LM is following as well. It is obvious that higher CSO correlates with higher LM. In order to achieve a rare-event-classification we want to limit the amount of CSO events being considered anomalous. Therefore CSO events above $1.5\frac{m^2}{s}$ are labelled anomalous. Days containing one of those events will also be labelled as anomalous. The daily time frames are chosen in order to capture the context, in this case temporal dependencies, and not just capture point anomalies, if we would consider each measurement on its own. This means out of 183 measured days, 57 are labelled anomalous.

3.3 Model Setup

The LSTM-AE is constructed with the help of the predefined sequential model by Keras [45]. For each Layer of the neural network Keras provides different architectures, which have a range of tunable parameters. In this case LSTM cells are used as neurons. The LSTM-AE expects a time sequence as input or more precisely a matrix with three dimensions. In order to provide that, a sliding window approach is used, which will model the features of the data set into a time sequence. And thus leads to an input matrix in the shape of:

number of windows **x** *number of data points per window* **x** *number of time series*

.

Data: Normalized Features
Result: Windowed Data for LSTM
initialize window size;
for $i <$ *window size* **do**
  window.append(feature1, ..., featureN);
  count = count + 1;
  if *count == stepsize* **then**
    windowlist.append(window);
    count reset;
  else
    continue;
  end
end

Algorithm 1: Windowing of Features

In algorithm 1 the pseudo-code for creating the windows is shown. As input the normalized features like FR, LM, FR and RM are expected. For the normalization the highest value of the corresponding time series is chosen normalization factor. First a window

size for a time period is chosen. The size is dependent on the first sighting of the plots of the data. For instance periodic behaviour for each day of the measurements were detected. More precisely every six hours patterns are starting to repeat. Therefore the window size is set to 72. Furthermore a step size has to be set. The step size indicates the number of minutes the windows is moved. In this case one was chosen with each step counting for 5 minutes, which results in an overlap of 355 minutes. A bigger step size is resulting in more coarse window overlapping and thus in less precision. After processing the data into the right input format, it is plugged into the LSTM-AE.

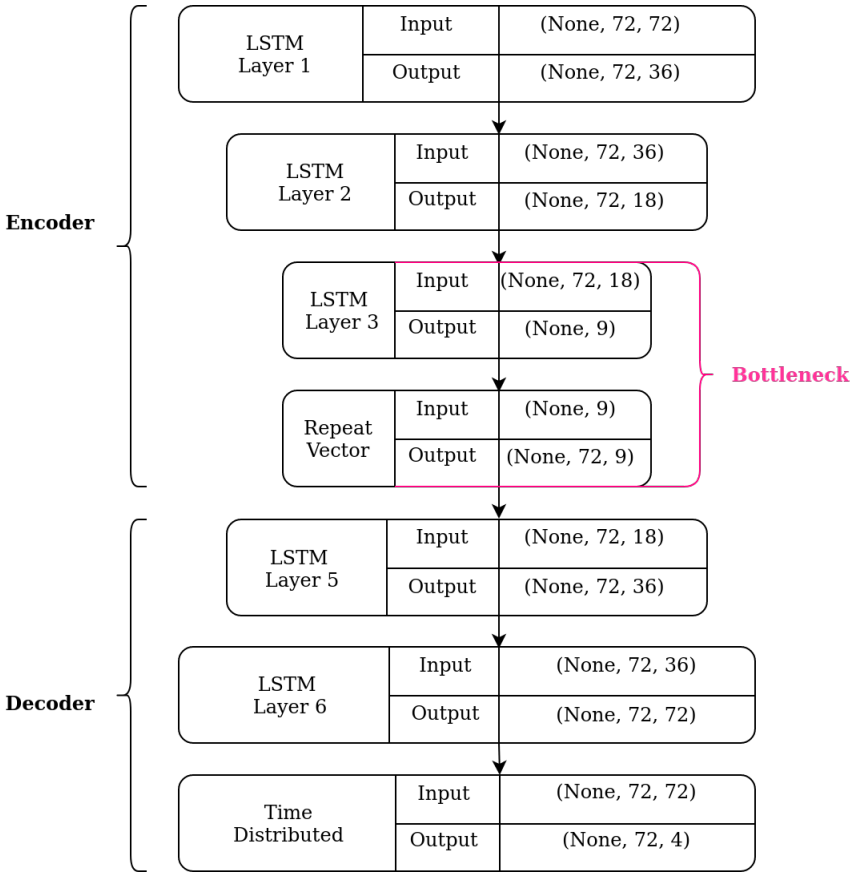

Figure 3.3: LSTM-Autoencoder architecture build with Keras

Figure 3.3 is showing the architecture of the implemented LSTM-AE. It has a typical stacked autoencoder structure with three layers building the encoder, three layers building the decoder and the bottleneck in between. The decoder and the encoder have three LSTM layers each. The input layer has as much nodes as the multiple of the size of the window. The shown repeat vector provides the output of the bottleneck, which is the feature vector. More precisely it takes the encoded feature vector and duplicates it in order to have the correct dimension for the decoding layers. The time distributed Layer at the end of the decoder creates a vector of the length equal to the number of features

outputted from the previous layer. It duplicates it as often as the number of time series, that are given as input to the windowing algorithm. At the end of the decoding phase there is a time distributed layer, which is transforming the three dimensional matrix back into a two dimensional windowed time sequence. As stated further above LSTM layers provided by Keras are used. One layer has 25 arguments, of which five were adjusted for this implementation. The first is the type of layer, that is used. As already stated and depicted in figure 2.5 LSTM layer, repeat vector and time distributed layer are used. For the LSTM Layer the rectified linear unit (ReLU) is used as activation function. The input shape of the 3 dimensional input matrix and the return sequence are set to true. It is important that the return sequence is set to true, because this makes each cell per time step emit a signal.
After setting up the model, the initial hyperparameter like epochs, batch size, shuffle and splitting size are chosen.

3.4 Hyperparameter

As described in 2.3 there are mainly four hyperparameter to consider while constructing the LSTM-AE. Those are the number of layers, the number of nodes in hidden layer, the number of nodes in decoding/encoding layers and the loss function. Three more parameters are set for the actual training - the number of epochs, the batch size and the learning rate. The tuning of those hyperparameter is done in two steps. First the ranges and values are chosen by research. In the second step the parameters are updated by the grid search method of SciKitLearn. Those steps are chosen, because grid search is a valid method for hyperparameter optimization, but the computation can get very expensive, if the search space is chosen to large [6]. Therefore the number of possible values is limited by constraints found through research, which is explained in the following.

- Number of layers: As stated in the autoencoder section 2.3, one layer is already enough to represent an arbitrary non linear function. But deeper networks ensure, that the network is able to learn and not just represent the function, which means a better generalization [24]. Unfortunately there is no perfect answer for the best number of layers [31]. Making the network infinite deep isn't an option either. This can lead to overfitting, especially for smaller data sets and at some point the gradient will vanish, even though the LSTM units will prolong it. Therefore the number of layers is dependent on the number of nodes in the input layer.

- Number of nodes: In this case the number of nodes in the hidden layers is dependent on the input layer. In this case the input layer has 72 neurons, which is the length of the window. The layers after that are divided by half until the smallest layer has an odd number, in this case 9.

- Number of nodes in bottleneck: The number of nodes in the bottleneck are implied by the number of layers and the number of nodes of the input layer, therefore in this case it is 9.

- Loss function: In this scenario the mean squared error (MSE) is used as loss function. As described in [22] the MSE is very often a choice for time series forecasting, in order to compare the quality of the forecast with the ground truth.

- Number of epochs: The number of epochs determines how many rounds the architecture is trained on the data set. Ideally it should be enough that the loss is converging near to zero, but not much more, because this can lead to overfitting. In this case the range is chosen after the elbow knick up to 100 epochs.

- Batch size: In this case a mini-batch approach is chosen. This means that the batch size is the number of samples from the training set, that is used to estimate the error gradient before the model weights are updated. A too big batch size may lead to worse generalization, too few samples may lead to a worse accuracy [2]. For computational purposes the batch size is chosen to match the power of the CPU and is therefore often a number by the power of two [19]. In this case a batch size in the range from 2^2 to 2^7 is chosen for grid search, which also corresponds to the window size of 72. Everything above exceeds the number of data points from the data sets.

- Learning rate: The learning rate controls how fast the model will adapt the weights of the given tasks and is a parameter of the optimizer. Lower learning rates indicate, that smaller changes are made to the weights each update. Thus more epochs are required. Bigger rates on the other hand can result in too fast convergence and therefore to suboptimal solutions [31]. The learning rate is computed by grid search and set to $0.1 \cdot 10^{-4}$.

- Optimizer: Optimizers are used to change the learning rate and weights of the network, in order to reduces the loss. The adaptive moment estimation optimizer (ADAM) is used for the training process. Compared to other optimizers it computes different learning rates for different parameters and it is chosen, because it works generally well without depending on hyperparameter tuning so much [20].

The results of the research and grid search are given as complete overview of the parameters in table 3.1.

	Layer	Input Nodes	Bottle-neck	Batch Size	Epochs	Optimizer	Learning Rate
Range	2 - 10	72 - 288	128	2^2 - 2^7	25 - 100	ADAM	0.1 - $5 \cdot 10^{-4}$
Chosen	6	72	9	2^5	50	ADAM	$1 \cdot 10^{-4}$

Table 3.1: Chosen hyperparameters - candidate values and chosen values

More about the influence of the hyperparameter tuning and different configurations and the performance of the LSTM-AE are given in the section 4, which is covering the results.

3.5 Evaluation Methods

The proposed model is a binary classification model. Therefore a confusion matrix is used as one tool to evaluate the results of the testing of the architecture. Even though the approach is semi-supervised, class labels are applied implicitly through the chosen threshold in the prelabelling phase. Thus it is possible to compare the predicted anomalies against the actual anomalies. In figure 3.4 the confusion matrix is depicted. It shows how the predicted and actual anomalies are related to each other. The true positive (TP) is achieved, if the actual and predicted anomalous data points are matching. The true negative (TN) is achieved, if the actual and predicted non-anomalous data points are matching. The false positive (FP) indicates a positive prediction of an anomaly, while there is actually no anomaly and vice versa in case of the false negative (FN). The other evaluation metrics in the confusion matrix are true positive rate (TPR) or recall, false positive rate (FPR) precision, miss rate, specificity, accuracy and the F1-score. The TPR or recall measures the percentage of anomalies in the data, while the precision indicates the percentage of true anomalies among all anomalies detected. The FPR measures the percentage of non-anomalies in the data, that were labelled as anomalous. The accuracy seems to be the obvious choice, because it gives the percentage of all the correctly identified cases. But it is most used when all the class distributions are similar. In this case the class distributions are imbalanced and therefore the F1-score will be taken into account as well. It gives the harmonic mean between precision and TPR and therefore extreme values will be punished more. The miss rate gives an indication about all the negative samples, that were falsely declared as anomalous. The last value is the specificity, which gives an insight about how much of the non-anomalous data has been predicted as non-anomalous.
The receiver operating characteristics (ROC) curve and the area under the curve (AUC) are chosen as graphical evaluation methods. This methods are feasible, because the task requires one versus rest classification. The ROC curve is plotted with TPR against the FPR, which is the same as $1 - Specificity$. Therefore the ROC curve displays a probability distribution. The AUC on the other hand is, like the name indicates, the area under the ROC curve. It is a measurement on how much the model is able to distinguish between classes. The bigger the area under the curve, the better the model is predicting. Therefore a value near to one would be optimal, while a value close to zero is indicating the worst measure of separability. A value of 0.5 indicates that the model has no class separation capacity.

3.6 Training schedule

After the hyperparameter are chosen and applied to the network, the model is trained. For the training a GPU server with Tesla K80 with 12 GB GDDR5 memory, 4vCPUs and 15GB RAM is used. As described in 3.1 there are measurements from 2016 and 2017. The training is done on the data set from 2016, while the testing is done on the data set from 2017. The testing set is chosen to be the whole time period from 2017,

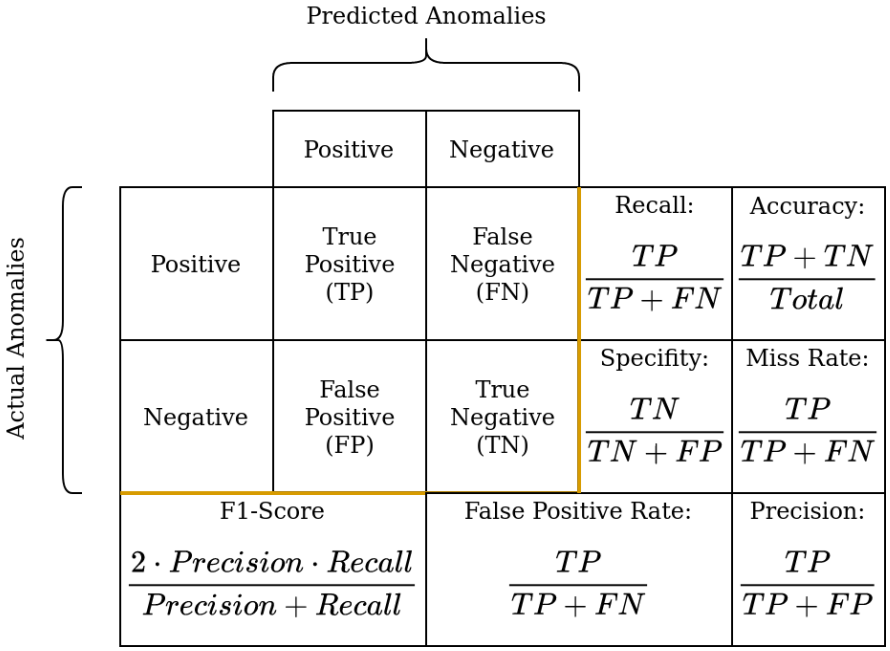

Figure 3.4: Confusion Matrix

because this is reflecting the seasonality of the data. During training the data set from 2016 is split into training and validation set with a ratio of 70 to 30. Normally the validation split in Keras is done by taking the last 20% of the input array. But in this case we have a time sequence. That means it doesn't make sense just to validate the last 20% of the data set of 2016, because it doesn't take the time dependency and seasonality into account. Therefore the shuffle option in the Keras model is activated, in order to select random windows for the validation set. Thus the absolute numbers are 12360 training samples (windows with 72 samples each) and 3068 validation windows for each input sequence. The number of the training samples and the the nature of the chosen architecture results in 247937 parameters to train. After the training is done, the model was applied to the data set of 2017. The output forecast sequence of the model is than compared to the data set of 2017 with the help of the mean absolute error (MAE) for each window:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i|$$

In this case n is the number of windows, y_i the predicted value and x_i is the original value from the measurements. This leads to a density function that gives the numerical error for each data point. Finally, a threshold is chosen on which cutoff error an forecast is considered anomalous and on which it is considered normal. The described ROC curve is used in order to find this threshold. Unfortunately most preexisting functions are expecting already labelled data, or data that isn't segmented into windows. Therefore in this thesis possible threshold candidates are chosen by iterating over all error values

(rounded to the third value after comma). For each value a dictionary is constructed with the information of the current threshold and of how many predicted windows are labelled as anomalous. With the help of those dictionaries the confusion matrix is applied to each threshold and the comparison of the corresponding windows, that are labelled as anomalous, with the original data set is done. Therefore the TPR and FPR are calculated and the ROC-curve computed. The ROC-curve in combination with domain knowledge is then used in order to determine the threshold. The explicit choosing and reasoning are described in more detail in the result section 4.

4 Results & Evaluation

In this section the results and their evaluation are presented. At first the results for the training i.e. model loss and parameter tuning are displayed in section 4.1. This includes the final model with the results of the validation set as well as the results of the unseen test set and the results of the performance of the validation sets on the models that are used for parameter tuning. After that in section 4.2 there is an evaluation and discussion about the results with the evaluation tools that are described in section 3.5.

4.1 Training Results

In this section the results of the training process are presented. It is structured as follows. For each category the results of the model with the best model parameters is shown for the validation and test set. In order to get an idea of the influence of the hyper parameter tuning the same plots is placed sided by side to make them easier to compare. At first the loss of the training process is presented. After that the density function and the mean absolute error of the predicted data set are compared to the original data set and the chosen threshold is shown. Based on that the found indices of the as anomalous detected days are discussed and the FP, FN, TP and TN are calculated. Finally the confusion matrix is applied to the outcome.

4.1.1 Training Loss

Figure 4.1 is showing the loss of the training and validation set depending on the number of epochs. The training error is marked in blue and the validation error is marked in orange. The number of chosen epochs was 50, with a batch size of 32. The training error starts to diminish very fast, after 5 iterations to a value under 0.01. After that the reduction of the error isn't as strong for the next epochs. It converges around a value of $1.49 \cdot 10^{-4}$ after epoch 25. The fast convergence is is due to the small data sets and the periodic pattern of the data. The validation is smaller than the training error in the beginning of the training. In most machine learning application the training error is supposed to be smaller than the validation error. But this is due to the properties of the Keras Framework. As stated in the Keras FAQ [44] weight regularizations like the MSE are turned off at validation time, which is reflected in the higher loss in the training at the beginning. Next to figure 4.1 is the same loss function depicted, but with a much higher resolution. The range of the y-axis was scaled down by a factor of 100. The error is still degrading, but by a very small amount. Therefore 50 is chosen as cutoff epoch, in order to prevent overfitting. There are some spikes present in the error curves, due to the small resolution.

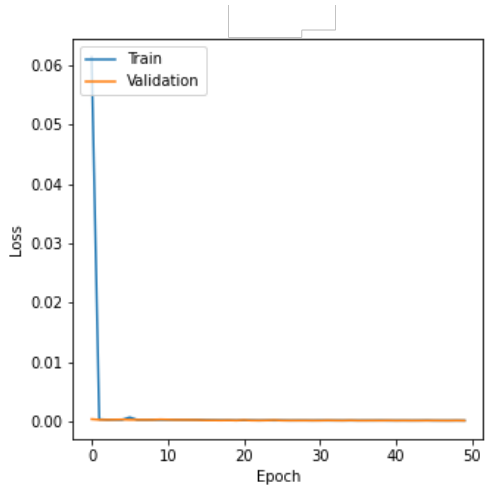

Figure 4.1: Training and validation loss for 50 epochs

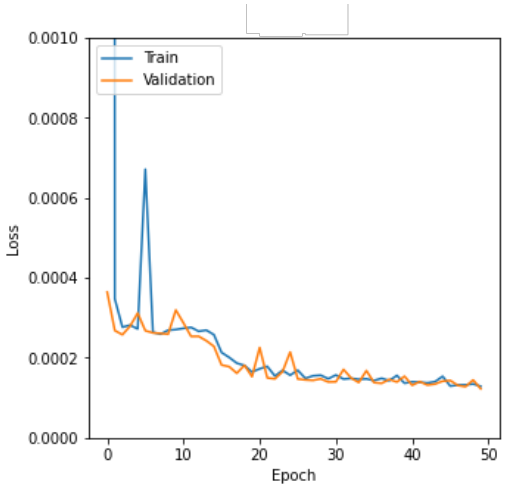

Figure 4.2: Training and validation loss for 50 epochs scaled down by a factor of 100

4.1.2 Prediction Error

Now the prediction error will be presented and visualized. The prediction error is the comparison between the original data set and the predicted data. In both cases the data sets are consisting of windows. As stated in 3 the mean absolute error (MAE) was chosen as comparison method. Figure 4.3 shows the density distribution of the MAE dependent on the number of samples. The range of the error values goes from 0 to a max value of 2.6 $\cdot 10^{-3}$. It can be observed that there is a peak around a MAE of 1.2 $\cdot 10^{-3}$ and most of the data has a smaller or equal MAE or more precisely 85.3%.

4.1.3 ROC Curve and AUC

The ROC curve gives the ratio between FPR and TPR. For each iteration of the MAE a threshold was chosen and the numbers of the corresponding FPR and TPR were calculated. The MAE ranges from 0 to 2.6 $\cdot 10^{-3}$ and 2600 different thresholds between those limits were used in order to calculate the TPR and the FPR. The result is seen in figure 4.4. The TPR is on the X-axis and the FPR on the Y-axis, both are ranging from 0 to 1, because both are indicating probabilities. As discussed in the methodology section 3 a linear line would indicate no separability capacity. However the depicted curve is way above the diagonal line which indicates a very good diagnostic ability. This is further strengthened by the AUC, which reaches a value of 0.91. Therefore this is indicating that the chosen model has an excellent measure of separability and it is up to the given task of detection the anomalies of the data set.
However the question of the best threshold remains. A perfect classifier would give a value were the TPR is 1 and the FPR is 0. As seen in figure 4.4, this is not a given option.

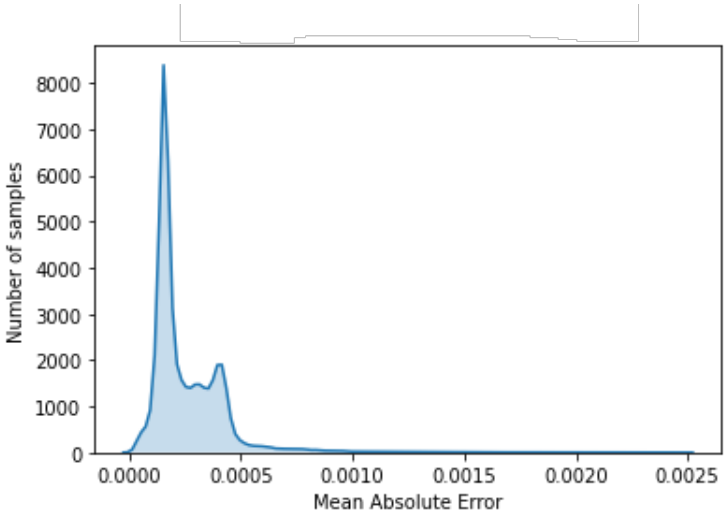

Figure 4.3: Density function of the mean absolute error

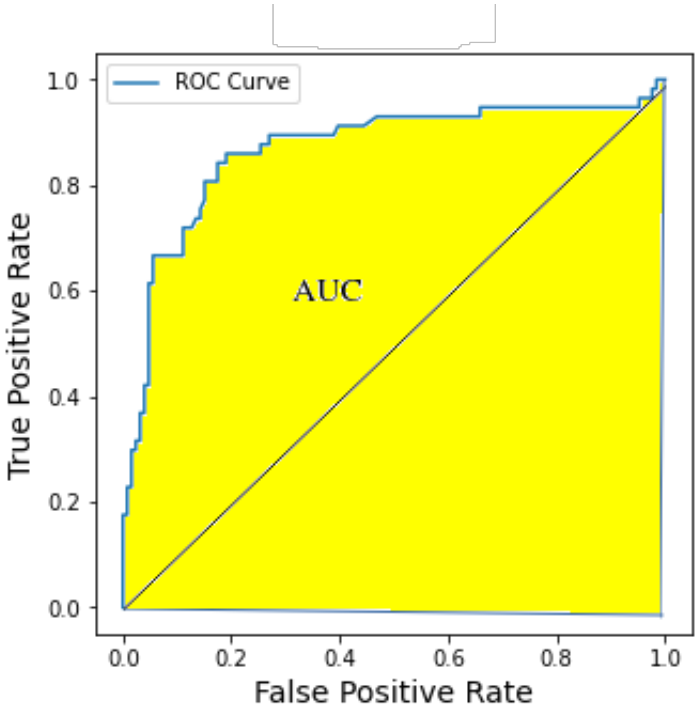

Figure 4.4: ROC curve and AUC

Therefore it is necessary to determine how to find the best trade off between the TPR and FPR. Table 4.2 shows different selected thresholds and their TPR and FPR values, as well as TP, FN, FP and TN in absolute numbers. The first two rows are showing the

edge cases with 0 TPR and 0 FPR respectively. This is were domain knowledge comes into play. Every time a CSO event occurs, the wastewater will be pumped into nearby water bodies. On the other hand, it is more important for the sewer system that the treatment plants don't get overcharged [16]. This should be avoided, therefore a ratio with a preference for the TPR closer to one is chosen, with a FPR as low as possible depending on the TPR. On top of that the FN should be as low as possible as well, because each instance that is labelled as event, but is in fact an non-anomalous event, could lead to unnecessary drain of waste water in to nearby water bodies.

4.2 Evaluation

In this section the results, which are given in section 4, are being evaluated. Therefore the evaluation methods discussed in 3 are applied. First the parameters of the confusion matrix are calculated for several threshold candidates. After that those values are analyzed and interpreted in regard to the chosen architecture and the tuned parameters. One of them is chosen for the final assessment of the LSTM-AE.

4.2.1 Confusion Matrix

The last step is finding a suitable threshold for the number of false positives, that we are accepting in trade for the number of actual detected anomalies. For this task the ROC curve, as seen in figure 4.4, is used as tool to find 14 suitable candidates out of 2600, that are then compared to each other. For the comparison of the candidates the parameters of the confusion matrix like TPR, FPR, precision, specificity, accuracy, miss rate and the F1-score are used and which are presented in table 4.2. In order to do so, at the very first step the FP, TP, FN and TN for the possible thresholds are calculated, which are presented in table 4.1.
Overall 14 thresholds are chosen along the ROC-curve. The thresholds and their respective parameters of the confusion matrix can be seen in table 4.2. In the beginning the edge cases of the ROC-curve are calculated. First the case where no false positives are detected, is analysed. Taking a look at the ROC-curve shows that the curve isn't continuous, but has steps inside. That means there are several TPR where the corresponding FPR has a value of zero. Therefore the highest TPR in relation to FPR of zero is chosen. This is also done for the other brackets of the curve, that will be analyzed. For the same TPR the lowest FPR is chosen. For the TPR being one, the TP is of course 1 as well. The miss rate is 0, because the FN are 0, which is desirable. But it can be observed, that specificity, accuracy, precision and F1-score are pretty low due to the high FPR. This means, everything was marked as anomalous, which is not a very good threshold for classification. Therefore it can be seen that the data is not linear separable.
As seen in figure 4.4, there are several plateaus to consider. For a first assessment thresholds for TPRs in 0.5 steps were evaluated. It can be observed in 4.4 and in 4.2, that there is a step descent for a TPR below 0.66, which corresponds to a threshold of 7.311 $\cdot 10^{-4}$. This means threshold with a lower TPR are listed as examples in the table, but

Threshold	TP	TN	FP	FN
$2.311 \cdot 10^{-4}$	57	2	124	0
$4.448 \cdot 10^{-4}$	52	76	50	5
$4.502 \cdot 10^{-4}$	51	80	46	6
$4.776 \cdot 10^{-4}$	49	94	32	8
$4.957 \cdot 10^{-4}$	49	97	29	8
$\mathbf{5.080 \cdot 10^{-4}}$	**49**	**102**	**24**	**8**
$5.203 \cdot 10^{-4}$	43	108	18	14
$6.181 \cdot 10^{-4}$	41	112	14	16
$7.312 \cdot 10^{-4}$	38	119	7	19
$7.734 \cdot 10^{-4}$	34	120	6	23
$8.481 \cdot 10^{-4}$	31	120	6	26
$8.615 \cdot 10^{-4}$	27	120	6	30
$9.672 \cdot 10^{-4}$	24	121	5	33
$1.353 \cdot 10^{-4}$	10	126	0	147

Table 4.1: Tresholds and their respective rates

not analyzed further. However another region of interest is between TPR values of 0.85 and 0.9, because this is the two nearest plateau before the ROC-curve gets to its knick, starting below 0.85. Therefore in this region the granularity of the analyzed parameters is reduced. This is done, because it can be observed in 4.2 that the TPR stays in a region of 0.8596 for several chosen thresholds. Therefore the other parameters can now be evaluated for this region. It is obvious that this also means that the TP stays with 49 and therefore the FN with 8. The FP values are ranging from 85 to 102 and the FP values from 41 to 24, which is depicted in table 4.1. In this case the choice falls on threshold $5.080 \cdot 10^{-4}$ for having the TP with the lowest FP in the analyzed region of interest.
For the parameters of table 4.2 is a similar picture shown. The miss rate is stable at 0.140, because it corresponds directly to the TPR. Again for a threshold of $5.080 \cdot 10^{-4}$ the specificity and precision. While miss rate, specificity, and precision are behaving linear to the TPR and FPR. The accuracy and F1-score are behaving more quadratic and have therefore a bigger influence on the decision. It can be observed in 4.2, that both values are ascending for a descending TPR until they are reaching a peak. After that their values are descending again for a smaller TPR. This is, because the accuracy and F1-score are putting TPR and FRP directly in relation to each other. Furthermore the F1-score has its highest value at 0.754 for a threshold of $5.080 \cdot 10^{-4}$. This is even better than having the highest value for accuracy, because the f1-score takes the influence of all the used time series like LM, CSO, FR and rain into account. Overall a threshold of $5.080 \cdot 10^{-4}$ is chosen as discriminant, which can be seen in bold letters in the tables 4.1 and 4.2 with its respective parameters.
Finally based on the chosen threshold of $5.080 \cdot 10^{-4}$ the prediction of the data set of 2017 is compared to the actual data set of 2017, where 49 anomalies out of the 57 are found.

Threshold	TPR	FPR	Specificity	Accuracy	Precision	F1-Score	Miss Rate
$2.311 \cdot 10^{-4}$	1.000	0.984	0.015	0.322	0.314	0.478	0.000
$4.448 \cdot 10^{-4}$	0.912	0.396	0.603	0.699	0.509	0.654	0.087
$4.502 \cdot 10^{-4}$	0.894	0.365	0.634	0.715	0.525	0.662	0.105
$4.776 \cdot 10^{-4}$	0.000	0.253	0.740	0.781	0.604	0.710	0.140
$4.957 \cdot 10^{-4}$	0.000	0.230	0.769	0.797	0.628	0.725	0.140
$\mathbf{5.080 \cdot 10^{-4}}$	**0.859**	**0.190**	**0.810**	**0.825**	**0.671**	**0.754**	**0.140**
$5.203 \cdot 10^{-4}$	0.807	0.150	0.849	0.836	0.707	0.753	0.245
$6.181 \cdot 10^{-4}$	0.719	0.111	0.888	0.836	0.745	0.732	0.333
$7.312 \cdot 10^{-4}$	0.666	0.055	0.944	0.857	0.844	0.745	0.403
$7.734 \cdot 10^{-4}$	0.596	0.047	0.952	0.841	0.850	0.701	0.456
$8.481 \cdot 10^{-4}$	0.543	0.047	0.952	0.825	0.838	0.659	0.526
$8.615 \cdot 10^{-4}$	0.473	0.047	0.952	0.803	0.818	0.650	0.578
$9.672 \cdot 10^{-4}$	0.421	0.039	0.960	0.792	0.827	0.558	0.834
$1.353 \cdot 10^{-4}$	0.175	0.000	1.000	0.7431	1.000	0.298	0.936

Table 4.2: Candidate thresholds and parameters of confusion matrix

This is also depicted in 4.5, where the MAE and the threshold are shown together. The MAE is dependent on the number of samples shown in blue. The threshold is depicted in orange. It can be seen, that the threshold is covering most of the samples beneath it, while foremost getting exceeded by the samples with the highest error rates and therefore minimizing the false classification of ambiguous samples.

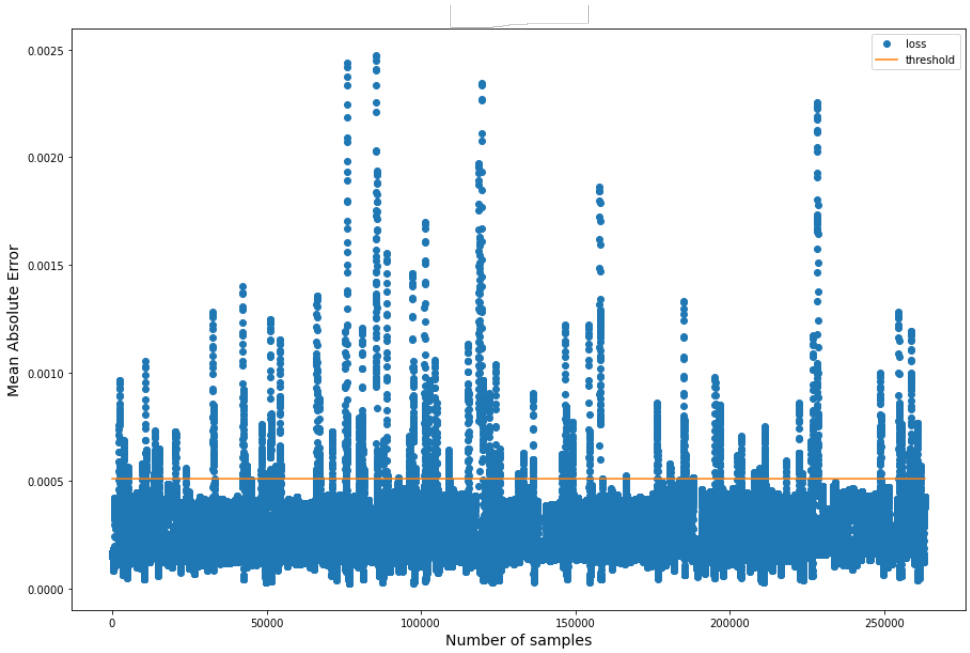

Figure 4.5: Threshold in comparison to the MAE

5 Conclusion

In this thesis a LSTM-AE for anomaly detection in the context of CSO events is presented. CSO events are a huge contributor for water pollution. Therefore the main motivation for developing a LSTM-AE architecture is predicting CSO events before they occur, in order to prevent waste water being pumped into adjacent water bodies instead of the nearby sewage treatment plants. The basis for developing this model consists of two data sets of the water levels of the sewer network of Berlin measured by installed sensors during the summer months.
The forecast model needs to be able to handle the attributes of the data sets. First there are temporal dependencies, because the data set includes measurements of the water level in 5 minute steps ranging from May to October. Secondly there are several water levels measured like LM, FR and CSO as well as precipitation. Therefore it needs to be able to handle multivariate time series. Quantitative analysis revealed that the data is not linear separable. Those conditions are covered by the LSTM cells of the architecture. The autoencoder structure is capturing the important features of those attributes into a feature vector, which is used to produce the forecast of the time series. The architecture is trained on non-anomalous data. Therefore CSO events are labelled anomalous, making it a semi-supervised model. Another step for the preprocessing of the data is the segmentation of the time series into overlapping windows in order to capture the temporal dependencies, which is the needed shape for the LSTM units. After that the hyperparameter tuning is conducted. The training/validation and testing are performed on time series of subsequent years respectively. The forecast is compared to the ground truth measurements and thus calculating the MAE.
The overall calculated MAE is low with a maximal value of $2.47 \cdot 10^{-3}$ and the majority of samples having a value around $0.1 \cdot 10^{-3}$. Based on this, the parameters of the confusion matrix are calculated and the ROC-curve computed, which gives a first impression of the separability capabilities of the model due to the AUC. The AUC reaches a value of 0.91, which marks the LSTM-AE as excellent classifier. Based on the ROC-curve a threshold is chosen for classification by incorporating domain knowledge in order to find the most suitable TPR to FPR ratio. In the end a threshold of $5.080 \cdot 10^{-4}$ is selected, taking foremost the importance of few FN and a high TPR, while secondly maintaining a FPR as low as possible. Respectively a high F1-score of 0.754 while keeping a low miss rate of 0.141 are taken into account. This means 49 out of 57 CSO events were detected. Thus it is shown, that using a LSTM-AE for anomaly detection can successfully be leveraged for the task of predicting CSO events before they happen.

5.1 Future Work

In this section future work is presented. As seen in the first part of the conclusion the LSTM-AE is already an excellent model for anomaly detection for CSO events, based on the evaluated metrics. But there is still room for improvement. Therefore in the following approaches are presented in order to increase the AUC as well as the TPR to FPR ratio and therefore increasing the models separability capability as well as the F1-score.

Gated Recurrent Unit

As mentioned, the lack of data points is a disadvantage for the LSTM-Autoencoder. One possible solution could be the replacement of the LSTM Units with gated recurrent units (GRU). Compared to LSTM, GRUs have a simpler architecture. They just have two gates instead of three, were the forget and input gates are coupled to a update gate. Wielgosz et al. [52] compared GRU and LSTM network performances in the context of anomaly detection for superconducting magnet properties, which also have a time dependent component. For the comparison the same parameters like number of layers, number of nodes and number of epochs were chosen. For testing both networks were fed with large (302MB), Medium (184MB) and small (47MB) data set. For the smaller data set the GRU achieved a higher accuracy by 5%. But for the large data set the LSTM surpassed the GRU. Therefore it should be considered to use GRUs for smaller data sets, or collect more data e.g from the winter months as well to further use the LSTM-AE potential. This is further backed up by Bengio et al [5], who also did a comparison between LSTM and GRU. They used music data sets like the Piano midi with 10 test files. This yielded similar results with the GRU reaching a higher F1-score over the LSTM.

Variational LSTM Autoencoder

This is used by Park et al [30] for anomaly detection for time series. In this approach a variational autoencoder is combined with LSTM Unit instead of the vanilla autoencoder for anomaly detection. A reconstruction-based error score to detect anomalies and a state-based threshold is calculated. In addition to the reconstruction error, a variational autoencoder (VAE) can compute the reconstruction log-likelihood of the inputs modeling the underlying probability distribution of data. A variational autoencoder is a generative model. Compared to a normal autoencoder, the latent space of continuous data, allowing easy random sampling and interpolation. This is because the bottleneck layer consists of two parallel layers, which output the mean and the standard deviation. Therefore a distribution is calculated, instead of a fixed vector, which leads to higher AUC in their experimental setup.

Time-Varying Likelihood Threshold

Another possible improvement could be made with the adjustment of the threshold. Park et al. [29] are proposing varying the threshold for anomaly detection. More precisely they are using a detection threshold that changes based on the execution progress. Though it requires likelihood estimates as output from the the model, which in this case is a hidden Markov model. Then the system compares it to a detection threshold, that is based on a probabilistic representation of the execution progress. If at any time the log-likelihood is below the current detection threshold, an anomaly is detected. Therefore the system learns a mapping from execution progress to non-anomalous log-likelihood, which is then used to generate a time-varying likelihood threshold with which it detects anomalies. This approach is able to provide higher true-positive rates at comparable false-positive rates.

Bibliography

[1] Algorithmia. *An introduction to time series forecasting*. 2020. URL: https://algorithmia.com/blog/an-introduction-to-time-series-forecasting (visited on 08/20/2020).

[2] Jason Brownlee. *Understanding LSTM Networks*. 2015. URL: https://machine%20learningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/ (visited on 05/20/2020).

[3] M. Celik, F. Dadaser-Celik, and A. Ş. Dokuz. "Anomaly detection in temperature data using DBSCAN algorithm". In: *2011 International Symposium on Innovations in Intelligent Systems and Applications*. 2011, pp. 91–95.

[4] Raghavendra Chalapathy and Sanjay Chawla. "Deep Learning for Anomaly Detection: A Survey". In: *CoRR* abs/1901.03407 (2019). arXiv: 1901.03407. URL: http://arxiv.org/abs/1901.03407.

[5] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[6] Marc Claesen and Bart De Moor. "Hyperparameter Search in Machine Learning". In: *CoRR* abs/1502.02127 (2015). arXiv: 1502.02127. URL: http://arxiv.org/abs/1502.02127.

[7] Jeff Donahue et al. "Long-term Recurrent Convolutional Networks for Visual Recognition and Description". In: *CoRR* abs/1411.4389 (2014). arXiv: 1411.4389. URL: http://arxiv.org/abs/1411.4389.

[8] G. Dong et al. "A Review of the Autoencoder and Its Variants: A Comparative Perspective from Target Recognition in Synthetic-Aperture Radar Images". In: *IEEE Geoscience and Remote Sensing Magazine* 6.3 (2018), pp. 44–68.

[9] Dimitris Effrosynidis. *Time Series Analysis with Theory, Plots, and Code Part 1*. 2020. URL: https://towardsdatascience.com/time-series-analysis-with-theory-plots-and-code-part-1-dd3ea417d8c4 (visited on 07/10/2020).

[10] Hassan Ismail Fawaz et al. "Deep learning for time series classification: a review". In: *CoRR* abs/1809.04356 (2018). arXiv: 1809.04356. URL: http://arxiv.org/abs/1809.04356.

[11] Felix Gers, Nicol Schraudolph, and Jürgen Schmidhuber. "Learning Precise Timing with LSTM Recurrent Networks". In: *Journal of Machine Learning Research* 3 (Jan. 2002), pp. 115–143. DOI: 10.1162/153244303768966139.

[12] Federico Giannoni, Marco Mancini, and Federico Marinelli. *Anomaly Detection Models for IoT Time Series Data*. 2018. arXiv: 1812.00890.

[13] Klaus Greff et al. "LSTM: A Search Space Odyssey". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (Oct. 2017), pp. 2222–2232. ISSN: 2162-2388. DOI: 10.1109/tnnls.2016.2582924. URL: http://dx.doi.org/10.1109/TNNLS.2016.2582924.

[14] D. M. Hawkins. *Identification of outliers*. Chapman and Hall, 1980. ISBN: 041221900X.

[15] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[16] Thomas Hofer et al. "A robust and accurate surrogate method for monitoring the frequency and duration of combined sewer overflows". In: *Environmental Monitoring and Assessment* 190 (Mar. 2018). DOI: 10.1007/s10661-018-6589-3.

[17] Iberedem Iwok and A. Okpe. "A Comparative Study between Univariate and Multivariate Linear Stationary Time Series Models". In: *American Journal of Mathematics and Statistics* 2016 (July 2016), pp. 203–212. DOI: 10.5923/j.ajms.20160605.02.

[18] Katarzyna Janocha and Wojciech Marian Czarnecki. "On Loss Functions for Deep Neural Networks in Classification". In: *CoRR* abs/1702.05659 (2017). arXiv: 1702.05659. URL: http://arxiv.org/abs/1702.05659.

[19] Nitish Shirish Keskar et al. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: *CoRR* abs/1609.04836 (2016). arXiv: 1609.04836. URL: http://arxiv.org/abs/1609.04836.

[20] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[21] Alexander Lavin and Subutai Ahmad. "Evaluating Real-time Anomaly Detection Algorithms - the Numenta Anomaly Benchmark". In: *CoRR* abs/1510.03336 (2015). arXiv: 1510.03336. URL: http://arxiv.org/abs/1510.03336.

[22] Tae-Hwy Lee. "Loss Functions in Time series forecasting". In: (Apr. 2007).

[23] Jiada Li et al. "Is Clustering Time-Series Water Depth Useful? An Exploratory Study for Flooding Detection in Urban Drainage Systems". In: *Water* 12 (Aug. 2020), p. 2433. DOI: 10.3390/w12092433.

[24] Yulong Lu and Jianfeng Lu. *A Universal Approximation Theorem of Deep Neural Networks for Expressing Distributions*. 2020. arXiv: 2004.08867 [cs.LG].

[25] G. Mahalakshmi, S. Sridevi, and S. Rajaram. "A survey on forecasting of time series data". In: *2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16)*. 2016, pp. 1–8.

[26] Nijat Mehdiyev et al. "Time Series Classification using Deep Learning for Process Planning: A Case from the Process Industry". In: *Procedia Computer Science* 114 (2017). Complex Adaptive Systems Conference with Theme: Engineering Cyber Physical Systems, CAS October 30 – November 1, 2017, Chicago, Illinois, USA, pp. 242–249. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2017.09.066. URL: http://www.sciencedirect.com/science/article/pii/S1877050917318707.

[27] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 03/16/2020).

[28] Maximilian Panzner and Philipp Cimiano. "Comparing Hidden Markov Models and Long Short Term Memory Neural Networks for Learning Action Representations". In: Aug. 2016, pp. 94–105. ISBN: 978-3-319-51468-0. DOI: 10.1007/978-3-319-51469-7_8.

[29] D. Park et al. "Multimodal execution monitoring for anomaly detection during robot manipulation". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 407–414.

[30] Daehyung Park, Yuuna Hoshi, and Charles C. Kemp. "A Multimodal Anomaly Detector for Robot-Assisted Feeding Using an LSTM-based Variational Autoencoder". In: *CoRR* abs/1711.00614 (2017). arXiv: 1711.00614. URL: http://arxiv.org/abs/1711.00614.

[31] Razvan Pascanu et al. "How to Construct Deep Recurrent Neural Networks". In: (Dec. 2013).

[32] O. I. Provotar, Y. M. Linder, and M. M. Veres. "Unsupervised Anomaly Detection in Time Series Using LSTM-Based Autoencoders". In: *2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT)*. 2019, pp. 513–517.

[33] Joseba Quevedo et al. "Temporal/Spatial Model-Based Fault Diagnosis vs. Hidden Markov Models Change Detection Method: Application to the Barcelona Water Network". In: June 2013. DOI: 10.1109/MED.2013.6608752.

[34] Chitta Ranjan. *LSTM Autoencoder for Extreme Rare Event Classification in Keras*. 2019. URL: https://towardsdatascience.com/lstm-autoencoder-for-extreme-rare-event-classification-in-keras-ce209a224cfb (visited on 04/08/2020).

[35] Chitta Ranjan. *Step-by-step understanding LSTM Autoencoder layers*. 2019. URL: https://towardsdatascience.com/step-by-step-understanding-lstm-autoencoder-layers-ffab055b6352 (visited on 04/04/2020).

[36] R. K. Agrawal Ratnadip Adhikari. *An Introductory Study on Time Series Modeling and Forecasting*. LAP LAMBERT Academic Publishing, Jan. 2013. ISBN: 978-3659335082.

[37] Antônio H. Ribeiro et al. "The trade-off between long-term memory and smoothness for recurrent networks". In: *CoRR* abs/1906.08482 (2019). arXiv: 1906.08482. URL: http://arxiv.org/abs/1906.08482.

[38] Salah Rifai et al. "Contracting auto-encoders: Explicit invariance during feature extraction". In: *In Proceedings of the Twenty-eight International Conference on Machine Learning (ICML'11*. 2011.

[39] Pascale Rouault. *Monitoring of Water Quality Parameters in Combined Sewer Overflows*. Tech. rep. Berlin, Germany: Kompetenzzentrum Wasser Berlin, Jan. 2009.

[40] S. Shakya and S. Sigdel. "An approach to develop a hybrid algorithm based on support vector machine and Naive Bayes for anomaly detection". In: *2017 International Conference on Computing, Communication and Automation (ICCCA)*. May 2017, pp. 323–327. DOI: 10.1109/CCAA.2017.8229836.

[41] Alex Sherstinsky. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network". In: *Physica D: Nonlinear Phenomena* 404 (Mar. 2020), p. 132306. ISSN: 0167-2789. DOI: 10.1016/j.physd.2019.132306. URL: http://dx.doi.org/10.1016/j.physd.2019.132306.

[42] G. Shmueli and K.C. Lichtendahl. *Practical Time Series Forecasting with R: A Hands-On Guide [2nd Edition]*. Practical Analytics. Axelrod Schnall Publishers, 2016. ISBN: 9780997847918. URL: https://books.google.de/%20books?id=SOtgvgAACAAJ.

[43] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. *A Comparative Analysis of Forecasting Financial Time Series Using ARIMA, LSTM, and BiLSTM*. 2019. arXiv: 1911.09512 [cs.LG].

[44] Keras Special Interest Group (Keras SIG). *Keras FAQ*. 2018. URL: https://keras.io/getting_started/faq/#why-is-my-training-loss-much-higher-than-my-testing-loss (visited on 08/27/2020).

[45] Keras Special Interest Group (Keras SIG). *Layers API*. 2018. URL: https://keras.io/api/layers/core_layers/dense/ (visited on 01/22/2021).

[46] Matthew Stewart. *Comprehensive Introduction to Autoencoders*. 2020. URL: https://towardsdatascience.com/%20generating-images-with-autoencoders-77fd3a8dd368 (visited on 05/11/2020).

[47] Ayesha Sulthana, S. Balasubramanian, and K. C. Latha. "A Time Series Analysis of Wastewater Inflow of Sewage Treatment Plant in Mysore, India". In: *International Journal of Current Research* 5 (Feb. 2013), pp. 248–253.

[48] Meir Toledano et al. "Real-time anomaly detection system for time series at scale". In: *Proceedings of the KDD 2017: Workshop on Anomaly Detection in Finance*. Ed. by Archana Anandakrishnan et al. Vol. 71. Proceedings of Machine Learning Research. PMLR, 14 Aug 2018, pp. 56–65.

[49] Neelam Tyagi. *Introduction to Time Series Analysis: Time-Series Forecasting Machine learning Methods & Models*. 2020. URL: https://medium.com/analytics-steps/introduction-to-time-series-analysis-time-series-forecasting-machine-learning-methods-models-ecaa76a7b0e3 (visited on 08/15/2020).

[50] Rohit Varma. *Why we need Multi Layer Neural Network (MLP)?* 2020. URL: https://medium.com/@Rohit_Varma/why-we-need-multi-layer-neural-network-mlp-d50425b8f37d (visited on 08/21/2020).

[51] Jingyu Wang. "Combined Sewer Overflows (CSOs) Impact on Water Quality and Environmental Ecosystem in the Harlem River". In: *Journal of Environmental Protection* 05 (Jan. 2014), pp. 1373–1389. DOI: 10.4236/jep.2014.513131.

[52] Maciej Wielgosz, Andrzej Skoczen, and Matej Mertik. "Recurrent Neural Networks for anomaly detection in the Post-Mortem time series of LHC superconducting magnets". In: (Feb. 2017).

[53] G. Yan. "Network Anomaly Traffic Detection Method Based on Support Vector Machine". In: *2016 International Conference on Smart City and Systems Engineering (ICSCSE)*. 2016, pp. 3–6.

[54] Qin Yu, Lyu Jibin, and Lirui Jiang. "An Improved ARIMA-Based Traffic Anomaly Detection Algorithm for Wireless Sensor Networks". In: *International Journal of Distributed Sensor Networks* 2016 (Jan. 2016), pp. 1–9. DOI: 10.1155/2016/9653230.

[55] Duo Zhang, Geir Lindholm, and Harsha Ratnaweera. *DeepCSO: Forecasting of Combined Sewer Overflow at a Citywide Level using Multi-task Deep Learning*. Oct. 2018. DOI: 10.13140/RG.2.2.21058.84161.