Evaluation of Machine Learning-based Methods for Continuous Water Quality Data Analysis

EVALUATION OF MACHINE LEARNING-BASED METHODS FOR CONTINUOUS WATER QUALITY DATA ANALYSIS

BY

XI WANG, B.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING & SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

Master of Science (2019) McMaster University
Computing & Software Hamilton, Ontario, Canada

TITLE:	Evaluation of Machine Learning-based Methods for Continuous Water Quality Data Analysis
AUTHOR:	Xi Wang B.Sc.
SUPERVISOR:	Dr. Emil Sekerinski Dr. John Copp
NUMBER OF PAGES:	x, 95

Abstract

Wastewater treatment facilities are increasingly installing sensors to monitor water quality. As these datasets have increased in size and complexity, it has become difficult to identify abnormal readings in a timely manner either manually or using simple rules that might have been sufficient previously. Two ammonia sensors were installed at the Dundas Wastewater Treatment Plant in November 2017. The collected ammonia concentration data shows a daily pattern. A learning-based method is implemented in this thesis to identify any readings which violate this daily pattern. The data points which were predicted to be anomalous were qualitatively ranked based on the severity and the likelihood of being faulty. The result of the learning-based method was evaluated and compared to other traditional detection methods.

Acknowledgements

I would like to thank my supervisors Dr. John Copp and Dr. Emil Sekerinski. It is essential to have their guidance during times of struggle in completing this thesis. Their doors were always open every time I had a question about the research or writing. Besides my supervisors, I would like to thank the rest of my thesis committee: Dr. Martin v. Mohrenschildt and Dr. Hassan Ashtiani for their encouragement, insightful comments, and hard questions. Dr. Martin v. Mohrenschildt generously offers his time, support and guidance throughout the preparation and review of this document.

Besides, I would also like to extend my gratitude to the entire research team at Dr. Emil Sekerinski's lab for welcoming me to the team, and their hard work in data collecting and sensor maintenance during the project. I would like to show our gratitude to Global Water Futures and Southern Ontario Water Consortium for their support for this research. Special thanks to my families and friends for their continuous motivation and encouragement.

Contents

List of Tables

List of Figures

Abbreviations and Symbols

ANN	Artificial Neural Networks
BPTT	Back-Propagation Through Time
CNN	Convolutional Neural Networks
EMA	Exponential Moving Average
ESD	Extreme Studentized Deviate
GRU	Gated Recurrent Units
KNN	K-Nearest Neighbors
LSTM	Long Short Term Memory
MSE	Mean Squared Error
PCA	Principal Component Analysis
RNN	Recurrent Neural Networks
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
WWTP	Wastewater Treatment Plant

Declaration of Academic Achievement

Under the supervision of my supervisors Dr. John Copp and Dr. Emil Sekerinski, I was responsible for the data analysis and interpretation, as well as the report writing. Dr.Sekerinski provided guidance on the theoretical development of this project. Dr.Copp provided advice on the statistical methodology and engineering background of water quality. Prabhbir Pooni and Winfield Lai, both members of the Water Quality Research Laboratory, were responsible for the sensor maintenance and data collection.

Chapter 1

Introduction

Ammonia is the natural product of the decay of organic nitrogen compounds. Ammonia in wastewater is a concern due to its detrimental impact on aquatic ecosystems. In wastewater, most ammonia comes from human urine or industrial discharges such as fertilizers. Ammonia can contribute to algal blooms when discharged to bodies of water. Excess algae deplete dissolved oxygen and release toxins which can lead to the destruction of aquatic life [Liang et al., 2000]. As such, many wastewater treatment plants (WWTPs) are required to limit the discharge of ammonia [Canada, 2013]. Controlling ammonia discharges from wastewater treatment plants can significantly reduce the load on the receiving water body. Modern wastewater treatment plants are usually designed with biological systems to treat ammonia. In the process of so-called *biological nitrification*, ammonia gets converted to nitrate using aerobic autotrophic bacteria [Ruiz et al., 2003]. Ammonia sensors are installed inside the WWTPs [Harrou et al., 2018] to monitor the ammonia concentration. The collected ammonia data can be used for the plant operation and optimization. The monitored faulty sensor readings need to be detected as soon as possible, so that operators can

be alarmed in time and start to diagnose factors which prevents WWTP from normal operation.

1.1 Problem

The term *anomaly* is used to describe patterns or instances which differ from the expected or normal behavior. Anomalies are also referred to as *outliers* or *novelties*. The term *anomaly detection* refers to the process of identifying anomalies. The detection is non-trivial as the ammonia concentration data is usually variable and noisy. Simple thresholds such as an upper or bottom threshold, where a violation of a threshold triggers an alarm, can be applied to detect anomalies. This simple static rule works well for situations such as an emergency when the concentration reaches extreme values. For example, the sensor stops working and keeps collecting zero readings. However, the normal range of ammonia concentration is not static. For the concentration in Figure 1.1, in winter time, an upper threshold can be set at 20mg/L as no reading seems to be above this value. However, using the same threshold is likely to trigger a lot of false alarms in the summertime, as normal concentration can be above 25mg/L. A threshold around 28mg/L would be better in this case. As a result, a single static threshold does not work for different months.

There are repeating patterns in the ammonia concentration data. A set of rules can be manually designed to describe the underlying structure of the ammonia concentration. It is possible to detect anomalies using such hand-crafted rules by using professional software such as *DataDesk* developed by Primodal. DataDesk allows users to use different pre-loaded statistical rules or customized rules to capture the features of the dataset. Readings which do not follow these rules can be identified as

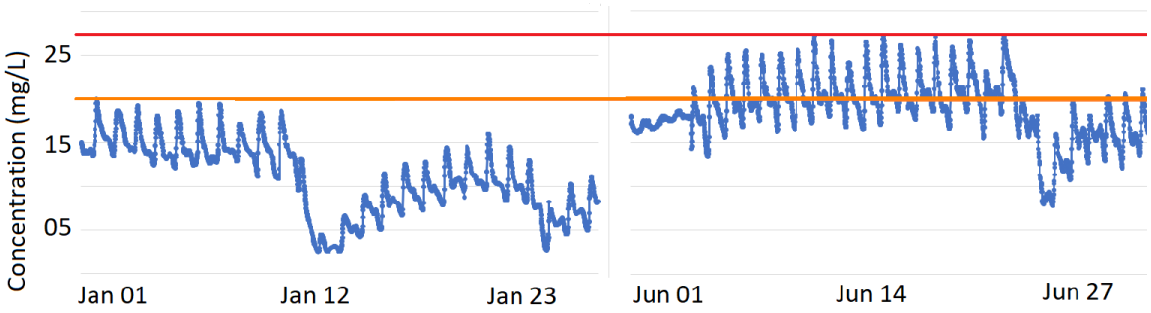

Figure 1.1: Potential Static Threshold

anomalies. However, the customization requires extensive knowledge and effort by a domain expert. In addition, the manual designed rules are usually dataset-specific, which means the rules that work for one may not work for another one.

1.2 Methodology and Goal

The data-driven approach is a preferable option as all the rules are derived from the dataset directly without much prior knowledge. An *LSTM (Long Short-Term Memory)* network was chosen as the data-driven approach due to its success in recent time series problems in different applications [Graves et al., 2013]. LSTM can learn high-level representations of datasets automatically with little or no need for manual feature engineering and domain knowledge. The repeating patterns are learned and stored as a predictive model. Prediction of future values is calculated based on its previous values and the predictive model. The anomalies can be detected and ranked based on the difference between the predicted values and the actual values. A good anomaly detection approach, or an *anomaly detector*, should detect most anomalies based on the concentration dataset only.

As the ammonia concentration is heavily affected by precipitation, some of the

detected anomalies are the result of rain events. The other anomalies, or the real faulty data caused by operations are of more interest. As the precipitation and flow data are available in this project, an attempt was made using the weather datasets to distinguish the real faulty data from the precipitation-caused anomalies.

1.3 Contribution

Different methods have been proposed to detect water quality anomalies in the past. Some approaches used statistical modelling techniques [Venkatasubramanian et al., 2003], while others used conventional data-driven techniques such as *PCA (principal component analysis)* [Sanchez-Fernandez et al., 2015]. However, not much previous work was found using LSTM to solve this specific problem for WWTPs. LSTM has been widely used in similar applications when dealing with time series datasets [Kwon et al., 2017] from totally different sources, such as the medical field or manufacturing. These datasets share some common characteristics with the ammonia dataset, i.e., they are all time series with repeating patterns. As LSTM has proven to be successful in many different fields, it can be reasonably deduced that LSTM might also be a suitable choice for the ammonia dataset.

This thesis project has three contributions. First, the architecture of neural networks is reviewed to explain the reasons why LSTM is suitable for time series and how a learning based method extracts information from sequential data. Secondly, an LSTM-based method was implemented to detect ammonia anomalies. Thirdly, the relationships between ammonia data and weather data were explored to better analyze the detection results.

1.4 Outline

This thesis is arranged as follows. Chapter 2 gives a systematic overview of related work and discusses their suitability for this project. The scope illustrates why deep learning is a suitable approach for this particular problem and introduces the fundamental architectures. Chapter 3 discusses the setup of the experiments and the datasets. Chapter 4 presents the results of different experiments and evaluation of performance. Chapter 5 concludes the thesis and gives a perspective for future work that could be conducted based on the results. The code for this project is available in a Gitlab directory [1].

[1] https://gitlab.cas.mcmaster.ca/waterquality/reports/tree/master/Wang19AutomatedDetection/Code

Chapter 2

Related Work and Background

More and more high-frequency sensors are being installed in WWTPs. Prior to the adoption of monitoring sensors [Rabiet et al., 2009], in order to find the anomalies of a WWTP, water samples needed to be manually collected from rivers and monitoring wells. The water samples are typically stored in bottles and sent back to be analyzed in an laboratory by someone with sufficient domain knowledge. This process is slow and expensive. Now with sensors in WWTPs, water quality quantities like ammonia concentration can be monitored continuously. The collected data can still be analyzed by experts using tools like DataDesk as in the past; however, the increasing volume of the stored data has made it difficult to assess the data quality in a timely manner. Anomalies may occur when the system is not operating properly. In such cases, the problem needs to be alarmed as soon as possible to avoid collecting faulty data. As it is unrealistic for a human engineer to monitor data 24/7, it thus becomes more advantageous to automate this process by employing algorithms.

2.1 Rule-based Approaches

In this project, the collected ammonia data is a variable time series. Anomaly detection in time series data is a difficult problem due to the complexity and variable nature it has [Chandola et al., 2009]. The most direct approach is to identify anomalies based on differences from normal data.

One possible statistical feature of ammonia anomalies is their derivatives, which measure the change in ammonia concentration over time. The change in concentration in a normal day is usually slow and smooth. Readings with a high derivative, indicating a rapid rate of change, are more likely to be an anomaly caused by an unusual events. The rate of change for all data samples can be calculated, and a threshold for this can be set [Rajasegarar et al., 2008] so that any points above that threshold are identified as anomalies.

Another option is to make use of the seasonality or repeating daily pattern of the ammonia data, with the *moving average (MA)* approach. A series of subsets is generated by moving lagging windows across the full dataset [Nakano et al., 2017]. The moving averages are calculated to reveal the average values over the given period. Ammonia anomalies are identified from the calculated moving average and the actual values difference. In a *simple moving average (SMA)*, every data sample in the window has the same weight on the resulted average. In the equation, x_t is the value at time t, and n is the size of the moving window.

$$SMA = \frac{x_t + x_{t-1} + ... + x_{t-(n-1)}}{n} \tag{2.1}$$

However, in the ammonia dataset the recent data usually has more influence or

importance than older data. Thus, the *exponential moving average (EMA)* was used instead, where greater weight is given to more recent values [Hansun, 2013]. In EMA the most recent data sample gets the most significant weight, and the weights on other data values decrease as they progress away from the current time. In Equation 2.2 α represents a constant smoothing factor between zero and one. As a result, EMA can respond more quickly to value changes than the SMA does.

$$EMA = \begin{cases} y_t = x_t, & t = 1 \\ y_t = \alpha(x_t) + (1-\alpha)y_{t-1}, & t > 1 \end{cases} \tag{2.2}$$

The unusual or rare ammonia concentrations are another indicator for anomalies. In statistics, the *standard deviation* σ is the measure for the amount of variation or dispersion. In a dataset with a low standard deviation, the data samples tend to be close to the mean value. If a dataset is approximately normally distributed, 99.7 percent of the data samples will fall within three standard deviations of the mean as shown in Figure 2.2. In this scenario, the remaining data are identified as anomalies because it is assumed that anomalies are usually rare events in a well-operated sensor.

The *Extreme Studentized Deviate (ESD)* test [Rosner, 1983] is an algorithm which detects multiple anomalies assuming data is normally distributed. By estimating the percentage of anomalies within the total dataset, ESD can detect the most extreme values as anomalies. However, the general ESD cannot be applied directly to the concentration dataset for two reasons. First, the concentration dataset is not normally distributed because of the repeating pattern. Second, the general ESD does not work well for datasets containing a high percentage of anomalies. ESD uses mean and standard deviation to model the data, but these two metrics are easily distorted

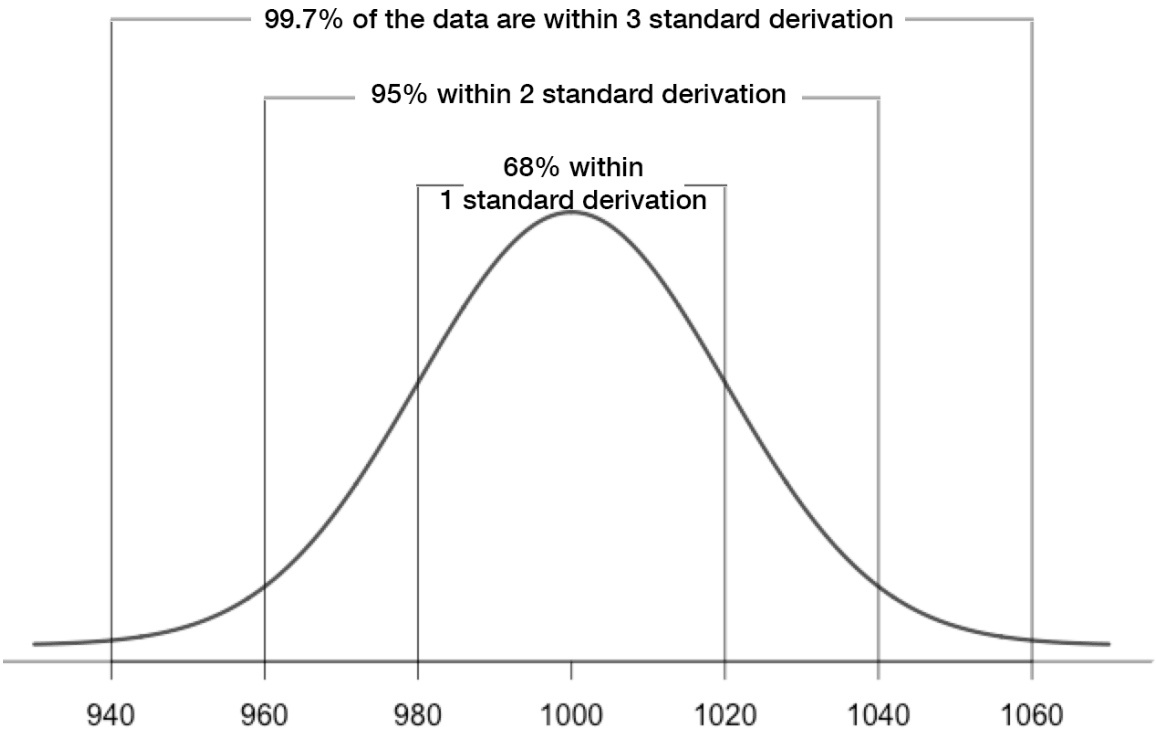

Figure 2.2: Three-Sigma Rule

by anomalies. For example, a single high value can inflate both of these measures significantly. Thus, to address these problems in other applications researchers from Twitter built the *Seasonal Hybrid ESD (S-H-ESD)* [Vallis et al., 2014].

In a time series, a trend component, T can be derived that it refers to the underlying movement, while a seasonal component S refers to the repeating movement as shown in Figure 2.3. S-H-ESD can decompose the residual component R from the trend and seasonal components using the decomposition technique called *STL* [Cleveland et al., 1990].

$$R_X = X - T_X - S_X \tag{2.3}$$

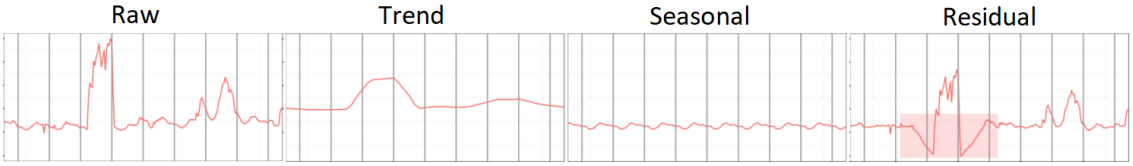

Figure 2.3: Example of Seasonal Decomposition by STL [Vallis et al., 2014]

The residual thus has a unimodal distribution that is suitable for ESD. A modified ESD can also be used, where mean and standard deviation are replaced with *median* and *MAD (Median Absolute Deviation)*. MAD refers to the median of the absolute deviations from the sample median. Both median and MAD are more robust measures. Unlike the mean value which can be distorted by a single anomaly, the median value may tolerate up to 50 percent of the data being anomalous. The S-H-ESD should thus be able to detect anomalies even if some spikes exist.

The three approaches discussed above, i.e., derivative, EMA and S-H-ESD are based on statistical features. As these features are easy to adopt, they were used as the baseline or benchmark in the experiments.

More complex statistical methods can be introduced to handle more complex datasets [Venkatasubramanian et al., 2003]. In practice, an expert can also make use of professional tools like DataDesk to design more advanced rules to identify anomalies. However, rule-based anomaly detection approaches usually struggle to identify real faults from normal process variations as it is often difficult to develop an accurate model that describes all physical or chemical factors in the process. Even though the statistical analysis can be custom-made for one dataset, the same set of rules usually do not apply directly to other WWTPs. It is expensive to craft a set of rules for every individual WWTPs. A dataset-specific approach indeed needs to be designed for the best possible features of each dataset. Instead of defining rules

manually, a data-driven approach, where the process is modelled without the explicit expression of the process, would appear to be a better option [Qin, 2009].

2.2 Machine Learning

Machine learning can derive high-level representations of data with little manual feature engineering or domain expertise. Both *supervised learning* and *unsupervised learning* are widely used for anomaly detection tasks [Patcha and Park, 2007]. In supervised learning, a classifier is trained to detect anomalies with training data being labeled as normal and abnormal. However, in practice sensor anomalies are usually rare events caused by critical failures and can be may not be present in the training data. If there are not enough representative abnormal data to be learned, the classifier may not work properly. Furthermore, labeling a large dataset can be a time-consuming task, and incorrect labels may result in higher false alarm rates. Unlike supervised learning, unsupervised learning uses unlabelled training data, as long as most of data is normal. Anomalies are detected in a regression setting. The predicted or expected values are estimated based on the previous values and the normal pattern, where the normal pattern is "learned" from normal data. The prediction error is calculated as the offset between the predicted and actual recorded values. A threshold for prediction error can be set to distinguish anomalies. As the ammonia dataset lacks labels and most data is normal, only the unsupervised machine learning is used as the data-driven approach in this project.

PCA (principal component analysis) and its extensions, where the original data gets projected onto a feature space, have been widely used for modeling and monitoring of WWTPs [Liu et al., 2014]. A PCA model is built to describe the correlations

between the features. A new data sample is projected using the model, which results in a reconstruction error. Anomalies are detected based on a high normalized error method [Sanchez-Fernandez et al., 2015]. In a similar application, water quality anomalies were identified in conductivity, pH, temperature and turbidity sensor data collected from a river and a WWTP [Alferes et al., 2013]. After the PCA model was trained with normal data, faults or abnormal conditions were detected by violations of confidence limits. There are some limitations for PCA as an anomaly detection approach. For example, PCA is highly sensitive to data perturbations. A few extreme data samples in the training dataset can heavily change the orientation of projection [Chalapathy et al., 2017]. This limit may be resolved by using more complex PCA variant such as robust PCA. However, once a PCA model is built for the training dataset, it explicitly describes the features of the training set. Using the same model in a different condition or time period can cause false alarms [Haimi et al., 2016]. It takes efforts to understand the new dataset and retrain the PCA model.

2.3 Deep Learning

Deep learning, as one method of the machine learning algorithms, has shown robust capabilities in a variety of tasks in recent years. Deep learning works well with raw data, and it has given satisfactory results from computer vision to natural language processing [Hinton et al., 2012]. Deep learning is also widely used in the field of anomaly detection and a survey of deep learning-based anomaly detection has been presented [Kwon et al., 2017]. The learning-based method was tested on different benchmark datasets and the result showed an improved accuracy in detecting

anomalies compared to conventional machine learning techniques. Compared to machine learning algorithms, deep learning can be viewed as a "black box" approach to a certain degree. Although the mathematics used to construct a neural network is clear and straightforward, the trained deep learning model is more implicit about how the output was arrived at. It thus makes it easier to reuse the model. For example, after a deep learning model is built for one WWTP, the model can be reused for a similar application in a new location. The learned pattern from a previous WWTP can help when there is not enough collected data in the new WWTP. Instead of designing or training a new model from scratch, training the model can be continued with the new data directly.

The concept of deep learning dates back to the 1970s [Ivakhnenko, 1971], and many of the critical algorithmic breakthroughs occurred in the 1980s and 1990s. The recent success of deep learning techniques has been facilitated by two factors: vast computational power and the availability of massive datasets as shown in Figure 2.4. Moreover, distributed hardware systems including GPUs have allowed deep learning architectures to be trained adequately. Enormous datasets including images, video, audio, and text are being collected across the internet [Chen and Lin, 2014], which allows neural network models to be appropriately fitted with millions of parameters. The architecture of neural networks is flexible such that they can be designed for specific tasks for different applications.

2.4 Artificial Neural Network

It is necessary to understand how deep learning algorithms work before determining the proper approach for the ammonia dataset. The basic structure of deep learning

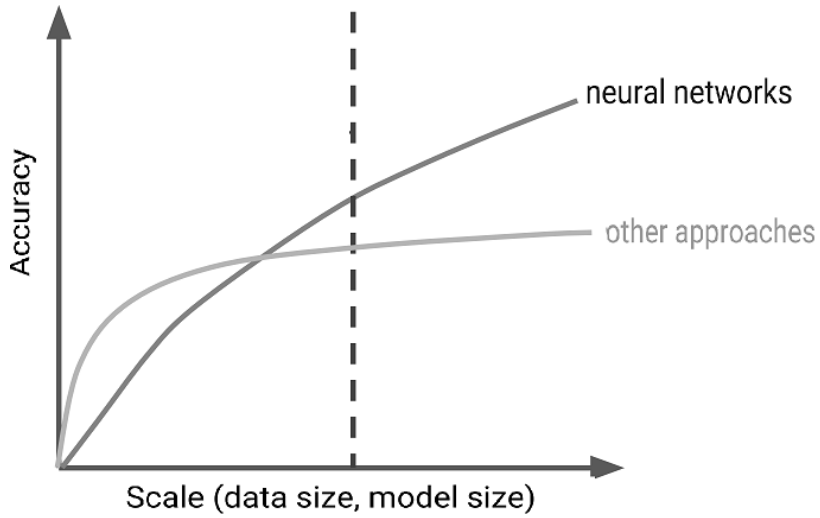

Figure 2.4: The Recent Success of Deep Learning [Patidar, 2018]

algorithms are *artificial neural networks (ANNs)*, which are inspired by the structure and function of the biological neural networks in the brain. In this thesis, all the neural networks are referred to as artificial neural networks.

In general, a neural network consists of an input layer, one or multiple fully connected hidden layers, and an output layer as shown in Figure 2.5. The *input layer* is the first layer which receives the input data. The input layer only forwards the input data without any processing. The *output layer* is responsible for processing the output. The layers in-between are referred to as *hidden layers*. Each layer contains multiple computational units, which are also called *nodes* or *neurons*. The sophisticated features of the dataset are learned and stored within the hidden layers. Hidden layers can be stacked, working as a pipeline where each layer does part of the task.

A neuron receives input vectors from neurons in the previous layer along the directed edges. Every edge contains a corresponding weight associated with it. The input data is then multiplied by the *weight* and subsequently added to the *bias*. The sum serves as input for the *activation function* or *transfer function*. The output is

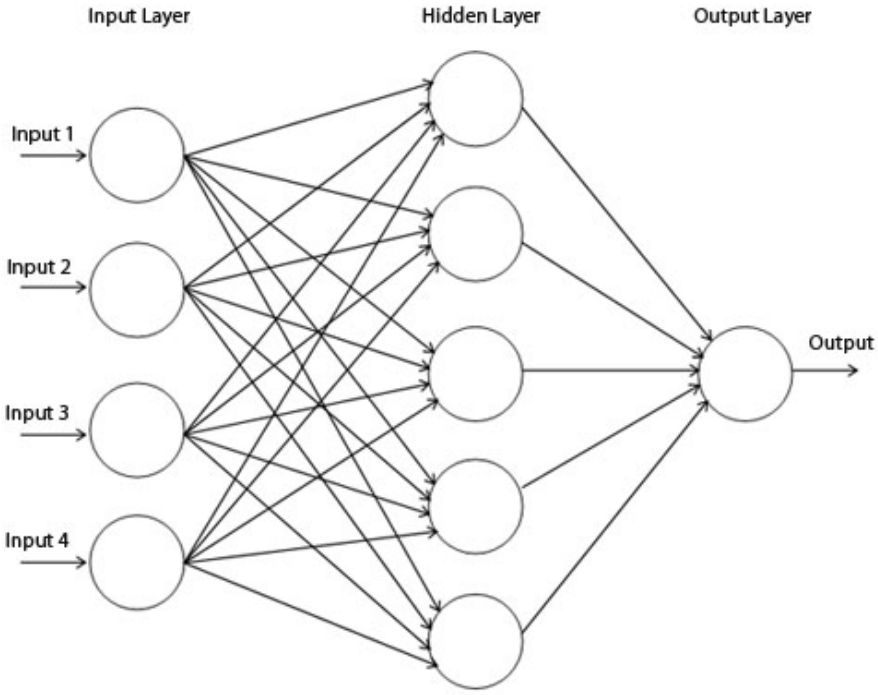

Figure 2.5: Neural Networks with One Hidden Layer

passed to neurons in the next layer. A single neuron can be represented mathematically by the following vector equation, where x represents the input, o represents the output, g represents the activation function, ω represents the weight, and b represents the bias.

$$o(x) = g(\omega \odot x + b) \tag{2.4}$$

The activation function can be linear, which is used for regression problems where continuous values are predicted. The output of a linear activation function is not constrained to any range. There are also some non-linear activation functions. The *Sigmoid*, *Tanh*, and *Rectified Linear Units (ReLU)* [Nair and Hinton, 2010] are three

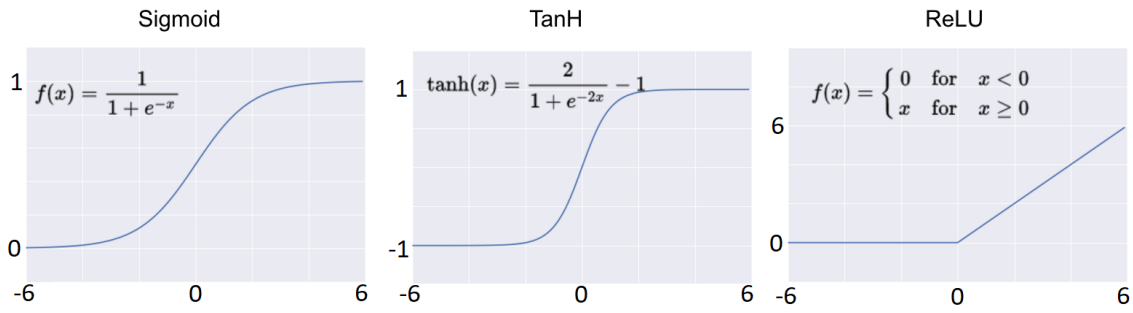

Figure 2.6: Commonly Used Activation Functions

commonly used non-linear activation functions as shown in Figure 2.6. The Sigmoid function is usually used for probabilities, as its output is restricted to between 0 and 1. The Tanh function can be used for similar situations with its output ranging from -1 to 1. ReLU is half rectified from the bottom. The output is zero when y is less than zero and $f(y)$ is equal to y when y is above or equal to zero.

2.5 Training Neural Networks

The normal repeating pattern of ammonia is learned by a neural network by the process called *training*. In the terminology of deep learning, the learned information is stored as *parameters*, i.e., weights and biases, while *hyperparameters* refer to all the variables which determine the network structure or how the network is trained. Hyperparameters are set before training starts, while the parameters are optimized during the training.

The difference between the predicted values and the actual values can be indicated by a *loss function*. A loss function refers to the performance of a predictive model on predicting the expected output. The loss function is also called the *cost function* sometimes as it represents some "cost" associated with the prediction. One commonly

used loss function is the *mean squared error (MSE)*, which measures the averaged squared distance between the predicted values and actual values.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.5}$$

The loss function depends on the learnable parameters of a neural network. Information about the repeating pattern in the ammonia dataset is stored as these learned parameters. The goal of training a neural network is to minimize the loss function by optimizing these parameters. The gradient is the measure of the change in the loss function corresponding to changes in the parameters. The algorithm used to calculate the gradients is called *backpropagation*. The "back" part of the name comes from the fact that the calculation of the gradient propagates backward through the neural networks. The gradient of the final layer of weights get calculated first while the gradient of the first layer of weights get calculated last. According to the chain rule of derivatives, the partial gradient computations from one layer get reused in the computation of the gradient in the next layer. The efficient computation of the gradient at each layer thus result from the backward flow of the error information.

The calculated derivatives are then used by an optimization algorithm, *gradient descent*, to adjust the weights up or down, depending on the direction that minimizes the loss function. The optimization is an iterative process, where the training data needs to be passed multiple times before it reaches the optimal result. The loss function should become smaller after each iteration. There are several variants of gradient descent algorithms. The first one, *batch gradient descent* computes the gradient of the loss function with the entire training dataset. It is slow and computationally unrealistic for datasets that do not fit in memory [Wilson and Martinez, 2003]. In contrast,

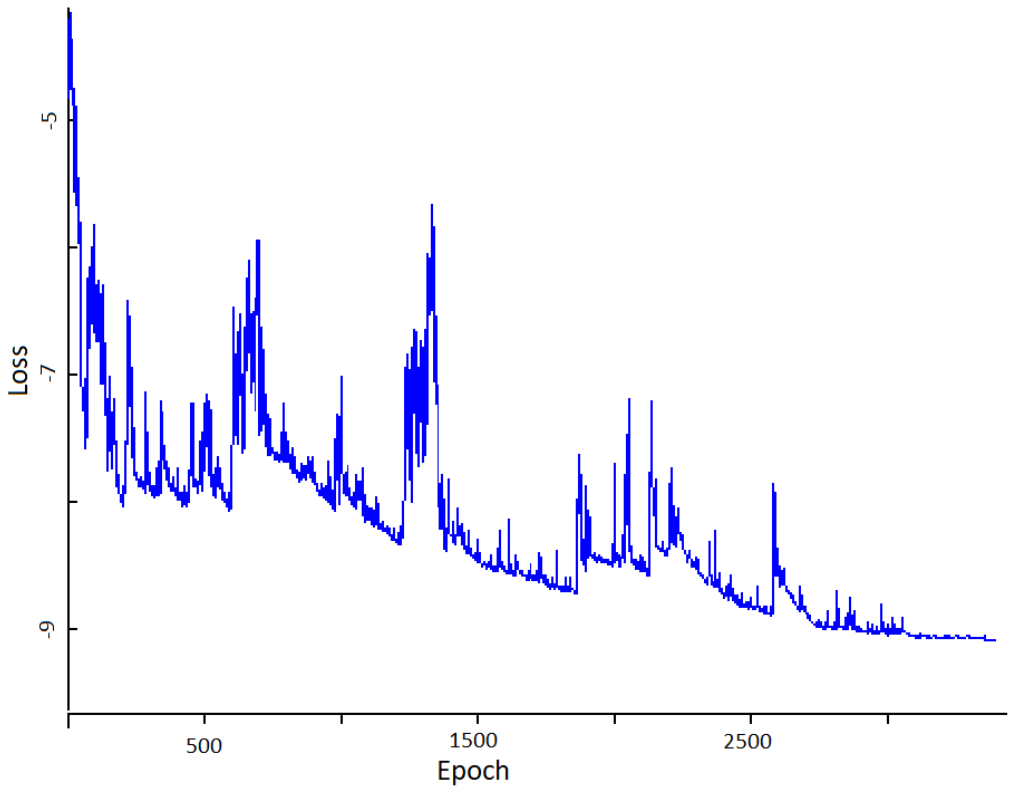

Figure 2.7: Fluctuations in SGD

a second algorithm, *stochastic gradient descent (SGD)*, performs a parameter update with every single training example and is usually much faster. The problem is that frequent updates can result in a noisy gradient signal as shown in Figure 2.7, which often makes it difficult for the parameters to settle on the optimal.

In practice, a third variant, *mini-batch gradient descent*, is more often employed. The mini-batch gradient descent is in between the two previous approaches. The training dataset is split into multiple subsets, each of which is called a *batch* and contains multiple training samples. Mini-batch gradient descent calculates the error and updates the parameters over a batch, so it takes the best features of the previous two gradient descent algorithms. The batch size needs to be set properly for the

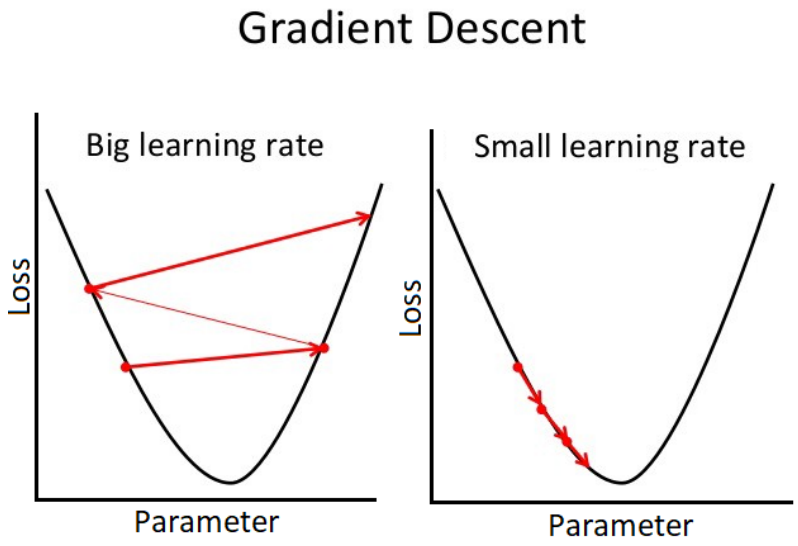

Figure 2.8: High Learning Rate vs. Low Learning Rate

dataset, so that the model can be trained efficiently. In mini-batch, it is possible to set the size of the steps when moving the parameters in the opposite direction of the gradient. The scalar value for the step size is called the *learning rate* [Bottou, 2012]. With a low learning rate, the training is more reliable as the parameters are being recalculated so frequently. However, calculating the gradient is time-consuming, so it will take a long time to reach the optimal. With a high learning rate, more ground is covered in each step, but the parameter changes are so significant that the optimizer may overshoot the minimum, the training may not converge or can even diverge [Donges, 2018] as shown in Figure 2.8. Thus, it is sometimes preferable to use a dynamic decay such that the learning rate can gradually decrease as parameters approach the optimum values.

Some popular mini-batch gradient descent optimization algorithms are *AdaGrad*, *RMSprop*, and *Adam*. AdaGrad adapts the learning rate to the parameters. It

uses low learning rates for parameters associated with frequently occurring features, and high learning rates for parameters associated with infrequent features [Duchi et al., 2011]. RMSprop is an unpublished algorithm proposed by Geoff Hinton in his Coursera lecture. The approach resolves AdaGrad's radically diminishing learning rates problem [Tieleman and Hinton, 2012] and uses a decay of the accumulated historical gradients. Adam stands for Adaptive Moment Estimation. Compared to RMSprop, Adam uses an exponentially decaying average of past gradients.

One pass over all the training datasets is referred to as an *epoch*. After every epoch, the parameters, i.e., weights and biases, should get closer to their optimum values which minimizes the loss function. *Underfitting* occurs when the algorithm does not fit the data well enough. In contrast, *overfitting* occurs where the model fits the training dataset well but cannot predict data that has not been seen during training. The reason for that is the network has memorized the training examples including the noise. Overfitting is a serious problem, as the learned model in this case should only remember the daily repeating pattern. One way to prevent overfitting is to use *"early-stopping"* [Prechelt, 1998]. The dataset is split into a training and a validation set. The values of the loss function when applied to the training set and validation set are compared after every iteration. New iterations can improve both the training and validation set initially. After several iterations the loss on the validation set starts to increase while the loss for the training set is still decreasing. The training process should stop at that point as shown in Figure 2.9.

Another commonly used technique to avoid overfitting is *dropout* [Srivastava et al., 2014]. As overfitting often occurs when the model is more complicated than required, it is efficient to simplify the model. During each training step, the output of some

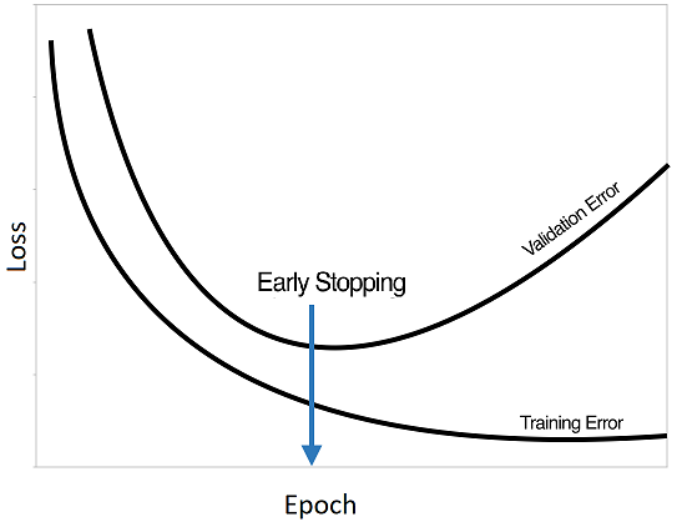

Figure 2.9: Early Stopping

random connections in the network gets removed by multiplying with 0. As a result, the training set in each epoch will be slightly different. This prevents the network from memorizing the exact training set. Both data splitting and dropout techniques were applied in the experiments to avoid the overfitting problem.

Conventional neural networks are limited when dealing with time series datasets. In conventional neural networks, each neuron in any hidden layers is fully connected to all neurons in the previous layer. Each neuron is also entirely independent and does not share any connections. The number of weights grows rapidly when the input size of the training data increases. It is thus slow and inefficient to train the ammonia concentration dataset with a conventional neural network. This problem can be addressed by using a convolutional architecture where all the training samples are padded to a fixed sized window. The window slides over the whole dataset and learns the features inside each window [Krizhevsky et al., 2012].

2.6 Convolutional Neural Network (CNN)

Convolutional neural networks (CNNs) are another neural network architecture and were initially designed for two-dimensional data i.e., images. CNNs model the cells in the human visual cortex and have been widely used in computer vision tasks [Lecun et al., 1998]. CNNs also have been adapted for one-dimensional data such as time series in recent years. There have been some attempts to use CNNs for time series dataset from other fields. Research has been conducted using 1D convolutional neural networks to detect anomalies in *electrocardiography (ECG)* signals [Kiranyaz et al., 2016]. An ECG dataset records the electrical activity of the heart muscle as it changes with time. Even though the ECG data comes from a totally different field, these data still share some common characteristics of time series data. A normal ECG also contains a repeating pattern caused by a heartbeat. The normal ECG signal is fed directly into a CNN to build the predictive model without any prior information. And the results exhibited excellent performance regarding both accuracy and speed. CNNs have also been applied in other time series domains, such as motor faults [Ince et al., 2016] and network intrusions [Vinayakumar et al., 2017]. In all these applications, a CNN anomaly detector proved to be efficient and accurate.

The main reason for this success is that a CNN can extract highly discriminative features efficiently. In a conventional neural network, all layers are fully connected, i.e., each neuron is connected to all neurons in the previous layer. In contrast, in convolutional neural networks, the layers are locally connected, i.e., each neuron is only connected to a small region of the previous layer. The parameters are shared with all neurons in one layer. As a result, CNNs require fewer parameters and are more efficiently computed. The locally connected feature of CNNs is achieved through

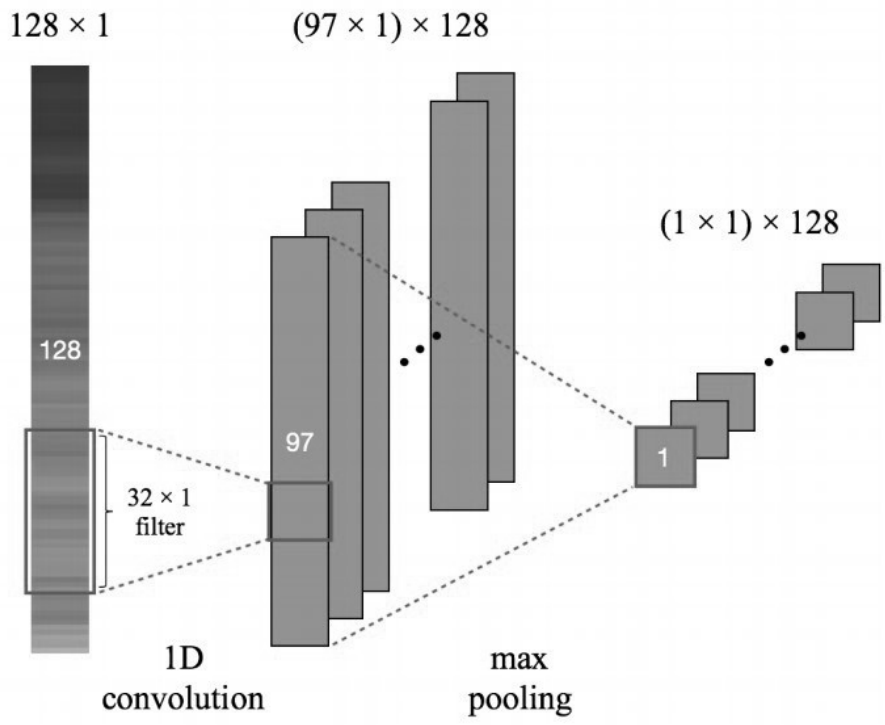

Figure 2.10: Example of One-Dimensional Convolutional Layer

two particular types of layers: *convolutional layers* and *pooling layers*. Convolutional layers use a parameter sharing scheme to control the number of parameters, where a one-dimensional window slides or convolves across the input data. The convolutional window is called a *filter* or *kernel*. The *stride* refers to the number of steps when the filter slides. The values of a filter are multiplied with the input data. Multiple filters can be applied to capture the features of the input data. In Figure 2.10 there are 128 elements in the input data. The filter size is set to 32, and there are 128 filters used in total. The output is 128-dimensional vectors, each of which contains 97 (128 − 32 + 1) elements.

Pooling layers are applied afterward to prevent overfitting by reducing the spatial

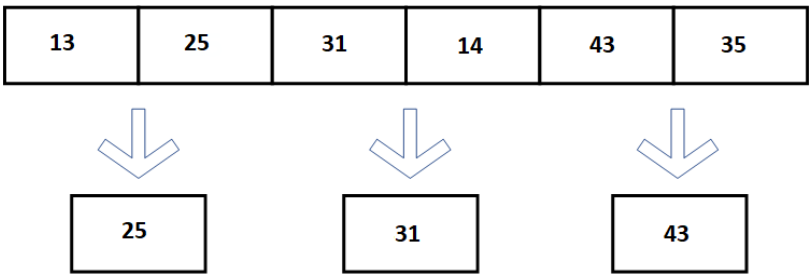

Figure 2.11: Example of One-Dimensional Maxpooling Layer

size and the number of parameters to be computed. A pooling layer summarizes the output from a convolutional layer, so it is sometimes also called a *subsampling layer*. The most commonly used pooling method is *maxpooling*, where only the maximum value within the pooling window is kept. Similarly, averaging the pooling keeps only the average value. In Figure 2.11, the pooling size is two, so the maximal value of every two elements is subsampled.

However, a CNN does not store any explicit memory for the past and its ability to model a time series is limited by the size of the time window [Frank et al., 2001]. The problem is that the window size needs to be carefully set. The dependencies in the time series cannot be captured using a window that is too small, while unnecessary information can be added when the window size is too large. Moreover, different sized windows may be required due to the long-range dependencies between training samples,i.e., an event downstream in time depends upon one or more events that came before.

CNNs may be suitable for specific datasets, where the input data is a multidimensional feature vector rather than as a sequence of data samples. The same computational steps are applied to each training sample without any memory of previous samples. For example, for an image recognition task, the sequence of the photographs

is unlikely to affect the trained classification. However, this assumption is violated in sequential datasets where the output of one step may depend on the inputs and outputs from previous steps. Like conventional neural networks, convolutional neural networks also have no explicit memory of the past, as all inputs and outputs are treated as independent of each other. In such cases, it is difficult to apply a memoryless network to variable sized inputs and outputs with nontrivial dependencies. Recurrent neural networks (RNNs) can provide a more elegant solution to solve these issues and handle complex time series dataset.

2.7 Recurrent Neural Networks (RNN)

Recurrent neural networks (RNNs) are designed for temporal tasks such as speech recognition by making use of temporal information. RNNs are called recurrent because they perform the same operation on every sample of a sequential input, with the output being dependent on the previous steps. For example, the ammonia concentration depends on its previous values. To predict that value accurately, the dependence needs to be memorized. An RNN layer receives its input sequence from the output sequence of the previous layer. The RNN layer also maintains the memory of the hidden state of the previous layer. The hidden state is maintained over all time steps as it captures temporal information about what has been calculated previously.

A single RNN layer with only one activation function can only operate at a single time scale. When the hierarchical information is complicated and the temporal information needs to be operated at different timescales, multiple RNN layers can be stacked on top of each other to form a *deep RNN (DRNN)* or a *stacked RNN*. In a DRNN, there are multiple layers with recurrent connections between the units in the

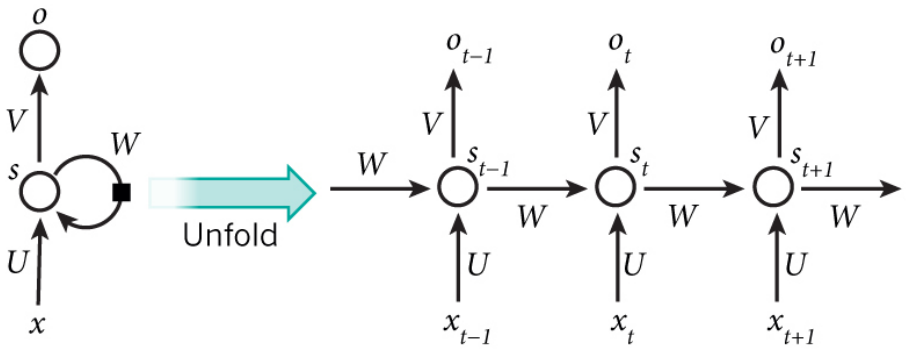

Figure 2.12: Unrolled Recurrent Neural Network Lecun et al. [2015]

same layer, as well as the conventional connections between units between different layers. Time dependencies can be learned for sequences with different lengths as the network is unrolled according to the length of the input sequence. For the ammonia dataset, the concentration is a function of the hour it occurs in a day. However, that value may also be affected by the day it occurs in a week. To predict as accurately as possible, it is beneficial to use DRNN, i.e. employing more than one recurrent layer.

RNN is implemented by unrolling a neural network on the temporal axis as shown in Figure 2.12. Each node at a time step takes an input from the previous node, and this is represented using a feedback loop. At time step t, x represents the input, s represents the hidden state, and o represents the output. The weights between different layers are represented by U, W, V, where U represents the weight between the input and hidden layers, W represents the weight for the recurrent transition between different hidden states, and V represents the weight for the hidden to output transition.

The main feature of an RNN is the hidden state, which is calculated based on the previously hidden state and the input at the current step. The first hidden state is typically initialized to all zeroes. The process of carrying memory forward can be

expressed mathematically. At any time t, the RNN unit receives the input x from the current time step and the hidden state s from the previous time step, where g represents the activation function and b represents the bias. The output is then calculated while the hidden state is also updated. The current output thus depends on all the previous inputs.

$$s_t = g(Ux_t + Ws_{t-1} + b_s) \tag{2.6}$$

Conventional RNN only learns information from previous time steps. However, in some cases, data samples from future time steps also matter. For example in language modeling tasks, it is beneficial to know the full context. *Bidirectional RNNs* were thus developed such that output depends on both the previous samples and the next samples [Schuster and Paliwal, 1997]. Two identical layers are given the same input sequence. The input sequence is fed in regular time order for one, and in reverse time order for the other. The outputs are calculated based on the hidden state of both networks. The results are usually concatenated at each time step. Bidirectional RNNs come with extra computational cost compared to conventional RNNs.

Training RNNs is similar to training conventional neural networks using back-propagation, where the number of layers is equivalent to the number of time steps in the input sequence. Comparing to a conventional neural network where parameters are different at each layer, RNN shares the same parameters U, W, V across all steps, as the same task is performed at each step with different inputs. The depth of RNN maintains the memory of temporal information instead of the hierarchical information. Neural networks can be viewed as nested composite functions, whether they are recurrent or not. Thus, adding a time element only extends the series of functions

for which the derivatives are calculated with the chain rule. The difference for RNN compared to the conventional neural network is that as the parameters are shared among all the time steps. As a result, the gradient at each output depends on the computation of the current time step as well as the previous time steps. For example, to calculate the gradient at *t=3*, two backpropagation steps are needed to sum up the gradients. This approach is called *backpropagation through time (BPTT)*. In theory, RNNs trained with BPTT can learn dependencies between steps that are arbitrarily far apart [Werbos, 1990], which may seem to be ideal to the ammonia dataset. However, in practice, the conventional RNN is only able to remember short-term memory sequences and thus becomes unsuitable for the application in this thesis. The limitation is caused by the *vanishing/exploding gradient problem*.

Vanishing/exploding gradient problems also happen in deep conventional neural networks. If each timestep is considered as a layer with weights going from one timestep to the next, the recurrent network will be at least as deep as the number of timesteps. This property makes the problem a lot more common for RNNs [Pascanu et al., 2013]. Vanishing gradient occurs when the gradients flowing through the network become very small as the chain rule is applied many times in the backpropagation [Hochreiter, 1998]. Two factors which affect the magnitude of gradients are the weights and the derivatives of activation functions that the gradient passes. A typical initialization of weights is from a Gaussian with mean zero and standard deviation one, which mostly yields weights of magnitude less than one. RNNs are formed with a chain of repeating modules. In a conventional RNN, each module has a simple structure such as a single Tanh layer as shown in Figure 2.13, where the derivative is bounded by 1. Multiplying these small numbers shrinks the gradient

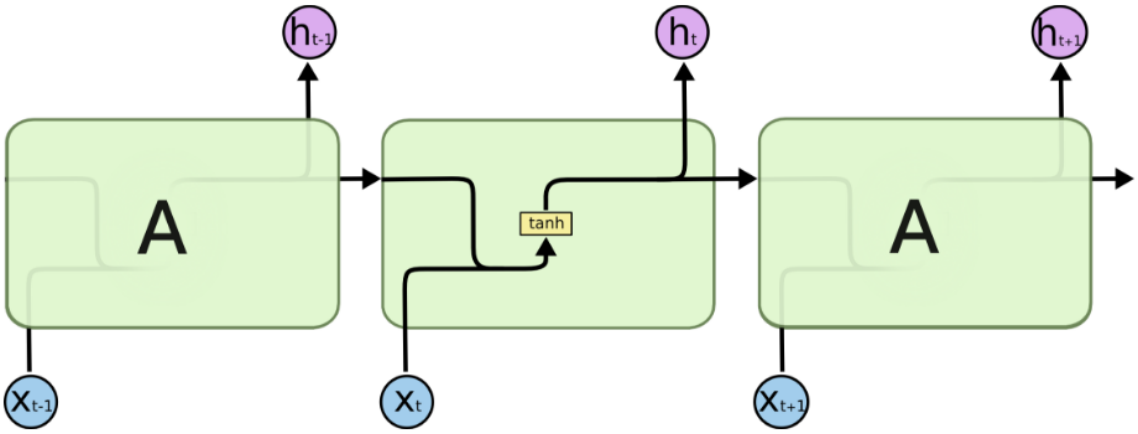

Figure 2.13: Conventional RNN Unit [Olah, 2015]

values exponentially. This problem may not be obvious for a shallow network. However, it usually requires more than one hidden layer to handle a complex dataset like the ammonia concentration. In such cases, the problem becomes much worse as more multiple layers of such non-linearities are stack on top of each other. The neurons in the earlier layers of the hierarchy thus learn very slowly. In the worst case, the network training may completely stop. The opposite case, i.e., exploding gradient problem [Pascanu et al., 2012] occurs when the magnitude is greater than one.

In practice, it is easier to handle the exploding gradients problem. Clipping the gradients at a pre-defined threshold is a very simple and effective solution [Bengio et al., 2013]. Several standard techniques have been proposed to solve the more problematic vanishing gradients. An ReLU activation function can be chosen instead of Tanh or Sigmoid activation functions, as the derivative of ReLU is one at every point above zero, creating a more stable network. Initialization of the weight matrix can be set more carefully such that the weight matrix is not initialized in the saturation region of the activation function. In recent time series applications, a more preferred

solution is to use a variant of RNN, called Long Short-Term Memory (LSTM). LSTM has proven to be useful in learning long-term dependencies, and it has become the model of choice for many temporal tasks. In the experiments in this thesis, LSTM was applied as the representation of RNN.

2.8 Long Short-Term Memory Network (LSTM)

LSTM (Long Short-Term Memory Networks) was first proposed in 1997 by the German researchers Sepp Hochreiter and Juergen Schmidhuber to solve the vanishing gradient problem [Hochreiter and Schmidhuber, 1997]. LSTM is perhaps the most popular RNN variant especially in temporal tasks like natural language processing. LSTM has been widely used in the field of anomaly detection [Malhotra et al., 2015] in different time series. For example, in an ECG application, LSTM was able to detect multiple types of abnormalities insides ECG signals [Chauhan and Vig, 2015]. LSTM was also successfully used in different fields such as automobile control [Taylor et al., 2016], and WWTP operation control [Inoue et al., 2017]. Even though there is not much previous research conducted on ammonia data, LSTM was able to learn temporal information and identify anomalies in all these time series applications. As the ammonia data is a typical time series with a repeating pattern, it can be deduced that LSTM should also be suitable for the ammonia dataset. But it is necessary to give a brief review of the development history and features to better understand the suitability of LSTM.

The structure of an LSTM network also contains the repeating module similar to conventional RNNs but it does more operations as shown in Figure 2.14. These modules are referred to as an *LSTM unit* or *LSTM block*. In a conventional RNN, the

vanishing/exploding gradient problem is caused by the repeated use of the recurrent weight matrix. The key idea behind LSTM is the cell state, which is decoupled from the output between hidden layers. Changes to the cell state are explicitly written by an explicit addition or subtraction so that each element of the state can stay constant without outside interference, which makes the memories in the cell state more stable. The gradient through the cell state is kept making it hard to vanish, while gradient explosion can be resolved through gradient clipping as discussed before.

In the original 1997 LSTM, the motivation was to make this recursive derivative a constant value so that gradients would neither explode or vanish. A new strcture called *constant error carousel (CEC)* was introduced as the central unit, and denotes the recurrent connection of the cell state. Its internal activation contains the state which acts as the memory for information from the past. The recurrent connection is a feedback loop with a time step equal to one. Gating mechanisms are introduced to enable the model to decide whether to accumulate specific information or not. The access to CEC is controlled by two gates, an *input gate*, and an *output gate*. The input gate prevents the information stored in CEC from being disturbed by the irrelevant input.

Similarly, the output gate prevents other units from interfering with information stored in CEC. When the input gate is open, the cell state is changed by addition. During the backpropagation, the error is reduced only when it enters and exits CEC, but it can stay unchanged in the CEC no matter how long distance it travels. This way the vanishing gradient problem is avoided. The original LSTM - CEC does not work well because the cell state tends to grow uncontrollably and eventually create an unstable network. In this case, it fails to learn the sequences in the dataset. As a

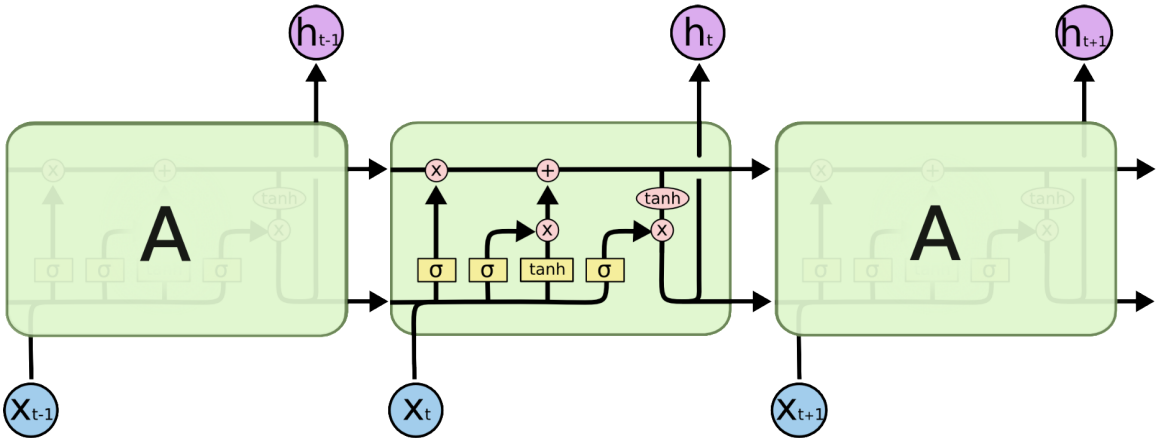

Figure 2.14: LSTM Unit [Olah, 2015]

result, in this thesis, original CEC was not employed. To solve these issues, a third gate, known as the *forget gate* was added to scale the previous cell state. Forget gates learn to reset LSTM memory when starting a new sequence. The forget gates allows the network to operate on different scales of time and therefore effectively model short as well as long-term dependencies. For the ammonia dataset, the dated values should be discarded with the forget gate, while the more recent values should pass through the input gate and get stored in the cell state. As a result, the modern LSTM as shown in Figure 2.15 was applied instead of the original CEC.

The internal structure is illustrated in Figure 2.14, where lines represent vectors between the output of one node and the input of others. The pink circles represent matrix operations. The yellow boxes represent learned neural network layers. Each layer is labeled with its activation function. As the purpose of the gates is to remove or add information to the cell state, the function for a gate is always Sigmoid. The output from the Sigmoid layer is a number between zero and one, where zero represents no

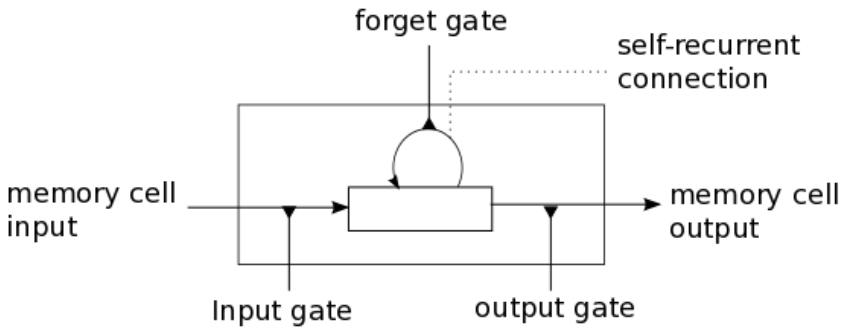

Figure 2.15: Three Gates of LSTM

information passed, and one represents all information passed.

The operation in an LSTM network can be broken down into three steps. The first one is the forget gate operation. The forget gate allows the model to discard irrelevant information from the previous cell state by evaluating the information given by the input at the current time step. The discard is implemented with the Sigmoid activation function, as it converges to 0 or 1. By setting it to 0, the information is forgotten. The memory also gets reset during this operation, when the memory is no longer relevant. The second operation is the update gate operation, where the model learns to accumulate specific information from the current time step by considering the previous output. The input gate allows specific information from the current time step to be added to the cell state. The third operation is the output gate operation, where the output of an LSTM unit is generated. The output gate is calculated using the same method as the forget gate and input gate. The purpose is to control what information flows out of the LSTM unit.

LSTMs proved to be successful in a variety of applications related to long-range dependencies, such as speech recognition, and language translation. In this project,

the data-driven experiments were conducted with a focus on the use of LSTM instead of conventional RNNs. As the LSTM unit is complicated, many variations of the LSTM architecture have been proposed to simplify the architecture. One popular LSTM variant is called *Gated Recurrent Units (GRU)* [Cho et al., 2014], which only has two gates, an update gate and a reset gate.

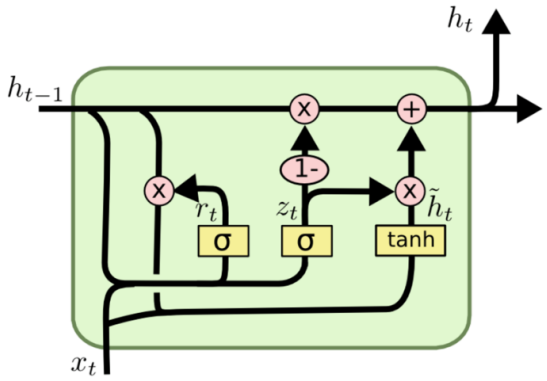

Figure 2.16: GRU Unit [Olah, 2015]

The problem of the vanishing gradient is eliminated by the update gate as it decides how much of the past information to pass to the future. The reset gate has the opposite functionality as it decides how much of the past information to forget. GRU is similar to LSTM, where the internal memory capacity is used to store and filter information using the gate mechanism as shown in Figure 2.16, and in many cases, GRU produces equally excellent results according to recent research [Jozefowicz et al., 2015].

Chapter 3

Datasets

3.1 Ammonia Data

In November 2017 this project was launched to acquire a real-world dataset from a municipal wastewater plant. Our industrial partner, Primodal, installed an RSM30 monitoring station in the Dundas Wastewater Treatment Plant (WWTP) in Hamilton, Ontario as shown in Figure 3.17. Two probes (Xylem VARiON Plus 700 IQ) with ion-sensitive electrodes were set up in the wastewater shown in Figure 3.18. The sensors took readings of both concentration and temperature.

Ammonia and nitrate were measured with potassium and chloride corrections. As the sensors were located at the influent to the activated sludge tanks, the concentration of nitrate was low and essential zero. The measured ammonia concentration was the target of this project.

Theoretically, anomalies can occur either at the data collection step or the data transmission step. In this case, there was no data transmission across a network. So, all the detected anomalies are assumed to occur at the data collection step, i.e.,

Figure 3.17: RSM30 Monitor Station at Dundas WWTP

sensors. As the sensors were working in a harsh environment, different procedures were undertaken to ensure the data quality. An automated air blast system was installed on the sensor and ran every four hours. One research team member also manually cleaned the sensors twice a week with a toothbrush as shown in Figure 3.19. Sensors are susceptible to drift errors over time so wastewater samples were taken from the WWTP and analyzed in the lab to compare with the sensor reading. The sensors on site were then calibrated when necessary to correct for any drift that had occurred since the previous adjustment.

Ammonia anomaly is difficult to define, as the "abnormality" largely depends on its context. Anomalies may also appear in different time scales. Specifically in this

Figure 3.18: Sensors in the Dundas WWTP

Figure 3.19: Before and After Cleaning

ammonia dataset, anomalies lasted for a few minutes (observed as a spike) such as Anomaly-1 in Figure 3.20 or several days (observed as an abnormal period). Due to the location of these sensors, the spike-type of anomalies or noise were common and usually occurred randomly. For example, debris at the sensor may cause the reading to change temporally. Sensor cleaning may also cause spike anomalies as the sensor was taken out of water for a few minutes and readings were thus disrupted during that period. In contrast, other anomalies last for much longer periods such as Anomaly-2 and Anomaly-3. In such cases, abnormal data was continuously recorded for multiple days. In this thesis, the focus was only on the long-term anomalies lasting for at least

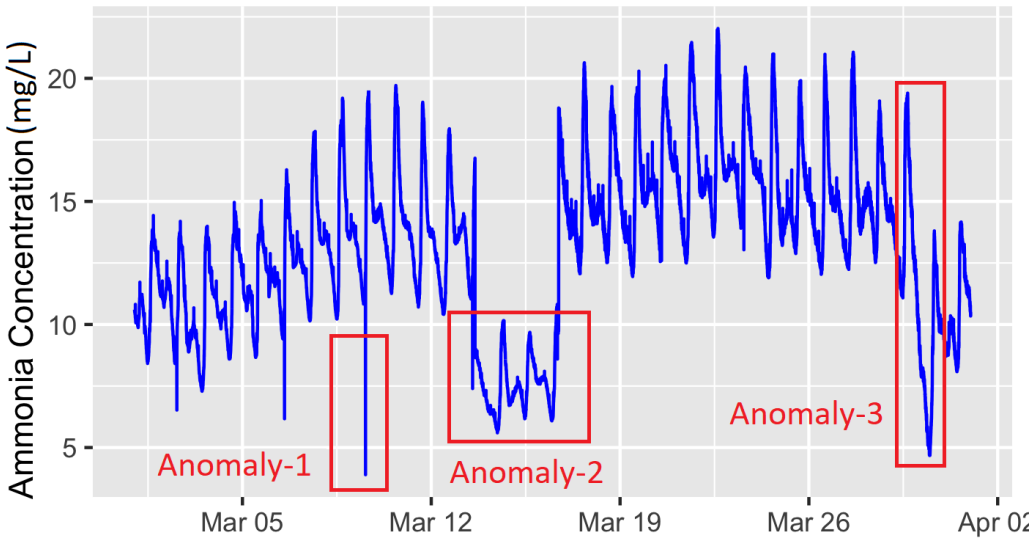

Figure 3.20: Concentration in March 2018

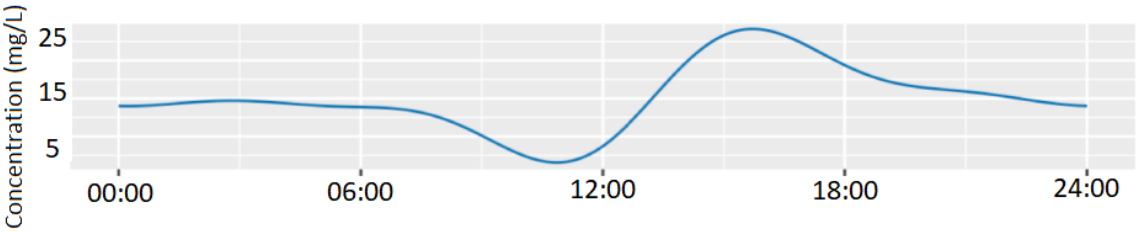

Figure 3.21: Ammonia Concentration Pattern in a Typical Dry Day

one day.

Ammonia collected at the treatment plant is influenced by daily, seasonal and weather issues and thus exhibits stochastic behavior. There are potentially many repeating patterns at different time scales such as weekly or monthly patterns. The focus of the anomaly detector in this thesis is to make use of the daily pattern only. On a typical dry day, the ammonia concentration demonstrates a repeating daily pattern as shown in Figure 3.21.

The ammonia concentration depends on both the ammonia load and the wastewater flow. The total ammonia load coming from industry and residents has a daily pattern, which is relatively stable regardless of the weather condition. However, the wastewater flow is profoundly affected by precipitation. Rainfall increases the flow amount and dilutes the ammonia concentration. The daily pattern of ammonia concentration is thus affected by rainfall as shown by Anomaly-3. In such cases, the concentration drops rapidly and comes back to its normal range in the following days.

"Serious" operational errors may also result in long-term anomalies such as Anomaly-2, caused by an incorrect calibration. In this case, the recorded concentration was approximately half of the normal readings. The sensor was re-calibrated at the next maintenance interval and normal readings returned. Another long-term anomaly is shown in Figure 3.22. In this case, the sensors were exposed to the air as the tank was pumped empty. As a result, the sensors recorded abnormal readings during that period.

There are eleven long-term ammonia anomalies in total in the collected dataset as determined by an expert in this field. Nine of the them were caused by precipitation, as evidenced by the measured behaviour and an available flow dataset. The other two operational anomalies were the ones in March and May as shown in Figure 3.20 and Figure 3.22.

3.2 Weather Data

It is beneficial to distinguish anomalies caused by precipitation from those caused by operational issues. The most straightforward way to distinguish precipitation-related

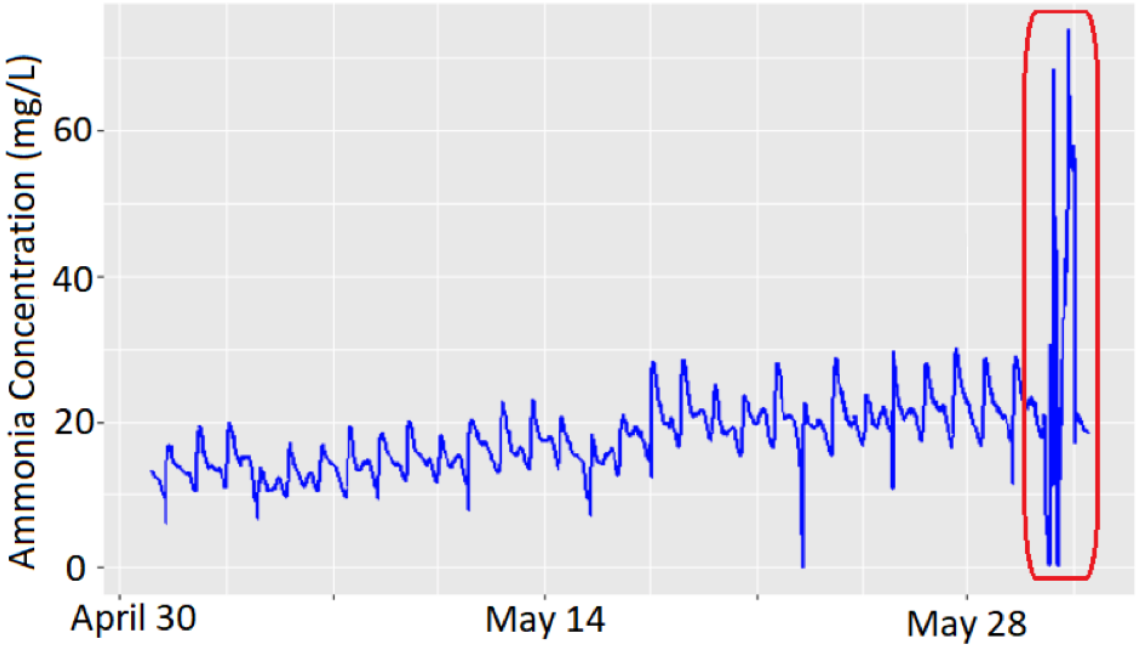

Figure 3.22: Concentration in May 2018

anomalies is to consult the precipitation data directly. Any concentration anomalies which occur on a rainy period are very likely to be caused by precipitation. To investigate this, precipitation data was downloaded from the Environment Canada website.

However, using precipitation data like this is limited. Firstly, the precipitation is only available daily, which is usually not enough resolution to feed into the neural network. Secondly, the precipitation data was collected from a meteorological station located 6 km northeast of the Dundas WWTP. The recorded precipitation therefore only reflects the situation at that location. In contrast, the wastewater flow into the Dundas WWTP comes from the whole catchment area through the sewer system. Thus, the precipitation data can only be used to estimate the flow qualitatively and not quantitatively.

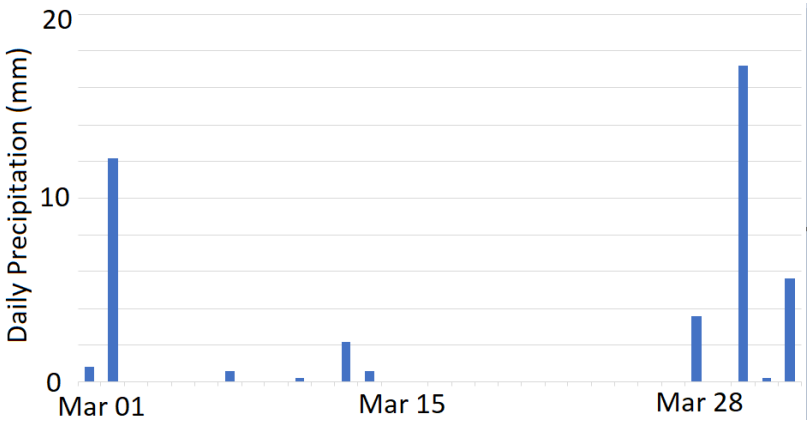

Figure 3.23: Precipitation in March 2018

In this case, because the sensors were installed at a treatment plant flow data was available. So the next option was to use the flow data as it is directly related to the concentration data. The flow increases with precipitation as shown in March 2018 in Figure 3.24. When there is no precipitation, the flow demonstrates a daily repeating pattern. After manual inspection, all similar anomalies in the flow data can be traced back to precipitation during the same period. Because anomalies occur in both concentration and flow when there is precipitation, the same detection process can be applied to these two datasets. By comparing the two detection results and eliminating the periods when they both exhibited anomalies, the "real" faulty data, such as Anomaly-2 in Figure 3.20, can be properly identified. Three months' data was provided by Dundas and five flow anomalies were identified. However, although flow data is collected by the Dundas WWTP, it should be noted that this data might not be available in other similar applications.

Temperature data on the other hand is much easier to access as the temperature sensor is attached to the concentration sensor. Like the concentration and flow data, the temperature data has a daily pattern, and can identify the precipitation indirectly

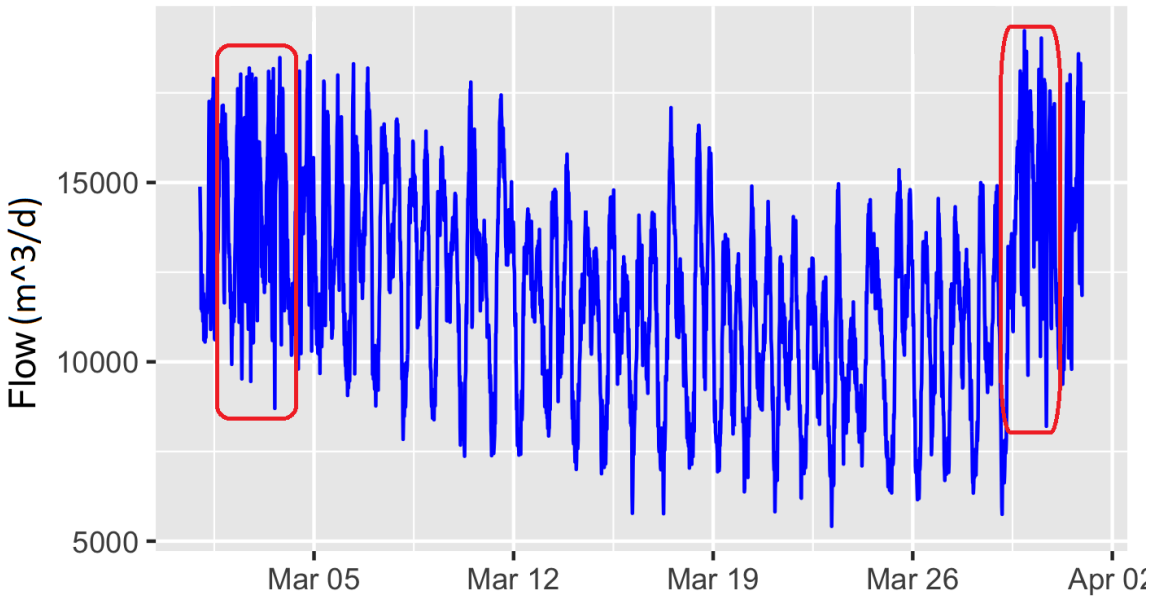

Figure 3.24: Flow in March 2018

under certain circumstances. For example, in Figure 3.25 the two temperature drops (observed as anomalies) are caused by precipitation. In these cases the temperature of the freezing rain or snow is lower than that of the wastewater. Note that the incorrect calibration of Anomaly-2 only affected the concentration but not the recorded temperature. The temperature data were normal during those three days. By comparing the concentration and temperature results, the "real" faulty data caused by the incorrect calibration can be identified. Ten temperature anomalies were identified in the sensor data.

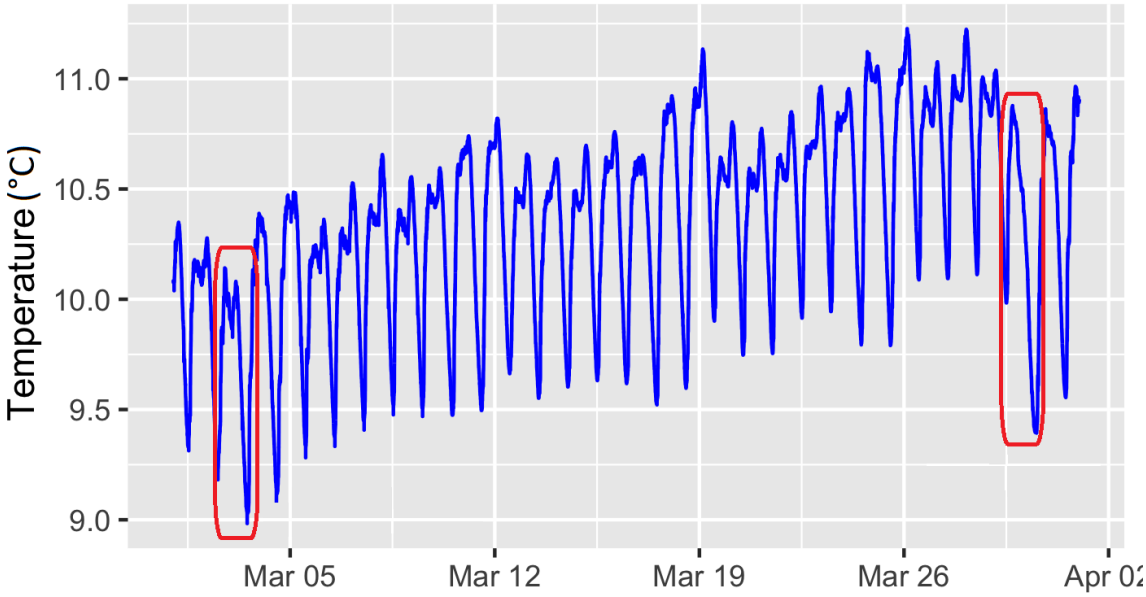

Figure 3.25: Temperature in March 2018

Chapter 4

Results and Discussion

4.1 Implementations and Hardware

The neural network was implemented with *TensorFlow* and *Keras* using Python. These libraries enable neural networks to be configured in a modular way by combining different neural layers, loss functions, optimizers, and activation functions. Neural networks with different hyperparameters needed to be compared to find the best choice for the ammonia dataset. The evaluation was conducted on both the detection results and training speed.

In this project, the training was done on a machine with a Nvidia GTX 1080 (8 GB of memory) and an Intel i7 6700 CPU (16 GB memory). Due to the required computations of deep learning, the training of a neural network can be significantly sped up by using a graphics processing unit (GPU). GPU-accelerated libraries for deep learning have been developed and NVIDIA has developed a library called *CuDNN* (CUDA Deep Neural Network library), where *CUDA* is a parallel computing platform and programming model. According to the documentation of Keras, CuDNN

provides highly tuned implementations of standard routines including both RNNs and CNNs. According to the documentation, a standard LSTM network is automatically accelerated by CuDNN whenever a suitable GPU is available. There are also special modules used by the GPU only. For example, CuDNNLSTM is the LSTM implementation which only runs on a NVIDIA GPU. Both CPU and GPU training were conducted, and their performances are compared.

4.2 Anomaly Detection Process

Anomaly detection in the ammonia data is handled by LSTM in a data-driven way. The method consists of three steps: training, prediction, and detection as shown in the Figure 4.26. In the training step, a model is learned from a training set, which contains information about the normal pattern. In the prediction step, each data point is predicted from its previous points and the predictive model. In the detection step, data points are identified as being anomalies based on the difference between the predicted value and the actual value.

The training of the predictive model takes most of the time and computing resources. In the training step, the selected training data is fed into the neural network. Two weeks of data from December 2017 was selected as the training set as shown in Figure 4.27, during this period there was no precipitation and no observable anomalies. The architecture of the network (i.e. the hyperparameters) is stored in a JSON file, where JSON stands for JavaScript Object Notation. The JSON file stores configuration information such as the activation function. The learned parameters (weights and biases) are stored in a HDF5 file, where HDF stands for Hierarchical Data Format. HDF5 file format is used as it is suitable for storing large collections of numeric

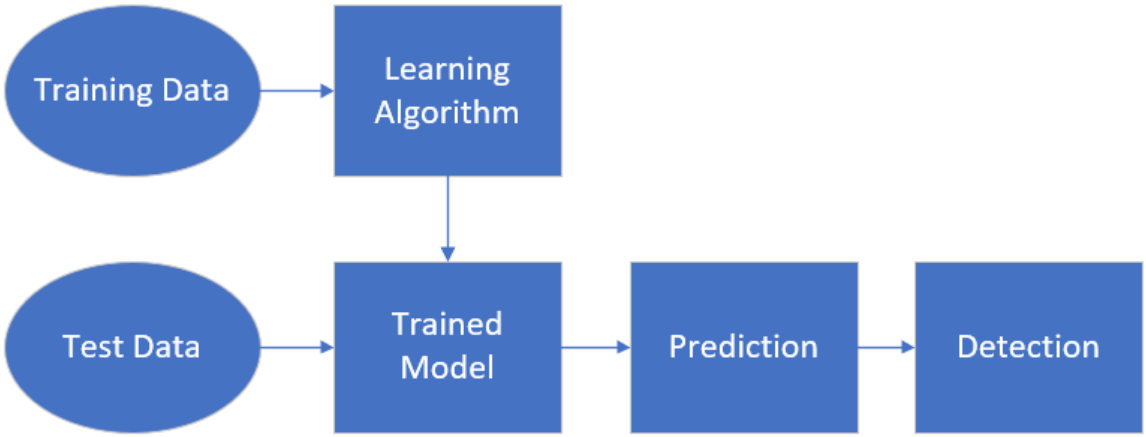

Figure 4.26: Anomaly Detection Process

arrays. Both of these are reused in the prediction part by Keras.

```
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
# Predefined Hyperparameter information is stored as a JSON file

model.save_weights("model.h5")
# Learned parameter information is stored as a HDF5 file
```

Listing 4.1: Save Predictive Model

The test data, which is new to the network, is then fed into a network using the same hyperparameters. For this project, six months of ammonia data from January 2018 to July 2018 were used. During this period, sensors were taken out for maintenance from time to time, and no readings were recorded during those periods as shown in Figure 4.28. As a result, the prediction step cannot be done on the whole six months' dataset all together. The longest continuous sections from each month

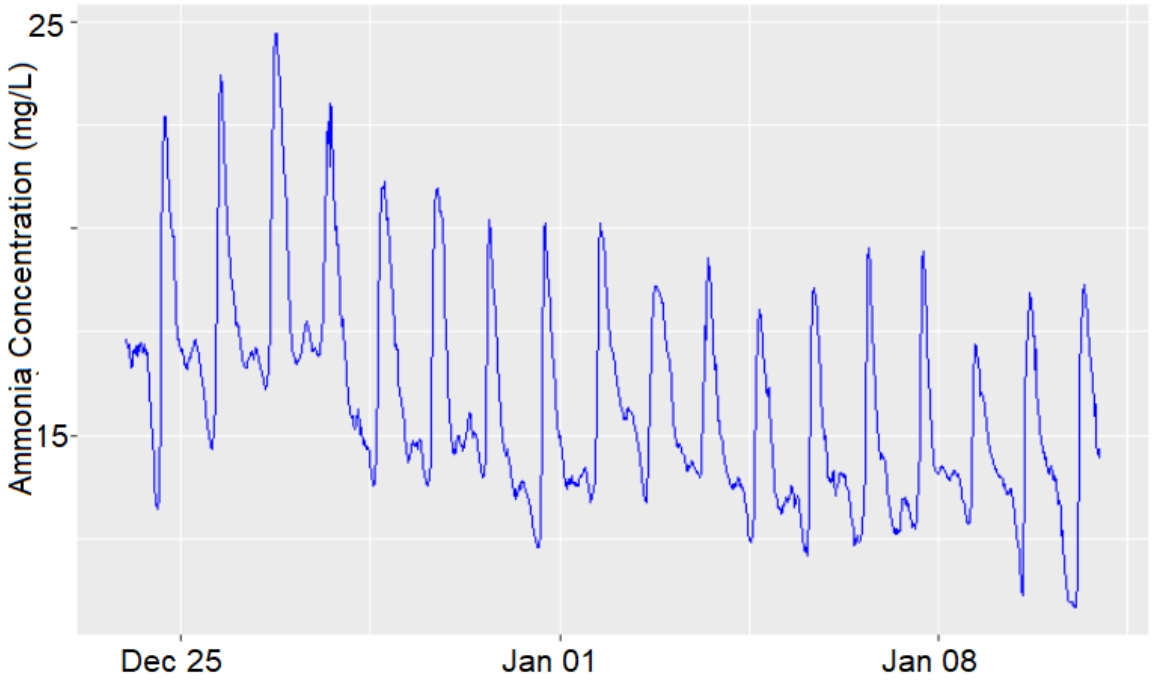

Figure 4.27: Training Data

were thus selected as the test set, such that anomalies (red boxes) were included in the test set but not the missing data (orange box).

Predictions based on previous samples and the predictive model (HDF5 file) were calculated as shown in Figure 4.29. For example, given values at time $k-5$, $k-4$, ..., $k-1$, the value at time k was predicted with the predictive model. Next, the future value at $k+1$ was predicted with historical values between $k-4$ and k. Even though it is possible to predict multiple time steps, the lookahead (the number of predicted samples) was set to one in this project, as the prediction accuracy usually decreases as lookahead increases.

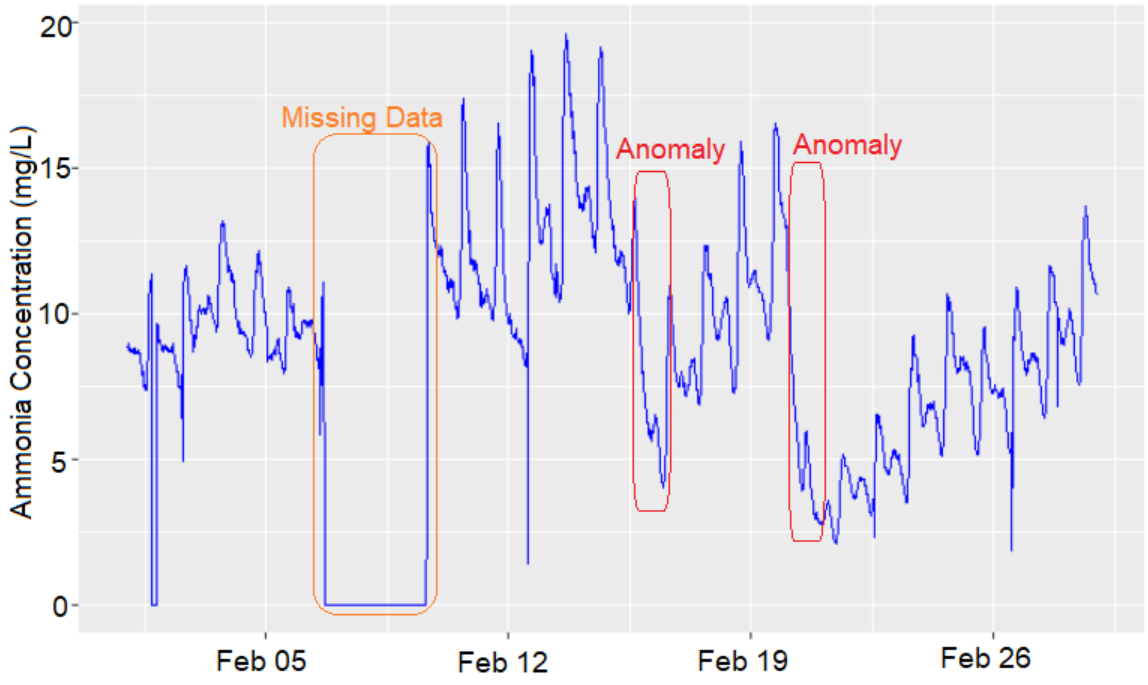

Figure 4.28: Raw Data for February 2018

```
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# JSON file is loaded to create model for test set


loaded_model.load_weights("model.h5")
# HDF5 file is loaded
```

Listing 4.2: Load Predictive Model

For the detection stage, the prediction error is calculated by the difference in the predicted value and the actual value received. The prediction errors are analyzed

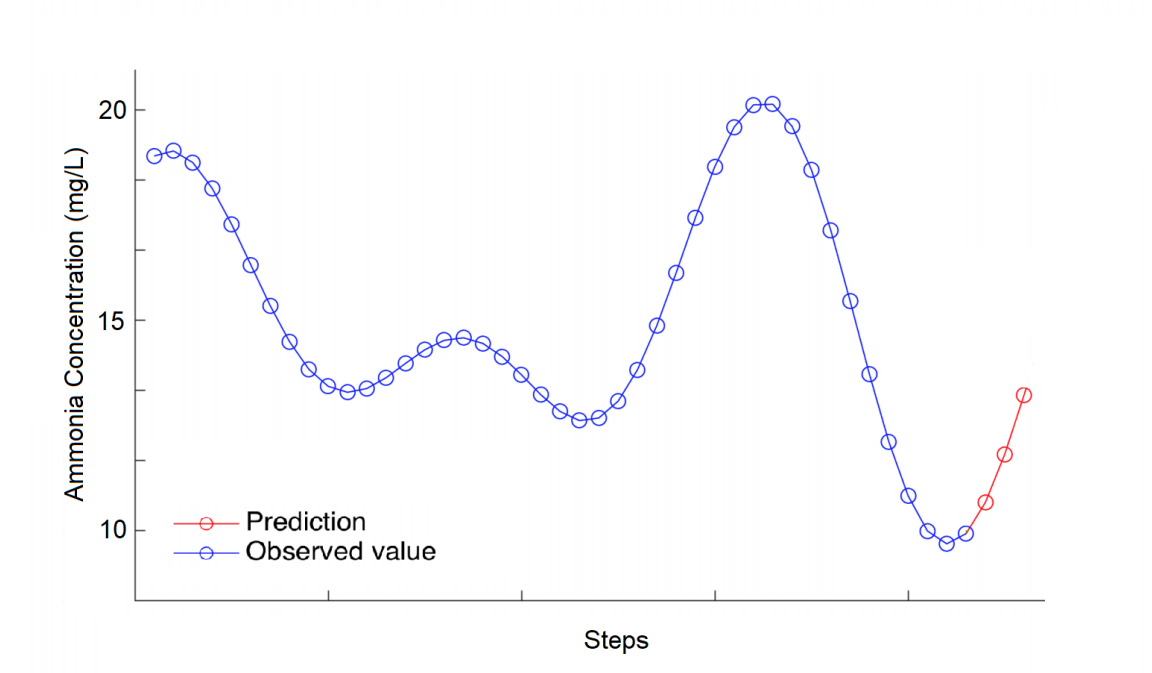

Figure 4.29: Predicting Future Values using the Past Values

and modeled using the Gaussian Tail Probability defined by *Numenta*, where the anomaly score is calculated based on the probability of errors [Taylor, 2018]. The anomaly score represents the likelihood that a point is an anomaly, with 0.0 being the lowest likelihood and 1.0 being the highest likelihood. For records that arrive every minute, an anomaly score or likelihood of 0.0001 means it occurs once out of every 10,000 samples, or about once every seven days. Ideally the threshold is set such that anomalies can be detected with as few false alarms as possible. Any score higher then the threshold is assumed to be an anomaly.

```
def compute_scores(y_validate, y_pred, normalize=True):
    errors = np.array((y_validate - y_pred) ** 2)
    # Prediction error is calculated as the differentially square
        between predicted and actual values

    if normalize:
        errors = errors / float(errors.max() - errors.min())
    # Prediction error is normalized

    likelihoods = []
    anomaly_likelihood = AnomalyLikelihood()
    for i in range(len(y_validate)):
        likelihood = anomaly_likelihood.anomalyProbability(y_validate[i
            ], errors[i], timestamp=None)
        likelihood = anomaly_likelihood.computeLogLikelihood(likelihood)
        likelihoods.append(likelihood)
    # Anomaly Score is calculated based on the probability of prediction
        errors

    N = len(likelihoods)
    anomalies = {'Guaranteed': np.zeros(N), 'Possible': np.zeros(N)}
    x = np.array(likelihoods)
    high_idx = x >= 0.5
    anomalies['Guaranteed'][high_idx] = 1
    # Any data points with score above the threshold is determined to be
        an anomaly
    return errors, likelihoods, anomalies
```

Listing 4.3: Detect Anomaly for Test Set

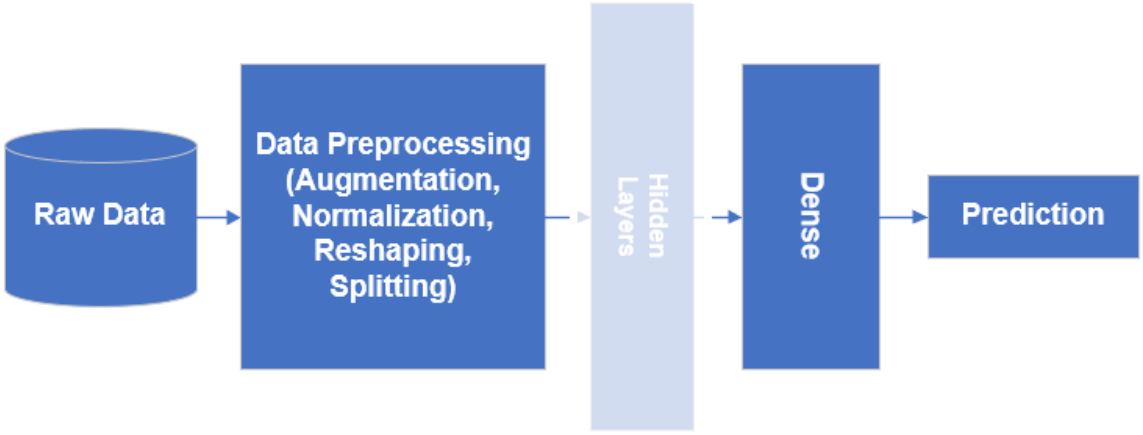

Figure 4.30: Neural Network Structure

The evaluation metrics of the detection algorithms focus on both the *false positives* and *false negatives*. A false positive occurs when a data sample is identified as abnormal, whereas it is normal. A false positive is also referred to as a false alarm. In contrast, a false negative occurs when an anomaly is identified as normal, while a true positive occurs when an anomaly is correctly identified as abnormal. An appropriate anomaly detector should detect as many anomalies as possible, while signaling as few false alarms as possible.

4.3 Neural Network Structure

Neural networks with different hyperparameters were examined. Each of them was formed with an input layer, one or more hidden layers, and an output layer. There were four preprocessing operations performed before the data was trained by the network: *data augmentation*, *data normalization*, *data reshaping*, and *data splitting*.

Data augmentation and data splitting are applied only to the training set and not the test set. Both are used to prevent overfitting. Data augmentation is the

opposite operation to dropout. In dropout, partial connection and corresponding weights between layers are discarded, while in data augmentation a small amount of training data is randomly duplicated and added to the training set.

```
duplication_ratio = 0.04
# The ratio of duplication is predefined as 0.04

def augment_data(X, y, duplication_ratio):
    nb_duplicates = duplication_ratio * len(X)
    X_hat = []
    y_hat = []
    for i in range(len(X)):
        for j in range(np.random.random_integers(0, nb_duplicates)):
            X_hat.append(X[i, :])
            y_hat.append(y[i])
    # The randomly duplicated data is added to the training set
    return np.asarray(X_hat), np.asarray(y_hat)
```

Listing 4.4: Data Augmentation

The *normalization* or *standardization* is a useful technique to reduce the variations in raw input data which may be significantly different in amplitude. The anomalies related to the amplitude may be handled by other threshold detectors. The focus of this thesis is to detect anomalies by using the daily pattern instead of the amplitude. According to some time series research in structural pattern mining [Shalabi and Shaaban, 2006], z-normalization helps mining algorithms focus on the structural similarities or dissimilarities rather than on the amplitude. The normalization used in this project, *z-normalization*, ensures all samples of the input vector are transformed

into the output vector whose mean is approximately zero with a standard deviation close to one. The mean of input $\bar{x}$ is first subtracted from the original values, and the difference is then divided by the standard deviation σ.

$$x = \frac{x - \bar{x}}{\sigma} \tag{4.7}$$

```
def z_norm(result):
    result_mean = result.mean()
    # Calculate the mean value
    result_std = result.std()
    # Calculate the standard deviation
    result -= result_mean
    result /= result_std
    # Conduct the z-normalization
    return result, result_mean
```

Listing 4.5: Data Normalization

Keras requires that the data be in a specific array so the data needs to be reshaped. A time window size needed to be specified in the data reshaping step. Even though multiple repeating patterns may exist in different time scales, the focus here was on the daily pattern. The window size is thus set as the length of a day. For example, one day's data had 96 readings when the readings were taken every 15 minutes. As the lookahead is one, the lookback or number of previous steps is 95 (96 − 1).

```
time_window_size = 96
# The size of window when reading is taken every 15 minutes
def create_train(data, time_window_size, duplication_ratio, normalize=
    True):
    nb_records = len(data)
    result = []
    for index in range(len(data) - time_window_size):
        result.append(data[index: index + time_window_size])
    result = np.array(result)
    # The window with size 96 slides through the training set, creating
        multiple training samples
    if normalize:
        result, result_mean = z_norm(result)
    print("Train data shape  :", result.shape)
    # The training samples are normalized
    train = result[:len(data), :]
    np.random.shuffle(train)
    X_train = train[:, :-1]
    y_train = train[:, -1]
    # The lookback is the first 95 readings in any window, while the
        lookahead is the last reading to be predicted
    X_train, y_train = augment_data(X_train, y_train, duplication_ratio)
    # The data augmentation is applied
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1],
        1))
    # The training samples are reshaped as numpy array to be fed into
        the input layer
    return X_train, y_train
```

Listing 4.6: Data Reshaping

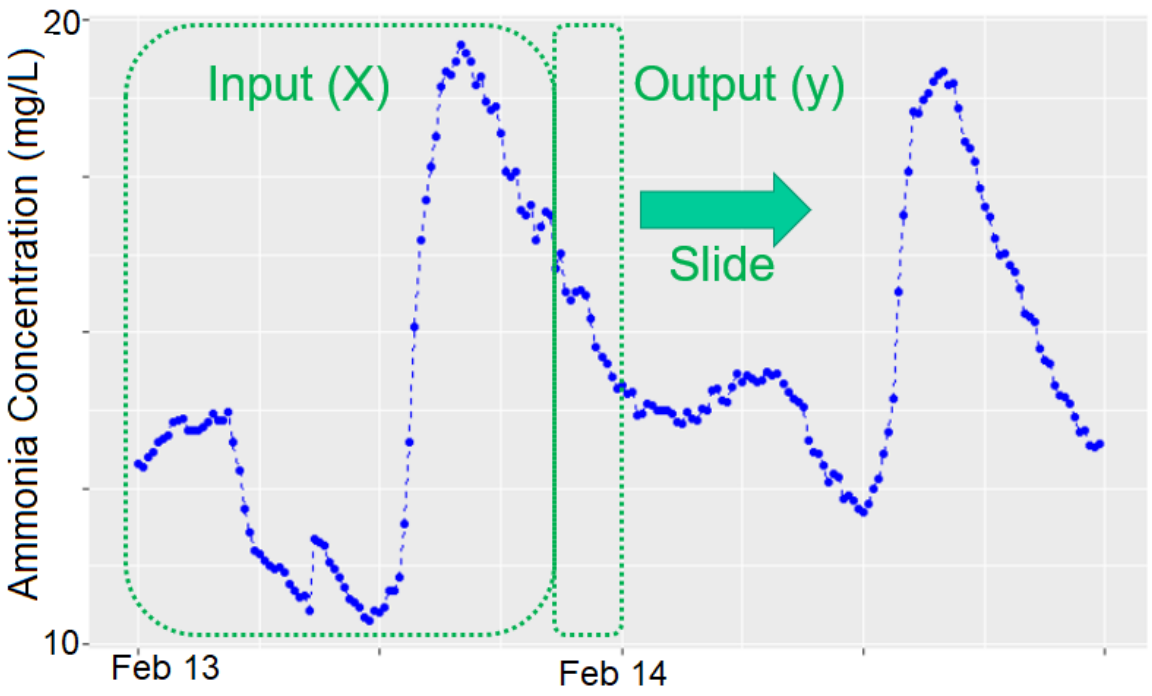

Figure 4.31: Feed Samples with a Shifting Window

Different training samples are overlapped and fed into the input layer as shown in Figure 4.31. The training set and test set are reshaped in a similar way, except that data augmentation is used for the training set.

In the last step, data splitting is employed. A small portion of the training set is split into a validation set as shown in Figure 4.32. In this case, twenty percent of the data was split into a validation set, and the rest was used for training. The loss for the validation set is calculated at the end of each epoch. When the loss value stops improving or even gets worse, training is interrupted.

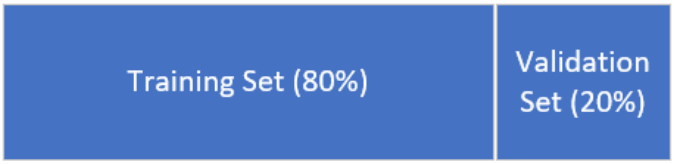

Figure 4.32: Splitting Data into Training and Validation

```
checkpoint = ModelCheckpoint(filepath='models/best_model.h5', monitor='
    val_loss', save_best_only=True)
# The trained model after each epoch is recorded, but only the best
    model is saved.

early_stopping = EarlyStopping(monitor='val_loss', patience=3, mode='
    auto')
# If the loss function for the validation set is not improving in the
    recent three epochs, the training is interrupted and the model three
     epochs ago is saved as the best model.

csv_logger = CSVLogger('models/history.csv', append=True, separator=',')
# The values of cost function are recorded for both training and
    validation set.

history_callback = model.fit(X_train, y_train, batch_size=batch_size,
    epochs=epochs, validation_split=0.2, verbose=1, callbacks=[
    checkpoint, csv_logger, early_stopping])
# Twenty percent of the training data is split as validation set.
```

Listing 4.7: Data Splitting and EarlyStopping

After the data has been preprocessed and fed into the input layer, different types

of hidden layers with different hyperparameters are used to learn the temporal information from the datasets. Dropout is applied between consecutive layers to avoid overfitting.

```
model.add(Dropout(0.2))
# Twenty percent of units are randomly discarded.
```

Listing 4.8: Dropout

Different networks use the same input and output layers, and the differences mainly exist in the hidden layers. CNNs were examined first due to their simple structure. LSTM and GRU were then used as representatives of RNNs. The comparison of different networks with different hidden layers is discussed in the next chapter.

The output layer is a fully connected dense neural network layer. In this project, a linear function was chosen as the activation function in the output layer as the continuous values are predicted. The number of neurons in the output layer is usually equal to the lookahead value, where each neuron represents one predicted future value. As only one value is predicted with the previous 95 readings in the time window, the output layer only contain one neuron. The loss function was chosen to be MSE (mean squared error) while Adam was chosen as the optimizer.

```
layers = { 'input': 1, 'hidden1': 64, 'hidden2': 256, 'hidden3': 100, '
    output': 1}
# The number of neurons of each layer in a four layer network

model.add(Dense(units=layers['output']))
# The output Layer contains only one neuron.

model.add(Activation("linear"))
# Activation function is chosen as linear function.

model.compile(loss='mse', optimizer='adam')
# Loss function is chosen as "MSE" and Optimizer is chosen as "Adam".
```

Listing 4.9: The Output Layer

4.4 Rule Based Method

Two libraries were used for rule-based anomaly detection. The S-H-ESD algorithm was implemented using the R library *AnomalyDetection* by Twitter [Twitter, 2015]. The moving average algorithms were implemented using the R library *Surus* by Netflix [Netflix, 2015]. These two methods are not learning-based methods, and they do not require a training step or much manual tuning. The test data was directly fed to the detectors, and the anomalies were detected automatically.

The S-H-ESD algorithm was applied to all the test data, with the detected anomalies indicated by green cycles. Two sample results were shown in Figure 4.33 and Figure 4.34. S-H-ESD was unable to detect precipitation-related anomalies.

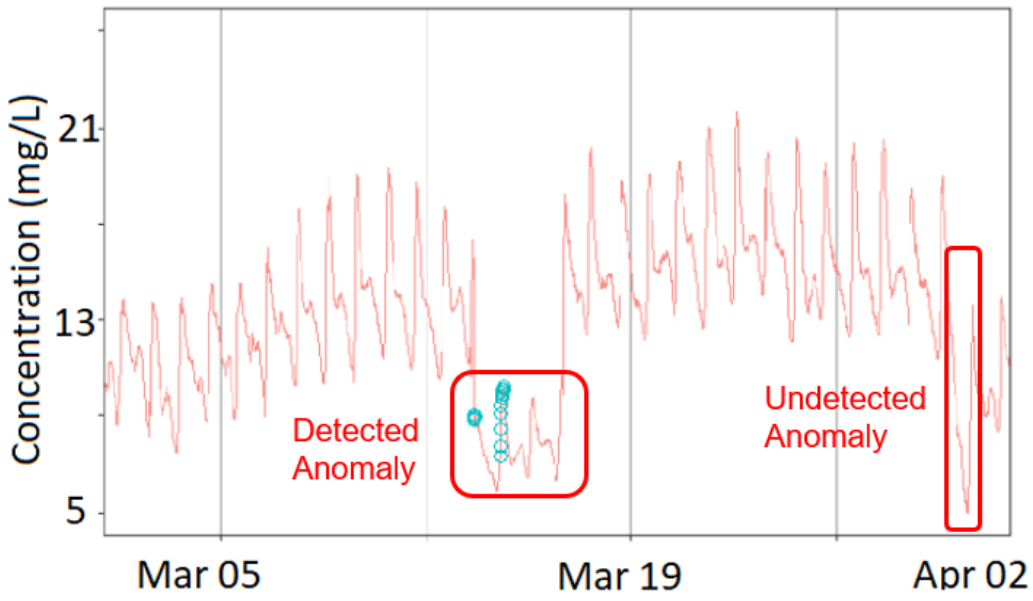

Figure 4.33: Sample Result for March 2018

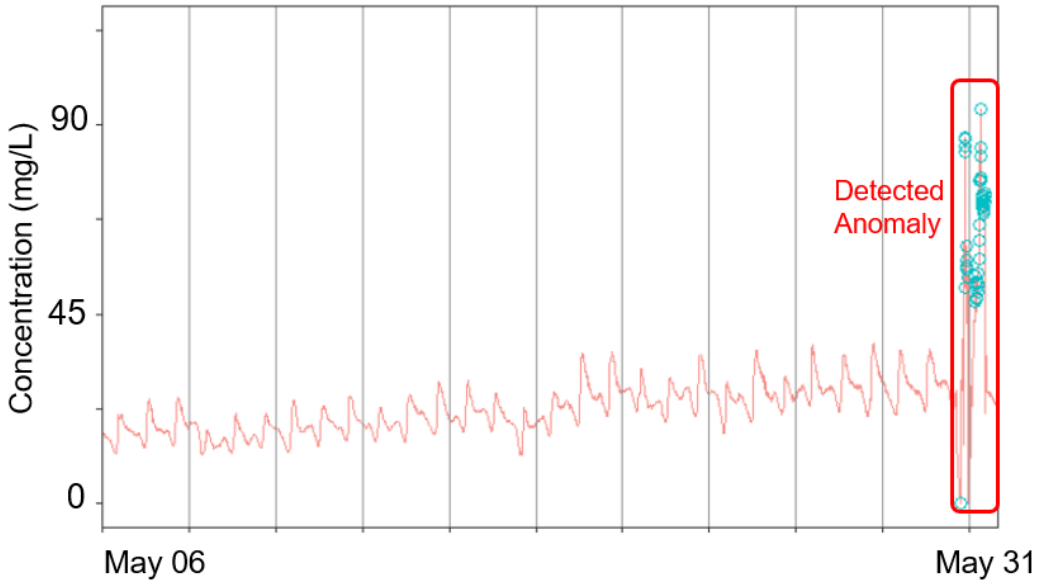

Figure 4.34: Sample Result for May 2018

The moving average algorithm was also applied to all test data, with the detected anomalies indicated by red dots. Two sample results are shown in Figure 4.35 and

	S-H-ESD Method
True Positive	2
False Positive	0
False Negative	9

Table 4.1: Detection Result by S-H-ESD Method

Figure 4.36. This method was able to detect precipitation-related anomalies, but it also returned a large number of false alarms.

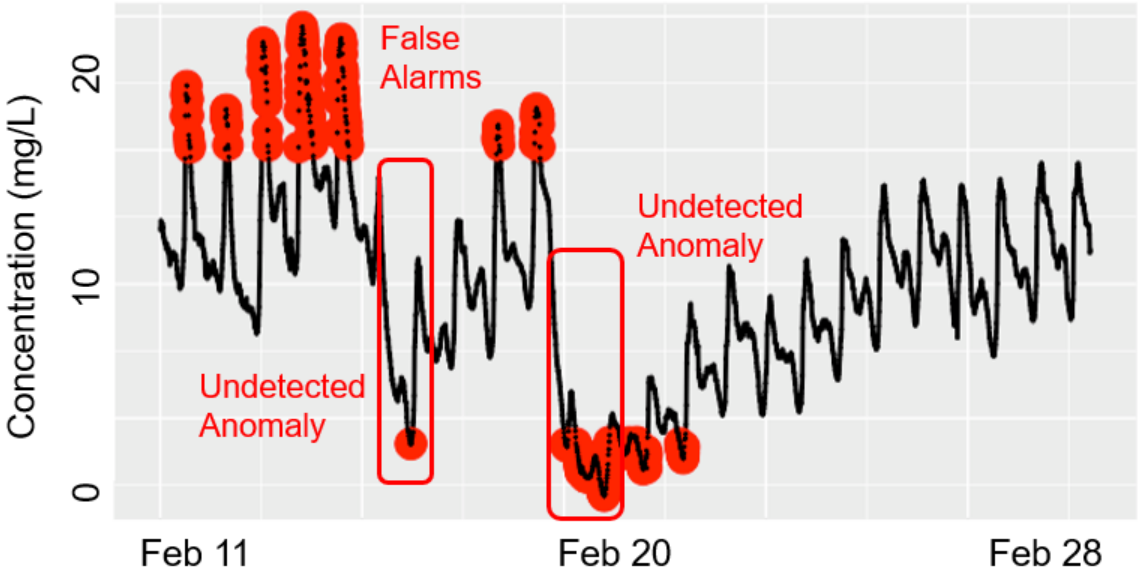

Figure 4.35: Sample Result for February 2018

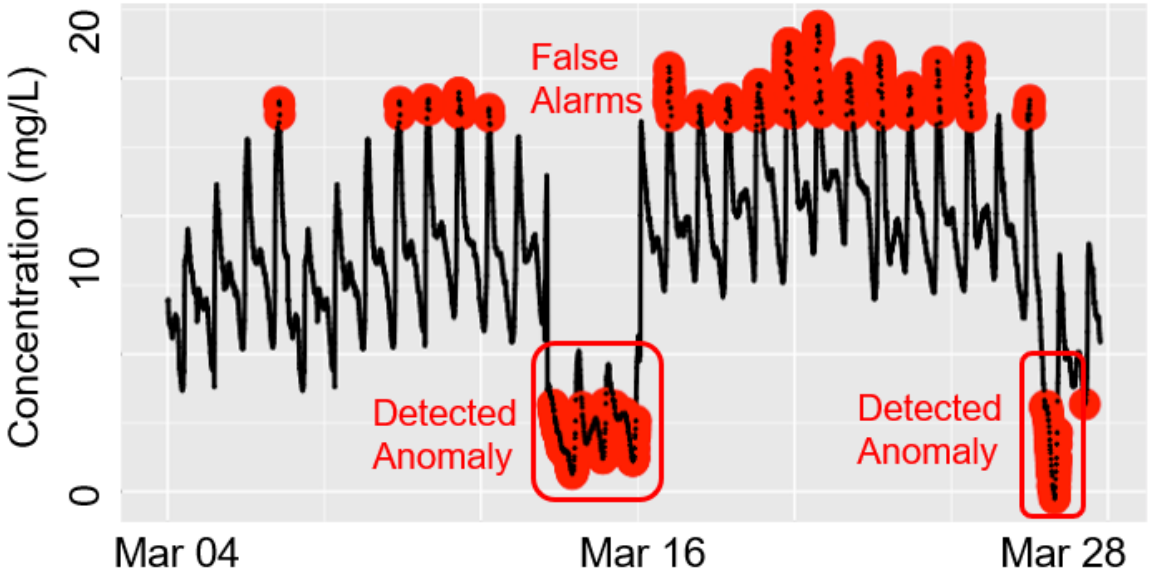

Figure 4.36: Sample Result for March 2018

	Moving Average Method
True Positive	3
False Positive	Numerous
False Negative	8

Table 4.2: Detection Result by Moving Average Method

4.5 Learning Based Method

The experiments in this thesis were conducted in an exploratory way, from simple to more sophisticated techniques.

4.5.1 CNN Based Method

The data was first analysed using an one-dimensional convolutional layer, *conv1D* in Keras. The CNN method used in this project was comprised of a single Conv1D layer,

a maxpooling layer and dense output layer with one neuron as shown in Figure 4.37.

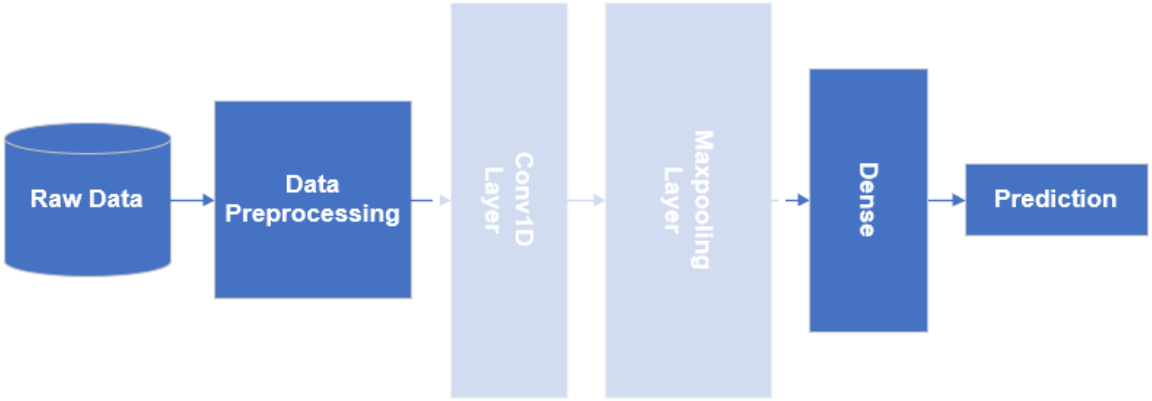

Figure 4.37: CNN Based Network

```
model.add(Conv1D(filters=256, kernel_size=5, padding='same', activation=
    'relu', input_shape=(time_window_size - 1, layers['input'])))
model.add(Dropout(0.2))
```

Listing 4.10: CNN Based Method

The training error for every iteration is shown in Figure 4.38. The validation error was used to avoid overfitting, as such the training stopped when the validation error was at its lowest level.

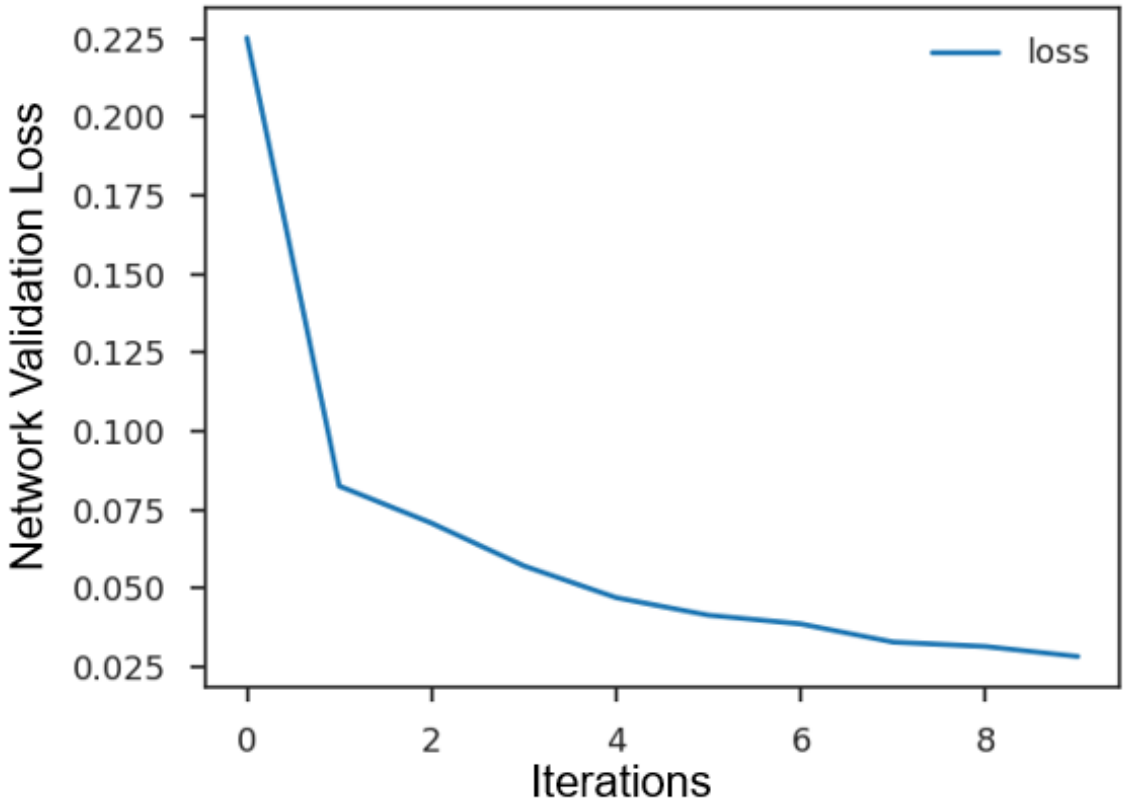

Figure 4.38: MSE Loss with Iterations

The CNN based methods detected more anomalies than the previous rule based methods and detected eight out of the eleven anomalies, as shown in Figure 4.39. However, the CNN detector returned a high rate of false alarms as shown in Figure 4.40.

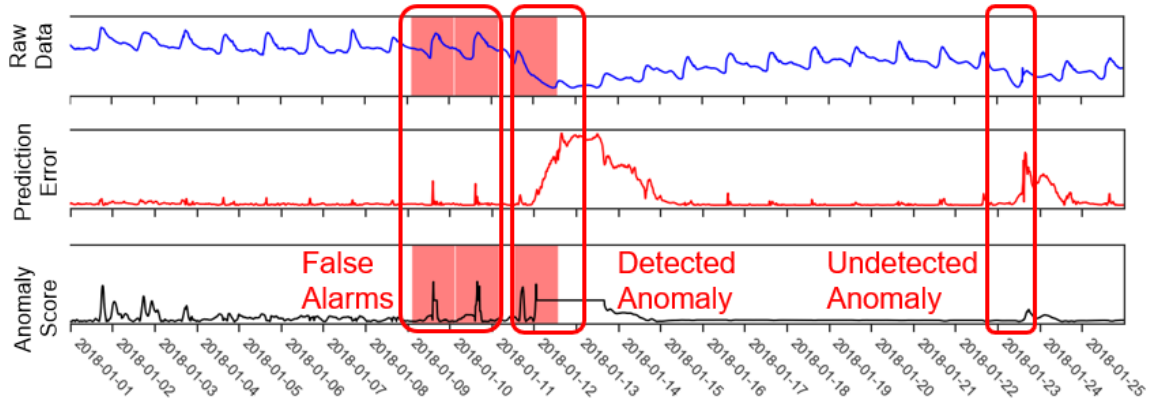

Figure 4.39: Sample Result for January 2018

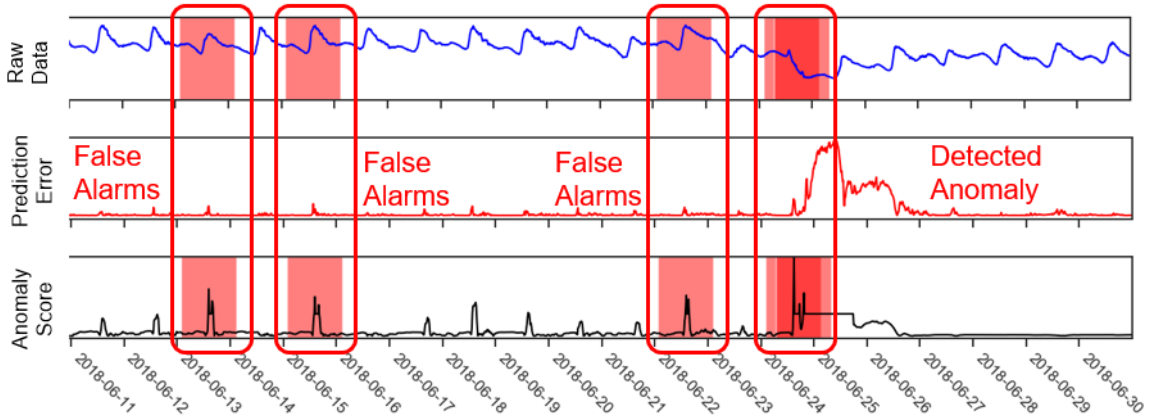

Figure 4.40: Sample Result for June 2018

	CNN Based Method
True Positive	8
False Positive	6
False Negative	3

Table 4.3: Detection Result by CNN Based Method

4.5.2 LSTM Based Method

Because LSTM can remember past information, it was expected that this method would provide a better result than the CNN method. Training was conducted using the same training dataset. The neural network used the same input and preprocessing steps, and only the hidden layers were replaced with LSTM. The LSTM method has a structure as shown in Figure 4.41.

Figure 4.41: LSTM Based Network

```
model.add(LSTM(units=layers['hidden1'], input_shape=(time_window_size -
    1, layers['input']), return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=layers['hidden2'], return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=layers['hidden3'], return_sequences=False))
model.add(Dropout(0.2))
```

Listing 4.11: LSTM Method

The training error from every iteration is shown in Figure 4.42. A lower calculated loss can be achieved with the LSTM method compared to the CNN method.

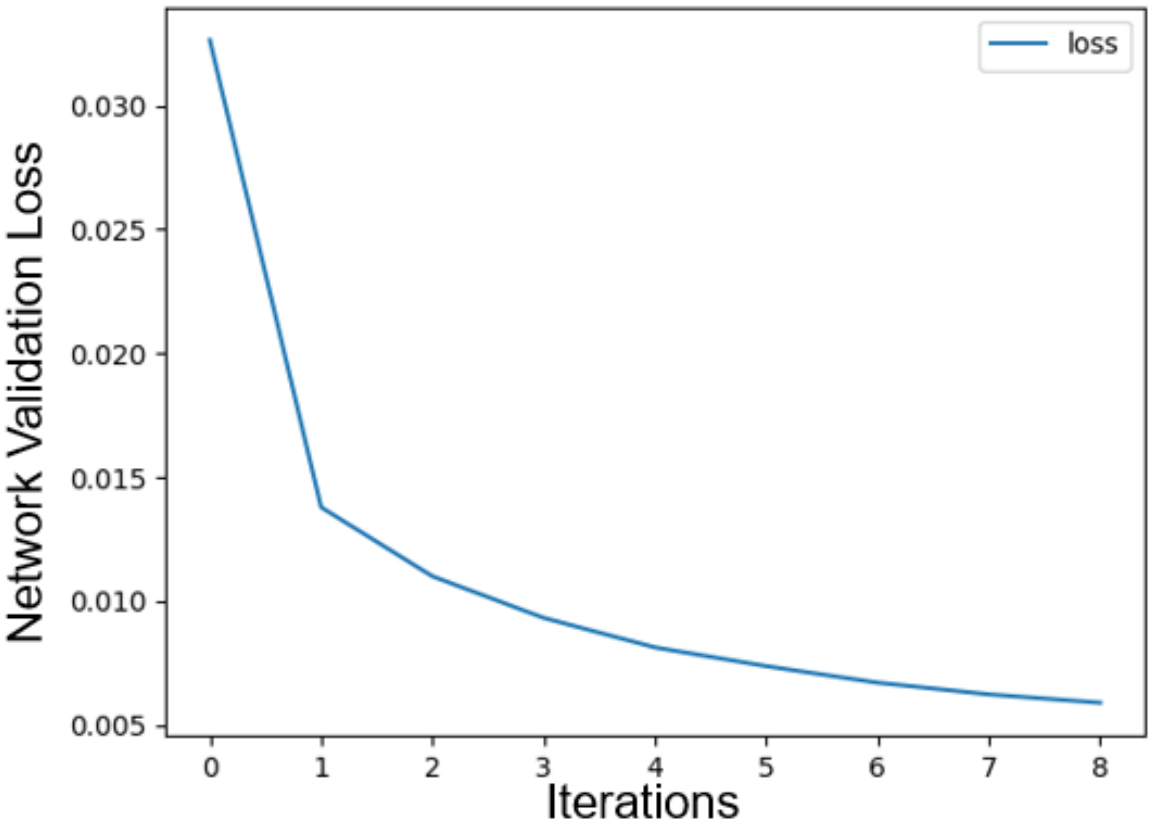

Figure 4.42: MSE Loss with Iterations

The LSTM method outperformed the CNN method by detecting more anomalies with fewer false alarms as shown in Figure 4.43 and Figure 4.44.

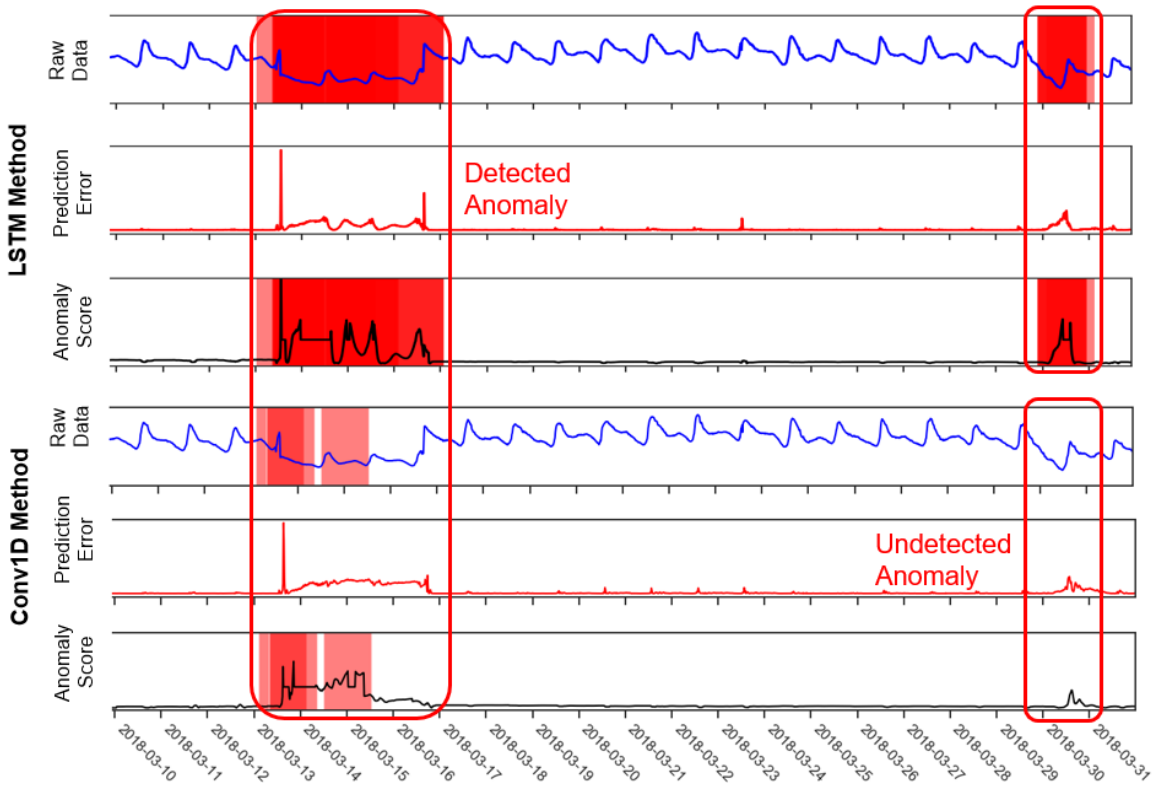

Figure 4.43: Sample Result for March 2018

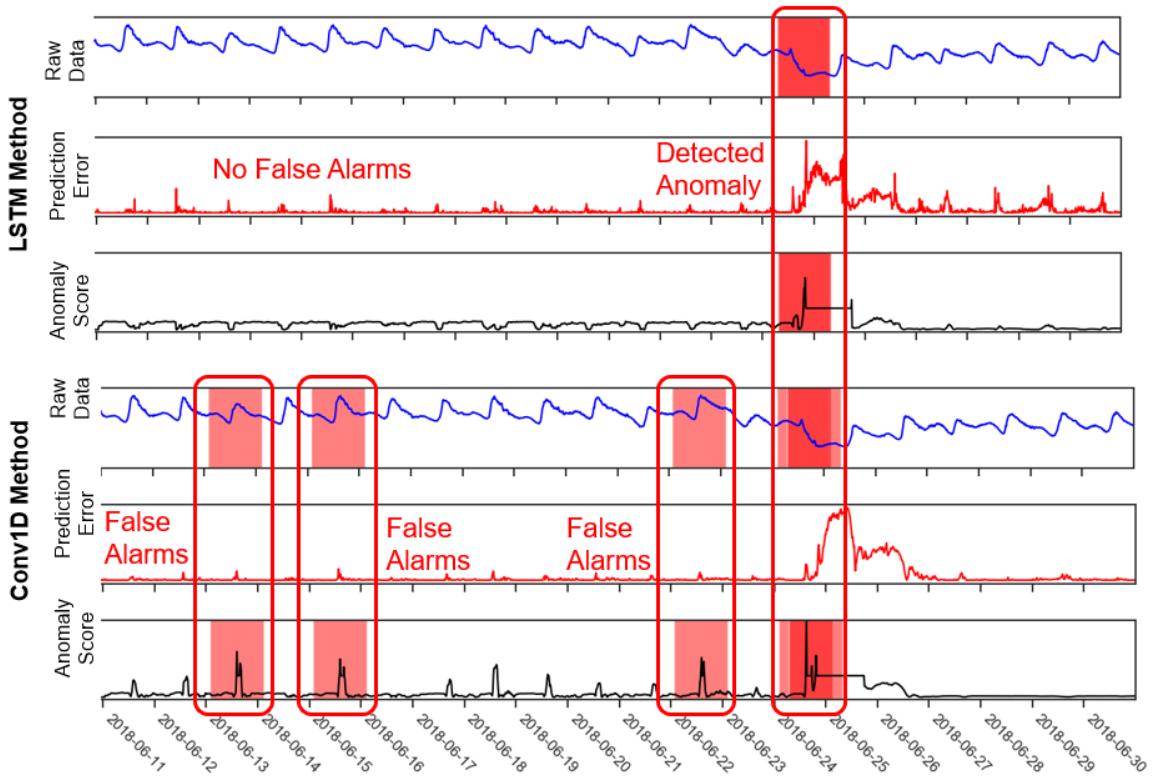

Figure 4.44: Sample Result for June 2018

Two variants of LSTM, GRU and bidirectional LSTM, were also used, where the LSTM layers in the neural network were replaced with GRU or bidirectional LSTM layers. Each variant (standard LSTM, GRU, bidirectional LSTM) returned the same number of anomalies. Each detected ten out of eleven anomalies with one false alarm. The major difference among these three LSTM variants was the training time.

	LSTM	GRU	Bidirectional LSTM
True Positive	10	10	10
False Positive	1	1	1
False Negative	1	1	1

Table 4.4: Detection Result by LSTM Method

GRU method merges the input and forgets gates into a single update gate. Owing

to this simplified architecture, GRU was computationally more efficient than LSTM. For bidirectional LSTM, each sample is predicted by both the future and past samples. The training time was thus almost double, owing to the presence of both positive (forward states) and negative (backward states) processes. The training time for each epoch is recorded in the Table 4.5.

	LSTM	GRU	Bidirectional LSTM
Training Time (s)	869	716	1423

Table 4.5: Training Time of LSTM Methods

4.5.3 CNN-LSTM Method

Even though the LSTM method can detect more anomalies with fewer false alarms than the CNN method, the training time for LSTM was significantly longer than that of CNN as shown in Table 4.6. The difference may even get larger with a larger training dataset.

	CNN	LSTM
Training Time (s)	21	869

Table 4.6: Training Time of CNN Method vs. LSTM Method

The CNN-LSTM method combines the convolutional layer with the LSTM layers, where the features are extracted from the dataset with convolutional layers, and the temporal relationship is learned by the LSTM layers. The CNN-LSTM method is comprised of a Conv1D layer and three LSTM layers as shown in Figure 4.45.

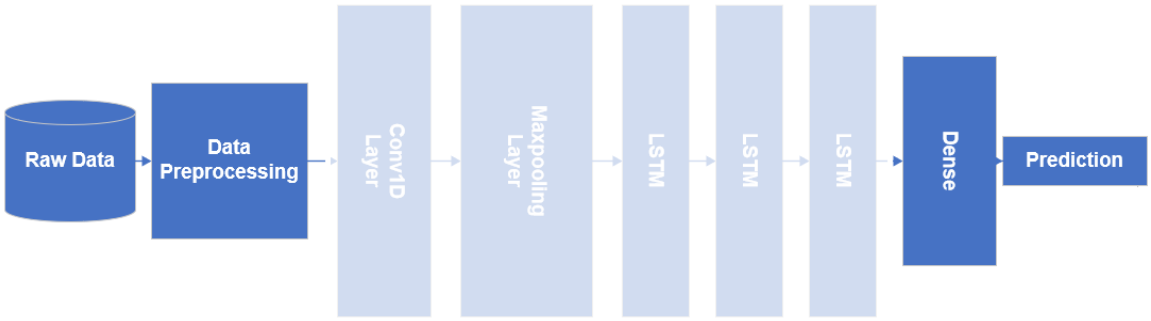

Figure 4.45: CNN-LSTM Network

```
model.add(Conv1D(filters=256, kernel_size=5, padding='same', activation=
    'relu', input_shape=(time_window_size - 1, layers['input'])))
model.add(MaxPooling1D(pool_size=4))

model.add(LSTM(units=layers['hidden1'], return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=layers['hidden2'], return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=layers['hidden3'], return_sequences=False))
model.add(Dropout(0.2))
```

Listing 4.12: CNN-LSTM Based Method

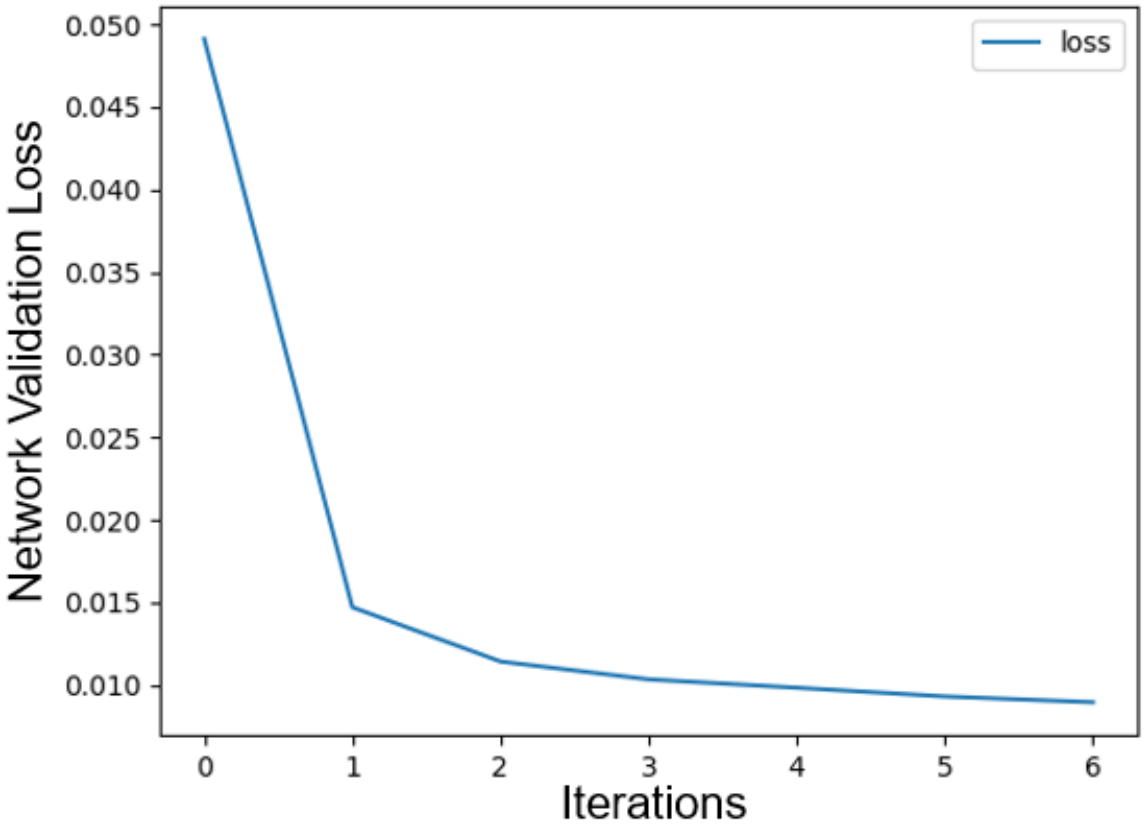

Figure 4.46: MSE Loss with Iterations

The CNN-LSTM method turned out to be a cost-effective anomaly detector. Table 4.7 lists the training time of each epoch and compares the CNN-LSTM and LSTM methods.

	CNN-LSTM	LSTM
Training Time (s)	82	869

Table 4.7: Training Time of LSTM Method vs. CNN-LSTM Method

CNN-LSTM method managed to detect nine out of eleven anomalies with two false alarms. The detection result were better than the CNN method with more anomalies being detected and fewer false alarms. A sample comparison is shown in

Figure 4.47.

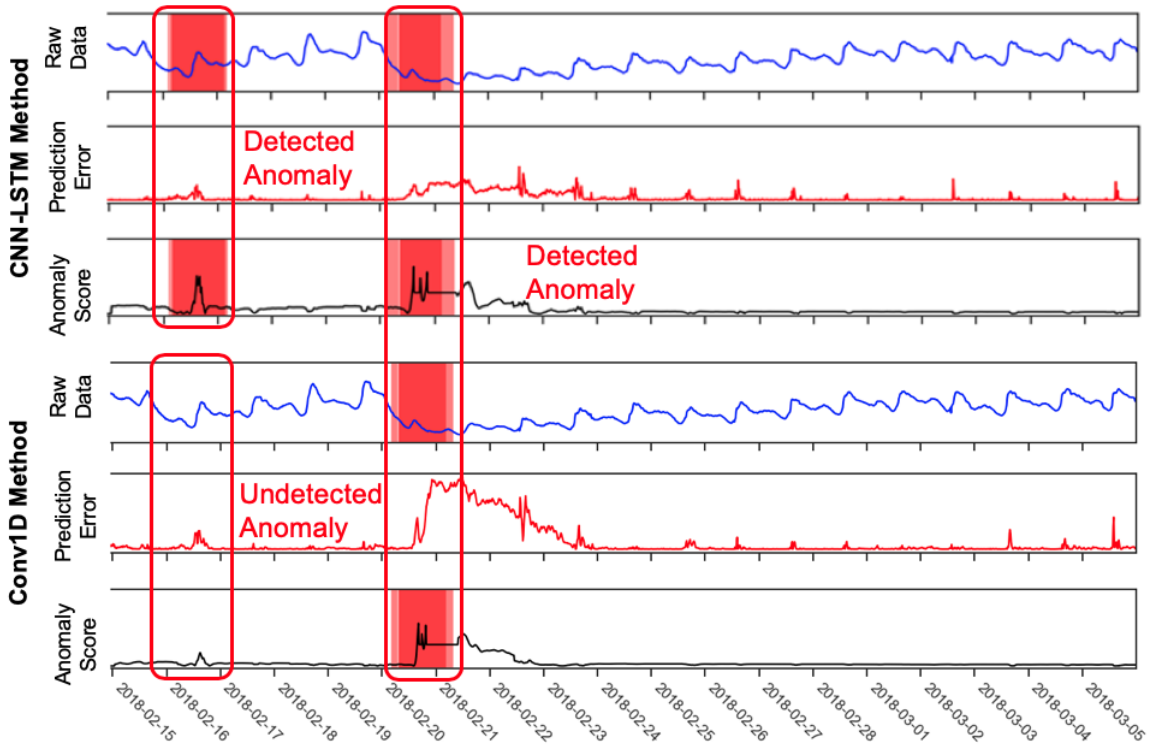

Figure 4.47: Sample Result for February 2018

	CNN-LSTM Method
True Positive	9
False Positive	2
False Negative	2

Table 4.8: Detection Result by CNN-LSTM Method

These results indicate that in most cases the LSTM method should be the first choice as it detected most anomalies with the fewest number of false alarms. If computing resources are limited, CNN-LSTM method can also be used.

4.6 Anomaly Detection on Weather Datasets

The same LSTM based anomaly detector was applied to the flow and temperature datasets in an attempt to detect the two real faulty ammonia anomalies. Only the LSTM-based detector was used as it provided the best results. For the flow data, the training data contained three weeks of normal readings selected from December 2017 to January 2018 as shown in Figure 4.48. Training was finished within nine epochs as shown in Figure 4.49.

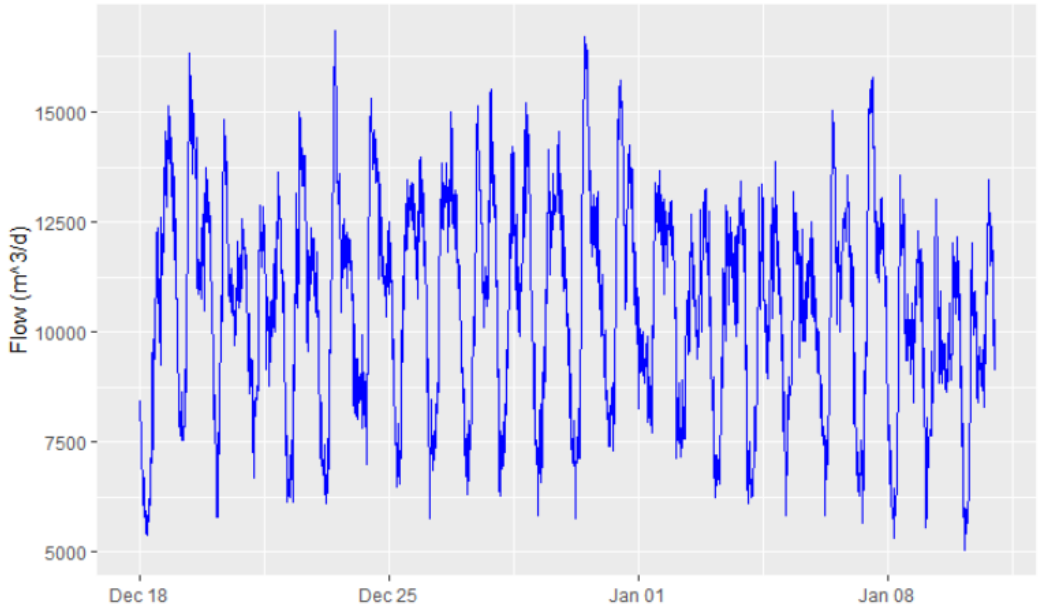

Figure 4.48: Training Data for Flow

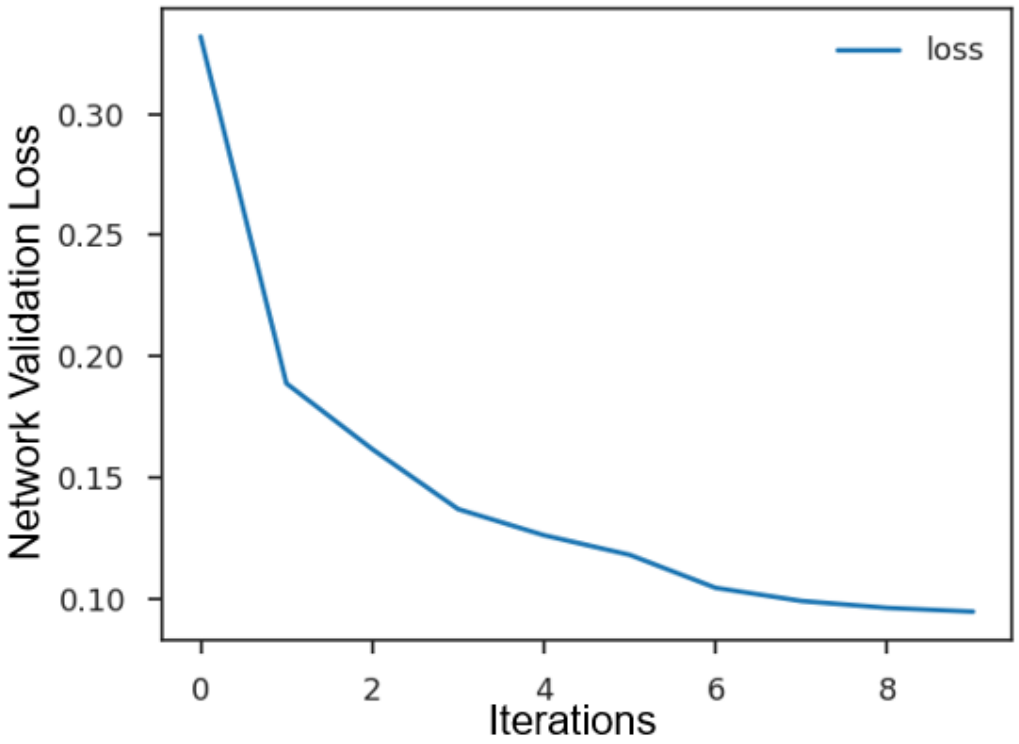

Figure 4.49: MSE Loss with Iterations

The flow data between January 2018 and March 2018 contained five anomalies, which were caused by five rain events. Comparing the ammonia and flow results, when both results contain anomalies, might be used to detect operational anomalies. As shown in Figure 4.50, the rain event in late March 2018 was successfully detected with the flow data. Filtering these two periods from the ammonia result is possible. However, eliminating anomalies with flow data might not always be possible. Flow data was provided by the Dundas WWTP, but it might not always be available at other locations and it cannot always be acquired easily in all possible locations.

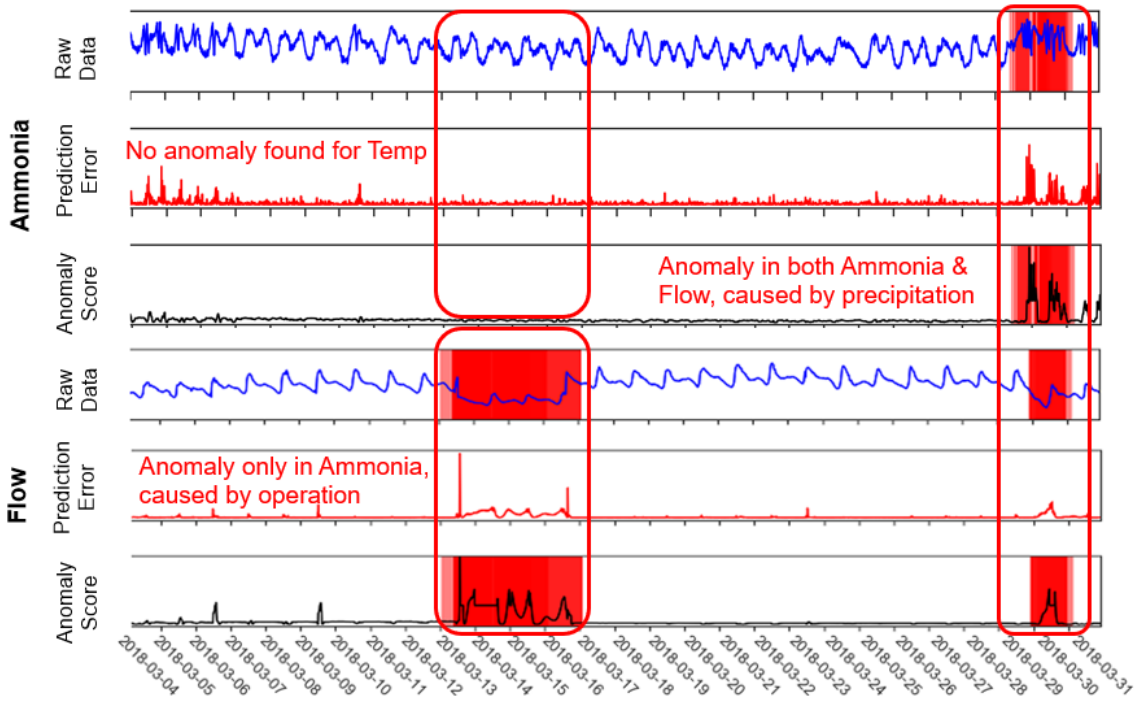

Figure 4.50: Detection Results of Flow in March 2018

	LSTM Anomaly Detector
True Positive	5
False Positive	1
False Negative	0

Table 4.9: Detection Result for Flow Data

As an alternative, temperature data was used because the temperature sensor is attached to the ammonia sensor. Similar to the ammonia and flow data, the longest section with normal readings was selected as the training set for temperature. The temperature training set contained readings from March 2018 as shown in Figure 4.51. Training was finished within nine epochs as shown in Figure 4.51.

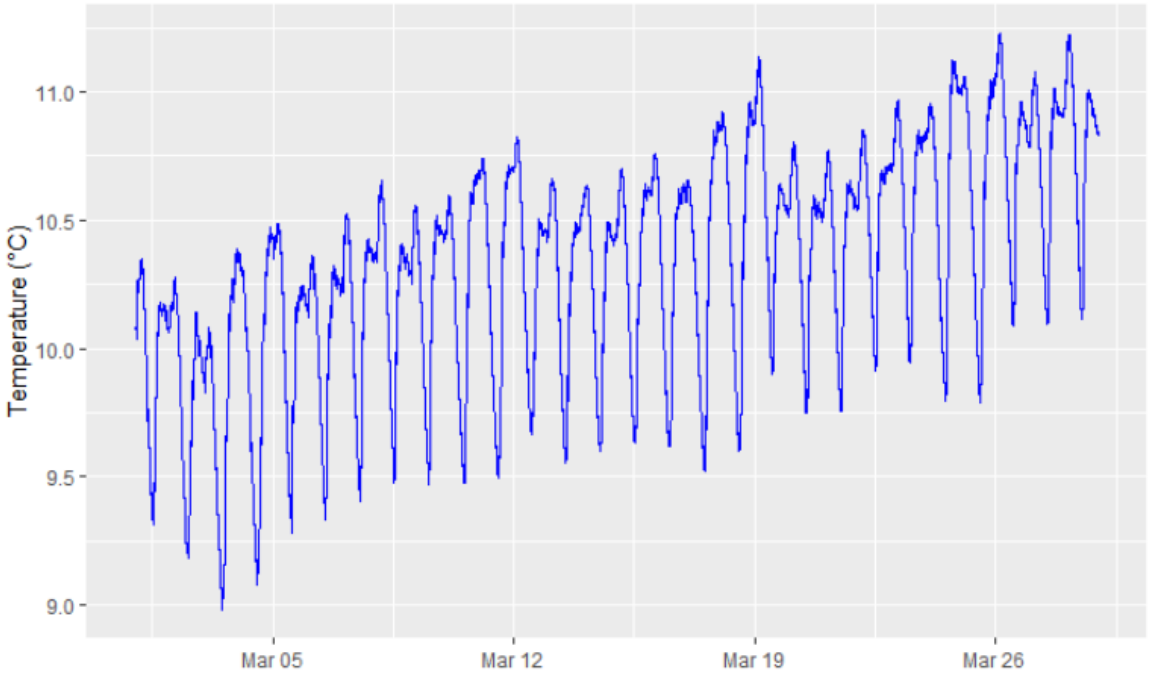

Figure 4.51: Training Data for Temperature

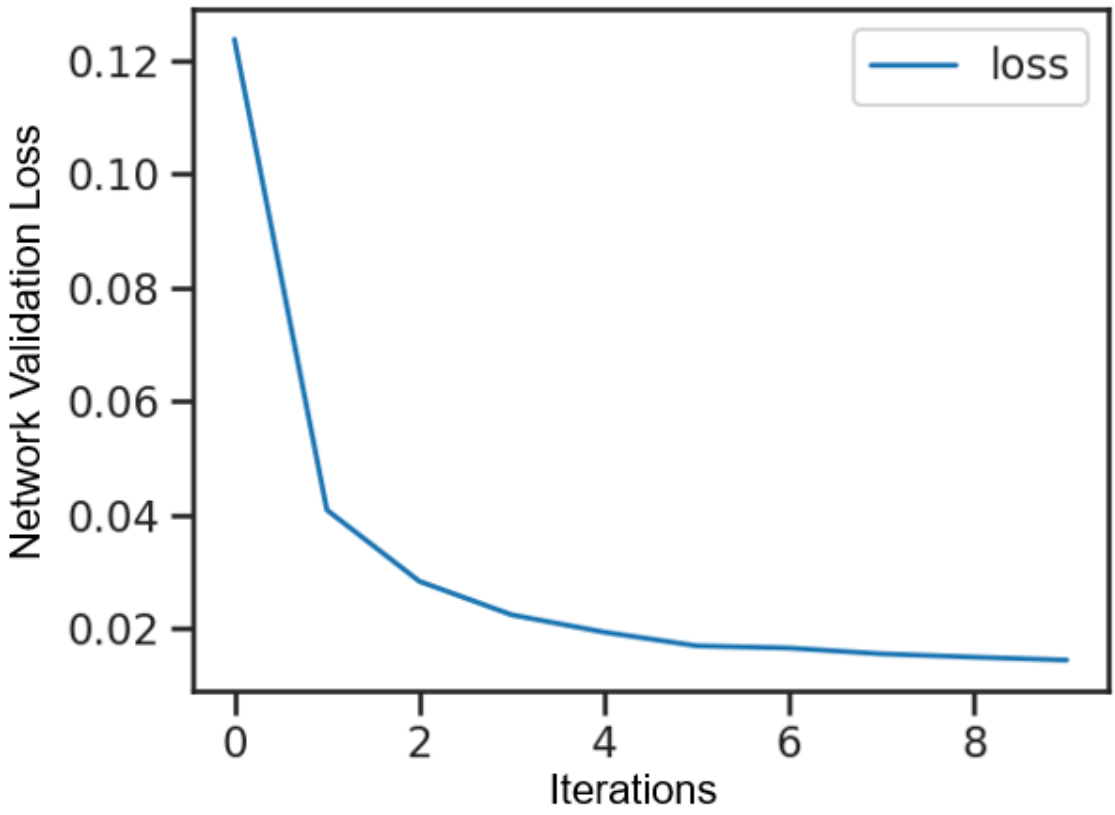

Figure 4.52: MSE Loss with Iterations

There are ten anomalies in the temperature dataset. Precipitation in January was detected in the temperature dataset as an anomaly as shown in Figure 4.53. However, the detection for temperature seems to be less accurate compared to the ammonia and flow cases as judged by the high number of false positive and false negative results.

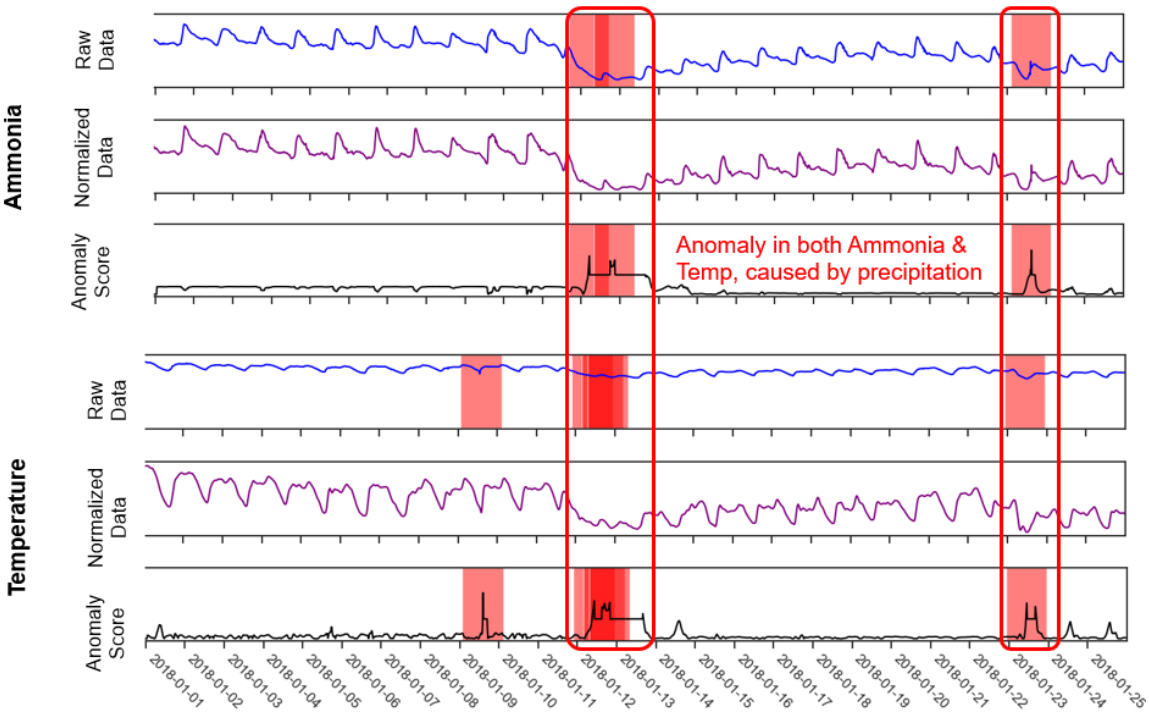

Figure 4.53: Detection Result of Temperature in January 2018

	LSTM Anomaly Detector
True Positive	6
False Positive	2
False Negative	4

Table 4.10: Detection Result For Temperature Data

The detection results for the temperature dataset show that six out of ten anomalies were detected. However, after further analysis, it was found that temperature alone could not be used to identify the precipitation-related ammonia anomalies. In the summertime such as June (Figure 4.54), when the rain temperature is close to the wastewater temperature, precipitation does not affect the recorded temperature so no temperature anomalies were detected. As a result, the detected anomaly (normal rain event) in Figure 4.54 can be misinterpreted as being a real anomaly as it is

only detected in the ammonia data and not in the temperature data.

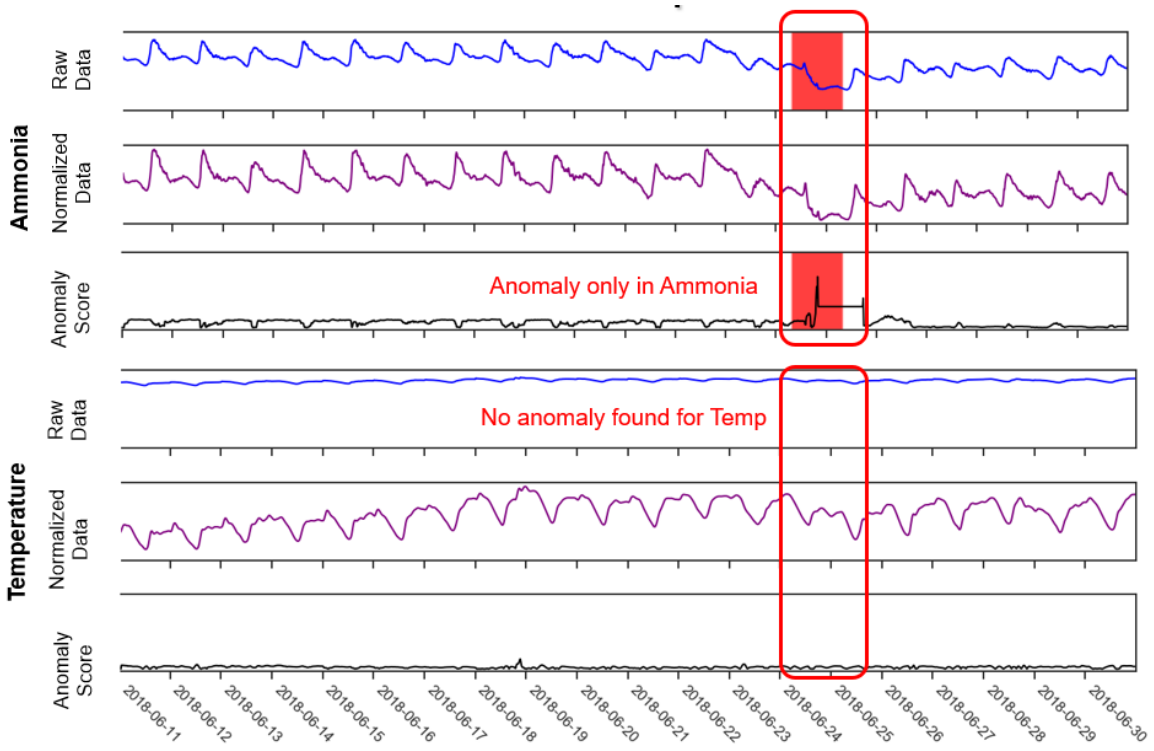

Figure 4.54: Detection Result of Temperature in June 2018

The opposite situation can also happen, that is, a real anomaly may be misinterpreted as a normal rain event. Figure 4.55 shows a case where both ammonia and temperature showed anomalies. Both anomalies were caused by the real operation problem, i.e. the sensor was exposed to the air. In this analysis, it was assumed that temperature anomalies, were only caused by precipitation, but in this case both anomalies were caused by the a real problem. In cases like this, when both sensors were not working properly, eliminating ammonia anomalies with temperature anomalies would be incorrect.

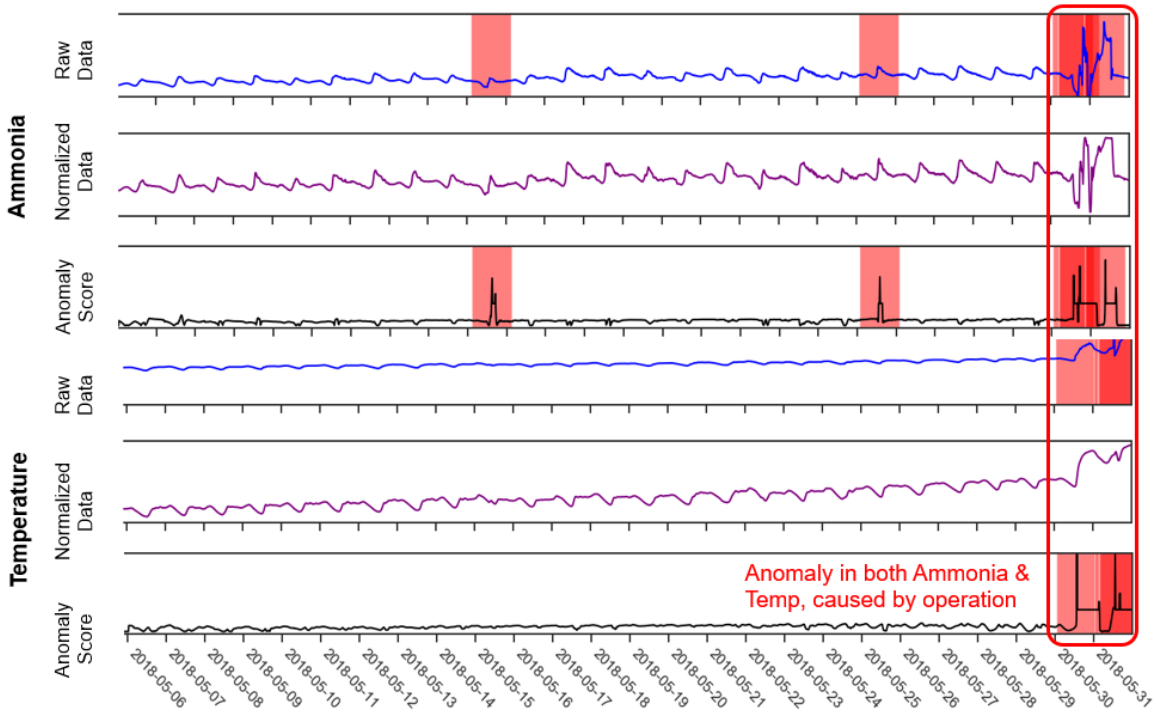

Figure 4.55: Detection Result of Temperature in May 2018

Chapter 5

Conclusions and Future Work

In this thesis, a data-driven approach was proposed to detect anomalous data in a real-world ammonia concentration dataset, using a learning based method.

The thesis started by studying the structure of neural networks. Next, an LSTM based method was applied to the ammonia dataset. The learning-based LSTM method outperformed the two rule-based algorithms by detecting ten out of eleven anomalies with only one false positive. In addition, convolutional layers were examined and the use of CNN combined with LSTM proved to be computationally efficient which could be important consideration when computing resources are limited. Finally, the relationship between ammonia and weather were explored. Temperature could only act as a suitable alternative for flow in winter time. Better alternative datasets should be studied to eliminate the impact of precipitation.

One potential limitation of this project is the dataset that was used. In this case the learning-based methods were tested on a newly collected and well-documented dataset dataset. Typical real-world datasets often lack detailed field notes related to sensor conditions. The sensors in this project were carefully maintained and did

not suffer from significant drift. The number of anomalies was thus limited and most of them were caused by weather. This might not be the case for other applications, where the sensors are not as carefully maintained. As a result, more tests of this learning-based method on datasets collected under different conditions should be conducted.

In addition, the LSTM detectors only made use of the daily pattern for anomaly detection. However, ammonia dataset also contains repeating patterns on different times scales. For example, concentration behaves slightly differently on the weekend from the weekdays, because some industrial facilities do not operate and people do not work in the weekend. With more collected dataset available in the future, monthly or even yearly patterns can also be studied. These are the factors to be considered in the future research.

Wavelet analysis is a signal processing technique which decomposes signals into multiple components and allows local features of the signals to be examined with details at the features' scale [Alarcon-Aquino and Barria, 2001]. Wavelets were widely used in signal analysis for decades [Zhengjia et al., 1996] and they allow features to be extracted over a broad range of time scales. Wavelet transforms have been successfully used as detection methods to identify anomalies in network traffic data [Kwon et al., 2006]. More recently, wavelets are combined with neural networks to detect anomalies in a time series dataset collected by health monitoring applications, where the detection method combines wavelet analysis and neural networks in a sequential manner [Kanarachos et al., 2015]. As a result, in the future research the LSTM detector might be examined with the filtered data instead of the raw data to combine the strengths of different algorithms.

Bibliography

Vicente Alarcon-Aquino and Javier A Barria. Anomaly detection in communication networks using wavelets. *IEE Proceedings-Communications*, 148(6):355–362, 2001.

Janelcy Alferes, Sovanna Tik, John Copp, and Peter A. Vanrolleghem. Advanced monitoring of water systems using in situ measurement stations: data validation and fault detection. *Water Science and Technology*, 68(5):1022–1030, Sep 2013. doi: 10.2166/wst.2013.302. URL https://doi.org/10.2166%2Fwst.2013.302.

Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, May 2013. doi: 10.1109/icassp.2013.6639349. URL https://doi.org/10.1109%2Ficassp.2013.6639349.

Léon Bottou. Stochastic gradient descent tricks. In *Lecture Notes in Computer Science*, pages 421–436. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-35289-8_25. URL https://doi.org/10.1007%2F978-3-642-35289-8_25.

Climate Change Canada. Treatment processes for removal of ammonia from municipal wastewater, Jul 2013. URL https://www.canada.ca/en/

environment-climate-change/services/wastewater/resource-documents/treatment-processes-removal-ammonia-municipal.html.

Raghavendra Chalapathy, Aditya Krishna Menon, and Sanjay Chawla. Robust, deep and inductive anomaly detection. In *Machine Learning and Knowledge Discovery in Databases*, pages 36–51. Springer International Publishing, 2017. doi: 10.1007/978-3-319-71249-9_3. URL https://doi.org/10.1007%2F978-3-319-71249-9_3.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009. URL https://dl.acm.org/citation.cfm?id=1541882.

Sucheta Chauhan and Lovekesh Vig. Anomaly detection in ECG time signals via deep long short-term memory networks. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, Oct 2015. doi: 10.1109/dsaa.2015.7344872. URL https://doi.org/10.1109%2Fdsaa.2015.7344872.

Xue-Wen Chen and Xiaotong Lin. Big data deep learning: challenges and perspectives. *IEEE*, 2:514–525, 2014. URL https://ieeexplore.ieee.org/document/6817512.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2014. doi: 10.3115/v1/d14-1179. URL https://doi.org/10.3115%2Fv1%2Fd14-1179.

Robert B Cleveland, William S Cleveland, Jean E McRae, and Irma Terpenning. Stl: A seasonal-trend decomposition. *Journal of Official Statistics*, 6(1):3–73, 1990.

Niklas Donges. Gradient descent in a nutshell, Mar 2018. URL `https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0?gi=e5e22712ec0e`.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12 (Jul):2121–2159, 2011. URL `https://dl.acm.org/citation.cfm?id=2021068`.

Ray J Frank, Neil Davey, and Stephen P Hunt. Time series prediction and neural networks. *Journal of Intelligent and Robotic Systems*, 31(1-3):91–103, 2001. URL `https://link.springer.com/article/10.1023/A:1012074215150`.

Alex Graves, Abdel rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, May 2013. doi: 10.1109/icassp.2013.6638947. URL `https://doi.org/10.1109%2Ficassp.2013.6638947`.

Henri Haimi, Michela Mulas, Francesco Corona, Stefano Marsili-Libelli, Paula Lindell, Mari Heinonen, and Riku Vahala. Adaptive data-derived anomaly detection in the activated sludge process of a large-scale wastewater treatment plant. *Engineering Applications of Artificial Intelligence*, 52:65–80, Jun 2016. doi: 10.1016/j.engappai.2016.02.003. URL `https://doi.org/10.1016%2Fj.engappai.2016.02.003`.

Seng Hansun. A new approach of moving average method in time series analysis. In *2013 Conference on New Media Studies (CoNMedia)*. IEEE, Nov 2013. doi:

10.1109/conmedia.2013.6708545. URL https://doi.org/10.1109%2Fconmedia.2013.6708545.

Fouzi Harrou, Abdelkader Dairi, Ying Sun, and Mohamed Senouci. Statistical monitoring of a wastewater treatment plant: A case study. *Journal of Environmental Management*, 223:807–814, 2018. URL https://www.sciencedirect.com/science/article/pii/S0301479718307394.

Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29 (6):82–97, nov 2012. doi: 10.1109/msp.2012.2205597. URL https://doi.org/10.1109%2Fmsp.2012.2205597.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06(02):107–116, Apr 1998. doi: 10.1142/s0218488598000094. URL https://doi.org/10.1142%2Fs0218488598000094.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. URL https://dl.acm.org/citation.cfm?id=1246450.

Turker Ince, Serkan Kiranyaz, Levent Eren, Murat Askar, and Moncef Gabbouj. Real-time motor fault detection by 1-d convolutional neural networks. *IEEE Transactions on Industrial Electronics*, 63(11):7067–7075, Nov 2016. doi: 10.1109/tie.2016.2582729. URL https://doi.org/10.1109%2Ftie.2016.2582729.

Jun Inoue, Yoriyuki Yamagata, Yuqi Chen, Christopher M. Poskitt, and Jun Sun. Anomaly detection for a water treatment system using unsupervised machine learning. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, Nov 2017. doi: 10.1109/icdmw.2017.149. URL `https://doi.org/10.1109%2Ficdmw.2017.149`.

Alexey Grigorevich Ivakhnenko. Polynomial theory of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics*, (4):364–378, 1971. URL `https://ieeexplore.ieee.org/document/4308320`.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350, 2015. URL `https://dl.acm.org/citation.cfm?id=3045367`.

Stratis Kanarachos, Jino Mathew, Alexander Chroneos, and Michael Fitzpatrick. Anomaly detection in time series data using a combination of wavelets, neural networks and hilbert transform. In *2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pages 1–6. IEEE, 2015.

Serkan Kiranyaz, Turker Ince, and Moncef Gabbouj. Real-time patient-specific ECG classification by 1-d convolutional neural networks. *IEEE Transactions on Biomedical Engineering*, 63(3):664–675, Mar 2016. doi: 10.1109/tbme.2015.2468589. URL `https://doi.org/10.1109%2Ftbme.2015.2468589`.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

Donghwoon Kwon, Hyunjoo Kim, Jinoh Kim, Sang C. Suh, Ikkyun Kim, and Kuinam J. Kim. A survey of deep learning-based network anomaly detection. *Cluster Computing*, Sep 2017. doi: 10.1007/s10586-017-1117-8. URL https://doi.org/10.1007%2Fs10586-017-1117-8.

DW Kwon, K Ko, Marina Vannucci, AL Narasimha Reddy, and S Kim. Wavelet methods for the detection of anomalies and their application to network traffic analysis. *Quality and Reliability Engineering International*, 22(8):953–969, 2006.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791. URL https://doi.org/10.1109%2F5.726791.

Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436444, 2015. doi: 10.1038/nature14539.

Yongkun Liang, Xie Quan, Jingwen Chen, Jong Shik Chung, Joon Y. Sung, Shuo Chen, Daming Xue, and Yazhi Zhao. Long-term results of ammonia removal and transformation by biofiltration. *Journal of Hazardous Materials*, 80(1-3):259–269, Dec 2000. doi: 10.1016/s0304-3894(00)00314-9. URL https://doi.org/10.1016%2Fs0304-3894%2800%2900314-9.

Yiqi Liu, Yongping Pan, Zonghai Sun, and Daoping Huang. Statistical monitoring of wastewater treatment plants using variational bayesian PCA. *Industrial & Engineering Chemistry Research*, 53(8):3272–3282, Feb 2014. doi: 10.1021/ie403788v. URL https://doi.org/10.1021%2Fie403788v.

Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short

term memory networks for anomaly detection in time series. In *Proceedings*, page 89. Presses universitaires de Louvain, 2015.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010. URL https://dl.acm.org/citation.cfm?id=3104425.

Masafumi Nakano, Akihiko Takahashi, and Soichiro Takahashi. Generalized exponential moving average (EMA) model with particle filtering and anomaly detection. *Expert Systems with Applications*, 73:187–200, May 2017. doi: 10.1016/j.eswa.2016.12.034. URL https://doi.org/10.1016%2Fj.eswa.2016.12.034.

Netflix. Netflix/surus, Mar 2015. URL https://github.com/Netflix/Surus/tree/master/resources/R/RAD.

Christopher Olah. Understanding lstm networks, 2015. URL http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2012. URL https://pdfs.semanticscholar.org/728d/814b92a9d2c6118159bb7d9a4b3dc5eeaaeb.pdf.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013. URL https://dl.acm.org/citation.cfm?id=3043083.

Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):

3448–3470, Aug 2007. doi: 10.1016/j.comnet.2007.02.001. URL https://doi.org/10.1016%2Fj.comnet.2007.02.001.

Shailna Patidar. Machine learning vs. deep learning - dzone ai, Sep 2018. URL https://dzone.com/articles/comparison-between-deep-learning-vs-machine-learni.

Lutz Prechelt. Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11(4):761–767, Jun 1998. doi: 10.1016/s0893-6080(98)00010-0. URL https://doi.org/10.1016%2Fs0893-6080%2898%2900010-0.

S. Joe Qin. Data-driven fault detection and diagnosis for complex industrial processes. *IFAC Proceedings Volumes*, 42(8):1115–1125, 2009. doi: 10.3182/20090630-4-es-2003.00184. URL https://doi.org/10.3182%2F20090630-4-es-2003.00184.

M. Rabiet, F. Brissaud, J.L. Seidel, S. Pistre, and F. Elbaz-Poulichet. Positive gadolinium anomalies in wastewater treatment plant effluents and aquatic environment in the hérault watershed (south france). *Chemosphere*, 75(8):1057–1064, May 2009. doi: 10.1016/j.chemosphere.2009.01.036. URL https://doi.org/10.1016%2Fj.chemosphere.2009.01.036.

Sutharshan Rajasegarar, Christopher Leckie, and Marimuthu Palaniswami. Anomaly detection in wireless sensor networks. *IEEE Wireless Communications*, 15(4):34–40, 2008.

Bernard Rosner. Percentage points for a generalized ESD many-outlier procedure.

Technometrics, 25(2):165, May 1983. doi: 10.2307/1268549. URL https://doi.org/10.2307%2F1268549.

G Ruiz, D Jeison, and R Chamy. Nitrification with high nitrite accumulation for the treatment of wastewater with high ammonia concentration. *Water Research*, 37(6):1371–1377, Mar 2003. doi: 10.1016/s0043-1354(02)00475-x. URL https://doi.org/10.1016%2Fs0043-1354%2802%2900475-x.

A. Sanchez-Fernandez, M.J. Fuente, and G.I. Sainz-Palmero. Fault detection in wastewater treatment plants using distributed PCA methods. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, Sep 2015. doi: 10.1109/etfa.2015.7301504. URL https://doi.org/10.1109%2Fetfa.2015.7301504.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. URL https://ieeexplore.ieee.org/abstract/document/650093.

Luai Shalabi and Zyad Shaaban. Normalization as a preprocessing engine for data mining and the approach of preference matrix. In *2006 International Conference on Dependability of Computer Systems*. IEEE, 2006. doi: 10.1109/depcos-relcomex.2006.38. URL https://doi.org/10.1109%2Fdepcos-relcomex.2006.38.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz. Anomaly detection in automobile control network data with long short-term memory networks. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, Oct 2016. doi: 10.1109/dsaa.2016.20. URL https://doi.org/10.1109%2Fdsaa.2016.20.

Matthew Taylor. numenta/nupic: 1.0.5, June 2018. URL https://doi.org/10.5281/zenodo.1257382.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2):26–31, 2012. URL https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Twitter. Anomaly detection library by twitter. *GitHub*, Aug 2015. URL https://github.com/twitter/AnomalyDetection.

Owen Vallis, Jordan Hochenbaum, and Arun Kejariwal. A novel technique for long-term anomaly detection in the cloud. In *HotCloud*, 2014. URL https://dl.acm.org/citation.cfm?id=2696550.

Venkat Venkatasubramanian, Raghunathan Rengaswamy, Surya N Kavuri, and Kewen Yin. A review of process fault detection and diagnosis: Part iii: Process history based methods. *Computers & Chemical Engineering*, 27(3):327–346, 2003.

R Vinayakumar, K P Soman, and Prabaharan Poornachandran. Applying convolutional neural network for network intrusion detection. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*.

IEEE, Sep 2017. doi: 10.1109/icacci.2017.8126009. URL https://doi.org/10.1109%2Ficacci.2017.8126009.

P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. doi: 10.1109/5.58337. URL https://doi.org/10.1109%2F5.58337.

D Randall Wilson and Tony R Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003. doi: 10.1016/s0893-6080(03)00138-2. URL https://dl.acm.org/citation.cfm?id=965272.

He Zhengjia, Zhao Jiyuan, He Yibin, and Meng Qingfeng. Wavelet transform and multiresolution signal decomposition for machinery monitoring and diagnosis. In *Proceedings of the IEEE International Conference on Industrial Technology (ICIT'96)*, pages 724–727. IEEE, 1996.