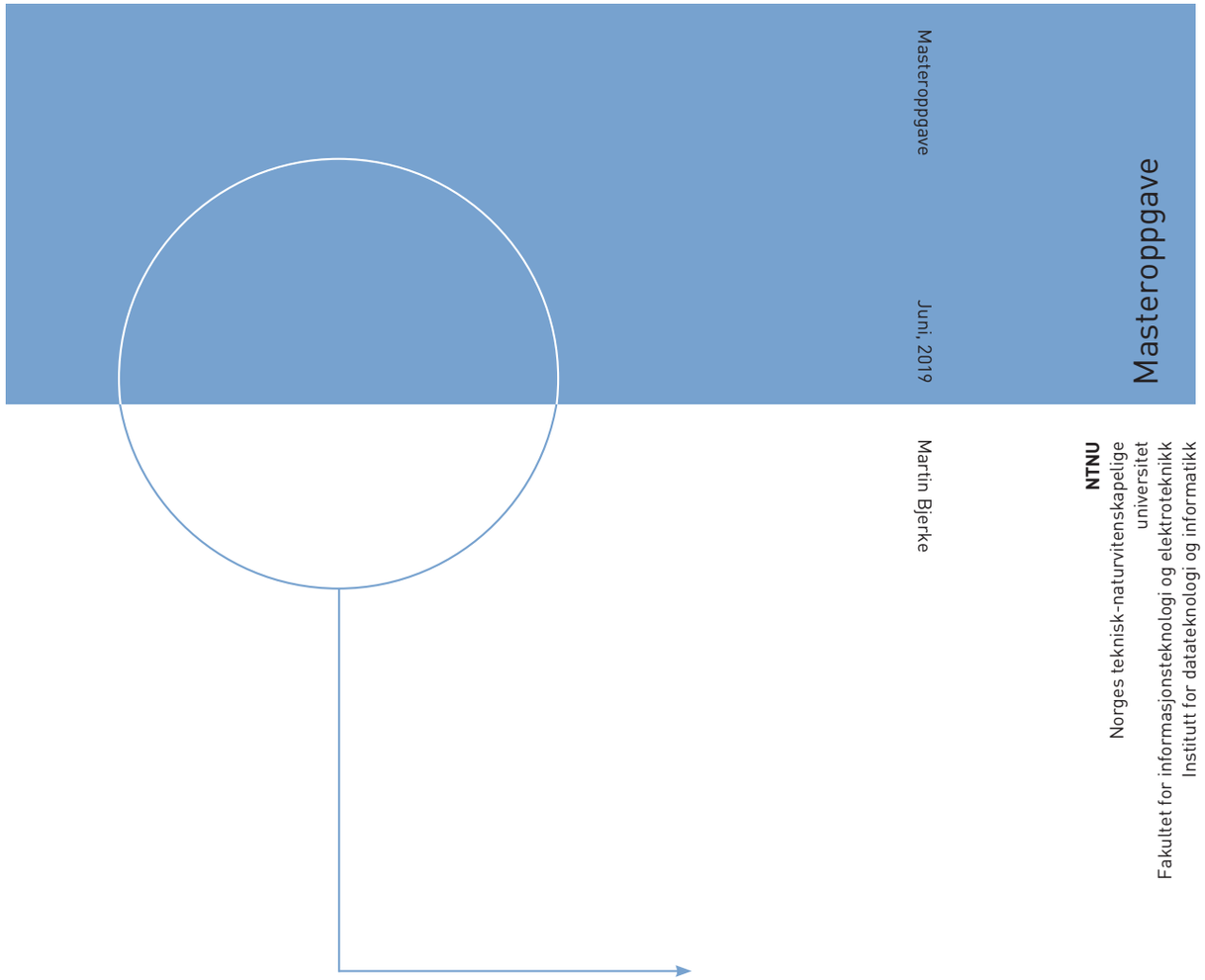


Masteroppgave

Juni, 2019

Martin Bjerke

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Masteroppgave

Martin Bjerke

Leak Detection in Water Distribution Networks using Gated Recurrent Neural Networks

Juni 2019

NTNU
Kunnskap for en bedre verden

NTNU
Kunnskap for en bedre verden

NTNU
Kunnskap for en bedre verden

Leak Detection in Water Distribution Networks using Gated Recurrent Neural Networks

Martin Bjerke

Master i Informatikk: Kunstig Intelligens
Innlevert: Juni 2019
Hovedveileder: Agnar Aamodt
Medveileder: Mari Hugaas

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknologi og informatikk

Preface

This Master's thesis was written as part of the study program Master of Science, Informatics: Artificial Intelligence at the Norwegian University of Science and Technology(NTNU) and was carried out during the fall of 2018 and spring of 2019. The topic of this thesis is based on a proof of concept project that was discussed, but not given, to me during my summer internship at Norkart in 2017. Norkart is a software provider for many of Norways municipalities and is therefore interested in new ways to apply technology to solve the municipalities issues.

Trondheim, June 9, 2019

Martin Bjerke

Acknowledgment

I would like to thank the following persons for their great help during my work on this thesis.

Mari Hugaas, Arne-Ronny Tøstibakken and Agnar Aamodt for supervising my work on this thesis and setting aside time for supervisor meetings on a regular basis.

This thesis was carried out in cooperation with Norkart and I would like to thank them for their help during the work on this thesis. Specifically I would like to thank Mari Hugaas for her help as supervisor and Arne-Ronny Tøstibakken for presenting the topic and problem description. Arne-Ronny has also help with supervising us and helped by answering domain specific questions concerning water distribution in Norway.

I would also like to thank Rita Ugarelli for her help searching for relevant work in the domain of Hydroinformatics, and answering domain relevant questions, without being directly connected to this thesis.

Lastly I want to thank my fellow students at Alke for keeping me company and helping me with discussing various problems that I ran into during the work on this thesis.

M.B.

Summary

In this master thesis two different Gated Recurrent Neural Network architectures have been tested on the multivariate time series classification problem of leak detection in water distribution networks (WDN). Three different machine learning approaches were considered, Convolutional Nerual Networks, Case-Based Reasoning and Reccurent Neural Networks. A leak detection system has been implemented and tested which incorporates a Gated Recurrent Unit (GRU) and Long Short Term Memory (LSTM) with different hyper parameters and have been shown to be able to detect leaks within two different WDNs, Net1 and Hanoi. The leak detection system managed a true positive rate of 86%, a false positive rate of less than 0.1%, an accuracy of greater than 99% and a detection time of less than 1 hour on the Net1. To our knowledge, this is the first time Gated RNNs have been applied for leak detection in water networks.

Sammendrag

I denne masteroppgaven har to forskjellige Gated Recurrent Neural Network-arkitekturer blitt testet på det multivariable tidsserie klassifikasjonsproblemet, lekkasjedeteksjon i vannfordelingsnettverk. Tre ulike maskininnlæringsmetoder ble vurdert, Convolutional Nerual Networks, Case-Based Reasoning og Recurent Neural Networks. Et system er implementert og testet basert på Gated Recurrent Unit (GRU) og Long Short Term Memory (LSTM) med forskjellige hyperparametere og har vist seg å kunne oppdage lekkasjer innenfor to forskjellige vannett, Net1 og Hanoi. Systemet klarte en true positiv rate på 86%, en false positiv hastighet på mindre enn 0,1%, en nøyaktighet på over 99% og en deteksjonstid på mindre enn 1 time på Net1. Så vidt vi vet, er dette første gang Gated RNN har blitt brukt for lekkasjedeteksjon i vannett.

Contents

List of Tables

List of Figures

Abbreviations

ANN Artificial Neural Network 6, 8, 15, 28–33, 47, 59

CBR Case-based Reasoning 13, 59

CNN Convolutional Neural Network 6, 13, 15, 18, 59

DMA District Metering Area 8–10, 27, 32

DT Detection Time 46, 47

EC Exclusion Criteria 22, 24

FPR False Positive Rate 46, 47

GRU Gated Recurrent Unit 20, 21, 39

IC Inclusion Criteria 22, 24

LSTM Long-Short Term Memory 20, 39

ML Machine-learning 6, 7, 13, 15, 24, 36, 55, 59

MLP Multi Layer Perceptron 15, 33, 55, 59

NFA Night-Flow Analysis 10, 11

NN Neural Network 6, 33, 55

QC Quality Criteria 22, 24, 25

RNN Recurrent Neural Network 6, 7, 13, 18, 30, 33, 36, 57, 59

RQ Research Question 6, 25

SLR Structured Literature Review 7, 22–24, 33, 55, 59

SVM Support Vector Machine 6, 28, 59

SVR Support Vector Regression 28, 29

SW Search Word 22, 24

TPR True Positive Rate 47

WDN Water Distribution Network 2, 5–13, 17, 18, 21, 25–31, 33–36, 38–40, 43, 45, 47, 53, 55–57, 59, 68, 72

Chapter 1: Introduction

1.1 Background and motivation

In 2016 the United Nations put clean water and sanitation as one of their sustainable development goals to be completed by 2030(United Nations, a). To reach this goal the UN set 6 sub goals, where one was: *"By 2030, substantially increase water-use efficiency across all sectors and ensure sustainable withdrawals and supply of freshwater to address water scarcity and substantially reduce the number of people suffering from water scarcity"*(United Nations, b). In 2011 it was predicted that the global demand for water would be 40% greater than the supply by 2030(Mounce et al., 2013), meaning the current supply must increase to reach this goal. One way to accomplish this is to reduce the current leakage in WDNs. Farley (2001) reported that in 1991, the water loss in developed countries, newly-industrialised countries, and developing countries was 8-24%, 15-24% and 25-45% of water supplied from treatment plants. Puust et al. (2010) reported more recently that this has improved in some countries like The Netherlands, where 3-7% of the water distributed is lost because of leaks, but there are still countries where over 50% of treated water is lost due to leaks. As of 2017, Norway loses 30% of the water distributed from water treatment plants due to leaks in WDNs(Statistisk Sentralbyrå, 2017).

Farley (2001) lists several problems caused by leaks, both for consumers, society, and water utility companies:

- Economical: Water utility companies use resources to treat the water that leaks from WDNs, so the cost for the companies are doubled if 50% of the water leaks out.
- Structural: Water leaking from pipe networks may erode the ground and cause great structural damage to surrounding infrastructure and structures, such as sinkholes. This is easily identified when pipes burst, although the damage is already done, smaller leaks having the same effect over time may not as easily be detected. However, if the leaks are detected, they can be fixed before to much damage is caused.
- Consumer inconvenience: Leaks and bursts may cause water shortage or lower pressures for consumers. Complaints about this are usually the first indication of a leak.
- Health risks: Pressure drops in pipes with holes may introduce contaminants in the pipes as the water surrounding the pipes are sucked in; this is known as *back siphonage*. There are great health risks connected to this as water pipes may lie in ditches together with sewer pipes which also leaks. Causing sewage to contaminate drinking water.

Traditionally, leak management has been done reactively by repairing leaks when the leak becomes visible above ground. In later years, more and more WDNs have been equipped with pressure and flow sensors leading to large amounts of data being collected. Along with making it possible to monitor WDNs online, it has also lead to problems interpreting this data because of its large volume, missing and erroneous data, and unpredictable pattern changes in the sensor readings(Mounce et al., 2013).

This increase in data has motivated researchers to apply machine-learning (ML) methods for leak detection, such as support vector machine (SVM)(Mashford et al., 2009; Mounce et al., 2010), artificial neural network (ANN)(Caputo and Pelagagge, 2002; Mounce et al., 2013), Kalman Filtering(Ye and Fenner, 2011), and several more (Wu and Liu, 2017; Chan et al., 2018). However, there have not been much research in using deeper neural networksand more complex neural networks (NNs), like convolutional neural networks (CNNs) or recurrent neural networks (RNNs), or other ML-methods like CBR for leak detection. Because the sensor data collected from WDNs has a temporal nature and current research using any type of ANNs have not been on par with other leak detection methods, it was tempting to test methods capable of taking advantage of this.

1.2 Goal

The goal of this thesis was to explore the use of MLs techniques to detect leaks in a WDN based on operational data and the possibility of improving current methods. To reach this goal, we had to identify which existing methods for leak detection were currently in use, which methods are being researched, and if any of these methods applied ML methods. To test the improved leak detection system, it was compared with results from current research using common domain specific measurements. To summarise, this thesis will answer the following research questions (RQs):

RQ1 What are the current methods used for leak detection, and which new methods are being researched?

RQ2 How can a ML method be implemented to improve the methods from RQ1?

RQ3 How does the solution to RQ2 compare to the methods found while answering RQ1?

1.2.1 Research Methodology

This thesis followed the design science research process(Peffers et al., 2006). The process is composed of 6 nominal steps;

1. *Problem identification and motivation;* Define research questions and justify why their answers are important.
2. *Objectives of a solution;* Define goals for a solution that answers the research questions.
3. *Design and development;* Create a solution.
4. *Demonstration;* Verify that the solution can reach the defined goals.

5. *Evaluation;* Evaluate and measure how the solution did in the demonstration.

6. *Communication;* Communicate the research questions, their answers, and the results of the solution.

Steps 1 and 2 were conducted as a structured literature review (SLR) process and would answer RQ1. The SLR would give an overview of which leak detection methods were currently in use, which methods were currently being researched and what their advantages and disadvantages were. As we did not have any previous knowledge of how leak detection in WDN were conducted, it also gave us insights into the problem domain and would identify possible ideas that could be applied in the development of a ML system for leak detection. The structured literature review was also used to find the best set of papers that covered leak detection while avoiding biases to specific authors. The approach and results are presented in Chapter 3, where Section 3.2 works as an extension of Chapter 2, and the process is discussed in Section 6.1.

After it was decided to create a leak detection system based on gated RNNs Steps 3, 4 and 5 were done over two iterations. The first iteration tested simple RNN architectures on a small WDN to identify problems to the approach and to verify possible solutions through two experiments. The second iteration tested the improved leak detection system on the Hanoi WDN, a popular WDN in the relevant literature. These iterations would answer RQ2, with the initial design and development step described in Chapter 4 and the implemented models are demonstrated, evaluated, and improved in Chapter 5.

The final evaluation of the leak detection system was presented in Section 5.6.3 and answers RQ3. The research approach and methodologies were evaluated and discussed in Chapter 6, before the answers to the research questions were summarised in Chapter 7 together with future work.

1.3 Context

This thesis was written over the duration of a year. The original intent of this research was to study how a case-based reasoning system could be implemented to classify faulty pipes ahead of time giving water utility companies a way to do proactive maintenance, instead of reactive. After an initial search and discussion with a domain expert, it was found to be difficult because of the lack of data and uncertainty of what data could be used for such an endeavor. This lead to a change in research into pipe burst prediction, then into the more general leak detection problem. Leak detection was chosen over burst prediction as the literature review in Section 3.2, showed that there are still problems with leak detection methods that can be improved and after discussion with Arne-Ronny Tøstibakken at Norkart it was clear that pipe bursts are quickly discovered and reported.

It was also intended to test the leak detection method presented in this paper using real-time operational data. However, the data could not be delivered in time from the data providers. Even though the leak detection system presented in this paper was only trained and tested on artificial data generated from historical data, the leak detection system was constructed to be applied to real operational data. The code developed for this thesis can be found on GitHubBjerke with a guide on how to run it.

Chapter 2: Background Theory

2.1 Hydroinformatics

Leak detection could be viewed as both an engineering and informatics problem and therefore lies in the field of Hydroinformatics. This was a relatively new field within informatics focusing on the application of information technologies to solve problems within hydraulics, hydrology, environmental engineering, and water-based systems as WDNs(Vojinovic and Abbott, 2017; K Price and Solomatine, 2019). Hydroinformatics was defined in 1991 by Abbott (1991) when research expanded from only the numerical modeling of water hydraulics to a more socio-technological field. Current research still focuses on the modeling of water, but now also includes generation of flood maps in cities, improved water system planning and management, sustainable drainage systems, and leak detection, which this thesis focused on. There has also been a change in methodology inside the field, from previously focusing on numerical methods to currently applying AI methodologies like ANN and genetic algorithms. These methods were used both alone to solve issues, or in combination with existing numerical methods to set their parameters.

Vojinovic and Abbott (2017) predicted that future research would focus on systems that could handle dynamic changes within a water system that were caused by climate change, i.e., changes that draw out over several years. Future systems should also handle shorter changes that span days like flooding, and even more extreme changes that occur over minutes.

2.1.1 Water Distribution Networks

Water distribution network (WDN) were pipe networks that distribute clean drinking water from water reservoirs out to the public. Figure 2.1 shows the Hanoi WDN from Vietnam, where sensors may be placed at any of the junctions or pipes to measure the flow and pressure. To supply the public with water, buildings were further connected to these junctions. In this work, it was assumed that the pressure sensors were in the junctions and the flow sensors were in the pipes. This was assumed because the data generation algorithm explained in Section 4.1.2 generated flow measurements for pipes and pressure measurement for junctions. In an optimal scenario, each of these junctions and pipes would be equipped with a sensor to be able to observe the network as a whole. However, this was not the case because of the costs of installing these sensors, motivating research into the optimal placement of these sensors within a WDN(Sarrate et al., 2014).

To monitor leaks, a WDN could be separated into several sub-networks called district metering areas (DMAs). The DMAs were connected through only a few pipes, and the pipes connect-

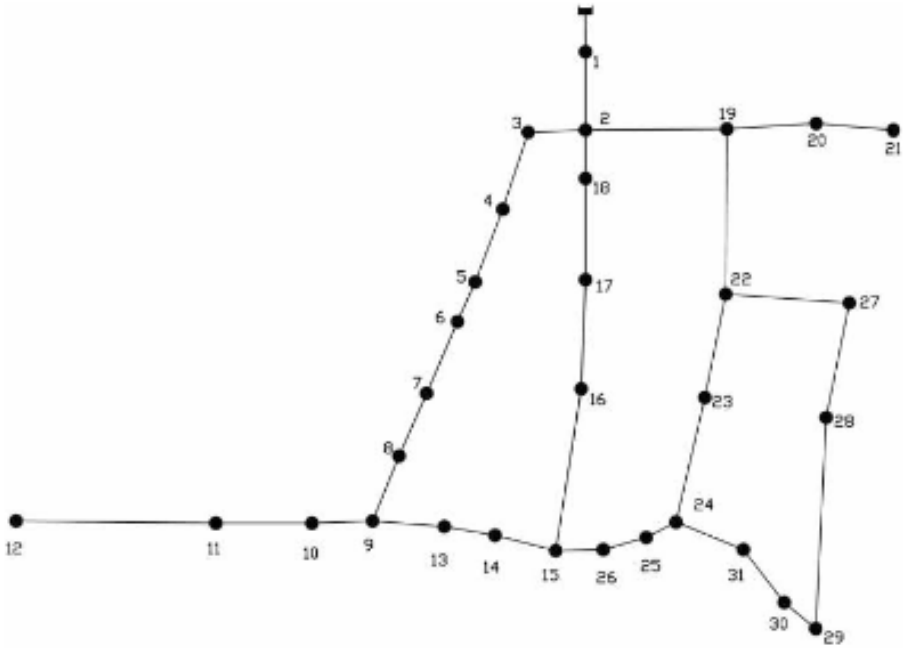


Figure 2.1: A depiction of the Hanoi WDN. The lines were pipes and nodes were junctions. The rectangle connected to node 1 was a water reservoir.

Source: Casillas Ponce et al. (2013)

ing them were equipped with pressure and flow sensors. A trivial example of this could be to split nodes 24, 23, 22, 27, 28, 29, 30 and 31 from the Hanoi WDN in Figure 2.1 into a single DMA, only needing sensors at the pipes between the nodes 25 and 24, and 22 and 19. A leak could be narrowed down to this part of the network with only measurements from 2 sensors given the leaks were large enough to affect one or both of the sensors.

Together with flow and pressure, water demand could also a measure in WDNs. Water demand was the total amount of water that was consumed by the consumers in the network, and all of the measures follow a diurnal pattern, as shown in Figure 2.2. This pattern was caused by a change in consumer demand throughout the day and week, with the lowest demand during night time, when customers usually sleep, and industry was not operating.

The water demand, and therefore also flow and pressure, follow a diurnal pattern. This pattern was caused by the change in customer demand based on the time of day and specific weekday. Weekdays usually follow the same pattern, while Saturday and Sunday have different patterns. The pattern also follows seasonal changes. Because of these patterns, a demand pattern could be approximated by decomposing it into three components and approximating each of the subcomponents; a weekly component, a yearly component and a random component to simulate the noise in the diurnal pattern(Vrachimis et al., 2018; Eliades and Polycarpou, 2012). This decomposition was used in the data generation framework described in Section 4.1.2, that was used to generate data for this thesis.

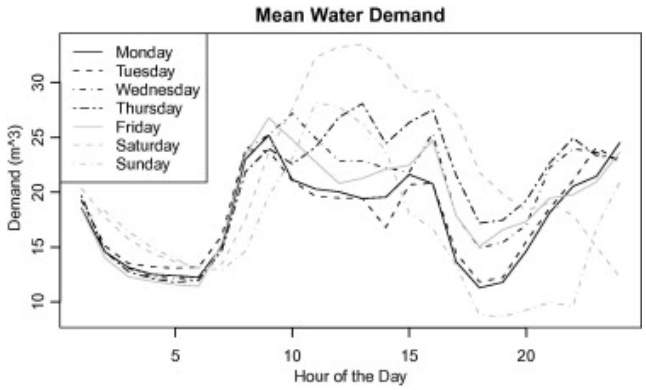


Figure 2.2: The diurnal pattern of water demand throughout a day. This was the mean water demand of a given day calculated over the 8 week data set of Herrera et al. (2010).

Source: Herrera et al. (2010)

2.2 Leakage detection

Leakages in WDNs may have different causes depending on the maintenance, original building of the infrastructure, placement and much more, but it was reported that from 3% to above 50% of the input into WDNs were lost due to leaks(Puust et al., 2010). This was not just an economic waste, through leakage of treated water, but also a health risk as low pressure in pipes may cause back siphonage.

Today, leaks were usually identified through minimum Night-Flow Analysis (NFA)(Eliades and Polycarpou, 2012) or by reports from customers to the water utility company. Reports of low pressure or discolored water were examples of customer reports that may indicate a leak. Minimum NFA requires flow data that was collected from sensors in a DMA.

In practice, flow sensors were placed at the connecting pipes between DMAs and pressure sensors were they are needed because they are cheaper and easier to installCasillas Ponce et al. (2013). It has been shown that pressure measurements were less sensitive to bursts if there was a limited number of pressure sensors in a network, and these settings flow measures have been shown to be more sensitive(Wu et al., 2018; Mounce et al., 2010). However, the same has not been stated for smaller or incipient leaks.

In minimum NFA, the flow measurements of previous nights were compared to identify any unexplainable changes in flow. The flow at night was used as this was the time of day where the customer demand was at its lowest, so the measurements were more stable and less likely to be disturbed by customers. This was either done manually, or an alarm was set to go off in a monitoring system if the minimum flow exceeds a chosen threshold. Measurement noise, customer demand trends, and seasonal differences may still cause leaks to go unnoticed by this approach as the method must take these into account. After a possible leak was identified, manual inspection was done to verify the leak and to decide how it was going to be handled.

Leaks could be separated into two types(Eliades and Polycarpou, 2012), incipient and abrupt. Incipient breaks were small breaks that gradually increased over time as the hole grew. Sudden breaks where the size of the hole remains close to constant were called abrupt breaks, and it was a standard assumption in the literature that leaks were of this type(Casillas Ponce et al.,

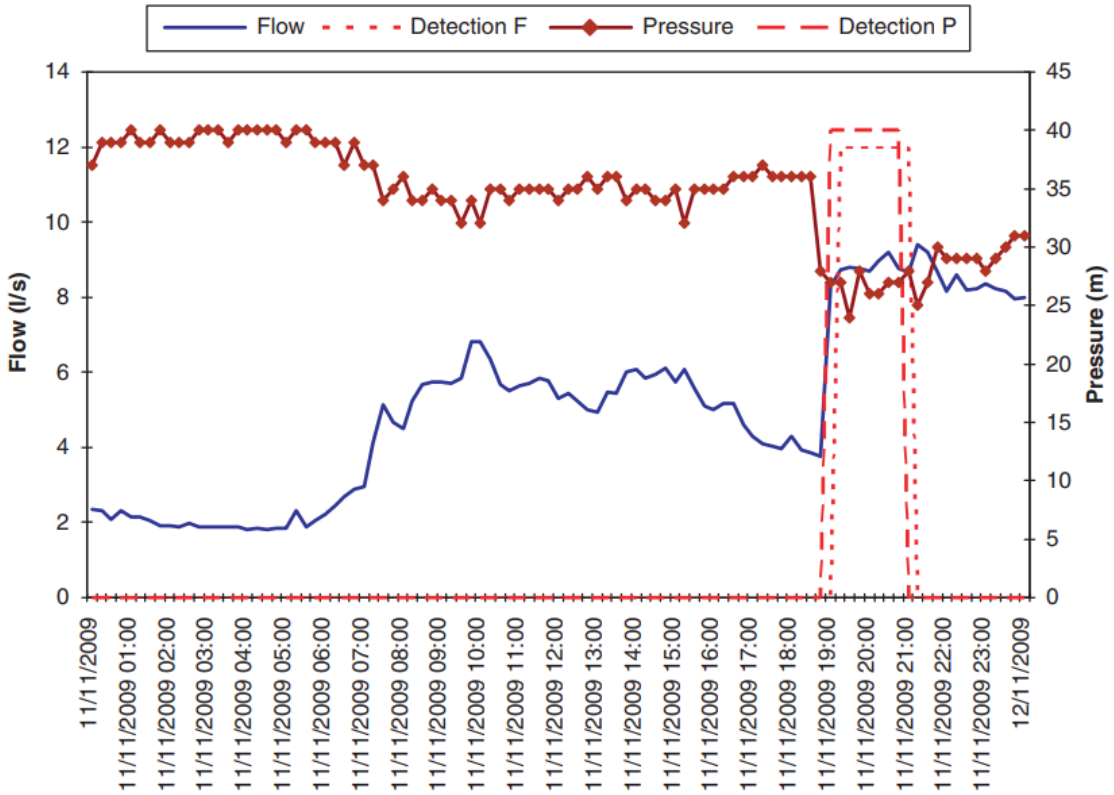


Figure 2.3: The flow and pressure measurements in a node of a WDN during an abrupt leak. The dotted lines mark where the leak was detected, after the first measurements register the drop in pressure and increase in flow.

Source: Mounce et al. (2010)

2013). Sufficiently large abrupt leaks were denoted called bursts. Current methods focus on discovering the abrupt changes in pressure and flow caused by these leaks, as discovering the slow change after incipient breaks were hard to differentiate from the changes caused by normal consumer demand. Minimum NFA also had this problem of deciding if a slow increase in flow was caused by a customer trend or an incipient leak. Figure 2.3 shows how an abrupt leak could be identified by observing the change in pressure and flow measurements.

An overall problem with leak detection was differentiating a leak from other abnormal events in the hydraulic data of a WDN. Abnormal events could be caused by several different events, such as maintenance activities as flushing(Mounce et al., 2010), unusual customer demand, industrial uses and weather(Herrera et al., 2010; Wu and Liu, 2017).

Reports and NFA might not give an accurate location of the leak. If this was the case more economically and resource intensive hardware based location methods must be used. These were usually very accurate, being able to localise a leak to within $\leq 2,5$m, compared to current software-driven methods, which was within ≤ 50m (Li et al., 2014). As the hardware methods were expensive and time-consuming, improving the software-driven methods accuracy of both detection and location was desirable. This work focused on leak detection, but leak localisation was mentioned as it was usually used in conjunction with detection algorithms.

Current software leak detection methods were usually split into three categories, Data-driven methods, Transient-based methods, and Model-based methods based on their approach. These were *leak detection* methods and must not be confused with similarly named ML-methods.

2.2.1 Data-driven methods

Data-driven methods focus on identifying leaks using online monitoring data and historic data of WDNs. The advantage of this was that these methods did not need knowledge of the WDN. Depending on what information was available, these methods viewed leak detection as different problems. If prior knowledge was present and labeled, leaks could be detected using pattern recognition between patterns in online and historic data. Mounce et al. (2013) built a library of pressure patterns connected to bursts and did pattern matching against this library. The system showed promise, but the library required expert domain knowledge to build. Leak detection could also be viewed as a classification problem where observable data was classified as leaks based on historic data, and if there were no labels available detection could be done by clustering(Wu et al., 2018). A drawback of these methods was that the system was dependent on historic data from the WDN. New or newly renovated WDN would not have any historic data. Another problem arises in real situations, as the available data may not be correctly labeled because of the difficulty of correctly pinpointing when a leak began given the historic data. The example leak is shown in Figure 2.3 would be easy to label because of the sudden change in pressure and flow, but incipient or smaller leaks might not have such a clear change in sensor readings that it would be easy to pinpoint its beginning.

2.2.2 Transient-based methods

These methods use the changes in pressure/flow that occurs after a burst or leak in the pipe network by tracking the pressure drop/flow increase as it spreads throughout the network(Eliades and Polycarpou, 2012). To track the transient pressure/flow waves, these methods require a high sampling frequency and more sensors than other methods(Wu et al., 2018). The high sampling frequency requirement makes these methods not applicable to current water networks as they have few sensors with a low sampling frequency.

2.2.3 Model-based methods

Leak detection by model-based methods was done by creating a numerical model of pressures and/or flows of a WDN without leaks from historic data over a 24 hour time window. Both pressure and flows could be modeled, but pressure was usually chosen because the sensors are easier and cheaper to install(Casillas Ponce et al., 2013). After modelling the ideal scenario without leaks, models were constructed which simulates a leak of a given magnitude at each node(possible place for a leak) in the WDN at each time step over a 24 hour time window. Ideally, these models would be constructed analytical, but this was a multivariable non-linear system of equations which may not have an explicit solution, which was why they were approximated instead or simulated using a hydraulic simulator like EPANETEPANET. The difference between the ideal scenario and the simulated scenarios with a leaky node at time k was stored in a sen-

sitivity matrix, as explained in Casillas Ponce et al. (2013):

$$S(k) = \begin{bmatrix} \frac{p_1^{f_1}(k)-p_1(k)}{f_1} & \cdots & \frac{p_1^{f_m}(k)-p_1(k)}{f_m} \\ \vdots & \ddots & \vdots \\ \frac{p_n^{f_1}(k)-p_n(k)}{f_1} & \cdots & \frac{p_n^{f_m}(k)-p_n(k)}{f_m} \end{bmatrix} = \begin{bmatrix} s_{11}(k) & \dots & s_{1m}(k) \\ \vdots & \ddots & \vdots \\ s_{n1}(k) & \dots & s_{nm}(k) \end{bmatrix} \tag{2.1}$$

where $p_i^{f_j}(k)$ was the expected pressure at sensor i when effected by a leak of magnitude f_j at node j, $p_i(k)$ was the pressure of sensor i when there were no leaks in the system, and the difference was normalised with the magnitude of the leak, f_j. $s_{ij}(k)$ represents the effect of a leak f_j on the pressure measured at sensor i at time k. This *sensitivity matrix* was what all model-based methods had in common, and they differed in how they use this matrix to identify leaks and if they used different variations of the matrix. Casillas Ponce et al. (2013) calculated the *residual vector*, the difference between the measured pressures in the WDN and the models estimation of them, and compared it with the columns of the sensitivity matrix using different distance measurements. The column vector closest to the residue vector indicated what magnitude of leak was currently present and at which node the leak was. These matrices could also be used to locate the leak after it has been discovered(Casillas Ponce et al., 2013; Soldevila et al., 2016b).

These methods have two limitations. Firstly, the sensitivity matrix was created, assuming leaks of specific sizes. If a real leak was of a different size, the residual vector might not match one of the columns of the sensitivity matrix, causing the system to miss label the leak or not identifying it at all. Additionally, the approximation of the sensitivity matrix requires historical data and deep knowledge of the system to estimate $p_i^{f_j}(k)$, or it was limited to the accuracy of the hydraulic simulator.

2.3 AI-methodologies

In later years more and more sensors have been placed in WDNs increasing the amount of available operational data. This has caused researchers to test implementations of different ML-methods for leak detection. To answer RQ2, a leak detection system was implemented based on one of the three considered ML-methods, Case-based reasoning (CBR), Convolutional neural network (CNN) and Recurrent neural network (RNN).

2.3.1 CBR

Case-based reasoning was a branch of machine learning based on how the brain induces solutions to new problems from solutions to previously experienced problems. A case-based reasoning system was often based on the framework introduced by Aamodt and Plaza (1994). The CBR-cycle has four phases, *retrieval*, *reuse*, *revise*, *retain*, each integrating with the case-base, operating environment, and/or the solution as shown in Figure 2.4. A CBR-system extracts the features needed from the problem it was supposed to solve during the retrieval phase to search the case base for similar problems retrieving them and their solutions. The *case base* stores previously encountered cases, the combination of problems and their solutions. *Similarity measures* were used to measure the similarity between different problems and solutions. The re-

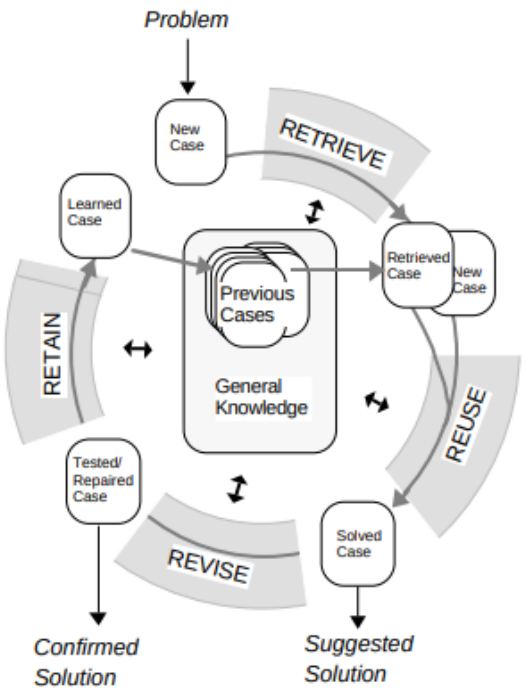


Figure 2.4: The CBR Cycle

Source: Aamodt and Plaza (1994)

trieval phase usually return several solutions so the system could use several possible solutions in the future phases.

Next, the system evaluates the retrieved cases for reuse. Some systems might also generate an explanation for why the cases were retrieved at this step. The retrieved cases might not be perfect solutions for the current problem, and therefore, the solutions may be *adapted*. These adaption could change one of the retrieved cases independently, or they might adapt a case based on the other retrieved cases to combine favorable features from each case.

After the system has adapted several possible solutions, the reuse phase ends. The solutions from the previous phase were evaluated in the *revise phase* to find the most fitting solution to the problem, and to discover any fault with the possible solutions. The evaluation could be done through a supervisor, applying it to the environment or with a model.

After the solution was applied, any knowledge discovered through the last phase was *retained* in the system. The simplest learning done in a CBR system was done through storing the new case in the case base. Other learning methods were, i.e., updating the cases in the case base that were similar to the newly solved case.

CBR was reviewed as a possible approach for leak detection. There were several aspects of a CBR-system that would be beneficial for this problem, exemplified in Mounce et al. (2013). The paper proposed and tested a system for burst detection in networks that could be reworked into a CBR system. The authors constructed a pattern library storing several burst patterns consisting of flow data and used pattern matching against this library to identify bursts. The patterns were stored as normalised vectors of flow data. The example showed to some degree that the burst patterns could be transferable. However, the approach also had problems; the library had

to be constructed manually by a domain expert, and the performance was not as good as other AI methods tested in the paper.

The analogy to a CBR system was clear; The system could be built using the pattern library as a case-base, where a case represents the pattern and its classification. Research in CBR proposes many possible improvements to the suggested system. The paper uses Euclidean distance between inputs and the vectors of the library to match patterns, an improvement could be another similarity measure as cosine distance or learning the measure (Stahl, 2005), although this would require training data and the representation of this data was not clear. Another improvement could be based on Smyth and Keane (1995), where the pattern library could be improved by retaining more general patterns and avoiding the swamping problem using competence-preserving .

However, after some serious attempts to develop a case base, we chose not to construct a CBR system because of the lack of data and domain expertise needed to construct the pattern library, as neither an expert nor suitable data sets were sufficiently available. Further more, the lack of research on using a CBR system with hydraulic data made it unclear how the cases best could be represented, and how the similarity measures could be improved. However, there have been attempts on other ML approaches for leak detection, such as Multi layer perceptrons (MLPs), which made us switch focus to more specialised ANNs.

2.3.2 Convolutional Neural Networks

Convolutional neural network (CNN) was a specialisation of regular ANNs that has one or more layers that use the convolution operation. A convolution operation could be viewed as a weighted function,

$$C(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-m)$$

Where I was the input function, K was a matrix of weights called the *kernel* and $*$ was the convolution operation. The input function was applied to each feature in the input; then it was applied to the kernel before being summed into a single output(Goodfellow et al., 2016). If the kernel was smaller then the input, it was applied to parts of the input at a time, as shown in Figure 2.5

Convolution has three desirable properties:

- *Sparse interactions*; By using a kernel smaller than the input, each layer was no longer fully connected with the previous layer as in regular ANNs as depicted in Figure 2.6. Limiting the number of connections between layers increases the efficiency of the network as fewer parameters were needed in the calculation of each node's output.

- *Parameter sharing*; Through the use of a smaller kernel, single weights in the kernel also affect several nodes in the next layer, contrary to regular fully connected networks, where each weight only affects a single node. The concept was shown in Figure 2.7. This sharing of weights reduces the storage requirements for the network and causes each weight to be applied to each input, except around the edge of the input, as shown in Figure 2.5.

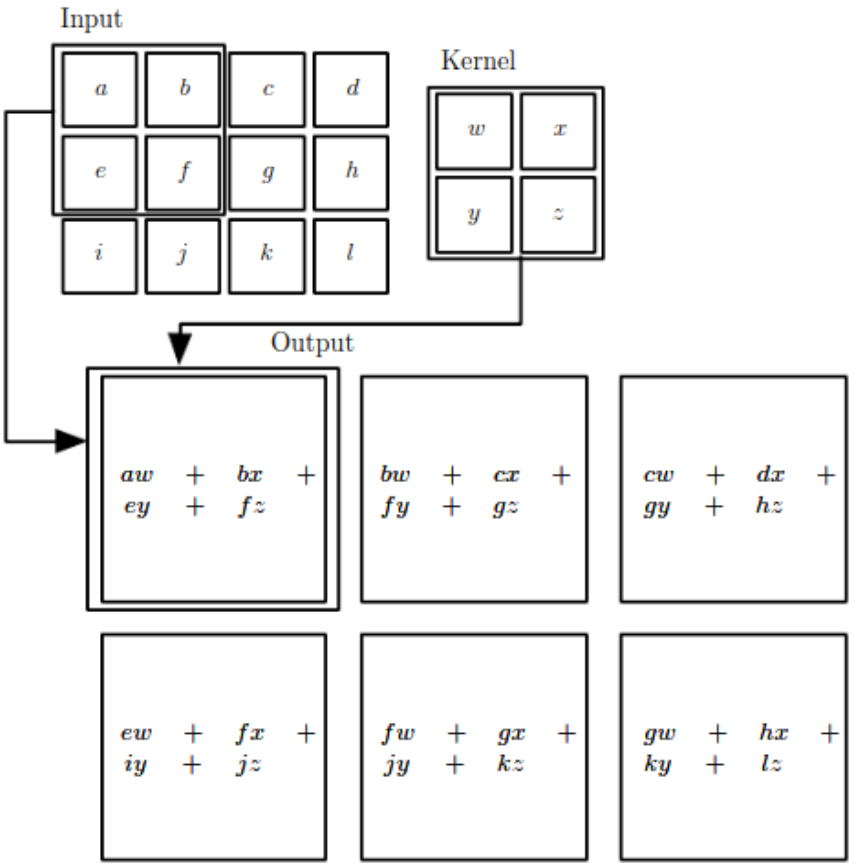
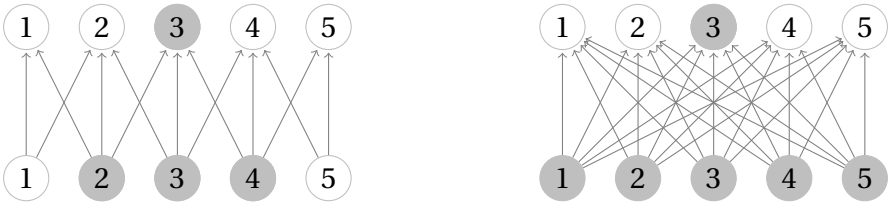


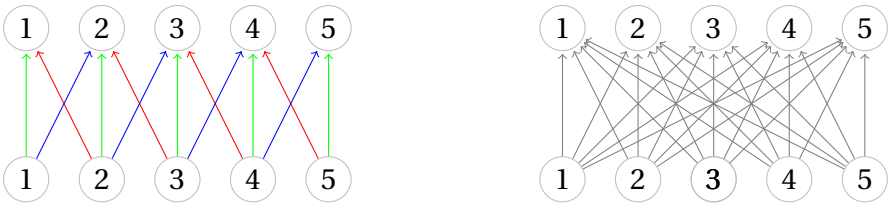
Figure 2.5: Example of how a smaller kernel was applied to a larger input. Notice that each weight in the kernel was applied to every input, except at the edges.



(a) The receptive field of a CNN with a kernel size of 3.

(b) The receptive field of a fully connected ANN.

Figure 2.6: The receptive field of a node indicates which nodes of the previous layer affects the node. The gray nodes of the previous layer represents the receptive field.



(a) Each coloured edge was a shared weight in a CNN with a kernel size of 3.

(b) A fully connected ANN where each edge was a separate weight.

Figure 2.7: The difference in weights used by a fully connected ANN and a CNN.

- *Equivariance representations*; Effects on the input also affects the output of a convolution function. Meaning that if a layer has time series data as input, lag in the input data will cause lag in the output of the layer.

A convolution layer could be viewed as a single complex layer or a combination of several layers. These layers were the convolution operation layer, then a detector stage(an activation function), then a pooling layer. The pooling layer was usually a statistical function that summarises inputs from several nodes in the previous layer into fewer output nodes, i.e., outputting the average of 4 nodes, or the maximum output of one of the previous nodes. Pooling layers gives a network invariance to translations in the input, and different combinations of pooling layers could give invariance to different kinds of translations, i.e., a network that identifies handwritten numbers could become invariant to both rotation of the input image and scaling by using several pooling layers. The use of pooling layers makes the network capable of focusing on abstract features in the input, and with deeper layers connecting these features into more abstract features, i.e., one layer may identify walls, while deeper in the network, a layer identifies corners by activating when two walls meet.

CNN invariance to translations makes them good with input that has a grid-structured topology such as 2-d data as images(Krizhevsky et al., 2012). The sensory data of a WDN can be viewed as both 1 dimensional, with a single sensor reading of each senor, or 2 dimensional, with the sensor readings over a time window of each sensor. One approach to applying CNN's as leak detection would be to let the CNN learn the local similarities in the sensory input, identifying a leak as the gap between the sensors that lie close to the leak and those that lie further away from the leak, as shown by the graph in 2.8. This gap was created by the leaks effect on the sensor readings through increased and decreased flow and pressure near the leak and this effect dissi-

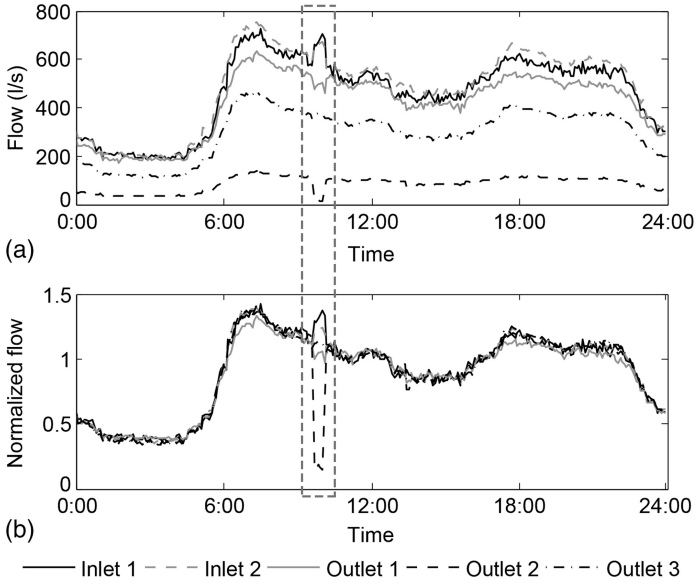


Figure 2.8: Plot of five flow meters. The gap between some of the sensors indicate a short burst in the network, where the most affected sensors lie closest to the burst.

Source: Wu et al. (2018)

pating trough the WDN away from the leak. Detecting this gap requires the input to be sorted such that positionally close sensors were placed close to each other in the input tensor. Both CNNs and the next approach, RNNs, could be used for leak detection as they were capable of handling temporal data. However, the next approach was chosen because the CNNs capability of handling temporal data was limited by its input. RNNs on the other hand, could store data from previous inputs in its hidden state and could therefore do classification based on previous inputs.

2.3.3 Recurrent Neural Networks

Recurrent neural network (RNN) were specialised for sequential data of the type $\vec{x}^0, \vec{x}^1, ..., \vec{x}^t$. The architecture of a RNN has temporal connections between nodes, i.e. the network has a connection to previous time steps as shown in Figure 2.9, where the weights $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{W}$ were shared between time steps. By sharing the parameters, the model was able to generalise over the whole input sequence. The architecture maps each input of the sequence into a hidden state, $h^{(t)}$ in Figure 2.9, which is used when calculating the next hidden state, $h^{(t+1)}$. This enables the model to handle sequences of varying length. The networks hidden state separates RNNs ability to handle sequences from CNNs, as convolution only detects local similarities within the input, the hidden state enables earlier input to affect the output of current input in RNNs.

The input sequence was mapped into the hidden state using different versions of Equation 2.2, depending on the implementation of the RNN.

$$\vec{h}^{(t)} = f(\vec{h}^{(t-1)}, \vec{x}^{(t)}; \vec{\theta}) \tag{2.2}$$

where $\vec{x}^{(t)}$ was the input at time step t, $\vec{h}^{(t-1)}$ was the hidden state at the previous time step, and

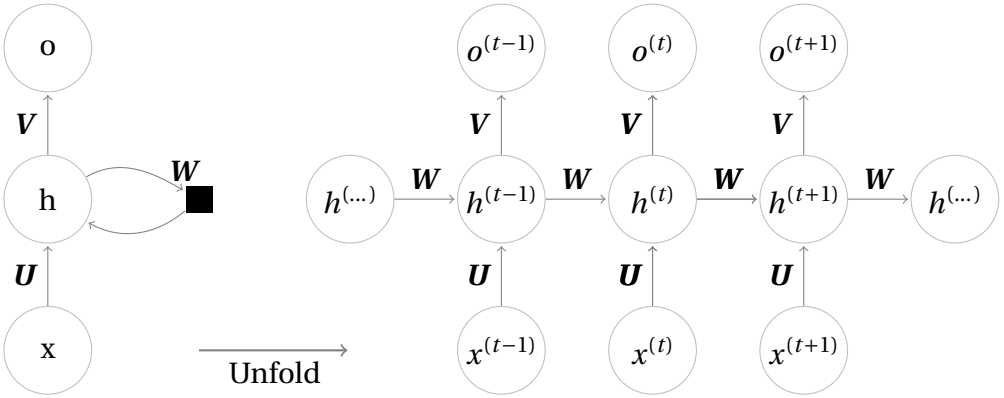


Figure 2.9: The folded and unfolded graph of a RNN with connections between hidden units. The black square represents the hidden state of the previous/next step. The folded graph gives a general overview of the architecture and the unfolded graph gives an explicit view.

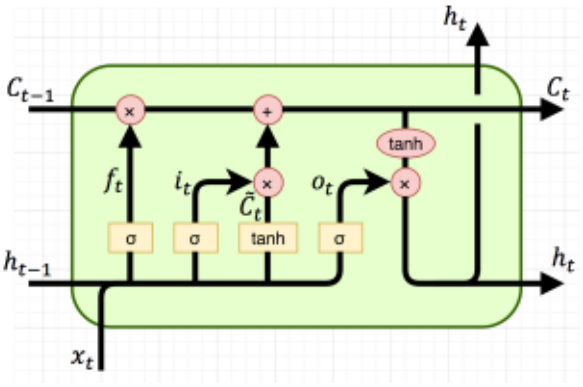
$\vec{\theta}$ was the parameters of the model. As the sequence grows and the hidden states dimension was fixed, the hidden state represents a summary of previous inputs.

When designing an RNN, there were three different main designs to choose from on how the units of the network were connected through time:

- The RNN could produce an output at each time step and have connections between the hidden units, as the network in Figure 2.9.
- The RNN could produce an output at each time step and have connections between the output of the previous layer and the hidden layer of the next layer.
- The RNN could produce a single output after reading the entire sequence.

The most powerful RNNs were the networks that have hidden-to-hidden connections as these could learn what was valuable to map into the hidden state and pass to the next time step, while also learning what to output at each time step. RNNs with output-to-hidden connections has the handicap that it could only map information from the past through its output, so what the networks learn to pass into the hidden state of the next layer was dependent on the training. However, these output-to-hidden networks were able to utilise *teacher enforcing*. This was done by using the ground truth of the previous time step $y^{(t-1)}$, together with the input $x^{(t)}$ to calculate $y^{(t)}$. This means the gradient could be calculated at each time step independently of previous time steps. When using hidden-to-hidden connections, the gradient was dependent on the previous layers gradient. RNNs using hidden-to-hidden connections could be viewed as Directed Graphical Models, and therefore use *Markov Assumptions*, so the number of computational complexity could be lowered on these networks.

In scenarios where the whole input sequence was needed for a prediction, *bidirectional* RNNs were used to get connections between inputs early in the sequence with inputs later in the sequence. These were implemented by using two sub-RNNs, where one RNN reads the input in the inputted order, and one reads the input in reversed order. The output of this network was the combination of these sub-networks, and the network could learn to connect earlier parts of the sequence with later parts. As the structure has nothing to do with the size of the sequence, bidirectional networks were also applicable to arbitrary-sized sequences.



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

Figure 2.10: The architecture of a LSTM cell and the calculations for each gate. Here x_t was the input, f_t, i_t and o_t was the activation of the forget, input and output gates respectively, $\tilde{C}_t$ was the activation of x_t and h_{t-1}, and C_t and h_t was the cell state and hidden sate, with h_t also being the output.

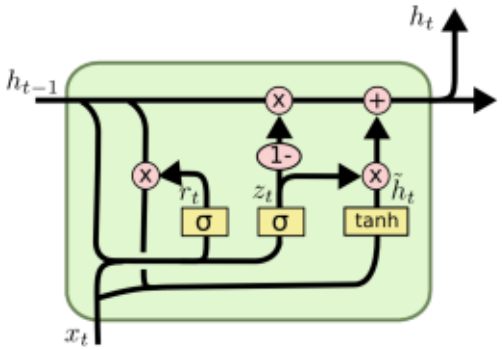
Source: https://isaacchanghau.github.io/post/lstm-gru-formula/

The central problem when trying to learn long-term dependencies with RNNs was the vanishing and explosion of gradients. Notice that in Figure 2.9 the matrix $\boldsymbol{W}$ was applied at each time step to the hidden state, so each output depends on the previous hidden states. Assume that $\boldsymbol{W}$ has an eigendecomposition $\boldsymbol{W} = \boldsymbol{V} diag(\lambda) \boldsymbol{V}^{-1}$, when $\boldsymbol{W}$ was applied t times, this becomes $\boldsymbol{W}^t = (\boldsymbol{V} diag(\lambda) \boldsymbol{V}^{-1})^t = \boldsymbol{V} diag(\lambda)^t \boldsymbol{V}^{-1}$. So if the magnitude of any λ_i in $diag(\lambda)$ isn't close to 1, W^t either explodes if $|\lambda_i| > 1$ or vanishes if $|\lambda_i| < 1$. This effect also impacts the gradient, resulting in either no learning because of small changes to the weights or unstable learning because of big changes to the weights.

To combat this problem, *Gated RNNs* were introduced, which focused on creating links through time that did not have vanishing/exploding gradients. This was done trough *control gates* that were weights and activation functions that were applied to a hidden state and input of the RNN. Through training, the net learns when and what information to store and remove from the hidden state using the gates. The Figures 2.10 and 2.11 show the architecture of two gated RNN units, Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU). In networks these units were swapped with the regular hidden unit, the h node in Figure 2.9, in the RNN, and they could be *stacked* after each other, feeding the output and hidden state to the next unit.

The LSTM unit has two temporal links, the cell state C_t and the hidden state h_t(which also functions as the output). As seen in Figure 2.10 C_t and h_t were only affected by input based on the activation and weights of the input, forget, and output gates. This gives the network the possibility to learn what to forget and what to store. The cell and hidden states have hidden-to-hidden connections(also called self-loops) that keep the gradients from vanishing or exploding based on context. LSTMs has been shown to learn long-term dependencies.

A GRU cell was a stripped version of the LSTM cell. The GRU cell has only one state, h_t, and combines the LSTMs input and forget gates into a single update gate for the hidden state. Although one might think this makes the GRU less effective Jozefowicz et al. (2015) shows that the GRU could outperform an LSTM on several tasks, except for language modeling, and when



$$z_t = \sigma\left(x_t U^z + h_{t-1} W^z\right)$$
$$r_t = \sigma\left(x_t U^r + h_{t-1} W^r\right)$$
$$\tilde{h}_t = \tanh\left(x_t U^h + (r_t * h_{t-1}) W^h\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 2.11: The architecture of a GRU cell and the calculations for each gate. Here x_t is the input vector, z_t and r_t is the activation of the update and reset gates, and h_t is the hidden state and output.

Source: https://isaacchanghau.github.io/post/lstm-gru-formula/

testing different LSTM architectures, they found that the best performing architectures were very close to the GRU architecture. Since none of the problems used for testing were similar to the leak detection problem, we chose to implement and test both gated RNNs. Because of the sequential, dynamic nature of the hydraulic data from a WDNs and the gated RNNs ability to learn temporal information was the main reason for choosing gated RNNs over the other two AI-methodologies outlines in this chapter.

Chapter 3: Literature Search

This chapter covered the literature search methods used to answer RQ1 and its results. The approach was evaluated in section 6.1.

3.1 Methods & Approach

A literature study was necessary to answer RQ1. To gain enough competence a *literature research protocol* was created, following the SLR process outlined in Kofod-Petersen (2018). In this process a predefined set of search engines, search word (SW), exclusion criteria (EC), inclusion criteria (IC) and quality criteria (QC) was used to find and choose relevant work for further study. A query was constructed by combining search words using AND and possible synonyms using OR, for example, "Leak or Anomaly AND detection", and the query was used with the predefined search engines. The abstract of the resulting papers were reviewed using the exclusion- and inclusion criteria and was either ignored or included for a more thorough study. The included papers were then reviewed and scored using the quality criteria, ensuring that only the best papers were used as a basis for further research.

The SW and criteria outlined by Figure 3.2 and the search engines IEEE Xplore and arXiv(IEEE, 2018; Cornell University Library, 2018) were used in the first attempt at a SLR. The engines used were chosen based on our prior experience searching for research papers and were thought to be sufficient as they both cover computer science literature. However, this was proven to be a bad assumption as it did not give any results that passed the predefined IC listed in Figure 3.2c. To discover other relevant search engines and publisher, we reached out to Rita M. Ugarelli at SINTEF Building and Infrastructure after searching for domain experts at NTNU. Rita referred us to Mounce et al. (2015) and as we only had one paper, a snowballing process (Wohlin, 2014) was considered instead.

The snowballing procedure was an iterative systematic literature review process. Firstly, a set of possibly relevant papers were identified to start the literature search. These first papers could be found using any other literature search method, for example, a database search or the previously described SLR. The tentative start set was then evaluated using different inclusion- and exclusion criteria, resulting in a final start set. Each paper in the start set was then used in the iterative snowballing process by conducting forward and backward snowballing. Backward snowballing consists of reviewing the references of a paper and forward snowballing reviews other papers that cite the paper. The papers included in one iteration was then used as the base for the next iteration. The process was summarised in Figure 3.1.

Mounce et al. (2015) was not directly relevant to this work, as it focuses on dealing with leak management, not detection and was therefore not used in the SLR. However, the author and

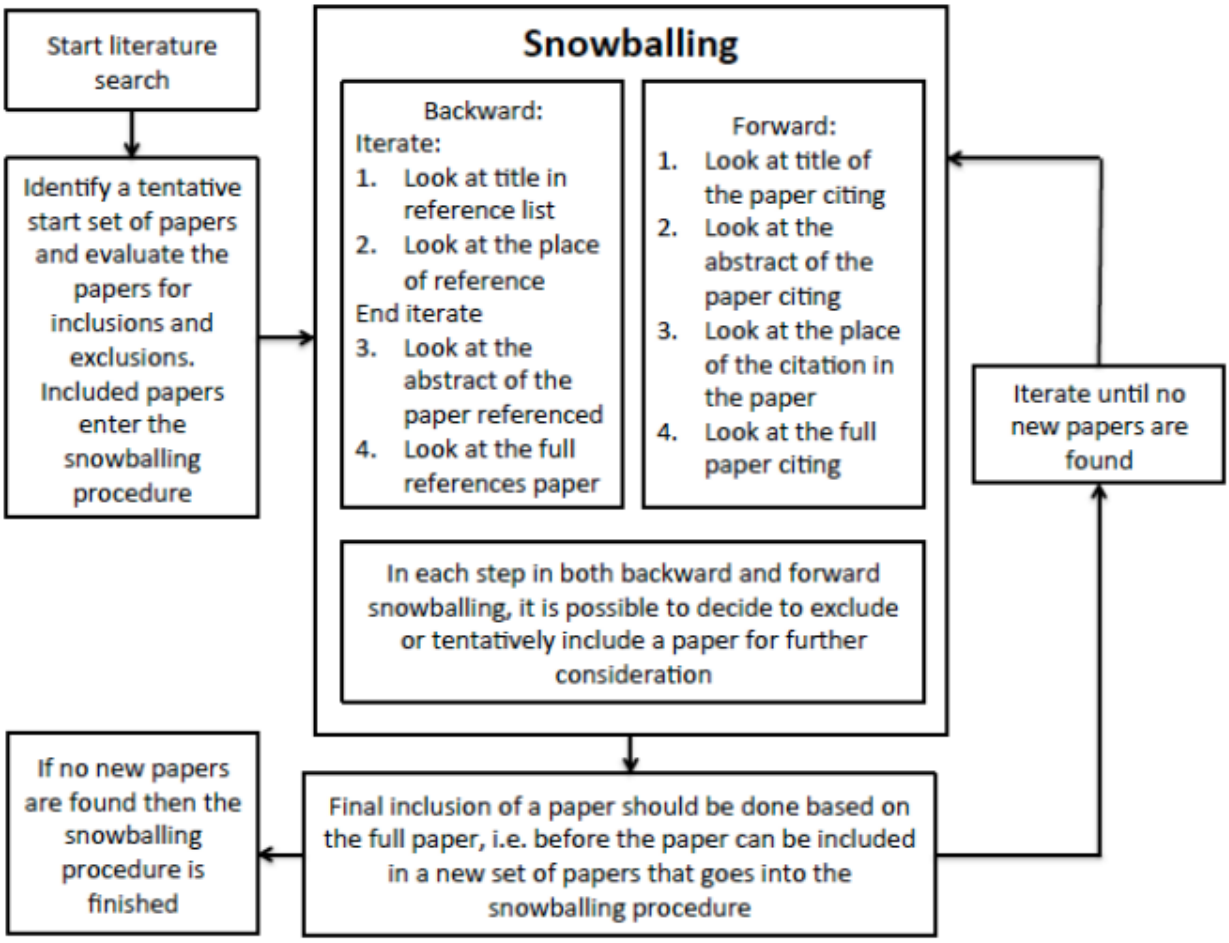


Figure 3.1: A summary of the snowballing structured literature review method.

Source: Wohlin (2014)

publisher have published several relevant papers. The paper was published using IWA publishing, who publishes the Journal of Hydroinformatics(IWA Publishing). Their search engine was added to the SLR process and used to create the initial start set. The snowballing approach was chosen to extend the SLR for the following reasons;

- It would give a broad understanding of the hydroinformatics field and its state of the art by adding papers reviewing current solutions in the field
- By using reviews of current statistical and ML approaches, sources for comparable results could be found
- The pool of papers discovered by the snowballing approach could give results that covered other publishers trough references
- Our lack of knowledge on the domain's terminology might have a greater effect on the results of a SLR than on a snowballing approach

The two first reasons were covered by a SLR after adding IWA publishings search engine, and the third by adding new search engines to the search after discovering them through the references in papers. However, the snowballing process would ensure all four guarantees, and we hypothesised that the snowballing approach would reduce the number of searches required.

The start set for the snowballing procedure was constructed using the SLR described above. The search engines IeeeXplorer, IWA Publishing, and arXiv were used, and all the search words and criteria used were listed in Figure 3.2. A combination of the chosen SW were used and the subsequent papers were selected or removed if their abstract matched one or more of the IC or EC. The search resulted in 34 papers in the tentative start set, using the IC/EC this was reduced to seven. No specific ML methods were used as search words as the goal was to identify what types of methods were in use, per RQ1. The addition of any ML methods would narrow the search down, and as methods have different names, it might cause papers to be missed.

The papers in the tentative start set were scored using the QC and the top three papers were chosen as a start set for the snowballing method[1]. For each QC the paper gets a score if it does not(0), partially(1/2) or fully(1) comply with the criteria.

The SLR resulted in the start set shown in Table 3.1. The snowballing procedure was conducted and by reviewing the title and abstract using the IC/EC. Between each iteration, the papers were read and papers that did not comply with the QC were dropped. The result of the full snowballing process was listed in Table 3.2.

Paper	QC1	QC2	QC3	QC4	QC5	QC6	QC7	QC8	Total
Casillas Ponce et al. (2013)	1	1	1	1	1	1	1	1	8
Deniss et al. (2012)	1	1	½	½	1	1	1	½	6.5
Eliades and Polycarpou (2012)	1	1	1	½	0	1	1	1	6.5
Soldevila et al. (2016a)	0	1	½	½	1	1	0	½	4.5
Sánchez-Fernández et al. (2015)	0	1	½	0	0	½	1	½	3.5
Vries et al. (2016)	1	1	½	0	½	0	0	0	3
Sun et al. (2016)	0	1	½	0	0	1	0	0	2.5

Table 3.1: Results of the SLR, the top three papers where chosen for snowballing.

Iterations	Papers		
Start set	Casillas Ponce et al. (2013)	Eliades and Polycarpou (2012)	Deniss et al. (2012)
1	-	Soldevila et al. (2016b) Herrera et al. (2010)	Mounce et al. (2010) Farley (2001)
2	-	Chan et al. (2018)	Mounce et al. (2013) Mounce (2013)
3	-	-	Wu and Liu (2017) Ye and Fenner (2011)

Table 3.2: Results of each iteration using snowballing. Each column shows which paper from the start set it originates from.

[1]The top three was chosen because of the time constraints of the thesis.

(a) Search words

SW1: Leak/Anomaly/Fault

SW2: Detection

SW3: Water distribution network (WDN)/District metering area (DMA)

(b) Exclusion criteria

EC1: The paper covers a method already covered by another paper(The highest scoring paper was kept)

EC2: The same study was found from a different source

(c) Inclusion criteria

IC1: The paper uses an AI-method to detect faults in WDNs

IC2: The paper was related to one of the RQ

IC3: The paper gives an unique view of a RQ

IC4: The paper gives an overview of the current methods for leak detection in WDN

IC5: The paper contains a fault detection method for WDN

IC6: The paper contains comparable results

(d) Quality criteria

QC1: The paper compares different leak detection methods

QC2: The paper has a clear aim for the research

QC3: The paper has a reproducible method

QC4: The paper has a reproducible result

QC5: The paper uses an accessible data set

QC6: The paper justifies the design decisions used in the solution

QC7: The paper contains comparable results

QC8: The paper uses explained and justifiable metrics

Figure 3.2: Search words and criteria used in the literature search, synonyms were separated with a /

3.2 Related works

The first papers read was the start set, Casillas Ponce et al. (2013), Eliades and Polycarpou (2012) and Deniss et al. (2012). These papers were chosen based on their scoring using the QC.

3.2.1 Start set

Casillas Ponce et al. (2013) presents five different model-based methods of leak detection using a sensitivity matrix and residue vector. The sensitivity matrix was constructed as explained in Section 2.2.3 by simulating the WDN pressure values with and without leaks at different nodes in the network. The residue vector, $\vec{r}$, was created by taking the difference between the expected simulated sensor values, without any leaks, and the actual measured pressure values. The five

different leak detection methods presented in the paper differ by how they measured the difference between the residue vector and the sensitivity matrix. The distance was measured at each time step within a time window of 24 hours, and each method summarised the difference over time by different means. The methods were:

- *Binarised sensitivity method* binarised the sensitivity matrix, S, and residue vector, $\vec{r}(k)$, using two parameters to threshold the values. Then the columns of the sensitivity matrix, S_j, was compared with the residue vector and if they were equal, $S_j(k) = \vec{r}(k)$, it indicated a candidate leak. Over the time window, these indications were stored in a vector, γ, where each component was the sum of indications on each node during the time window. The largest component then indicated which node leaked.

- *Angle between vectors method* calculated the cosine distance, α_j, between S_j and $\vec{r}(k)$. The mean angle of each α_j within the time window was calculated. The leak was then identified as the node with the smallest mean angle, i.e., the column vector, that lay closest to the residue vector indicated a leak.

- *Correlation method* and *Euclidean distance method* worked the same as the angle between vectors method, but uses correlation and Euclidean distance instead of the angle between columns and the residue vector.

- *Least square optimisation method* used the least square method by optimising:

$$J_{f_j} = \min_{f_j} \sum_{k=1}^{L} | \vec{r}(k) - s_j^{f_j} k |^2 \quad j = 1, ..., m$$

  for each candidate leak size, f_j, and the node, j, with the smallest J_{f_j} was the leak node.

The experiments in this paper were done on two WDNs, one in Hanoi and one in Quebra. Two hundred experiments were conducted per network, where each experiment added a demand noise of ±4% and a pressure measurement noise of ±2%. The leaks were between 20 and 80 l/s on the Hanoi network and between 0.01 and 1l/s on the Quebra network. From Table 3.3 it was clear that the best methods were the optimisation and angle methods. These methods all share the same disadvantages as the model-based methods mentioned in Section 2.2.3.

Method / Network	Binarisation	Correlation	Angle	Distance	Optimisation	Wavelet
Hanoi	48	60	98(31)	46.67	96	21
Quebra	71.5	94.5(56)	98.5	13.33	98.25	62

Table 3.3: Accuracy of the Model-based methods presented in Casillas Ponce et al. (2013) and Deniss et al. (2012). The results using the angle method of Deniss et al. (2012) were in parentheses.

Deniss et al. (2012) used model-based leak detection together with wavelet analysis to generate a comparison matrix that was then used with a voting system to localise leaks. The sensitivity matrix was constructed in the same way as presented in Section 2.2.3, but it also constructed a

sensitivity matrix for the unknown leak, $S_a = P - P_a$ where P_a were the sensor measurements of a time window. The residue therefore represented a matrix, $R_a = P - P_a$, instead of the residue vectored used in Casillas Ponce et al. (2013). On the columns of S, S_a and R_a a continuous wavelet transform was performed using complex Shannon wavelets(the precise transformation could be viewed in the paper), obtaining matrices of complex numbers. The matrices were expanded by separating the real and imaginary parts into separate columns; then they were binarized by setting the values to 1 or 0 if the real and imaginary parts were above or below 0, respectively. At the end of this process, the binarized matrices of the transformed columns of S and R_a represent a leak at a specific node at a given time step. These matrices were compared using an XOR-operation and summed into a comparison matrix where leaks could be identified to be the indexes with the smallest values.

The paper also implemented the previously mentioned angle between vectors method(Casillas Ponce et al., 2013) for comparison and tests the methods on the Hanoi and Quebra WDNs. The tests done on a single leak were similar, but Deniss et al. (2012) tested using several other levels of measurement and demand noise, and the measurement noise was increased to ±4%. This paper also used leak sizes in a smaller range, $30-80 l/s$ and $0.01-0.1 l/s$ for Hanoi and Quebra respectively. Table 3.3 shows that the accuracy of the wavelet method was not as good as that of the other model-based methods. However, it must be noted that the measurement noise added in Deniss et al. (2012) was ±4% versus the ±2% in Casillas Ponce et al. (2013), which might explain why the results of the reproduction of the angle method were worse. The result difference might also be caused by the difference in how the WDN was modeled, as Casillas Ponce et al. (2013) used a software framework, EPANET, to model the water network, while this paper used its own decomposition based approach. As mentioned in Section 2.2.3, these methods are dependent on the accuracy of the model when constructing the sensitivity matrix, as clearly shown here where two different models give huge differences in results. This, and the other results noted in the paper hints at the high impact of noise in the measurements on the leak detection methods and the importance of the model in these model-based methods.

Eliades and Polycarpou (2012) presented a model-based method where a Fourier Series was used as a model, and CUSUM was used to detect a leak. Other model-based methods require well-calibrated models, such as the EPANET simulations and decompositions used in the previously mentioned methods, while this paper suggested an adaptive model that learns over time. The model was constructed by decomposing the DMA inflow signal into two components, the long-term trends(yearly seasonal changes) and weekly trends, an example of this decomposition could be seen in Figure 4.1. The DMA inflow at time step k could then be described as

$$q(k) = r(k)s(k)(1 + n(k))$$

where $r(k)$ described the long-term trend, $s(k)$ the weekly trend and $n(k)$ described a uncertainty component with zero mean normal distribution. The long-term trend was calculated from historical data using least-squares optimisation on a Fourier series representation of the trend. The weekly trend, $s(k)$, was represented as two vectors, one of Fourier functions and one of the Fourier coefficients, where the coefficients were updated using a learning rule. A leak was detected if the CUSUM of the mean value of the inflow signal was greater then a given threshold. This threshold could be changed to adjust the resulting FPR and TPR of the system. The CUSUM method was compared with the baseline night-flow analysis method outlined in Section 2.2. The testing was done on historical data from a DMA in Limassol, Cyprus, and the

method managed at best a TPR of 94.5% and FPR of 0%. However, this method had a detection delay of 9.8 days when it got the best results, and the other results also showed an average detection delay of several days. The night-flow method did not get as good results as the CUSUM method, with its best being a TPR of 73% and FPR 0%, it was however faster by having a detection delay of a couple of days. In contrast, the previously mentioned methods have a detection delay of at most the size of their time window.

3.2.2 First iteration

Mounce et al. (2010) presented a SVM approach for novelty detection in pressure and flow data. SVMs find the optimal hyperplane to separate two classes, by choosing two vectors $\vec{w}$ and $\vec{b}$ such that $\vec{w}^T\vec{x} - \vec{b} = y$, where $\vec{x}$ was the input data and y was the class of $\vec{x}$. SVMs could be extended to be applicable to regression, Support Vector Regression (SVR), by having a real valued output $f(\vec{x}, \vec{w})$, rather than a discrete class, y, so the SVR model becomes $f(\vec{x}, \vec{w}) = \vec{w}^T\vec{x} - \vec{b}$. The paper takes into account the diurnal pattern in WDN data and therefore constructs 96 weekday, 96 Saturday and 96 Sunday models, one model per sensor measurement each day. When the difference between the predicted value of the model and the observed value was significantly large it was classified as an abnormal event. When trained using 3 months of data, with sensor readings every 15 minutes, and tested on 6 months of data, this method managed a TPR of 78% with a detection time of 2.5 hours.

Herrera et al. (2010) did not implement any leak detection methods, but rather compared different machine learning methods in forecasting water demand. This paper was chosen because of its application of ANN and other methods on water data. The methods compared in the paper were ANN, Projection pursuit regression (PPR), Multivariate adaptive regression splines (MARS), SVR, random forests, and a weighted function. The ANN was implemented with 3, 5, and 7 hidden nodes and learning rates of 10^{-4}, 10^{-3} and 10^{-1}. Each method was evaluated 20 times, where a random time step was chosen, and the model was trained using the eight previous weeks of data and the next two weeks were used for testing. The methods task was to predict the water demand for the next time step, $t+1$, based on the water demand of the current, t, and previous hour, $t-1$, the water demand at that time last week, $t+1-24\times 7$, and the last known values of temperature, wind velocity, atmospheric pressure and rain. The paper identified that the dynamics of the target time series changed over time, and therefore suggested two ways to update the models during testing, growing, and sliding windows. When using growing windows, the model was first trained using the original eight weeks. After a set sized time window, the previous inputs and their ground truth were added to the training data, and a new model was trained using this expanded training set. When using a sliding window, a new model was trained using the previous eight weeks of data, re training the model with the updating interval as the growing window. During testing window sizes of 1, 12, and 24 hours were used. This type of online learning would not be possible for a leak detection model as a leak cannot be verified live, unlike the predictions as they were verifiable within the next sensor measurement. The experiments were evaluated using RMSE, MAE, Nash-Sutcliffe efficiency and a modification of Nash-Sutcliffe[2], and the results were shown in Figure 3.3. The ANN model achieved best results when using a sliding window approach where the model was updated every 12 hours when

[2]The equations could be found in the paper as equations 16, 17, 18 and 19 respectively.

evaluated using MAE. However, in each of the evaluations, it was clear that the ANN model was inferior to other methods.

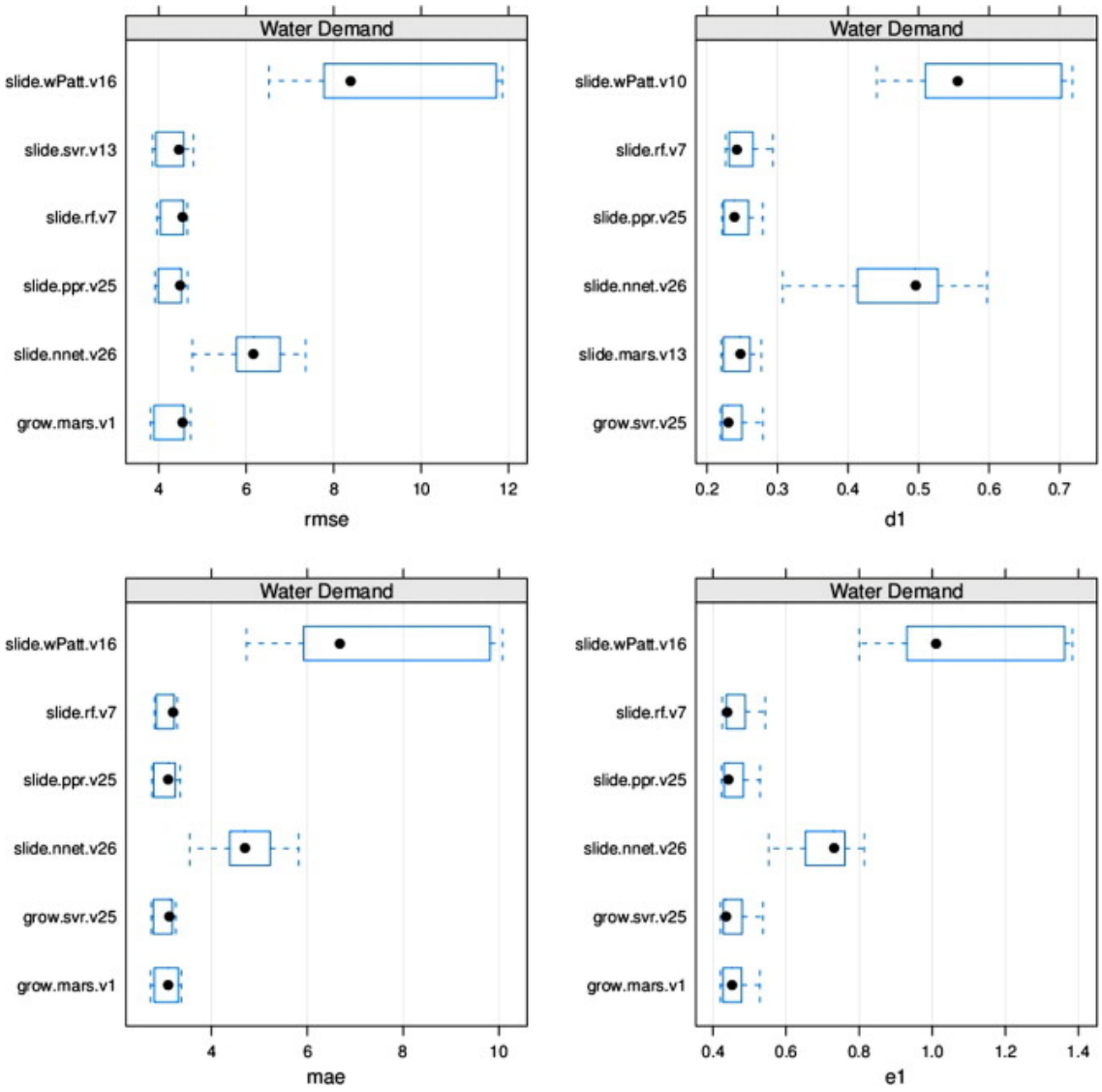


Figure 3.3: The best results of water demand prediction using ANN (nnet), projection pursuit regression (ppr), multivariate adaptive regression splines (mars), SVR (svr), random forests (rf) and a weighted function (wPatt). The evaluation was done using RMSE, MAE, Nash-Sutcliffe efficiency (e1) and a modification of Nash-Sutcliffe (d1). The slide and grow prefixes indicates if the model was trained using sliding or growing windows.

Source: Herrera et al. (2010)

Soldevila et al. (2016b) used k-Nearest Neighbour to localise leaks, given it had already been identified. This paper was read because it generated the training data similarly to how it was done in this work, and it reviews the limitations of model-based approaches. It identifies the problem of labeling real data, as it was not possible to guarantee when a leak starts. Because of this, the paper suggested using a model of the WDN to create correctly labeled training data.

The paper tested its leak localisation approach using the Hanoi network, with a leak size between 25 and 75 l/s, Gaussian noise on the pressure measurement with an amplitude of ±12.5% of the mean value of all pressure residuals, and a demand uncertainty of ±5% around the nominal demand. It was shown that the methods performance had trouble handling the demand uncertainty when presented with only one-time step, having aa accuracy of 43.48%. However, it was also shown that the accuracy increased significantly when using a whole day as input, gaining an accuracy of 89.95%. The model was also tested on a real case, Nova Icària, a part of the Barcelona WDN. The model was first trained using only generated labeled data and was verified to locate a leak node within a topological distance of 7 nodes. When tested with the real leak, it managed to localise the leak node within a distance of 13 nodes. Given the network consists of 1036 nodes and 5 sensors, this could be considered as good results. It also showed that a system trained with generated data could be successfully applied to real data.

Lastly, we would like to mention Farley (2001). As the title suggests, *"Leakage management and control: A best practice training manual"*, it was not an academic paper, but a reference work. It was used during the work on this thesis as such to gain greater insights into the domain, to look up terms, best practices and other leak domain-specific knowledge, but was not read cover to cover. It was referenced in several of the studied papers and according to Google Scholar been cited by 119 articles.

3.2.3 Second iteration

Mounce et al. (2013) presented a method for leak detection by matching operational data with predefined data patterns stored in an expert constructed library of burst patterns. The paper identified other attempts at using machine learning techniques for leak detection, and that they usually followed the same process; Learn from previous data to make a prediction of expected future data and defines rules for deciding when the measured data deviates enough from the expected value for it to be considered abnormal. These were prone to problems concerning a large number of parameters that needed to be tuned and poor data quality. The paper was originally chosen because of its mention of associative ANNs, but under further study, it was shown not to go into enough detail in its approach to be more relevant to this thesis.

Mounce (2013) presented a comparative study of ANN implementations for prediction of WDN flow data. It pointed out the use of ANNs for leak detection was usually done by training the model to predict future hydraulic data, then a secondary method/process analyses the output for a leak, the model were not trained to classify the leak directly. The paper also pointed out the lack of research using RNNs within hydroinformatics[3]. Furthermore, it pointed out that regular ANNs were designed to detect static pattern. As mentioned in Herrera et al. (2010), the hydraulic data collected from WDNs were dynamic of nature, indicating that simple ANN architectures were unfit for the domain, and further encouraging research on using RNNs. The paper tested four ANN architectures, Multi layer perceptron (MLP), Mixture density network (MDN), Time Delay Nerual Network (TDNN) and Time-lagged Recurrent Network (TLRN), in predicting flow data for a sensor 24 hours ahead. The models were trained and tested on two data sets with eight and three months of training data and six and five weeks of testing data, respectively. The data were normalised to a range of 0 to 1. The results measured using MAE were summarised

[3]Referred to as networks with temporal memory in the paper.

Network	Data set			
	Structure	1	Structure	2
MLP	96,4,1	0.0385	96,2,1	0.0398
MDN	96,10,1 GMM2	0.0362	96,15,1 GMM2	0.0417
TDNN	96,10,1	0.0160	96,10,1	0.0192
TLRN	96,500,1	0.0168	96,200,1	0.0219

Table 3.4: The MAE of the predicted a flow value 24 hours ahead using different ANN structures from Mounce (2013).

Citation	Method	Data type	TPR	FPR
68	Nonlinear Kalman Filter	Simulated Data	87%	0.01%
57	ANN, SPC, BIS	Historical Data	76%	10/8%
27	Ensemble CNN-SVM	Engineered Test	98.2%	0.2%
77	Multiclass SVM	Simulated Data	99.5%	-

Table 3.5: TPR and FPR of some of the reviewed methods in Chan et al. (2018). The citation number refers to the citation within the paper and "-" means that the measurement was not available in the paper. BIS = Bayesian Inference System, SPC = statistical process control

in Table 3.4. From the table, it was clear that the networks with a temporal link had better results. The author of the paper also pointed out that the temporal networks required less training epochs, between 300 and 500, while the non-temporal networks required 900 or more epochs. TLRN also had a more predictable performance compared to the TLFN.

Chan et al. (2018) gave a review of current leak detection methods, current research papers with their methodologies, limitations, and results. As the paper listed the papers case studies, it would be used to find cases using the same WDNs as this thesis. Two papers listed used the Hanoi network, one was Soldevila et al. (2016b), the other used a Bayesian classifier. Of the machine learning approaches reviewed, problems with detecting small leaks and the need for large amounts of historical data was listed as limitations for several of them. The authors of the paper also pointed out that several of the methods(not just the ML ones) required detailed knowledge of the WDN that cannot be known for certain and that a large number of sensors were required. Table 3.5 showed the TPR and FPR of some of the reviewed methods.

3.2.4 Third iteration

Wu and Liu (2017) presented a review of data-driven leak detection. It identified the following problems with data-driven methods:

- Historical data was hard to label correctly as it was difficult to determine when a leak began. This also affects detection time, as the labels cannot be guaranteed to be correct.
- Tests done using generated data usually gives better results than those done using historical data.
- It was hard to compare data-driven methods as the tests conducted has different parameters, both in evaluation criteria, network size, leak size, and leak duration.

Method	Data type	TPR	FPR	Leak size
ANN, TDNN	Historical Data	75%	0%	2-10%
MDN and FIS	Historical Data	100%	15%	-
ANN, SPC, BIS	Historical Data	100%	8%	5-16%
Modified SPC	Historical Data	80%	10%	-

Table 3.6: TPR and FPR of the reviewed methods in Wu and Liu (2017). The leak size was presented as a percentage of the average DMA inflow. FIS = Fuzzy Inference System

- Faulty sensors or missing measurement data was a problem in real data.

The paper also presented the TPR and FPR results of the reviewed papers, which were shown in Table 3.6

Ye and Fenner (2011) presented a Kalman filtering approach, where the Kalman filter was used to estimate the flow or pressure in a DMA. The paper suggested that the presented method was more computationally efficient, had a shorter DT, and needed less data to train, than ANN approaches. Leak detection was done by reviewing the residue vector. As the Kalman filter estimated either flow or pressure, a non-zero residue indicates a leak. When the Kalman filter models the DMA flow, the residue would directly correspond to the leak size. The method assumed that the flow or pressure at time k was equal to the flow or pressure at time $k-1$. To be able to apply this to the diurnal data, each time step of a week had its own filter, resulting in 672 independent filters with a sensor sampling every 15 minutes. Also, because of this assumption, the method only needed data from the previous week. The method was tested using both engineered tests and historic data. The article did not state the FPR or TPR, but successfully detected leaks that corresponded with customer complaints. It concluded that the method was better for detecting sudden bursts and gradually changing leaks than long-term existing leaks.

Chapter 4: Implementations

The SLR showed that previous attempts at using ANNs for leak detection have only used shallow MLPs, meaning the networks has few hidden layers. Only Mounce (2013) tests recurrent networks, but only to predict the values of a single sensor, not leak detection nor using data from several sensors. Because of the bad results using shallow network models, the lack of research using RNNs and the temporal nature of hydraulic data, we proposed the application of gated RNNs for leak detection in WDNs. A possible reason that RNNs have not been tested before, might be twofold; They require more data to train since they have more weights than MLPs, and MLPs has been shown not to work, causing researchers to focus on other, non NN, methods. However, as more data from WDNs have been gathered and tools to generate artificial data from historical data have been created, it has become possible to create and train larger networks, as RNNs, for leak detection. The leak detection system using a Gated RNN would not need several models for each time step, as the RNN should be able to handle the diurnal pattern. Furthermore, the leak detection method would be part of the system, so the threshold tuning used by other leak detection systems would be automated.

4.1 Frameworks

4.1.1 Pytorch

For easy and quick prototyping of different network structures, we choose to use PyTorch(PyTorch Community, a) as the deep learning framework. Other popular deep learning frameworks considered were TensorFlow and Keras(Migdal and Jakubanis, 2018; Abbass et al., 2018). All three support gated RNNs, and while TensorFlow was more commonly used, it was not chosen as the other two frameworks had a simpler API more fitting for prototyping. There was little difference between Keras and PyTorch, but the Object Oriented Programming(OOP) interface of PyTorch made it easier to both debug while developing and customise. While Keras required fewer lines of code(Migdal and Jakubanis, 2018), and was, therefore, quicker to prototype in, this results in fewer ways to customise the code. As the focus of this thesis was to test and possibly customise networks to solve a problem, together with our prior experience with the framework, PyTorch was chosen.

4.1.2 LeakDB

Vrachimis et al. (2018) presented a benchmarking tool for leak detection algorithms written in Python and Matlab. The benchmarking was done by creating a data set of simulated hydraulic

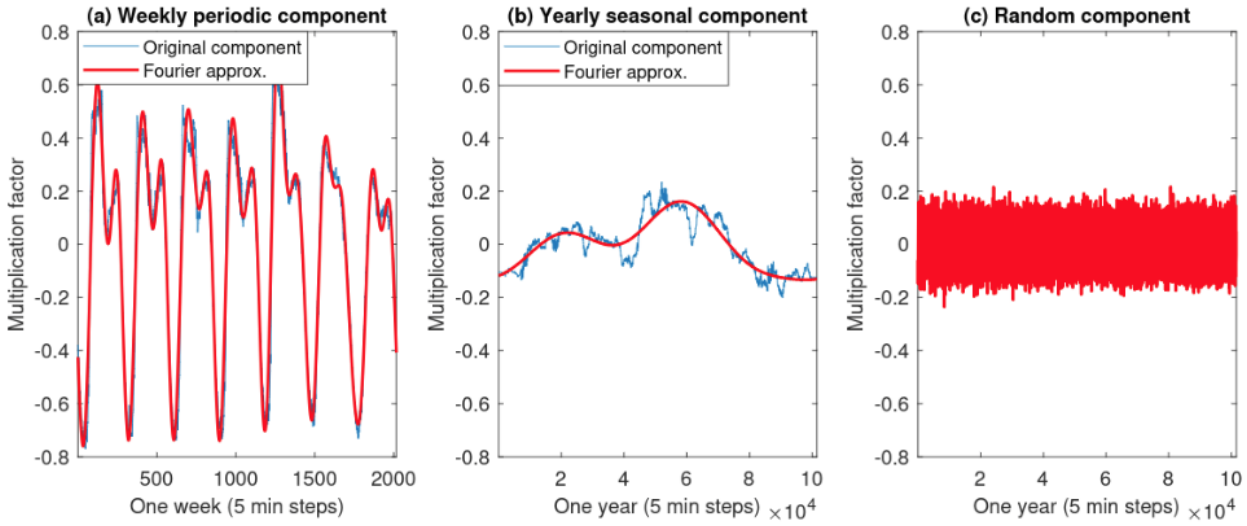


Figure 4.1: The three components of a decomposed demand pattern, showing both the original component and the Fourier approximation. (a) Shows the weekly pattern, (b) the yearly component and (c) the random component.

Source: Vrachimis et al. (2018)

data from WDN, called *scenarios*, and testing the algorithms on these scenarios. The data generation was based on historical data and implemented in Python and used the Water Network Tool for Resilience(WNTR), a python library for simulation and analysis of WDNs(Klise et al., 2018). WNTR was compatible with EPANET which was widely used to analyse and simulate WDNs(EPANET; Soldevila et al., 2016b; Deniss et al., 2012; Casillas Ponce et al., 2013).

The LeakDB algorithm used an EPANET `.INP` file that described a WDN, an example can be viewed in Appendix B, and created a new WDN for each scenario in the dataset. The parameters(i.e., pipe length and diameter) of each new WDN was changed based on an uncertainty value, μ, which changed the original parameter with $\pm\mu$% of its original value. In LeakDB μ was set to 25, while it was set to 0 while generating data for this thesis as the leak detection models should be trained for a specific WDN, where the only uncertainty was in the pressures, flow and demand data.

WNTR needed a demand pattern to simulate the pressure and flow at each node and edge in the WDN. A demand pattern can be decomposed into three components; a weekly component, a yearly component, and a random component(Soldevila et al., 2016b; Eliades and Polycarpou, 2012; Vrachimis et al., 2018). LeakDB approximated each of these components to create the demand pattern used by WNTR. The weekly and yearly components were created by approximating historical data using Fourier series and represented the weekly and yearly demand trends as in Soldevila et al. (2016b); Eliades and Polycarpou (2012). The random component represented random events in the network, as unpredictable consumer demand and maintenance, and was a normal distribution with zero mean and a standard deviation of $\epsilon = 0.33$. Figure 4.1 showed an example of each of the components. To generate different patterns for each scenario, the coefficients of the Fourier series approximation was randomised in a range of $\pm$10% of their original value.

The leaks were simulated at the WDN nodes by increasing the demand, and the leak was placed at a node that was randomly chosen in the network at a random time interval. The demand increase was based on the diameter of the hole, $\phi \in [2cm, 20cm]$, and the pressure at the network node. There were two types of leaks simulated; abrupt leaks and incipient leaks. Incipient leaks were small and increased over time; these leaks were the hardest to detect as they may be mistaken for changes in the water network, change in consumer demands or other events that cause a natural change on the sensor data. The leaks were originally given in m^3/h; this was converted to l/s to concur with the notation used in literature.

When generating a data set LeakDB also used the following parameters; the number of scenarios to generate, start and end date for the scenario, the measurement frequency and how many scenarios would be simulated in parallel. Each scenario was stored separately, with folders for flow, demand and leak, and where each nodes sensor readings were stored in a separate `.csv` file. For each measurement, there was a time stamp and value. The added demand of the leaks was stored together with time stamps, and the labels were stored as either 0 or 1 indicating a leak together with the specific time step. An example can be found in Appendix C.

As mentioned above, only parts of the LeakDB code was used; this was specifically the `leakDBGenerator` and `demandGenerator`. `leakDBGenerator` generated the scenarios and started the WNTR simulations, and `demandGenerator` generated the demand pattern used by WNTR from Matlab files. The code of leakDBGenerator used removed API calls for WNTR and Pandas, a data structure and analysis tool for Python, which had to be updated for the code to function. This was done by replicating the old API calls using newer API calls to not introduce unknown bugs to the code.

During training and testing, it was discovered a bug when creating the incipient leaks. These leaks were implemented by slowly increasing the hole of the leak between two-time stamps, the leak start and the peak leak time. From the peak time until the end time this hole should remain unchanged and there should be a continuous leak. The continuous leak from the peak time until the end time was however not created, but the labels were, creating miss labeled data after each incipient leak. The bug was corrected by extending the leak to the end time. The bug was discovered and corrected late in the work on this thesis and therefore affected the results of the first iteration.

LeakDB was implemented with the Python `multiprocessing` package to speed up the generation process by utilising parallelism. However, the implementation did not take into account that each sub process running in parallel inherited the same random seed, in effect causing each simultaneous sub process to generate the same scenario. This was discovered during the debugging of the previous bug, so this also affected the first experiment as well. Four sub-processes were used in creating this data set, so of the 200 scenarios generated, only 50 of them were unique. As this work did not require the computational boost offered by parallelism, it was fixed by setting the number of sub processes used to one.

The effect of the bugs on the experiments in this thesis were discussed in Sections 5.6 and 6.2. It should also be mentioned that not all scenarios generated by LeakDB were suited. There could be scenarios with unrealistically short leaks, e.g., a leak duration of 15 minutes since the start and end time stamps were chosen at random. These were abnormalities, so they were removed only when discovered.

4.2 Data Sets

The data sets used in this thesis was generated using LeakDB, as explained in Section 4.1.2, and the two WDNs Hanoi and Net1. LeakDB simulated the hydraulic data of the WDN and generated the flow, pressure, and water demand every 15 minutes over 30 days, one such simulation was named a scenario. This measurement frequency was chosen because it was the standard in Norway, while in the literature this may vary with frequencies between 5 and 60 minute with most being either 15 or 30 minutes. Figure 4.2 shows example sensor readings at each node in the Net1 WDN for ten time steps.

As mentioned in Section 2.2, it was unrealistic to assume that sensors were placed at each junction. However, this work focused on testing a ML-approach and comparing it to other relevant work. Since the work in Section 3.2 uses all the sensors, this was done in this work aswell.

Both pressure and flow data could be used for leak detection, but since it was unreasonable to require both flow and pressure data, one of the two were chosen. As most of the methods in Section 3.2 used pressure data this was chosen. Each sensor value, x_i was normalised at each time step using the following formula:

$$x_i = \left| \frac{x_i}{\max|\vec{x}|} \right|$$

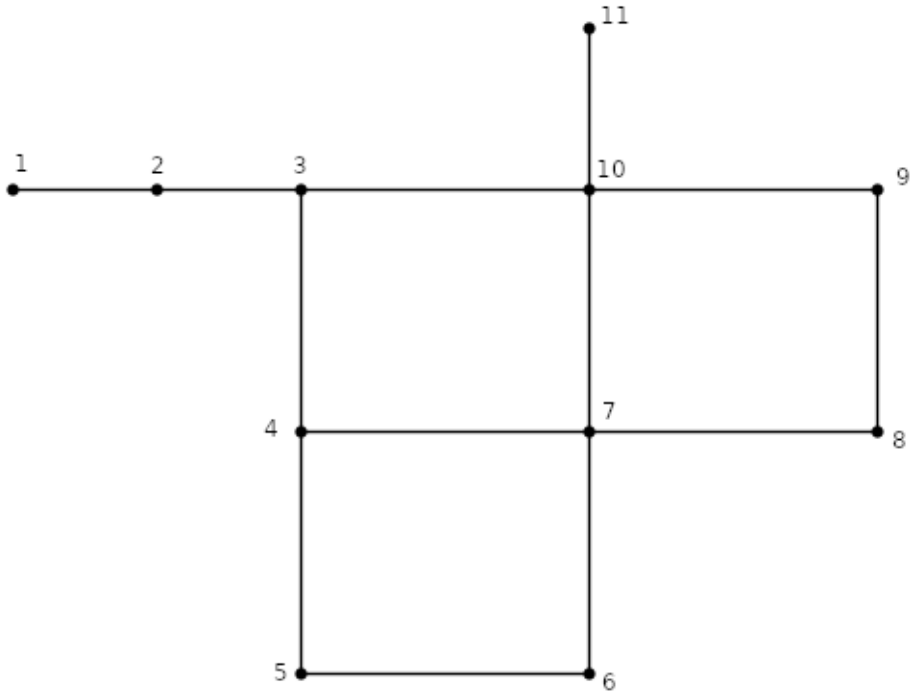
Where $\vec{x}$ was the sensor value at a specific time step, in other relevant papers the max value was chosen from all the sensor values within a given time window. Both methods were tested, and it was found that it had little difference, so to reduce the amount of data pre-processing normalisation was done per time step. In a live leak detection system, this would remove the need to store a time window of previous measurements.

Each time step was labeled as either a leak, 1, or a not a leak, 0. So the leak detection system would have an input size equal to the number of pressure sensors in the WDN and an output size of 1. The inputs would range from 0 to 1 because of the normalisation, and the output should be 0 or 1.

There was an overweight of nonleak training data in the data sets since LeakDB generated scenarios without leaks and leaks did not usually appear for more than 50% of a scenario. Table 4.1 shows the underweight of leak labels in the data sets. As explained in Section 5.5.1 this caused some models to classify every input as a nonleak as this gave the system accuracy of around 70% on average. Two possible solutions to this balancing issue were tested, removing all nonleak scenarios from the training sets and weighting the loss function. Removing the nonleak scenarios was enough on the Net1 WDN, but not for the Hanoi WDN. The weighted loss function was therefore implemented and tested for that WDN and will be explained later in Section 4.3.3.

4.3 Leak detection system

As mentioned in Section 2.3.3, RNNs were specialised for sequential data, such as the sensor data from a WDN, and Gated RNNs can learn what temporal data to store. Because of this, we chose to build a leak detection system based on Gated RNN architectures, where a single model would learn the temporal information necessary for leak detection, avoiding the need for several



Sensor 1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Sensor 2	79.271	79.691	80.137	80.635	81.153	81.531	81.967	82.359	82.692	83.012
Sensor 3	90.199	90.576	90.924	91.314	91.716	92.025	92.378	92.67	92.916	93.207
Sensor 4	84.438	84.867	85.263	85.707	86.164	86.515	86.918	87.249	87.53	87.86
Sensor 5	82.346	82.722	83.109	83.515	83.936	84.365	84.793	85.216	85.627	86.022
Sensor 6	83.799	84.191	84.6	85.03	85.458	85.888	86.315	86.735	87.142	87.507
Sensor 7	83.732	84.167	84.693	85.21	85.729	86.123	86.539	86.878	87.155	87.452
Sensor 8	83.893	84.289	84.714	85.172	85.63	86.053	86.481	86.882	87.254	87.597
Sensor 9	85.331	85.725	86.155	86.618	87.075	87.495	87.928	88.332	88.709	89.046
Sensor 10	83.015	83.532	84.036	84.604	85.162	85.533	85.967	86.313	86.583	86.824
Sensor 11	36.576	36.948	37.332	37.733	38.151	38.58	39.008	39.434	39.85	40.25

Figure 4.2: Example of pressure measurements from the sensors of Net1. Each column was the sensor readings at a specific time step.

Data set / WDN	Total	Total without non Leak Scenarios	Training Set
Net1 1	26.7%	34.9%	15.9%
Net1 2	20.5%	30.1%	32.3%
Hanoi	22.5%	28.6%	28.4%
Hanoi(Only large leaks)	22.5%	28.6%	36.1%

Table 4.1: The percentage of leak labels in the data sets, showing the imbalance between leak and non leak data in the data sets.

models trained on specific time periods as was done in other related work(Mounce, 2013; Ye and Fenner, 2011; Herrera et al., 2010).

To detect a leak, a linear layer was added after the Gated RNN. This would take the output of the Gated RNN as input and decode final hidden state into a single output, where 0 indicated a nonleak, and 1 a leak. This approach resembled the model-baed methods were the Gated RNN would "model" the WDN and the linear layer would do the leak detection. The difference was that in this system, the model and detector were connected, so both were trained together, increasing the synergy between the two parts of the leak detection system.

The leak detection system can be summarised as follows, and an overview was shown in Figure 4.3. The input to the system was a feature vector with the latest sensor reading of each sensor. The output of each layer of the gated RNNs was fed as input to the next layer. The output of the final RNN-layer was used as input to a linear layer to map it to a single output. The sigmoid function was applied to this single output to get a value between 0 and 1. This linear layer together with the activation function worked as a decoder of the output and as it was a part of the implemented model would be trained to output a score close to 0 and 1, automating the threshold tuning mentioned in Mounce (2013). During testing, this final output was binarised by rounding the output,

$$class(x) = \begin{cases} 0 & \text{if } x < 0.5 \\ 1 & \text{otherwise} \end{cases}$$

The leak detection system was implemented without any handling of faulty sensors, as this was viewed to be outside of the scope of this initial proof of concept work.

4.3.1 Gated RNN

Since there was not a big difference between LSTMs and GRUs, both Gated RNNs were tested on the smaller Net1 WDN to identify which were more fitting. The Gated RNNs were implemented using their respective PyTorch modules(PyTorch Community, b,c) and example code for each implementation can be viewed in appendix A. After testing both Gated RNNs, it was decided to use GRUs for the final leak detection system trained on the Hanoi WDN because the slightly better results of the GRUs shown in Section 5.5.2 and a GRU has fewer parameters to train than an LSTM.

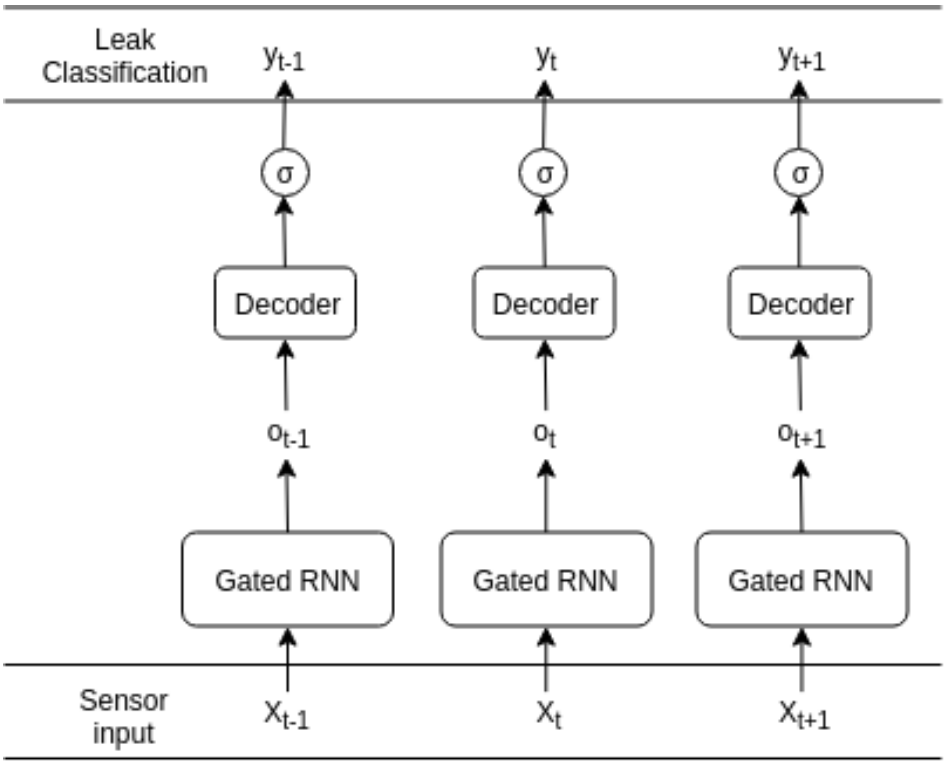


Figure 4.3: An overview of the leak detection system. x_t was the pressure readings which in this work was from the generated data set, but in a real world setting would be input from the monitoring system of a WDN, o_t was the output of the Gated RNN, *Decoder* was the final linear layer, σ was the sigmoid function and y_t was the systems calssification of the input, x_t, and a value between 0 and 1.

4.3.2 Hyper Parameters

To test which architectures would work, different hyper parameters were tested. Both LSTMs and GRUs used the same hyper parameters and the ones tested were listed in Table 4.2. As the most mentioned limitation for data-driven leak detection methods was the amount of data needed to train the models, the hyper parameters were chosen to be small to lower the amounts of weights needed, and therefore the amount of data needed. The parameters were chosen in an attempt to identify which parameter had an effect on the leak detection system, and what that effect was. Specifically, the hidden state size, as we wondered if it would be a correlation between the hidden state size and the size of the WDN. Adam(Kingma and Ba, 2014) was chosen as an optimiser with parameters $\beta \in (0.9, 0.999)$. As it was an adaptive learning-rate optimiser it was chosen to avoid having to manually tune the learning rate, but the initial learning rate had to be chosen. Trough initial testing, it was found that using a learning rate greater than 0.003 caused the system to stagnate at a local optima and label everything as a nonleak. In the first experiments on Net1 dropout was not used, as it cannot be used on one layer architectures and would have resulted in an advantage to the architectures with more layers. However, on the final experiment on Hanoi, it was added with a dropout chance of 0.3 to help the models generalise.

Hyper Parameter	Value
Number of Layers	1, 2, 3
Hidden state size	5, 10, 15, 20
Learning rate	0.003

Table 4.2: The hyper parameters chosen for the initial test on the Net1 WDN.

Hidden State Initialisation

The first hidden state of an RNN was one of the parameters to initialise. h_0 was a matrix and can be initialised in different ways, where the two simplest ones were to fill it with either random values or a specific value. There were more complex methods of finding the optimal h_0 trough machine learning, but this was viewed as over complicating the problem as the leak detection system should run for several time steps, so the initial h_0 should be of little consequence. Of the two simple methods initialising h_0 to a matrix of random numbers from the standard normal distribution was chosen. This was done so that the system would learn to be indifferent to the initial state, as it just contained noise(Zimmermann et al., 2012).

Leak Classification

A fully connected linear layer was used to decode the final output of the Gated RNNs to get one or two outputs, depending on how the output was described. The output of the linear layer was then sent through a sigmoid function to ensure that the output value was between 0 and 1, where 0 meant that the input was classified as a nonleak and 1 a leak.

4.3.3 Loss function

Cross-entropy was chosen as the loss function for the first two iterations of the leak detection system tested on the Net1 WDN. Both the Mean Absolute Error (MAE) and Mean Squared Error (MSE) were considered, but both loss functions caused the models to stagnate and classifying every input as a nonleak. These functions were more suited for tasks where the outputs were real-valued and not classifications. Cross-Entropy had not this problem and was therefore chosen.

The leak detection system constructed to be used on the Hanoi WDN used the cross-entropy loss function, showed in Equation 4.1, where l and nl denotes the leak and nonleak classes, y_c was a binary indicating if, c was the correct classification and x_j was the system's classification of that class. When there were only two classes, the Equation can be rewritten to be the BinaryCrossEntropy, where x was the output, and y was the correct class.

$$\text{CrossEntropy}(\vec{x}) = -\left(\sum_{c \in l,nl} y_c \log x_c\right) \tag{4.1}$$

$$\text{BinaryCrossEntropy}(x) = -\big(y \log x + (1-y)\log(1-x)\big) \tag{4.2}$$

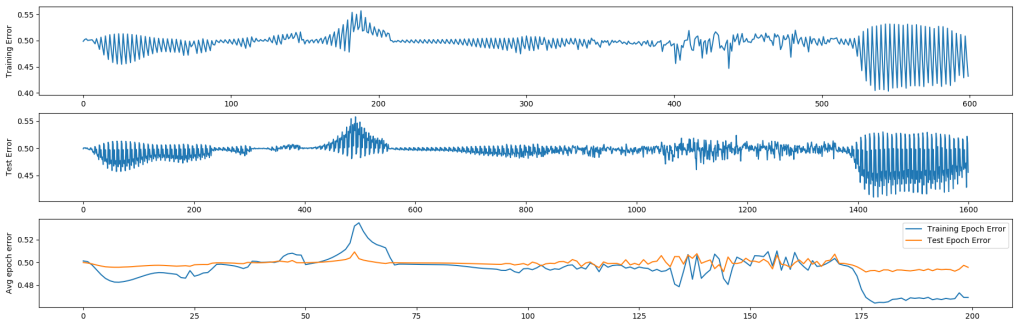
Cross-entropy with weighting on the classes was tested to combat the imbalance in the data set. The weighted version of the loss function was shown in Equation 4.3 where w_l was the

weight for the leaked class and w_{nl} was the weight for the nonleak class. PyTorch only implements the cross-entropy function with weights, not binary cross entropy.

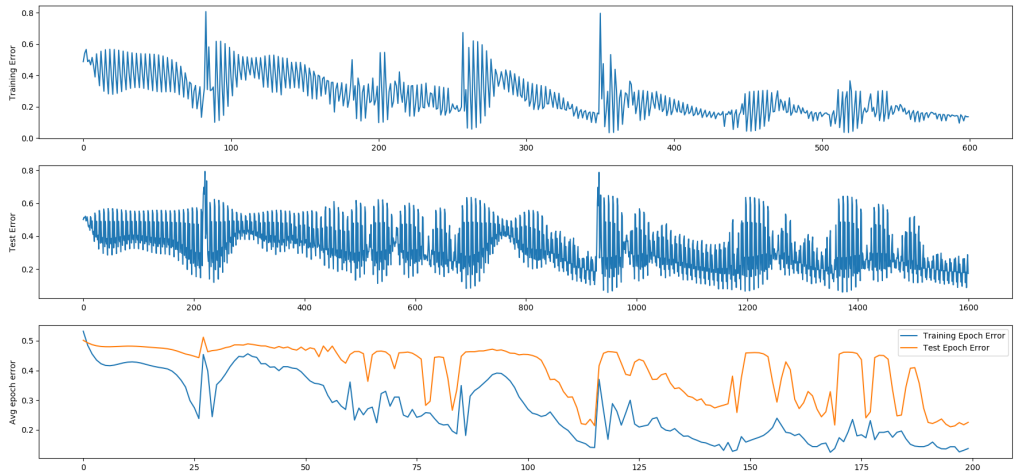
$$\text{WeightedCrossEntropy}(x) = -\left(\sum_{c \in l,nl} w_c y_c \log x_c\right) \tag{4.3}$$

So to use weighted cross entropy, the output, y_t, of the final linear layer was converted to a score for each class, $(y_t, 1 - y_t)$. Using two output nodes of the final layer would have worked as well, but this would give the weighted cross entropy and advantage because of the slightly larger output layer.

The weights were tested with $(w_l, w_{nl}) \in \{(0.32, 0.68), (1, 1.47)\}$, where both were based on the percentage of imbalance in the training set, the first being the percentage and the second were both weights were multiplied by 1.47 to see if a higher loss overall would have any effect. The training set used consisted of 3 scenarios were 32% of the data set was labeled as leaks. A smaller training set was used to easier identify possible problems with specific scenarios using the weighted loss function. The weighted cross entropy was compared with the binary cross entropy shown in Figure 4.4. When tested it was clear that the weighted cross entropy did not converge, but the binary cross entropy did.



(a) Weighted Cross Entropy



(b) Binary Cross Entropy

Figure 4.4: Training graph of weighted and not weighted cross entropy. The weighted cross entropy model stagnated with an avg epoch error of 0.48, while the binary cross entropy managed an avg epoch error of > 0.2.

Chapter 5: Experiments & Model Improvements

5.1 Water Distribution Network

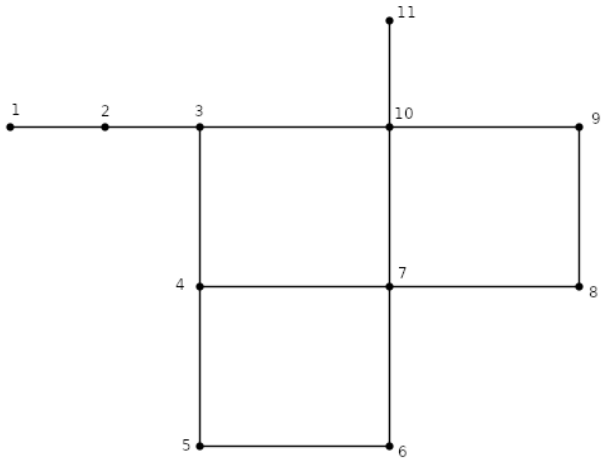
The training and test data used were generated using the updated code of LeakDB explained in Section 4.1.2 using the Hanoi and Net1 WDNs depicted in Figure 5.1. The Hanoi network was a real WDN consisting of 32 node, each having a pressure sensor. It was regularly used in the literature for testing leak detection systems. The network has two different representations in literature, Figure 5.1b shows the representation used in this thesis. Net1 is a small example WDN from EPANET consisting of 11 nodes. This was used to test the leak detection system on a lower scale before testing it on the Hanoi network.

5.2 Data

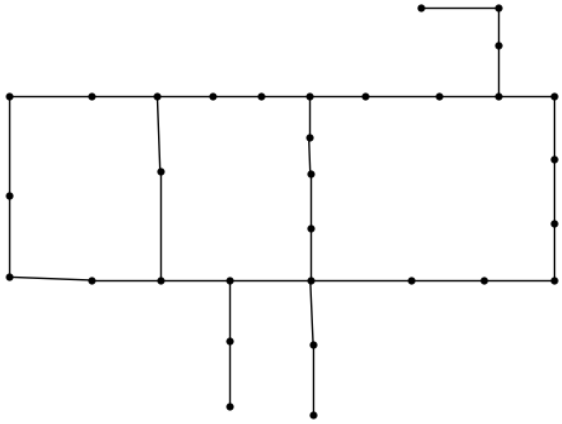
The data was generated using LeakDB and the pressure data was read and normalised as described in Section 4.2. The sensor data was stored without the normalisation, and rather normalised before it was fed to the leak detection system to simulate how real world input would be given. The system takes sensor readings from each sensor at each time step as input, as shown in Figure 5.1, and has an output at each time step between 0 and 1 during training and binarised during testing.

Two data sets of 200 and 85 scenarios were generated for Net1 and a data set of 400 scenarios were created for Hanoi. Each scenario lasted from the 1st until the 30th of January and had sensor reading every 15 minutes, so each scenario contained 2880 data points. Generating every scenario within the same time frame was done to avoid any problems that might be caused by training the leak detection system on data from different seasons, as that is out of scope for this thesis. This should not give the system any advantages compared to other related work, as they also use data spanning consecutive weeks.

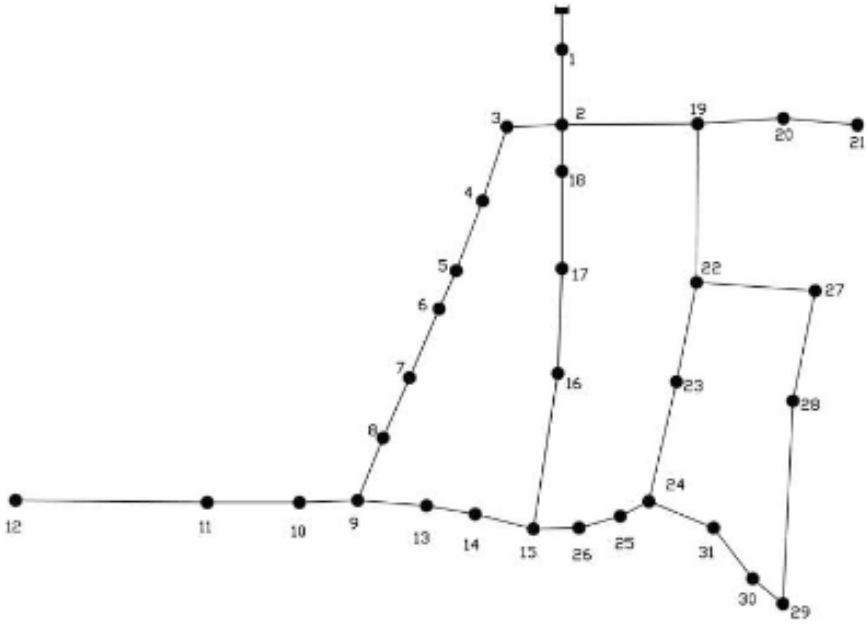
A scenario could contain from 0 to 2 leaks of different types, and leaks could overlap. All of the scenarios would not be used, but the amount ensured that there was a large enough pool of scenarios to choose from when constructing the training and test data sets. As mentioned in Section 4.1.2, not all scenarios were realistic, and when any abnormal scenario was discovered, they were replaced. Scenarios were chosen for the training and test sets at random, but it was quickly discovered that this lead to a imbalance in the data sets. This was done to combat the imbalance problem mentioned in Section 4.2, to ensure that the leak detection system was ex-



(a) Net1



(b) Hanoi



(c) Hanoi

Source: Casillas Ponce et al. (2013)

Figure 5.1: The Net1 and Hanoi water distribution networks. The nodes represent pipe links and lines are pipes. The Hanoi network is represented differently in the literature, (b) is the representation used in this thesis.

posed to different sized leakages and to remove unrealistic scenarios(leaks with a duration of less than 1 day) from both the training and test sets. On the Net1 data sets, 50 scenarios were first chosen at random, creating a training set of 40 scenarios and a testing set of 10 scenarios. Then the training set was sanitised by replacing non leak and unrealistic scenarios with random scenarios. Only the unrealistic scenarios were replaced in the training set, as the system had to be tested using non leak scenarios as well.

Sensor 1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Sensor 2	0.7338	0.7330	0.7275	0.7220	0.7157	0.7057	0.6962	0.6867	0.6781
Sensor 3	0.5959	0.5949	0.5873	0.5791	0.5709	0.5566	0.5447	0.5314	0.5210
Sensor 4	0.5608	0.5597	0.5508	0.5422	0.5317	0.5168	0.5034	0.4906	0.4781
Sensor 5	0.6853	0.6867	0.6962	0.7057	0.7165	0.7340	0.7504	0.7668	0.7817
Sensor 6	0.5100	0.5094	0.5007	0.4909	0.4792	0.4618	0.4458	0.4360	0.4175
Sensor 7	0.3809	0.3802	0.3729	0.3655	0.3565	0.3440	0.3328	0.3201	0.3090
Sensor 8	0.4317	0.4313	0.4237	0.4157	0.4067	0.3933	0.3827	0.3689	0.3579
Sensor 9	0.4694	0.4695	0.4597	0.4462	0.4342	0.4135	0.3959	0.3819	0.3653
Sensor 10	0.3156	0.3141	0.3031	0.2907	0.2773	0.2586	0.2438	0.2249	0.2077
Sensor 11	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Classification	0.17	0.17	0.17	0.22	0.35	0.59	0.82	0.91	0.94

Table 5.1: Example of inputs and classifications from the leak detection system using data from Net1. Each column is a input vector containing normalised pressure readings and the bottom cell is the prediction of that vector. Only predictions above 0.5 were identified as leaks during testing.

When testing the leak detection system on the Hanoi WDN the data was split into categories based on the leak sizes used in the related works in Section 3.2. From these categories, the test and training sets were constructed so that they would cover every leak size equally. The 400 scenarios in the data set were split into four categories:

- Small leak scenarios; Scenarios with leak sizes less then $40l/s$
- Medium leak scenarios; Scenarios with leak sizes between $40l/s$ and $80l/s$
- Large leak scenarios; Scenarios with leak sizes above $80l/s$
- Non leak scenarios

Some scenarios fit multiple categories as they had several leaks. As long as the leaks did not overlap it was categorised as both. If they did overlap the scenario was categorised based on the largest leak because there was no way to identify which leak the leak detection system reacted to, but it is assumed to be the largest of the two, as will be noted in Section 5.5.2.

5.2.1 Training set

The training sets created for the two experiments on Net1 contained 40 of the generated scenarios which was generated with different demand patterns. Using 40 scenarios instead of a single scenario as the methods found in Section 3.2 was done in an attempt to make the leak detection system indifferent to non leak related changes in demand and to expose the system to different leaks types, sizes, lengths and leak locations.

It was also noted in Section 5.5.1 that the leak detection system was not able to classify the smaller leak, it was therefore attempted to increase the number of smaller leak scenarios in the training set. For the Hanoi training set 13 scenarios from each leak category were first chosen for the training set.

5.2.2 Test set

The test set was constructed to contain leaks of different sizes and scenarios without leaks. The different sized leaks would identify any problems with detecting leaks of a specific size, while the non leak scenarios would ensure that the leak detection system did not classify false positives. If false positives became a problem, it could be solved by only counting the classification after several consecutive classifications had been done, as done in the other relevant work. This would lower the false positive rate (FPR) at the expense of increasing the detection time (DT).

The first test set consisted of 10 scenarios listed in Table 5.2. Scenario 135 contains a relatively small leak compared to the others and would identify if the models were able to detect small leaks.

Scenario	Leak Type	Peak Size(l/s)	Duration(d, hh:mm)
2	Incipient	76	(15, 13:00)
10	Abrupt	63	(10, 18:30)
17	Abrupt	139	(11, 22:30)
25	Incipient	128	(1, 4:00)
40	-	-	-
86	Abrupt	94	(17, 05:00)
127	Abrupt, Incipient	155, 107	(0, 15:30), (12, 15:30)
134	-	-	-
135	Abrupt	15	(4, 8:45)
191	-	-	-

Table 5.2: The test cases of the first iteration. "-" indicates that there were no leaks.

The data sets on Net1 were regenerated after the first iteration because of the bugs mentioned in Section 4.1.2, and the new test set can be viewed in Table 5.3. The test set contained two non leak scenarios, one small leak(the first leak in scenario 83), abrupt, incipient and both abrupt and incipient leak scenarios.

Scenario	Leak Type	Peak Size(l/s)	Duration(d, hh:mm)
3	-	-	-
4	-	-	-
70	Incipient	85	(7, 8:15)
73	Abrupt, Incipient	115, 25	(12, 4:15), (2, 4:00)
76	Abrupt	142	(6, 13:45)
77	Incipient	93	(2, 18:15)
81	Abrupt	33	(1, 01:15)
82	Abrupt	97	(7, 15:30)
83	Abrupt, Incipient	16, 114	(0, 18:45), (12, 16:45)
84	Incipient	58	(7, 10:30)

Table 5.3: The test cases of the second iteration. "-" indicates that there were no leaks.

The test set generated for the last experiment on the Hanoi Network was presented in Table 5.4. It was constructed by chosen 3 of every leak category and two non leak scenarios.

Scenario	Leak Type	Leak Category	Peak Size(l/s)	Duration(d, hh:mm)
309	-	-	-	-
366	-	-	-	-
20	Abrupt, Incipient	Small & Large	15, 205	(11, 17:45), (0, 6:15)
177	Incipient	Small	24	(3, 0:15)
379	Abrupt	Small	22	(3, 9:45)
13	Incipient	Medium	56	(5, 21:15)
157	Incipient	Medium	73	(28, 11:15)
236	Abrupt, Abrupt	Medium & Large	61, 147	(1, 18:15), (7, 19:30)
114	Abrupt	Large	223, 114	(7, 05:00), (8, 22:15)
257	Incipient, Incipient	Large	127, 328	(10, 7:15), (1, 05:30)
324	Abrupt	large	94	(8, 06:45)

Table 5.4: The test cases of the Hanoi iteration. "-" indicates that there were no leaks.

5.3 Metrics

The performance of the leak detection system was evaluated using benchmarks found in literature so the system could be compared with other leak detection systems. The following benchmarks were tracked:

- *True positive rate (TPR)* indicated how many of the data points labelled as leaks in the test set are classified correctly by the leak detection system.
- *False positive rate (FPR)* indicated how many of the data points labelled as non-leaks were misclassified as leaks by the leak detection system.
- *Detection time (DT)* was the time between a leak starts and when it was detected by the leak detection system.
- *Accuracy* how many of the data points were labelled correctly by the leak detection system.

True positive rate (TPR) and FPR were used in much of the literature, both in machine learning and leak detection. DT was specified in Wu and Liu (2017) and the measurement could be calculated from several other papers as it was usually mentioned, although not specified as detection time.

When calculating TPR, FPR, accuracy and DT, the output was binarised by rounding the output of the leak detection system to get either 0 or 1, where 1 indicated a leak. The binarisation was not used during training, as this would affect the error-term used in gradient decent.

5.4 Approach

The leak detection system went through two experiments with using the Net1 WDN and one using the Hanoi WDN. In the first two experiments the system was tested using the hyper parameters described in Section 4.3 to identify which hyper parameters had an affect on the performance of the system. Previous attempts at using ANNs trained their system on a single scenario,

splitting it into train and test data. This caused the system to know only one demand pattern, but as noted in Section 2.2, a change in demand was a normal event that could be mistaken for a leak. As LeakDB generates scenarios with different demand patterns, using several different scenarios in the training set might make the system indifferent to these normal demand changes.

The leak detection system was was trained for at least 50 epochs for each hyper parameter, where one epoch contained all the scenarios. If the avg error on the test set did not stagnate, it would then be run until it stagnated. This was done to avoid under fitting the implementation with more weights. Over fitting did not appear as an issue on 50 epochs. The hidden state was reset at the beginning of each scenario to avoid information from the previous scenario to be transmitted over to next scenario. Each epoch trained and tested the system on 40 and 10 scenarios, totalling 115200 and 28800 data points.

As mentioned in the previous section, this approach did not work on the Hanoi data set, which therefore was constructed differently ending with two training sets of 39 and 40 scenarios, one with 13 from each leak category and one with 40 leak scenarios with a leak size larger than $80 l/s$.

5.5 Results

5.5.1 Net 1: First iteration

A summary of the results of the first iteration can be found in Table 5.5. The accuracy and FPR is the average of the 10 test scenarios and TPR is the average of the 7 test scenarios with leaks. In addition to the FPR, TPR, accuracy and DT, five things of note was identified during the testing.

1. The average error of the test set was greater then the average error of the training set at each epoch. This was credited to the difference in the data sets, and implied that the training set contains more difficult scenarios.

2. Some architectures stagnated while learning after classify every input as 0. The specific architectures were the ones with no detection time and 0% TPR in the result tables.

3. None of the architectures managed to classify scenario 135.

4. The models had a low TPR relative to other leak detection methods mentioned mentioned in Section 3.2, even tough the accuracy was high.

5. Both GRU and LSTM architectures classified consistently, i.e. once a input was classified it did not change for some time.

5.5.2 Net1: Second iteration

The results of the second iteration are summarised in Table 5.6. Of the points mentioned in the first iteration number 1 and 5 still applied. Scenario 135 was replaced, but none of the architectures managed to detect the leak in scenario 81 and the first leak in scenario 83. So the

GRU						
Number of layers	Hidden state size	Epochs	TPR	FPR	Accuracy	DT
1	5	50	68.03%	0.53%	95.62%	5.00
	10	50	74.29%	2.07%	95.42%	3.00
	15	50	77.43%	1.52%	95.63%	1.86
	20	60	75.61%	3.39%	95.53%	4.29
2	5	50	0.00%	0.00%	75.69%	-
	10	50	71.53%	2.30%	95.39%	3.14
	15	50	74.17%	3.58%	95.33%	4.14
	20	50	78.79%	1.57%	95.70%	2.43
3	5	60	68.86%	0.78%	95.61%	1.86
	10	60	78.66%	5.95%	94.45%	3.57
	15	70	74.47%	2.79%	95.31%	1.43
	20	90	56.01%	2.09%	89.74%	233.43

LSTM						
Number of layers	Hidden state size	Epochs	TPR	FPR	Accuracy	DT
1	5	60	73.40%	1.20%	95.53%	3.14
	10	50	67.99%	0.43%	95.60%	7.43
	15	70	69.73%	0.71%	95.61%	3.57
	20	70	76.09%	2.26%	95.23%	4.14
2	5	70	75.26%	5.07%	94.25%	2.86
	10	70	67.30%	0.25%	95.64%	8.57
	15	60	75.26%	1.28%	95.50%	3.86
	20	60	67.84%	0.42%	95.68%	3.29
3	5	50	0.00%	0.00%	75.69%	-
	10	50	0.00%	0.00%	75.69%	-
	15	50	0.00%	0.00%	75.69%	-
	20	50	0.00%	0.00%	75.69%	-

Table 5.5: The results using different GRU and LSTM architectures with 40 training and 10 test scenarios after the first iteration on Net1. Detection time is given as an average of the number of time steps it took before the leak was found. "-" indicates that no leak was found.

minimum leak size the leak detection system was able to detect lies between $33 l/s$ and $58 l/s$. The incipient leak in scenario 73 was smaller than this, but it happened during the bigger abrupt leak, suggesting that bigger leak was the one being detected.

5.5.3 Hanoi

The leak detection system was trained using scenarios from every category, but it was discovered that the system could only detect leaks greater than $80 l/s$, the training set was changed to only contain large leaks. This will be explained further in Section 5.6.3. The results are summarised in Table 5.7. Because the system was unable to detect leaks using pressure, flow data was also

GRU						
Number of layers	Hidden state size	Epochs	TPR	FPR	Accuracy	DT
1	5	50	85.66%	0.09%	99.13%	7.43
	10	50	85.73%	0.0%	99.24%	6.57
	15	60	84.63%	0.0%	99.00%	12.00
	20	50	86.21%	0.00%	99.30%	3.43
2	5	60	86.13%	0.00%	99.28%	4.86
	10	50	86.26%	0.00%	99.29%	3.63
	15	60	16.11%	0.02%	83.87%	1098.00
	20	70	86.44%	0.02%	99.31%	2.71
3	5	120	13.96%	0.00%	83.19%	-
	10	50	0.00%	0.00%	80.51%	-
	15	100	0.00%	0.00%	80.51%	-
	20	50	0.00%	0.00%	80.51%	-

LSTM						
Number of layers	Hidden state size	Epochs	TPR	FPR	Accuracy	DT
1	5	50	85.91%	0.11%	99.16%	6.14
	10	50	85.74%	0.04%	99.17%	7.00
	15	50	85.66%	0.27%	99.03%	7.43
	20	60	85.79%	0.36%	98.98%	6.71
2	5	50	85.33%	0.08%	99.10%	9.43
	10	50	76.63%	0.02%	97.58%	9.43
	15	50	86.24%	0.21%	99.14%	3.50
	20	60	86.24%	0.03%	99.26%	4.0
3	5	50	85.90%	0.22%	99.07%	6.71
	10	70	85.58%	0.02%	99.18%	3.71
	15	50	86.31%	0.08%	99.24%	3.57
	20	60	0.00%	0.00%	80.51%	-

Table 5.6: The results using different GRU and LSTM architectures with 40 training and 10 test scenarios after the second iteration on Net1. Detection time is given as an average of the number of time steps it took before the leak was found. "-" indicates that no leak was found.

tested. With flow data, the system managed to detect leaks medium sized using the original 39 scenario training set.

5.6 Evaluation

5.6.1 Net1: First Iteration

After the first iteration of training and testing the leak detection system with different hyper parameters the results were evaluated and five points of note were discovered and listed. The bug mentioned in Section 4.1.2 was discovered during the evaluation of the results of this first

Data used	Epochs	TPR	FPR	Accuracy	DT
Pressure	110	39.01%,	25.04	267.16	
Flow	175	58.66%(87.65%)	0.76%	90.68%	3.56

Table 5.7: The results from using a GRU based leak detection system with pressure and flow sensors on the Hanoi WDN. The TPR of only detectable leaks were presented in parenthesis. Detection time is given as an average of the number of time steps it took before the leak was found. "-" indicates that no leak was found.

iteration and affected several of the points from Section 5.5.1. Four of these points were worth evaluating further before implementing and verifying an improvement.

Some of the models stagnated during training, classifying everything as nonleaks. This was credited to the existence of non leak scenarios in the training set. A scenario with a leak usually has under 50% of the data points labelled as leaks, causing a imbalance in the data set. Adding several scenarios without any leakages causes this imbalance to increase remarkably, as each scenario contribute to 2.5% of the total training data. The non-leak scenarios were therefore replaced in the training set with leak scenarios in an attempt to reduce the imbalance.

Scenario 135 was the scenario with the smallest leak size of the test leaks and is viewed as a hard leak to detect. Because of this no special considerations were done to fix this, other then the previously mentioned replacements of non-leak training scenarios.

The last two points were caused by the same problem, and is credited to the LeakDB bug. The bug caused the incipient leak to have a "tail" after the peak leak time where normal data was mislabelled as leaks. This could clearly be seen when inspecting the TPR of each scenario, were the abrupt leaks had a TPR of 90-100% while the incipient leaks had a TPR of 40-85%. If the bug was the cause of the low TPR for incipient leaks it greatly increased the belief that the leak detection system was suitable to identify leaks at the same level of other leak detection methods.

The last thing of note was the generally lower FPR of the LSTM methods. This was also affected by the LeakDB bug, but by reviewing the classifications done by the LSTM and GRU leak detection systems its clear that the LSTM is more consistent in its change in classification. The GRUs had a "lagging" tail after a leak where it changed between classifications several times. The FPR in the LSTM based system were caused by the time it took for the system to change its classification. The FPR of both the GRUs and LSTMs were assumed to be caused by the system learning from the incipient leaks that normal data after a leak should be classified as a leak.

Before the next iteration both bugs in LeakDB were fixed and both the training and testing sets were regenerated. The non leak scenarios in the training set was also replaced with leak scenarios.

5.6.2 Net1: Second Iteration

From the results in Table 5.6 it was clear that the problem with stagnation was not fixed. Most of the 3 layer architectures in this iteration and LSTM architectures in the first iteration did not converge to other optima then classifying everything as non leaks. When reviewing the graphs of training, testing and average epoch error of the architectures it was clear that the leak detection system did not have a stable learning curve. An example could be viewed in Figure 5.2 and more training graphs from the system could be found in Appendix D. Because of this, the amount of

time it took to train the 3 layer architectures and no other indications to other advantages of using 3 layers, they were not tested on the Hanoi network.

The leak detection system did not detect small leaks, as the leak in scenario 81, and we identified two possible improvements to fix this. Increasing the number of small leaks in the training set and add weights to the loss function to increase the loss when a leak was miss classified. Of these two, the first was viewed as less intrusive. Weighting the loss function could cause a increase in FPR. As the FPR was low compared to the other methods there was still "room" for this, but the second option did not intuitively have this problem. Weighting the loss function might also have a positive effect on the stagnation problem as the system would have a higher loss for miss classifying a leak then miss classifying a non leak.

The previous assumptions that the LeakDB bug caused the TPR of incipient leaks to be low and the FPR to be high seemed to be correct, as the FPR is generally lower in this iteration and the average TPR of the incipient leaks are higher, as shown in Table 5.8.

Experiment	Gated RNN	Scenario	Avg TPR	Avg TPR /wo stagnated
1	GRU	2	70.6%	77.0%
		25	46.4%	46.9%
		127	85.2%	92.9%
	LSTM	2	47.0%	70.6%
		25	31.5%	47.2%
		127	57.4%	86.1%
	Total		56.4%	70%
2	GRU	70	56.1%	84.2%
		73	61.9%	91.2%
		77	57.2%	85.8%
		83	54.7%	82.1%
		84	58.3%	87.5%
	LSTM	70	87.4%	95.3%
		73	91.5%	99.8%
		77	88.5%	96.5%
		83	85.8%	93.6%
		84	88.3%	96.3%
	Total		73.0%	97.2%

Table 5.8: The average TPR of each incipient leak scenario with and without the leak detection systems that stagnated.

The results of this iteration, shown in Table 5.6, show little more of notice other than the 2 layer architectures are slightly better the the 1 layer ones and that the GRUs are slightly better than the LSTMs. Because of this the Hanoi network will only be tested using GRU models with 2 layers. The different hidden state sizes does not seem to have any affect on the results of the leak detection system either. So it was chosen to be 20 as it had the best results.

It must be noted that the leak detection system has a TPR of 99% and FPR of $< 0.5\%$ when

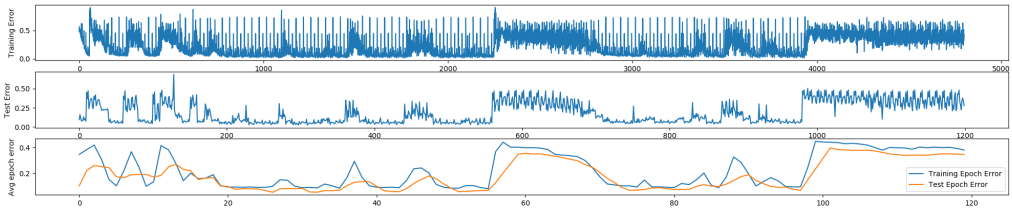


Figure 5.2: The training, test and average epoch error during training of the GRU with a hidden state size of 5 and 3 layers in the second iteration. The training and test error graphs shows the training error per scenario. The Avg epoch error graph averages the the training and testing errors per epoch.

only considering leaks of size greater than 33 l/s. In effect becoming one of the best methods for leak detection compared with the methods from Section 3.2. Most other related work also focuses on abrupt leaks, but from the results in Table 5.8 it is clear that this system was capable of detecting incipient leaks just as well as the abrupt leaks.

5.6.3 Hanoi

While training the leak detection system on the Hanoi WDN, it was discovered that the system was unable to detect the small or medium leaks on this WDN. This was confirmed by training the system several times on single small and medium leak scenario. After this was discovered, the system was trained using 40 random leak scenarios of a large size so the system could be trained and tested using the test set in Table 5.4. The results listed in Table 5.7, and the systems incapability to detect small or medium leaks in the Hanoi network, showed that the system was not applicable to the network. However, the TPR, FPR and DT of the detected leaks were very good. The system was also tested using the flow sensors of the network. The system was able to detect medium and small leaks while using flow sensors and the results shown in Table 5.7 show potential when only medium and larger leaks were counted. This was to be expected, as it was mentioned in Section 2.2, flow sensors were more prone to react to leaks, and the leaks have a larger impact on the normalised values, as seen in Figure 5.3. It seemed that the systems problem with detecting leaks from the pressure data was caused by the diurnal pattern remaining in the data after the normalisation, as opposed to the flow data. The great change in flow caused by the leak was more abrupt than the change in pressure.

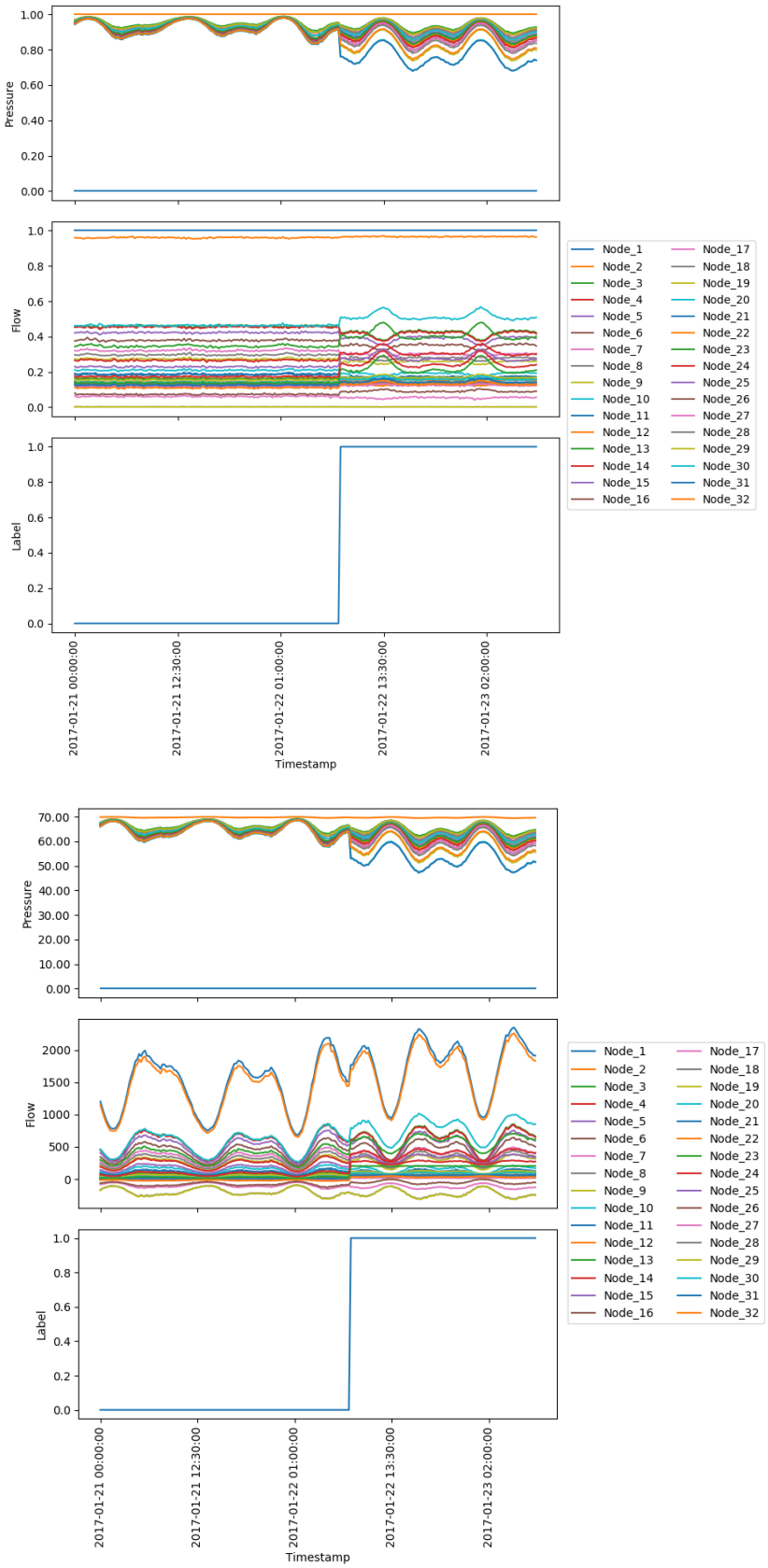


Figure 5.3: The graphs showed an abrupt leaks impact to the normalised and not normalised flow and pressure reading. The data was normalised using the normalisation described in Section 4.2.

Chapter 6: Discussion

6.1 Literature Search

As mentioned in Chapter 3, the initial search resulted in 34 candidate papers for the initial set for snowballing. Counting each papers references and citations, a maximum of 833 articles were evaluated during the snowballing process. However, many of these references and citation were overlapping, so the number is artificially large. The amount of paper reviewed ensured that we got a broad understanding of the current methods and research within the domain, answering RQ1, inspiring the leak detection system for RQ2 and finding several relevant papers to use when answering RQ3. There were few search engines used in the initial literature search which affected the start set, as the results would be biased towards the publisher's papers. This should have been avoided by increasing the number of search engines used in the initial search. The snowballing process counteracted this as we got papers from a different publisher and we are confident that the papers presented in Section 3.2 cover the state of the art within leak detection methods that utilises ML-techniques.

In hindsight, we should have spent a greater effort on a SLR instead of the snowballing process. The process took a lot of effort and time that could have been used working on the leak detection system of RQ2. The papers listed in Figure 3.2 might be found by a better SLR if more search engines where involved. Since there were few relevant papers that used other NN methods then MLP a SLR should have also been conducted focusing on finding NN solutions for other multivariate time series classification problems that might be applicable for leak detection.

6.2 The Leak Detection System

As mentioned in the introduction, more and more sensors have been placed in WDNs, leading to an abundance of data and the focus of data-driven approaches to leak detection. However, this data is not published openly, so it can not be used for research. The bugs discovered using the LeakDB framework and the lack of updates to the project might cause its credibility to be questioned, but as other papers use the same methodology to model WDNs only the original implementation could be criticised. If the bugs were discovered earlier in the work, we would have implemented the method from scratch to simplify it and have it streamlined to our implementation. Though the data is generated, it was generated based on historical data, so we confident that similar results to those presented in this work can be replicated on real sensor data. Although with worse scores as generated data usually produced better results, as pointed out by Chan et al. (2018), because of the more accurate labeling.

The data sets used during training and testing were all from January. Since hydraulic data follows a yearly trend, it was likely that the leak detection system became biased towards data of that month. Since the data was generated, we could have trained the system using data from several different time frames. However, this might introduce problems as the system would have to handle the seasonal differences in the hydraulic data. This problem was viewed as outside of the scope of this work, and should not affect the comparability of our results because related work used data from within a specific period as well.

It was unrealistic to gather as much leak data on a single WDN as was used to train the leak detection system presented in this work. However, since the data was generated based on historic data, generating this amount can be seen as realistic. It was clear that the system was dependent on the quality of the simulated WDN, but this was also true for all the model-based methods, as described in Section 2.2.3. The difference was that the model-based methods require a simulation for each specific leak size, leak node, and time step. Whereas the leak detection system presented here only required simulations of leaks for a training set.

It was clear from the final results on Hanoi that the leak detection system presented in this work did not scale well without any changes to the architecture. This was not a surprise as the number of inputs almost tripled, 11 in Net1 and 32 in Hanoi. When the system was tested on the Net1 with different architectures it was intended that the results of the different hyper parameters would give us an indication on what hyper parameters would affect the results and how they would affect them. However, this was not the case as most architectures got similar results.

When tested on Hanoi, both flow and pressure were tested as input to the network, and it was discovered that the leak detection system performed better with flow data. This was expected, but this indicates that the presented system would perform better on a DMA than in a WDN, as the DMAs use flow sensors at the input and output of the DMA. It also showed that the system had an advantage over other model-based leak detection methods, as it could use either flow or pressure measurements, while these were limited to pressure data.

The greatest advantage of this leak detection system was its capability to detect incipient leaks. Related work usually assumes that a leak was abrupt, and only discover these. This system shows potential as it has shown that it could detect incipient leaks on the same scale as abrupt leaks.

6.2.1 Comparison with related works

The results from Section 5.5.2 showed great promise. As mentioned in the section, if compared with other leak detection methods it was best when ignoring the leaks less than 33 l/s causing the TPR of the best version of the leak detection system to reach a TPR of 99% and FPR of < 0.5. Without ignoring the smaller leaks, the system still managed a respectable average TPR of 86.44%, FPR of 0.2%, accuracy of 99.31% and a detection time of 2.71 time steps(45 minutes). Compared to the results listed in Table 6.1 only the Ensemble CNN-SVM beat the leak detection system presented here. Depending on how one weighs the TPR compared to FPR, as most of the methods with higher TPR also had a high FPR, except for Ensemble CNN-SVM. When reviewing the output of our leak detection system, the false positives only appeared straight after the leak was over, and the "tail" of false negatives were shorter than 24 hours. In practice, this will, therefore, be an FPR of 0% as the system operators will know that a leak there have recently been

Paper	Method	Data Type	TPR	FPR
Chan et al. (2018)	Model-based	Historical Data	79.5%	20.5%
	Model-based	Historical Data	73.0%	0%
Chan et al. (2018)	ANN, TDNN	Historical Data	75%	0%
	MDN and FIS	Historical Data	100%	15%
	ANN, SPC, BIS	Historical Data	100%	8%
	Modified SPC	Historical Data	80%	10%
Wu and Liu (2017)	Nonlinear Kalman Filter	Simulated Data	87%	0.01%
	ANN, SPC, BIS	Historical Data	76%	10/8%
	Ensemble CNN-SVM	Engineered Test	98.2%	0.2%
	Multiclass SVM	Simulated Data	99.5%	-

Table 6.1: Results of other relevant work from Section 3.2 that present the TPR and FPR of the methods.

fixed and can ignore the leak detection straight after a fix.

Other leak detection methods used a specific threshold to decide when the output of the methods would be classified as a leak or not. Because of this threshold, it was possible to manually tune the TPR and FPR of the system by increasing/decreasing the threshold causing higher/lower TPR and lower/higher/ FPR, depending on the wishes of the operator. This was not possible in our system as we decided to avoid this rather to have the system learn this, and therefore not needing to tune this threshold.

Several methods mentioned in Section 3.2 handle the diurnal pattern in hydraulic data by using several models that are trained on only the data of a given time step during a specific day, resulting in hundreds of models. The leak detection system used in this work was capable of handling the diurnal data and detect leakages without the need to retrain parts of the model on separate time steps of a day or by using several models. As the leak detection system was trained on data where the leaks were fixed, there were no leak labels before and after the leak; we were confident that the system would not need any reset or retraining after a leak was fixed.

6.3 Contributions

This work has shown an application of a gated RNNs based system for the multivariate classification problem; leak detection in WDNs with promising results.

6.3.1 Future works

This work has only shown that gated RNNs can be used in leak detection using artificial data. Future works should train a leak detection system on simulated data and test it on real operational data to see if it is possible to use only artificial data during training, removing the need to gather data from infrequent leaks that are hard to label.

The leak detection system presented here showed greater results when using flow data instead of pressure data. To take advantage of this, the system should be tested on leak detection

within DMAs as these are smaller than WDNs and are equipped with flow sensors, exploiting the systems advantage, while avoiding its disadvantage.

We are confident that the leak detection system could be expanded to use both flow and pressure input, but this would require some feature engineering to identify how the sensor readings should be fed to the system. Two simple implementations could be either one Gated RNN that takes both flow and pressure sensor values as input, or two separate Gated RNNs, one for flow and one for pressure, with a larger linear network after to combine the outputs of the two Gated RNNs.

Chapter 7: Conclusion

7.1 Current Methods for Leak Detection

A SLR was conducted to answer RQ1 which found that many state of the art leak detection methods use different types of ML-techniques as SVMs, Kalman filtering and ANNs. Most of the papers that tested ANNs showed that they did not do well compared to other methods. We hypothesized that this was caused by the simple ANNs used, a MLPs with few layers and hidden nodes. A comparative study of different ANN approaches showed that ANNs with temporal links are better than regular ANNs when predicting the flow data of a single flow sensor. This motivated us to test a temporal network for leak detection.

7.2 A Improved Leak Detection System

A leak detection system was implemented based on a Gated RNN to answer RQ2, after three different ML-methods, case-based reasoning (CBR), CNN and RNN, were considered. It was decided to use the Gated RNNs because of the temporal nature of the sensor data used for leak detection. This decision was based only on a review of the methods, not any testing, which might have shown one of the other methods to be better.

The leak detection system was first tested on a simple WDN to test different hyper parameters for the Gated RNNs. The system managed a TPR of 86%, an FPR of $> 0.1\%$ and a detection time of fewer than 2 hours. These are promising results and makes us confident that the system can be used for leak detection. However, the system struggled with detecting small leaks in the network, which is a regular problem within the leak detection domain. On the other hand, it had no problems with detecting incipient leaks, which is something other leak detection methods usually struggle with. Lastly, it was tested on the Hanoi WDN, which is a popular network within the literature where it was confirmed that flow measurements give better results than pressure data.

7.3 Comparing the Leak Detection System

To answer RQ3, the leak detection system was compared to other leak detection methods discovered while answering RQ1 it was found that the systems result on Net1 was only overall outmatched by one other method, an Ensemble CNN-SVM method reviewed in Wu and Liu (2017).

Bibliography

Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Commun.*, 7:39–59, 1994.

W. Abbass, Z. Bakraouy, A. Baina, and M. Bellafkih. Classifying iot security risks using deep learning algorithms. In *2018 6th International Conference on Wireless Networks and Mobile Communications (WINCOM)*, pages 1–6, Oct 2018. doi: 10.1109/WINCOM.2018.8629709.

M. B. Abbott. *Hydroinformatics: information technology and the aquatic environment*. Avebury Technical, Aldershot, UK, 1991.

Martin Bjerke. Leak Detection System. URL https://github.com/kattn/Master.

Antonio C. Caputo and Pacifico M. Pelagagge. An inverse approach for piping networks monitoring. *Journal of Loss Prevention in the Process Industries*, 15(6):497 – 505, 2002. ISSN 0950-4230. doi: https://doi.org/10.1016/S0950-4230(02)00036-0. URL http://www.sciencedirect.com/science/article/pii/S0950423002000360.

Myrna V. Casillas Ponce, Luis E. Garza Castañón, and Vicenç Puig Cayuela. Model-based leak detection and location in water distribution networks considering an extended-horizon analysis of pressure sensitivities. *Journal of Hydroinformatics*, 16(3):649, 2013. doi: 10.2166/hydro.2013.019. URL http://dx.doi.org/10.2166/hydro.2013.019.

T. K. Chan, C. S. Chin, and X. Zhong. Review of current technologies and proposed intelligent methodologies for water distributed network leakage detection. *IEEE Access*, 6:78846–78867, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2018.2885444.

Cornell University Library. arxiv, 2018. URL https://arxiv.org/.

E. A. Claudia Deniss, G. C. Luis Eduardo, and V. Adriana. Multi-leak detection with wavelet analysis in water distribution networks. In *2012 20th Mediterranean Conference on Control Automation (MED)*, pages 1155–1160, July 2012. doi: 10.1109/MED.2012.6265794.

D. G. Eliades and M. M. Polycarpou. Leakage fault detection in district metered areas of water distribution systems. *Journal of Hydroinformatics*, 14(4):992, 2012. doi: 10.2166/hydro.2012.109. URL http://dx.doi.org/10.2166/hydro.2012.109.

EPANET. Epanet | water research. URL https://www.epa.gov/water-research/epanet.

Malcolm Farley. 2001.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Manuel Herrera, Luís Torgo, Joaquín Izquierdo, and Rafael Pérez-García. Predictive models for forecasting hourly urban water demand. *Journal of Hydrology*, 387(1):141 – 150, 2010. ISSN 0022-1694. doi: https://doi.org/10.1016/j.jhydrol.2010.04.005. URL http://www.sciencedirect.com/science/article/pii/S0022169410001861.

IEEE. Ieee xplore, 2018. URL https://ieeexplore.ieee.org/Xplore/home.jsp.

IWA Publishing. Journal Of Hydroinformatic.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 2342–2350. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045367.

R K Price and Dimitri Solomatine. A brief guide to hydroinformatics. *UNESCO-IHE*, 01 2019.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

K.A. Klise, R. Murray, and T. Haxton. An overview of the water network tool for resilience (wntr). In *1st International WDSA/CCWI Joint Conference, Kingston, Ontario, Canada*, July 2018.

Anders Kofod-Petersen. How to do a structured literature review in computer science, 2018. URL https://research.idi.ntnu.no/aimasters/files/SLR_HowTo2018.pdf.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999134.2999257.

Rui Li, Haidong Huang, Kunlun Xin, and Tao Tao. A review of methods for burst/leakage detection and location in water distribution systems. *Water Supply*, 15(3):429–441, 12 2014. ISSN 1606-9749. doi: 10.2166/ws.2014.131. URL https://dx.doi.org/10.2166/ws.2014.131.

J. Mashford, D. D. Silva, D. Marney, and S. Burn. An approach to leak detection in pipe networks using analysis of monitored pressure values by support vector machine. In *2009 Third International Conference on Network and System Security*, pages 534–539, Oct 2009. doi: 10.1109/NSS.2009.38.

Piotr Migdal and Rafał Jakubanis. Keras or pytorch as your first deep learning framework, 2018. URL https://deepsense.ai/keras-or-pytorch/.

S. R. Mounce, R. B. Mounce, T. Jackson, J. Austin, and J. B. Boxall. Pattern matching and associative artificial neural networks for water distribution system time series data analysis. *Journal of Hydroinformatics*, 16(3):617–632, 10 2013. ISSN 1464-7141. doi: 10.2166/hydro.2013.057. URL https://dx.doi.org/10.2166/hydro.2013.057.

S.R. Mounce. A comparative study of artificial neural network architectures for time series prediction of water distribution system flow data. In *AISB2013 Symposium: Machine Learning in Water Systems*, pages 5 – 12, 2013. URL http://eprints.whiterose.ac.uk/83574/.

S.R. Mounce, C. Pedraza, T. Jackson, P. Linford, and J.B. Boxall. Cloud based machine learning approaches for leakage assessment and management in smart water networks. *Procedia Engineering*, 119:43 – 52, 2015. ISSN 1877-7058. doi: https://doi.org/10.1016/j.proeng.2015.08.851. URL http://www.sciencedirect.com/science/article/pii/S1877705815025217. Computing and Control for the Water Industry (CCWI2015) Sharing the best practice in water management.

Stephen R. Mounce, Richard B. Mounce, and Joby B. Boxall. Novelty detection for time series data analysis in water distribution systems using support vector machines. *Journal of Hydroinformatics*, 13(4):672–686, 11 2010. ISSN 1464-7141. doi: 10.2166/hydro.2010.144. URL https://dx.doi.org/10.2166/hydro.2010.144.

Ken Peffers, Tuure Tuunanen, Charles Gengler, Matti Rossi, Wendy Hui, Ville Virtanen, and Johanna Bragge. The design science research process: A model for producing and presenting information systems research. *Proceedings of First International Conference on Design Science Research in Information Systems and Technology DESRIST*, 02 2006.

R. Puust, Z. Kapelan, D. A. Savic, and T. Koppel. A review of methods for leakage management in pipe networks. *Urban Water Journal*, 7(1):25–45, 2010. doi: 10.1080/15730621003610878. URL https://doi.org/10.1080/15730621003610878.

PyTorch Community. Pytorch | deep learning framework, a. URL https://pytorch.org/.

PyTorch Community. Lstm | pytorch documentation, b. URL https://pytorch.org/docs/stable/nn.html#gru.

PyTorch Community. Lstm | pytorch documentation, c. URL https://pytorch.org/docs/stable/nn.html#lstm.

R. Sarrate, J. Blesa, F. Nejjari, and J. Quevedo. Sensor placement for leak detection and location in water distribution networks. *Water Supply*, 14(5):795–803, 04 2014. ISSN 1606-9749. doi: 10.2166/ws.2014.037. URL https://doi.org/10.2166/ws.2014.037.

Barry Smyth and Mark T. Keane. Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, pages 377–382, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-363-8, 978-1-558-60363-9. URL http://dl.acm.org/citation.cfm?id=1625855.1625905.

A. Soldevila, R. M. Fernandez-Canti, J. Blesa, S. Tornil-Sin, and V. Puig. Leak localization in water distribution networks using model-based bayesian reasoning. In *2016 European Control Conference (ECC)*, pages 1758–1763, June 2016a. doi: 10.1109/ECC.2016.7810545.

Adrià Soldevila, Joaquim Blesa, Sebastian Tornil-Sin, Eric Duviella, Rosa M. Fernandez-Canti, and Vicenç Puig. Leak localization in water distribution networks using a mixed model-based/data-driven approach. *Control Engineering Practice*, 55:162 – 173, 2016b. ISSN 0967-0661. doi: https://doi.org/10.1016/j.conengprac.2016.07.006. URL http://www.sciencedirect.com/science/article/pii/S0967066116301526.

Armin Stahl. Learning similarity measures: A formal view based on a generalized cbr model. In Héctor Muñoz-Ávila and Francesco Ricci, editors, *Case-Based Reasoning Research and Development*, pages 507–521, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31855-2.

Statistisk Sentralbyrå. Vannforsyning og beredskap. kommunalt drikkevann., 2017. URL https://www.ssb.no/statbank/table/11787/.

Jilong Sun, Ronghe Wang, and Huan-Feng Duan. Multiple-fault detection in water pipelines using transient-based time-frequency analysis. *Journal of Hydroinformatics*, 18(6):975, 2016. doi: 10.2166/hydro.2016.232. URL http://dx.doi.org/10.2166/hydro.2016.232.

A. Sánchez-Fernández, M. J. Fuente, and G. I. Sainz-Palmero. Fault detection with distributed pca methods in water distribution networks. In *2015 23rd Mediterranean Conference on Control and Automation (MED)*, pages 156–161, June 2015. doi: 10.1109/MED.2015.7158744.

United Nations. The sustainable development agenda, a. URL https://www.un.org/sustainabledevelopment/development-agenda/.

United Nations. Goal 6: Ensure access to water and sanitation for all, b. URL https://www.un.org/sustainabledevelopment/water-and-sanitation/.

Zoran Vojinovic and Michael B. Abbott. Twenty-five years of hydroinformatics. *Water*, 9(1), 2017. ISSN 2073-4441. doi: 10.3390/w9010059. URL http://www.mdpi.com/2073-4441/9/1/59.

Stelios G. Vrachimis, Marios S. Kyriakou, Demetrios G. Eliades, and Marios M. Polycarpou. LeakDB : A benchmark dataset for leakage diagnosis in water distribution networks. Zenodo, July 2018. doi: 10.5281/zenodo.1313116. URL https://doi.org/10.5281/zenodo.1313116.

D. Vries, B. van den Akker, E. Vonk, W. de Jong, and J. van Summeren. Application of machine learning techniques to predict anomalies in water supply networks. *Water Science and Technology: Water Supply*, 16(6):1528, 2016. doi: 10.2166/ws.2016.062. URL http://dx.doi.org/10.2166/ws.2016.062.

Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, pages 38:1–38:10, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2476-2. doi: 10.1145/2601248.2601268. URL http://doi.acm.org/10.1145/2601248.2601268.

Yipeng Wu and Shuming Liu. A review of data-driven approaches for burst detection in water distribution systems. *Urban Water Journal*, 14(9):972–983, 2017. doi: 10.1080/1573062X.2017.1279191. URL https://doi.org/10.1080/1573062X.2017.1279191.

Yipeng Wu, Shuming Liu, Kate Smith, and Xiaoting Wang. Using correlation between data from multiple monitoring sensors to detect bursts in water distribution systems. *Journal of Water Resources Planning and Management*, 144(2):04017084, 2018. doi: 10.1061/(ASCE)WR.1943-5452.0000870. URL https://ascelibrary.org/doi/abs/10.1061/%28ASCE%29WR.1943-5452.0000870.

Guoliang Ye and Richard Andrew Fenner. Kalman filtering of hydraulic measurements for burst detection in water distribution systems. *Journal of Pipeline Systems Engineering and Practice*, 2(1):14–22, 2011. doi: 10.1061/(ASCE)PS.1949-1204.0000070. URL https://ascelibrary.org/doi/abs/10.1061/%28ASCE%29PS.1949-1204.0000070.

Hans-Georg Zimmermann, Christoph Tietz, and Ralph Grothmann. *Forecasting with Recurrent Neural Networks: 12 Tricks*, pages 687–707. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35289-8. doi: 10.1007/978-3-642-35289-8_37. URL https://doi.org/10.1007/978-3-642-35289-8_37.

Appendix A: Model implementations

The two models implemented can be seen on the next two pages. For a textual description, including hyper parameters and the reasoning for them, see section 4.3.

A.1 GRU model

```
import torch
import torch.nn as nn
import torch.optim as optim

# Set seed for reproducability for my code, will not transfer to
# other platforms
torch.manual_seed(1)

# Input size is set to the number of pressure sensors in the wdn
# The hidden size is tested with 5, 10, 15 and 20
# Number of layers was tested with 1, 2, 3
inputSize = #numbSensors
hiddenSize = 20
numLayers = 2


class GRU(nn.Module):
    lr = 0.003
    lossFunction = nn.BCELoss()
    optimizer = optim.Adam
    output = nn.Sigmoid()

    def __init__(self):
        super(GRU, self).__init__()
        self.hidden = self.init_hidden()
        self.gru = nn.GRU(
            input_size=inputSize, hidden_size=hiddenSize,
            num_layers=numLayers)
        self.decoder = nn.Linear(
            hiddenSize, 1)

    def init_hidden(self, hidden=None):
        if hidden is not None:
            self.hidden = hidden
        else:
            self.hidden = (
                torch.randn(numLayers, 1, hiddenSize))

        return self.hidden

    def forward(self, inp):
        output, self.hidden = self.gru(inp, self.hidden.detach())
        output = self.decoder(output)
        output = self.output(output)
        return output
```

A.2 LSTM model

```
import torch
import torch.nn as nn
import torch.optim as optim

# Set seed for reproducability for my code, will not transfer to
# other platforms
torch.manual_seed(1)

# Input size is set to the number of pressure sensors in the wdn
# The hidden size is tested with 5, 10, 15 and 20
# Number of layers was tested with 1, 2, 3
inputSize = #numbSensors
hiddenSize = 20
numLayers = 2

class LSTM(nn.Module):
    lr = 0.003
    lossFunction = nn.BCELoss()
    optimizer = optim.Adam
    output = nn.Sigmoid()

    def __init__(self):
        super(LSTM, self).__init__()
        self.hidden = self.init_hidden()
        self.lstm = nn.LSTM(
            input_size=inputSize, hidden_size=hiddenSize,
            num_layers=numLayers)
        self.decoder = nn.Linear(
            hiddenSize, 1)

    def init_hidden(self, hidden=None):
        if hidden is not None:
            self.hidden = hidden
        else:
            self.hidden = (
                torch.randn(numLayers, 1, hiddenSize),
                torch.randn(numLayers, 1, hiddenSize))

        return self.hidden

    def forward(self, inp):
        output, self.hidden = self.lstm(
            inp, (
                self.hidden[0].detach(),
                self.hidden[1].detach()))

        output = self.decoder(output)
        output = self.output(output)
        return output
```

Appendix B: INP file

A WDN can be represented as a `.INP` file. It describes placements of junctions, pipes, tanks, reservoirs and other parameters of a WDN. Below is the `.INP` file of the Net1 network used in this thesis. For clarity some of the white space has been trimmed.

```
[TITLE]
 EPANET Example Network 1
A simple example of modeling chlorine decay. Both bulk and
wall reactions are included.

[JUNCTIONS]
;ID              Elev            Demand          Pattern
 10              710             0                       ;
 11              710             150                     ;
 12              700             150                     ;
 13              695             100                     ;
 21              700             150                     ;
 22              695             200                     ;
 23              690             150                     ;
 31              700             100                     ;
 32              710             100                     ;

[RESERVOIRS]
;ID              Head            Pattern
 9               800                     ;

[TANKS]
;ID          Elevation     InitLevel     MinLevel     MaxLevel     Diameter     MinVol   VolCurve
 2           850           120             100        150          50.5         0              ;

[PIPES]
;ID          Node1     Node2     Length     Diameter     Roughness     MinorLoss     Status
 10          10        11        10530      18           100           0             Open     ;
 11          11        12        5280       14           100           0             Open     ;
 12          12        13        5280       10           100           0             Open     ;
 21          21        22        5280       10           100           0             Open     ;
 22          22        23        5280       12           100           0             Open     ;
 31          31        32        5280       6            100           0             Open     ;
 110         2         12        200        18           100           0             Open     ;
 111         11        21        5280       10           100           0             Open     ;
 112         12        22        5280       12           100           0             Open     ;
 113         13        23        5280       8            100           0             Open     ;
 121         21        31        5280       8            100           0             Open     ;
 122         22        32        5280       6            100           0             Open     ;
```

```
40
41 [PUMPS]
42 ;ID              Node1           Node2           Parameters
43  9               9               10              HEAD 1   ;
44
45 [VALVES]
46 ;ID         Node1         Node2           Diameter      Type   Setting        MinorLoss
47
48 [TAGS]
49
50 [DEMANDS]
51 ;Junction          Demand        Pattern           Category
52
53 [STATUS]
54 ;ID                Status/Setting
55
56 [PATTERNS]
57 ;ID                Multipliers
58 ;Demand Pattern
59  1                1.0         1.2         1.4         1.6         1.4         1.2
60  1                1.0         0.8         0.6         0.4         0.6         0.8
61
62 [CURVES]
63 ;ID                X-Value       Y-Value
64 ;PUMP: Pump Curve for Pump 9
65  1                1500          250
66
67 [CONTROLS]
68  LINK 9 OPEN IF NODE 2 BELOW 110
69  LINK 9 CLOSED IF NODE 2 ABOVE 140
70
71
72 [RULES]
73
74 [ENERGY]
75  Global Efficiency    75
76  Global Price         0.0
77 Demand Charge         0.0
78
79 [EMITTERS]
80 ;Junction          Coefficient
81
82 [QUALITY]
83 ;Node              InitQual
84  10               0.5
85  11               0.5
86  12               0.5
87  13               0.5
88  21               0.5
89  22               0.5
90  23               0.5
91  31               0.5
92  32               0.5
93  9                1.0
```

```
 2                 1.0

[SOURCES]
;Node              Type          Quality        Pattern

[REACTIONS]
;Type       Pipe/Tank          Coefficient


[REACTIONS]
 Order Bulk              1
 Order Tank              1
 Order Wall              1
 Global Bulk             -.5
 Global Wall             -1
 Limiting Potential      0.0
 Roughness Correlation   0.0

[MIXING]
;Tank              Model

[TIMES]
 Duration             24:00
 Hydraulic Timestep   1:00
 Quality Timestep     0:05
 Pattern Timestep     2:00
 Pattern Start        0:00
 Report Timestep      1:00
 Report Start         0:00
 Start ClockTime      12 am
 Statistic            None

[REPORT]
 Status               Yes
 Summary              No
 Page                 0

[OPTIONS]
 Units                GPM
 Headloss             H-W
 Specific Gravity     1.0
 Viscosity            1.0
 Trials               40
 Accuracy             0.001
 CHECKFREQ            2
 MAXCHECK             10
 DAMPLIMIT            0
 Unbalanced           Continue 10
 Pattern              1
 Demand Multiplier    1.0
 Emitter Exponent     0.5
 Quality              Chlorine mg/L
 Diffusivity          1.0
 Tolerance            0.01
```

```
148
149 [COORDINATES]
150 ;Node           X-Coord         Y-Coord
151  10             20.00           70.00
152  11             30.00           70.00
153  12             50.00           70.00
154  13             70.00           70.00
155  21             30.00           40.00
156  22             50.00           40.00
157  23             70.00           40.00
158  31             30.00           10.00
159  32             50.00           10.00
160  9              10.00           70.00
161  2              50.00           90.00
162
163 [VERTICES]
164 ;Link           X-Coord         Y-Coord
165
166 [LABELS]
167 ;X-Coord         Y-Coord         Label & Anchor Node
168  6.99           73.63           "Source"
169  13.48          68.13           "Pump"
170  43.85          91.21           "Tank"
171
172 [BACKDROP]
173  DIMENSIONS     7.00            6.00            73.00           94.00
174  UNITS          None
175  FILE
176  OFFSET         0.00            0.00
177
178 [END]
```

Appendix C: Generated Data Files

Example of a `.csv` file containing simulated pressure values from a node in the Hanoi WDN.

```
Timestamp,Value
2017-01-01 00:00:00,69.897
2017-01-01 00:15:00,69.913
2017-01-01 00:30:00,69.932
2017-01-01 00:45:00,69.946
2017-01-01 01:00:00,69.949
2017-01-01 01:15:00,69.956
2017-01-01 01:30:00,69.954
2017-01-01 01:45:00,69.948
2017-01-01 02:00:00,69.939
2017-01-01 02:15:00,69.924
2017-01-01 02:30:00,69.9
2017-01-01 02:45:00,69.874
2017-01-01 03:00:00,69.837
2017-01-01 03:15:00,69.801
2017-01-01 03:30:00,69.76
2017-01-01 03:45:00,69.726
2017-01-01 04:00:00,69.677
2017-01-01 04:15:00,69.645
2017-01-01 04:30:00,69.6
2017-01-01 04:45:00,69.573
2017-01-01 05:00:00,69.57
2017-01-01 05:15:00,69.577
2017-01-01 05:30:00,69.59
2017-01-01 05:45:00,69.604
2017-01-01 06:00:00,69.635
2017-01-01 06:15:00,69.673
2017-01-01 06:30:00,69.695
2017-01-01 06:45:00,69.712
2017-01-01 07:00:00,69.749
2017-01-01 07:15:00,69.764
```

Appendix D: Training and Testing Graphs

During training and testing each graph was stored to be used when reviewing the models, these are some of the graphs. Does it converge quickly, slowly, or not at all are some of the ways they were reviewed. The training and test error graphs shows the training error per scenario. The Avg epoch error graph averages the the training and testing errors per epoch.

D.1 First Iteration

D.1.1 GRU

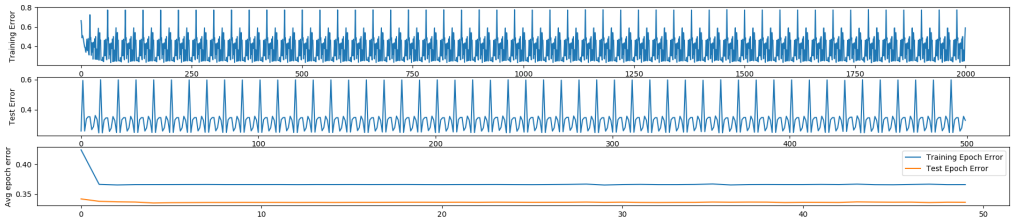


Figure D.1: GRU hs5 nL1

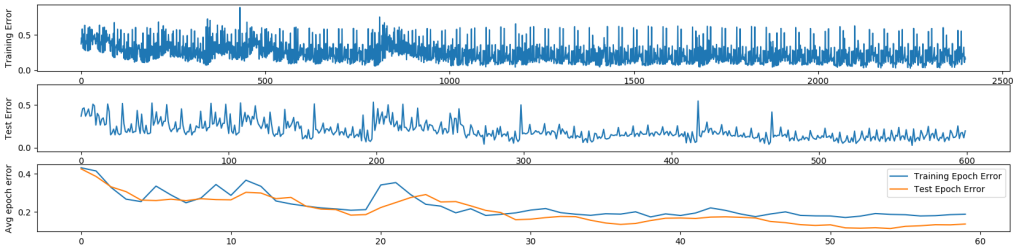


Figure D.2: GRU hs5 nL3

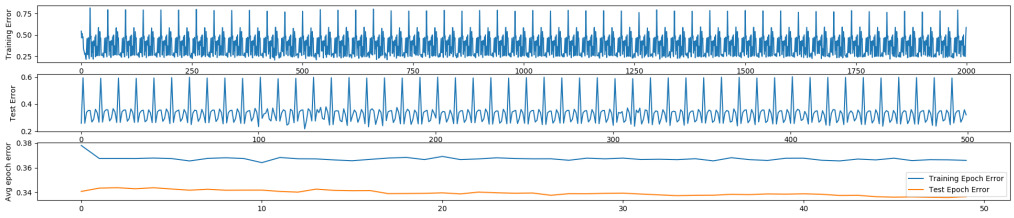


Figure D.3: GRU hs10 nL1

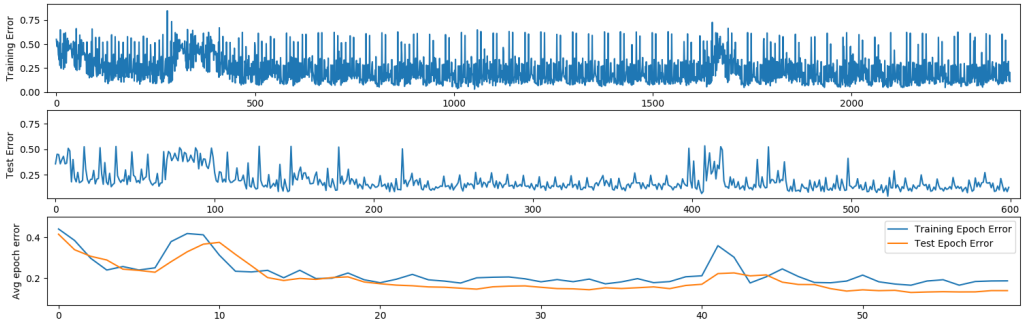


Figure D.4: GRU hs10 nL3

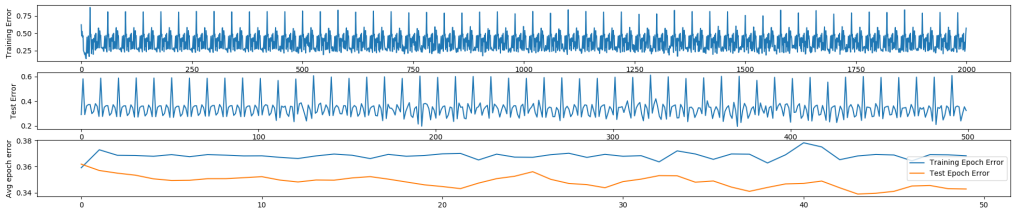


Figure D.5: GRU hs15 nL1

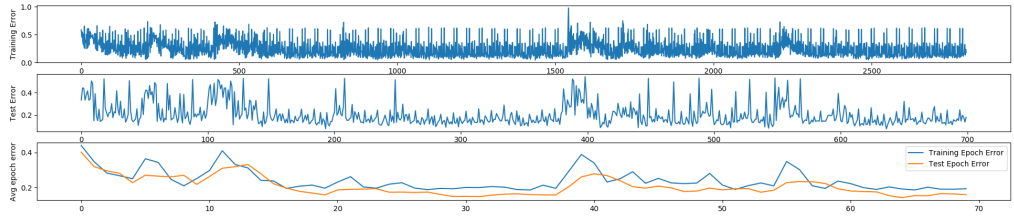


Figure D.6: GRU hs15 nL3

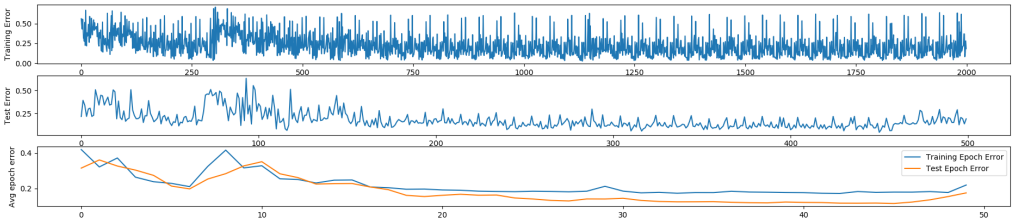


Figure D.7: GRU hs20 nL2

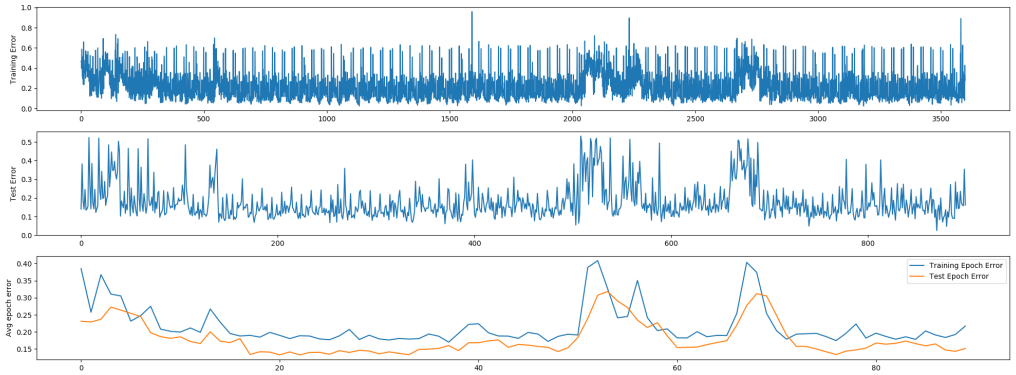


Figure D.8: GRU hs20 nL3

D.1.2 LSTM

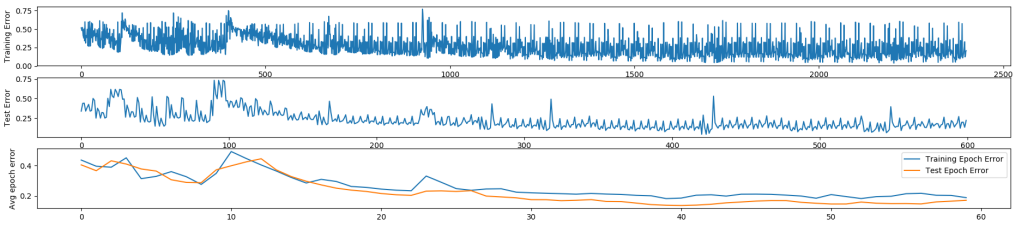


Figure D.9: LSTM hs5 nL1

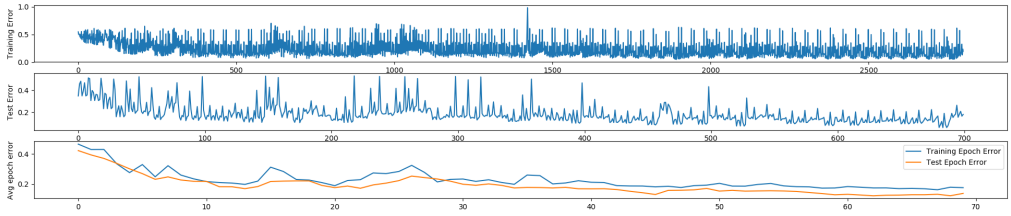


Figure D.10: LSTM hs5 nL2

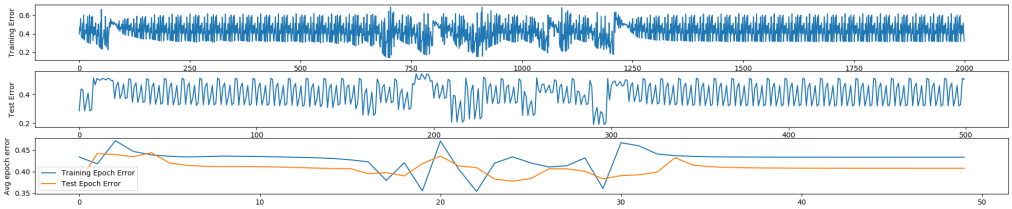


Figure D.11: LSTM hs5 nL3

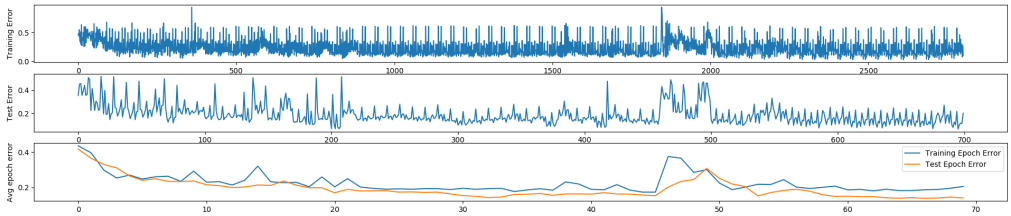


Figure D.12: LSTM hs10 nL2

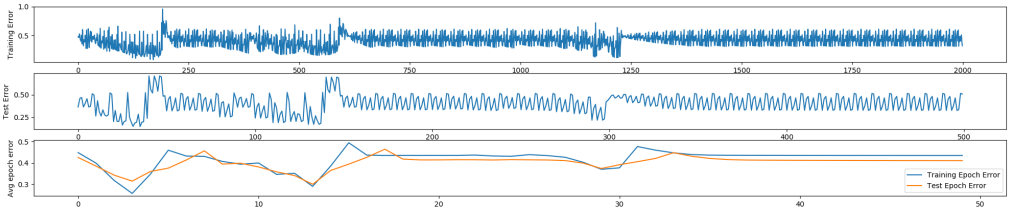


Figure D.13: LSTM hs10 nL3

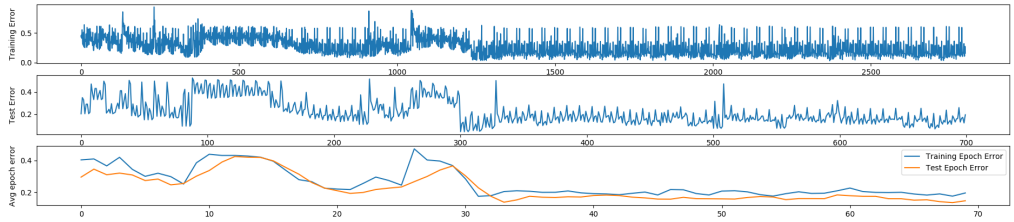


Figure D.14: LSTM hs15 nL1

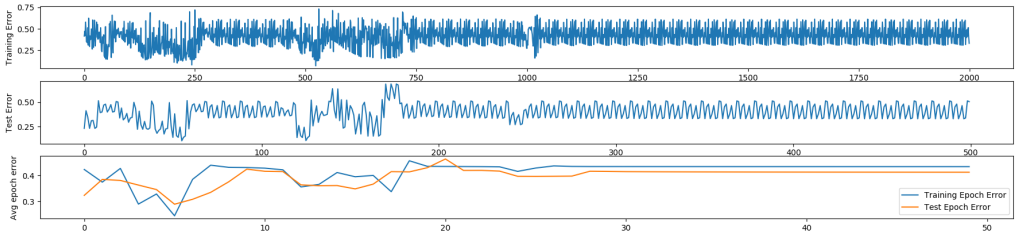


Figure D.15: LSTM hs15 nL3

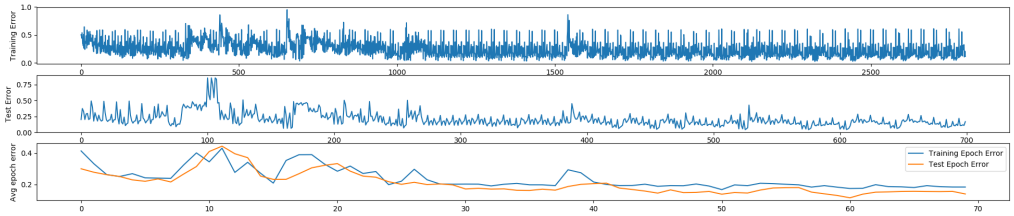


Figure D.16: LSTM hs20 nL1

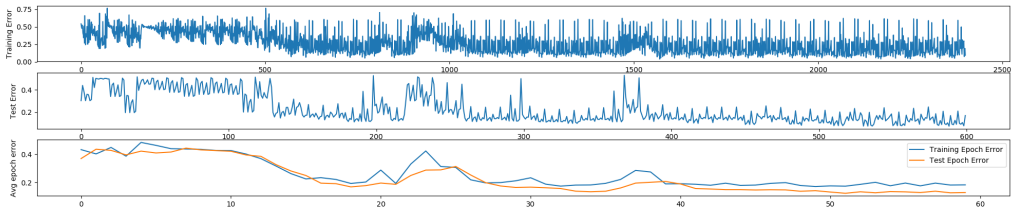


Figure D.17: LSTM hs20 nL2

D.2 Second Iteration

D.2.1 GRU

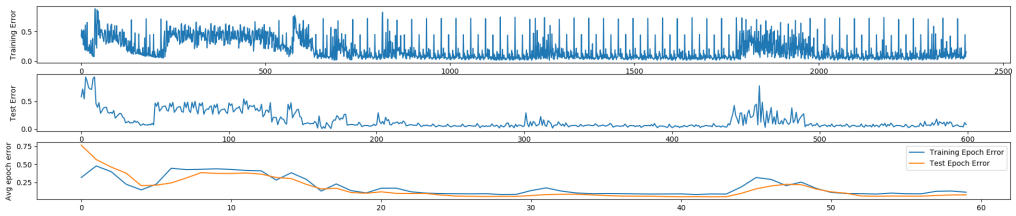


Figure D.18: GRU hs5 nL2

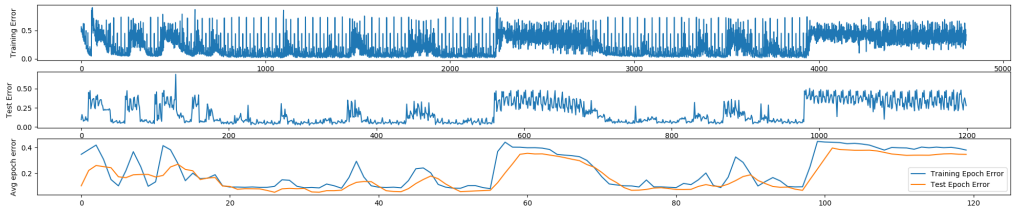


Figure D.19: GRU hs5 nL3

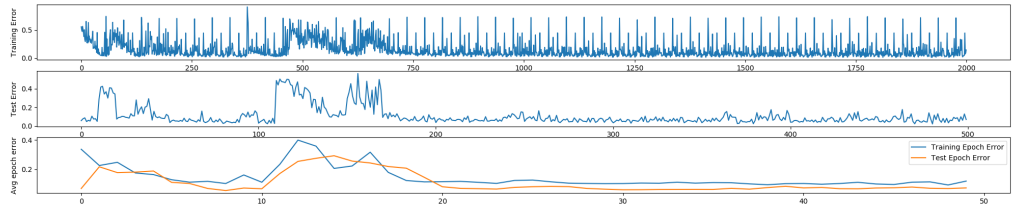


Figure D.20: GRU hs10 nL2

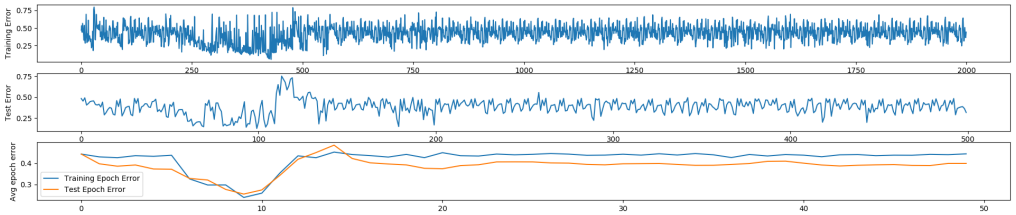


Figure D.21: GRU hs10 nL3

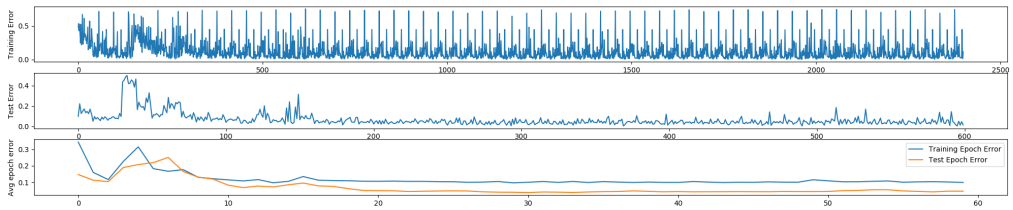


Figure D.22: GRU hs15 nL1

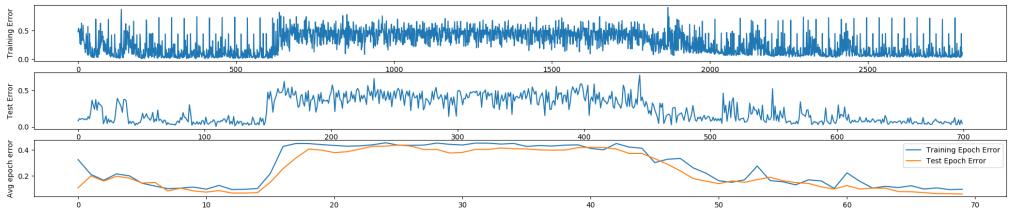


Figure D.23: GRU hs20 nL2

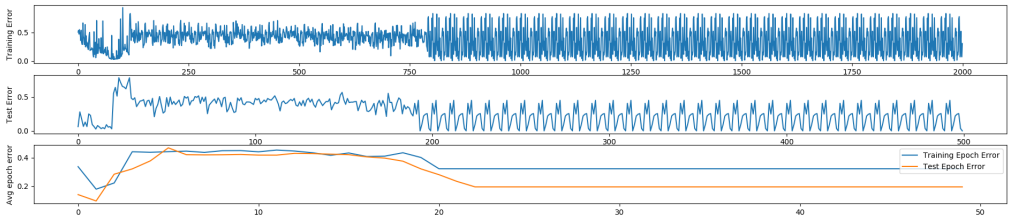


Figure D.24: GRU hs20 nL3

D.2.2 LSTM

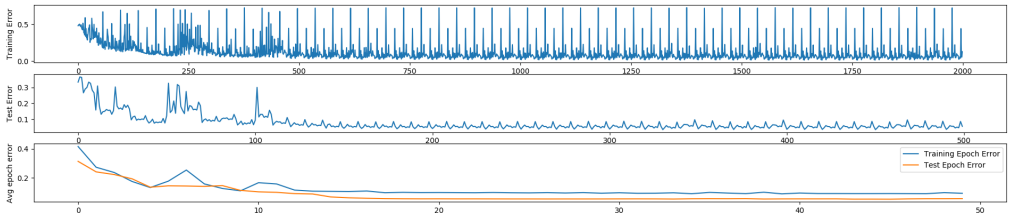


Figure D.25: LSTM hs5 nL1

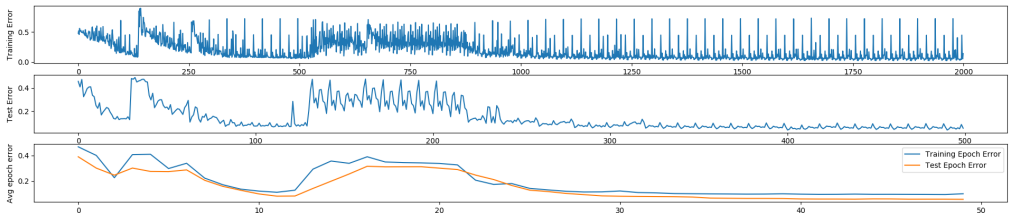


Figure D.26: LSTM hs5 nL2

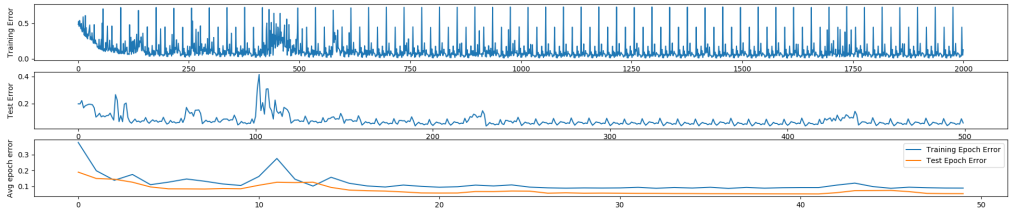


Figure D.27: LSTM hs10 nL1

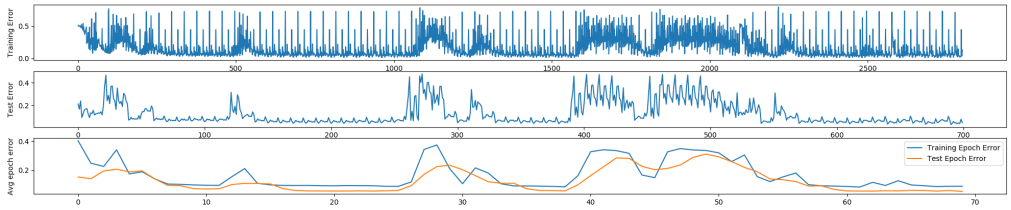


Figure D.28: LSTM hs10 nL3

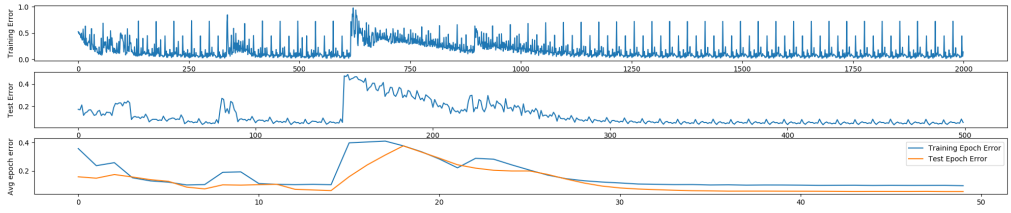


Figure D.29: LSTM hs15 nL1

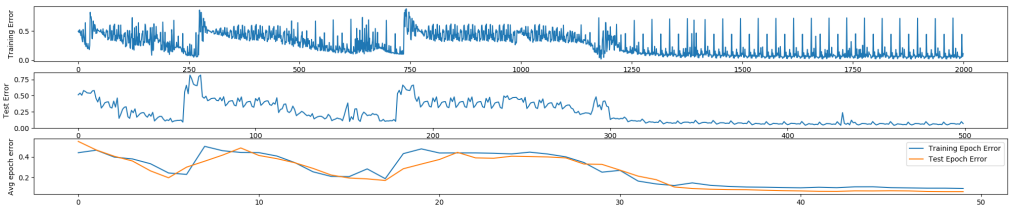


Figure D.30: LSTM hs15 nL2

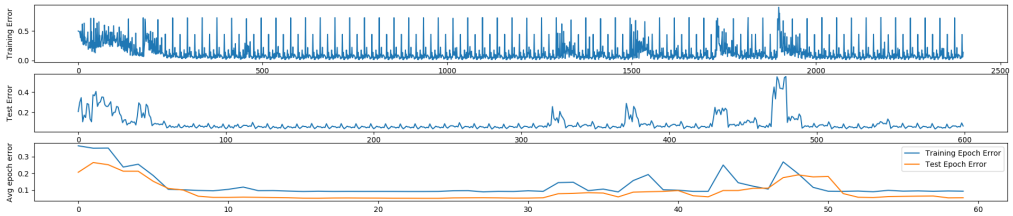


Figure D.31: LSTM hs20 nL1

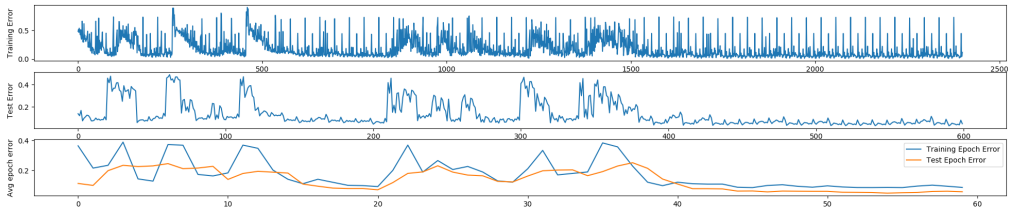


Figure D.32: LSTM hs20 nL2