Prediction of alarms in a pump station using neural networks

STAFFAN ALDENFALK JANSSON
Prediction of alarms in a pump station using neural networks

Author: Aldenfalk Jansson Staffan

Supervisor: Fullemann Alexander
Examiner: Boman Magnus

A degree project submitted in fulfilment of the requirements for the Masters Degree, written at

Xylem Inc.
Xylem water solutions AB

17 November 2017

KTH Royal Institute of Technology
School of Information and Communication Technology (ICT)
Department of Communication Systems
SE-164 40 Kista, Sweden
Abstract

Prediction of pump station alarms based on data is an interesting future service for companies as it allows them to both offer a new service and reduce the downtime of the pumps. In order to predict the alarms a study of the available data was done in order to examine feasibility and identify problems. The chosen solution was to use two neural networks which were connected in a chain to create a complete solution. The first neural network used long short term memory (LSTM) neurons in order to recursively predict time-series data from sensors, such as sump water level and pump electric current, this was then used by the second LSTM neural network in order to determine if these parameter values would trigger an alarm. The second LSTM network was unable to determine if an alarm would happen and thus the whole solution did not work. There were two main reasons for this, the first being that the alarms stop time did not correlate to the time-series sensor data which created an uncertainty of which parameter levels actually belonged to an alarm, making the LSTM network unable to identify what an alarm is. The second reason was that the data was downsampled too much making it even harder for the LSTM network to identify what an alarm was. This thesis has helped Xylem further understand the use and needs of machine learning, which will help Xylem progress further into the area of predictive and smart services.
Acknowledgements

For this thesis I have gotten help from several people, the one who has helped me the most is my supervisor at Xylem Alexander Fullemann who helped me during all stages of the thesis. A big thank you, to Alexander Fullemann! I would also like to thank Mikael Nilsson, Kristin Hylmö, and Jan de Vreugd for helping me to understand the data, without them this would have taken a lot longer. There are a lot more people I would like to thank for advice I have gotten through this thesis, some of them being Jürgen Mökander, Ad Damme, Alexander Tollet, Nicholaus Claudio, and my examiner Magnus Boman. Thank you for all of the help I have received from all of you, this thesis would not have been as good without you.
# Contents

Abstract  

Acknowledgements  

Abbreviations and Acronyms  

1 Introduction  
  1.1 About Xylem  
  1.2 Machine learning  
  1.3 Problem description  
  1.4 Goals  
  1.5 Data  
    1.5.1 SCADA data overview  
    1.5.2 Pump station overview  
    1.5.3 Alarm  
  1.6 Research methodology  
  1.7 Delimitations  
  1.8 Structure of this thesis  

2 Background  
  2.1 What is a neural network?  
    2.1.1 Architecture of a neural network  
  2.2 Problems and solutions concerning the data  
    2.2.1 Different dimensions of the input  
    2.2.2 Threshold alarms  
    2.2.3 Switch alarms  
    2.2.4 Normalization  
    2.2.5 Sample speed  
    2.2.6 Downsampled data  
    2.2.7 Naming ambiguity  
  2.3 Problems and solutions concerning neural networks  
    2.3.1 Overfitting  
    2.3.2 Number of neurons and layers  
    2.3.3 How the results are evaluated  
  2.4 Related work
3 Method
  3.1 Evaluation process .............................................. 21
  3.2 Data ............................................................... 21
  3.3 Chosen method ..................................................... 23
    3.3.1 Neural network configurations . ............................ 25
      3.3.1.1 Batch size ................................................. 25
      3.3.1.2 Activation functions, loss function and optimizer .... 25
      3.3.1.3 Neurons and layers ...................................... 26
      3.3.1.4 Dropout and regularisation ............................ 26
  3.4 How the final test is constructed ............................... 27
    3.4.1 Result evaluation ........................................... 27

4 Implementation .................................................... 29
  4.1 Experimental Setup .............................................. 29
    4.1.1 Hardware ...................................................... 29
    4.1.2 Software ...................................................... 29
  4.2 Changes made to the chosen method during implementation .... 29
    4.2.1 Limitations with Keras ..................................... 30
    4.2.2 Re-label data ............................................... 30
      4.2.2.1 Approach one ............................................. 30
      4.2.2.2 Approach two ............................................. 31
    4.2.3 Data cleaning ............................................... 31

5 Analysis and results .............................................. 33
  5.1 Tests to determine trend network structure .................... 33
    5.1.1 Activation function ......................................... 33
    5.1.2 Number of neurons and layers ............................... 34
    5.1.3 Dropout and regularization ................................ 34
    5.1.4 Batch size ................................................. 35
  5.2 Tests to determine alarm network structure .................... 36
    5.2.1 Number of neurons and layers ............................... 36
    5.2.2 Dropout and regularization ................................ 36
    5.2.3 Further analysis and changes to the network .............. 37
  5.3 Final test ........................................................ 38

6 Conclusions ......................................................... 40
  6.1 Comparison of the results with the goals ....................... 40
  6.2 Reflections on issues relevant to this thesis project ......... 40
    6.2.1 Note about alarm network configuration .................. 41
  6.3 Future work ...................................................... 41
## Abbreviations and Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGD</td>
<td>Stochastic Gradient Descent</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>HMM</td>
<td>Hidden Markov Model</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short Term Memory</td>
</tr>
<tr>
<td>GRU</td>
<td>Gated Recurrent Unit</td>
</tr>
<tr>
<td>FFNN</td>
<td>Feed Forward Neural Network</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphical Processing Unit</td>
</tr>
<tr>
<td>RMSE</td>
<td>Root Mean Squared Error</td>
</tr>
<tr>
<td>SCADA</td>
<td>Supervisory Control And Data Acquisition</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This chapter introduces the problem and its context, the constraints and delimitations associated with this project, the structure of this thesis, and the goals that are aimed to be fulfilled.

1.1 About Xylem

*Xylem* is a company focused on providing water solutions. It consists of around 16000 employees and the office in Sweden Sundbyberg has around 450 employees. There are many brands in *Xylem’s portfolio, such as Flygt, Wedeco, Lowara, Godwin, and Sensus.* The Xylem office Stockholm where the thesis was done mainly works with pumps, mixers and their associated monitoring systems. *Xylem* has manufactured pumps for many years, this means that they have a large variety of pumps with different sizes and purposes. Everything from small pumps that can be used in a summer cabin to larger pumps that are used in treatment and nuclear plants. *Xylem* always strives for improvement and now want to further explore the technology of machine learning in order to see how it can benefit them. For this thesis *Xylem* has limited the problem to consist of their small and mid-range pumps used in pump sumps which transports sewage water.

The goal for *Xylem* is both to see what machine learning can do for them in this problem scope and to see where it can be applied well. After this thesis there are plans to see what other problems that can be solved using machine learning and what is needed in order to solve them.

I have previously done my bachelor thesis and worked for three summers at *Xylem.* The work I did then was related to what additional information you could obtain about a pump by either adding a new sensor or increasing the use of an existing one. The sensor information I worked with was: video, sound, vibration, and magnetic field. The vibration and magnetic field were tested in combination with each other while the others were tested separately.
1.2 Machine learning

There are three big building blocks in the concept of machine learning: input, output and, model selection. The output is what you want to find, the input is where you think it can be found and a model is chosen that can find it in the input. How this differs from traditional programming is well explained by Jason Brownlee [1] where:

1. **Traditional programming**: Input and program is run on the computer to produce the output.
2. **Machine Learning**: Input and output is run on the computer to create a program. This program can be used in traditional programming.

In other words machine learning creates a program that is able to give an output based on previous inputs and outputs, Figure 1.1 shows how this relates to traditional programming, from Jason Brownlee[1]. This also means that the model used in the program wont be able to find an output it has not seen before.

![Figure 1.1: Traditional approach and how machine learning relates to it. The data represents the input and the program has a model incorporated into it. From Jason Brownlee[1]](image)

The following text is heavily dependent on the book Artificial Intelligence: A Modern Approach[2]. There are four sections of models in machine learning which are used depending on what input and output data is available, these four are:

1. **Supervised learning**: When both input and output data is available.
2. **Unsupervised learning**: When only the input data is available.
3. **Semi-supervised learning**: When only a small amount of the inputs has a corresponding output.
4. Reinforced learning: When no historical input or output is available, the parameter learning is then based on a trial and error reward system.

**Supervised learning** is the easiest and fastest method to use since the selected model can be evaluated directly on historical data. This allows quicker adjustment of the model parameters or a change to another model if it does not work. It is often quite expensive to get to the stage where supervised learning can be used as it is often expensive to label all the output data. One example of this is object recognition in images where a competition has been held each year since 2010 on the data set ImageNet[3] which as of 2016 has over 10 million hand-annotated images with labels of what is in it.

**Unsupervised learning** is used in order to infer a function to describe a hidden structure in the input data. This can for example be used in clustering, anomaly detection, neural networks, or to find hidden variables. Since it only works with the input data it is fairly cheap to use since there will be no manual labeling of the output. It also means that it can only find things that exist in the input. For example if the input data consist of the parameters water-level and pump-current it would be impossible to predict alarms since there is no information about what an alarm is. But it would be possible to use it to detect abnormalities in the data and warn when they happen.

**Semi-supervised learning** often use both supervised and unsupervised learning in order to accomplish its task. First unsupervised learning is used in order to label or infer a label to the unlabeled output data and then supervised learning is used in order to accomplish its task. This means that it has the advantages of supervised learning but also the disadvantages of unsupervised learning where it is hard to see if it has correctly labeled the data. An example of where semi-supervised learning can be used is the following: A pumping station has been logging the water level since 2001, and 2016 it started labeling the data with the alarm overflow, which means that there is too much water in the pumping station. If we now want to predict the alarm overflow using all of the data from 2001, semi-supervised learning can be used in order to first infer the label overflow on the data from 2001 to 2016 and then supervised learning could be used to predict the alarm.

**Reinforced learning** is unique in the way that it does not require any historical input or output data but instead learns from trial and error. This requires an environment where it can make mistakes and start over. A clear scheme of what is good and bad is also needed. A simulation of the problem to be solved and its environment is useful in this case. An example could be a simulation of a pumping station where it tries to learn how to best control the start and stop of the pumps.

For the concrete problem of predicting the future it falls under the supervised learning section. The input will be the historical data that has been collected in the pumping station and the output will be the historical data time shifted one or more time steps. In order to look far into the future two different techniques can be used, either time shift the data several steps or predict one step which is then used in order to predict the next step creating a recursive model.
1.3 Problem description

When an alarm happens in a pumping station today the actions to restore it is done afterwards when the damage is already done. When something goes wrong it often influences its surroundings which can cause even more damage. If the error instead was predicted, the damages it would cause could be prevented. This could potentially save money, reduce the downtime of a pumping station, and reduce the impact of damages on the environment.

In order to predict when an alarm will happen machine learning will be used. Where a neural network will be trained on historical data from pumping stations collected using Xylem’s Supervisory Control And Data Acquisition (SCADA) systems. The historical data is expected to consists of a large variety of pumping stations, with varying data quality. Making this data understandable for a neural network will be one of the first tasks for this thesis.

Predicting the future has always been a hard task which is why it will be crucial to measure the uncertainty of the predictions. A prediction can never be 100% certain, but if the probability of it is known, it can be used to make decisions, for example: perform maintenance on a pump. Because of this one part of the problem will be to create an estimation of the probability that an alarm will happen.

This thesis is the first known published attempt to apply machine learning to Xylem’s collected data SCADA globally. Because of this many more challenges are expected to be found.

1.4 Goals

The goals of this project is to answer the following questions:

- Is it possible to predict alarms with the available data?
- How long in advance can the alarms be predicted?
- How reliable is the predicted alarm?

1.5 Data

1.5.1 SCADA data overview

In order to predict alarms in a pumping station, using machine learning, historical data is collected from Xylem’s SCADA system. This system has a lot of variation both in the hardware it is monitoring and which software that is used to log. The data that is available for this thesis comes from four different countries, which uses three different SCADA systems. Each of these countries has different mother languages which in the cases where the customer gets to freely write, has a big influence on names of parameters. Systematic differences in the languages between the countries could also be found. The
three SCADA systems gives lots of freedom to the customer where they can choose if they want to use standard equipment and configuration, or if they want to customize and program the equipment. Because of this there can be a big difference between customers in the number of extra sensors and optimized control algorithms. The pumping stations this thesis will look at are the ones which uses two wastewater-pumps in a sump, two examples of this configuration can be seen to the left in Figure 1.2. There are three common steps between the countries and SCADA systems in how the data is collected, these are listed below and also illustrated in Figure 1.2:

1. When a pump is installed in a pumping station the customer gets to choose which sensors the pump should have and if they should be logged.

2. The pumping station collects the information from each pump and extra sensors in the station. The only sensor which is not considered an extra sensor is the high level float switch which is in almost every station.

3. The sensor and pump information from each pumping station is sent to SCADA software which saves the information.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure1}\caption{The figure illustrates an overview of different setups and devices that can be used, image made by a former employee at Xylem named Lasse Larsson (unpublished). At the bottom the grey pumps can be seen together with different sensors that can be used to measure water level. Slightly above that in the middle there is a monitor attached to a green pipe, this is a sensor that measures the water flowing in or out from the station. In the middle layer some of the different consoles and controlling systems can be seen. The cloud at the top illustrates where all the information is stored, each red line from it represents a different pumping station.}
\end{figure}
1.5.2 Pump station overview

The data that this thesis will handle is the one from the last stage, the collection of data from several pumping stations. This data goes through several layers before it reaches the last stage which means that there are many instances where it can be distorted or changed. Following is a list of difficulties in the data and some of the things that can happen in a pumping station:

1. The pumping stations will each log their own sets of parameters, with their own format. One pumping station might be logging: \textit{water level} as percentage of total filled sump and, leakage. While another pumping station is logging: \textit{water level} as mA from a pressure sensor, \textit{leakage}, and the \textit{current} used by the pump.

2. Each pumping station will have its own set of alarms which can trigger. This is because the pumping stations measures different things, a station cannot give the alarm \textit{current overload} if the current is not measured.

3. How often the parameters are logged can be different between pumping stations and there can be differences between parameters in the same station. For example \textit{pump running time} is in some stations logged 4 times a day while \textit{water level} is logged once every 5 minutes.

4. There are several things that can happen to a sensor in a pumping station, some of them are: it is changed to another type/model, it can be wrongly tagged, it can break, it can be disturbed (magnetic fields, loose cables, fat structure and so on.).

5. The data that is received can be wrongly tagged, the resolution changed, or a parameter can be changed, removed, or added.

6. External influences on the station can occur where a pump is changed, a pipe breaks, or maintenance is done, and so on.

7. The different components in a station can change settings such as changed start and stop levels, turned off automatic pump cleaning and so on.

8. Power network error, the reserve generator usually turns on but some sensors might not work anymore and other errors might be caused by this.

1.5.3 Alarm

There are generally two types of alarms that can happen, switch and threshold alarms. The threshold alarm is characterized by the fact that a readable parameter goes over a threshold which sets off an alarm. While the switch alarms are characterized by either an unseen sensor value or a digital switch which sets off an alarm. This means that the switch alarms does not have a direct correlation to the read parameters. There are also alarms which are a combination of both threshold and switches, or the same alarm can be triggered either by threshold or switch. In order to reduce the number of false alarms there is often a delay of 10 seconds in order to ensure that the alarm is real.

One consistent thing about the data from all of the different countries is that it uses the same codes for tagging the alarms and the data is always labeled where an alarm has happened. Since the solution then does not need to identify what an alarm is, the problem can be considered to be supervised.
1.6 Research methodology

The approach to solving the problem will be reductionism where the big problems will be reduced to smaller ones. Each of the smaller problems will briefly be described with some possible solutions. The idea is then to use the smaller solutions in order to create one big algorithm that incorporates the best solutions.

The literature study was done in order to find which neural network that would work the best, how to configure the network and how others had solved a similar problem. This was done by searching through KTH’s online library by searching for keywords relevant to this problem and another search was done in Google’s search engine. The terms searched for were connected to neural networks. The first searches was broader where terms such as “neural network prediction” where used and once a promising network had been found, more specific search terms where used such as “LSTM time series prediction”. The references found that was relevant to the solution was then shared with one other student who also did their master thesis with neural networks, who then shared his findings with me.
Chapter 1. Introduction

1.7 Delimitations

For this thesis there are a number of delimitations.

- No deeper knowledge of how each of the pumping stations works.
- No deeper knowledge of how each of the alarms is generated.
- No deeper research into why each of the alarms are triggered.
- The assumption is made that all of the alarms are correctly tagged.
- The assumption is made that the parameter data from a pump station is correctly tagged, so if it says water level it is water level.
- The solution will focus on using neural networks.

1.8 Structure of this thesis

Chapter 2 provides the background necessary and goes into the problems associated with this thesis, it also discusses some possible solutions. Following this Chapter 3 goes through the chosen implementations, and the reasons to why it was chosen. Chapter 4 goes through the hardware and software used, and any deviations from the chosen implementation from the previous chapter. In Chapter 5 the results are analyzed and finally, Chapter 6 offers some conclusions and suggests future work.
Chapter 2

Background

This chapter formulates and defines all of the methods, principles, and models that are believed to be relevant to the problem stated in Chapter 1. These methods may be combined in a number of different ways.

2.1 What is a neural network?

A brief introduction to what a neural network is and its terminology is given here in order for the reader to have an easier time to understand the following chapters. The explanation will be about a feed-forward neural network (FFNN) which is the base used for more complicated ones.

2.1.1 Architecture of a neural network

A neural network consists of several building blocks, the three most important ones are: input layer, hidden layer, and output layer. These three layers are connected in a feed-forward way which can be observed in Figure 2.1, where each square represents input data such as water level and the circles represent neurons, the image is from [4].

A neuron can be built in many different ways depending on what task the network is constructed to accomplish. The basic structure of a neuron is illustrated in Figure 2.2, the image is from [4]. Each input to a neuron is multiplied with a weight, which in the figure is shown as $W_n$, and then summarized and run through an activation function. The output from the activation function is then either used as the input for the next layer of neurons or as the output for the whole network. The activation function is what allows a neural network to capture non-linear behavior of data and there are some different kinds that can be used such as linear, tanh, sigmoid, hard sigmoid and rectified linear unit (ReLU) [5].

When a neural network is trained it is the weights of each neuron that is adjusted. What in essence happens is that each time the network does a mistake it finds the weights that created the error and adjusts them so that the network will be correct the next time it encounters a similar error. Because of this a neural network needs to iterate through the training data several times. Because if it adjust the weights to fix one error it might
have trouble with an error it previously adjusted for. In machine learning an iteration through the whole training set is called an epoch and will be the term used here after.

How much each weight should be adjusted is its own science but its biggest breakthrough was the backpropagation algorithm which was formulated by Linna, Seppo [6]. It managed to speed up the training and is now used in the large majority of neural networks. What it in essence does is that it calculates the derivative of each weight with respect to the error, which is then used in order to change the weights value. In order to speed up the training further an optimizer can be used which takes the values calculated from the backpropagation and decides how much the weight shall be changed. There are several different optimizers, some of them are Stochastic gradient descent (SGD), Adagrad [7], and Adam [8].

During training when the input data has gone through the whole network, it is also run through a loss function which calculates a price to pay for inaccuracies. With the loss
function it is possible to punish big and small errors differently so that a higher price is payed for big errors or vice versa.

2.2 Problems and solutions concerning the data

The large problem of predicting alarms is in this section split into smaller problems with possible solutions. This is done in order to give a good overview of what needs to be considered when solving the bigger problem.

2.2.1 Different dimensions of the input

Each pumping station in the data sets has its own set of sensors and parameters that it reads in, one station can read in 7 values while another reads in 9. The number of parameters for a station can also change over time with the addition of new sensors. The values that the pumping stations read in are also different, it could read in temperature, current, and many more which can all be of different types.

A neural network expects the input to have the same dimensions every time, which makes the changing of input dimensions a problem. This problem has been encountered in image recognition[9] many times, as one image can have the size 640x480 while another is 1280x720. There are several techniques to solve this in image recognition and what many of them exploit is that each value in a image is a measurement of the same type, color intensity. In the case of a pumping station this is not true as one parameter can read current while another reads temperature. There is however one technique from image recognition that can be used, zero padding. It works by finding the largest image, for example 1280x720, and then adding zero columns and rows to the smaller images until they are of the same size as the largest image. The same principle can be applied to pumping stations, but instead of searching for the largest image it instead finds the names of all the available parameters that can be recorded in a pumping station and uses that as the “largest image”. This approach allows the neural network to make use of all the available data but it will slow it down slightly as the input gets bigger. This reduction in speed should however be very small as these computations are done in parallel.

2.2.2 Threshold alarms

The threshold alarms introduced in Section 1.5.3 are characterized by a parameter going over a set value, which causes the alarm to go off. This is a very easy behavior to catch with a lot of machine learning algorithms, this also means that this is a trivial problem for neural networks and will thus not be discussed further.

2.2.3 Switch alarms

The switch alarms introduced in Section 1.5.3 are characterized by the fact that they are only shown in the output. The switch that goes off is not shown in the logged data only its result. The alarm that happens can still be linked to a logged parameter. For
example if the temperature goes over $k$ degrees Celsius one of the bearing layers which protect the pump from leakage can melt, this in turn causes a small leakage which build up over time until the digital leakage sensor sends the alarm leakage. This example being a indirect correlation through a physical system where the temperature alarm caused the alarm leakage alarm.

This puts the demand of connecting previous events and patterns in the input data with the outputs on the chosen algorithm. Mathematically this can be expressed as:

$$P(A_{n+1}|t_1, t_2, ..., t_n)$$ (2.1)

Where $A_{n+1}$ stands for one alarm at time $n + 1$ and $t_n$ stands for the input parameter data at time $n$. The further back in time an event and alarm are connected the more computationally heavy the model has to be. In order to make it computationally possible to do this in a reasonable time it is common to use the Markov assumption which states that the current time only depends on its closest time step, in other words $t_n$ only depends on $t_{n-1}$. This leads to it being mathematically expressed as:

$$\prod_{k=0}^{n} P(A_{n+1}|t_k)$$ (2.2)

This is a lot faster to compute. This is a model that works well when the event that leads up to the alarm are close to each other in time but gets worse the further away they are.

Another way of solving this type of problem is to use Recurrent Neural Network (RNN) or Long Short-Term Memory (LSTM)-network which is the RNN’s extension. The RNN works by letting the previous output of the network be used together with the new input in order to create a new output. When doing this it weights how much of the previous output it should keep in order to create an accurate new output. This network has the same problem as the Markov assumption, the further away in time the worse performance.

The LSTM-network solves this by adding a memory into the network that is able to remember the important events which can cause a certain alarm and forget the unimportant ones. The LSTM-network also has the advantage that it solves the vanishing gradient problem [10] which made it harder to build deep networks. It does however have the problem of being slow to train which can be seen in Soheil Bahrampou et al.’s report [11], as it is the one that takes the longest time. The LSTM-network was discovered quite recently by Sepp Hochreiter et al. [10] 1997, since then a lot of variation of the LSTM neuron has come up. In 2015 Rafal Jozefowicz et al. [12] did an empirical study on different structures of RNN and compared it to different structures of LSTM and Gated Recurrent unit(GRU). From their study it can be seen that it is possible to find a structure which outperforms LSTM and GRU on some tasks with a small margin, but not on all. They could also see that the performance between LSTM and GRU is almost the same if the LSTM’s forget gates bias is initialized as 1 and dropout is used when there are over 5 million parameters in the model. K. Greff et al. [13] also did a study on eight different architectures of LSTM with different settings in 5400 experimental runs, their conclusion was that no change to the original LSTM network would significantly change the performance.
2.2.4 Normalization

Normalization in machine learning consists of two things: the first is formatting the data so that it is consistent between the different units that produce data, in this case pumping stations. The second is to format the data so that it is consistent between the different input parameters.

The first point of making the data consistent between the pumping stations might seem a little obvious but it is still very important especially since there are so many things that can go wrong there. For example the parameter water level can be measured using different units, such as meter, centimeter, or percentage to full. The starting reference point can also be different, one pumping station can use the bottom of the sump as reference while another uses the sea level. The most important point here is not that the best unit is used, but that all the pumping stations uses the same unit.

The second point of changing the input so that it is consistent between the parameters consists of rescaling the range of each parameter so that they are approximately within the same range. There are two reasons for this, the first one being that if the input range is in the same range as the activation functions operating range from the start, the neural network does not need to learn how to push the inputs to be within the operating functions range. Thus making the training faster, and in some instances a better result can be achieved. The second reason is that it places the same importance on all of the parameters. If parameter A had values in the range [0,1] and parameter B had values in the range [0,100] then parameter B’s derivative can be 100 times larger which can cause the network to more strongly associate the errors and successes to parameter B and hardly make any changes at all to the weights associated to parameter A. This can cause the network to overly rely on the parameters which has the largest operating range instead of the parameters which gives the most relevant information.

When normalizing data to be consistent between input parameters, there are two normalizers that are popular: max−min scaler, and scaling it to a normal distribution. The max−min scaler forces all of the parameter values to be within a certain range which makes it a good choice for activation functions that has a set range where it best operates. The equation for the max−min scaler can be seen in formula 2.3 where \( t \) is the value to be normalized, notice that this formula will force all the values to be in the range [0,1].

\[
t_{\text{normalized}} = \frac{t - t_{\text{min}}}{t_{\text{max}} - t_{\text{min}}} \tag{2.3}
\]

Scaling the data to be within a normal distribution is done by subtracting the mean and dividing with the standard deviation, as shown in formula 2.4, where \( \mu \) is the mean and \( \sigma \) is the standard deviation. This forces 68.4% of the parameter values to be within the range [-1,1]. This also means that 31.6% of the values will be outside of the range which can make this normalization better suited for activation functions that operates in a broader range.

\[
t_{\text{normalized}} = \frac{t - \mu}{\sigma} \tag{2.4}
\]
The parameters used for the normalizations are calculated from the training data. This is done in order to ensure that the solutions parameters only come from the training data so that it can give a correct representation of its results on new unseen data.

### 2.2.5 Sample speed

Each parameter read in will have its sample speed set. Some will have the same but others will not. For example in some stations the accumulated run time of the pump is reset to zero for each day and then sampled at 4 different times during the day while the water level can be measured once a minute.

A solution for varying sample speeds between parameter can be solved by giving the parameter the same value until it is sampled again, or having a network such as RNN or LSTM remember the parameter value. It can also be important to know how fast something is happening and if it happens faster at some specific time, for instance the water level can rise faster around 7 o’clock when people wake up and shower. Just inserting the time stamp into the algorithm might work but probably not good because time is measured as a cycle, once the minutes reaches 60 it starts from zero again. This can be hard for a neural network to capture. One solution to this is to divide the time stamp cyclic parts separately and project them to a unit circle using a \( \sin \) and \( \cos \) function. This will make it much easier for the network to see how close in time things are happening. Another way of solving this problem would be to take the difference in time between each time stamp and use that as a input instead. This would make it easier for the network to see how fast something is happening but it would make it harder to tie it to certain times.

### 2.2.6 Downsampling data

The data available for this thesis that has alarms in it has all been downsampled. This has been done by sampling a parameter each second and then the average has been taken for x minutes and saved as one sample. How many seconds it has taken average of differ from the different SCADA systems, and in some systems it is dependent on how old the data is. The ones that are most used are 5 min, 1h, and 1 day. The exact times for alarms are however saved in all the SCADA systems.

What downampling does is that it eliminates any pattern that might have existed in that time period. If the data is saved at the downsampled time interval of 5 min then it has sampled down 300 points to 1 and erased any pattern that might have existed in those 300 points. Data with a high sample speed can be used to improve algorithms which has a majority of data with low sample speed but the other way around does not work, it is the slowest sample speed which sets the limits.

Saving the alarms at their exact times does unfortunately not give much when the data has been downsampled. Consider the following example: The resolution of 5 min is used, an alarm is saved with the start time 20:23:05(h,min,sec) and stop time 20:54:31. The closest related parameters for the start will be at the time 20:20:00 and for the stop 20:55:00 thus the start and stop time will be moved there. This is done in order to have some parameter values to associate the alarms with. What is then known is that during the 5 min time of 20:25:00 one or more parameters reached critical values and the
alarm was triggered. The mean value that is available for study at 20:25:00 is now built from 3 minutes and 5 seconds of trend data that does not correspond to alarm levels and 1 minute and 55 seconds which do correspond to alarms levels. Which can make the trend data at 20:25:00 misleading, making it harder for the algorithm to properly identify what an alarm is. If an alarms duration is shorter than the downsampling time the whole alarm can be misleading for the algorithm making it harder yet again for an algorithm to properly identify what an alarm is.

There are some methods to improve this, unfortunately they can not be applied after the downsampling has already been done but will be listed for further work.

1. Only saving a parameter value when it is changed.
2. When downsampling, also save a measurement of the spread of data, for example: max, min, and/or standard deviation.
3. Create a black box function which saves data with a higher resolution prior to an alarm.

2.2.7 Naming ambiguity

One of the biggest challenges when working with large amount of data is naming ambiguity. When adding together data from different pumping stations you want the algorithm to learn all it can about a parameter, using all the available data about it. If for instance water level is tagged as WL in one station and TLEVEL in another, these needs to be re-tagged to a common tag. If this is not done the algorithm will learn two separate representations between the two tags even though they are the same thing.

The data that is available for this thesis needs to break through several hurdles if all of it shall be used. First, the data from each country has different mother languages which mean that some parameter names will be in that language. Second, three different SCADA system is present, which all can have different standard parameter names with different definitions. Third, each pumping station can be installed by a different operator, or customer which thinks parameters should be named a certain way, and depending on the freedom allowed, implement it.

When it comes to machine learning one of the common knowledges is that the more training data the better result. So by using all the available data for one parameter the results achieved from that parameter can be expected to improve.

2.3 Problems and solutions concerning neural networks

2.3.1 Overfitting

When training a neural network something called overfitting can occur. What this means is that the trained model adopts to well to the training data and performs poorly on previously unseen test data. There are several ways to counter overfitting but the first step is to detect when it happens. This is done by separating the data into three sets:
training, test and validation. Each of the three sets has its own purpose and the sizes of sets are different. The training set usually consists of 80-90% of the total data and is used to train the model. The test set represents new real data that the model has not seen before and is used in order to see how well the model performs in reality. The size of it is the 10-20% that the training set does not occupy.

The validation set has the most uses as it is first used in order to tune the bigger parameters, usually called hyperparameters, such as the number of neurons and layers. Secondly it is used to give an indication of overfit. This is done by evaluating the results on both the training and evaluation set during each training epoch. When the results continue to become better on the training set but worse on the validation set, overfitting is probably happening. When this happens the training is stopped and the epoch which gave the best result on the validation set is used, this technique is called early stopping [14]. If there is a big difference in the results between the training and validation set from the start, it can be an indication that the selected model is too complex, as it is fitting to well to the training set and should be changed to a less complex one. The size of the validation set is usually 10-20% of the training data, but there are some different ways of choosing which 10-20% of the data to use in order to reduce the variance of the result evaluation. One way of creating a validation set is called cross-validation [15] where the training data is divided into \( n \) number of equally sized portions where each portion is in turn used as validation set while the rest is training data. The mean value of the different validation sets is then used to represent the sets result. This gives a more accurate measure on the validation set but it is also \( n \) times slower, since the model needs to be trained \( n \) times more.

Since a neural networks is a relatively complex model, several techniques have been developed specifically for neural networks to counter overfitting. One of the more successful ones being dropout[16] which temporarily freezes a percentage of the neurons during a batch training, this forces the remaining neurons to move towards a new solution where the neurons are more independent of each other. Dropout is successful both because it manages to reduce the overfit but also because it’s able to produce better end results for a neural network. When using dropout it is recommended to increase the number of neurons according to \( \frac{n}{p} \) where \( n \) is the optimal number of neurons without dropout and \( p \) is the percentage of neurons that is not frozen on average. The percentages of neurons that are temporarily frozen are drawn from a Bernoulli distribution; this creates a variance on how many neurons that are frozen each time.

Another way to reduce overfit in a neural network is to either put a limit on how large a weight can become or to punish larger weights. The technique of putting a limit on how large a weight can be is called max-norm, which is the recommended one to use with dropout [16]. By setting a limit the network is forced to not overly rely on a few neurons which can cause less overfit. The other type of regularizer punishes larger weights by attaching a penalty term to the weights that is added to the loss function. The two most common ones being \( L1 \) and \( L2 \), the equations for them can be seen in formula 2.5 and 2.6 respectively, where \( f(x) \) is the loss function, and \( c \) is an adjustable constant that decides how hard to punish large weights.

\[
f(x) + c \left( \sqrt{\sum_{i=0}^{N} W_i^2} \right)
\] (2.5)
The size of the weights is then regularized either linearly with $L_1$, or quadratically with $L_2$. Because it punishes the weights differently, the two regularizes have different properties. The $L_2$ regularizer is much harsher to single weights being much higher than others which cause the weights values to be closer to each other. The $L_1$ punishes the weights linearly which means that if one weight is decreased by 2 another weight can be increased by 2 without changing the cost. The $L_1$ regularizer is thus able to keep some weight large while setting others to zero, creating a more sparse network.

One of the best ways of reducing overfitting is however to increase the amount of training data as this will increase the variation of the data thus making a neural network less prone to overfit. Increasing the amount of training data usually yields better results overall with neural networks but more data is not always available or can be expensive to get which can make this a hard solution to use.

### 2.3.2 Number of neurons and layers

From the three different types of layers of a neural network (input, output, and hidden) it is only the hidden layer in which you are required to decide the number of neurons and layers. This is a challenging task as each problem can require a different amount of neurons and layers. The reason for this is that if the number of neurons are too low there is a risk that the network fails to capture the events in the data and produces a poor result, this phenomena is called underfitting. If the number of neuron is instead too large, overfitting will occur. There are no clear rules or methodical way of deciding the number of neurons and hidden layers but Jeff Heaton [17] gives some rules of thumb to follow:

1. The number of hidden neurons should be between the size of the input layer and the size of the output layer.
2. The number of hidden neurons should be $2/3$ the size of the input layer, plus the size of the output layer.
3. The number of hidden neurons should be less than twice the size of the input layer.

He also gives some insight on how the number of hidden layers should be chosen: “Problems that require two hidden layers are rarely encountered. However, neural networks with two hidden layers can represent functions with any kind of shape. There is currently no theoretical reason to use neural networks with any more than two hidden layers. In fact, for many practical problems, there is no reason to use any more than one hidden layer.”

Since this book was written 2008 a lot has happened in the neural networks especially within image recognition as in 2010 an image recognition competition was started called Large Scale Visual Recognition Challenge (ILSVRC), as this was considered a hard task. The first winner of the competition using neural networks, Alex Krizhevsky et al. [18],
used 8 layers. It could be seen consistently over the years the competition run, that the addition of more layers improved the result. In order to see if this was indeed the truth Sergey Zagoruyko et al [19] did an experimental study where their original claim was that width would improve the results more. This was the case as they managed to decrease the number of layers on one network from 1000 to 16, and managed to improve the result. The summarized wisdom they could draw from this experiment on the datasets they tested was the following:

1. widening consistently improves performance across residual networks of different depth;
2. increasing both depth and width helps until the number of parameters becomes too high and stronger regularization is needed;
3. there does not seem to be a regularization effect from very high depth in residual networks as wide networks with the same number of parameters as thin ones can learn same or better representations. Furthermore, wide networks can successfully learn with a 2 or more times larger number of parameters than thin ones, which would require doubling the depth of thin networks, making them unfeasibly expensive to train.

Both Jeff Heaton and Sergey Zagoruyko et al. came to the same conclusion regarding the depth of a network which is that the more complex the data is, the deeper the network should be, otherwise width contributes more.

There are however different effects of adding more layer to a network depending on what kind of neuron that is used. The kind of network that Sergey Zagoruyko et al. worked with, and was the focus of Jeff Heaton’s book, was feed forward networks which worked with data that did not have time dependence. In order to better model data with time dependence the more complex neuron RNN or LSTM can be used. What an addition of a layer does for those neurons is that it also allows them to create a memory of the abstract representation of the previous layer which can make the neural network remember more complicated time patterns [20].

Another good approach when choosing the architecture of a neural network is to find a similar problem that has been solved with a neural network and adopt the structure. The adopted structure can then be adjusted to better fit with the current problem.

2.3.3 How the results are evaluated

There are several different parameters that can be used in order to represent the results, some of the more common ones being: root mean square error (RMSE), mean error(ME) and mean relative error (MRE). It is important to choose one that is representative and useful in both the academic world and the company Xylem. For instance the measurement RMSE is good when comparing to other academic reports as it is often used but telling Xylem that one alarm can be predicted with RMSE=5 does not really tell them much.

After discussing with my supervisor at Xylem, a few measurements were extra interesting for them, namely:
Chapter 2. Background

1. Chance that it is a false alarm
2. Probability that an alarm will start in n time steps

The important thing to note here is that it is the start time of an alarm that is of importance and not the time it is active. One goal of the thesis is to look several time steps into the future in order to predict an alarm. Evaluating how far the neural network can predict during each epoch would give the best possible network among the ones tested. This does however take an infeasible long time to compute during training. If one pump station has logged its data with a 5 min resolution and there exist data for 2 years, there are approximately 210000 samples from one pumping station. In order to see how well it predicts 1 month (8640 time steps) forward in time with a recursive model it needs to do 8640 predictions for each of the 210000 samples in the data. One prediction for 8640 time steps forward takes approximately 5 seconds on my current computer which would put the total amount of time for one station to approximately 12 days. As such this is a measurement which is too expensive during training but can be employed to a fully trained model.

When trying to predict into the future it gets harder the more precise one has to be. In order to make the model more usable it can be wise to include a measurement of the precision which could be the mean time difference between the alarm happening and the predicted start.

2.4 Related work

Mayadevi, N. et al. [21] used SCADA data in order to create a Power Plant Equipment Fault Forecasting system. There are two things that make this report interesting; the first is that the data is probably similar to the data used in this thesis. The other thing is that they try to forecast the fault which is very similar to what this thesis is attempting. This was solved using an automatic rule creation system mixed with expert knowledge in order to classify what to alarm about and an artificial neural network in order to predict. The evaluation metric employed measured the mean absolute error and root mean squared error but it seems to be done on the same data as the training was done on which gives a high risk of overfitting. The work still gives an indication that it is possible to use neural networks to predict errors in an industrial application.

The following works compared different types of neurons and architectures that could be used for predictive models.

Zheng Zhao et al. [22] used LSTM in order to forecast the traffic in the fifth ring road in Beijing. In the report it is compared to the general RNN, SVM, ARIMA model, RBF networks and SAE model. From this it can be seen that the LSTM is better at forecasting the traffic especially when looking further into time. The furthest they looked was 60 minutes on data that was sampled at 5 minute intervals. This was measured in Mean Relative Error (MRE) which was between 16.25% to 17.63% error when measuring 60 minutes into the future which can be compared to RNN which was between 20.36% to 29.64% error. This work proves the superior capabilities of LSTM compared to RNN when it comes to forecasting.
Daniel L. Marino et al. [23] used LSTM to forecast the building energy load testing both 1 minute time steps and 1 hour time steps. What they discovered was that the standard LSTM performed poorly on the 1 minute time steps because the input and output was too similar to each other which created a naive network. They managed to solve this by changing the architecture of the network so it became a sequence to sequence network where one part was an encoder and the other the decoder. When this was done they managed to forecast 60 hours ahead with a small error rate.

S.L Ho et al. [24] did a 2002 comparison between recurrent neural network(RNN), multilayered feed forward neural network(MFNN) and Box-Jenkins autoregressive integrated moving average (ARIMA) in order to predict system failure. It could be seen that ARIMA and RNN performed well on short term forecasting but had some trouble on long-term forecasting. The MFNN was inferior in both long and short term forecasting.

Frank, R. J. et al. [25] used a feed-forward neural network with a sliding window in order to predict the next value. This was done on 4 different datasets and the influence of different sized windows was studied. It could be seen that it was no easy task to decide the optimal window size and that it had a big impact on the result where the difference could be several percentage points.
Chapter 3

Method

This chapter further describes the data and the manipulations done to it, what type of neural network was chosen and why. It also goes through the chosen method, its settings and what tests that will be run.

3.1 Evaluation process

The different problems introduced in Chapter 2 were looked at first separately and then a base solution of the neural networks architecture was decided upon, with each of the separate problems making adjustments to the network in order to improve it.

When it came to the base architecture the switch alarms and the related works had the biggest influence and it was decided that LSTM neurons would be used. The reason for this was that it was deemed to be a superior type of neuron when it came to both predicting and remembering previous events. It should also be able to solve the problem of different sample speeds without any changes. LSTM is also able to model time series which is one of the properties of the data.

The chosen solutions to the problems introduced by the data are presented in section 3.2 and the chosen solutions introduced by a neural network are presented in section 3.3.

3.2 Data

The different datasets from the four different countries were each studied separately and because of the sometimes large differences between their formats, it was decided that data from only one country would be used for the thesis. The criteria for choosing which data to use was the following:

1. Low naming ambiguity
2. High resolution
3. Large amount of alarms
4. Large amount of data
5. Understandable format

The criteria which had the highest priority when choosing the dataset to use was low naming ambiguity and understandable format. The dataset which best fulfilled these demands was the data obtained from Country A and was the one used for this thesis. No detailed comparison will be given in this report about the different datasets because of commercial concerns.

The country A dataset was saved into three sets, one for trend data such as water level, one for meta data such as the depth of the pumping station, and one for history of alarms. It was decided to only use the trend data as input to the network and if necessary use the meta data for normalization. In some early test runs it was seen that if the alarms data was used as input the neural network would only learn that if an alarm has happened it will continue to happen thus not learning to predict.

It was quickly seen during some initial test runs that the time it took to train the network would increase linearly with the amount of training data. Because of this some different approaches to the training data were tested in order to reduce the training time. The first step was to remove input parameters to the neural network. My supervisor at Xylem thought it would be a bad idea to remove parameters that I did not believe would help the network and came with the idea that it can be good to know if a parameter is active. Because of this I decided to only remove parameters that had a high naming ambiguity, not used in more than one station, or never used. The amount of time saved on this was minimal, which is probably because the neural network handles the input in parallel.

The next thing I tried was changing to delta logging where a sample was removed if it was identical to the previous one. This proved to be very inefficient because if even one out of all the parameters was different the sample would remain, this lead to less than 5% of the samples being removed which was deemed too small of a number and thus delta logging was not used.

The third approach tested was to only use data around alarms as training data. This managed to greatly reduce the time it took to train as there was only a small amount of time in pumping stations where alarms occurred. However this approach can increase the chance of false alarms as the network has seen fewer areas where no alarms occurs, and the prediction of trend data can become worse as it has less data to train on, further details in Section 3.3.

The problem with different dimension of the output was solved using zero padding which can be read about in Section 2.2.1.

For normalization it was decided to use $\max - \min$ scaling because it is able to push the data to be within the activation functions range used in the neural network. However some extra normalization was done on the trend parameter water level because it is logged in almost every pumping station, is a very important trend parameter, and it is logged in a unique way in the country A dataset. The way country A logged the water level was meter above nautilus level which makes the parameter very hard to use between pumping stations as they are all on different height from nautilus level. In order to make it comparable between the pumping stations a prior normalization was
done on each pumping station separately, where the high level alarm value was used as maximum, and the minimum water level of the pumping station read from the meta data was used in a \( \text{max} - \text{min} \) scaler. After this was done, all values that were obviously wrong were set to \(-0.1\). The water level was then normalized again but now together with all of the other parameters. The reason for using the high level alarm value is that this can be set differently between stations but in order to make it easier for the neural network to associate a water level to an alarm it was decided that it was to be used instead of the maximum water level of a pumping station.

The dataset from country A had an even downsample rate of 5 minutes which made it easier to work with. Because it has an even downsample rate it was decided that no adjustments should be made concerning time dependencies in the data.

As previously stated the parameters that had a high naming ambiguity was removed, one of them were \( \text{TANA4} \) which stands for: trend data, analogue data in input 4, where the analogue input can be any parameter, and the trend data parameter \( \text{TCOUNTER} \) was also removed as it was not known if it counted the same thing between stations.

### 3.3 Chosen method

The base of the neural network was as previously stated decided to be LSTM but in order to better use the training data to predict both the trend data and the alarms it was decided that two neural networks would be created, one for predicting the alarms, and one for predicting the trend data. This also made it easier to evaluate the results for predicting alarms and trend data.

Both of the networks are trained under the same principle of only predicting one time step ahead based on trend data. In order to predict further into the future the trend network is used recursively where it predicts one sample which is then used in order to predict the next sample which is illustrate in Figure 3.1. The alarm network is then able to use the samples that the trend network has predicted in order to make its own prediction making it possible to predict several time steps.

In order to train the networks to their maximum potential the use of training data was different between the two neural networks. The trend network uses all of the training data available while the alarm network uses the data cut out around the alarms. The trend network is then able to draw experience from a vast amount of data where all of it is relevant, while the alarm network gets experience from the relevant parts of the training data while keeping the training time lower.

There was one more reason for training the alarm network with cut out parts around alarms, the reason being that the network would very often get stuck in a local optimum where it would predict that no alarms happened at all. The reason for this was believed to be that there was too few alarms happening, when all of the data was run only 0.06% of it contained alarms. Thus when the area around an alarm was cut out two new parameters was introduced; time-prior an alarm and time-after an alarm.

The time-prior and time-after alarm settings were decided using a trial and error approach where the parameters were lowered until the network could get out of the local optimum. The network used was a one layered LSTM with 10 neurons and a
dropout layer where the dropout parameter was set to 0.2. It was seen that the time-after parameter was connected to the batch size where the time-after parameter had to be at least batch size +1. This was because the training data length had to be dividable by the batch size, otherwise data had to be removed, and it was not desirable to remove data where an alarm happened.

Using trial and error, where the ratio of active alarms in the training data was gradually increased, it could be seen that the network would get stuck in a local minimum when less than 2.06% of the data was marked as active alarms and it managed to get out of the local minimum when 2.44% of the data was active alarms. This means that the threshold for getting stuck in a minimum is somewhere between 2.06% and 2.44%. In order to ensure that it was not too close to the threshold it was set so that 3.06% of the data was marked as active alarms. This gave a time-prior of 2 days (576 time steps) and time-after of 12 hours (132 time steps). Since the time-after was set to 132 time steps it limited the batch size to a maximum of 131. The alarms are tagged as ones when active and zeros when inactive.

What this time restriction in the training data means is that the network for alarms will only be trained to use data from 2 days back when predicting if an alarm will happen the next five minutes. This limitation will not exist in the neural network which predicts the trend data, because there are more variations in the data which causes more errors forcing the network to create a solution which is not a constant value.

So by using two neural networks we have one for the alarms which is time-restricted in how far back it can see and one neural network for trend data which does not have any restriction in how far back it can see. How far back it actually uses values is however unknown.
The two different networks will be built and optimized separately in order to save time, but will work together to predict alarms.

3.3.1 Neural network configurations

There are some different settings that can be done on a neural network which can improve the results and also reduce overfitting. This section will discuss the different settings, which of them that need to be tested, and which of them that are taken from literature. When the parameters had to be tested to be found it was run on a smaller subset of the data, using 7 stations as training data and 2 as validation.

3.3.1.1 Batch size

A bigger batch size generally increases the speed of how fast the network trains, each time the network runs through a batch of training data it updates its weights. This leads to the weights being updated fewer times per pumping station when a high batch size is used, making it faster, but it also means that the pumping stations needs to be run through more times, leading to a bigger number of epochs.

A correctly set batch size can also increase the performance of a neural network. A recommendation for LSTM is to set the batch size to be the same length as the patterns in the data. For the alarm network this can be approximated to be the same length as the alarms. The mean, median and standard deviation time were calculated for the alarms, which gave the following: mean=5.9, median=1, std=53.0. This shows that the majority of the alarms are short with the length 1 but there exists alarms which are much longer. In order to ensure that the network uses the information from the short alarms a low batch size of 8 is set.

For the trend data there is no known pattern length which means that this parameter has to have tests run in order to find the optimal batch size. In order to speed the testing up the batch size will be set to a bigger value of 4096 when the other parameter are tested such as neurons, and layer. Once all the other parameters have been tested the optimal batch size will be searched for.

3.3.1.2 Activation functions, loss function and optimizer

As mentioned in [13] no changes to the original structure of the LSTM improved its performance significantly, thus no changes will be made to its activation functions. The output layers activation function can however be optimized for the two neural networks.

For the alarm network the alarms are tagged as 1s when active and 0s when inactive, this forces the output activation function to be able to force the output values to be between or equal to one and zero. A sigmoid can be used for this but it has the unwanted behavior, in this case, of never being exactly one or zero but going towards it. Because of that a variation of it will be used called hard sigmoid which is linear between one and zero with approximately the same incline as the sigmoid but is exactly one or zero at the borders. This allows the network to be exactly correct when predicting an alarm instead of almost correct as it would be with a normal sigmoid.
For the trend network it was decided that two different activation functions would be tested, the sigmoid and the linear activation function. The reason for this is that when doing regression it is common to do linear regression where the output is unbounded. While the sigmoid bounds the output between zero and one, which demands that the data is normalized between zero and one.

The same optimizer will be used for both of the networks namely Adam. The creators[8] of the optimizer Adam compared it to a lot of other optimizers on different datasets and could see that it was superior. They also managed to find a set of default setting which worked well on several datasets; those settings will be used for this thesis as well and they can be found in [8].

The loss function used will be the same for both networks, and will be the mean squared error, it punishes big errors extra hard which is a wanted behavior for both of the networks.

3.3.1.3 Neurons and layers

There is no set number of neurons or layers that is known to be optimal for different problems. This means that it needs to be tested, there are however some tips and rules of thumb that can be used for the starting guess, these are mentioned in section 2.3.2.

In order to find how many neurons to use at least 4 tests needs to be run on each network with n-layers. First a test with the guessed most optimal number of neurons, then two more tests where the number of neurons is raised by m or lowered by m. The fourth test will then be towards the direction where the best network was found. For example if the network with m additional neurons achieved a better result than the initial guess then the fourth test will be with 2m more neurons then the initial test. This will continue until an increase or decrease of neurons does not give a better network. The first test with the number of neurons will be done with one hidden layer.

After the test for the optimal number of neurons is done, the number of layers will be increased by one and the optimal number of neurons found from the first test will be split between the layers. Then the optimal number of neurons test will be remade with the starting point being the previously found optimal number of neurons. During this testing dropout and regularisation is turned off and the results is evaluated on the validation set.

3.3.1.4 Dropout and regularisation

In the original paper for dropout[16] it is recommended that a weight regulizer is employed together with dropout. Three different regularizers will be tested, max-norm, L1, and L2.

The two networks will have the same tests run in order to find the optimal setting. First a dropout layer is added between the LSTM layers and the output layer, with a dropout value increasing from 0.1 to 0.9 with a step size of 0.1. When the optimal value has been found the three different regularizers are tested with the optimal dropout value in order to see which of them gives the best result.
3.4 How the final test is constructed

Once the optimal settings for the two networks have been found, all of the data from country A will be used in the training, validation and test set. The split will be 20% test, 80% training and 10% of the training data as validation set.

When both of the networks have been trained, the trend network will step through the training set in order to give the alarm network values to predict on. The trend network will at time step \( t \) recursively predict \( n \) time steps ahead which the alarm network will use to predict alarms. Once this is done the trend network goes to time step \( t+1 \) and the predictions are done again.

It was seen during some early test runs that this approach took a lot of time as it took 5 seconds to predict 1 month (8640 time steps) with the trend network. Since the full data from a pumping station was around 220000 time steps it would take approximately 13 days to run through one pumping station. Because of this it was decided that the prediction would be limited to 1 week into the future and the result would only be evaluated on the areas where the alarms happened and some random areas where no alarm happened. This was done in order to make the testing feasible to complete in a reasonable amount of time while still giving a representative result.

3.4.1 Result evaluation

In order to evaluate the results both academically and for the understanding of Xylem two different metrics will be used, one for the training of the network and one for presenting the end results. The metric used for training the network will be root mean squared error (RMSE) as it fits well with the loss function used in the network and it is a well-used measurement in the academic world. Notice that the metric used during training will only measure one time step into the future.

The reason for using a different metric for the end result is both to increase its understandability, and to make it more detailed. Making it more detailed makes it slower to compute which is not a wanted behavior during training.

The measurements used for the end results will be precision, missed alarms and accuracy. The definitions used and its help definitions are explained below:

\[
\text{Prediction} = \begin{cases} 
P(n) = 1 & \Rightarrow \text{Algorithm predicts an active alarm } n \text{ time steps into the future.} \\
P(n) = 0 & \Rightarrow \text{Algorithm predicts no active alarm } n \text{ time steps into the future.}
\end{cases}
\]

\[
\text{Accuracy} = \begin{cases} 
P(n) = 1 \text{ and there is an alarm within the region } [n - n/4, n + n/4] & \Rightarrow \text{Hit}(n) \\
\text{Hit}(n) \\
\text{Hit}(n) \\
\text{Accuracy}(n) = \frac{\sum \text{Hit}(n)}{\sum \text{Hit}(n) + \sum \text{False}(n)}
\end{cases}
\]
\[ Misses(k) = \begin{cases} 
  if \ P(n) = 0 \text{ For the whole region where it could hit.} \\
  Misses(k) \text{ increases by one, where } k \text{ is the distance to the alarm.} 
\end{cases} \]

\[ Precision(n) = \{ \text{Mean distance between alarm and hits } n \text{ time steps into the future.} \] 

If the precision is assumed to follow a normal distribution its standard deviation can be used to determine how big part of the hit region is used, which can be a good measurement to have when deciding the size of hit region for future optimizing.
Chapter 4

Implementation

This chapter describes the software used and which hardware it was implemented on. It also discusses changes made to the chosen method during implementation.

4.1 Experimental Setup

This section describes the hardware and software used to implement the solution.

4.1.1 Hardware

The implementation was done on the stationary computer Dell XPS-8910 which had 16 GB ram, Intel core i7-6700k CPU, 256 GB SSD, 2TB normal hard drive and a Geforce GTX 1080 with 16 GB memory. All of the programs were run on the SSD disk to increase the speed, and the GPU was used to increase the training speed of the network.

4.1.2 Software

The programming language used was Python and in order to help the implementation it was installed with Anaconda which includes several scientific toolboxes. In order to implement the neural network the toolbox Keras was installed with the backend Theano which according to [26] was the fastest when it came to running a LSTM network.

Most of the practical problems I encountered using Keras I was able to solve using the different guides Jason Brownlee[1] had written.

4.2 Changes made to the chosen method during implementation

Some problems or mistakes done late in the project are presented here. The different ways used to solve it is also presented with the corresponding result.
4.2.1 Limitations with Keras

The toolbox Keras had almost all off the functions needed in order to properly implement the solution. There were some custom functions that had to be created in order to get the implementation performing as wanted. The first problem was that if a network had been trained with batch size $k$, where $k > 1$, it would predict $k$ times based on true data one step into the future, instead of predicting only one step based on $k$ values. The batch size was not a parameter that could be changed on a finished trained network, in order to change the batch size to one, so that the network could predict one time step recursively, the weights had to be copied to a network with the same structure but with the batch size set to one.

When predicting $n$ time steps from time $t$ the hidden memory states would get updated so that they corresponded to time step $n+t$. So when the next prediction was to be done from $t+1$ the hidden memory states from $t$ had to be loaded. There were no functions in Keras for loading and saving the hidden memory states so these where written manually and inserted into my source code.

4.2.2 Re-label data

The alarms in the dataset had a start and stop-time marked, it came to my attention quite late that the stop-time that an alarm was marked with might not be the correct stop time because it could work according to two different models. The first one being that when the parameters that had triggered the alarm went back to normal values the stop-time was set. The second one being that the stop-time is set when an operator has time to look at it and see that the alarm is no longer active.

The problem arises when the stop-time is set according to the second way as this will falsely mark data when no alarm is happening as an alarm. This in turn will make it much harder for an algorithm to correctly identify what an alarm is, and if the algorithm can not identify what an alarm is it can not predict it either.

4.2.2.1 Approach one

In an attempt to solve this, semi-supervised learning was introduced in order to re-label the alarm data so that it signaled for alarm only when it was an actual alarm. The algorithm used was nearest neighbor where the sample at the start time was set as alarm. The sample one time step before the start time, and one sample one time step after the stop time was set as noalarm. All the other points were set to unknown which the algorithm had to label.

This approach gave non satisfactory results where all the of the unknown samples were either classified as alarm or no alarm. This is believed to come from the fact that the data was downsampled where somewhere within a 5 minute interval one or more values triggered the alarm while all the algorithm saw was the mean. Semi-supervised learning was thus decided to not be used to re-label the alarms.
4.2.2.2 Approach two

A simpler approach was instead used and only one type of alarm was re-labeled. The alarm *high level* was re-labeled to only be *high level* when the *water level* was above its set threshold which was obtained from the meta data.

The re-labeling was only done for the final test, where all of the data was used.

4.2.3 Data cleaning

Not all of the available parameters were used. In order for a parameter to be included in the ones sent to the algorithm a set of conditions had to be fulfilled in order to ensure that they will be helping the algorithm, and not confuse it and increase the training time. If a parameter did not fulfill all of the conditions from *List1* it was removed.

**List1: General data requirements**

- It has to have a tag which explains what it is.
- The tag needs to explain the same value across several pumping stations.
- It should not be an aggravated parameter that only counts up.
- The thing the parameter is measuring should actually be used/plugged in.
- The parameter should exist in more than 2 stations.

Alarms were also removed either because there were too few of them, or because it was not an interesting alarm to predict. Pumping stations were also removed if they did not fulfill all of the conditions from *List2*.

**List2: Pump station requirements**

- The pumping station has alarms that has happened.
- The pumping station does not have any unique names on the parameters.
- The pumping station only contains one sump.
- No obvious mistake in the tagging of the data.
- It has tags for its data.

The order and procedure in which everything was removed was the following. The first step was looking at how many times the alarms actually happened across the pumping stations. A threshold was set so that if an alarm happened less than 10 times across the pumping stations it was removed. Then a list with the 20 most common alarms was created which was went through with my supervisor at *Xylem*, and the alarms of no interest were removed. Next the pumping stations which did not fulfill the conditions
from List2 were removed. A new list was then created with the alarms that were still in the data and together with my supervisor from Xylem the alarms which were of no interest were removed. All of the different kinds of tagging for the alarm high water level were also renamed into one tag instead. And finally any parameter that did not fulfill the conditions in List1 was removed.

This lead to 9 out of 23 pumping station surviving the culling which was used in order to find the optimal settings for the two networks. A larger data set of 105 stations was obtained for the end test and out of them 75 survived the culling. The 84 stations that survived the culling took approximately 65 gigabytes of space on the hard drive.

The trend parameters that were left after the culling were the following:

- **TLEVEL** - the water level of the sump.
- **TOVERFLOW** - how many liters of water that overflows the sump per second.
- **TP1CURRENT** - current used by pump 1.
- **TP2CURRENT** - current used by pump 2.
- **TP1CAPACITY** - a measurement of how much water pump 1 manages to pump at the moment
- **TP2CAPACITY** - a measurement of how much water pump 2 manages to pump at the moment
- **TINFLOW** - how many liters per second that is flowing into the pumping station, either it is approximated or measured with a flow meter
- **TEXTOVERFLOWLEVEL** - at what level the pumping station overflows, when overflow happens the sensor that measures water level is recalibrated according to this value.

and the alarms still left were, note that all of the different names for high level alarm were re-labeled as A_1:

- **A_1** - High level of water, the water level goes above a set threshold
- **A_11** - Motor thermal failure in pump 1
- **A_12** - Motor thermal failure in pump 2
- **A_15** - High current in pump 1
- **A_17** - High current in pump 2
- **A_34** - Water overflow
- **A_36** - Pump 2 high temperature
Chapter 5

Analysis and results

This chapter describes the results together with an analysis.

5.1 Tests to determine trend network structure

The tests run in order to determine the structure of the trend network is here presented and discussed. The final chosen network is mentioned in the end.

5.1.1 Activation function

During this test the hidden layer consisted of eight LSTM neurons. The two activation functions tested against each other were sigmoid- and linear-activation functions, which was tested on the output layer. 100 epochs was run with the linear-activation function and it could be seen that it converged after 19 epochs with a RMSE of 0.01780 on the training data, and a RMSE of 0.01015 on the validation set.

When the sigmoid was used it could quickly be seen that it had not converged after 100 epochs so it was increased to 400 epochs. After 349 epochs it was believed to have converged with a RMSE of 0.01500 on the training data, and a RMSE of 0.00841 on the validation data. In order to be sure another run was done with 2000 epochs and it could be seen that the training error continued to decrease slowly to 0.013321 while the validation error went up and down usually between 0.00843 to 0.00853. From this is can be seen that the first optimum found after 349 epochs is probably a local minimum and a better solution can be found after more epochs.

From these tests it can be seen that the sigmoid activation function is a lot better when it comes to results but is also a lot slower. It took the network with the sigmoid function 107 epochs before it had a similar performance to the linear. Since the results is more important than speed, the sigmoid function will be used in the further tests but the number of epochs will be limited to a maximum of 400 in order to save time.
5.1.2 Number of neurons and layers

The testing for the number of neurons was done as discussed in Section 3.3.1.3 where the three first networks tested were 4, 8, and 16 number of neurons in the hidden layer. After that the next network tested was towards where the optimal solution can be, as can be seen in table 5.1 the results got better with a larger network compared to 8.

Table 5.1. Results for using one hidden layer.

<table>
<thead>
<tr>
<th>n-neurons</th>
<th>n-layers</th>
<th>n-epochs</th>
<th>min validation RMSE</th>
<th>training RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>224</td>
<td>0.01028</td>
<td>0.01774</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>349</td>
<td>0.00841</td>
<td>0.01500</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>400</td>
<td>0.00772</td>
<td>0.01201</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>242</td>
<td>0.00853</td>
<td>0.01757</td>
</tr>
</tbody>
</table>

Each test was run for 400 epochs with the best epoch result being displayed in the table.

As can be seen in table 5.1 16 neurons in the hidden layer gave the best result. It can also be seen that it has not yet reached its optimum as 400 was the maximum number of epochs in that test run. The number of layers was then increased by one and the test repeated, this time starting with 16 neurons as the minimum.

Table 5.2. Results for using two hidden layers.

<table>
<thead>
<tr>
<th>n-neurons</th>
<th>n-layers</th>
<th>n-epochs</th>
<th>min validation RMSE</th>
<th>training RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>8/8</td>
<td>2</td>
<td>222</td>
<td>0.00842</td>
<td>0.01631</td>
</tr>
<tr>
<td>16/16</td>
<td>2</td>
<td>202</td>
<td>0.00820</td>
<td>0.01461</td>
</tr>
<tr>
<td>24/24</td>
<td>2</td>
<td>104</td>
<td>0.00817</td>
<td>0.01489</td>
</tr>
<tr>
<td>32/32</td>
<td>2</td>
<td>354</td>
<td>0.00827</td>
<td>0.01317</td>
</tr>
<tr>
<td>40/40</td>
<td>2</td>
<td>143</td>
<td>0.00825</td>
<td>0.01431</td>
</tr>
</tbody>
</table>

Each test was run for 400 epochs with the best epoch result being displayed in the table. The first test was done with 8 neurons in both of the hidden layers, and the next test with 16 neurons in both of the hidden layer and so on.

From table 5.2 it can be seen that the addition of an additional hidden layer did not improve the performance of the network. From this it can be seen that the best network out of the ones tested was one hidden LSTM layer with 16 neurons.

5.1.3 Dropout and regularization

A dropout layer was then added between the LSTM layer and the output. The number of neurons in the hidden layer was then increased according to \( \frac{16}{p} \) where 16 was the optimal number of neurons in the ones tested and p is the parameter that will be tested. The number of epochs is still limited to 400 even though it was shown in the number of neurons test that it still had not converged after 400 epochs. This is done in order to save time as this test is expected to take 17 hours. The results can be seen in Table 5.3.
Table 5.3. Result for the dropout test.

<table>
<thead>
<tr>
<th>$p$</th>
<th>n-epochs</th>
<th>min validation RMSE</th>
<th>training RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>275</td>
<td>0.00716</td>
<td>0.00990</td>
</tr>
<tr>
<td>0.2</td>
<td>316</td>
<td>0.00776</td>
<td>0.01022</td>
</tr>
<tr>
<td>0.3</td>
<td>105</td>
<td>0.00775</td>
<td>0.01300</td>
</tr>
<tr>
<td>0.4</td>
<td>365</td>
<td>0.00793</td>
<td>0.01188</td>
</tr>
<tr>
<td>0.5</td>
<td>159</td>
<td>0.00819</td>
<td>0.01416</td>
</tr>
<tr>
<td>0.6</td>
<td>400</td>
<td>0.00873</td>
<td>0.01447</td>
</tr>
<tr>
<td>0.7</td>
<td>273</td>
<td>0.00979</td>
<td>0.01678</td>
</tr>
<tr>
<td>0.8</td>
<td>256</td>
<td>0.01295</td>
<td>0.02052</td>
</tr>
<tr>
<td>0.9</td>
<td>379</td>
<td>0.01632</td>
<td>0.02577</td>
</tr>
</tbody>
</table>

Each dropout $p$ value was run for 400 epochs with the best epoch result being displayed in the table.

It can be seen that the best setting for dropout was $p = 0.1$. This setting is now used when testing regularizers. The three tested are max-norm, $L_1$, and $L_2$ with few variations of their settings. The results for each of the regularizers can be seen in Table 5.4, where it can be seen that the addition of a regularizer did not improve the results.

Table 5.4. Result for the regularizers test.

<table>
<thead>
<tr>
<th>Max-norm constant</th>
<th>n-epochs</th>
<th>min validation RMSE</th>
<th>training RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>375</td>
<td>0.00895</td>
<td>0.01397</td>
</tr>
<tr>
<td>3.5</td>
<td>327</td>
<td>0.00894</td>
<td>0.01366</td>
</tr>
<tr>
<td>4</td>
<td>294</td>
<td>0.00864</td>
<td>0.01308</td>
</tr>
<tr>
<td>$L_1$ constant</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0001</td>
<td>82</td>
<td>0.02253</td>
<td>0.03700</td>
</tr>
<tr>
<td>0.001</td>
<td>68</td>
<td>0.02302</td>
<td>0.03600</td>
</tr>
<tr>
<td>0.005</td>
<td>120</td>
<td>0.02530</td>
<td>0.03647</td>
</tr>
<tr>
<td>0.01</td>
<td>106</td>
<td>0.02572</td>
<td>0.03666</td>
</tr>
<tr>
<td>0.05</td>
<td>168</td>
<td>0.02605</td>
<td>0.03680</td>
</tr>
<tr>
<td>$L_2$ constant</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0001</td>
<td>149</td>
<td>0.02251</td>
<td>0.03668</td>
</tr>
<tr>
<td>0.001</td>
<td>106</td>
<td>0.02206</td>
<td>0.03788</td>
</tr>
<tr>
<td>0.005</td>
<td>86</td>
<td>0.02258</td>
<td>0.03773</td>
</tr>
<tr>
<td>0.01</td>
<td>57</td>
<td>0.02274</td>
<td>0.03776</td>
</tr>
<tr>
<td>0.05</td>
<td>365</td>
<td>0.02292</td>
<td>0.03701</td>
</tr>
</tbody>
</table>

Each regularizer was run for 400 epochs with the best epoch result being displayed in the table.

5.1.4 Batch size

The batch size was tested with the optimal network found from the previous tests. Since the other settings for the network were tested first with a batch size of 4096 it is possible that this test was biased for 4096 as the other parameters were optimized for this. The results can be in table 5.5.
Table 5.5. Result for the batch size test.

<table>
<thead>
<tr>
<th>Batch size</th>
<th>n-epochs</th>
<th>min validation RMSE</th>
<th>training RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>87</td>
<td>0.00770</td>
<td>0.01237</td>
</tr>
<tr>
<td>512</td>
<td>31</td>
<td>0.00832</td>
<td>0.01525</td>
</tr>
<tr>
<td>1024</td>
<td>86</td>
<td>0.00773</td>
<td>0.01516</td>
</tr>
<tr>
<td>2048</td>
<td>396</td>
<td>0.00753</td>
<td>0.00997</td>
</tr>
<tr>
<td>4096</td>
<td>275</td>
<td>0.00716</td>
<td>0.00990</td>
</tr>
<tr>
<td>8192</td>
<td>399</td>
<td>0.00764</td>
<td>0.01000</td>
</tr>
</tbody>
</table>

Each batch size was run for 400 epochs with the best epoch result being displayed in the table.

The best trend neural network among the ones tested was a one layered LSTM network with 160 hidden neurons with a dropout layer with \( p = 0.1 \), an output layer with sigmoid activation functions, and the batch size set to 4096.

5.2 Tests to determine alarm network structure

The tests run in order to determine the structure of the alarm network are presented and discussed. Before the validation score is calculated the values are rounded to the closest zero or one. The final chosen network is mentioned in the end.

5.2.1 Number of neurons and layers

When one hidden layer was used the number of neurons tested was 1, 2, 3, 4, 7, 10, 13, 16, 32 and the only time the network went out from the local minimum of only predicting no alarms was when 2 neurons were used. This made 2 neurons the obvious choice. Another layer was added and 4 different structures were tested \((1,1), (1,2), (2,1), \) and \((2,2)\) all of them got stuck in the local optimum of only predicting no alarms.

5.2.2 Dropout and regularization

The first test done with dropout resulted in only two networks being able to get out of the local optimum of only predicting no alarms. The \( p \) values for these networks were 0.2 and 0.6 with their respective number of neurons 10 and 3. The networks was each run for 200 epochs and the best result was achieved by \( p = 0.2 \) which gave an validation RMSE of 0.00977, training RMSE of 0.08357 after 8 epochs. The reason for the training RMSE being higher than the validation RMSE is that the training RMSE was run on the cut out sequences around an alarm while the validation RMSE was done on the full data from 2 pumping stations. The validation RMSE when the network got stuck was 0.01169. Max-norm was then tested using different constants, the results can be seen in Table 5.6.
Table 5.6. Result for the max-norm test.

<table>
<thead>
<tr>
<th>max-norm constant</th>
<th>Min validation RMSE</th>
<th>n-epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.01084</td>
<td>16</td>
</tr>
<tr>
<td>3.5</td>
<td>0.01092</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>0.01092</td>
<td>6</td>
</tr>
</tbody>
</table>

Each constant used for the max-norm test was run for 100 epochs and the result from the best epoch is displayed.

$L1$ regularization was tested on the 10 neuron network with the different parameters settings: 0.005, 0.01, 0.05, and 0.1, these where all tested for 40 epochs and all of them was unable to get out of only predicting no alarm. This means that $L1$ regularization was bad for this network. Similarly $L2$ regularization was tested with the parameter values 0.005, 0.01, 0.05, and 0.1, was similarly unable to exit the local minimum of predicting no alarms.

The best network out of those tested was thus single layered LSTM with 10 neurons, with dropout $p=0.2$, and max-norm constant equal to 3.

5.2.3 Further analysis and changes to the network

When looking further into the performance of this network it can be seen that it is very close to predicting no alarms at all, meaning it has a very poor performance. Upon closer inspection it can be seen that it only predicted the alarm high level and it only managed to correctly do so when an alarm had already started, meaning it was late and it has hardly any practical value. An attempt to improve the network was done by removing the time shift from the alarm data and reducing the length of the training sequences so that it was closer to 50/50 in alarm tags in the training sequences. By doing this the network will become a classifier network instead, leaving all the prediction capabilities to the trend network, and by increasing the ratio of alarms in the training data the network will start to give more alarms. Both false alarms and hopefully true alarms, thus the validation RMSE is expected to rise but the practical use of the network will also hopefully rise.

The time-prior was changed to 36 (3 hours) and the time-after was changed to 12(1 hour) which made it so that in 49% of the sample points there were an alarm.

Since we already know that this network will produce false alarms we can put higher priority on the training error being low while still not overfitting instead of taking the lowest validation error since a low validation error can be because it is mostly predicting no alarms.

The best network was chosen as the minimum of the validation and training RMSE added together which gave the best network after 77 epochs with a training RMSE of 0.1539 and Validation RMSE of 0.0170585. This network was much better at classifying an alarm when it actually happened, it was however still only able to classify the alarm high level. It did produce more false alarms, but when looked at further the false alarms it could be seen that the water level was above its alarm level and should have been marked as an alarm. From this it could be seen that the validations sets alarm had been wrongly marked and the parts which was false alarm actually was a true alarm. These parts only happened prior to 2016 where there were no alarm tags in any of my data.
I also noticed that for some longer regions where there was an alarm, according to the
water level, only part of it had been marked as an alarm.

This shows that the algorithm is doing what it is supposed to on the correctly tagged
validation set and that the quality of data is poor. The next step now is to increase the
amount of training data and hope that the majority of it is correctly marked so that
the network will not get stuck in a local minimum again of not predicting any alarms.
By increasing the amount of training data the results should be better, if it is correctly
marked.

5.3 Final test

For the final test the amount of training data was increased and the best networks from
the previous result were used. The training data now contained 56 pumping stations,
the validation set 7 pumping stations, and the test set 20 pumping stations.

The alarm network immediately got stuck in the local minimum where it did not predict
any alarms at all. In an attempt to make it predict the alarm high level the alarm was
relabeled according to section 4.2.2. After the relabeling the alarm network was still
unable to get out of the local minimum.

When the relabeling was done it could be seen that only 11.8% of the data that was
marked as the alarm high level was above the correspondingly set alarm level. There
are two reasons for this, the first being how Xylem has marked up the start and stop
time of the alarms. When an alarm triggers its start time is set and an operator is
called, either by e-mail, or SMS. When the operator has seen the call he can either log
in or go to the site and when he has fixed the alarm or deemed that there are no longer
an alarm, the stop time is set. This creates a delay between when the alarm parameter
says it stopped and when the stop time was set. If this happens often and with a large
delay, the algorithm will have a very hard time to distinguish what parameter values
belong to a certain alarm.

The other reason the algorithm was unable to correctly identify alarms, was because of
the down sampling of the data. When an alarm starts it does not often start at exactly
a 5 min interval which makes the alarm parameter data mix with non-alarm parameter
data. If an alarm starts at 20:23(hh:mm) it will be moved to 20:25 but the parameter
values there are an average of the values between 20:20 and 20:25 which makes it so that
out of the five minute average, only 2 minutes contained alarm levels of the parameters.

This problem reoccurs when the alarms are too short compared to the resolution. If an
alarm is active for 1 minute but the resolution is 5 minutes, 80% of the corresponding
parameters it is saved to will be averaged to levels which does not correspond to alarm
levels.

It is worth to note that by reducing the amount of data prior to an alarm used during
training it is possible that it became harder for the network to identify the switch alarms.
But considering that it had troubles to identify the easiest alarm high level, which is a
threshold alarm, increasing the time prior would probably not yield a better result.

The trend network managed to reduce its RMSE on the validation set by 5 times and
13 times on the training set. This shows that the increase of training data improved
the result of the trend network. The validation RMSE went down to 0.00139 and the training RMSE went down to 0.00074. However since the alarm network did not work, further analysis of the performance of the trend network will not be done since it will not increase the probability to predict an alarm.
Chapter 6

Conclusions

This chapter compares the results from Chapter 5 with the goals defined in Chapter 1, talks about future work concerning the work conducted in this thesis project, and reflects on issues relevant to this thesis project.

6.1 Comparison of the results with the goals

- Is it possible to predict alarms with the available data? - No
- How long in advance can the alarms be predicted? - not applicable
- How reliable is the predicted alarm? - not applicable

6.2 Reflections on issues relevant to this thesis project

One of the hardest tasks in this thesis was to understand and get the relevant data from the available SCADA data. About 2 months was spent on extracting, understanding and formatting the data. Information about the data would come up in all stages of the thesis. One which was discovered late and had a major impact on the results was how the alarms start and stop times were set, where the stop time was set manually. From Xylem’s perspective all of the alarms where correctly tagged which made it harder to find this property.

With the chosen solution for predicting alarms the data quality was too low. The two factors which played the biggest part in the low data quality was the downsampling of trend data and how the stop times for alarms were set, more about downsampling and how it affects the data can be read in Section 2.2.6. These two factors together made it impossible for the neural network to properly identify an alarm, and if the algorithm can not identify what an alarm is, it can not predict it either.

When the alarm network was run on the first validation set which contained two pumping stations, it could be seen that many of the errors the alarm network got were not an actual error, it was the validation dataset which had missing alarm labels. This made it clear that a tool for quality checking the data was necessary, otherwise the wrong conclusions could be drawn based on poor data quality.
The trend network was trained to a low RMSE value but I could not properly evaluate how well this network would work for predicting together with the alarm network, since the alarm network did not work. As such I was unable to properly evaluate how well a neural network with LSTM neurons would work for this task.

6.2.1 Note about alarm network configuration

It was seen when the thesis was finished that a mistake had been made in the configuration of the alarm network, the addition of max-norm did not improve the network but was used anyway. This mistake is thought to come from a sloppy comparison and its influence on the end result is deemed to be minor or non-existence, the affected tests will thus not be remade.

6.3 Future work

One thing that has become important in this thesis is the need to quality check data. When doing this for smaller amounts of data it is fairly easy to do it manually but as the amount of data grows this gets increasingly harder. For this purpose a tool needs to be developed which can quickly determine the quality and reliability of data. Since a dataset can be used for different purposes the metric the tool uses to evaluate the data needs to be changeable. If this tool had been available before my thesis the conclusion that it would not work because of the quality of the data could have been drawn a lot earlier.

In order for the SCADA system from country A to benefit from machine learning using the approach presented in this thesis the quality of their data needs to be measured, and improved. Doing this would increase the application areas and reduce the time it takes to start analyzing the data. The main points that need to be considered or improved are:

- **Normalization** - making data comparable between pumping stations, customers, and SCADA systems.
- **Naming ambiguity** - set clear standards for what labels means and applying them to all their systems.
- **Downsampled data** - the downsampling needs to be remodeled, removed, or improved so that it allows advanced analytics, otherwise data from several years will be saved which can not be used.
- **Separate channel for labeling events according to the parameters which caused them** - if an events happens for x seconds the start and stop-time needs to be set according to the parameters which caused the event.

When the two major problems have been solved with the data quality, a new attempt into predicting alarms should be made. The recommended approach would be to first construct a network that can identify what an alarm is based on trend data, in other words ensure that the alarms are separable from normal running, and then construct a
network which predicts trend data. Combining these two will then create a model which would have a higher chance of predicting alarms.
References


