TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Sergei Lukauškin 194061IADB

Network Operations Center Monitoring Automation using Machine Learning on the Example of Elisa Eesti AS

Bachelor's thesis

Supervisors: Hayretdin Bahsi
PhD
Valdo Kiks
Bachelor

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Sergei Lukauškin 194061IADB

Võrguoperatsioonide keskuse monitooringu automatiseerimine masinõppe abil Elisa Eesti AS-i näitel

Bakalaureusetöö

Juhendajad:	Hayretdin Bahsi
	Doktorikraad
	Valdo Kiks
	Bakalaurusekraad

Tallinn 2022

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Sergei Lukauškin

16.05.2022

Abstract

We live in a time when every second person has a portable device that has access to the Internet and a mobile network. In order to provide users with access to it, data centres are required to process incredible streams of user data every second. Each error resulting from the processing of requests can be very costly for the telecommunications service provider. In order to avoid such problems and quickly locate existing ones, network operations centres are used to facilitate data management. These centres always have a monitoring system to track bugs and create tickets for problem cases. Recently, for a faster response to new flaws in the system, tools have appeared in the world to automate the error handling process. Such solutions helped filter out unnecessary or fake errors or automatically correct the rest. However, as in any system, this solution has several flaws. The problem lies in the time of recognition of errors and the quality of such recognition. In order to optimize the process of finding errors, this thesis considers the possibility and effectiveness of using machine learning methods as a means of recognizing the validity of errors obtained from such monitoring systems. This will minimize the time required for the system to validate errors and increase the accuracy of these predictions.

This thesis is written in English and is 42 pages long, including 6 chapters, 8 figures and 2 tables.

Annotatsioon

Elame ajal, mil igal teisel inimesel on kaasaskantav seade, millel on juurdepääs Internetile ja mobiilivõrgule. Selleks, et anda kasutajatele sellele juurdepääs, peavad andmekeskused töötlema iga sekund uskumatuid kasutajaandmete vooge. Iga päringute töötlemisest tulenev viga võib telekommunikatsiooniteenuse pakkujale olla väga kulukas. Selliste probleemide vältimiseks ja olemasolevate kiireks leidmiseks kasutatakse andmehalduse hõlbustamiseks võrguoperatsioonide keskusi. Nendel keskustel on alati seiresüsteem vigade jälgimiseks ja probleemsete juhtumite jaoks piletite loomiseks. Hiljuti on maailmas uutele süsteemivigadele kiiremaks reageerimiseks ilmunud tööriistad veakäsitluse automatiseerimiseks. Sellised lahendused aitasid välja filtreerida mittevajalikud või võltsvead või ülejäänu automaatselt parandada. Kuid nagu igal süsteemil, on ka sellel lahendusel mitmeid puudusi. Probleem seisneb vigade tuvastamise ajas ja sellise tuvastamise kvaliteedis. Vigade leidmise protsessi optimeerimiseks käsitletakse käesolevas lõputöös masinõppemeetodite kasutamise võimalust ja efektiivsust sellistest seiresüsteemidest saadud vigade õigsuse tuvastamise vahendina. See vähendab süsteemil vigade kinnitamiseks kuluvat aega ja suurendab nende prognooside täpsust.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 42 leheküljel, 6 peatükki, 8 joonist, 2 tabelit.

List of abbreviations and terms

AI	Artificial Intelligence
API	Application Programming Interface
CLI	Command-Line Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSP	Communications Service Provider
FD	Flapping Detection
GPU	Graphics Processing Unit
HMM	Hidden Markov Model
HTTP	Hypertext Transfer Protocol
LSTM	Long-Short Term Memory
ML	Machine Learning
MLP	Multilayer Perceptron
NN	Neural Network
NOC	Network Operations Center
OPEX	Operating Expense
PD	Perpetual Detection
PReLU	Parametric Rectified Linear Unit
QA	Quality Assurance
QoS	Quality of Service
ReLU	Rectified Linear Unit
REST	REpresentational State Transfer
RNN	Recurrent Neural Network
RPC	Remote Procedure Call
SIMD	Single Instruction, Multiple Data
SIT	System Integration Testing
SNMP	Simple Network Management Protocol
UAT	User Acceptance Testing

Table of contents

List of figures

List of tables

1 Introduction

The monitoring industry is an essential part of any IT system. Especially if we are talking about data centres or network operation centres with an enormous customer base and the need to maintain the QoS at a high enough level [1]. With the increase in the scale of the enterprise, the number of logs collected from the servers also grows. Therefore, it becomes more and more challenging to maintain even with an increased number of employees. Reaching the scale of big tech makes enterprises use special utilities to collect and organize information about managed devices or IP networks and modify that information to change device behaviour. One of such utilities is the Simple Network Management Protocol (SNMP), widely used for network monitoring and simplifies the network elements configuration process. Simple Network Management Protocol technology is well documented and has been the research topic many times, though pure protocol cannot handle errors. Error handling is made possible by applying third-party services to a monitoring system. There are plenty of ready SNMP-based monitoring solutions available, and many of them support error handling. One of those solutions is NOC Project, which provides Root Cause Analysis, topology correlation, escalation, active probing and passive alarm condition detection in Syslog and SNMP traps [2]. Unfortunately, the more complex cases are frequently not handled properly and are ignored instead of notifying the support team.

The main goal of the current thesis is to analyze network operation centre systems which use different monitoring solutions, such as the NOC Project. Existing algorithms work by strict rules and spend too much time waiting for needed events to arrive. Events could be predicted by using machine learning to prevent such behaviour. Network operation centres may have millions of recorded events or traps per minute. The fault management and handling systems play a significant role in a system of this size. Dozens of products include customizable solutions that automate resolving issues and creating tickets for system administrators, but only a few can handle more complex situations like false positive and false negative alarms. False alarms are pretty standard in existing error handling systems, and on some occasions, one can account for up to half of all cases.

Furthermore, monitoring services may send multiple simultaneous events that include both the error and fixed events, which may confuse many existing algorithms.

The latest algorithms include the pre-processing method of resolving toggling behaviour and dealing with such circumstances. The effectiveness of methods is questionable, but not as much as the speed and the algorithm behind them. The solution to the problem is to include machine learning in the error handling pre-processing step to increase the speed and efficiency of the algorithm and reduce the production cost of the system. The solution would analyze the received trap batch and resolve false alarms, including edge cases. The prediction may not give such high accuracy as the algorithm solution, but it can be close enough to handle errors properly and sometimes even be more beneficial than the old algorithm. Depending on alarm type, the time taken to pre-process step may exceed hours or even days, so this aspect is critical to testing and production environments.

The data flow in the monitoring service begins with sending events from the server to services subscribed for processing and parsing for further processing. Once an event has been converted into a structure suitable for further processing, it can be sent to a service responsible for managing and creating tickets. If the algorithm decides the issue referenced by received alarms is valid, the ticket is created. Otherwise, the alarm is rejected, and a ticket is not created.

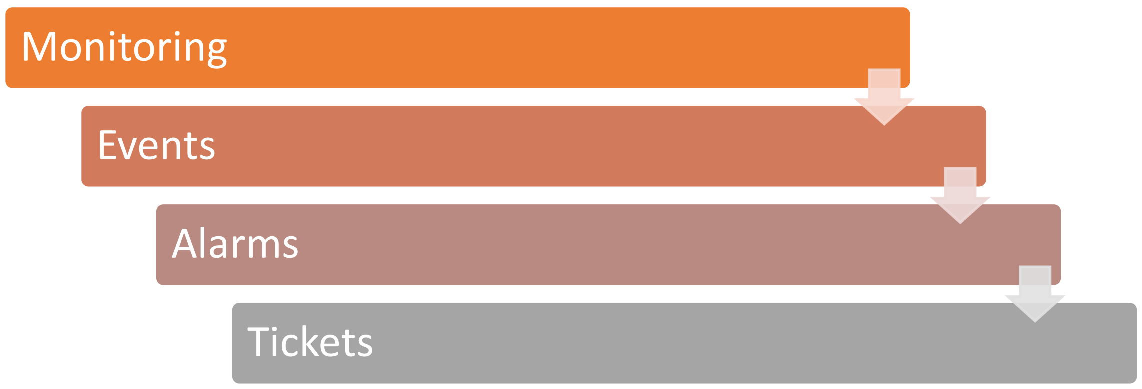

Figure 1. Data flow of monitoring service

2 Alarm Pre-Processing

The pre-processing step is used to manage events received from the server correctly. This step helps to filter out alarms before they reach further steps. The exemplary implementation of the pre-processing module can handle Flapping Detection (FD) and Perpetual Detection (PD) cases for relevant algorithms. One of the cases should always retrieve false or invalid alarms on high-level triggering. Nevertheless, there are many edge cases in the real world, and some false alarms come through from time to time. Such edge cases can occur due to high latency or adverse weather conditions. Behaviour like that could be prevented by a more significant alarm query or changes in module configuration. However, this dramatically affects the speed of processing and cannot always guarantee the correctness of the result due to the time frame set in the configuration.

2.1 Perpetual & Flapping Detection

Most networks have many temporary issues that trigger alarms but are cleared without creating a ticket. While flapping Detection only checks for the second received alarm being positive, the Perpetual Detection helps resolve such occasions by providing a query of recent alarms with the same identifier. The query has the two most critical settings in the current module: a perpetual wait time and a query size. Query size refers to the number of positive alarms that should be waited for until further steps can be taken.

Perpetual wait represents the time till an unfilled query may be considered cleared. Thus, the ticket is not created. Optimizing the algorithm could be done by applying other sources of information, for example, the current day of the week or weather conditions. The problem is that it might take a long time for an endless wait configuration until the algorithm moves on. Increasing the query size improves the prediction rates but slower the whole process. Lowering the wait time benefits alarm processing and average resolution time but decreases the certainty of predictions. Many possible configurations exist, but a simple algorithm has its stop point, where statistics will not raise more.

Flapping Detection means the system produces multiple alarms with different states in a short time range. This is typical behaviour due to bugs, configuration errors, and system failures. The operation algorithm of this service is slightly different from Perpetual Detection, but there are similarities, and it will be possible to unify in the prediction process.

3 Machine Learning

Machine learning is a broad field with many possibilities in statistical analysis and predictions based on data. Flapping Detection is a straightforward step that only reacts too quickly by switching alarm states, and its optimization will not be so tangible if not harmful. The production environment logs always include results of the ticket created by the algorithm and the process of declining them or, in other words, marking them as false. Example input-output pairs provided lead to supervised learning. The supervised learning methods with prediction include classification and regression, but as there is a need for resolving more than one type of alarm, the classification algorithm is the best choice [3]. The algorithm used for a supervised learning classification occasion is a neural network since dealing with the processing of complex structures with lots of categorical data [4].

There are two groups the network model needs to classify. The first one is valid alarms, which should pass the pre-processing step and create a ticket depending on the further stages. The second group consists of invalid alarms, and the tickets must not be created for those as they will be filtered out during the pre-processing step. If the alarm was classified as invalid, it is named "false" and processed separately.

Input node must be represented by a nested variable-length list of received alarms of the same identifier. The equivalent data structure is a ragged tensor in the machine learning field. Inner layers will consist of LSTM (Long Short-Term Memory) model and a few optimization layers [5]. Outputs include four classified label nodes.

Data features might vary from the network monitoring configuration and setup. However, the features must not heavily depend on one particular configuration like SMTP- or gRPC-based monitoring [6, 7].

A unified set of features includes:

- An alarm identifier is an alpha-numeric categorical value that is based on reference.
- Occurrence time – numerical timestamp – the time alarm occurred.
- Location, site, node – categorical – features show where the alarm belongs.

- Equipment – categorical – the equipment used on a machine that broke. Alarms will be grouped by location and alarm id and processed as an input.

3.1 Dataset

Dataset can be viewed as a collection of data objects, also called records, points, vectors, patterns, events, cases, samples, observations, or entities. Data objects are described by several *features* that capture the fundamental characteristic of an object [8]. Data collection is always a trouble for any dataset. Using programmatically generated data can only train the model to follow the strict rules the data is generated with. To be more precise, collected data should be gathered from a production environment. Some occasions could be generated algorithmically, though the model trained with such data might struggle with real-world situations. For the testing and prototype model training programmatically, generated data would suit fine, but further development and training are better with data collected from a real network. The inner neural network structure will quickly swap the dataset if needed. Most monitoring services can produce logs. The logs received from the service can be parsed and processed with scripts to produce data for training. For security purposes, the received data should be anonymized. The data for the dataset should also be categorized, e.g., the string typed values must be categorized and exchanged for numerical references.

Data that could be gained from the monitoring service has two most common sources [1]:

- Predicting monitoring frequency sampling: in this case, the optimum monitoring time interval is calculated as the first monitoring time interval at which the difference between the current and previous measurement metrics values are below a certain threshold.

- Predicting monitoring data: An optimum monitoring data can be obtained by investigating the metrics values at specified frequency sampling. Intuitively, measurement metrics which does not change frequently are not necessary to monitor frequently.

3.1.1 Dataset Processing

Any supervised machine learning for neural networks needs labelling. The dataset labelling is the machine learning process to identify the raw data that also allows labelling the informative data and meaningful data to provide context to it, and machine learning can use that data to learn from it. Data labelling is a critical concept due to adding context to data before using that in the training model. Furthermore, the data labelling helps select a correct approach when it is crucial to improve the scalability and quantity factors. The dataset labelling has many approaches, which can be done by using a combination of several methods. For example, the amount of production data might be too significant to label manually. Active learning prioritizes the data labelled to have the highest impact on training a supervised model. Active learning can be used in situations where the amount of data is too large to be labelled, and some priority needs to be made to label the data smartly. For example, the amount of data gathered from the production environment will not count millions of alarms, though manually labelling each data would take too long. However, due to its use with the LSTM model, it is not advisable to use active learning [9]. Instead, the logs parsing utility will automatically compare the alarm states with the resulting state and produce a separate labelling file for each unique issue.

The pre-processing step is done before providing the dataset for training. This step includes filtering out all the invalid or corrupted data and checking the validity of data, for example, timestamps or coordinate values [10]. This is needed to make sure the model will only retrieve correct data, which will raise the prediction rate.

Before proceeding with the pre-processing step, the dataset should also be filled with corrective data, which is the data, for improving the recognition of edge cases. The corrective data might include any other data that correlates with the initial data. For example, the weather data or the server rack status at the time alarm occurred. The randomization technique could reduce the prediction loss and increase the accuracy for the edge cases. The numeric values like timestamps, longitude or latitude could be modified using randomization to refine the model edge case handling. For example, a timestamp modified by a few hundred milliseconds might reduce the time correlation for the model and pay attention to other features. Randomization improves the prediction accuracy and helps recognize the relation between the alarm’s timestamps.

Dataset size influences the prediction rate a lot. Various methods could increase the number of items in the dataset. As the input data is represented by an array of alarms with the same alarm identifiers, the dataset could be populated by duplicates. For example, an array which includes four alarms will be duplicated without one or several last alarms, leaving it with only part of the initial number of alarms. Such duplication might increase the size of the dataset by a considerable amount and will increase the prediction rate a lot. The duplicated data will be handled separately and not related to the initial array. The unique alarm identifier remains the same to simulate a real-world situation where once the alarm arrives, it is added to earlier received alarms and sent to the model. The issue case would not be closed until the model can give over 80% of prediction certainty.

After initial training, the validation stage is needed for testing the trained model. A training set is implemented in a dataset to build a model, while a test (or validation) set validates the built model. Data points in the training set are excluded from the test set. Usually, a dataset is divided into a training set and a validation set. The dataset will be split into 70% and 30% for training and testing datasets [10].

3.2 Model Structure & Optimization

3.2.1 Input Layer

To increase the model accuracy, it must go through an optimization process. With an enormous dataset, the training time will increase respectively. Training the model in the production workflow may affect the development release cycle. The first layer in the neural network model is the input layer, a vector of alarms with variable sizes to support different alarms. Unstable layers with uncertain tensor dimensions are often used for text recognition, but it suits the current case well. The vector inner structure is a fixed size with pre-defined features, and the only variable size is the length of the vector.

3.2.2 Inner Layers

The native representation of variable length vector creates a tensor with the maximum size the system could work with and matches other structures to this size, filling undefined spots with zeroes. This could be achieved by using an additional embedding layer. It will keep padding for the data to ensure that all sequences are the same length. The padding step affects the model’s performance and should not be set to a high value. The padding change has an exponential correlation to performance and training time. The loss decreases much slower if the padding is too high. The slight padding values will decrease the maximum length of the alarms vector the model could handle.

The inner part of the layers is an LSTM model, which is an extension of recurrent neural networks (RNNs) [9]. The idea of RNN is to consider the previous output and store it in the memory for a short period. With LSTM, there is no need to keep a finite number of states from beforehand as required in the Hidden Markov model (HMM), and it helps to solve an issue with long term storage. Furthermore, LSTM helps resolve the connection between the alarms with the *O(1)* weight complexity, increasing the loss minimization and gradient calculation speed.

3.2.3 Activation Layers

Activation functions are applied to the weighted sum of inputs in the hidden layer(s) and the output layer. Inputs can be raw data or the output of a previous layer. Most of the activation functions are non-linear. Linear functions are mainly used for regression problems and would not suit to current use case. A linear function takes the input z and returns the output, cz, a multiplication of the constant c and the input. In mathematical form, this would be written as $f(z) = cz$ — no input changes and is represented by a single straight line on the plot. Any function that is not linear can be classified as a non-linear function. The graph of a non-linear function is not a single straight line. It can be a complex pattern or multiple linear components [11]. The leading contenders for activation functions are Sigmoid and some variations of ReLU.

The sigmoid activation function (logistic function) is a non-linear function which is primarily used in logistic regression models and has an s-shaped graph. The sigmoid function converts its input into a probability between 0 and 1. It is common to see this function being used for hidden layers in MLPs, CNNs and RNNs. The leading contenders for activation functions are Sigmoid, Binary Step and some variations of ReLU. The rest of the functions are too computationally expensive or are mainly used for other neural network architectures.

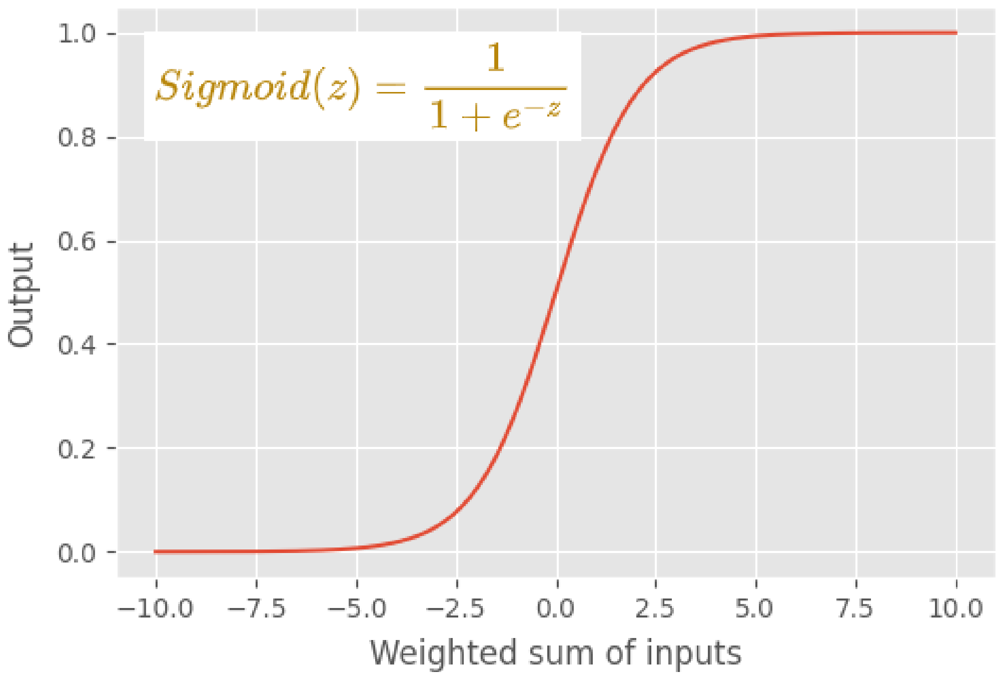

Figure 2. Sigmoid activation function [11]

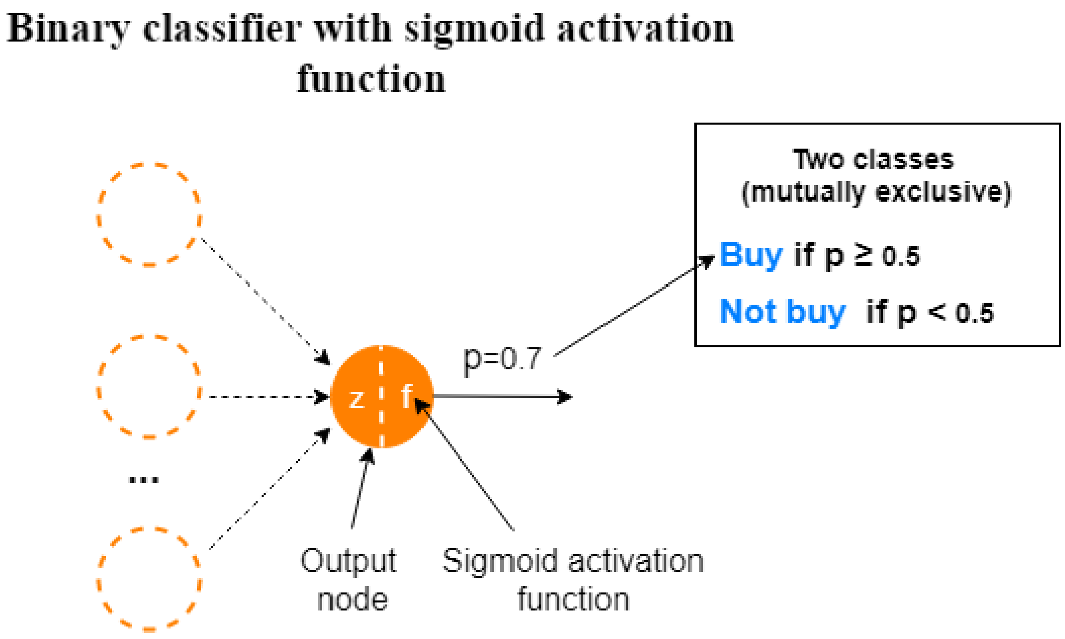
Binary classifier with sigmoid activation function
Two classes (mutually exclusive)
Buy if p ≥ 0.5
Not buy if p < 0.5
p=0.7
z
f
...
Output node
Sigmoid activation function

Figure 3. Example: binary classification with a sigmoid function [11]

Another activation function is ReLU which is an acronym for Rectified Linear Unit. ReLU resolves the vanishing gradient problem, which is an issue for the sigmoid function. The function is also computationally inexpensive and up to 6 times faster than the sigmoid or the tanh functions. In addition, the ReLU convergence is faster than sigmoid and tanh functions because of the fixed derivate for one linear component and a zero derivative for the other linear component; therefore, the learning process is much faster. The only issues with the ReLU function are a dying ReLU problem and the computational issues due to enormously big function output.

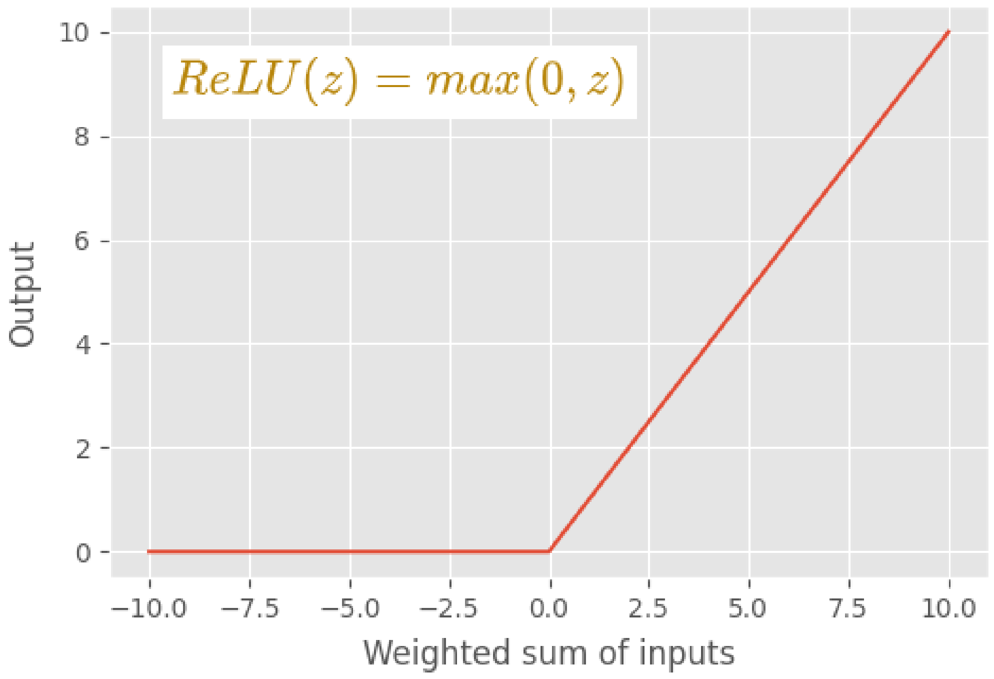

Figure 4. ReLU activation function [11]

Many ReLU functions like leaky ReLU, PReLU (Parametric ReLU) or ReLU6 solve dying ReLU problems and optimize the learning process. For example, the leaky ReLU differs from the original ReLU by not using any linear component with zero derivatives. Excluding the zero derivatives also increases the computational performance.

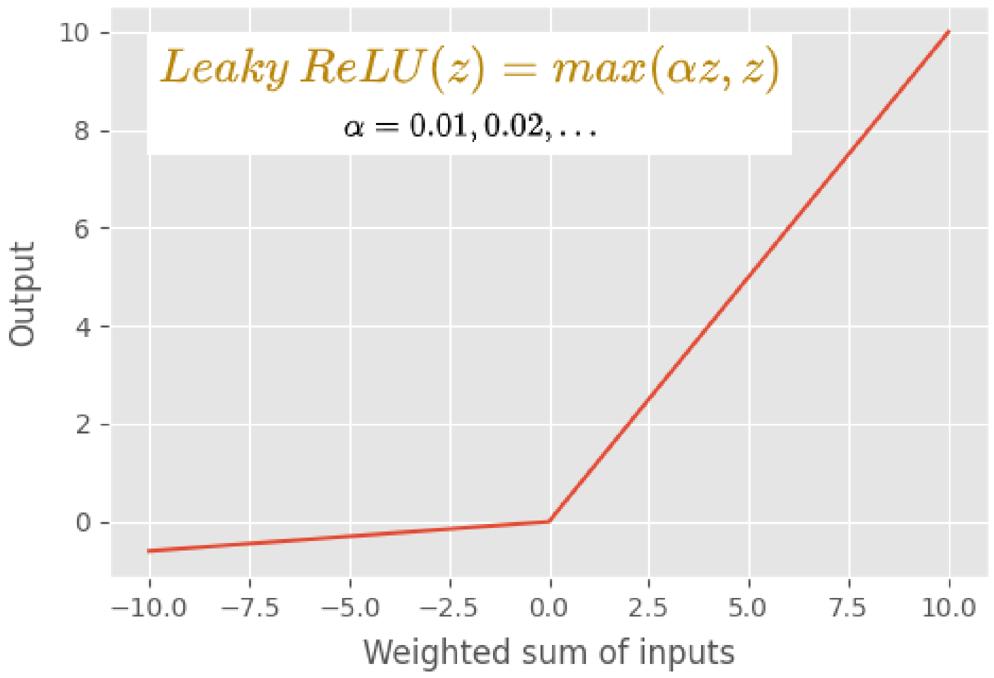

Figure 5. Leaky ReLU activation function [11]

As the most effective activation function for the binary classification, the leaky ReLU suits perfectly. The leaky ReLU solves the vanishing gradient problem and the dying ReLU issue. Furthermore, the increased learning performance of the function optimizes the training process, so the ReLU and its derivatives are chosen for further research.

3.2.4 Optimization

Every neural network needs a loss function to calculate the loss of each iteration. Binary cross-entropy is a commonly used loss function for binary classification problems. It is intended to use only two categories, either 0 or 1. It is a loss function that is utilized in binary classification tasks. A theoretically perfect model has a binary cross-entropy loss of 0.

Optimization is a mathematical discipline that determines the most effective solution in a well-defined sense. Optimization algorithms in machine learning aim at minimizing an objective function, which is intuitively the difference between the predicted data and the expected values field [12]. The most common optimization problem encountered in machine learning is continuous function optimization, where the input arguments to the function are real-valued numeric values, e.g., floating-point values. The output from the function is also a real-valued evaluation of the input values. Many different optimization algorithms can be used for continuous function optimization problems.

RMSprop is one of the fastest and the most efficient first-order optimization algorithms for the neural network, but it does not work well with large datasets and is outperformed by other algorithms. Adam is another algorithm which includes momentum and bias correction.

The graphs below represent the comparison of the RMSProp, Adam and Adamax algorithms. The dataset used comprises 60 000 handwritten training examples and 10 000 testing examples. The RMSProp and Adam are close, but RMSProp produces slightly better results than Adam [12]. Depending on the dataset, the results may vary, but it is decided to use both algorithms and compare the results for the thesis.

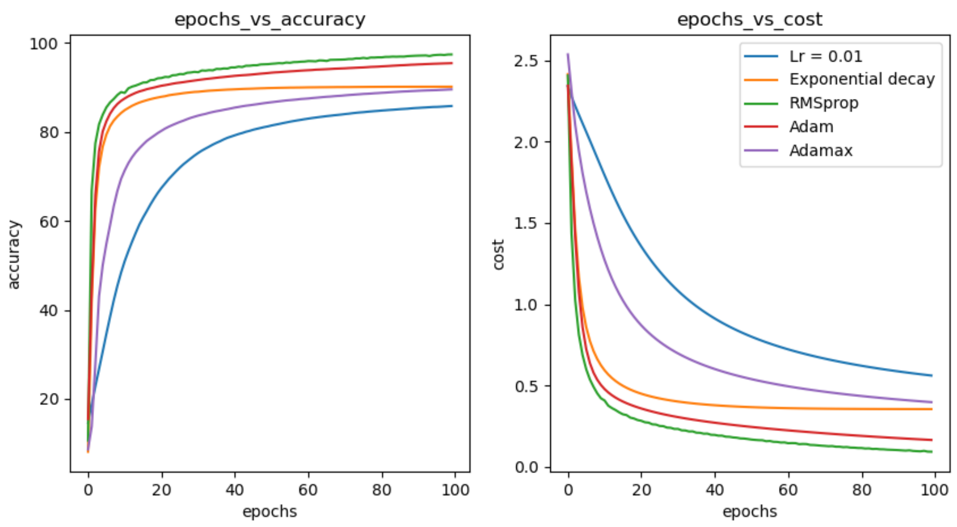

Figure 6: RMSProp, Adam and Adamax optimization algorithms [12].

3.3 Related Work

Many scientific papers by famous universities wrote about machine learning used for network automation and the applications the machine learning could be used for. In the research paper written by Danish Rafique and Luis Velasco, unsupervised learning was described in practice as “neither easily accessible nor abundantly available” [4]. The unsupervised learning technique was criticized for usage in the network section, though many supervised learning solutions were found. The reinforcement learning algorithms could be used, but it mainly handles the situations where the relationship between the data is needed to be resolved then the general input-output labelling. On the other hand, supervised learning could be used in many different fields in networking. The hardest part about applying machine learning is the data aspect. The fundamental aspect of building an ML model is to separate the available data set into a training, validation, and test sets. The reason to divide data into these sets is to avoid overfitting the training data.

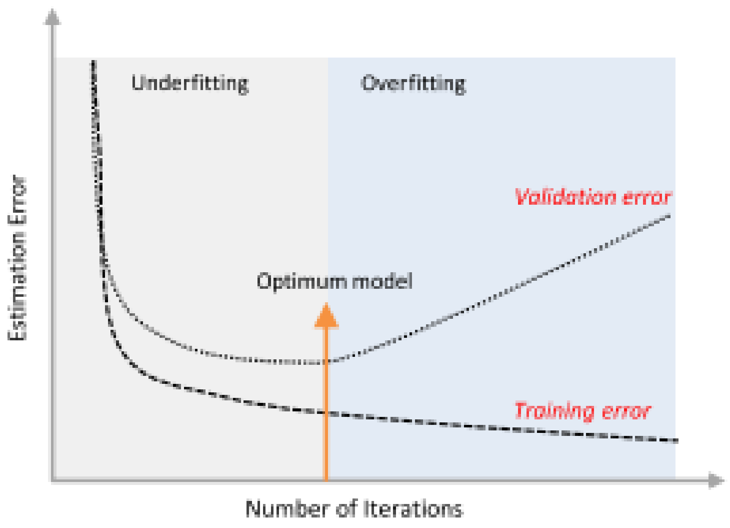

Figure 7. Underfitting versus overfitting from the research of Danish Rafique and Luis Velasco [4]

It can be seen from the graph above (Fig. 7) that as the model approaches convergence—underfitting, which refers to an overly simplistic model—the training and validation data sets show similar results; however, beyond this phase, training errors continue to improve, whereas validation errors start deteriorating—overfitting, which refers to an unnecessarily complex model. Therefore, the model that enables the best performance for the validation set is selected as the optimum model. Furthermore, various cross-validation approaches may split the training and validation set for model construction based on data characteristics. Finally, the "Predictive Maintenance" section describes how machine learning applies to fault management and maintenance. For example, early detection of equipment failure states and consequent remedial actions can prevent network downtime and enable scheduled preventive maintenance.

The part of the Network Operations Center could include Cloud Computing, which is rapidly becoming an accepted computing paradigm [1]. However, the unsupervised and reinforcement machine learning paradigms are not suited for predicting the alarms received from monitoring. The Elisa Virtual NOC integrates with many monitoring solutions. One of such solutions is Nagios, which lacks the database to be named a proper lightweight monitoring solution. It is also hard to configure, but the data produced by

Nagios is still easily processable and usable with the machine learning automation solution [1].

Many research papers describe the use cases for machine learning within the SNMP dataset for describing network anomalies [13, 7]. The research primarily aims to prevent basic attacks and vulnerabilities from being exploited on the SNMP server, though the approach they use is also capable of handling more complex cases like false alarms. Researchers concentrate on passing the network traffic data directly to the machine learning classifier to train the model while integrated into the service. Classification is one of the supervised machine learning techniques mainly applied to intrusion and error detection. In the thesis written by Ghazi Al-Naymat [13], only AdaboostM1, Random Forest and multi-layer perceptron algorithms are compared. Algorithm choice could be explained by the thesis dedicated to low-level intrusion detection and does not profoundly investigate fault management.

3.3.1 Fault Management

The methodology of applying machine learning to a fault management system is described in the research paper by Denise W. Gürer [14]. Despite being old, the information from this document is still relevant for the research. Denise states that automation of network management activities can benefit from Artificial Intelligence (AI). For example, the neural network model is integrated with a telecommunications synchronous optical network with asynchronous transfer mode. Alarm filtering can be thought of as four processes [15]:

- Compression – reduce occurrences of the same alarm.
- Count – substitution a specified number of occurrences of the same alarm into a single alarm.
- Suppression – filter out all low-priority alarms if high-priority alarms are present.
- Generalization – refers to an alarm by the superclass, which domain experts determine.

Currently, most AI-based fault diagnosis systems struggle with some limitations. One such is the inability to handle unforeseen situations due to either new alarms in the system or changed network topology. This requires a model to be trained each time on a new topology if the model is not suited to live learning and train all the time while running.

Live learning will surely increase the adaptation rate to the new environment without retraining the model. Artificial networks do not scale well and are not suited for large domains constantly changing their topology and evolving. The dataset for training must avoid fuzziness in it to avoid fuzzy rules from being formed, and the rudimental, uncorrelated, ambiguous, and incomplete data should be excluded. In order to maintain the model, the system must be well understood.

The research paper proposes a hybrid AI system that combines multiple algorithms matching each task rather than creating a single service to deal with all the tasks. The downside of using hybrid AI is the requirement of familiarity with all the methods and techniques. The process includes collecting alarms coming from the system and categorizing them to be managed by a specific algorithm.

The alarm correlation is a necessary step to be applied after the alarm filtering. The author suggests that using a feedforward neural network (NN) is effective in medical diagnosis [16], target tracking, and data compression systems. In addition, the neural network solves many problems of alarm correlation [15].

- Similar conditions or edge cases (Pattern matching)
- NN is flexible due to functions approximation
- Incomplete or uncertain data

The most necessary part of the research for the current thesis is the “Fault Identification” [14]. It shows how the alarms can be identified by comparing the Expert Systems and case-based reasoning (CBR). Unlike ES, the CBR does not require extensive maintenance and can handle new and changing data through the ability to use an analogy. CBR can also learn from experience gained from the analysis of new cases.

As shown in Figure 8, the CBR process consists of:

- Retrieval
- Interpretation and adaptation
- Evaluation and repair
- Implementation
- Evaluation and learning

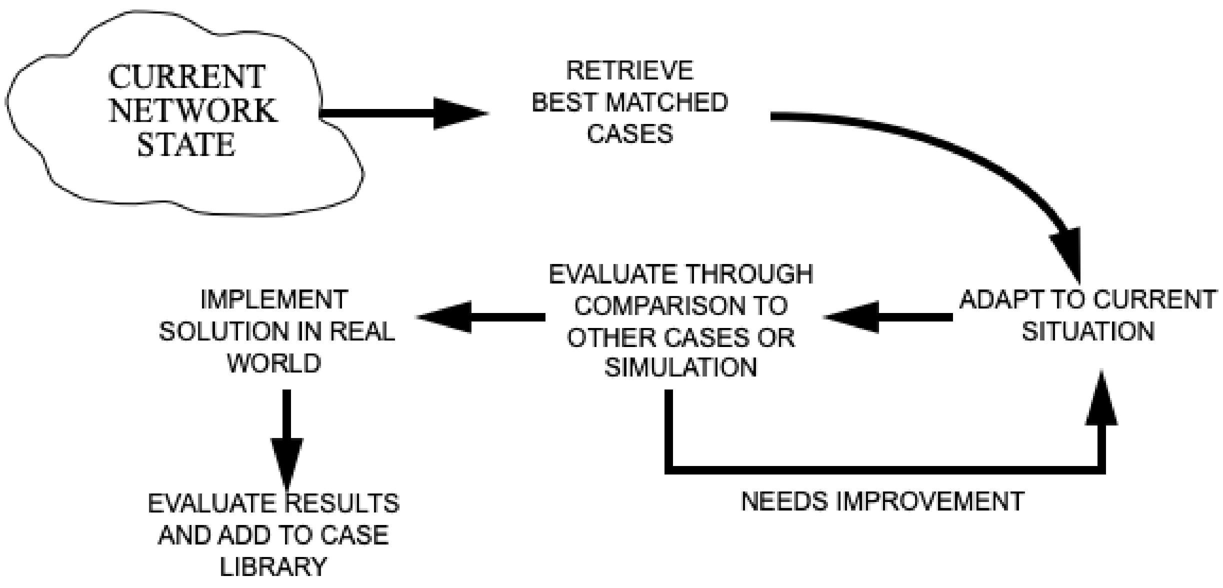
CURRENT NETWORK STATE
RETRIEVE BEST MATCHED CASES
ADAPT TO CURRENT SITUATION
EVALUATE THROUGH COMPARISON TO OTHER CASES OR SIMULATION
IMPLEMENT SOLUTION IN REAL WORLD
EVALUATE RESULTS AND ADD TO CASE LIBRARY
NEEDS IMPROVEMENT

Figure 8. The case-based reasoning process [14]

To be considered ideal, the system should automatically retrain itself when the new cases come and handle the unknown occasions without any issues. The research also covers fault correction, but this is out of the current thesis' scope. [16]

4 Solution

4.1 Local Hardware & Software

The selection of neural network hardware depends on many factors. Some of the points to be considered include whether the network will be memory or computation bound. In addition, the topology and architecture of the network are important deciding factors when choosing hardware for neural networks. Whatever the size of the network, some testing must be performed to understand the real-time needs of the neural network hardware implementation. It is also worth considering the need to work with confidential data, which implies a ban on storing non-anonymized data in unencrypted form. Finally, not only the hardware is needed for training, but for the production environment, once the trained model is executed as a service, it is necessary to choose a performant enough hardware.

Scalability and usability with microservices are other factors which affect the workflow. The services always depend on processing before sending a result to an end-user. The scalability resolves this issue by running multiple instances of the same application or service. Scalability could be an issue with neural networks because the computing power needed for model execution varies with the inner layers' architecture and hardware used. Running multiple service instances using a neural network model would not be cost-effective and will increase the OPEX (Operating Expense).

The Python programming language with TensorFlow [17] library with Keras support module was used for the training. Hardware part included personal computer with CUDA or OptiX compatible GPU. This would improve the training speed and not pass data further than the local network. In addition, a Python JIT compiler like Numba could speed up the training process. With Numba's help, it is possible to move low-level computations to compiled form. Compiled numerical algorithms in Python can approach the speeds of compiled programming language, which is a superior increase. Moreover, the multithreading, SIMD vectorization optimization and GPU acceleration will be more budget effective than the pure Python scripts.

4.2 Production System Description

The system to which this solution will be applied is a specific existing enterprise solution named Elisa Virtual Operations Center (vNOC). It uses microservice architecture written mainly in Kotlin programming language but includes many services written in Python, TypeScript and Golang programming languages. In addition, most services rely on the Spring Boot library and use it for state management.

The system is divided into SNMP enrichment, ticketing, and rule engine. The SNMP enrichment parses and extends the received alarms with the needed metadata. The ticketing manages the ticket creation process and sends it to the ticket dashboard or third-party services like Jira, where other employees can work on the received ticket. The rule engine is used precisely for the configured rule application to the alarms gathered from the SNMP enrichment service. The Kafka provides a foundation for the current data platform, including event-driven microservice architecture. The deployment is managed by Kubernetes clusters using “Argo CD” for continuous development and “GitHub Actions” as a continuous integration solution.

The problem described in the thesis is related to the rule engine service, which is a bridge between SNMP enrichment service and ticketing service. The service uses Final State Machine to manage the state of each separate alarm and keep maintainability on a high level. The Perpetual Detection and Flapping Detection stages are handled by a rule-based algorithm on a pre-processing step. The algorithm only includes perpetual wait time and the minimum query size settings. Once the minimum query size is achieved, the pre-processing step is completed. The alarm is considered perpetual and passed once the query size exceeds the required amount and the wait time surpasses the perpetual wait time.

The solution will be the sub-service for a rule engine service to replace the existing Flapping Detection and Perpetual Detection algorithms. The communication between the sub-service will be handled by Kafka messaging broker using the Python corresponding library. The trained model will be executed each time it receives an alarm and share the state with the rule engine service. The algorithm will handle the pre-processing step in the result.

4.3 Testing

Testing for the solution is divided into pre-deployment testing, System Integration Testing (SIT) and User Acceptance Testing (UAT) steps and the QA tester is responsible for the SIT and UAT testing. Pre-deployment testing includes writing unit tests, integration tests and end-to-end tests using the internal Python library. There are 11 unit tests, which ensure that the prediction for the common cases is high enough and the model activation function is not too strict. Integration tests check whether the communication between services works without issues using the service mocking technique.

Unit testing sometimes showed unpredictable results depending on the model prediction percentage. Due to this, the initial prediction threshold was introduced that would let results vary, not affecting the test efficiency. This change ensured tests were passed every time, even when the accuracy was on the edge of succeeding.

4.4 Existing Statistics

The statistic is collected from the data exported from the internal Grafana service and the public resources. The current statistic shows that the Communication Service Providers (CSPs) produce 8 million calls per second from over 150 connected systems. The Elisa monitoring solution helps CSPs increase QoS and enhance customer experience through preventive, accurate error detection and rapid problem resolution. [18] As of 1 February 2022, the percentage of successfully resolved issue cases collected from Elisa Polystar company's internal statistics monitoring solution was slightly over 90%. About 10% of cases are not handled properly and are created the ticket when not needed and vice versa. The average waiting time does not exceed two days, though some cases might require seven or more days until the system resolves the case validity. The goal of the service is to reduce the waiting time needed and increase the resolving rate. The service can also be tough to configure and requires much testing in the simulated environment before deployment to the production environment. Although vNOC usage can decrease the resolution time by 79% and the incidents count by 71%, it is still far from perfect [18].

4.5 Training Results

The programmatically generated dataset structure includes algorithmically generated identifiers and location values. The initial dataset included 10 000 alarms divided into 3 500 issues with unique issue identifiers. After the pre-processing step, the number of alarms did not change, but the total unique cases count grew to 18 000 due to randomization and duplication techniques. Next, the dataset was divided into training and validating datasets. 70% or 12 600 cases were used as a training dataset, and the left ones were used for testing.

Table 1. Dataset features used for training

Name	**Type**	**Comment**
alarm_id	Categorical	Alarm case identifier reflecting the error type
occurrence_time	Numerical	The timestamp of alarm occurrence
location	Categorical	Location identifier
site	Categorical	The network centre identifier
node	Categorical	The node identifier
equipment	Categorical	A set of equipment used

For the first attempt, a 100 epoch was trained. The optimizer used was Adam, and for the loss computing algorithm, the binary cross-entropy was chosen. The embedding layer padding value was set to 10 as the maximum number of alarms for one case did not exceed ten alarms in the training dataset. The loss reduced rapidly and then stopped at 0.4752, and the prediction rate was around 79.21%. Once the padding was increased up to 15, the loss raised and became 0.5158 and the prediction rate reduced to 74.92%. Adding a hidden layer before the LSTM layers helped increase the accuracy to 83% and reduce the loss to 0.41.

Setting the epoch value to 10 000 and reducing the learning rate helped achieve the loss of 0.2412 and the accuracy of 88%, and then the data overfitting occurred. However, adding more layers to the model increased the training time and execution time by 2. The additional data like temperature did not solve the issue, but the additional feature of alarm receiving time improved the prediction rate to almost 90%.

The better prediction rates could be achieved with additional layers added to the model, which will increase the training and execution time by a lot. Adding features and improving the pre-processing step could raise the model accuracy without significant time costs. The best result was achieved using two additional layers between the embedding and LSTM layers using the weather and the alarm receiving time. Epochs count was 100 000, and the learning rate was set to 0.000001. This setup gave a loss of 0.21 and an accuracy of 91%.

4.6 Service Architecture

To properly integrate the solution into the production environment, it is necessary to create a Kubernetes deployment script, and the Helm chart as the existing system uses it to manage the cluster versioning. The Python programming language was used to write a prototype version of the service. Python is the interpreted language with easy syntax, and writing in it can produce relevant results in a short period.

The application integrates with other monitoring solutions and provides REST API to communicate with other services. FastAPI Python library was used, which provides an HTTP request multiplexer with the data typing. Another benefit is that it is faster than most competitors like Flask, as it suggests a fully asynchronous backend with support for multithreading. The dependencies and packaging are managed by Poetry, which provides an easy-to-use CLI interface to set up the project and build it to a package. This decision can also handle different Python version usage. The default Python library was used for logging, but the logging integration using API service is also available with some configuration.

The application does not have any internal data storage included. Instead, it relies upon the external Kafka storage for storing the data. However, if there is a need for separate internal storage, the microservice architecture allows for creating an internal database for storing the data. In addition, the service provides many ways to integrate with the services, including the REST API, Kafka and gRPC.

The configuration is done by passing environment variables to the application. This is the most secure and flexible implementation. To improve the development experience, a separate settings interface is created. It helps to centralize all the configuration needed and increase the application’s maintainability. Environment variables can also pass the path to the neural network model though the variant of passing the path using the CLI exists.

The production environment would require increasing the performance of the service as the data flow requires over 5 million predictions per second. However, this could be solved using the Kubernetes replica set to run multiple service instances and provide load balancing to handle such load. Even a lag of a millisecond is critical for such an ecosystem.

4.7 Service Integration Results

The solution was tested in the local testing environment. Further results will only include the usage of the model with the highest accuracy and the lowest loss values. Before integrating into an enterprise, the solution was tested locally using the environment consisting of Docker containers deployed on a local machine. The dataset with fake data improved resolution time by up to 17.3% for the local testing environment. Due to the usage of a service that uses a model with fake data instead of the production one, the prediction rate lowered from an initial 90% down to 79%. The service using a model trained with the operational data would have a higher prediction rate.

Access to the operational data is prohibited due to being confidential, and the data cannot be gathered for the thesis. The integration into an operational SIT, UAT or production environment is also strictly prohibited by the company’s privacy policy. Also, the direct integration to the production branch could drastically change the service workflow and affect the statistic. Therefore, changes are applied to a separate development Kubernetes cluster. Due to restrictions, only the model trained with the fake data will be applied to the integration step. Further model training with operational data is for the company’s internal use only.

After the testing inside Docker containers, the service is tested for scalability. A new Kubernetes environment using Minikube was created, and the deployment script was written. Access to the Virtual NOC services is gained through the internal container registry using the company’s internal network. The script uses the replica set structure for deployment with the replicas amount set to 10. Once the service had been deployed, the load testing was done using a Python script for sending requests to the service REST API. The success would mean handling at least 10 000 requests per second with only ten replicas of service being balance loaded. This means each replica should handle the load of 1 000 requests per second.

The load testing showed an impressive result of 11 273 requests per second for ten replicas of REST API. The latency for each request is slightly under 112 milliseconds. Asynchronous processing helped reduce the total processing time. For the complete load testing, the average accuracy of the model was equal to around 68%, which is an impressive result considering the fact model was not trained to handle amounts of alarms this large.

5 Future Work

Further development involves increasing the efficiency of an existing algorithm. This can be achieved by revising the internal structure of the neural network model and adding new features to the dataset. The features are the most critical thing in the model training. Due to the lack or excess of features. Whether or location-based features would be an excellent addition to the existing features and be added to the dataset pre-processing stage. Such features would increase the prediction rates for conditions caused by weather or the location, for example, deterioration of communication quality due to the formation of ice on antennas and other equipment or equipment failure due to high average temperatures.

The training shown in the thesis does not consider highly loaded traffic, though the model showed an impressive result for such occasions. The additional training for such cases will be included in the future development of the service.

Improving detection rates for the Flapping Detection algorithm is also necessary. Currently, Perpetual Detection is the main goal as it takes much time to execute. Flapping Detection would not benefit as much because it does not need many alarms to decide, but if some known algorithms spend too much time on the process, the neural network model could be used. It might be necessary to train a separate model for such cases because the flapping detections are different from Perpetual Detection, and the labelling would significantly differ.

The problem with any machine learning algorithm is data overfitting, which can be solved by increasing the number of units or hidden layers. An increase of hidden layers might worsen the performance and training time but will raise the model’s accuracy under the data. The LSTM model was considered because of being able to handle the relations between the alarms’ timestamps better though the model could be improved. This might include customization of the initial LSTM model.

Further development may include a solution to analyze and predict the next alarm before entering the system. This will require an increase in the amount of data for the analyst and a deeper analysis of the information received. There is no such implementation on the market at the moment. Even predicting one-fifth of the total number of events can significantly increase turnover speed.

The speed of this service is limited by the speed of the Python interpreter. If it is necessary to process requests faster and improve scaling parameters, the service can be rewritten in Java, Kotlin or Go since most of the existing services in the company are already written in these languages. The number of developers in these languages also exceeds the number of Python developers. Therefore, the latency using compiled languages would decrease significantly, and the execution time of the neural network model will reduce. The TensorFlow bindings for listed languages could be used for this purpose.

6 Summary

In conclusion, the solution provided in the thesis, which includes machine learning, might struggle on some occasions. The prediction rate will not be that close to the rule-based algorithms unless the dataset has enough features, and the pre-processing data stage will be configured to filter out all the data that could affect the training. Many things might influence the prediction rates and change the model behaviour making it hard to control. Using machine learning will help resolve edge cases and reduce the issue resolution time.

The data gathering process was done algorithmically. For real-world data, the active learning methodology will be used. The production data gathering was checked using the local testing environment using the SNMP-based monitoring service. The logs produced were parsed using the Python script and processed with the active learning to do the labelling. Dataset pre-processing filtered out less than 2% of invalid data due to parser imperfections and other issues. The pre-processing step also exchanged string typed data for categorical values.

During the thesis writing, many methods and configurations were used. The exception is the production data, which was not obtained, but the programmatically generated data was used for the prototype stage using the pre-processing steps described in the thesis. The optimization function comparison did not notice changes to the prediction rates, though using the Adam optimization algorithm helped reduce training time. The loss function comparison showed that the binary cross-entropy is the quickest one for the binary classification needs and slightly increased the performance of the training script. Embedding the initial dataset with padding and masking zeroes to exclude them from the output is the part that influences the result a lot. The size padding can vary the prediction rate by a few per cent and increase the training and execution time. The data overfits the model if the embedding step is skipped and stays around 65%. Once the embedding layer is included with padding of 10, the prediction rate rises to about 75-80%, but the change also increases the training time by 40%.

The integration step for the enterprise services is straightforward as it only includes running the execution script and configuring the path to the trained model. The script is scalable as the trained model execution time, CPU, GPU, and memory usage are low enough to scale the neural network properly. Also, the model is configurable to match the needs of the enterprise. Settings like padding, optimization function, loss function, and activation function are configurable and could be changed in future.

The Helm chart is used for version management, and the Kubernetes deployment script is added to the Harbor repository used in the enterprise for container image management.

References

[1] H. G. Abreha, C. J. B. Cano, A. D. L. Oliva, L. Cominardi and A. A. Saloa, "Self-Adaptive Monitoring in Fog Computing by Leveraging Machine Learning," 2018. [Online]. Available: https://www.academia.edu/40052411/Machine_learning_in_fog_monitoring. [Accessed 15 May 2022].

[2] NOC, "NOC Project - Open-Source Network Managemenet," 2019. [Online]. Available: https://getnoc.com. [Accessed 5 May 2022].

[3] S. Marsland, Machine Learning: An Algorithmic Perspective, CRC Press, 2011.

[4] D. Rafique and L. Velasco, "Machine Learning for Network Automation: Overview, Architecture, and Applications [Invited Tutorial]," 10 October 2018. [Online]. Available: https://upcommons.upc.edu/bitstream/handle/2117/125214/jocn-10-10-D126.pdf. [Accessed 16 February 2022].

[5] S. Hochreiter, "Long Short-term Memory," December 1997. [Online]. Available: https://www.researchgate.net/publication/13853244_Long_Short-term_Memory. [Accessed 5 May 2022].

[6] S. Pandey, M.-J. Choi, Y. J. Won and J. W.-K. Hong, SNMP-based enterprise IP network topology discovery, Int. J. Netw. Manage., 2011.

[7] A. Hwoij, M. Al-kasassbeh and M. Al-Fayoumi, "Detecting Network Anomalies using Rule-based machine learning within SNMP-MIB dataset," January 2020. [Online]. Available: https://www.researchgate.net/publication/339088761_Detecting_Network_Anomalies_using_Rule-based_machine_learning_within_SNMP-MIB_dataset. [Accessed 15 May 2022].

[8] P. Pandey, "Data Preprocessing: Concepts," Towards Data Science, 25 November 2019. [Online]. Available: https://towardsdatascience.com/data-preprocessing-concepts-fa946d11c825. [Accessed 05 May 2022].

[9] H. Sak, A. Senior and F. Beaufays, "Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling," 2014. [Online]. Available: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43905.pdf. [Accessed 5 May 2022].

[10] U. Verma, "Data Cleaning and Preprocessing," Analytics Vidhya, 19 November 2019. [Online]. Available: https://medium.com/analytics-vidhya/data-cleaning-and-preprocessing-a4b751f4066f. [Accessed 5 May 2022].

[11] R. Pramoditha, "How to Choose the Right Activation Function for Neural Networks," Towards Data Science, 19 January 2022. [Online]. Available: https://towardsdatascience.com/how-to-choose-the-right-activation-function-for-neural-networks-3941ff0e6f9c. [Accessed 05 May 2022].

[12] R. Sanghvi, "A Complete Guide to Adam and RMSprop Optimizer," Medium, 20 February 2021. [Online]. Available: https://medium.com/analytics-vidhya/a-complete-guide-to-adam-and-rmsprop-optimizer-75f4502d83be. [Accessed 5 May 2022].

[13] G. Al-Naymat, M. Al-Kasassbeh and E. Al-Hawari, "Using machine learning methods for detecting network anomalies within SNMP-MIB dataset," January 2018. [Online]. Available:

https://www.researchgate.net/publication/327445794_Using_machine_learning_methods_for_detecting_network_anomalies_within_SNMP-MIB_dataset. [Accessed 15 May 2022].

[14] D. W. Gürer, I. Khan and R. Ogier, “An Artificial Intelligence Approach to Network Fault Management,” 2007. [Online]. Available: http://www.sce.carleton.ca/netmanage/docs/An_AI_Approach.pdf. [Accessed 5 May 2022].

[15] S. Aidarous and T. Plevyak, Telecommunications Network Management into the 21st Century: Techniques, Standards, Technologies, and Applications, Wiley-IEEE Press, 1994.

[16] Pubrica Academy, “Overview of artificial neural network in medical diagnosis,” 16 October 2020. [Online]. Available: https://pubrica.com/academy/medical-writing/overview-of-artificial-neural-network-in-medical-diagnosis/. [Accessed 15 May 2022].

[17] GitHub. Inc, “TensorFlow–An open source machine learning framework for everyone,” GitHub. Inc, 2018. [Online]. Available: https://www.tensorflow.org/. [Accessed 15 May 2022].

[18] Elisa Polystar, “Network Monitoring - Focus your effort,” Elisa Eesti AS, 2021. [Online]. Available: https://www.elisapolystar.com/network-monitoring. [Accessed 5 May 2022].

[19] Elisa Polystar, “Polystar and Elisa Automate combine operations, bringing advanced automation and analytics solutions to operators worldwide,” Elisa Eesti AS, 4 May 2020. [Online]. Available: https://www.elisapolystar.com/combine-operation-automation/. [Accessed 16 February 2022].

[20] B. Cone, “The Best NOC Tools and Software (2022): An Expert Guide,” NOC Lifecycle Solutions, 25 January 2022. [Online]. Available: https://www.inoc.com/blog/noc-tools-and-software. [Accessed 16 February 2022].

[21] H. Kumar, M. Babu and S. Baskaran, “Machine Intelligence at the NOC,” Ericsson, 7 June 2018. [Online]. Available: https://www.ericsson.com/en/blog/2018/6/machine-intelligence-at-the-noc. [Accessed 16 February 2022].

[22] N. Williams, “Evaluating Machine Learning Algorithms for Automated Network Application Identification,” 2006. [Online]. Available: https://researchrepository.murdoch.edu.au/id/eprint/36 413/1/CAIA-TR-060410B.pdf. [Accessed 22 December 2021].

[23] A. Saabas, “Deploying Machine Learning Models on Node.js,” Bolt, 19 October 2018. [Online]. Available: https://medium.com/bolt-labs/deploying-machine-learning-models-on-node-js-ff25c540cb13. [Accessed 5 May 2022].

[24] I. H. Sarker, “Machine Learning: Algorithms, Real-World Applications and Research Directions,” 22 March 2021. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7983091/pdf/42979_2021_Article_592.pdf. [Accessed 15 May 2022].

[25] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed., Prentice Hall, 2009.

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Sergei Lukauškin

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Network Operations Center Monitoring Automation using Machine Learning on the Example of Elisa Eesti AS”, supervised by Hayretdin Bahsi
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until the expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the Tallinn University of Technology library until the expiry of the copyright term.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons’ intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

16.05.2022

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Example of SNMP trap data structure for VSWR alarm

Table 2. Example of SNMP-trap with VSWR alarm

Field Name	**Example Value**
alarmAckTime	*null*
alarmAddionalInfo	RF Unit Name=0-20-0
alarmCSN	123456789
alarmCategory	FAULT
alarmClearCategory	1
alarmClearType	0
alarmConfirm	2
alarmDevCsn	12345
alarmExtendInfo	Cabinet No.=0, Subrack No.=20, Slot No.=0, TX Channel No.=0
alarmID	12345
alarmLevel	3
alarmMOName	SITENAME_TAL123-Cabinet No.=0, Subrack No.=20, Slot No.=0,0-20-0
alarmOccurTime	2021/08/12 - 13:32:12 +02:00[+01:00]
alarmOperator	*null*
alarmProbablecause	RF Unit VSWR Threshold Crossed
alarmRestore	2
alarmRestoreTime	*null*
alarmServiceAffectFlag	0
alarmSpecificproblems	*null*
alarmType	5
ipAddress	*null*
trapTime	*null*