TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

SABOKTAKIN HAYATI

PREDICTION AND DETECTION OF ABNORMAL USAGE OF AN ELEVATOR

Master's thesis

Examiners: Dr. Tomi Krogerus and
Assoc. Prof. Heikki Huttunen

The examiners and topic of the thesis were approved on 30 May 2018

ABSTRACT

SABOKTAKIN HAYATI: Prediction and detection of abnormal usage of an elevator
Tampere University of Technology
Master of Science Thesis, 50 pages
November 2018
Master's Degree Program in Science and Engineering
Major: Data Engineering
Examiners: Dr. Tomi Krogerus and Assoc. Prof. Heikki Huttunen

Keywords: forecasting, predictive maintenance, LSTM, anomaly, machine learning, RNN, deep learning

In this thesis we used machine learning to detect the anomalous use of elevators by measuring the behavior at any specific time. We examined the data for unusual patterns in comparison with observed samples from the history of the elevator. We investigated forecasting the future use of the elevator to define an abnormal behavior for elevators and tried to address this issue in two approaches. First, we used Long Short-Term Memory to forecast future usage, and we optimized the result by extracting features and removing the noisy part of the data. Then we compared actual usage with our prediction and used 99.7% confidence interval (three σ rule) to find out anomalies. The second approach used a local outlier factor to find out the distance of each week's usage of the elevator from the other weeks. Then we took the intersection of these two methods, performed a set of post-processing actions to decrease the ratio of false positives and remove anomalies which were not sustained.

PREFACE

With the fast emergence of the Internet Of Things(IoT) and the advent of sensors, there has been an enormous explosion of data. This extensive data offers great analysis opportunities for developing the field of predictive maintenance. Different machine learning methods have attracted attention due to their state-of-the-art performance in this field, especially deep learning; I was fortunate to have the opportunity to work in this field. I have always wanted to have a challenging task that is at the same time useful and have the potential to solve a problem in people's everyday lives.

Without the great people who are my friends and support me, I would not have been able to undertake this journey. First, I want to state my deepest gratitude to Heikki Huttunen and Tomi Krogerus, my supervisors for their valuable time, and the patience and support that I received from them.

Special thanks also go to Jerker Björkqvist for his comments. Also, special thanks go to Morteza Zabihi, Pedram Ghazi, and Ilari Kampman, who are kind friends and always open to helping as inspiring scientists who were available whenever I had problems, always making things look easier in python coding and machine learning topics.

Then I have to mention that I have had the good fortune to study, work and live with some wonderful people in Tampere: Thank you all. Moreover, I would like to thank my family, and my beloved aunt Marziyeh Hayati and her family for their love and support. To my brothers, Abolfazl and Morteza: thank you kindly for always being supportive and wonderful. Finally, I would like to dedicate this study to my late mother and father, and I hope that I have made them proud of me.

Tampere, Finland, 09 October 2018
Saboktakin Hayati

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

ABOD	Angle-based Outlier Detection
ACC	Accuracy
ANN	Artificial Neural Network
AUC	Area under the Curve
BPTT	Back Propagation through Time
CAGR	Compound Annual Growth Rate
CNN	Convolution Neural Network
FN	False Negative
FP	False Positive
GRU	Gated Recurrent Unit
IOT	Internet of Things
LSTM	Long Short Term Memory
LOF	Local Outlier Factor
NN	Neural Network
TN	True Negative
TP	True Positive
TUT	Tampere University of Technology
PDF	Probability Density Function
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic

1. INTRODUCTION

Predictive maintenance monitors the status of the system by performing a set of online or periodic tasks on different types of equipment in order to predict the right time for maintenance. The main goals are predicting the proper time for scheduling maintenance operations, and preventing unexpected downtimes due to failures. If the maintenance team has the right information in time, the maintenance plan can be optimized in different aspects.

It has been proven that investing in predictive maintenance is beneficial for companies and saves the budget. This has been experimented in a work undertaken by superior corporations such as General Motors [1]. In addition to saving funds, the use of personnel and equipment becomes more efficient. Small issues can be addressed before turning into failures; uptime will then be raised while servicing costs will be reduced. Also, the end user who uses the product will be more satisfied because there is less system failure or downtime.

Additionally, since the use of spare parts is becoming intelligent in predictive maintenance, this can affect the environment by producing less waste and managing resource consumption. For example, assuming the standard for changing the timing belts of engines is based on time intervals, so with the intelligence provided from predictive maintenance, this standard could be flexible based on different factors in different engines. For this thesis, the elevator system has been chosen as a use case for studying predictive maintenance methods. Elevators can be defined as electro-mechanical complex systems which are used to carry things vertically [2] (or horizontally [3]). Figure 1.1 presents the basic model of the elevator.

Elevators are intertwined with people's lives in such a way that life without them could not be imagined. Experts estimate that there are more than one million elevators in the United States and each of them serves up to 20 thousand persons on average, and 18 billion travel per year [4]. Also, the market size of elevators and escalators will grow with a compound annual growth rate(CAGR) of 6.01% and will be $ 125.22 billion by 2021 [5].

An elevator is a complex system which can consist of hundreds of pieces of equipment. The maintenance of such a complicated system is challenging and needs to be addressed with up-to-date techniques. Therefore, there is vast scope for building and developing predictive maintenance frameworks in this area.

According to Skog et al. [6], in order to respond to increasing demand in the elevator market, the industry has identified the need to collect data from elevators, and nowadays most modern elevators are connected to the cloud, and collected data are utilized to implement predictive and preemptive maintenance plans.

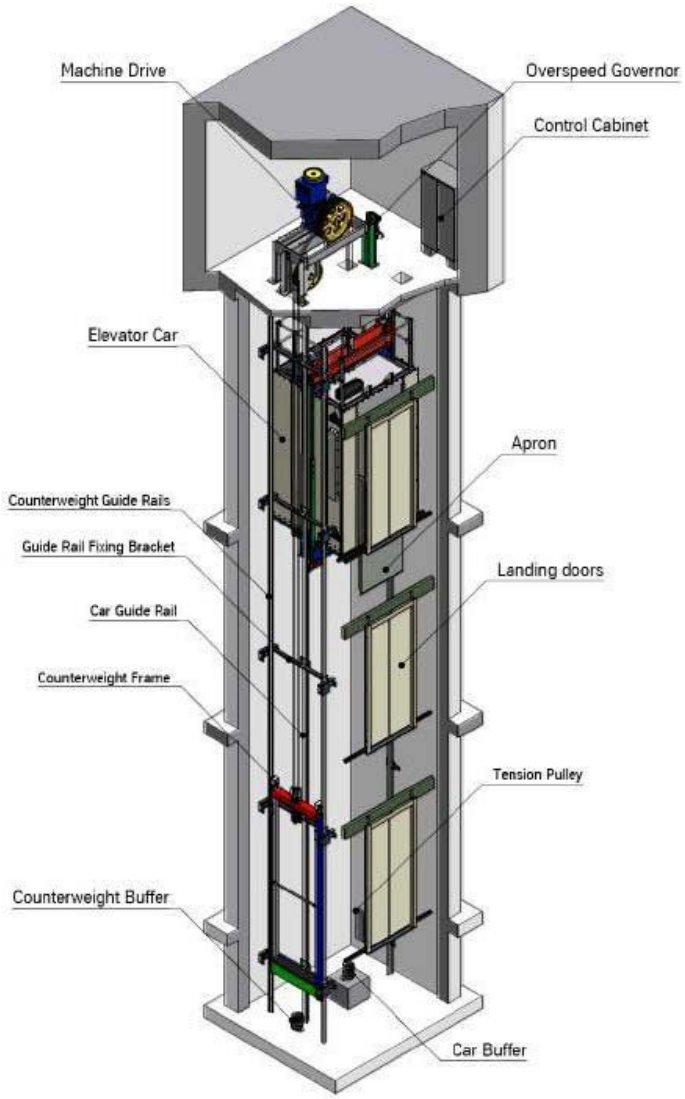

Figure 1.1. *Schematic view of a basic elevator. [2]*

Predicting the future or forecasting is not a routine task and there are two significant problems.

First, fully automatic forecasting methods are often complicated and have a lot of hyperparameters. Therefore, fine tuning these algorithms is a hard and resource/time-consuming task. Second, the experts who are responsible for the analysis task in companies typically have domain knowledge of and expertise in the services or products that they support in the organization, but they often do not have knowledge of time series forecasting or anomaly detection techniques. Experts in both domains are rare because prediction and anomaly detection are specialized skills requiring considerable experience. [7]

The demand for predictive maintenance raised expectations in some related areas, such as the prediction of the behavior of equipment and anomaly detection. We intend to provide a model for reaching this goal by building a continuous iteration which includes every accessible source of tacit knowledge and information. This is the typical approach for solving complex problems with machine learning algorithms and human knowledge. Figure 1.2 illustrates this continuous iteration. This study focuses on the part which is indicated by the red rectangle.

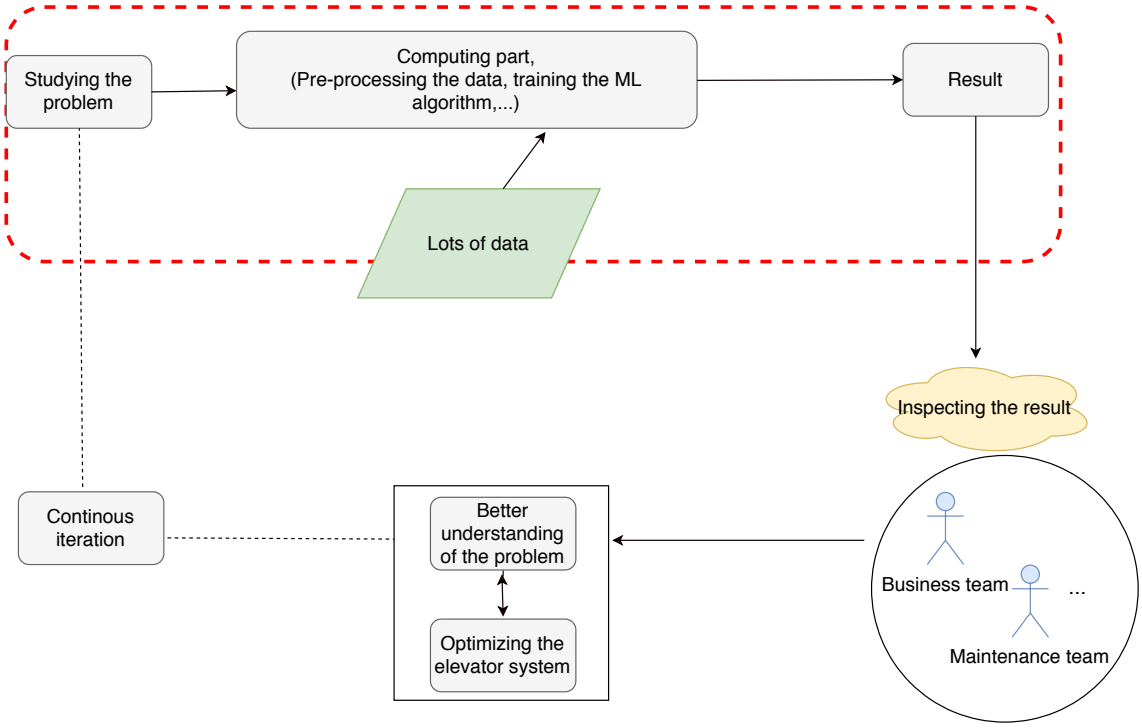

Figure 1.2. *A common approach to solving complex problems with machine learning algorithms and human knowledge. Adapted the idea from [8]*

The following chapters are constructed as follows. Chapter 2 provides a theoretical background, which includes anomaly detection basics, machine learning, neural networks, deep learning, recurrent neural networks and different types of these, evaluation methods, local outlier factor and angle-based outlier detection. Chapter 3 talks about an overview of the solution, input data and preprocessing. In Chapter 4 technical details of the implementation and results are presented. Finally, conclusions and possible future work are provided in Chapter 5.

2. THEORETICAL BACKGROUND

2.1 Anomaly detection

Anomaly detection is the action of finding sections of data which do not follow the regular or normal pattern. Anomalies are notably different from the other data, and there is a high possibility that they are generated by different routines. An anomaly is also called an outlier, abnormality, discordant and deviant. Anomaly detection has different applications, such as in intrusion detection [9], fraud detection [10], medical diagnosis [11], earth science [12] and sensor events [13]. In all these applications, there is a usual pattern, which could be a set of observations or actions, and the normal behavior of the model is defined based on these patterns. In terms of anomalies, the generated pattern often contains valuable information about the abnormal characteristics and entities that are involved in the process of making this event. It should be noted that there are also some related concepts such as novelty and noise. There are some terms related to anomaly domain which are defined in the following.

- **Novelty**

  Novelty is an updated and unseen behavior in the typical pattern of the system; for example, removing a coffee machine to another floor can be considered as a novelty. The pattern of daily usage of the elevator has been changed to organize having a coffee: it is normal and unobserved in the data.

- **Noise**

  Noise can be defined as an event in the data that is not of interest to the researchers but acts as an impediment to unveiling new patterns; typically, noise is unwanted and does not contain useful information. Noise removal is driven by the need to remove the undesired points before any data analysis is performed. Noise accommodation refers to immunizing a statistical model estimation against anomalous observations. [14]

- **Outlier scores**

  Some outlier detection algorithms sort the severity of the outlierness of each point as an output. The most far away data points have a high score in outlierness. Thus every datapoint would be assigned to a decimal number as an output of the algorithm.

- **Binary labels**

  The other type of output is a binary label that indicates whether a data point is an outlier or not. By setting thresholds on outlier scores, outlierness can be converted to binary labels. Also, the threshold is chosen according to the distribution of the scores. [15]

The separation between noise and an outlier is not precisely defined. Application and analyst subjective are two important factors in distinguishing between noise and outlier. As figure 2.1 illustrates, noise and outliers are also termed a weak and strong outlier, respectively [15].

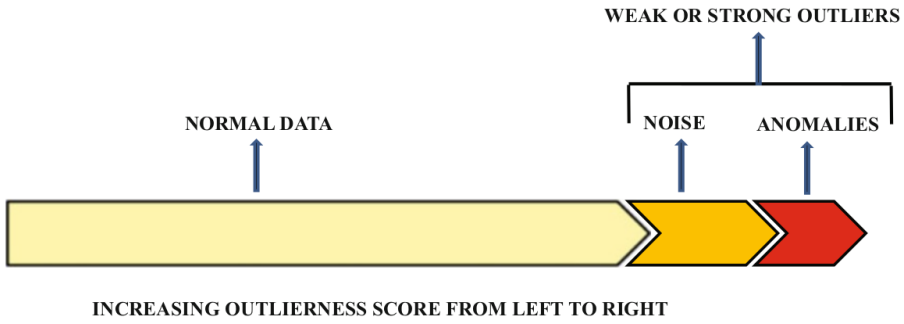

Figure 2.1. *The spectrum of data, from normal to anomalies. [15]*

2.1.1 Challenges in anomaly detection

This section discusses the different aspects of anomaly detection mentioned earlier. Various factors such as the type of data, the existence of labeled data as well as the limitations and requirements of the application domain and categorizing anomaly detection algorithms are presented.

Generally, defining the normal behavior of data, then dividing the data into two different sections is not a straightforward task, especially when the labeled data are not available. Some challenges are addressed here.

- Defining a boundary as being an outlier or not is not trivial. Thus, should a data point near to the border, on the borderline, be defined as normal or not?
- In some contexts, normal patterns keep updating and changing. So the current definition of normal might not be acceptable for future works.
- Most of the time, generalization is not possible. The notation of anomaly is different in various domains. For example, in the medical domain, a subtle fluctuation is an anomaly, while in a stock market that might be considered as normal.
- The noise which comes with the original data and tends to be like an anomaly: how could this be removed with less affect on valuable information?
- Lack of ground truth or labeled data. Without having any annotated data, validating/-training a model is difficult.

Due to these facts, in most cases anomaly detection is not easy to solve. In fact, most of the current anomaly detection techniques solve a particular type of problem. The problem is affected by different factors, such as the availability of the ground truth data.

In the application domain, the type of anomalies need to be detected. Figure 2.2 shows the main factors associated with anomaly detection techniques.

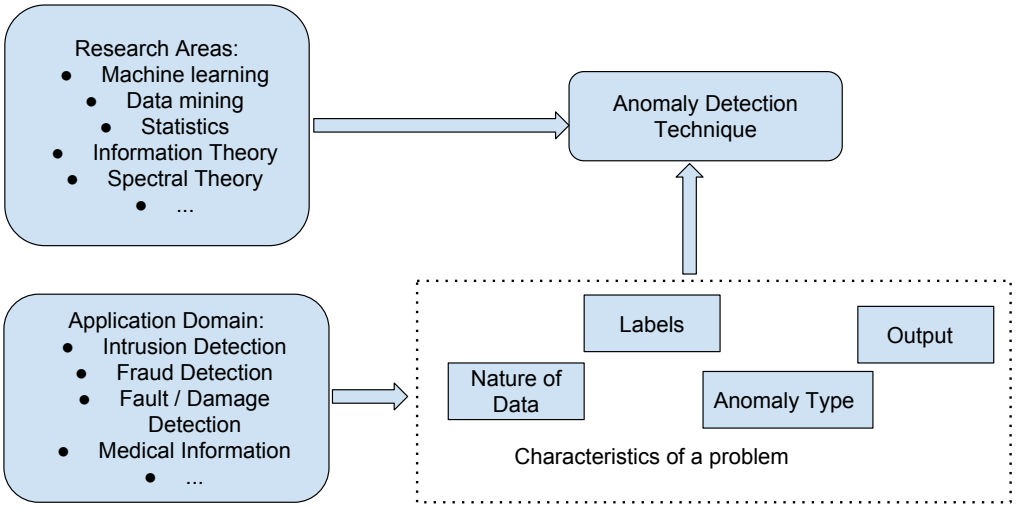

Figure 2.2. *Main factors related to anomaly detection method. Adapted from [14]*

Type of input data

The characteristics of the input data is an essential aspect of any anomaly detection algorithm. Data input consists of a set of observations/samples which can be described by a set of variables (fields, features). Variables can be binary, continuous or categorical. Data instances can consist of only one variable or multivariable, univariate and multivariate, respectively. The most typical data types are those which are related to each other: for example, graph data, spatial data and sequence data. In graph data, the data instances are modeled as a graph, the relationships are edges, and each vertex represents one data: for example, social media streams like Twitter. In spatial data, each data is related to its neighboring instances: for example, wireless sensor networks (WSN). In sequence data, each data point is linearly ordered, such as protein sequences, time series data and genome sequences.

2.1.2 Types of anomalies

Anomalies can be grouped regarding their types; thus, different anomaly detection tools are appropriate for different types. The following are different types of anomalies.

- **Point anomalies**

  If a particular data instance can be named as anomalous in comparison with the rest of the data, then the instance is termed a point anomaly; this is the most straightforward class of anomalies. For example in fraud detection of a credit card, if there is a transaction that is highly different from others, this might be considered as an anomaly.

- **Contextual anomalies**

If a data instance is anomalous in a particular context, but not otherwise, then it is termed a contextual anomaly (conditional anomaly). For finding a contextual anomaly, each data instance needs to has two types of attributes: contextual and behavioral. Contextual attributes are used to identify the neighborhood of the desired instance. For example, in time-series data, time is a contextual attribute that identifies the position of an individual in the whole series, while all the non-contextual attributes are called behavioral attributes. For example in spatial data, describing the average humidity on the earth, the amount of humidity at any location is a behavioral attribute. Therefore, the values of behavioral attributes within a different context can be considered as normal or an anomaly. Contextual anomalies have been most commonly used in time-series data. Figure 2.3 shows an example of the monthly temperature of a specific area over the past few years. A temperature of $38°F$ might be normal during the winter (at time t_1) at that place, but the same value at a different time, during the summer (at time t_2), would be considered as an anomaly.

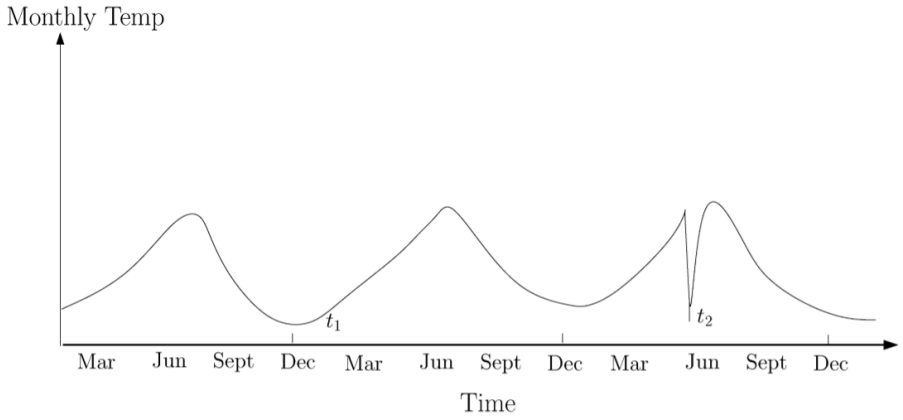

Figure 2.3. *Contextual anomaly t_2 in a time-series (monthly temperature). The temperature at time t_1 (during the winter) is the same as that at time t_2 (during the summer) but happens in a different context and hence is not recognized as an anomaly. [14]*

- **Collective anomalies** If a group of related data instances is anomalous concerning the entire data set, it is termed a collective anomaly. Single data instances in a collective anomaly may not be anomalies by themselves, but their accumulation is anomalous. This type is valid for data types which are related to each other, sequence data. Figure 2.4 presents the human electrocardiogram. There is an abnormally long time existence of low value which can be considered as a collective anomaly; note that the single low value is reasonable.

The next concept is data labels. Data labels are correlated to data points and represent whether the data is an anomaly or not. Annotating, which is the act of putting labels on each data instance, is an expensive job, and sometimes impossible. For example, finding abnormal behavior in an airplane engine in all fully operational circumstances. Moreover, covering all possible anomalous behavior is impossible due to the dynamic nature of the system. Based on the availability of the labels, algorithms can act in three modes:

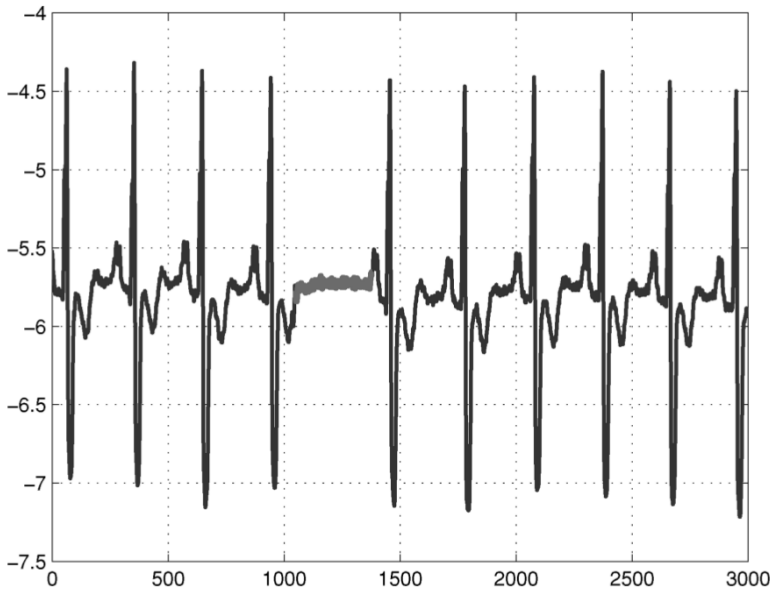

Figure 2.4. *Collective anomaly detected in an atrial premature contraction of a human electrocardiogram output. [14]*

- **Supervised anomaly detection**

  The assumption here is that there is a label for every training instance whether abnormal or normal. Then the problem becomes similar to classification problems. A conventional approach is creating a model with train data and feeding the new data to the model, compared against the model to identify which class it belongs to. Usually, the number of anomalies is much less than normal instances; thus, the issues related to imbalanced class distributions should be addressed carefully.

- **Semi-supervised anomaly detection**

  In this mode, every data instance which belongs to the normal class has a label. Then the model is built based on the train data, and new data is fed to that and the anomalies identified. Labeling data in this mode is much cheaper than when supervised; thus, many anomaly detection algorithms widely use this mode.

- **Unsupervised anomaly detection**

  There is no label in this mode; thus, unlike supervised learning, there is no labeled training dataset. The techniques in this mode make the implicit assumption that normal instances are more frequent than anomalies in the test data. If this assumption is not valid, then such algorithms suffer from a high false alarm rate.

In the machine learning area and this thesis, rather similar terminology will be used for categorizing machine learning problems.

2.2 Time series and similarity measures

A time series data is like a 1-dimensional array, whereas indexes are replaced with timestamps. Equation 2.1 defined a row representation of time series T, which is a sequence of pairs

$$[(v_1, t_1), (v_2, t_2), ..., (v_i, t_i), ..., (v_n, t_n)](t_1 < t_2 < ... < t_i < ... < t_n) \tag{2.1}$$

Where each v_i is a data value in a d-dimensional data space, and each t_i is the timestamp at which v_i occurs. And n is the length of the input.

Two major factors for achieving effectiveness and high performance when managing time series data are representation methods and similarity/dissimilarity measures. Time series are high dimensional data, and working with them is challenging and expensive, so the demand for developing representation techniques which can reduce dimensionality without losing fundamental information is high. Furthermore, the definition of distance in regular data types (e.g., ordinal or nominal data types) is trivial. On the other hand, the distance between time series needs to be thoughtfully addressed in order to indicate the hidden (dis)similarity of the data. This is especially useful for similarity-based, clustering, classification and other mining algorithms of time series. [16] The techniques that represent time series with reduced dimensionality are not within the scope of this thesis, but this section discusses two distance metrics: Euclidean distance and dynamic time warping. Figure 2.5 depicts Euclidean distance and DTW difference in finding the similarities of two signals over the time.

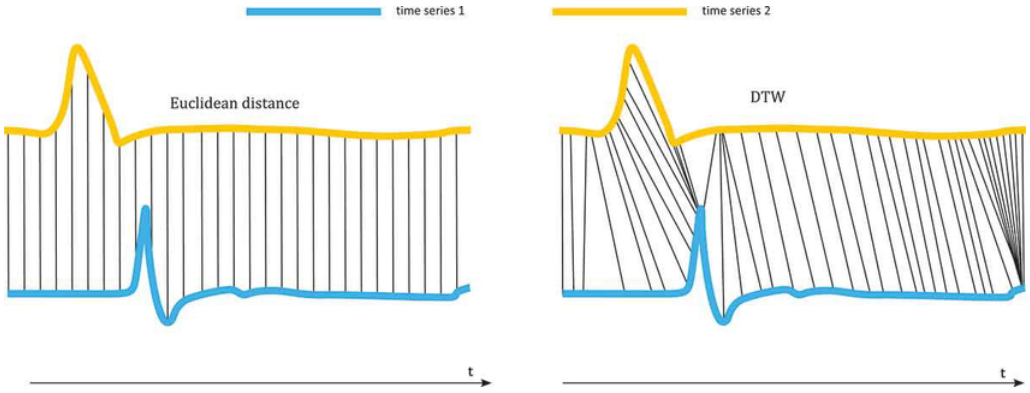

Figure 2.5. *A comparison between the distance measurement of two time series in Euclidean and DTW. [17]*

- **Euclidean distance**

  Euclidean distance is one of the most straightforward measure algorithms. Besides being intuitive, Euclidean distance implementation is easy and parameter-free. Additionally, in terms of complexity it outperforms the DTW, especially in large datasets. On the other hand, since Euclidean distance mapping between points is fixed, unlike DTW, this distance measure is sensitive to the noise and cannot handle local

time-shifting. Noise in this context can make some data points of time series be dampened, or completely dropped, or cause misalignment. The Euclidean distance is a well-known method of measuring the distance between two series, that is, the sum of squared differences. For two series S and Q of the same length l, their distance is defined as the following:

$$D(S;Q) = \sum_{i=1}^{l} (S_i - Q_i)^2 \tag{2.2}$$

- **Dynamic time warping**

  Dynamic time warping (DTW) is a classic, popular method for finding an optimal alignment between two sequential series. DTW originated from the field of automatic speech recognition. DTW has been successfully verified to automatically cope with time deformation challenges and different speeds associated with time-dependent data points. The program/pseudocode 2.1 illustrates the work of DTW algorithm, and the following is an explanation of the pseudocode. The algorithm works by taking two points as arguments to find out their distance. First, it sets the distance at each point to infinity, and then for each point it seeks a way to match that to the point in other time series, in such a way that the distance would be minimized. It gets the minimum of either moving one forward, moving another point forward, keeping them matched up and getting that minimum distance.

```
1  FUNCTION DTW(s1,s2)
2    D_ARR = Array []
3    lengthS1 = Array [0,length(s1)]
4    FOR I = 0 to lengthS1
5          ## at first, All the distances set to infinity.
6          D_ARR[(I, -1)] = float('inf')
7    lengthS2 = Array [0,length(s2)]
8    FOR I = 0 to lengthS2
9          D_ARR[(I, -1)] = float('inf')
10   D_ARR[(-1, -1)] = 0
11   FOR I = 0 to lengthS1
12         FOR J = 0 to lengthS2
13           distance = (s1[I]-s2[J])**2
14           Min=minimum(D_ARR[(I-1,J)],D_ARR[(I,J-1)],D_ARR[(I-1,J-1)])
15           D_ARR[(I, J)] = distance + Min
16
17   ret = sqrt(D_ARR[lengths1-1, lengths2-1])
18   RETURN ret
19 ENDFUNCTION
```

Program 2.1. *An example implementation of DTW algorithm in pseudocode. The function DTW() receives two input sequences and returns a dynamic time-warped distance.*

The fact that DTW is not a standalone distance metric like Euclidean distance should be taken into account. In other words, the only meaningful way of using DTW is in a

comparative sense. DTW has one notable drawback which could found from this snippet, namely that it is resource consuming and so does not get a good score regarding performance.

2.3 Machine learning

Machine learning (ML) is the process, where the machine learns from data and tries to build a framework for producing reliable, accurate predictions and identifying patterns from high-dimensional data. It is specially for sets of tasks, that are too hard, complex and error prone for a human to do. Additionally, there is a rather old-fashioned classic definition of machine learning by Mitchel [18]: ”A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T , as measured by P , improves with experience E”.

Machine learning algorithms are often divided into the three groups of supervised learning, unsupervised learning, and semi-supervised learning, depending on the type of available data. There is also another group of machine learning algorithms in addition to these three, called reinforcement learning, This learning method is different and done by interacting with the environment and making actions and refinements with assigning punishments or rewards [19].

- **Supervised learning**

  Supervised learning is where the model has the chance to learn from a set of labeled examples: it means every input$(x_1, x_2, ..)$ mapped with an output $(y_1, y_2, ..)$. The goal is finding a proper function that approximates the model behavior in a generalizable manner. Thus, the function receives x_i as in input and predicts y_i as an output. Depending on a task, the output can be a real number, regression or a class label, classification.

- **Unsupervised learning**

  In the case of unsupervised learning, the model receives inputs without any target labels. There is no supervision for the model to learn the right behavior; the algorithm should learn from the hidden structure of the data. Density estimation (estimating underlying PDF for prediction) and k-means clustering are two notable examples of unsupervised learning.

- **Semi-supervised learning**

  Semi-supervised learning is something in between; it uses both labeled and unlabeled data for building the model. This method is motivated by the fact that labeled data is often costly to produce, whereas unlabeled data is generally not expensive. Typically, a mixture of data includes a large amount of unlabeled data and a small set of labeled data. However, how to mix them is a challenging technical question that needs to be addressed carefully. [20]

2.4 Neural networks

Neural network (NN) algorithms are categorized as supervised machine learning algorithms, so they try to find the patterns from the labeled datasets and build a model for prediction, but in a similar manner to the human brain (when the brain performs similar tasks). In other words, a neural network's main function is to receive a set of inputs to perform progressively complicated calculations and use the output to answer a specific question; for example, does the input image belong to a cat or not?

The structure of the neural network is like any other kind of networks, there is an interconnected network of nodes which are called neurons, and the edges that join them together. Figure 2.6 shows an example of a single unit neuron.

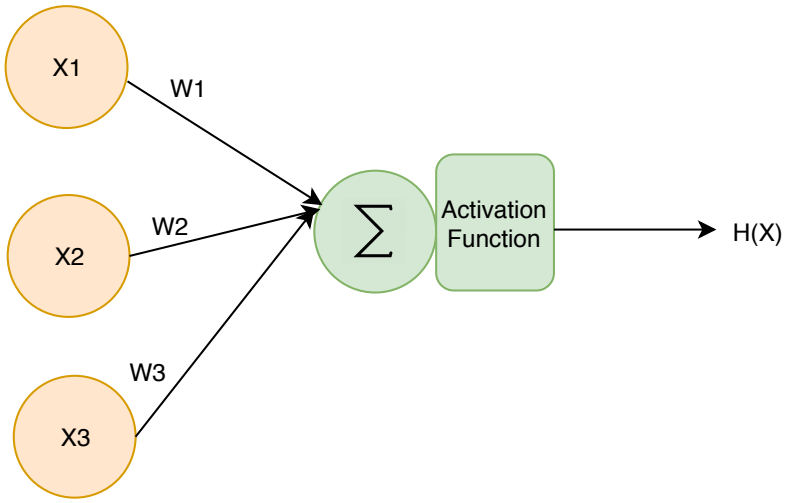

Figure 2.6. *A single unit of neural network with an activation function and 3 inputs.*

This single neuron has 3 inputs x_1, x_2, x_3 and each input has its own weight. The activation function will be explained later, and the equation is defined as:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

$$h(x) = g(x_1 w_1 + x_2 w_2 + x_3 w_3) = \mathbf{W^T X} \tag{2.3}$$

where g is the activation function, and this function transforms the input to a different range (e.g.[0,1]). Note that in this example bias is not considered.

A neural net can be thought of as the stack of classifiers together in a layered web, since each node in the hidden and output layers has an embedded classifier (activation function).

Take a node in the first hidden layer as an example: it gets input from the input layer and activates, and its score is then passed on as input to the next hidden layer for further activation. So it advances until it reaches the output layer, where the scores determine the results of the classification at each node. This happens for each set of inputs and called forward propagation. Figure 2.7 shows a feed-forward neural network, and this example model utilizes a bias term in order to prevent zero-sum distorting the learning procedure.

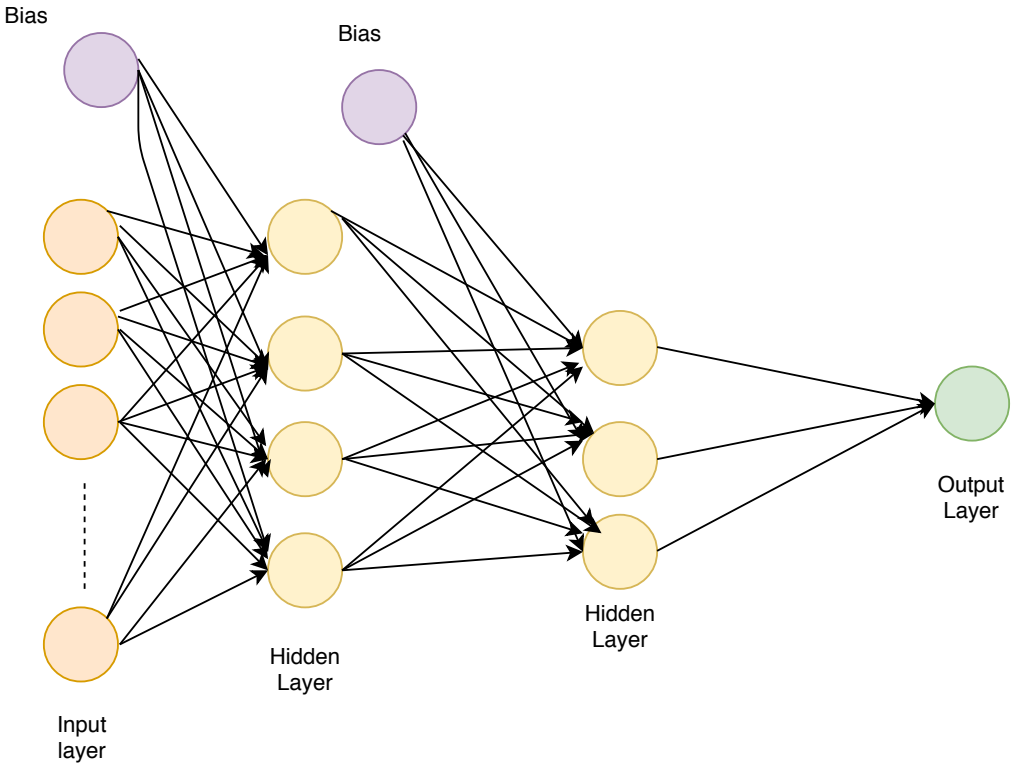

Figure 2.7. *Example of a simple multilayer neural network with 2 hidden layers.*

The reason this model is called feed forward is that there are no backward connections which feed the outputs back to the network. The following equation is for calculating the output for neuron j:

$$t_j = \sum_{i=1}^{m} w_{ji} x_i + b_j, \tag{2.4}$$

where $\mathbf{x_i}$'s are inputs, $\mathbf{w_{ji}}$'s multiplied weights, m is the number of inputs, and b is the bias term. The variable i is the i^{th} layer of the network and j the j^{th} hidden unit of the layer. In order to simplify the above equation, let $b_k = x_0$, then we have:

$$t_j = \sum_{i=0}^{m} w_{ji} x_i = \mathbf{w_k^T x}, \tag{2.5}$$

then the output will be produced by passing t_j through the activation function g.

$$h_j = g(\mathbf{w_k^T x}), \tag{2.6}$$

where h_j is the output. There is also backpropagation which does the reverse of forward propagation, training the networks from the last layer to the first layer backwards.

2.4.1 Activation function

Activation functions are a set of functions which are working at the end of hidden units and inject non-linear complexities to the network model. They can be thought of like switch components which are operating on the data before the output. Figure 2.8 shows the most popular activation functions.

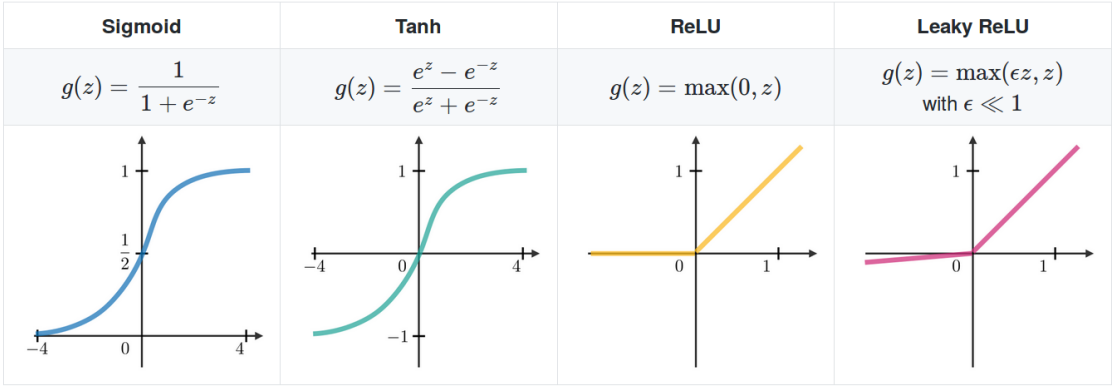

Figure 2.8. *Some of the most popular activation functions. [21]*

2.4.2 Methods for learning in neural networks

In most supervised machine learning algorithms, including neural networks, there should be a proper solution for estimating parameters based on training data. The ultimate objective of all optimizers is to reach the global minima, where the cost function achieves the least possible value/model error. A well-known, effective and easy-to-use method is gradient descent. The gradient descent algorithm calculates the weights of the model in many iterations by minimizing a cost function at each step. Figure 2.9 depicts a visualization of an example of a cost function.

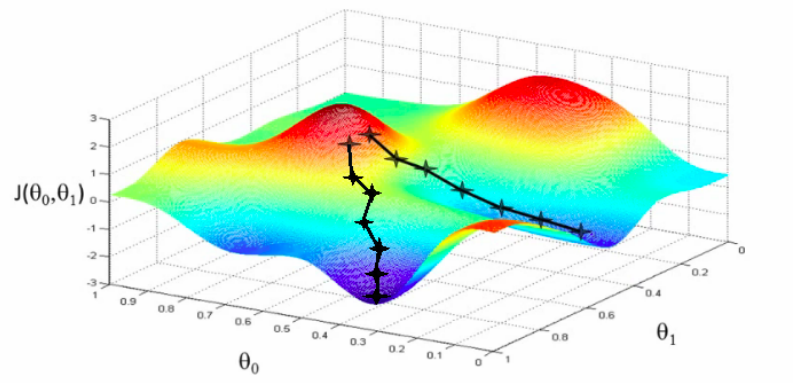

Figure 2.9. *An example of cost function. [22]*

The following formula depicts how NN updates the weights during the training.

$$W_j = W_j - \lambda \partial F(W_j)/\partial(W_j) \tag{2.7}$$

where W_j is a vector of parameters, F is the cost function, $\partial F(W_j)/\partial(W_j)$ is the derivative with respect to W_j, and λ is the learning rate.

At each step, the expression finds the new values for weights (and biases) by taking the gradient and reducing that from the old value, so by reducing it goes towards the global minimum. There is a possibility of being trapped in a local minimum (instead of global) if the derivative is not monotonic. In this situation, an easy way is repeating the process by initializing different W_j and computing the cost function again. Also, it could happen from applying a wrong learning rate. If a small value of a learning rate is selected, the model will reach the local minima; however, it might take a long time to converge. On the other hand, by assigning a big learning rate, the model could never be able to converge to the global minima and will always fluctuate around the global minima. Figure 2.10 depicts these two effects of choosing a bad learning rate.

Learning rate is a hyper-parameter that regulates how much adjustment would be applied on the model with respect to loss gradient. The value of the learning rate can be assigned or adaptively changed with methods such as Adam [23] or RMSprop.

Rmsprop is an optimization method which works by holding a moving average of the squares of last gradients and then using the root of this average to normalize current gradient [24]. The update equation for weight parameter w_i (an alternative version of equation 2.7) is:

$$\Delta w_i \leftarrow -\frac{\lambda}{\sqrt{s}} \frac{\delta \varepsilon}{\delta w_i} \tag{2.8}$$

Where the λ is the learning rate and s is the moving average that can be defined as :

$$s \leftarrow \alpha \Delta w_i - (\alpha - 1)(\frac{\delta \varepsilon}{\delta w_i})^2 \tag{2.9}$$

Where α is a positive constant called the momentum and is usually fixed to the value 0.9.

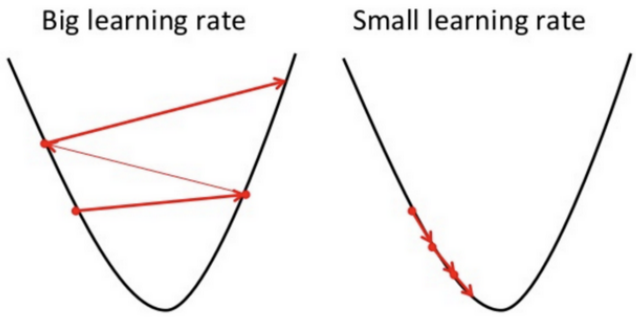

Figure 2.10. *Two critical situations: a big or small learning rate is selected. [25]*

2.5 Deep learning

Deep learning is a part of machine learning which has a rather different learning representation of data and emphasis on learning consecutive layers of representations. The deep learning model does not necessarily have to have a deep architecture; the word deep here stands for the idea of hierarchical representation learning. The number of layers of the model are termed as the depth of the model. Deep learning models are automatically learned from training data. The other machine learning approaches which are focusing on only one or two layers representation of data are named as shallow learning [26]. In deep learning, these hierarchical layered representations are learned via neural networks.

In the following section, the theoretical knowledge of a few common deep neural networks, such as the CNN and RNN families will be presented.

2.5.1 Convolutional neural network

This section begins by introducing a convolution operation and their application in deep learning since a convolutional neural network holds on top of the definition of the convolution operation. A convolution is a mathematical operation between two signal/functions which has the output of integral of the product of the two functions where one of them is reversed [27]. The equation below shows the convolution operation.

$$f * g = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = \int_{-\infty}^{\infty} g(\tau)f(t-\tau)d\tau \tag{2.10}$$

convolution is very similar to the correlation operation, the only difference they have is no function is reversed in the correlation. The convolution operation can be generalized to more than one dimension, and in fact in two-dimensional (2D) convolutions are also notably useful in fields such as image processing. In image processing tasks, the reversed function in convolution is termed as kernel or filter, so a kernel passes over the first function (image).

Neural network models that use convolution operations are called convolutional neural networks (CNN), or convnets. Convnets are a type of deep neural network that are used in most computer vision applications. Convnets work very well for spatial data [28–30] and are capable of encoding the data properties into layered architecture. Convnets have the two following interesting properties.

- **Translation invariance of features**

  Learning a specific feature or pattern does not relate to the location of that feature. It can be recognized in the new images regardless of the location. For example, if the task is bird detection, the system is learned from training images and builds the model. The system knows about the feather pattern and can detect that in any new image, like the real visual world. It happens for the human brain as well: the image of a mountain is always mountain, no matter from which angle you are looking at it or with one eye or from how far away you are looking at the picture.

- **Spatial hierarchies of patterns**

  The sequence of convolutional layers can learn sophisticated patterns since final patterns are made on the top of the output of early convolutional layers. Figure 2.11 shows a flow which also refers to the concept of hierarchy of patterns.

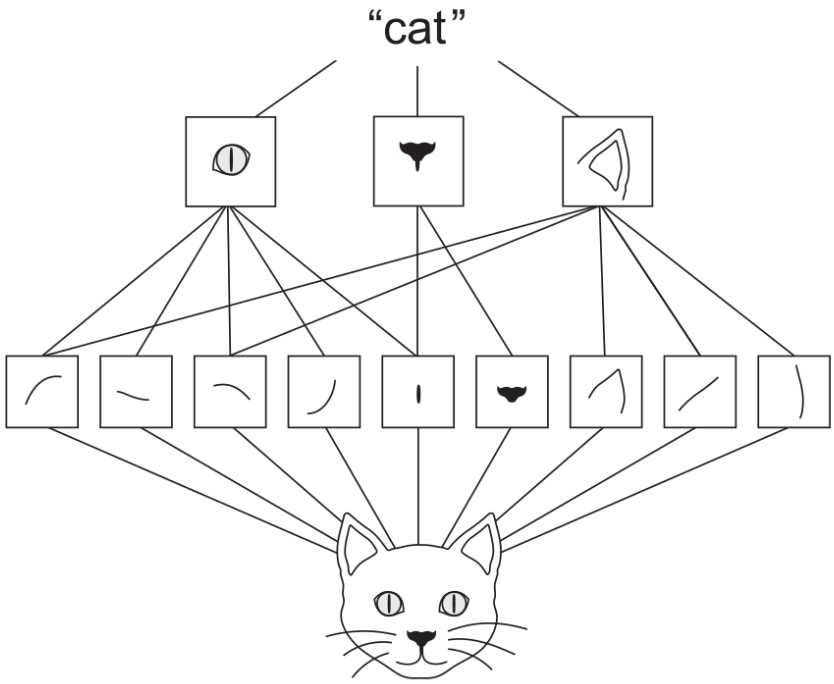

Figure 2.11. *The visual nature produces a spatial hierarchy of visual components: hyper-local edges merge into local objects such as ears or eyes, which combine into high-level entities such as “cat.” [31]*

The following is an explanation of the CNN architecture by showing a classification example, illustrated in figure 2.12. The convnet normally consists of three main components/layers: fully connected, convolutional and pooling layers. Here, the assumption is that spatial data is an image. CNN receives images as an input in three dimensions: width, height, and depth. Depth is the number of channels, so if the image is in RGB format, this parameter is equal to 3 (red, blue, green), and is transferred through the layers. A CNN is made up of stacked layers: each layer has a simple shared task; it transforms a 3D input to a 3D output with some procedures that may or may not have parameters. The input data is passed through the network and reaches the convolutional layer. Feature extraction is performed with dot product between filter weights and corresponding local input pixel window; the weights for the convolutions at each location are shared. Then an activation map will be produced by sliding each filter window across the width and height of the whole input image. In most cases, a non-linear activation function, such as Relu or sigmoid, will be added after each convolutional layer to introduce non-linearity into the convolutional (a linear operation) layer output. The next layer is a pooling layer; it replaces a window in the input with a single output: it could be the maximum of the input or average of the input; the output of this layer is always downsampled. After many convolutional and pooling layers,

the fully connected layers/dense layers appear. Dense layers are similar to hidden layers in traditional neural networks, every input is connected to outputs and commonly have an activation function and learn global patterns of the inputs.

There are three primary concepts in CNN architecture that should be taken into account during to implementation: receptive fields, shared weights, and pooling technique.

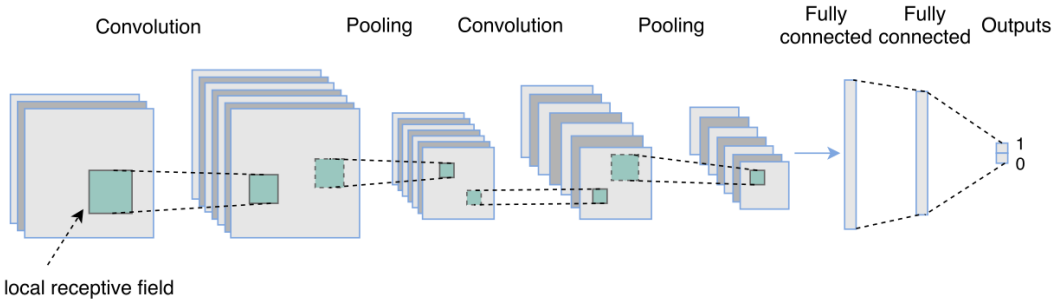

Figure 2.12. *Example of image classification with CNN. [32]*

- **Receptive field**

  Each neuron is only connected to a local region of the input volume. The spatial area of this connectivity is a hyper-parameter called the receptive field of the neuron (or filter size), and the size of this connectivity/filter/receptive field along the depth axis is equal to the depth of the input volume. The connections are local alongside the width and height but not along the depth of the input volume. Figure 2.13 depicts the input volume an image with the size [32*32*3] and a receptive field/filter size 5 *5. Each neuron in the convolutional layer will have 5*5*3 = 75 +1 bias parameter as an input volume.

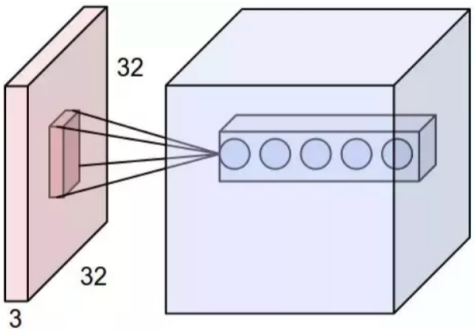

Figure 2.13. *Example of receptive field, the input volume in red, an image. Also, an example volume of neurons in the first convolutional layer. Each neuron in the convolutional layer is connected only to a local area in the input volume spatially, but to the full depth (here, all color channels). [33]*

- **Shared weights**

  Shared weights mean using the same weight vector (and same bias) to perform the convolution operation between one filter and a local receptive field. The idea

of translation invariance is behind the shared weights; a filter which can detect a pattern is position independent, and regardless of their position in the spatial field the feature can be detected. Shared weights also increase the generalization ability and learning efficiency by decreasing the number of parameters that are required in the convolutional layer.

- **Pooling technique**

  It is common to regularly insert a pooling layer after a convolutional layer in CNN in order to reduce the spatial size of the output's of the previous layer and in general the number of parameters and computation load in the model. Max-pooling is a well-known and common pooling technique. Basically, a pooling filter/window fixes a local pooling area, so there is no need to surf over the complete matrix at one time. The max-pooling of a 2×2 window is a process of traversing over the field within the pooling window and selecting the maximum element, which is depicted in figure 2.14. The output of the max-pooling layer is downsampled into a half size matrix with a 2×2 pooling window/filter of stride 2.

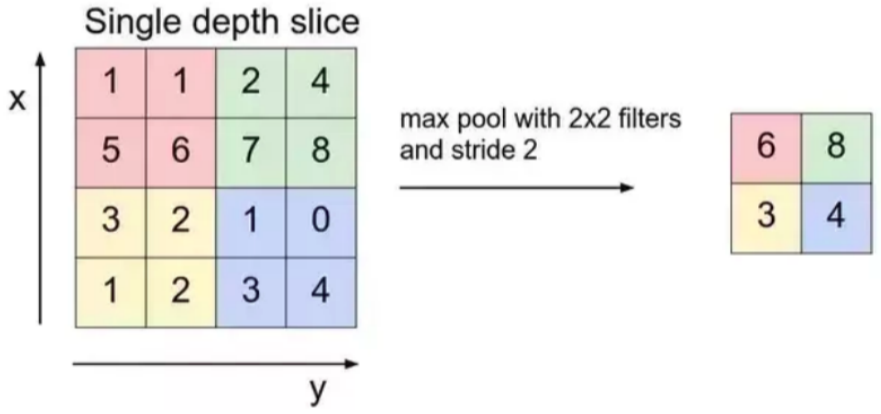

Figure 2.14. *An example of max pooling, here shown with a stride of 2 which means each max is taken over 4 numbers (small 2x2 square). [33]*

2.5.2 Recurrent neural network

Recurrent neural network (RNN) is the type of neural network that considers the temporal aspect of data by having a memory for input data points. Thus, RNNs are suitable for data points which have a temporal nature, such as time series or an entity of meaningful words/objects like sentences and genome data sequences.

A recurrent neural network processes series by iterating through the series elements and keeping a state of holding information about the relative output to everything the network has fed until this time. In fact, an RNN is a type of neural network that has an internal loop. Figure 2.15 illustrates this idea.

Program 2.2 shows a simple pseudocode implementation of RNN. The input is a 2-dimensional tensor (timesteps, features). Timesteps are the number of states in time

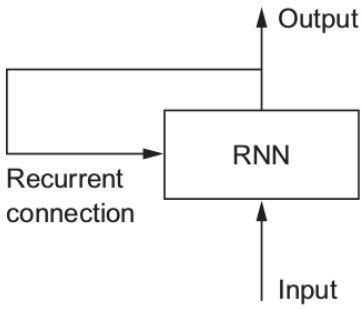

Figure 2.15. *A simple RNN unit. [31]*

```
1 state_t_init = 0 // initial state at t
2 for input_t in input_sequence:
3         output_t = f(input_t, state_t_init)
4         state_t_init = output_t
```

Program 2.2. *A simple RNN*

that the RNN wants to keep in memory. It iterates over timesteps and at each time step *t*, it combines two elements: the current state and input at the time *t*, then creates an output at *t*. Then the output of *t* is set to be the (current) status of *t+1*. At the first time step, the current status is hard coded as 0 since there is no *t-1* status.

RNNs can be grouped by how they operate on sequences, as shown in figure 2.16. In the picture, each box is a vector and the arrows denote functions. Inputs are in red, outputs in blue, and green vectors hold the RNN's current state. From left to right:

- One to one (Without RNN). For example, classification.
- One to many, like image captioning (getting an image and describing that image).
- Many to one, like sentiment analysis (what is the feeling behind the given sentence).
- Many to many, like machine translation (translating a sentence from English to another language).
- Synched many to many, like video or time series classification.

According to Andrej Karpathy [34], one of the main tasks of RNN is to classify effectively sequential data. The approach used in RNN for finding errors is a gradient-based approach which is called Backpropagation through time (BPTT) [35]. The error is defined as the difference between targets and predictions, and as shown in figure 2.17 it progresses as the time goes forward.

In practice no simple RNN is in use. The primary issue is the memory limitation for memorizing past timesteps that it has. For real cases, the sequence the model has to learn is long-dependent and impossible to learn, due to the vanishing gradient problem. Vanishing

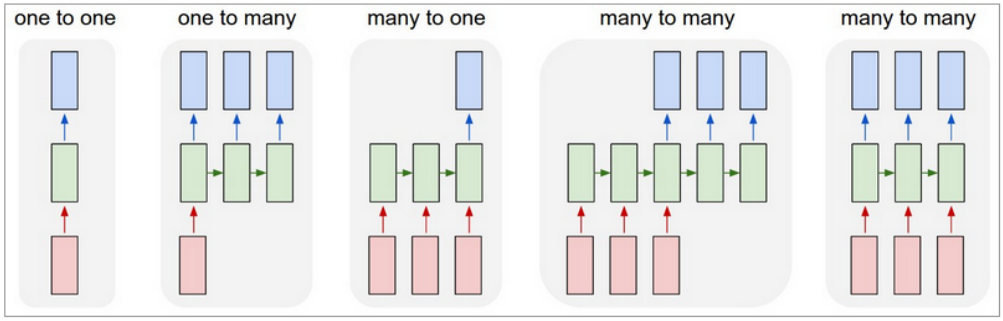

Figure 2.16. *Different examples of recurrent neural network sequential models . [34]*

gradient is a fundamental problem that leads the model to have a long training time and low accuracy score or stopping the process of training. The gradient at any point is equal to the product of the previous gradients up to that point. When the gradients of early neurons are small, between 0 and 1, then the neurons of the further layers will become lower and slow in updating the weights and eventually stop updating the values. Various solutions have been proposed for solving vanishing gradient problem, for example, initializing the weights of the network with an appropriate weight [36], applying a different activation function (RELU) [37], and evolving RNN to a new architecture like Long short-term memory, LSTM and Gated recurrent unit, GRU [38]. The following section will discuss about most common method for avoiding vanishing gradient in RNN, LSTM and GRU.

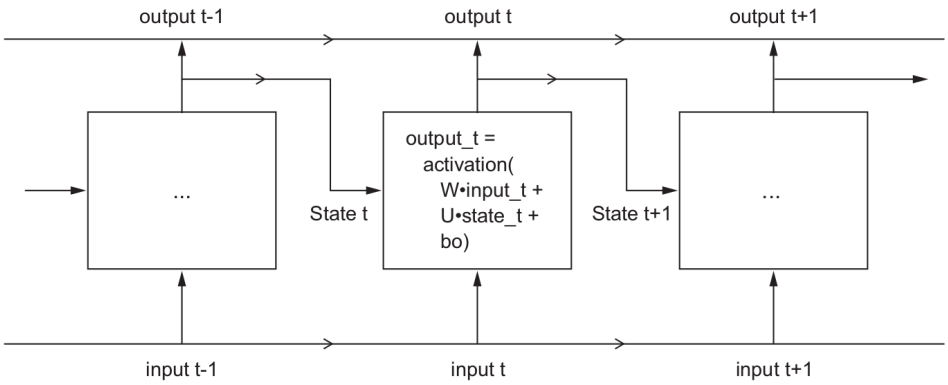

Figure 2.17. *An unrolled LSTM over time. [31]*

2.5.3 Long short-term memory and gated recurrent unit

Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are widely applied in sequential data processing, and they can also help to avoid vanishing gradient [38]. The following notations and equations were obtained by [32, 39].

Long Short-Term Memory (LSTM)

Hochreiter et al. [39] have proposed Long Short-Term Memory (LSTM) and it simply adds

a way to transfer information over many timesteps. The information is saved for later usage. Thus, the earlier data points in the sequence will not be dropped gradually due to vanishing gradient effect. Additionally, it can forget information when it is necessary. Each LSTM unit/cell includes four gates/sections:

- Input gate (i_t), regulates the input to the unit.
- Forget gate (f_t), decides when to reset/forget the LSTM unit state.
- Output gate (o_t), regulates the output to the cell.
- Cell state gate (c_t), it updates the state regarding information flow.

If data x = (x_1 ,x_2 , . . . , x_n) considered as feeding sequence input and h_t as an output function, the following equations would calculate the output signal:

$$f_t = \sigma(w_f.h_{t-1} + b_f) \tag{2.11}$$

$$i_t = \sigma(w_i.h_{t-1} + w_i.x_t + b_i) \tag{2.12}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(w_c h_{t-1} + w_c.x_t + b_c) \tag{2.13}$$

$$o_t = \sigma(w_o.h_{t-1} + w_o.x_t + b_o) \tag{2.14}$$

$$h_t = o_t \odot tanh(c_t) \tag{2.15}$$

Where the elementwise multiplication operator is denoted by $\odot$. σ indicates the logistic sigmoid activation function. W is the weight and as it is depicted, each gate in the model has a specific weight and bias. Also, the bias is denoted by b.

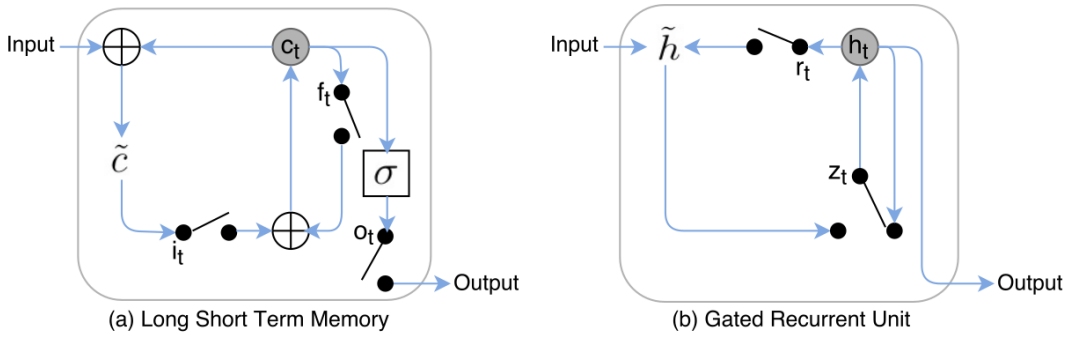

Figure 2.18. *The architecture of LSTM(a) and GRU(b). [32]*

Gated recurrent unit

GRU is an alternative for LSTM and is shown in figure 2.18. The GRU unit does not have to use a memory unit to control the flow of information like the LSTM unit [40]. GRU is slightly faster than LSTM since it has fewer parameters. At each time step the update gate z_t calculation is based on the input i_t and h_{t-1}, in order to decide about allocating memory to h_t or forgetting it. The following are the GRU equations.

$$h_t = (1 - z_t).h_{t-1} + z_t.h_t \tag{2.16}$$

$$z_t = \sigma(w_z.i_t + u_z.h_{t-1}) \tag{2.17}$$

$$h'_t = tanh(Wi_t + r_t \odot h_{t-1}) \tag{2.18}$$

$$r_t = \sigma(w_r.i_{t-1} + u_r.h_{t-1}) \tag{2.19}$$

where r and z gates are representative of the reset and update. W and U are weight matrices. Activation and candidate activation operations are denoted as h and h'.

The GRU layer reduces the number of parameters and needs fewer data to generalize, thus it is cheaper to run compared with LSTM. On the other hand, the representational power of LSTM is better than GRU [31]. The GRU and LSTM layers both use the same principle but they have differences in representational power and cost factor.

2.5.4 Recurrent dropout layer

This section discusses overfitting as a common issue in machine learning problems. Additionally, the section provides an explanation of dropout technique for both simple neural networks and RNN. Dropout is widely utilized as a solution for overfitting.

Overfitting

Overfitting is a common problem for every machine learning algorithm. The balance between optimization and generalization is a significant issue in machine learning. Optimization tunes the model in order to get the best possible performance from the training data in the train dataset, while generalization measures the performance of the test (never seen before) dataset. There is no direct control in the generalization, although separating a part of the train dataset as a hold-out validation dataset and validating the trained model could be beneficial. During the training or so-called learning phase of a model, after a specific number of iterations, generalization stops improving and the validation metrics start to fall. This is the point that the model starts to learn from the noises and memorize the specific patterns related to the training dataset and irrelevant to the test dataset. There is a set of solutions to prevent overfitting. The best thing is to have more data from different variations if there is any. Other solutions include, stopping training if the validation performance starts to downgrade, putting a limitation on what information can be stored in the model so the model can support focusing on the most outstanding patterns, which then give a better possibility to generalize properly, and adding weight regularization penalties especially dropout technique [41] which we are going to elaborate on more, both classic and RNN dropout.

Dropout

Dropout is developed by G. Hinton et al [41] and nowadays has become one of the most effective regularization ways for neural networks. Dropout operates on layers and when randomly setting to zero a number of features of the layers during the training phase, the

randomness is in choosing the features and not the amount of dropout. The amount of dropout is chosen by a fraction rate parameter which is a number between 0 to 1. Equation 2.20 illustrates the concept after and before dropout on the output of one layer with an example.

$$[0.2, 0.5, 1.3, 0.8, 1.1] >>> AfterDropout : [0, 0.5, 1.3, 0, 1.1] \tag{2.20}$$

But applying the same task on recurrent neural networks is not an easy task. It has been proved that adding and applying dropout before a recurrent layer prevents learning rather than improving regularization. Yarin Gal and Ghahramani in 2015 [42] proposed a new way of using dropout in recurrent networks. The same pattern of dropping units should be applied in every time step, and no changes should have occurred from timestep to timestep in order to aid the network to propagate its learning error through time. Moreover, a temporally fixed dropout mask should be operated onto the inner recurrent activations of the layer to regularize the representations made by the recurrent gates of layers such as GRU and LSTM.

2.5.5 Bidirectional recurrent neural network

A bidirectional RNN is a common RNN model that can perform better in many ways than a regular RNN, especially in the subfield of natural-language processing (NLP). Recurrent neural networks are fundamentally dependent on the order. Reordering or shuffling the input sequence can directly affect the result and representations of the RNN. This is the reason they perform well on problems where the ordering aspect is essential, such as NLP or time series forecasting problems.

A bidirectional RNN utilizes the order sensitivity of RNN to build a better model. It includes two normal RNN, which process the input sequence in one direction: one of them starts from the oldest (chronologically) and the other starts from the newest (anti-chronologically). In the end, both representations will be merged. Figure 2.19 depicts the working of a bidirectional RNN layer.

The idea is to obtain the patterns that may be missed in one direction processing. There is no rule that can deterministically categorize problems to the chronological process version or vice versa. In general, if the problem is forecasting the weather, chronological processing of the layer performs better than the reversed-order version: and naturally so, because the best prediction of tomorrow's weather is today. On the other hand, when the input sequence is a sentence anti-chronological processing might be more appropriate.

In machine learning, representations of the data that are different yet useful are always worth utilizing. They provide a new angle of looking at the data and enable the model to learn the features of the data that were dropped by other approaches; thus they can facilitate increasing the performance of the model. [31]

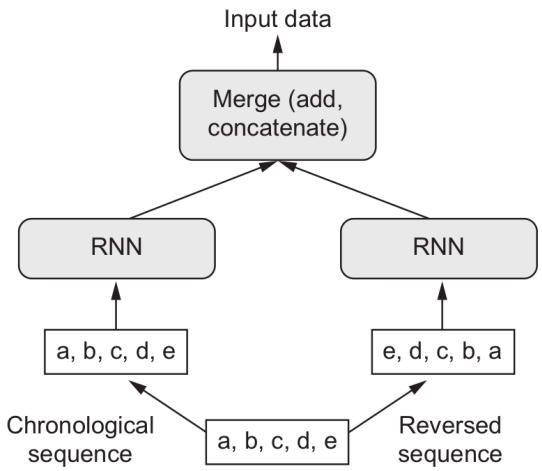

Figure 2.19. *An example of bidirectional RNN. [31]*

2.6 Evaluation metrics

Evaluation methods are used in order to determine how much the learned model fits different data sets or measures their applicability. According to N Japkowice [43], the following steps should be considered in the evaluation process:

- Determine the desired feature of the classifier that the evaluation metrics should work with.
- In order to validate, choose the right confidence estimation method.
- Perform the evaluation method and analyze the results.
- Get insights into the results with the help of domain knowledge.

In the following section,mean squared error and the evaluation metrics which are derived from the confusion matrix will be reviewed.

2.6.1 Mean squared error

Mean squared error (MSE) is the error that is computed by measuring the difference between an estimated (predicted) value of the attribute and the estimator (predictor). This is an error, thus less is better. As can be understood from formula 2.21, the error can not be negative.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2 \tag{2.21}$$

where $\hat{Y}_i$ is a vector denoting predictions of with the length of n and Y_i an array representing the grand truths (true labels).

2.6.2 Metrics based on confusion matrix

It is essential to have a set of metrics to access the performance of a proposed classifier/diagnostic analysis since we can decide about the next step in the analysis by checking the performance. One of the most common descriptive ways to measure the performance of a built model/classifier is using a table called contingency table or confusion matrix which is made by ground truths and predicted labels as is shown in figure 2.20. A confusion matrix includes 4 major elements:

		Predicted classes	
		Positive	Negative
Actual classes	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

Figure 2.20. *A confusion matrix*

- **True positive**: if the prediction and sample are both positive.
- **False positive**: if the prediction is positive for a negative sample.
- **True negative**: if the prediction and sample are both negative.
- **False negative**: if the prediction is negative for a positive sample.

Different types of errors are not equal and depending on the domain they might have different values. For example in a cancer detection system, the weight of having a false positive is much higher than having a false positive.

A confusion matrix provides a set of useful metrics for measuring the model/classifier performance in different aspects. Table 2.1 shows some of them.

Table 2.1. *Error metrics derived from confusion matrix*

Error metric	Definition
Accuracy	TP+TN /(All predictions)
Sensitivity: TP rate	TP/(TP+FN)
Specificity: TN rate	TN/(TN+FP)
Fall out: FP rate	FP/(FP+TN)

- **Precision**

  Precision is the number that tells the success rate of true positives in comparison to the total number of positives. It can be reached by dividing true positives to the number of all positives (TP + FP). This metric is useful in imbalanced classes, like the cancer detection example.

- **Recall**

  Recall shows the portion of the real positive samples that have been detected. Real positives consist of TP and FN. Thus it can be reached by dividing true positive samples into true positive and false negative samples. The recall metric is meant to measure the classifier performance regarding the detection of all the positive samples.

- **Precision-Recall curve**

  Recall and precision are built for different purposes. A recall metric measures the number of false negatives, while precision metric measures the number of false positives. The balance needed between these two metrics is reached by a precision-recall curve.

  A precision-recall curve is a tool for visualizing the tradeoff between precision and recall in each level of thresholds. An example is shown in the figure 2.21. In the beginning, a high area under the curve represents high precision (low false positive rate) and a larger area under the curve represents higher recall (low false negative). So briefly, a system with high recall and low precision has a lot of results, but most of the predicted results are incorrect. On the other hand, a system with high precision and low recall has few results, but most of them are correct. Precision-recall curves are typically used in binary classification to study the output of a classifier, like in the works of Avati et al. [44]. They represent a deep learning based algorithm to predict patients that need palliative care.

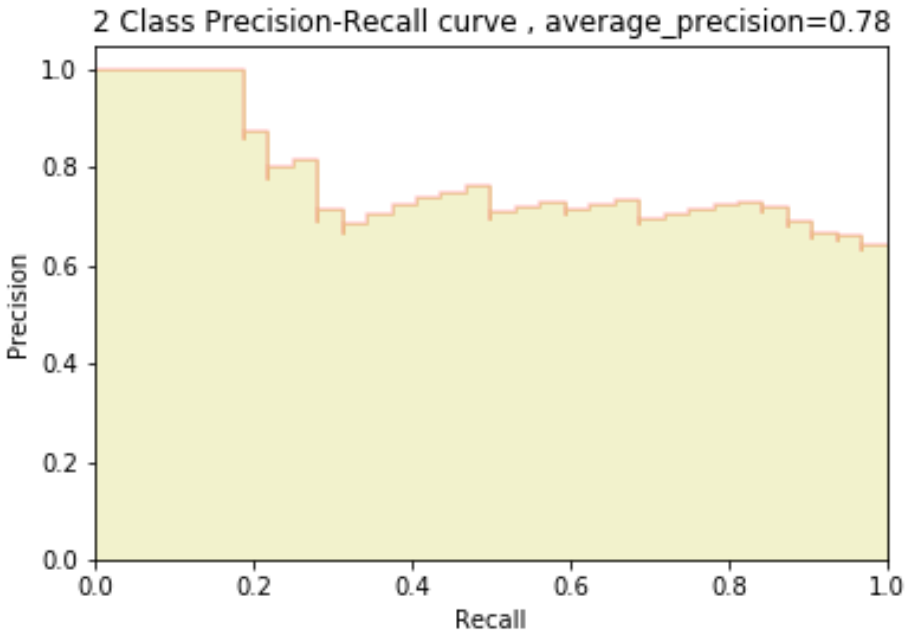

Figure 2.21. *An example of a precision-recall curve for a classifier trying to distinguish between the two first classes of the toy dataset (iris data). Inspired from [45]*

.

- **Average precision**

  Average precision is a way of representing a summarized version of the precision-recall curve. Average precision calculates the integral (average) of $p(r)$ over the interval from $r = 0$ to $r = 1$; this is the area under the precision-recall curve. This integral is in practice substituted with a weighted average of precisions at each threshold and the weights are the recall from the previous threshold level. Here is the formula:

  $$AP = \sum (r_n - r_{n-1})\, p_n \tag{2.22}$$

  where r_n and p_n are the recall and precision at the n^{th} threshold level. A pair (r_k, p_k) is named as an operating point.

- **F1 score**

  Additionally, there is a metric which is called F1 score or F measure. The F1 score can be thought of as a harmonic mean of the recall and precision, and utilized for measuring the accuracy of a classifier. Precision and recall both contribute evenly in the F1 score, while F1 score reaches its best value at 1 and worst score at 0. Here is the formula:

  $$F = \frac{2RP}{R + P} \tag{2.23}$$

  where R is recall and P is precision.

2.7 Local outlier factor

Local Outlier Factor (LOF) was proposed by Breund et al. in 2000 [46], and it is based on an idea that the density around an outlier object is considerably different from the density around its neighbors.

LOF uses the relative density of an object compared to its local neighbors instead of global distribution, and uses that as an indicator of the degree of the object being an outlier. Thus, it was the first algorithm which shows the degree of being an outlier (outlierness) for each object in the dataset. Also, before density based approaches most algorithms could not identify local outliers. It is local in the sense that only a restricted neighborhood of the object is taken into account. Figure 2.22 shows an example of LOF: note to the degree of outlierness (red circles/outlier score) for each object.

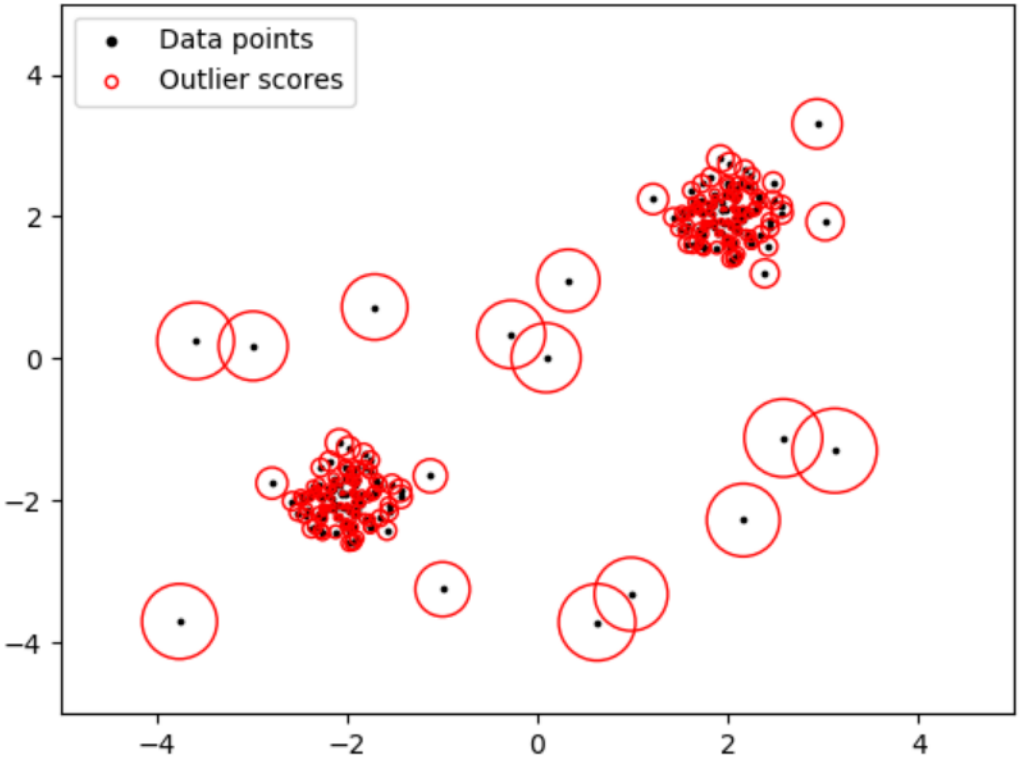

Figure 2.22. *Exmple of LOF with a random data. [45]*

The followings are some definitions of this method which will be discussed with an explanation of the algorithm.

- **K-distance of an object**

  The distance between and object P and its k^{th} nearest neighbor. K-distance(P) is defined as the nearest distance between P (the object) and its K^{th} nearest neighbor, when k is positive. The method of measuring the distance is flexible, but it affects the result.

- **Reachability distance of object O from P**

The reachability distance of an object O with respect to object P is the distance of the two objects: at least the k distance of object P, when they are not far away from each other. This could be described as:

$reach\_dist_k(O, P) = max\{K - distance(P), d(O, P)\}$

Using reachability distance exerts a powerful effect upon reducing the statistical fluctuations of the distance between object O and P.

- **Local reachability density (lrd) of an object** P

  The local reachability density, lrd of P is defined as :

$$lrd(O) := 1/\left(\sum reach - dist_k(O, P)\Big/ |N_k(O)|\right) P \in N_k(O) \tag{2.24}$$

  In parentheses is the average reachability of an object P that is based on k nearest neighbors of P. Intuitively, the local reachability density of an object P is the inverse of average reachability distance. If we take an object whose neighbors are quite far away from it. Then the distance of the object from its neighbors would be significant, meaning that we get small density, and that makes sense since all the neighbors of the object are far away from each other. On the other hand, the closer the objects are, the more the density of the object would be increased.

- **(Local) Outlier factor for an object** P

  The local outlier factor for an object (lof) P is defined as:

$$lof_k(O) := \sum (lrd(P)/lrd(O))/\left|N_k(O)\right| P \in N_k(O) \tag{2.25}$$

  where *lof* is the average of the ratio of the local reachability density of P and its nearest neighbors, the outlier factor object P obtains the degree indicating the outlierness of on object P. If it approximately equals 1 then the object is not an outlier; in the case when it is more significant than 1, it is an outlier.

2.8 Angle-based outlier detection

Angle Based Outlier Detection algorithm (ABOD) is the algorithm for detecting outliers in high dimensional data and was developed in 2008 by H.Kriegel et al. [47]. The most significant difference of ABOD from most algorithms for outlier detection in low dimensional space is that it is not based on distance. A distance approach does not perform well on high dimensional space due to the curse of dimensionality problem. The curse of dimensionality is a concept which was introduced by Bellman in 1957 [48] and refers to different issues that arise when analyzing data in high-dimensional spaces that do not occur in low-dimensional environments. For example, the calculation of outliers with methods based on distance becomes meaningless in high-dimensional space.

As the dimensionality of data rises, almost all pairs of points are equally far from each other, so any solutions which rely implicitly or explicitly on distance would not be applicable for high dimensional space. There are some solutions for dealing with this issue, including finding a more robust distance function, finding outliers in the subspace of the original features spaces, and ABOD algorithm, which will be discussed in the following.

ABOD proposes the use of directions and distances to find the outliers. Comparing the angles between pairs of vector distances points out the difference of the normal datapoint and outlier datapoint. Figure 2.23 explains the idea behind ABOD: the variance of the outlierest point will be smaller than the variance of normal data. In other words, the object is an outlier if most other objects are located in similar directions; and the object is not an outlier if many other objects are located in a different direction. The formula for a given point p and the angle between p_y an p_x where x and y are any points inside the datasets is defined as

$$ABOD(p) = VAR\left(\frac{< \vec{p}_x, \vec{p}_y >}{\|\vec{p}_x\|^2 * \|\vec{p}_y\|^2}\right) \tag{2.26}$$

where the angle-based factor is a variance over the angles between point p to all pairs, weighted by the distance of the points.

If the spectrum of observed angles for a given point is broad, that means other points in all possible directions will surround that point, therefore the point is inside the cluster and $ABOD(p)$ is high. Thus, this point is not an outlier. If the spectrum of observed angles for a point is rather small, other points will be positioned only in a specific direction and the point is outside of some sets of outputs that are grouped together. Thus, this point will be an outlier and $ABOD(p)$ is small. Where there is no specific threshold, it is defined by the domain space and sensitivity of the system.

ABOD in some terms is a great algorithm: it is parameter free, does not rely too much on distance, and gives a list of oultlierness for each point. On the other hand, there are some weak points like specifying a threshold, which could be a challenging task, and time-complexity.

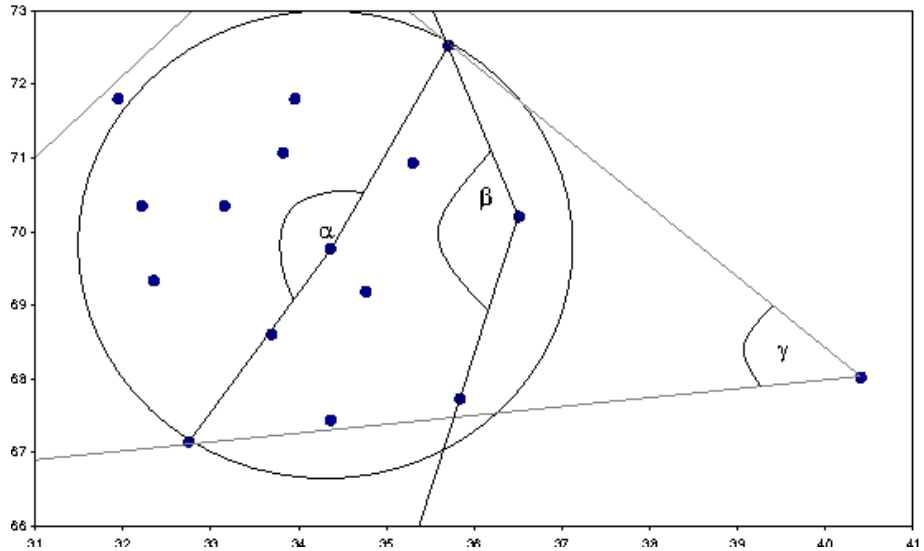

Figure 2.23. *The intuition of Angle-based outlier detection (ABOD). [47]*

3. IMPLEMENTATION

This chapter describes the approach studied in this thesis to predict and generate time series signals based on history, then compare with actual data to find out the abnormalities. Some of the methods which were described in the previous chapter were chosen for implementation, such as LSTM and LOF. The other methods are proposed for two reasons: future work and if there is an improvement in the provided data (having a variant and satisfactory set of annotated data).

3.1 Overview of solution

As shown in figure 3.1, after the preprocessing phase there are two different approaches for finding anomalies: unsupervised and supervised. The supervised approach is a neural network model which can predict the future usage of the elevator, then compares it with actual usage and reports points of the actual usage which are too different from the prediction. The unsupervised approach is a local outlier factor which determines abnormalities plus a degree of oultlierness for each item in the dataset. Following that, there is a post-processing component which includes a set of actions for reducing the number of false positives such as obtaining the intersection of these two approach results and prioritizing anomalies based on certain rules. The reason that we decided to have this model is to see what the results are and to take into use two different approaches individually and together (while taking the intersection of their results).

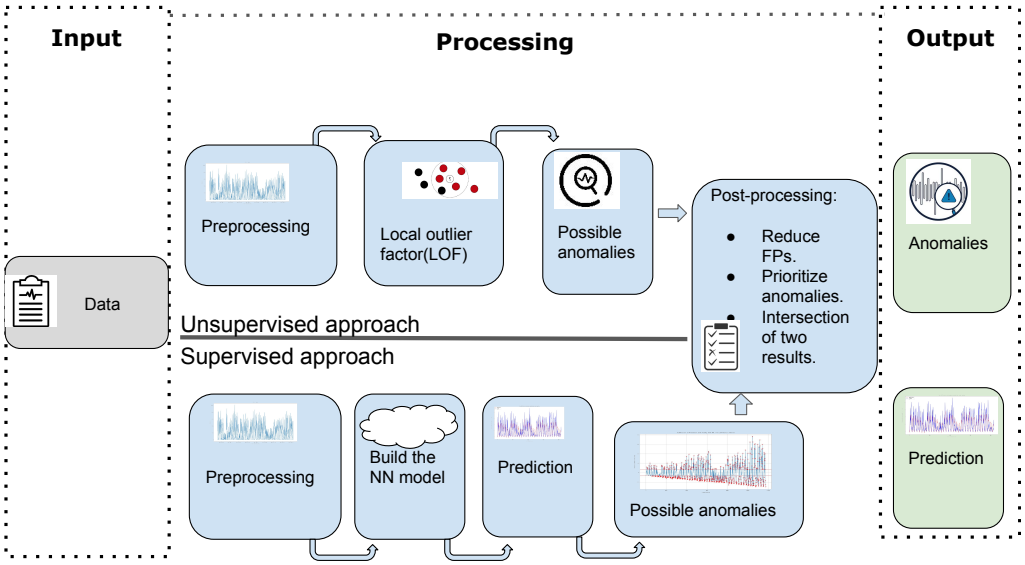

Figure 3.1. *The proposed solution for a robust model to deal with anomaly detection and forecast the usage of an elevator.*

3.2 Input data and preprocessing

The data was fetched from a cloud server to a local computer and stored as Pandas data frame object. Python pandas is a package/library which provides a flexible, fast, reliable and high-level structure designed for working with ”labeled” or ”relational” data [49].

The data frame is a 2-dimensional data structure in Pandas that is similar to a SQL database table. There are several features which are tracked when the elevator is working. Among all these, the distance metric was chosen. Distance indicates how far between two floors the elevator has traveled. The stored data could have been presented differently, but time series representation was chosen.

Arguably, most of the problems in the world are represented by time series, or if not, could be converted into times series: like Internet of things data, monitoring illnesses or industrial devices, speech recognition and measuring cooperate business metrics. The main difference between time series and non-sequential data is taking into account their internal sequential nature and driven information, such as autocorrelation, trend and seasonal variation.

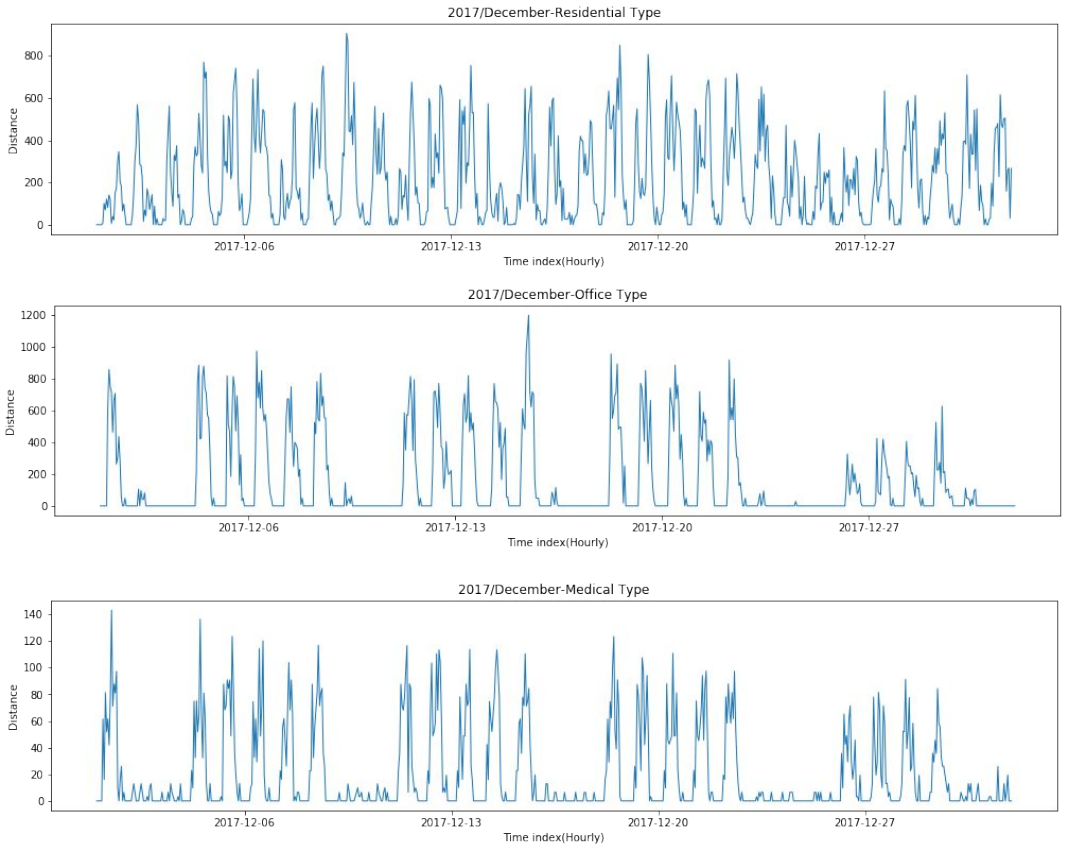

Figure 3.2. *Samples of elevator usage in September 2017 for three different building types: residential, office and medical.*

Figure 3.2 illustrates different building type distances within September 2017. With a quick visual inspection the difference in weekends can be identified. The data points out that the plot is hourly based resampled data and this will be explained in the next section.

Preprocessing

All the features are collected in the system when the event is triggered. The events are acquired during elevator movements. Thus, data were resampled with hourly frequency. Figure 3.3 shows one-week of data after resampling. Data cleansing was done to remove some elevator movements.

Then the resampled data was quantized in weekly buckets. Weeks are appropriate since there is a possibility to find patterns in different days of weeks and weekends. One of the purposes of this study is to reveal patterns in different weeks and compare them to each other. Then the train data were normalized by Scikit-learn python library [45]. Train dataset of distance values are normalized by removing the mean and scaling to unit variance. The unit and variance of the training data are also used for normalization for any new data set.

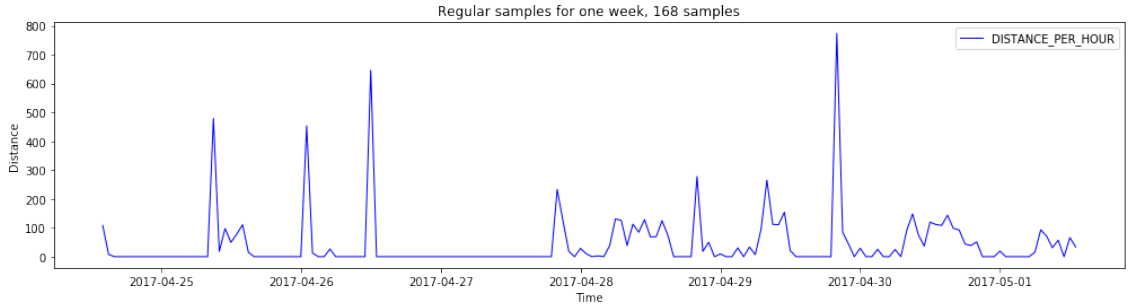

Figure 3.3. *A regular time series which is made up of hourly resampled data, all of the usages of each hour are summed up and stored (for one week/168 hour).*

4. EXPERIMENTS AND RESULTS

Our experiments have been done by Python 3 programming language and Keras which uses Tensorflow as a back-end framework. Keras is a python open source library which has an application in neural networks and was developed by François Chollet [50]. Keras can be running on top of CNTK [51], theono [52] and Tensorflow [53]. In this study, Tensorflow was chosen. Tensorflow is a symbolic math library which was developed by the Google brain team and takes advantage of dataflow programming paradigm implementation for performing the tasks. Data flow programming considers all the operators as a node in a directed graph and tries to model everything based on that graph in order to use some features of the functional programming paradigm: for example, a high capability of parallel programming [54].

The rest of this chapter will describe the details of the experiments as follows. Section 4.1 will show the features and feature extraction from time indexes. Section 4.2 will discuss about the model, the parameters, the input and training procedure and the results. Section 4.3 will discuss artificial anomaly scenarios and the result with faulty scenarios. Section 4.4 presents the result. Finally, section 4.5 discusses the implementation of the three sigma rule, then the set of activities performed for removing false positives and increasing robusticity, including LOF setup and a function which utilizes specific rules on the results.

4.1 Feature extraction

We want to have the best possible description of the data for feeding to the model, and one of the main tools for doing this is feature extraction. The feature extraction here refers to the means of building new features from an original set of data. These features should represent the main properties of the initial data set while showing that from a different aspect: this could provide a better learning opportunity for the model. Feature extraction tries to find a new way of describing a non-trivial entity for a problem/system which finally converges the model to gain a better understanding of the data.

Table 4.1. *New features derived from timestamps*

Feature	Definition
”Daylight”	(Boolean) the definition of a day: if 7:00 AM< time < 10:00 PM
”Hour”	The hour part of timestamp.
”IsWeekDay”	(Boolean) is it weekday or not?
”WeekNumber”,	The number of week.
”Weekday”	The day of the week index.

We tried to find different features of time which appear useful for using as in input for the model. Since the data is per week, we tried to find different time characteristics within the week which could be more informative and relevant, such as weekend or weekday indicator. Table 4.1 presents these features.

4.2 Training with long-short term memory model

During the experiments, different time slots were tried. Finally, we chose three months in 2017 and trained the model and tested within the same three months in 2018. Different time intervals in a different location have their own effects of trend and seasonality: for example, an elevator for a mall has a different pattern of traffic each year on Black Friday. In order to reduce the effects of having trend and seasonality on the data, we chose the same time for two different years. Table 4.2 depicts these times.

Table 4.2. *Train and test dataset dates.*

dataset	Start	End
Train(2017)	2017-03-01 00:00	2017-04-30 23:00
Test(2018)	2018-03-01 00:00	2018-04-30 23:00

After doing all the preprocessing including removing uncompleted weeks, the number of data points was 1137 for both train and test. The input data should be transformed properly to be a suitable format as an input for the LSTM model. In the LSTM algorithm, the basic idea is looking back at a certain time, then predicting a certain time in the future. In other words, if the current time is t, use $t-1$, $t-2$, $t-3$,..., $t-k$ (here k=168) to predict $t+p$ (in this case p=1); thus, the definition of the problem for the LSTM will be as follows. Within the given data go as far as the lookback parameter specified and predict the output. A lookback parameter is fixed to one week (168 hours) of observations and a target output to 1 hour.

The reason that one week was chosen is that a week is a good time slot for preserving seasonality and trend. It is also suitable for comparing different weeks in order to find any differences and changes. Therefore, the way the algorithm works for training the model is taking 168 hours and predicting the 169th hour, then going to the next step by shifting all the looking back function one index forward, as it depicted in figure 4.1.

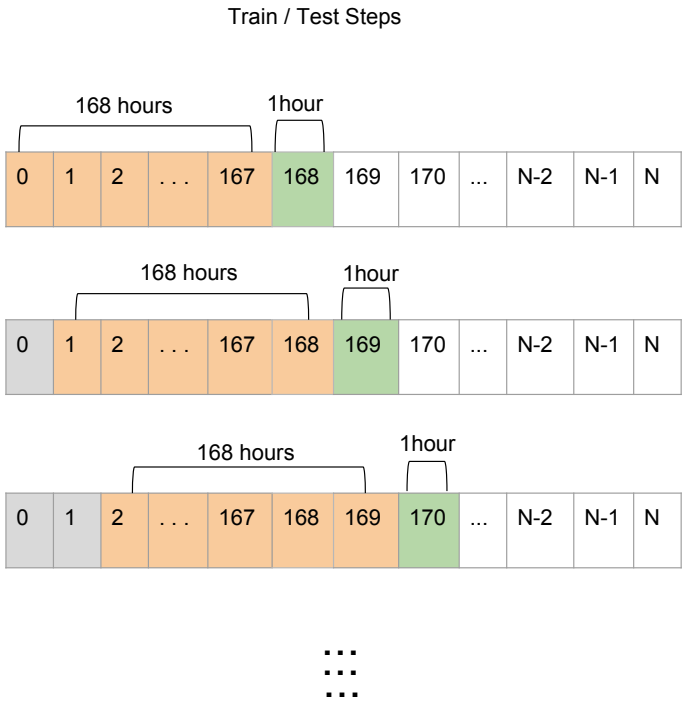

Figure 4.1. *Train and test steps in the data.*

Figure 4.2 represents the architecture of the model. The model takes a 3-dimensional array as an input. The input consists of the number of samples (1176), timestamps (168), and the number of features (6). Then the model is followed by 50 units in the LSTM layer. After that, there is a dropout layer which would randomly select 20 percent of neurons to ignore during training. Then again an LSTM layer, but a different number of units, 100 units, and after that a dropped out layer with the 20 percent of ignored neurons in training. This is followed by a dense layer which has a linear activation function. Since the output here is a number, the activation function is a linear regression.

Figure 4.2. *The final LSTM network architecture which is used as a model.*

The error function which is used for find out the amount of loss is minimum square error MSE, which was explained in the chapter. Also, the model used ”RMSprop” as an optimizer.

We also take 20 percent of data for the validation part. Figure 4.3 illustrates the rate of loss over the number of epochs.

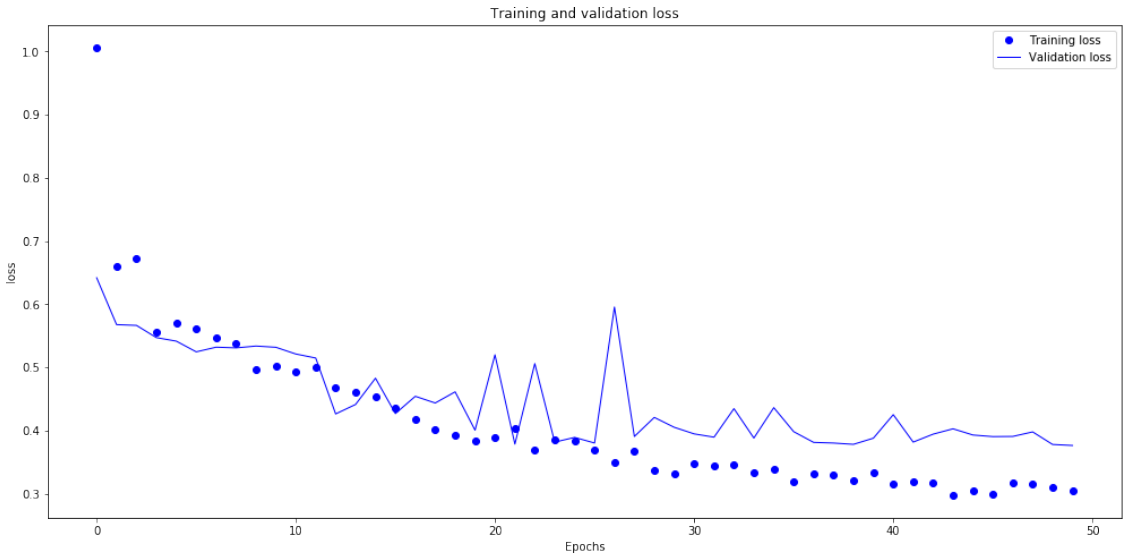

Figure 4.3. *Training and validation loss over the number of epochs.*

Figure 4.4 shows a visual representation of how close the prediction and real measured values are.

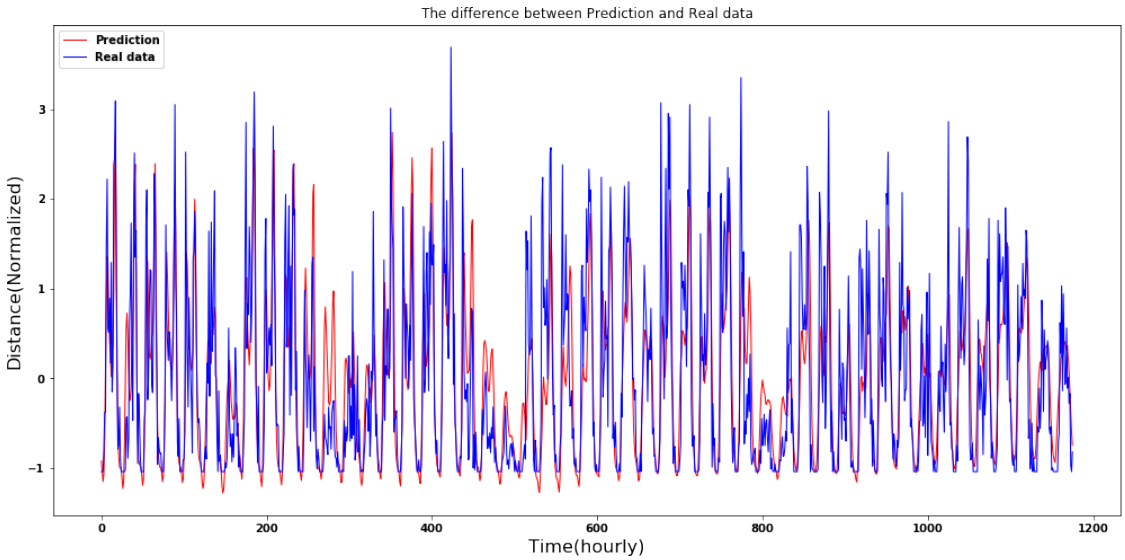

Figure 4.4. *The difference between prediction and real data (test dataset).*

4.3 Artificial anomaly scenarios

The data which has been used in this study suffers from a lack of annotated anomaly or faults. Therefore, we could not be sure whether the anomaly which is detected is true or not. The absence of usage in the elevator is considered as a severe anomaly, and the simulation of such an event needed to be tested in the model. Moreover, finding an increasing trend in usage pattern would be interesting. Thus, similar situations were created and embedded in

the test data. Two adjusted fault data are:

- **Scenario 1:** The elevator stopped working for two consecutive days. Scenario one may not be considered as an anomaly during the holidays, but since it is compared with a different year, the same date, it does make sense. This situation is simulated by randomly assigning 48 hours of test data with zero.
- **Scenario 2:** The usage of an elevator receiving an increasing linear trend over time. The situation is simulated by building an increasing linear trend and multiplying it by test data pointwise. The increasing linear trend is built by plotting a line between two points with the value 1 and 2, the first plot in figure 4.5.

For simplicity, the above fault scenarios will be mentioned by their number in this thesis. Figure 4.5 shows the linear increasing trend that fitted the original data as a linear trend and both original and trended. Notice the different scaling of the y-axis in the first plot.

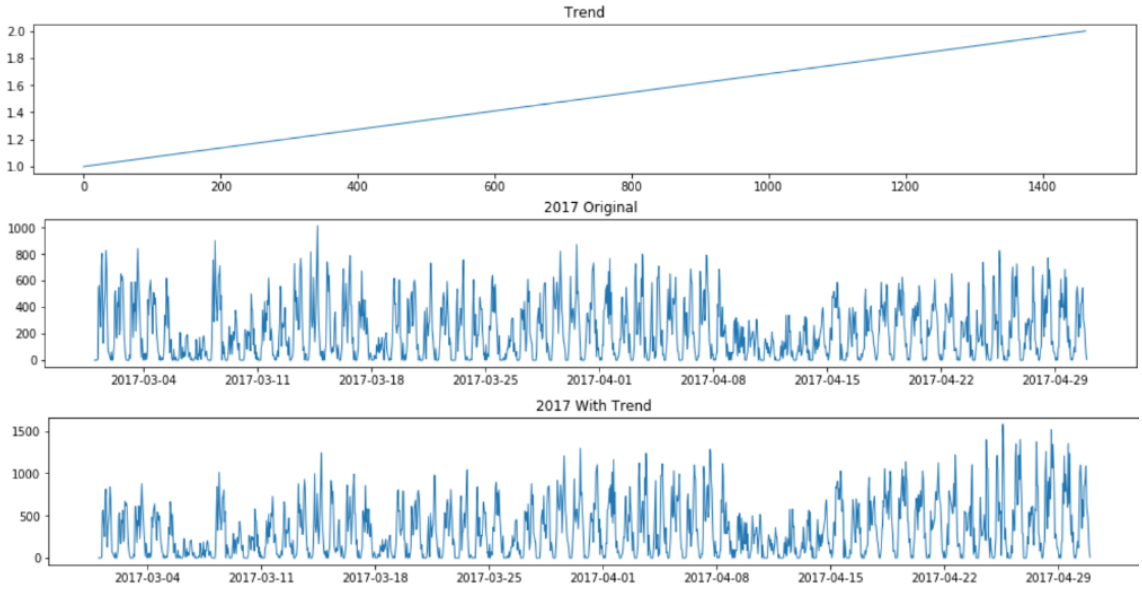

Figure 4.5. *Added linear trend, original and trended data, which are related to simulation of fault scenario 2.*

4.4 Result

Since the distribution of data was close to normal distribution, we could apply three sigma rule for finding anomalies. We used the standard deviation of the difference of prediction and real data for applying three sigma rule on the error, the difference of the prediction and what really happened/grand truth.

Three sigma rule (also called the **empirical** or **68-95-99.7 rule**)

Three sigma rule states that for a normal distribution, almost all of the data will happen within three sigmas (standard deviations) of the mean. The empirical rule also states that [55]:

- 68% of data happen within the first standard deviation from the mean.
- 95% of data happen within two standard deviations.
- 99.7% of data happen within three standard deviations.

Figure 4.6 depicts the three sigma rule and the percentage of data which are normally distributed inside the intervals.

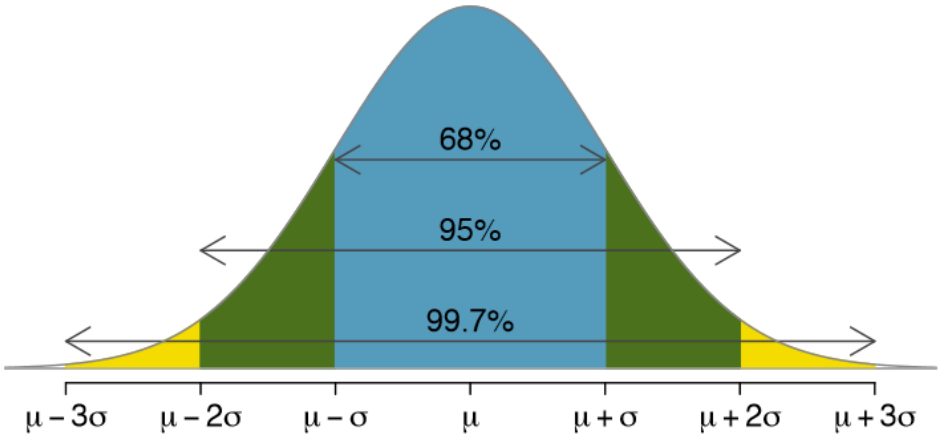

Figure 4.6. *Three sigma rules on a normal distribution. [56]*

The next step is to find a point-wise distance for every predicted point and plot that distance. The distance is found by plotting the subtraction of real data (which really happened) from the prediction (what is predicted to have happened). Then, three σ rule is performed on the subtraction and each data point considered above or below the threshold is an anomaly. Figures 4.7 and 4.8 indicate three sigma intervals with red lines.

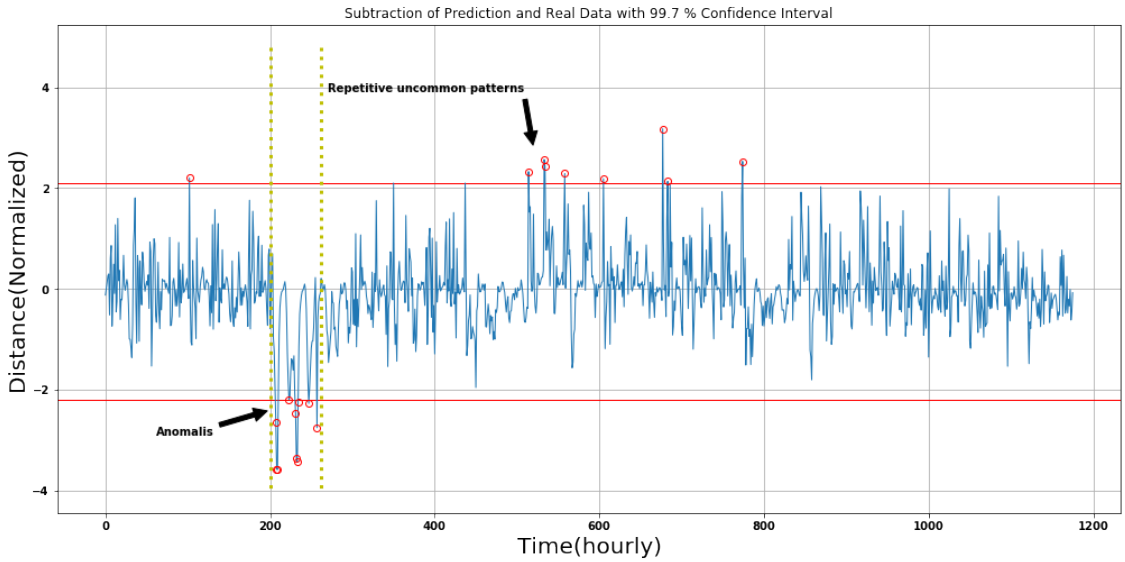

Figure 4.7. *Testing the model for fault scenario 1 and detecting the absence of usage and weekend patterns.*

Figure 4.7 shows the result when the model is tested with fault scenario 1 and figure 4.8 depicts the result when the model is tested with fault scenario 2.

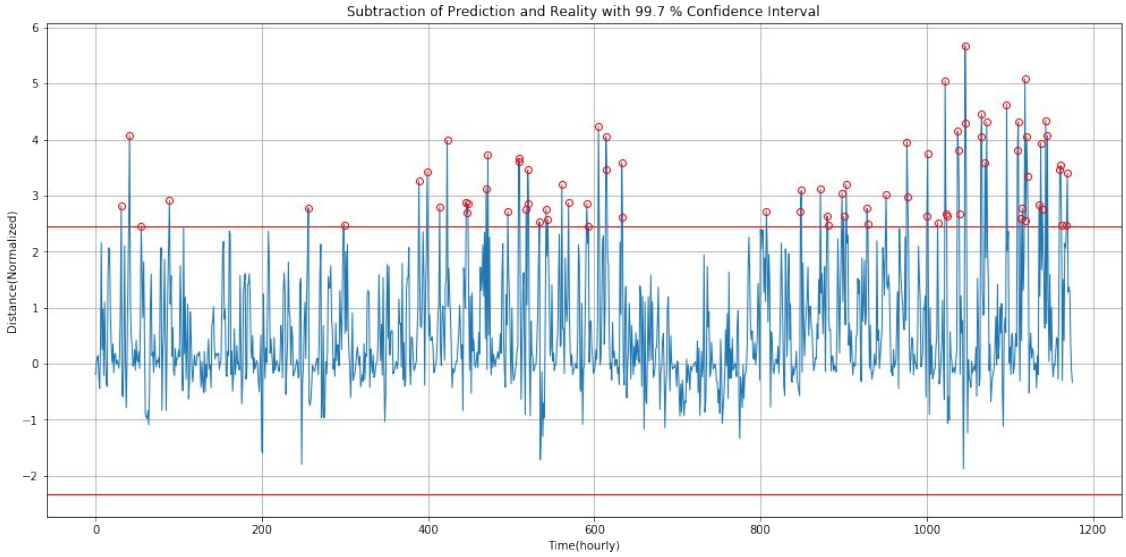

Figure 4.8. *Testing the model for fault scenario 2 and detecting the increasing trend.*

4.5 Post-processing actions

Post-processing actions are a set of actions which are built in order to reduce the number of false positives and produce a more reliable model. All the points were detected as anomaly using LSTM approach which is not necessarily true. Also, if they are correct, they do not have the same importance. It was proved by Shipmon et al. [57], that by defining a set of rules on the data, the number of false positives can be decreased. So rules depending on a domain could be defined variously.

The rule function created here categorizes outlier data by their index, therefore those anomalies which have a consecutive index have a higher possibility of being an outlier, like figure 4.7. Also, anomalies can be prioritized in such a way that the groups which are more crowded are more severe.

In addition, the result of LOF when the number of neighbors was set to two was as expected in scenarios one and two. We used the default settings of LOF implementation in Scikit-learn. So the intersection of these post-processing activities verified the results which were reached by the supervised approach.

5. CONCLUSION

Our study applies two different methodologies for detecting anomalies: one is based on LSTM and predicts future usage, the other is unsupervised and identifies anomalies. The idea is to announce the intersection of these two end-to-end solutions as an anomaly in order to make the system more robust.

After preprocessing the data, the LSTM model predicts with 37% loss. Then a common approach is used to indicate the index when the observed dataset value falls outside of a 99.7 % confidence in intervals of the prediction. The second method, which is unsupervised and called LOF, identifies the outliers based on the distance they have with their neighbors. We used March and April 2017 as a train dataset and the same months in 2018 as a test dataset. Different building types were used, but the main one in the experiments was a residential building. After testing with 2018 dataset a repeated different pattern was observed on weekends. In order to find the casuality of this difference, the question raised was, is there any correlation between weather condition and people's tendency to go out? We could not answer this question since we do not have the needed data for investigation.

In order to test the system with multiple unusual behavior examples, two anomaly scenarios were artificially created: lack of usage and an increasing trend for elevator usage. Both were detected successfully by the system when tested.

The fact is that the dataset is imbalanced and there is no particular label for the unusual behavior of an elevator; thus, the high number of false positives should be taken into account carefully. We set one of our goals as reducing the false positive(FP) rate to avoid unnecessary on-call services while making sure that those real outages will be detected. We were inspired by [57] and have a post-processing function to imply a set of rules for data and to remove the possible false positives.

We can draw a conclusion from this study that it is feasible to predict a time series for anomaly detection. By combining this with the unsupervised method and a set of rules, we can achieve a more reliable result for different parties such as maintenance team and customer. Providing the solutions for optimization and improvement of the elevator system is beyond the scope of this thesis.

This master's thesis was part of the OPENS-project funded by a public funding agency for research called Business Finland and industrial partners. The university partners are the Laboratory of Automation and Hydraulic Engineering and Department of Computer Science of Åbo Akademi University. The aim of OPENS is to develop methodologies to enable predictive maintenance solutions. The topic and orientation of this study were set according to the OPENS objectives.

Future work

During this study, some challenges were found that potentially could be considered as future work. First, we have to utilize the information about special events like holidays from the calendar. For example, if there is a national holiday, lack of usage of an office elevator is expected. Also, measuring the uncertainty level of occurring unseen/new patterns in the test data would be a firm step to increase the model reliability [58].

In addition, working more on LOF and investing in ABOD and the way of distance measurement by approaching the data other than just time series could be of interest.

Moreover, An interesting direction for future work is fine-tuning the LSTM model and trying newly published architectures for time series in order to increase the performance. In addition, there is room for optimization of the model performance if we provide the model with more data. Even investing in time series augmentation techniques for having more data could be worth trying.

REFERENCES

[1] "Is predictive maintenance effective?." EnVibe of Houston, Tx. Available:https://envibe.com/about-us/predictive-maintenance-is-cost-effective/ "[Online; accessed on 06-08-2018]".

[2] M. Selek, H. Terzioglu, and F. A. Kazan, "The design of training elevators for effective learning," in *Information Science and Control Engineering (ICISCE), 2015 2nd International Conference on*, pp. 956–959, IEEE, 2015.

[3] "thyssenkrupp elevator ag." Elevator AG. Available:https://multi.thyssenkrupp-elevator.com/en/ "[Online; accessed on 23-08-2018]".

[4] K. M. M. J. B. K. J. H. Tomi R. Krogerus, Michael Woldegebriel, "Bayesian framework for health monitoring in elevator systems." An unpublished work, 8 2019.

[5] "Elevators and escalators market by type global forecast to 2021." marketsandmarkets.com. Available:https://www.marketsandmarkets.com/Market-Reports/elevator-escalator-market-221792102.html "[Online; accessed on 06-08-2018]".

[6] I. Skog, I. Karagiannis, A. B. Bergsten, J. Härdén, L. Gustafsson, and P. Händel, "A smart sensor node for the internet-of-elevators—non-invasive condition and fault monitoring," *IEEE Sensors Journal*, vol. 17, no. 16, pp. 5198–5208, 2017.

[7] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018.

[8] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2017.

[9] L. Portnoy, *Intrusion detection with unlabeled data using clustering*. PhD thesis, Columbia University, 2000.

[10] R. J. Bolton and D. J. Hand, "Statistical fraud detection: A review," *Statistical science*, pp. 235–249, 2002.

[11] K. Zhang, M. Hutter, and H. Jin, "A new local distance-based outlier detection approach for scattered real-world data," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 813–822, Springer, 2009.

[12] S. P. Verma and E. Santoyo, "New improved equations for nak, nali and sio2 geothermometers by outlier detection and rejection," *Journal of Volcanology and Geothermal Research*, vol. 79, no. 1-2, pp. 9–23, 1997.

[13] Y. Zhang, N. Meratnia, and P. J. Havinga, "Outlier detection techniques for wireless sensor networks: A survey.," *IEEE Communications Surveys and Tutorials*, vol. 12, no. 2, pp. 159–170, 2010.

[14] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: Survey. a modified version of this technical report will appear," *ACM Computing Survey, ACM*, 2009.

[15] C. C. Aggarwal, "Outlier analysis," in *Data mining*, pp. 237–263, Springer, 2015.

[16] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: experimental comparison of representations and distance measures," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1542–1552, 2008.

[17] P. Ranacher and K. Tzavella, "How to compare movement? a review of physical movement similarity measures in geographic information science and beyond," *Cartography and geographic information science*, vol. 41, no. 3, pp. 286–307, 2014.

[18] T. M. Mitchell *et al.*, "Machine learning. wcb," 1997.

[19] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[20] X. Zhu, "Semi-supervised learning literature survey," *Computer Science, University of Wisconsin-Madison*, vol. 2, no. 3, p. 4, 2006.

[21] S. Amidi, "Cs 229 - machine learning." Stanford University. Available:https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning "[Online; accessed on 10-09-2018]".

[22] V. Vryniotis, "Tuning the learning rate in gradient descent." Available:http://blog.datumbox.com/tuning-the-learning-rate-in-gradient-descent/ "[Online; accessed on 08-10-2018]".

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] J. Bayer, C. Osendorfer, S. Diot-Girard, T. Rueckstiess, and S. Urban, "climin-a pythonic framework for gradient-based function optimization," *TUM, Tech. Rep.*, 2015.

[25] R. Gandhi, "A look at gradient descent and rmsprop optimizers." Statistics How To. Available:https://towardsdatascience.com/a-look-at-gradient-descent-and-rmsprop-optimizers-f77d483ef08b "[Online; accessed on 09-10-2018]".

[26] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *Nature communications*, vol. 5, p. 4308, 2014.

[27] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[29] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on audio, speech, and language processing*, vol. 22, no. 10, pp. 1533–1545, 2014.

[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[31] F. Chollet, *Deep learning with python*. Manning Publications Co., 2017.

[32] L. ZHU, "Teaching development project: Gene expression prediction with deep learning," 2017.

[33] "Convolutional neural networks (cnns / convnets)." Stanford CS231n: Convolutional Neural Networks for Visual Recognition. Available:http://cs231n.github.io/convolutional-networks/ "[Online; accessed on 08-10-2018]".

[34] A. Karpathy, "The unreasonable effectiveness of recurrent neural networks." Andrej Karpathy blog. Available:http://karpathy.github.io/2015/05/21/rnn-effectiveness/ "[Online; accessed on 01-10-2018]".

[35] M. C. Mozer, "A focused backpropagation algorithm for temporal," *Backpropagation: Theory, architectures, and applications*, p. 137, 1995.

[36] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv preprint arXiv:1504.00941*, 2015.

[37] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.

[38] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[39] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[40] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Gated feedback recurrent neural networks," in *International Conference on Machine Learning*, pp. 2067–2075, 2015.

[41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[42] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, pp. 1050–1059, 2016.

[43] N. Japkowicz, "Why question machine learning evaluation methods," in *AAAI Workshop on Evaluation Methods for Machine Learning*, pp. 6–11, 2006.

[44] A. Avati, K. Jung, S. Harman, L. Downing, A. Ng, and N. H. Shah, "Improving palliative care with deep learning," in *Bioinformatics and Biomedicine (BIBM), 2017 IEEE International Conference on*, pp. 311–316, IEEE, 2017.

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[46] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *ACM sigmod record*, vol. 29, pp. 93–104, ACM, 2000.

[47] H.-P. Kriegel, A. Zimek, *et al.*, "Angle-based outlier detection in high-dimensional data," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 444–452, ACM, 2008.

[48] M. Brown, K. Bossley, D. Mills, and C. Harris, "High dimensional neurofuzzy systems: overcoming the curse of dimensionality," in *Fuzzy Systems, 1995. International Joint Conference of the Fourth IEEE International Conference on Fuzzy Systems and The Second International Fuzzy Engineering Symposium., Proceedings of 1995 IEEE Int*, vol. 4, pp. 2139–2146, IEEE, 1995.

[49] W. McKinney, *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. " O'Reilly Media, Inc.", 2012.

[50] F. Chollet *et al.*, "Keras." https://keras.io, 2015.

[51] F. Seide and A. Agarwal, "Cntk: Microsoft's open-source deep-learning toolkit," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2135–2135, ACM, 2016.

[52] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A cpu and gpu math compiler in python," in *Proc. 9th Python in Science Conf*, vol. 1, 2010.

[53] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: a system for large-scale machine learning.," in *OSDI*, vol. 16, pp. 265–283, 2016.

[54] T. B. Sousa, "Dataflow programming concept, languages and applications," in *Doctoral Symposium on Informatics Engineering*, vol. 130, 2012.

[55] Statistics How To! Available:http://www.statisticshowto.com/empirical-rule-2/ "[Online; accessed on 05-09-2018]".

[56] C. Pascual, "A look at gradient descent and rmsprop optimizers." dataquest.io. Available:https://www.dataquest.io/blog/basic-statistics-in-python-probability/ "[Online; accessed on 09-10-2018]".

[57] D. T. Shipmon, J. M. Gurevitch, P. M. Piselli, and S. T. Edwards, "Time series anomaly detection; detection of anomalous drops with limited features and sparse examples in noisy highly periodic data," *arXiv preprint arXiv:1708.03665*, 2017.

[58] L. Zhu and N. Laptev, "Deep and confident prediction for time series at uber," in *Data Mining Workshops (ICDMW), 2017 IEEE International Conference on*, pp. 103–110, IEEE, 2017.