Chair of Automation

Master's Thesis

Machine Learning in the Context of Time Series

Stefan Herdy, BSc

November 2020

Statutory declaration

I declare in lieu of oath, that I wrote this thesis and performed the associated research myself, using only literature cited in this volume.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Ich erkläre, dass ich die Richtlinien des Senats der Montanuniversität Leoben zu "Gutewissenschaftliche Praxis" gelesen, verstanden und befolgt habe.

Weiters erkläre ich, dass die elektronische und gedruckte Version der eingereichtenwissenschaftlichen Abschlussarbeit formal und inhaltlich identisch sind

Leoben, November 26, 2020

Stefan Herdy
Matr.Nr.: m01610562

Acknowledgements

First of all, I would like to thank my supervisor Professor Paul O'Leary for the great support during the master thesis and during the entire studies. Despite the special situation this year I had a great support from him due to his great engagement and his early reactions to the changing situations. He has a strong motivation to encourage us students to put much effort into gaining an understanding of different technical problems and also outside of my studies I was able to learn a lot from Paul for my future path, for which I am very grateful.

The Chair of Automation as a whole and the way of working and living together there is a great example of productive and pleasant cooperation, which motivates every one to give their best and enjoy their work.

I also want to thank my family for the mental and also financial support during my entire educational path.

The mental help of my friends was also very important during the entire studies and I'm very grateful for that.

Finally I want to express my gratitude for the help of the whole university staff and my student colleagues for the beautiful time in Leoben.

Abstract

The major goals of modern applied technology is to optimize processes, make them more cost and resource efficient, increase the security of a process and many more. In order to achieve this, it is necessary to understand the underlying process. However, one can only gain understanding of something by collecting and evaluating data and information. Nowadays, many technological systems are equipped with sensors to monitor their condition and to be able to make possible statements about a future condition. Data that arises from the monitoring of such systems as a function of time is called time series data. The analysis of time series data is a growing field and, especially with the help of the field of machine learning, which has become popular in recent years, some new methods for time series analysis can potentially be developed.

If data is to be analysed and knowledge is to be gained from it, the data should represent reality as well as possible. If a few data points differ significantly from the collective, this usually indicates an abnormal process or a faulty measurement. The detection of such outliers has two main reasons. On the one hand, outliers indicate an error or an unexpected event that often needs to be recognized, and on the other hand, cleaning up a data record, i.e. deleting outliers, can improve the results of a data analysis. An outlier is a data sample that contains abnormal trends in it. These can arise due to errors in the process, errors in the measurement or also through one-off events that do not negatively influence the process. Such anomalies can often be quickly identified by humans after a good visualization. Usually very large amounts of data have to be evaluated, which would cost much time when its done manually. Therefore, there is a desire for automatic detection of these anomalies in the data. There is a possibility to detect these outliers with the help of machine learning algorithms. In the last few years, machine learning has become very popular and became a promised solution to a variety of modern problems, such an automatic detection of anomalies in time-dependent data is a non-trivial task and requires the use of suitable algorithms that are able to learn and recognize typical time profiles of the data. The development of such machine learning models and the evaluation of the developed models for time series analysis in general are the major parts of this thesis. The first chapters of this thesis are an introduction to the basic concepts of machine learning and are essential to understand the following machine learning applications on the time series data. After this introduction the implemented applications are explained. The main

part of this thesis is the introduction of a new machine learning method that can be used in time series analysis. The goal of this thesis is to evaluate the limits of machine learning and to learn how machine learning can be applied to time series data in a reasonable way. The last chapter contains an overview of the results and explains what we can learn from this work for future machine learning applications. So the main three chapters of this thesis are an introduction to to machine learning and neuronal nets, the development of new machine learning methods for time series analysis and finally an interpretation of the results.

Kurzfassung

Ein Hauptziel der modernen Technik besteht darin, Prozesse zu optimieren, sie kosten- und ressourceneffizienter zu gestalten, die Sicherheit eines Prozesses zu erhöhen und vieles mehr. Um dies zu erreichen, ist es notwendig, den zugrunde liegenden Prozess zu verstehen. Man kann jedoch nur dann ein Verständnis über komplexe Prozesse erlangen, wenn man Daten und Informationen sammelt und auswertet. Heutzutage sind viele technologische Systeme mit Sensoren ausgestattet, um ihren Zustand zu überwachen und mögliche Aussagen über einen zukünftigen Zustand treffen zu können. Daten, die sich aus der Überwachung solcher Systeme als Funktion der Zeit ergeben, werden als Zeitreihendaten bezeichnet. Die Analyse von Zeitreihendaten ist ein wachsendes Feld, und insbesondere mit Hilfe des in den letzten Jahren populär gewordenen Bereichs des maschinellen Lernens können möglicherweise einige neue Methoden zur Zeitreihenanalyse entwickelt werden. Wenn Daten analysiert und daraus Wissen gewonnen werden soll, sollten die Daten die Realität so gut wie möglich darstellen. Wenn sich einige Datenpunkte erheblich vom Kollektiv unterscheiden, deutet dies normalerweise auf einen abnormalen Prozess oder eine fehlerhafte Messung hin. Die Erkennung solcher Ausreißer hat zwei Hauptgründe. Einerseits weisen Ausreißer auf einen Fehler oder ein unerwartetes Ereignis hin, das häufig erkannt werden muss, und andererseits kann das Bereinigen eines Datensatzes, also das Löschen von Ausreißern, die Ergebnisse einer Datenanalyse verbessern. Ein Ausreißer ist eine Datenprobe, die abnormale Trends enthält. Diese können durch Prozessfehler, Messfehler oder auch durch einmalige Ereignisse entstehen, die den Prozess nicht negativ beeinflussen. Solche Anomalien können vom Menschen nach einer guten Visualisierung oft schnell erkannt werden. Normalerweise müssen sehr große Datenmengen ausgewertet werden, was bei manueller Ausführung viel Zeit kosten würde. Daher besteht der Wunsch nach einer automatischen Erkennung dieser Anomalien in den Daten. Es besteht die Möglichkeit, diese Ausreißer mithilfe von Algorithmen für maschinelles Lernen zu erkennen. In den letzten Jahren ist maschinelles Lernen sehr populär geworden und zu einer versprochenen Lösung für eine Vielzahl moderner Probleme geworden. Eine solche automatische Erkennung von Anomalien in zeitabhängigen Daten ist eine nicht triviale Aufgabe und erfordert die Verwendung geeigneter Algorithmen die in der Lage sein sollen, typische Zeitprofile der Daten zu erlernen und zu erkennen. Die Entwicklung solcher Modelle für maschinelles Lernen und die Bewertung der entwickel-

ten Modelle für die Zeitreihenanalyse im Allgemeinen sind die Hauptbestandteile dieser Arbeit. Die ersten Kapitel dieser Arbeit sind eine Einführung in die Grundkonzepte des maschinellen Lernens und wichtig, um die folgenden Anwendungen des maschinellen Lernens auf Zeitreihendaten zu verstehen. Nach dieser Einführung werden die implementierten Anwendungen erläutert. Der Hauptteil dieser Arbeit ist die Einführung einer neuen Methode des maschinellen Lernens, die in der Zeitreihenanalyse verwendet werden kann. Ziel dieser Arbeit ist es, die Grenzen des maschinellen Lernens zu bewerten und zu lernen, wie maschinelles Lernen auf vernünftige Weise auf Zeitreihendaten angewendet werden kann. Das letzte Kapitel enthält einen Überblick über die Ergebnisse und erklärt, was wir aus dieser Arbeit für zukünftige Anwendungen des maschinellen Lernens lernen können. Die drei Hauptkapitel dieser Arbeit sind daher eine Einführung in maschinelles Lernen und neuronale Netze, die Entwicklung neuer Methoden des maschinellen Lernens für die Zeitreihenanalyse und schließlich eine Interpretation der Ergebnisse.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

The major goals of modern applied technology is to optimize processes, make them more cost and resource efficient, increase the security of a process and many more. In order to achieve this, it is necessary to understand the underlying process. However, one can only gain understanding of something by collecting and evaluating data and information. Nowadays, many technological systems are equipped with sensors to monitor their condition and to be able to make possible statements about a future condition. Data that arises from the monitoring of such systems as a function of time is called time series data. The analysis of time series data is a growing field and, especially with the help of the field of machine learning, which has become popular in recent years, some new methods for time series analysis can potentially be developed.

If data is to be analysed and knowledge is to be gained from it, the data should represent reality as well as possible. If a few data points differ significantly from the collective, this usually indicates an abnormal process or a faulty measurement. The detection of such outliers has two main reasons. On the one hand, outliers indicate an error or an unexpected event that often needs to be recognized, and on the other hand, cleaning up a data record, i.e. deleting outliers, can improve the results of a data analysis. An outlier is a data sample that contains abnormal trends in it. These can arise due to errors in the process, errors in the measurement or also through one-off events that do not negatively influence the process. Such anomalies can often be quickly identified by humans after a good visualization. Usually very large amounts of data have to be evaluated, which would cost much time when its done manually. Therefore, there is a desire for automatic detection of these anomalies in the data. There is a possibility to detect these outliers automatically with the help of machine learning algorithms. In the last few years, machine learning has become very popular and became a promised solution to a variety of modern problems. Such an automatic detection of anomalies in time-dependent data is a non-trivial task and requires the use of suitable algorithms that are able to learn and recognize typical time profiles of the data. The development of such machine learning models and the evaluation of the developed models for time series analysis in general are the major parts of this thesis.

To be able to evaluate the developed models it is necessary to evaluate the models on real data. The data for the evaluation is provided by the Keller Grundbau GmbH. This company has specialized in a process in which the soil is penetrated at regular intervals with high force and vibration to increase its load bearing capacity. A drill head is driven into the ground in stages with great force and vibrations in the range around 100 Hz. When the maximum depth is reached, the drill head is gradually pulled out of the hole. When pulling out, a cavity is created in the hole, which is filled with a filler. After that the drill is moved into the hole with great force and high vibrations in order to compress the filler. After the drill is pulled a little bit out of the hole, the filling and compacting starts again. This process is repeated until the hole is completely compressed. During this compression, there are usually smaller breaks in which the filler has to be filled into the machine. This process results in a total compaction of the subsoil and increases the load-bearing capacity for buildings on it. During the entire process, time-dependent data (called time series data) such as drilling depth, vibration frequency etc. are measured and saved. At a total of four locations, data was collected for around 2500 such compaction points.

The first chapters of this thesis are an introduction to the basic concepts of machine learning and are essential to understand the following machine learning applications on the time series data. After this introduction the implemented applications are explained. The main part of this thesis is the introduction of a new machine learning method that can be used in time series analysis. The goal of this thesis is to evaluate the limits of machine learning and to learn how machine learning can be applied to time series data in a reasonable way. The last chapter contains an overview of the results and explains what we can learn from this work for future machine learning applications. So the main three chapters of this thesis are an introduction to to machine learning and neuronal nets, the development of new machine learning methods for time series analysis and finally an interpretation of the results.

Chapter 2

Machine Learning

The basic concept of machine learning is to use statistical analysis to generate models that are able to learn from training data. This trained machine learning models are applied for automated data processing tasks. In some cases, this models can perform tasks that humans are not able to do, but today this cases are not very often. In most of the time machine learning algorithms are used to replace humans. Humans have the ability to gain an understanding from the underlying data, but machine learning algorithms are only able to learn patterns and some relationships from it, so in general humans are better in analysing data. Today it is important to find out where the limits of machine learning lie and how we can use machine learning in a meaningful manner. The field of machine learning became popular in the 1990s due to a number of new discoveries such as support vector machines and the use of long short-term memories in neuronal nets. The rapidly developing computer industry that produces processing units with a strong growing computational power, gave the field of machine learning a big boost in the past few years. [1] [7]

The main machine learning tasks are:

1. Classification

   The machine learning algorithm is trained to assign an input to a specific category. Therefore, the model learns to find boundaries between them. In this case, a category is a subset of the data that is defined by the user and depending on the specific task. The result obtained by the model is categorical. [8]

2. Regression

   Regression is used to model the relationships between a target variable and one ore more explanatory variables. The output of a regression is numeric. [8] [9]

3. Clustering

   With this kind of task the algorithm tries to find a given number of clusters with samples that have similar features.

4. Anomaly Detection

   Anomaly detection is a collective term of finding anomalies in data. This can be achieved by classification, regression or clustering methods.

5. Clustering

   With this kind of task the algorithm tries to find clusters or groups of similarities and relationships between the input samples.

There are two different ways of training and applying machine learning models. If a model is used in an unsupervised way, data is put into it without any labeling or target values. The second way is supervised learning. The input is split into the data and its labels.[1] [4]

2.1 Supervised methods

Supervised learning methods try to learn known dependencies between a given training data and the expected outcome. The input and the expected outcome (referred to as labels) are vectors or matrices in a given shape. According to [1], the learning algorithm tries to learn a mapping function

$$M = C^d \rightarrow C^m \tag{2.1}$$

that is able to map the input **x** of a given data set

$$S = (x_1, y_1), ..., (x_n y_n) | x_i \in C^d, y_i \in C^m \tag{2.2}$$

to the output **y** with minimum loss. The loss is a measurement of the error between the output of the model and the target values. It is described in 3 in more detail. The dimensions m and d don't need to be equal. This means it is possible to learn a one-to-one, a many-to-many, a one-to-many or a many-to-one mapping. Typical supervised models are:

1. Support Vector Machines
2. Decision Trees
3. Neuronal Nets

For supervised models there are a few metrics that help to get a measurement of the models accuracy on the training data and also of the generalizability of the model.

2.1.1 Confusion Matrix

For classification tasks a confusion matrix visualizes the real and the predicted classes to evaluate the accuracy of the model for every class. In figure 2.1 a confusion matrix for a binary classification task (only two classes) is shown.

TABLE 6 Confusion matrix.

	Actual positives	Actual negatives
Predicted positives	True positives	False positives
Predicted negatives	False negatives	True negatives

Figure 2.1: Confusion Matrix [1]

For this binary classification task there is only a positive and a negative class. True positive and true negative mean a correct classification of the positive and negative classes. False negative and false positive mean an incorrect classification. A false negative is an incorrect classification of a sample from the positive class. Analogously, the false positive behaves the other way round. A confusion matrix restricted to binary classification. It can also be applied to an arbitrary number of classes.

Consider a confusion matrix C of k classes. That means the confusion matrix has k rows and k columns. The accuracy α of the model is a percental value of how much classifications are correct with respect to all classifications. One way to compute it is

$$\alpha = \frac{\sum_{i=1}^{k} C_{ii}}{n} \tag{2.3}$$

with n as the number of tested samples. If the accuracy is used to improve the model it is important to take the different consequences of false negatives and false positives in real applications into account. This can be achieved by weighting of the different matrix entries in the confusion matrix C. [1]

2.1.2 Precision-Recall Curve

To understand this possibility of evaluation we have to introduce new variables. The Precision π for the jth class is given as

$$\pi(j) = \frac{C_{jj}}{\sum_{i=1}^{k} C_{ij}}. \tag{2.4}$$

Precision is a measurement of how much the samples that are classified as class j are

relevant (right classified). The recall for the jth class is given as

$$\rho(j) = \frac{C_{jj}}{\sum_{i=1}^{k} C_{ji}}, \tag{2.5}$$

and gives information on how many of the samples that belong to class j are classified as class j. [1]

A plot of the recall against the precision is a way of visualizing the classification results (Fig. 2.2)

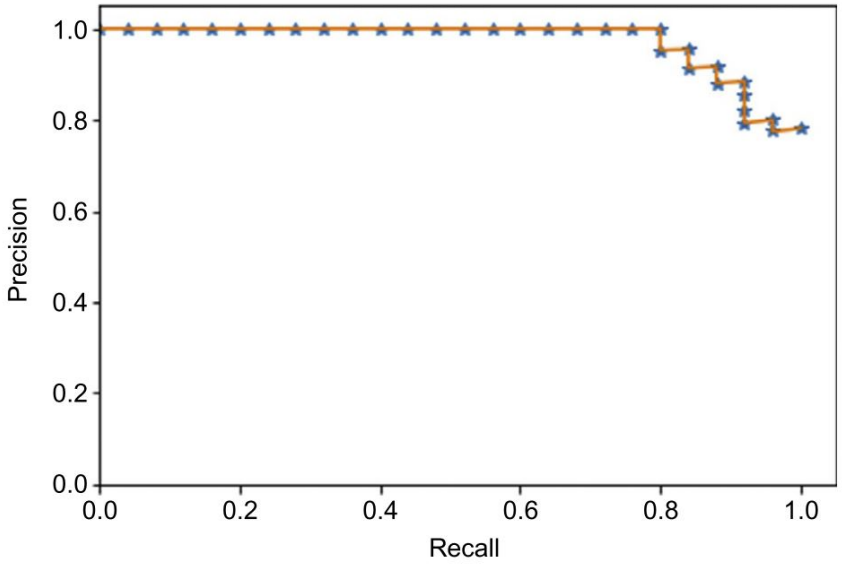


Figure 2.2: Precision recall curve as a measurement for evaluation in machine learning. Every point in the diagram is computed with a certain threshold for a binary classification. This threshold defines whether a sample should be classified as class 0 or class 1 and is a value between zero and one. A perfect classification has at least one point at position (1,1) with means that with at least one threshold an absolutely correct classification with no false positives and no false negatives is possible. [1]

2.1.3 ROC curve

The receiver operating characteristics (ROC) curve of a binary classification is a plot of false positives vs. true positives and summarizes the performance of the classification in the positive class (Fig. 2.3).[1] [2]

The PR and the ROC curves can also be applied to multiclass applications. Then, for every class there is a separate curve.

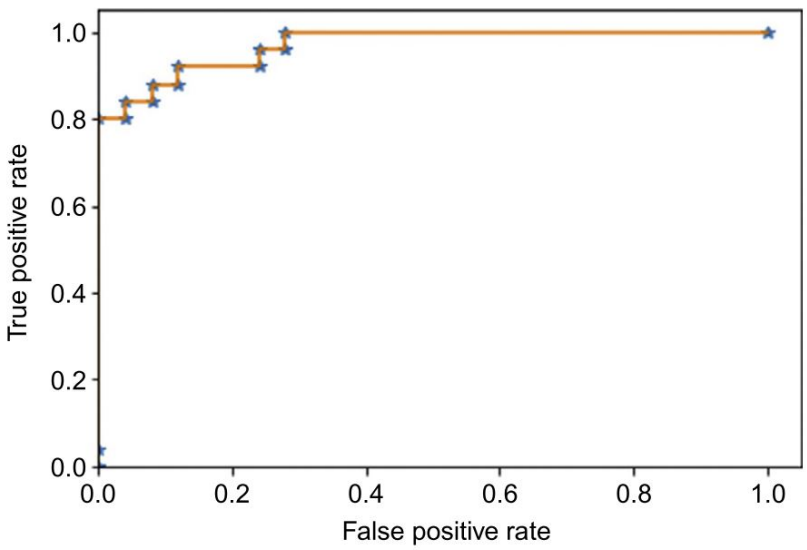


Figure 2.3: The computation of the ROC curve is similar to the PR curve. For different thresholds of a binary classification, the true positive rate and the false positive rate is evaluated and drawn into a diagram. An indicator for a good classification in the positive class is a curve that is close to point (0,1). [1] [2]

2.2 Unsupervised methods

In many applications the input data for a machine learning task is not labeled. The data can be labeled by hand, but often there are huge amounts of samples to be trained on and so, a labeling by hand would require much time. Therefore, unsupervised algorithms are a way to analyse data with no required labeling. The learning task in unsupervised machine learning is to find structures and patterns in the data that can be used afterwards.

2.2.1 Clustering

Clustering is one of the most used unsupervised learning techniques. The aim is to find class bounds for a given number of classes where the distance between the centroids of the class and its belonging samples is minimized. The most popular clustering algorithms are:

1. Density-based Clustering
2. K-Means Clustering
3. Hierarchical Clustering

2.2.2 Dimensionality reduction

The aim of the dimensionality reduction is to compress the data by reducing the number of features per sample. One of the most common unsupervised techniques for this task is the principal component analysis (PCA). The PCA tries to reduce an input in a p dimensional space $\mathbb{R}^p$ to a q dimensional space $\mathbb{R}^q$,where $q < p$, with a minimum loss of information. This is accomplished by summarizing correlations in the features. The new gained features are called principle components.

Chapter 3

Arificial Neuronal Nets

Artificial neuronal networks (ANNs) have been proposed in the 1940s as an elementary computing unit in the human cortex. ANNs have been successfully used in many applications in supervised and unsupervised models. For a successful learning progress big amounts of data are used. Especially, the increasing computing power of graphics processors, driven by the gaming industry, has made it possible to calculate large amounts of data. [3] [4]

3.1 Artificial Neuron

An artificial neuron should resemble the biological neuron in its function. Such an artificial neuron is shown in (Fig. 3.1).

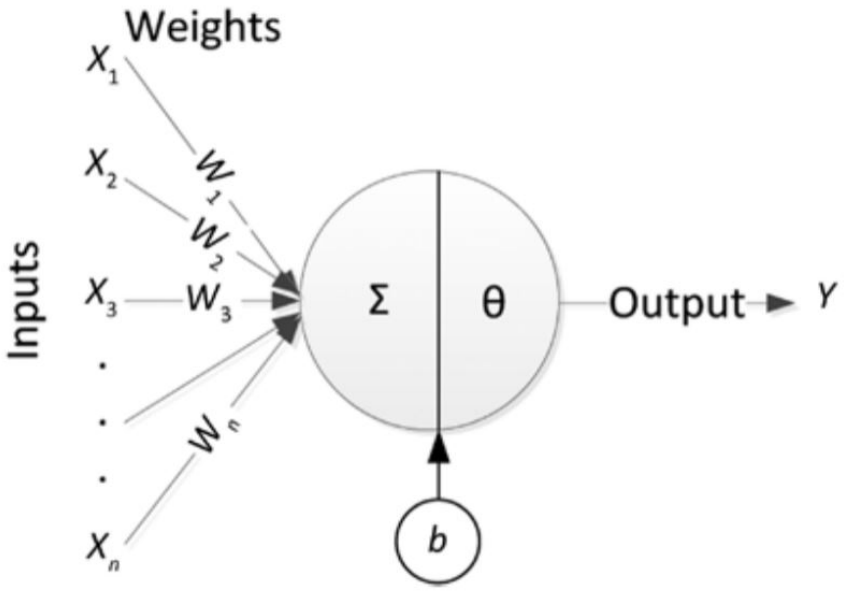


Figure 3.1: The artificial neuron as smallest unit of a neuronal network [3]

The inputs **x** are connected to the neuron through weighted connections. The summation, the bias b, and the activation function determine the output y. The influence of these parameters is explained in more detail below. The bias is a learned value that shifts the output of a neuronal net.

According to [3] the output of a neuronal network can be derived by the following equations. The artificial neuron can be described with the function

$$y = \theta \left(\sum_{i=1}^{n} \boldsymbol{w_i x_i} + b \right). \tag{3.1}$$

In matrix form this equation can be written as

$$y = \theta \left(\boldsymbol{wx} + b \right), \tag{3.2}$$

where the weights **w** are in a column vector and the input **x** is in a row vector. The activation function θ shapes the output of the neuron. There are multiple activation functions such as the hard limiter (Fig. 3.2)

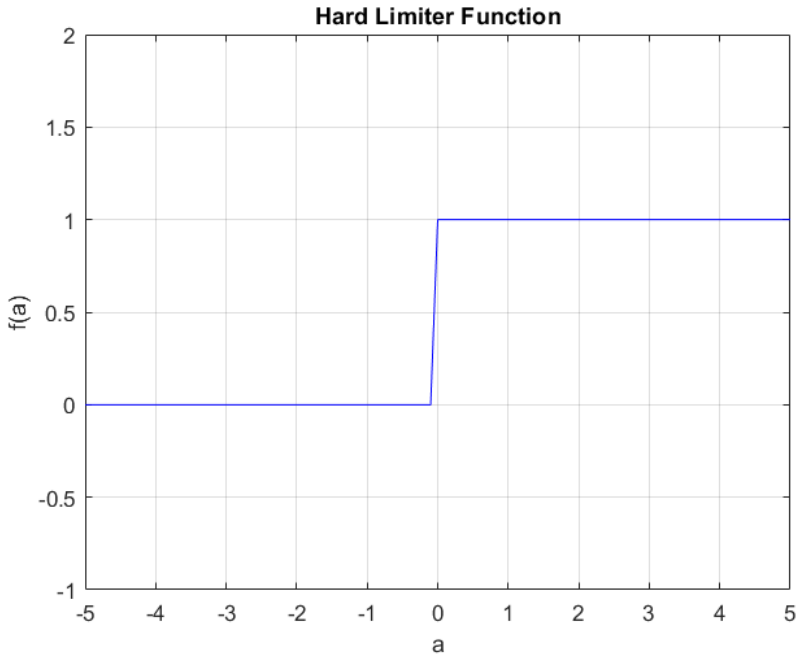


Figure 3.2: Hard Limiter

$$\theta(a) = \begin{cases} 0, & \text{if } a < 0 \\ 1, & \text{if } a > 0 \end{cases}, \tag{3.3}$$

the saturating linear function (Fig. 3.3)

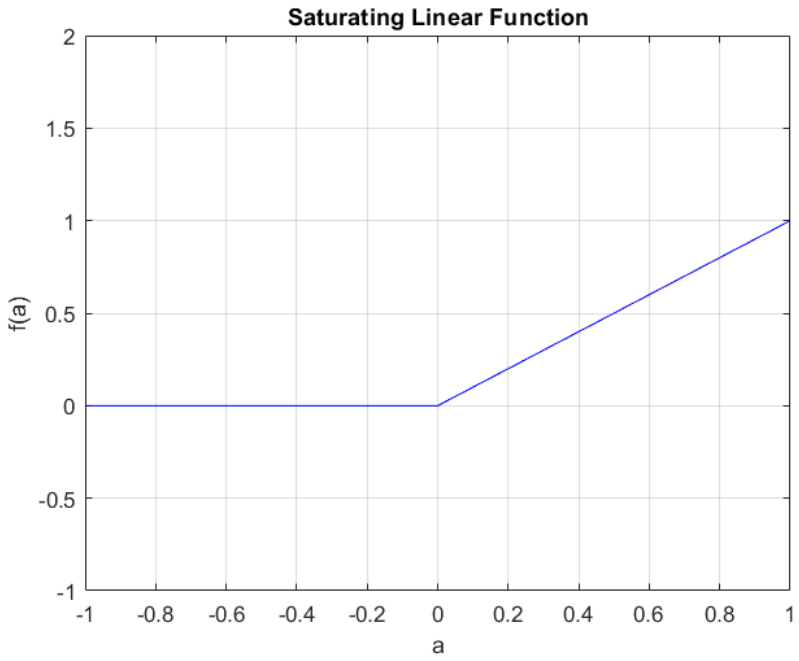


Figure 3.3: Saturating Linear Function

$$\theta(a) = \begin{cases} 0, & \text{if } a < 0 \\ a, & \text{if } 0 < a < 1 \\ 1, & \text{if } a > 0 \end{cases}, \tag{3.4}$$

the log-sigmoid function (Fig. 3.4)

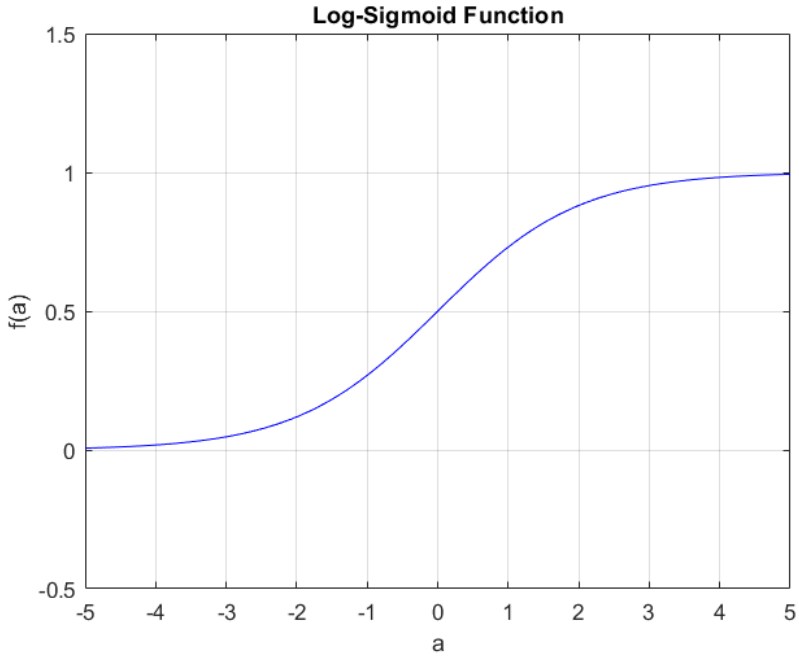


Figure 3.4: Log-sigmoid Function

$$\theta(a) = \frac{1}{1+e^{-a}}, \tag{3.5}$$

or the Hyperbolic tangent sigmoid function (Fig. 3.5)

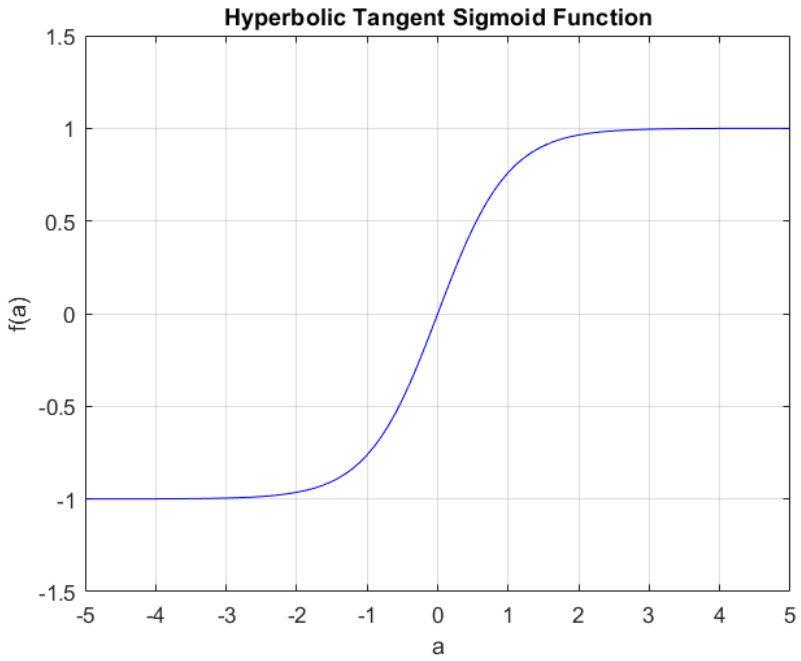


Figure 3.5: Hyperbolic Tangent Sigmoid Function

$$\theta(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}. \tag{3.6}$$

The bias b shifts the activation function to the right or the left. So the weights W, the bias b and the activation function transform the input X to an output Y.

3.2 Artificial Neuronal Network

A neuronal network is a connection between many neurons as shown in Fig. 3.6. The input X is connected to every neuron to get an output y_1 to y_n with n as the number of neurons.

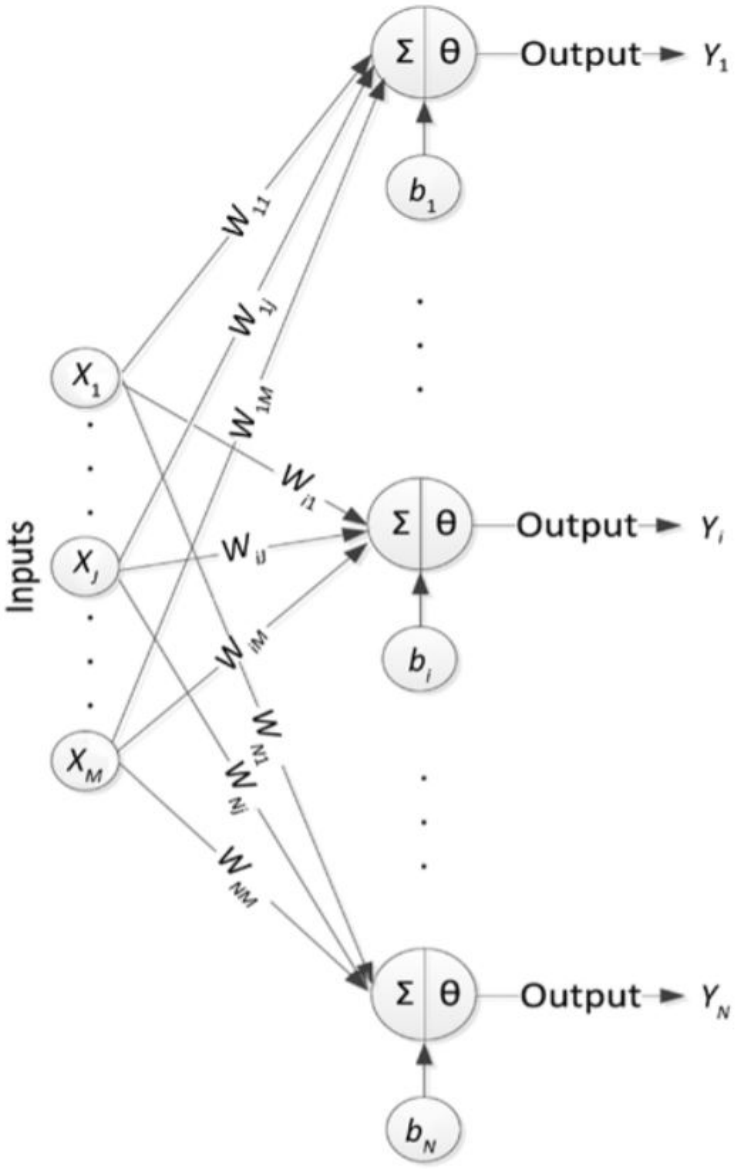


Figure 3.6: An artificial neuronal net is made by multiple artificial neurons that are connected together [3].

A layer of neurons can be represented as

$$W = \begin{bmatrix} W_{11} & \dots & W_{1M} \\ \vdots & \ddots & \\ W_{N1} & & W_{NM} \end{bmatrix}. \tag{3.7}$$

The output Y of an artificial neuronal network can be computed as

$$\boldsymbol{y} = \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_i \\ \vdots \\ \mathbf{y}_N \end{bmatrix} = \begin{bmatrix} \theta\left(\sum_{j=1}^{M} W_{1j}\mathbf{x}_j + \mathbf{b}_1\right) \\ \vdots \\ \theta\left(\sum_{j=1}^{M} W_{ij}\mathbf{x}_j + \mathbf{b}_i\right) \\ \vdots \\ \theta\left(\sum_{j=1}^{M} W_{Nj}\mathbf{x}_j + \mathbf{b}_N\right) \end{bmatrix} = \theta\left(W\boldsymbol{x} + \mathbf{b}\right) \tag{3.8}$$

with $\boldsymbol{b} = \begin{bmatrix} \boldsymbol{b}_1 \\ \vdots \\ \boldsymbol{b}_i \\ \vdots \\ \boldsymbol{b}_N \end{bmatrix}$.

3.3 Deep Artificial Neuronal Network

A deep neuronal net is made of a few layers of neurons that are added together. The deep neuronal network consists of an input layer, a few hidden layers and an output layer that can give a regression or classification output back. In Fig. 3.7 we have a three layer neuronal network.

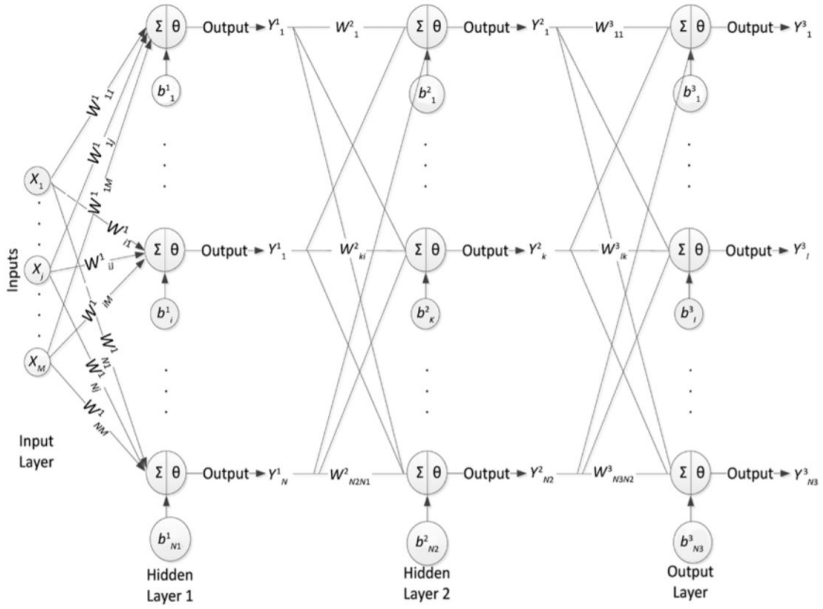


Figure 3.7: In a deep Artificial Neuronal Net many layers of artificial neurons are stacked together [3].

The output of the third layer $\mathbf{y}_3$ we obtain by this three layer network is

$$\boldsymbol{y}_3 = \begin{bmatrix} \boldsymbol{y}_{1,3} \\ \vdots \\ \boldsymbol{y}_{i,3} \\ \vdots \\ \boldsymbol{y}_{N,3} \end{bmatrix} = \theta\left(W_3\boldsymbol{y}_2 + \boldsymbol{b}_3\right) = \theta\left(W_3\theta\left(W_2\boldsymbol{y}_1 + \boldsymbol{b}_2\right) + \boldsymbol{b}_3\right) = \theta\left(W_3\theta\left(W_2\theta\left(W_1X + \boldsymbol{b}_1\right) + \boldsymbol{b}_2\right) + \boldsymbol{b}_3\right). \tag{3.9}$$

3.4 Training and Backpropargation

Given a set of labeled data x$\mathbf{X}$ with its labels t$\mathbf{T}$ the neuronal net tries to find the optimal weights, such that a cost function is optimized. The error between the target $\mathbf{t}$ and the output of the network should be minimized. A possible cost function for a labeled set of p pairs ($\mathbf{x}$,$\mathbf{t}$) is the mean squared error

$$E = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{y}_i - \boldsymbol{t}_i)^2. \tag{3.10}$$

The training process consists of the following steps:

- Initialization: The weights and biases are initialized randomly. For a good training process, the assigned values are very small.
- Feedforward: The input $\mathbf{x}$ is fed trough the network and the error between the output $\mathbf{y}$ and the Target $\mathbf{t}$ is computed.
- Feedback: The update of the assigned values is made trough backpropargation. The equations for the updates are

$$W_{ij}^k(t+1) = W_{ij}^k(t) - \alpha\frac{\partial E}{\partial W_{ij}^k}, \tag{3.11}$$

$$b_i^k(t+1) = b_i^k(t) - \alpha\frac{\partial E}{\partial b_i^k}, \tag{3.12}$$

where α is the learning rate. α defines how fast or slow the algorithm learns and affects whether the algorithm converges or not. To minimize the cost function it is necessary to compute the gradient of the error. Therefore it is essential for it to be differentiable. The network finds the derivatives of the network by moving from the last layer to the initial layer. By the chain rule, the derivatives for one layer are computed by multiplication of the derivatives of the following layers. In very deep networks (networks with many hidden layers) many multiplications of of small gradients (gradients close to zero) can

lead to vanishing gradients and stop the network from learning. It is also possible that big gradients lead to exploding gradients that can also stop the network from training. One possible solution to this problem is the use of a long short-term memory network which is described in more detail in chapter 4. [3] [2]

3.5 The loss function

The loss function evaluates the error between the network output and the target. There are several loss functions and it is important to adapt the loss function to the specific task. The most important influence is whether the classification task is a regression or a classification. The mean squared error has already been introduced. According to [2], another common error functions are the binary cross entropy loss

$$E = -\frac{1}{n}\sum_{i=1}^{N} \boldsymbol{y}_i log \boldsymbol{t}_i + (1-\boldsymbol{y}_i) log(1-\boldsymbol{t}_i) \tag{3.13}$$

for binary classification tasks, the cross entropie

$$E = -\frac{1}{n}\sum_{i=1}^{N} \boldsymbol{y}_i log \boldsymbol{t}_i \tag{3.14}$$

for multiclass classification tasks and the mean absolute error

$$E = \frac{1}{n}\sum_{i=1}^{N} |\boldsymbol{y}_i - \boldsymbol{t}_i|. \tag{3.15}$$

3.6 Programming of a Neuronal Network

There are a lot of frameworks and libraries in different programming languages that help building machine learning algorithms, especially neuronal nets with all its applications. Especially for the programming languages Python [1], R [2], C++ [3], Java [4] and Matlab [5] there are a lot of machine learning libraries available. Most of the scripts for this thesis where made in Matlab. Thus, the following section shows how a neuronal network is implemented especially in Matlab. The first task is to process the input data, so that it has the right format for the used methods and classes. The preprocessing of the data is very important and has a huge influence on the training success. In most of the machine learning tasks, the preparation requires the most of the time. All neuronal network applications

[1] ©Python Software Foundation, www.python.org
[2] ©The R Foundation, www.r-project.org
[3] ©2020 Standard C++ Foundation, https://isocpp.org/
[4] ©2020 Oracle® www.java.com
[5] ©The MathWorks Inc., Natick, MA, United States of America, www.mathworks.com

are realized by the following steps.

3.6.1 Data Preprocessing

The main task of the preprocessing of the input data is to bring the data into a shape and data type that can be used by the input layer. This can simply be accomplished by reshaping the input. To improve the learning process, standardization and normalization are two widely used opportunities.

Normalization describes the process of scaling the data between 0 and 1. This is made by the equation

$$x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min}}. \tag{3.16}$$

Standardization scales the data to a mean of zero and a standard deviation of 1. The equation

$$x_{standardized} = \frac{x - \mu}{\sigma} \tag{3.17}$$

with μ as the mean of the data and σ as the standard deviation.

This two ways of data preparation help the network to adjust its weights to the data to reduce the prediction or classification loss.

3.6.2 Definition of the Layer Structure

The layer structure defines in which order the used layers should be and which specifications they have. The following layer specifications are defined:

- Size of the input data

  The different layer structures need different input shapes. It is mandatory that the input data has the exact structure and shape as the layer requires. The training and testing data also need to have the same structure.

- Number of hidden layers

  The number of hidden layer define the complexity of the system.

- Type of hidden layers

  There are a lot of different types of layers. The classical, so called fully connected layer and the LSTM layer are introduced in this chapter and in chapter 4. There are a lot of other layer structures like convolutional layer, flatten layer, bilstm layer and many more. Every type of layer is well suited for specific tasks. That means that a good knowledge of the different layer types is obligatory for succeeding in neural network applications.

- Number of hidden units for every layer

  The number of hidden units define the number of neurons for each layer. The more neurons one layer has, the more complex representations of the input data the layer is able to learn. An increasing number of hidden neurons means more computing effort and often very strong graphics processing units are needed to have enough memory and power to compute very high dimensional data.

- Special layer specifications

  Some layers need extra specifications. For example the convolutional layer, that is often used in image processing tasks, needs a specific filter size to be able to perform the convolution. But also many other layers need some special input arguments.

- Size of the output data

  Machine learning algorithms in general can be used for classification or regression tasks. The network can be trained to assign the input to a specific class or to perform a regression on the input data. The size of the output data is depending on which application is used.

- Activation Function

  As mentioned before activation functions are nonlinear functions that shape the output of a neuron. These activation functions can also be specified in the layer structure.

In Matlab, a network is initialized with its layers as follows: An object of type layer graph is made and saved as a variable. The first network is an example for a convolutional network and the second example is an LSTM network, as it was used for a few computations for this thesis.

Listing 3.1: Convolutional network

```
imageSize = [18 18 1]
convsize = 7;

net = layerGraph([
imageInputLayer(imageSize,'Name','input_encoder','Normalization','none')
convolution2dLayer([convsize 1], 2000, 'Padding','same', 'Name', 'conv1')
reluLayer('Name','relu1')
convolution2dLayer([convsize 1], 1000, 'Padding','same', 'Name', 'conv2')
reluLayer('Name','relu2')
convolution2dLayer([convsize 1], 1000, 'Padding','same', 'Name', 'conv3')
reluLayer('Name','relu5')
fullyConnectedLayer(2 * latentDim, 'Name', 'fc_encoder1')
]);
```

Listing 3.2: LSTM network

```
net = layerGraph([
sequenceInputLayer(1, 'Name', 'input1')
lstmLayer(500, 'Name', 'lstm1')
lstmLayer(200, 'Name', 'lstm2')
lstmLayer(100, 'Name', 'lstm3')
lstmLayer(50, 'Name', 'lstm2', 'OutputMode', 'last')
fullyConnectedLayer(1, 'Name', 'fc1')
]);
```

3.6.3 Setting of the Training Options

The training options define the behaviour of the network during the training process. The most important options are:

- Maximum Training Epochs

  During training the whole training data is passed through the network multiple times. If the data is passed through one time, a so called epoch is finished. The maximum number of epochs defines when the training is finished. The number of training epochs is a very important parameter. If a neuronal network is trained for a too long period then the weights can adapt to the training data very well, but the network is not able to be applied to other data in a useful manner any more, because it is too specialized to the training data. This issue is called overfitting. To avoid overfitting, the network is tested on validation data with no weight update after every epoch to evaluate the validation error. Because the weights are only updated on the training data the validation data can be used to judge how well the trained network is suited for unknown data and the maximum training epochs can be adopted.

- Batch Size

  The batch size defines after how many input samples the weights should be updated. In practical applications a batch size of 16 or 32 turned out to be a good value.

- Learn Rate

  As shown before, the learn rate controls the gradient descent. That means if and how fast the training converges.

- Drop Out Rate

  To avoid overfitting it is possible to set a dropout rate. Drop out means that at every iteration, a small percentage of the weights get deleted and set back to initial random values.

- Gradient Threshold To avoid exploding gradients it is possible to set a gradient threshold as a maximum value a gradient can be.

The setting of the options can be made in Matlab as follows:

Listing 3.3: Training options

```
maxEpochs = 5;
miniBatchSize = 32;

options = trainingOptions('adam', ...
'MaxEpochs',maxEpochs, ...
'GradientThreshold',1, ...
'InitialLearnRate',0.005, ...
'MiniBatchSize',miniBatchSize, ...
'LearnRateDropFactor',0.1, ...
'Plots','training-progress');
```

3.6.4 Network Training and Testing

The training is made with a part of the available input data. Not all data is used for training. As mentioned before the data is split into a training and a validation part. In practice the data is split into train, validation and test data. During training, the error on the training and validation data are shown to stop the training after a few epochs to avoid overfitting. As one tries to improve the network, the hyperparameter (layer specifications and options) are adopted to get better results on the validation data. During this process, the tuning of the hyperparameter is influenced by the validation data. To get a completely independent evaluation of the whole algorithm it is necessary to use the training data, that had no influence on the whole process. The training itself can take a long time depending on the complexity of the network, the number of training samples and the size of the input data. It is recommended to use graphics processor units (GPUs) for the training, because they have a strong computing power. Originally made for the gaming industry, strong GPUs are widely used for machine learning tasks and helped to resolve very difficult machine learning tasks in the past few years. Big companies provide servers and GPU clusters to use for computationally intensive machine learning projects and help small companies, students and interested people to perform their training efficiently.

In Matlab the training is performed by the function

```
net = trainNetwork(XTrain,YTrain,net,options);
```

where XTrain and YTrain are the training samples and its assigned labels. The network (net) and the options are the parameters specified before. The testing of the network is

executed by

```
YPred = predict(net,XTest);
```

with the inputs net as the trained network and XTest as the input data.

Chapter 4

Long Short-Term Memory

The long short-term memory (LSTM) layer is a type of layer in a neuronal network that is well suited for sequencial data. The input of sequential data is not independent and the order in which the data occurs is important. LSTM layers have a structure that is able to learn time dependant patterns. For the understanding of an LSTM it is important to understand how recurrent neuronal networks (RNNs) work first. [4] [10]

4.1 Recurrent Neuronal Networks

RNNs are also networks that are made for training on sequential data. The basic concept of RNNs is weight sharing. This means to apply the same operation at each time instant. In this way, the network is not depending on a fixed length of sequences, but can adapt to any length of a sequence. An input $\mathbf{x}$ is put into the RNN. Because the input of an RNN is a sequence, for every time n there is an input vector $\mathbf{x}_n$. The output of the RNN for every time n is the output vector $\mathbf{y}_n$. To learn time dependant patterns the network has to be able to take the previous timesteps into account. This is made by introducing a state vector $\mathbf{h}$. The state vector h at a time n is $\mathbf{h}_n$. Every state vector $\mathbf{h}_n$ is depending on the previous state $\mathbf{h}_{n-1}$ and produces in combination with $\mathbf{x}_n$ the output $\mathbf{y}_n$. This operations involved in an RNN for a sequence of length n are shown in figure 4.1.

The model is described by a few unknown parameter vectors and matrices U, V, W, $\mathbf{b}$ and $\mathbf{c}$ which are used for the transformations of $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{h}$ and learned during training. This model can be described by the equations

$$\boldsymbol{h}_n = f(U\boldsymbol{x}_n + W\boldsymbol{h}_{n-1} + \boldsymbol{b}) \tag{4.1}$$

and

$$\boldsymbol{y}_n = g(V\boldsymbol{h}_n + \boldsymbol{c}), \tag{4.2}$$

where g and f are some nonlinear functions. The function f has a similar function to the

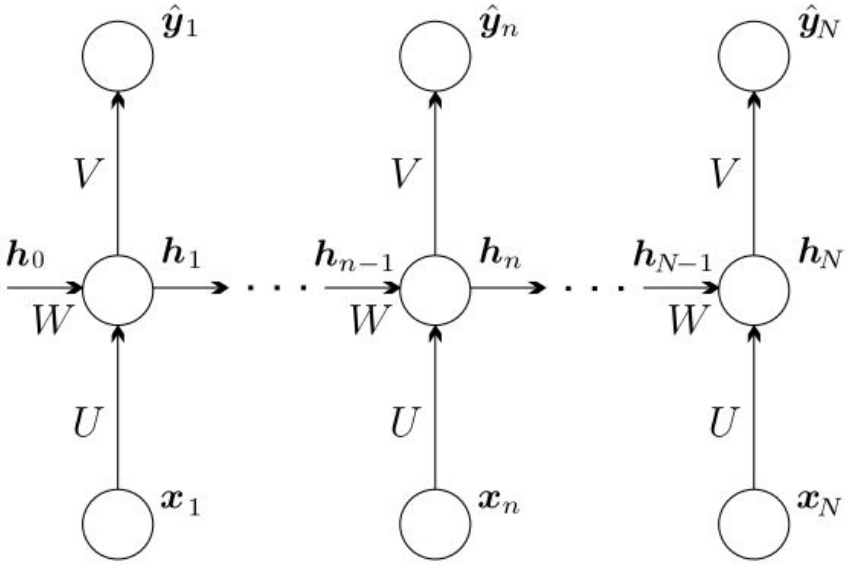


Figure 4.1: RNN Process [4]

activation function in a normal fully connected neuronal network. The output nonlinearity g is often a softmax function. The softmax function is used for classification tasks and enforces the output to add to one to get the probabilities of belonging to a certain class. In recurrent neuronal networks, the same operation is apllied for every time step of a given sequence. The layer is moving through the sequence and updates its weights for every step. This can be compared to a very deep neuronal network. To be able to learn time dependent patterns over longer sequences the network has to be able to keep information as long as possible. Because of the vanishing gradient problem, as shown in chapter 3, information can often not be kept over longer sequences. This problem is a big issue in applying neuronal networks on time series data. [4] [10] [11]

4.2 Long Short-Term Memory

To overcome the vanishing gradient problem Hochreiter and Schmidhuber introduced a new state, the cell state **s**, in addition to the hidden state of recurrent networks. Nonlinear elements, known as gates, control the information flow in the system. There are many different implementations of the LSTM structure, but the basic concept is the same in most cases. A basic LSTM structure can be seen in figure 4.2. [4] [3] [12]

Again we have the parameter matrices W and U, that are learned during training, the hidden state h and the bias b. The nonlinear gates are the activation functions σ (sigmoid function) and tanh (hyperbolic tangent function). The two sets of variables that build the LSTM's memory are the hidden state **h** and the cell state **s**. For every timestep n, the LSTM cell receives the input vector $\mathbf{x}_n$ and the two previous states $\mathbf{s}_{n-1}$ and $\mathbf{h}_{n-1}$ and

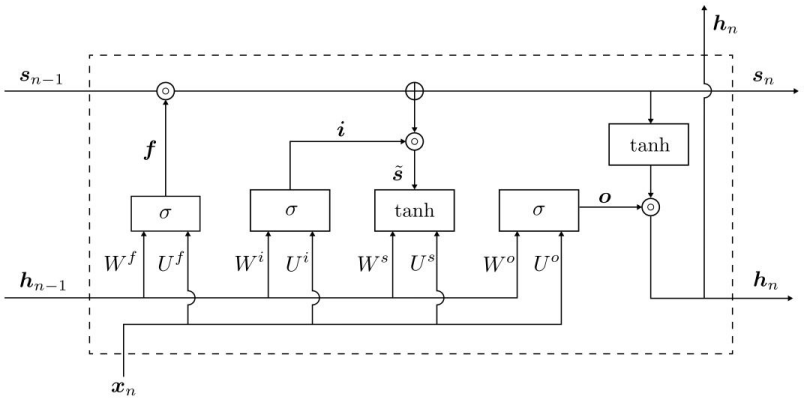


Figure 4.2: LSTM Cell [4]

passes the updated states $\mathbf{s}_n$ and $\mathbf{h}_n$ to the next timestep n+1. The two states **s** and **h** are updated by the vectors **f**, **i**, $\hat{\boldsymbol{s}}$ and **o**. The equations that can be seen in 4.2 are

$$\boldsymbol{f} = \sigma(U^f \boldsymbol{x}_n + W^f \boldsymbol{h}_{n-1} + \boldsymbol{b}^f), \tag{4.3}$$

$$\boldsymbol{i} = \sigma(U^i \boldsymbol{x}_n + W^i \boldsymbol{h}_{n-1} + \boldsymbol{b}^i), \tag{4.4}$$

$$\hat{\boldsymbol{s}} = tanh(U^s \boldsymbol{x}_n + W^s \boldsymbol{h}_{n-1} + \boldsymbol{b}^s), \tag{4.5}$$

and

$$\boldsymbol{o} = \sigma(U^o \boldsymbol{x}_n + W^o \boldsymbol{h}_{n-1} + \boldsymbol{b}^o). \tag{4.6}$$

All update vectors are the combination of the input states with the learned matrices W and U and the learned bias vector **b**.

The update of the states is a simple Hadamard product (element wise multiplication) of the update vectors and the states. The equations that these vectors to the states are

$$\boldsymbol{s}_n = \boldsymbol{s}_{n-1} \circ \boldsymbol{f} + \boldsymbol{i} \circ \hat{\boldsymbol{s}}, \tag{4.7}$$

and

$$\boldsymbol{h}_n = \boldsymbol{o} \circ tanh(\boldsymbol{s}_n). \tag{4.8}$$

The hidden state is passed to the next timestep, but also provides the output of the sequence $\mathbf{y}_n$ at the time n. In many cases the output $\hat{\boldsymbol{y}}_{\boldsymbol{n}}$ is computed by a softmax nonlinearity from the hidden state $\mathbf{h}_n$.

The basic idea behind the LSTM cell is the additional cell state s that is updated by the update vectors, but able to pass information from one to another timestep directly. So it is possible to keep information over long sequences without loss. The parameter matrices W and U help the system to keep relevant information and are learned during the training process. Depending on the specific task the weights of the matrices W and U get adopted

during the training of the model. With this neuron architecture, the model is able to take the information of previous timesteps into account. This weights are able to adopt to time series patterns according to the specific application. [4] [12]

Chapter 5

Autoencoders

An autoencoder is a neuronal network that consists of two parts. In most of the applications, an encoder reduces the dimensions of the input data and transforms the data to a latent space with a given number of dimensions. A decoder then reconstructs the input data from the latent space. So aim of an autoencoder is to reconstruct the input data after it is reduced in its dimensions. As the neuronal net gets trained, the weights in the neuronal net get optimized to keep as much data as possible to mimimize the loss between the real input data and the reconstructed data. The less dimensions the latent space has the more the data gets compressed and lost. During the training, the weights of the autoencoder get optimized to find a good representation of the average data so that the input can be reconstructed with minimum loss. When an input data is very different from the rest of the data, the reconstruction does not work very well, because the weights of the neuronal network are fitted to reconstruct these outliers. To detect outliers it is a possible way to train the autoencoder with the whole data and to test the autoencoder with the same data. The loss between the real data and the reconstruction and also the latent space representation can be measure for the outlierness of the input data. [13] [14]

There are different types of autoencoders that have different functions. A few of these are the undercomplete autoencoder, the denoising autoencoder, the sparse autoencoder and the variational autoencoder

5.1 Undercomplete Autoencoder

If the dimension of the latent space has smaller dimensions than the input data, the autoencoder is called a undercomplete autoencoder (Fig. 5.1). The reduction of the dimensions forces the autoencoder to compress the data into a lower dimensional space.

Due to the reduction of the dimensions this application works well to detect outliers in the overall shape of the data. To detect local anomalies it is better to use other computations.

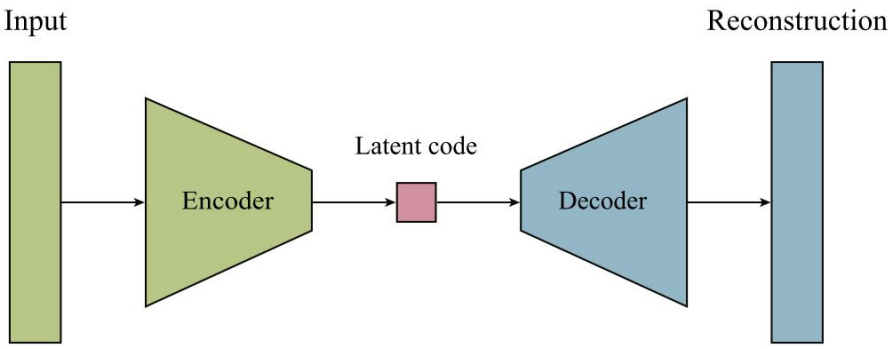


Figure 5.1: Undecomplete Autoencoder. In the latent space we have a reduced dimensionality of the data.

5.2 Denoising Autoencoder

A denoising autoencoder has the same structure as the undercomplete autoencoder, but the input is partially corrupted by masking or adding noise to some values of the input vector in a random manner. The network then learns to denoise and recover the data as shown in figure 5.2. An application for time series analysis is the denoising of measured data as a part of data preprocessing, if a sensor has some noise.

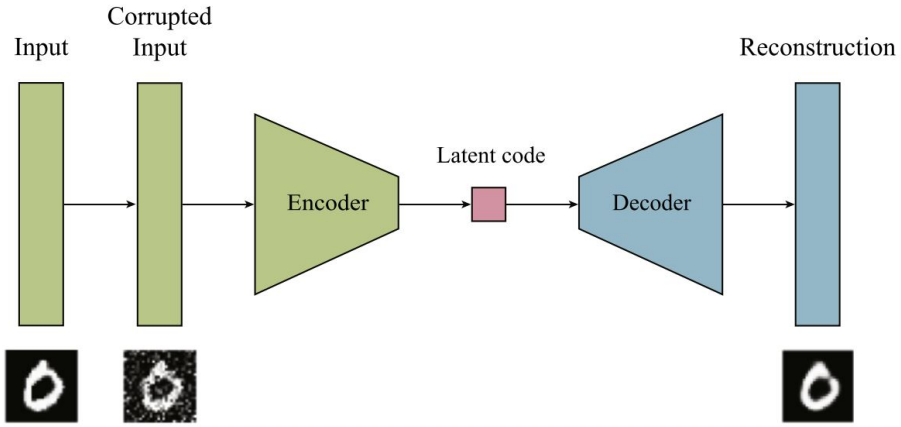


Figure 5.2: The denoising autoencoder is trained to remove noise in the input data.

5.3 Variational Autoencoder

Passing the data trough a latent space with a high dimensionality, the autoencoder is able to encode and decode the data with a very low information loss. This is quite good but can lead to a severe overfitting. If the model is overfitted, some points in the latent space can give meaningless output. Especially for generative tasks it is important that the representations in the latent space lead to meaningful outputs. To avoid this overfitting a regularization is needed. Instead of encoding the input as a single point, it is encoded as a

distribution in the latent space with a mean and its variance (Fig. 5.3). [5] The data then

Figure 5.3: Variational Autoencoder [5]

is sampled and put through the decoder. Sampling means to train the decoder with the values of the mean plus a random number $\boldsymbol{\varepsilon_i}$ multiplied with the learned sigma (σ). The corresponding Matlab Code is shown in the following listing.

Listing 5.1: Sampling of the latent space representation of a variational autoencoder

```
compressed = forward(encoderNet, x);
d = size(compressed,1)/2;
zMean = compressed(1:d,:,:);
zLogvar = compressed(1+d:end,:,:);

sz = size(zMean);
epsilon = randn(sz);
sigma = exp(.5 * zLogvar);
z = epsilon .* sigma + zMean;
```

For every training iteration the input vector **x** gets encoded. The encoded data gets split into two halfs. The Mean values and the corresponding sigma (or log variance) values. The result of the sampling is a vector **z** that contains the latent space representation that is computed with

$$\boldsymbol{z} = \boldsymbol{\varepsilon} * \sigma + \boldsymbol{z_{Mean}} \tag{5.1}$$

With this regularization, the variational autoencoder is able to learn a latent space that has no representations as single points, but a mean and a variance. In figure 5.4 the latent space of an autoencoder is shown. On the left side without regularization and on the right side there is a regularization. With a regularized learning process, the autoencoder is able to decode data that is different from the learned data in a meaningful way.

The variational autoencoder is able to create meaningful representations between different points in the latent space, as shown in 5.5.

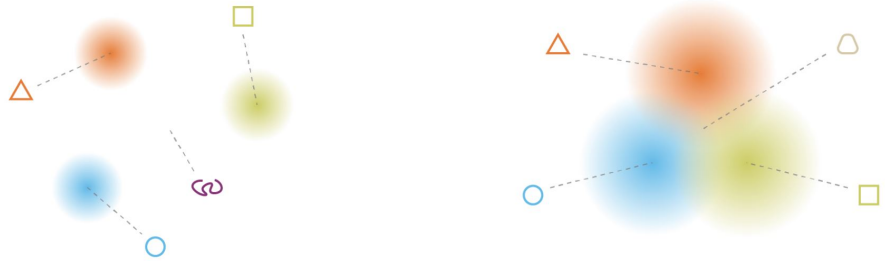


Figure 5.4: Schematic latent space representation of an autoencoder (left side) compared to a variational autoencoder (right side) [5]

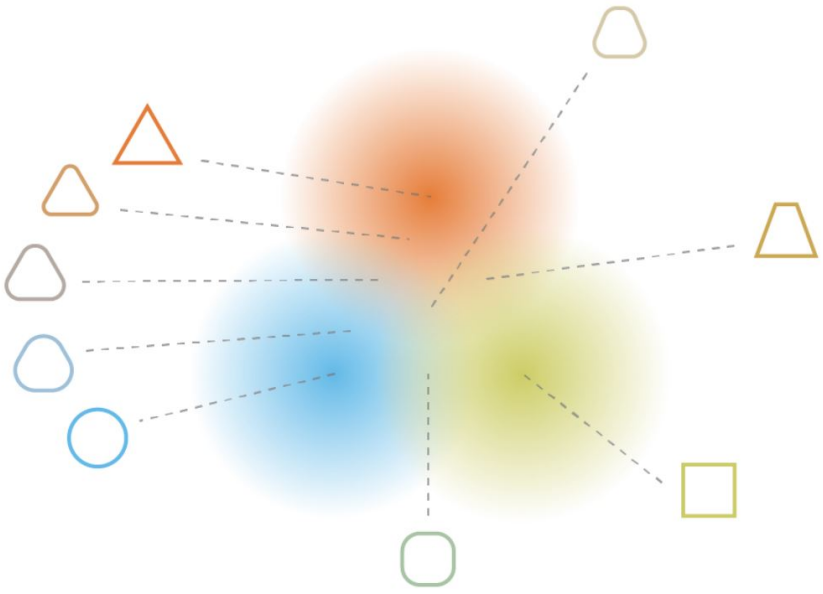


Figure 5.5: Probabilistic latent space representaion of a variational autoencoder [5]

Chapter 6

Underlying Data

Time series data is data that is measured and collected at different points in time. The importance of this kind of data and its analysis is growing due to the increasing production of time series data in the monitoring of industrial machines, the internet of things etc. For the improvement of industrial processes, weather prediction or many other applications it is necessary to measure and evaluate data in the context of time. Many naturally occurring processes follow specific rules, which means that a future state is depending on one or more previous states. This dependencies lead to patterns that are characteristic for certain processes. Time series analysis is the field of analysing this data which means extracting information out of the underlying data to be able to gain some knowledge from it.

With the proposed methods it is not only possible to analyse time series data, but sequential data in general. This can be for example DNA data or data that is gained from written text. Time series data, as a subsection of sequential data is in relation to the dimension of time, so it is also very important to take the order of the occurring data points and any possible time dependant patterns into account. It is important to know, that there is no need that time series data has to be one dimensional. When more than one measurement is taken from a certain process per timestep, we obtain two dimensional data. One dimension is the dimension of time and the second dimension specifies the so called number of channels.

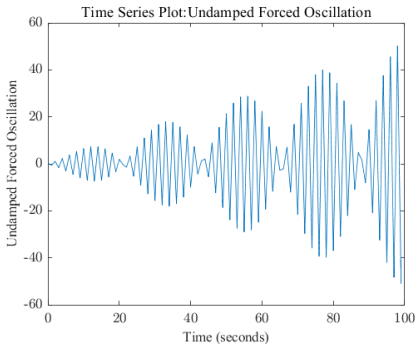
6.1 How to gain Knowledge from Time Series Data

To extract information from time series data there are a variety of methods available. The application of this methods on the underlying data is called time series analysis. In figure 6.1a and figure 6.1b two exemplary time series plots are shown to visualize the process of time series analysis. In figure 6.1a an undamped forced oscillation is shown like it occurs

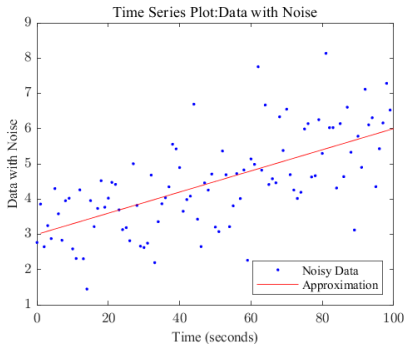
in many natural processes. The underlying equation is

$$y(t) = 0.5 * x * sin(3t) + cos(\frac{7}{5}t). \tag{6.1}$$

The aim of time series analysis is to draw conclusions about the underlying mathematical relations from the raw sequential time series data. If one is able to find out that the data in this example can be represented by a simple mathematical formular, high quality information is gained from the data for further purposes. In many natural processes, the measured data has some noise in it due to the uncertainty of the measurement etc. Even if it does not seem that the noisy data in figure 6.1b has any information in it on a first look, we can also extract some information from this data. If we approximate the data points we get a curve we are able to evaluate. The gained values like mean value, slope, standard deviation etc. can give us also information of the monitored process. Consequently, time series analysis describes a variety of methods to extract as much information as possible from time series data with high quality to be able to understand the data that is measured. In this thesis these kind of methods are machine learning methods and in more detail different models of neuronal nets.



(a) Undamped forced oscillation with two excitation forces as an example of time series data.



(b) In most of the data that is measured from physical processes, noise is added to the data, but this does not prevent one to gain information from it.

6.2 Exemplary Data

To be able to evaluate the proposed methods it is necessary to evaluate the models on exemplary data. The data that was made available to me consists of the raw data and some data derived from it. The raw data are time series data for each individual drilling point, which were measured by the Keller company.

The two main data sources for further calculations were the time series data for each borehole and key performance indicators that were statistically evaluated for each borehole. This exemplary data forms the data basis for the evaluation of this work. The terms borehole, geodrilling point or simply point are the same and refer to a single compaction

point (a single data sample).

The most important data source for further calculations is the depth data over time as shown in (Fig. 7.2).

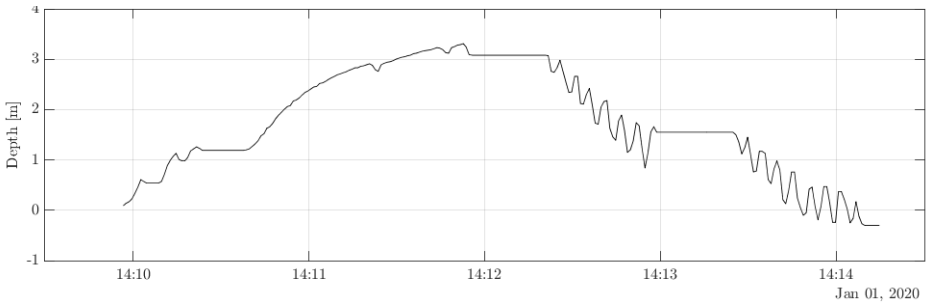


Figure 6.2: Depth data over time of a drilling point at the site Seestadt Aspern as an example of the raw input data

6.2.1 Raw Data

The raw data is time series data provided by Keller. In figure 6.3 an example of the time series data is shown.

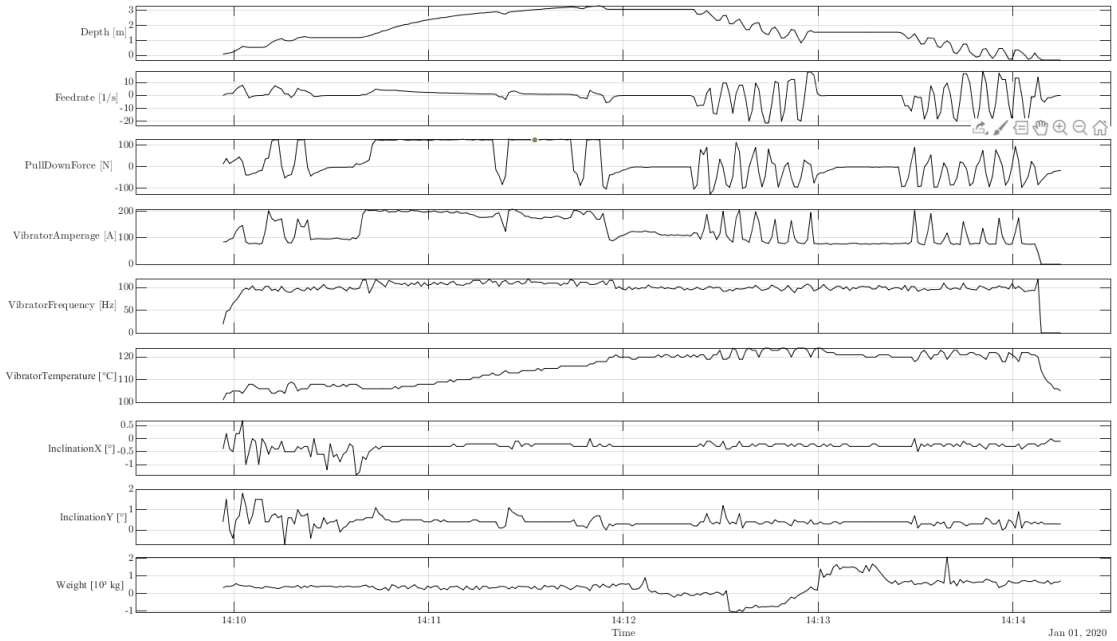


Figure 6.3: Exemplary raw input data data over time of a drilling point at the site Seestadt Aspern

6.2.2 Derived Data

Derived data was calculated for each point from this data such as key performance indices, discontinuity data and a statistical summary.

Point KPIs

The key performance indices (KPIs) were defined for every point and calculated from the raw data (Table 6.1). The purpose of the KPIs is to summarize the time series data according to suitable indicators and to increase the clarity of the data by presenting important process data in a compact manner.

Table 6.1: Point KPIs

	KPI	Unit
Duration	14,61108333	min
DurationPerMeter	1,462570904	min/m
PreTime	2,092666667	min
PenetrationTime	1,57335	min
CompactionTime	11,3807	min
PostTime	12,51841667	min
PetetrationTimePerMeter	0,157492492	min/m
CompactionTimePerMeter	1,139209209	min/m
NonProductive	58,46681922	%
StartDepth	8,97	m
MaxDepth	8,97	m
EndDepth	-0,09	m
nrGravelFills	3	
nrComStepsTotal	30	

Discontinuity

Locations, where time series data is discontinuous can be determined by convolutional methods. C^{n+1} discontinuities are a measure for discontinuities in the $n-th$ derivative of the raw data. It showed, that this measure for the discontinuity was a suitable input for the time series prediction (Chapter 7), which was made to detect local outliers. An example for the C^1 discontinuity is shown in Figure 6.4. [6]

The discontinuity was computed by convolution, as shown in the following code.

Listing 6.1: Computing the discontinuity with the Code snippets [6]

```
%%
% Define the continuity degree
%
% This defines the degree of continuity forced during the
% constrained approximation, e.g., $C^0$ continuity. This is
% suitable if we are trying to detect $C^1$ discontinuities.
%
continuity = 1;
%%
```

```
% _Define appropriate degrees for the polynomials_
%
% The continuity constraint plus 1 is the minimum polynomial
% degree required.
%
dLeft = continuity + 1 ;
dRight = dLeft ;
ds = [dLeft, dRight];

seq = taylorCnSequence( ls, continuity, ds );

discontinuity_data = conv( values, seq, 'same');
```

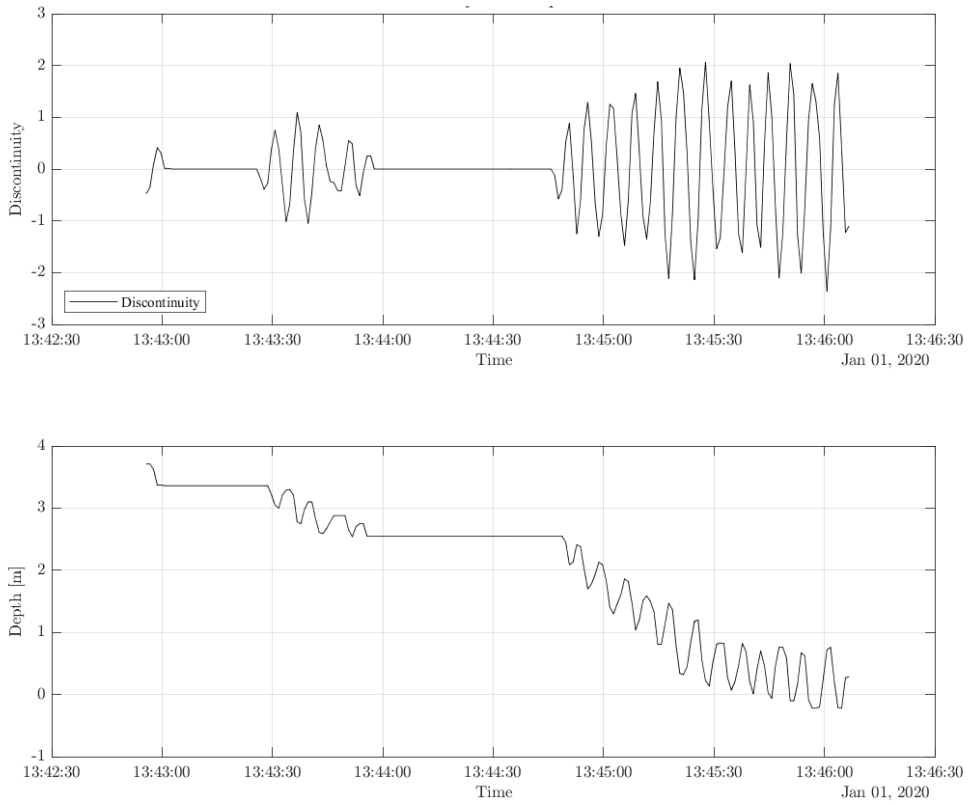


Figure 6.4: C^1 discontinuity as a measure of the discontinuity of the depth data. The upper subfigure shows the discontinuity that was computed from the depth data, shown on the lower subfigure. The convolution was performed with a support length of ls = 7.

Chapter 7

Time Series Prediction

7.1 Outlier detection via prediction (basic principle)

In time series prediction an LSTM network is trained to predict future timesteps depending on the previous timesteps, as shown in figure 7.1. It is possible to use machine learning algorithms for prediction tasks. A neuronal network is trained with a sequence (n previous timesteps) as input to predict the corresponding future sequence (m future timesteps) with minimum loss. So if there are regular patterns in the data the weights of the network get adapted to those patterns during learning and these trained networks are later used for the prediction. If there are local anomalies in the data the prediction error rises. Consequently, a high prediction loss indicates an anomaly and the degree of the error can be used as a measure of the outlierness of a data point. To train and evaluate the models, the exemplary data from the Keller Grundbau GmbH was used, but in general, this methods can be applied to all kinds of time series data. In this application the time series prediction is used to detect outliers in the depth data (Fig. 7.2), in the derived depth data and in the discontinuity of the depth data. In this case we used long short-term memory (LSTM) layers in the network, because this type of layer is made for analysis of time series data and therefore well suited for our task (See chapter 4).

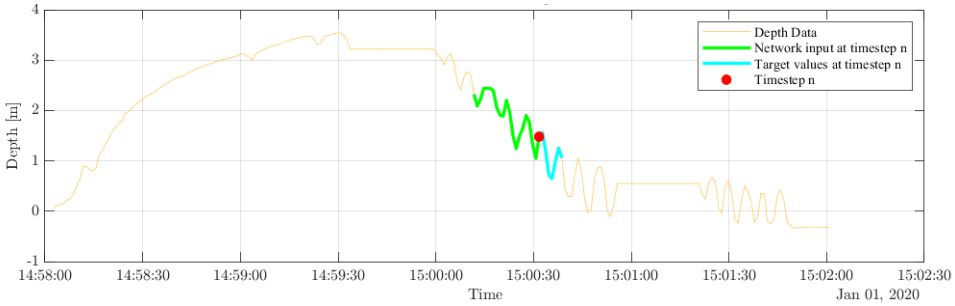


Figure 7.1: Scheme of the time series prediction. For every timestep, the network learns to predict one or more future timesteps depending on the previous timesteps. This image shows the network input and the future timesteps (target values) for an arbitrary timestep n.

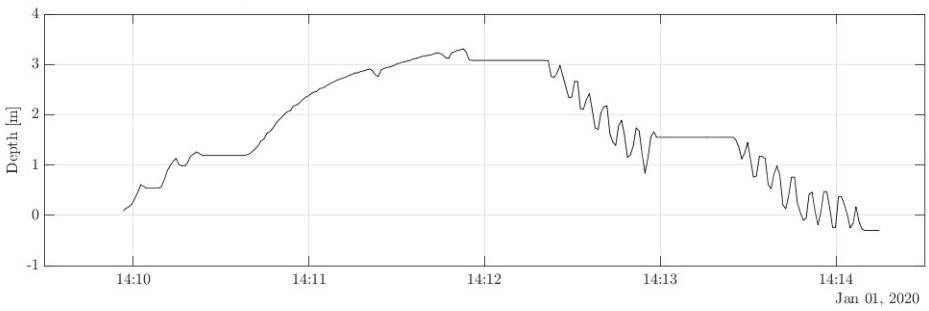


Figure 7.2: Exemplary time series data as input for the time series prediction

7.2 Prediction of the discontinuity data

Small anomalies in the compaction steps lead to bigger ones in the discontinuity data, so the same process of predictive anomaly detection applied to the discontinuity data led to better results. The networks got trained with the following specifications:

Listing 7.1: Defining the network architecture of an LSTM network for time series prediction

```
%% Define LSTM Network Architecture
% An LSTM regression is used to predict a sequence of timesteps based on
% previous timesteps

inputSize = 1;
numHiddenUnits = 1500;
numResponses = 1;

layers = [ ...
sequenceInputLayer(inputSize)
bilstmLayer(numHiddenUnits)
fullyConnectedLayer(numResponses)
regressionLayer]

%% Specify the training options.

maxEpochs = 1;
miniBatchSize = 8;

options = trainingOptions('adam', ...
'MaxEpochs',maxEpochs, ...
'GradientThreshold',1, ...
'InitialLearnRate',0.005, ...
'MiniBatchSize',miniBatchSize, ...
'LearnRateSchedule','piecewise', ...
```

```
'LearnRateDropPeriod',10, ...
'LearnRateDropFactor',0.1, ...
'Verbose',0, ...
'Plots','training-progress');
%% Train the network
% Train the network based on the above defined settings
if LoadNet == false;
prednet = trainNetwork(XTrain,YTrain,layers,options);
end
```

The predictive error is very low, if the data of a point follows regular learned patterns (Fig. 7.3).

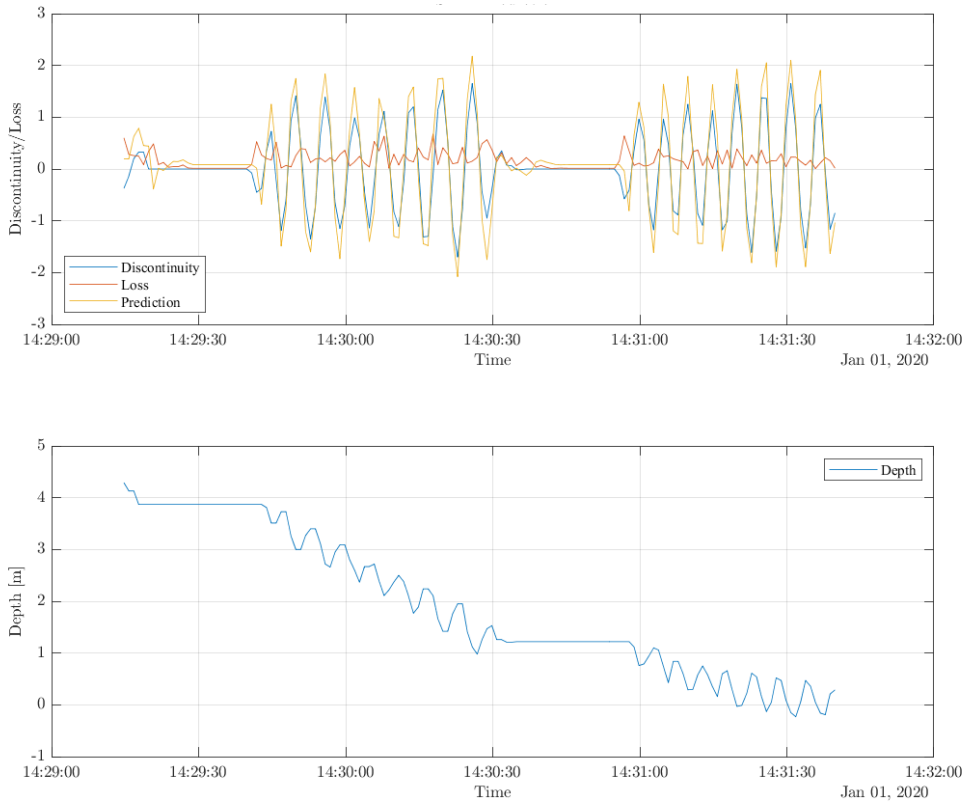


Figure 7.3: LSTM Prediction. On top, the computed discontinuity, the prediction and the loss between the prediction and the discontinuity data is shown. Discontinuity and depth data with no anomalies result in a low error over the whole sequence The used sequence is an example for a normal sequence that follows regular patterns compared to the other training data. The network is able to predict the data points with low loss. The raw data is shown on the lower subfigure.

Small anomalies in the compaction steps can be easily found in the same way as shown before (Fig. 7.4 and 7.5).

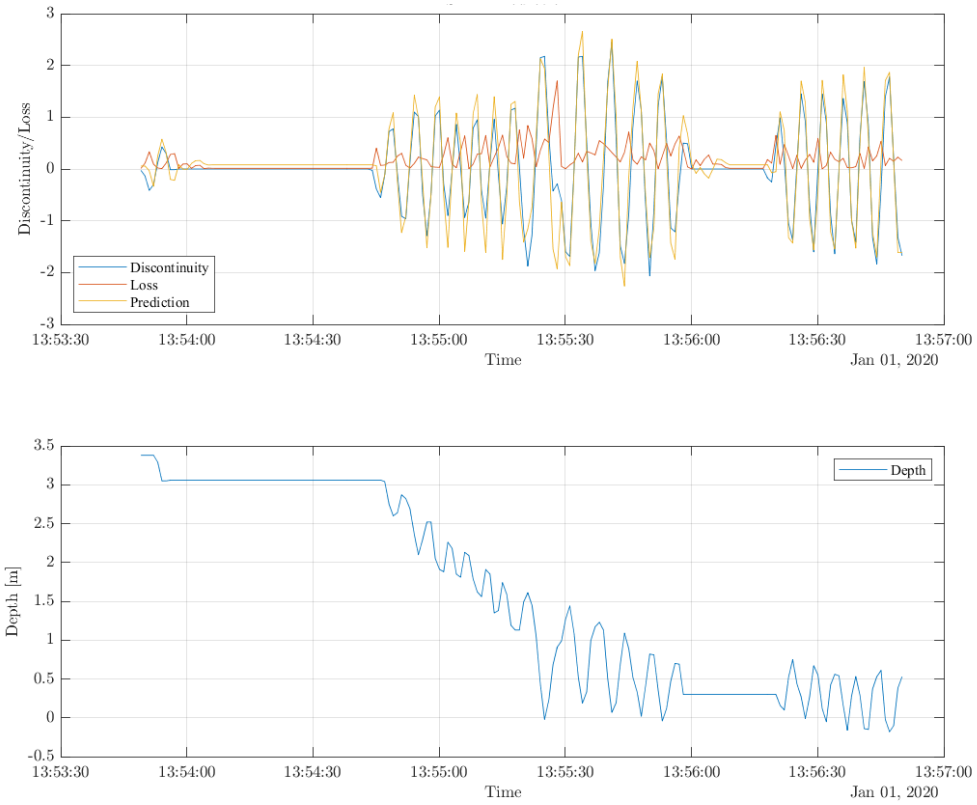


Figure 7.4: LSTM Prediction. Small anomalies in the depth data, that is shown on the lower image, also result in anomalies in the derived discontinuity data. The weights of the trained LSTM network got optimized to regular occurring patterns in the data during training. An anomaly in the discontinuity data results in a high prediction error. This high prediction error is a peak in the loss curve and indicates an anomaly at this position (as shown in the upper subfigure).

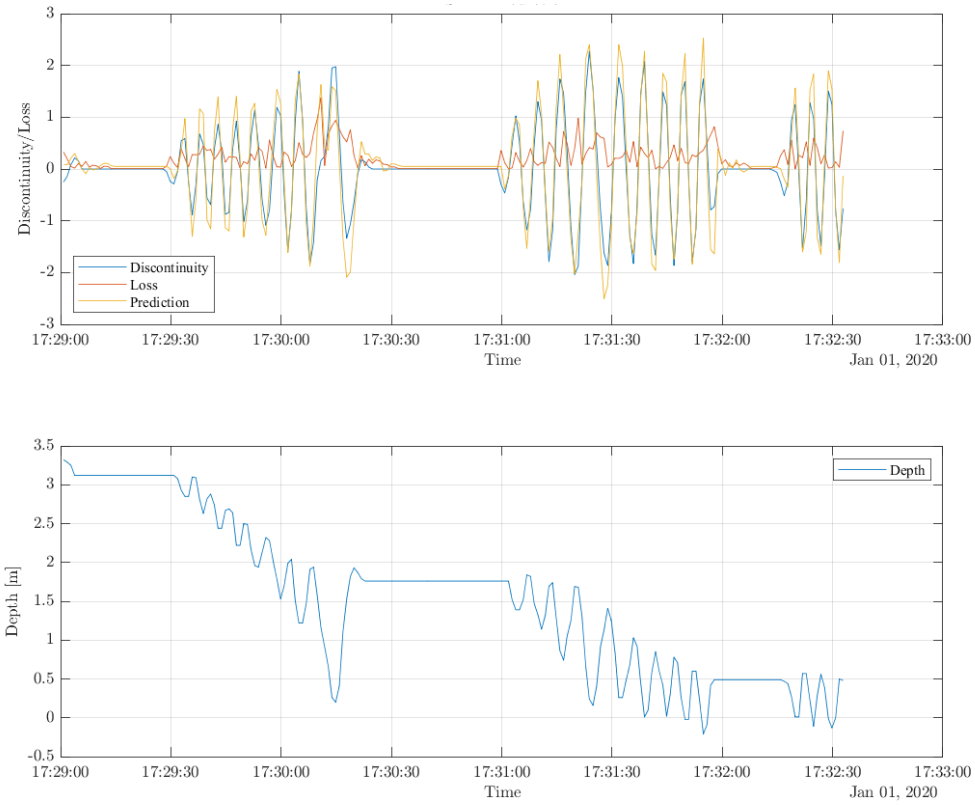


Figure 7.5: LSTM Prediction. Again, small anomalies in the depth data result in a peak in the error function.The anomalie in the raw data at 17:30:15 on the lower image leads to an anomalie in the discontinuity, shown on the upper subfigure, which lead to a pak in the prediction loss.

The maximum error for every point is a measurement for its outlierness. A visualization of the maximum error of every point can give a quick overview about many points from a dataset, as shown in 7.6.

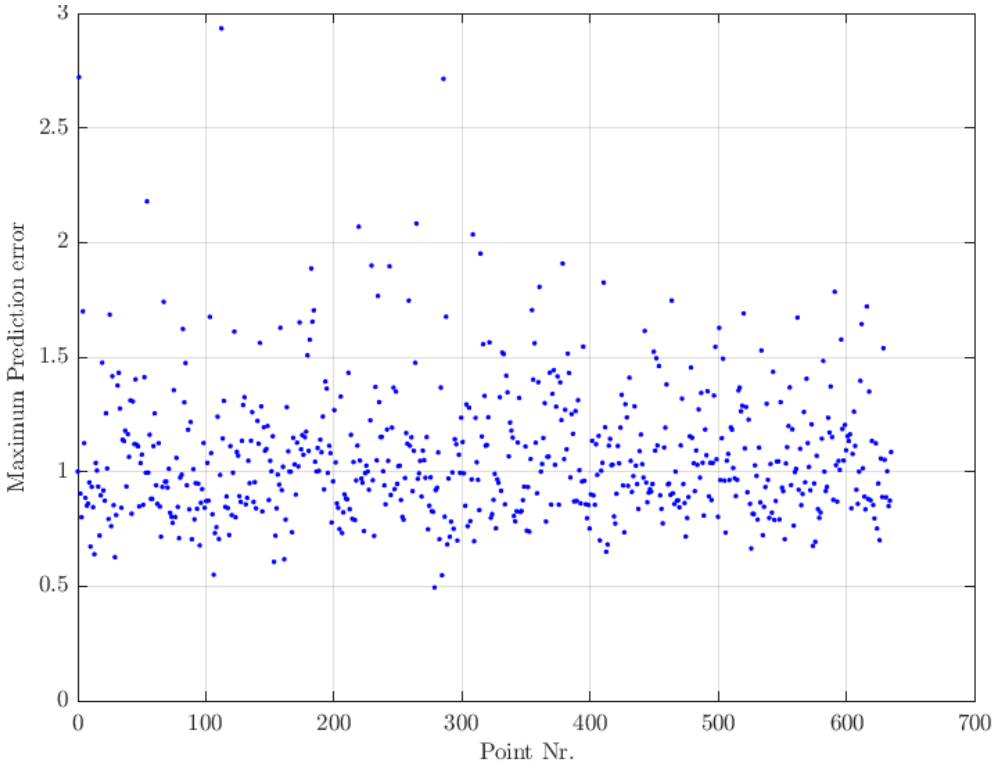


Figure 7.6: The maximum loss of all points from an exemplary dataset (the site Seestadt Aspern from the Keller data). The maximum error is very different for different points and can vary by a factor of 6 at this application. This indicates, that there are points with very regular patterns in the dataset and points with anomalies.

7.3 Time Series Prediction Results

To get a measure of the performance of the time series prediction 635 geodrilling points from the evaluation data from Keller (site Seestadt Aspern) were labeled by hand and compared to the results of the time series prediction. A point, that contains an outlier was labeled with a 1 and a point with no outliers with 0. The results of the time series prediction were labeled the same. For the classification of the time series prediction a threshold was used. A maximum loss above a certain threshold led to a classification into class 1 (Outlier). The threshold was varied from a loss of 0.5 to 2 in steps of 0.01 to optimize the threshold and to be able to see the best performance. For every threshold the classification by hand and the classification of the model were compared and the Pearson correlation coefficient

$$\rho(\boldsymbol{a},\boldsymbol{b}) = \sum_{i=1}^{n} \frac{(\boldsymbol{a_i} - \mu_a)^2}{\sigma_a * \sigma_b} \tag{7.1}$$

was computed with **a** and **b** as the two vectors that contained the classes, μ_a and μ_b as the means of the two vectors and σ_a and σ_b as the standard deviations. The best

achieved correlation coefficient was $\rho = 0.6219$ at an optimal threshold of 1.23 and indicates a medium to strong positive correlation. In figure 7.7 the manual classification compared to the maximum loss of the timeseries prediction per sample is shown. The corresponding confusion matrix to this results can be seen in figure 7.8. The majority of the predictions lie in the true positive and true negative class wich indicates a working prediction. Although a few classifications were wrong, which means that there is some need for improvement to this kind of classification.

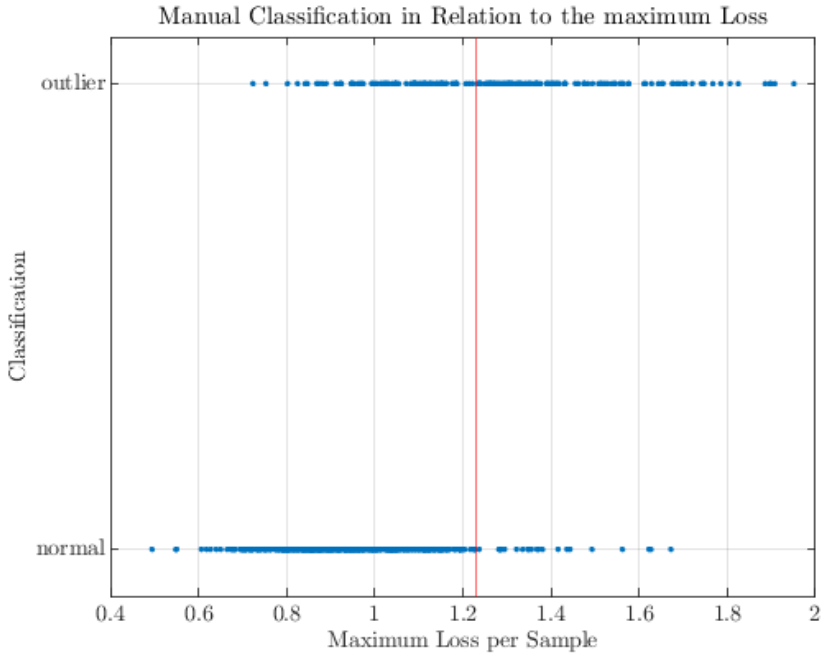


Figure 7.7: Manual classification compared to the maximum loss of the timeseries prediction per sample. The red line represents the optimal computed threshold of 1.23 for the exemplary dataset.

From a maximum prediction error of 0.5 to 2.0 the accuracy of the prediction was computed to evaluate the model. The classification of the data was made with the optimal threshold of 1.23 again.

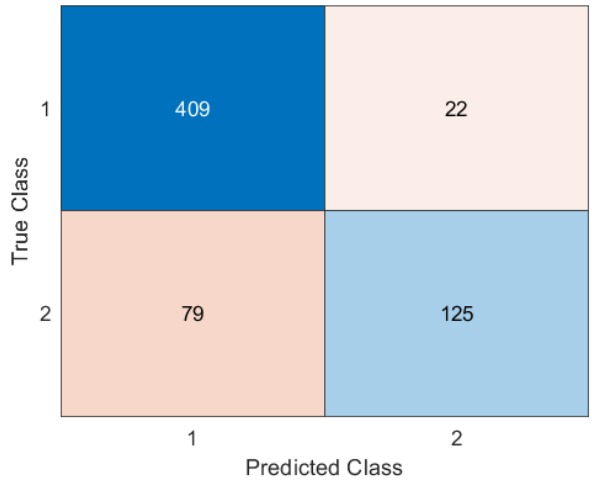


Figure 7.8: Confusion matrix of the time series prediction results. The true class represents the manual classification and the predicted class represents the time series prediction results.

Obviously, time series prediction is a working method for detecting local anomalies in time series data. The peaks in the prediction loss always occurred due to anomalies in the data. Nonetheless there are two main issues that come with unsupervised time series prediction and in unsupervised machine learning outlier detection in general.

The first issue is, that the data needs some preprocessing for a good detection performance. The anomaly detection of the discontinuity data worked much better than with the depth data. In this case the translation of the data into a form where the anomalies can be detected more easily was necessary. The "learning" part of Machine Learning needed some human input, although every necessary information was in the depth data.

The second issue describes the non application-oriented behaviour of the LSTM model. The model was able to detect outliers in the data. But in an industrial process some outliers can be an indicator for severe problems in the process and others do not affect the process in any negative way. A machine learning model can not differentiate between those types of anomalies in the measured data, when it is trained in an unsupervised way, but for industrial processes, it is very important that a model that is able to analyse data autonomously is able to do that unsupervised.

Chapter 8

Variational Autoencoder

8.1 Visualization in the latent space

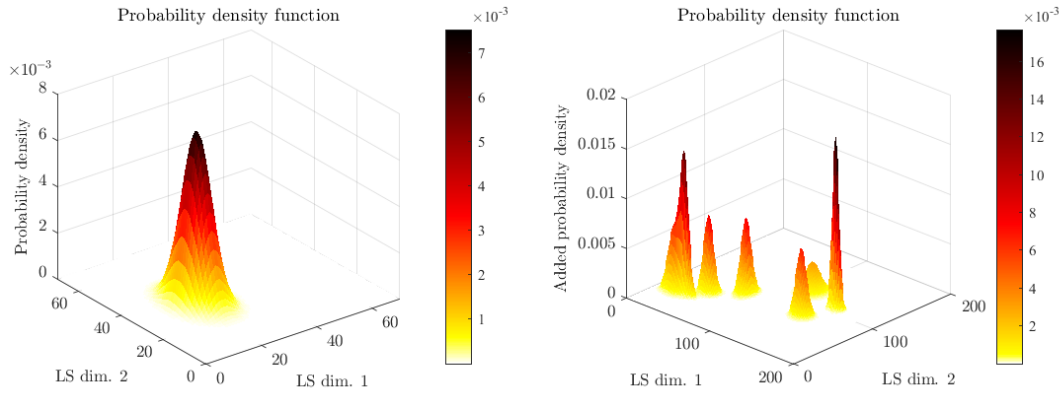
As described in chapter 5 a variational autoencoder can be used to reconstruct the input data and to detect outliers through comparing the input data and the reconstruction, but they are not only used to reconstruct the input data. A trained autoencoder is trained to keep as much information as possible to reduce the reconstruction loss. So the encoder maps the input data to representations in the latent space where the decoder can reproduce the data as good as possible. An outlier in the input data is then also an outlier in the latent space. Similar input data is close to each other in the latent space so it is possible to see class patterns. This chapter builds the main part of this thesis and proposes an new method to detect outliers via an LSTM variational autoencoder.

8.1.1 Two dimensional probability distribution

For this chapter it is important to understand what a probability distribution in a two dimensional space is. A probability distribution is a mathematical function that describes the probability of the values that a specific variable can have. In this case we assume a normal distribution. A one dimensional probability distribution can be specified by a mean value and a value for the standard deviation. In a two dimensional space, we obtain a multivariate normal probability density function with two mean values and

$$\Sigma = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{bmatrix} \tag{8.1}$$

as the matrix sigma that defines the standard deviations. The probability of the occurrence of a single normal distributed variable in a two dimensional space can be represented with a single probability density function (PDF), as shown in figure 8.1a. From a sequence of normal distributed variables we obtain a probability density function by adding the single PDFs, as shown in figure 8.1b



(a) Single PDF in a two dimensional space. The x and y axis represent the values of a two dimensional variable. The z axis describes the probability density.

(b) A sequence of normal distributed values can be visualized by adding the PDFs.

To visualize a whole dataset that consists of sequences of normal distributed values, we can visualize the representing normal probability density function for the whole dataset (figure 8.2).

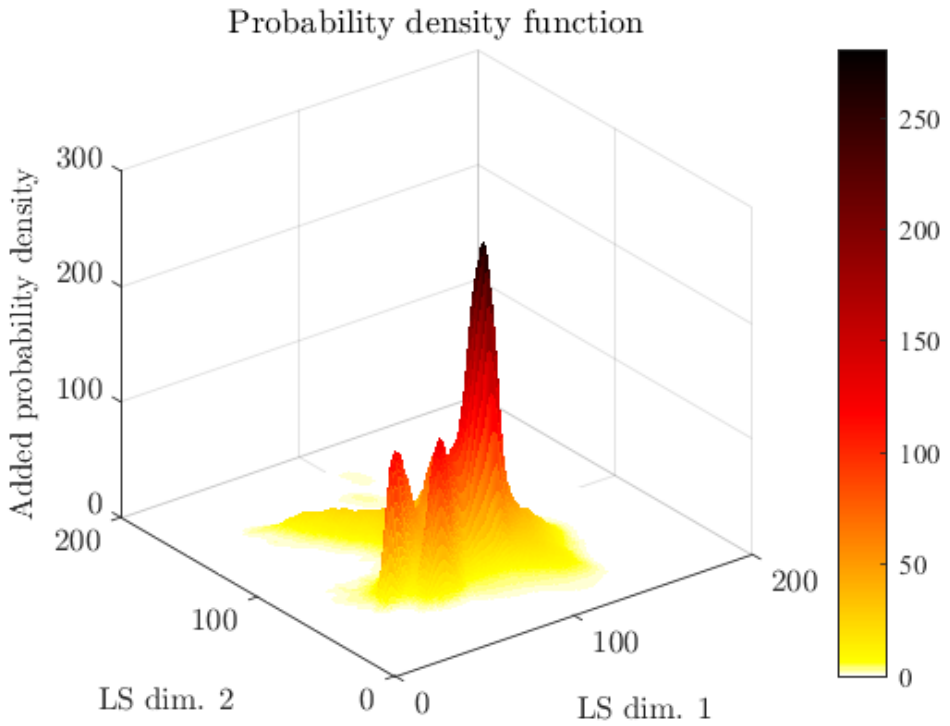


Figure 8.2: Probability distribution function of a whole dataset obtained by a summation of the PDFs from every sample. Again, the x and y axis represent the values of a two dimensional variable. The z axis describes the probability density.

8.1.2 Probability distribution in the latent space

First, the imput data gets passed through the encoding layer of the variational autoencoder. The autoencoder that was used consists of LSTM layers. As mentioned in chapter 4, the output of an LSTM layer is a sequence. That means the representation of the input data in the latent space of the variational LSTM autoencoder is a sequence, too. Because a variational autoencoder was used, the representation of in the latent space is a sequence of means and a sequence of variances for every latent dimension. In a two dimensional latent space, the two mean sequences build a trajectory and can easily be plotted for every point. In combination with the variances it is possible to compute a probability distribution for every timestep of every sample. So for every data sample, and every timestep of this sample we get a probability distribution in the latent space. The sum of this probabilities build a probability distribution function (PDF) for a whole dataset. In figure 8.3 the PDF of the exemplary data of site Seestadt Aspern is shown. The probability distributions for every timestep of every point are added and mapped to an image of size 200x200. One can easily see the areas with high and low distributions. A more detailed description of this computations can be found in the code description of this chapter.

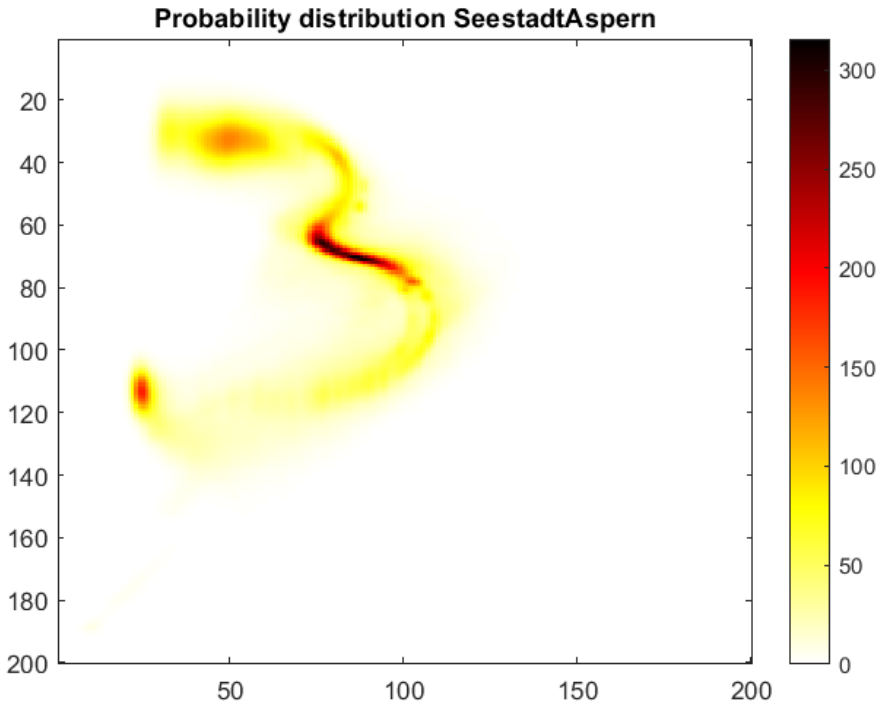


Figure 8.3: PDF of the site Seestadt Aspern. For every timestep of every data sample, a two dimensional probability distribution is computed from the two means and the two variances. This PDFs are added for the whole site. The colour indicates the added probability density of every data samples at a specific point in the two dimensional latent space.

The comparison between the timesteps of one sample and the PDF of the whole

dataset can be used to identify outliers in the input data. In figure 8.4 a normal data sample, wich means a sample that contains no anomalies compared to the other samples, in comparison to the PDF is shown. The plot of the depth data, the reconstructed depth data, the latent space representation and the probability for every timestep can be seen in 8.5. The LogVar is the logarithmic variance. The variance σ^2 is computed by

$$\sigma^2 = e^{LogVar} \tag{8.2}$$

and the furthermore standard deviation is computed from the variance as

$$\sigma = \sqrt{\sigma^2}. \tag{8.3}$$

This two equations can be transformed to

$$\sigma = e^{0.5LogVar}. \tag{8.4}$$

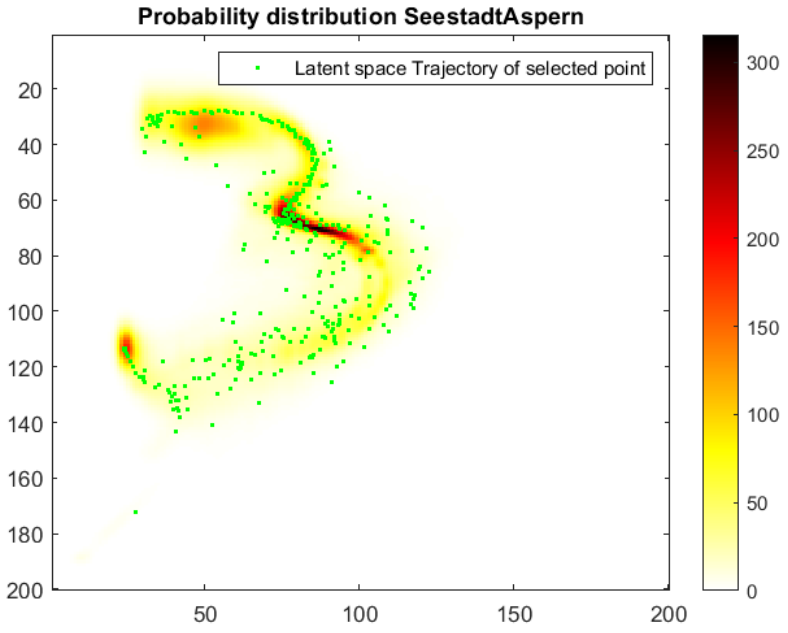


Figure 8.4: Normal point in PDF. Compared to the probability distribution function of the whole dataset, the points of the datasample lie in areas of higher probabilities, which indicates , that the data sample is non anomalous. This PDF is a sum of all PDFs for every timestep. For sample lengths of 400 timesteps, this means that this is a summation of 400 PDFs.

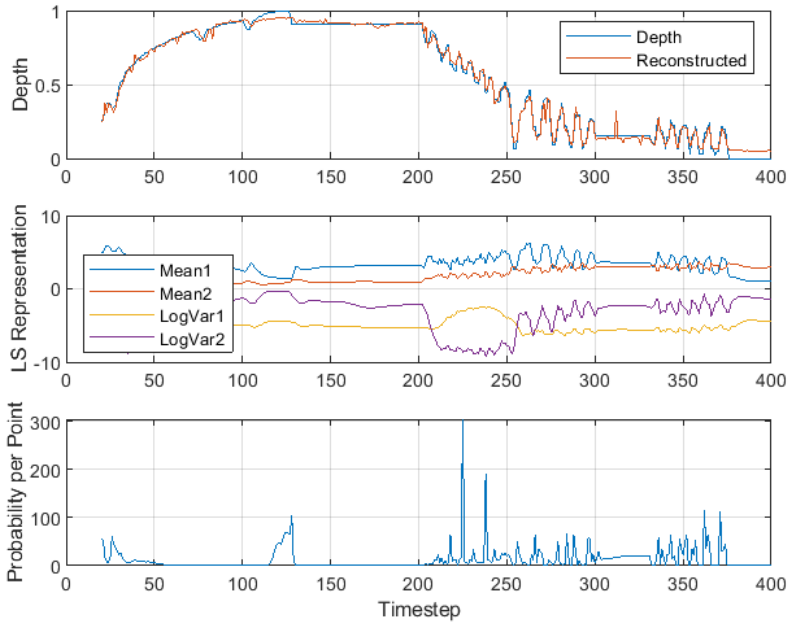


Figure 8.5: Time series plot of normal point. For every timestep of this sample the two dimensional point of the latent space representation gets compared with the PDF for the specific timestep to evaluate its probability. Top: Exemplary time series data as input and the reconstructed sequence, Middle: In a two dimensional space the latent space representation consists of two sequences of mean values and two sequences of variances, Bottom: For every timestep, the probability density was computed to evaluate the overall probability of a data sample.

In figure 8.6 an abnormal point is shown. Compared to a normal point, the latent space representation of the timesteps do not lie in areas of high probabilities. The probability of a point can be computed and it showed, that normal points have an approximately three times higher probability than a point with anomalies. Again, the plot of the depth data, the latent space representation and the probability for every point is shown in figure 8.7.

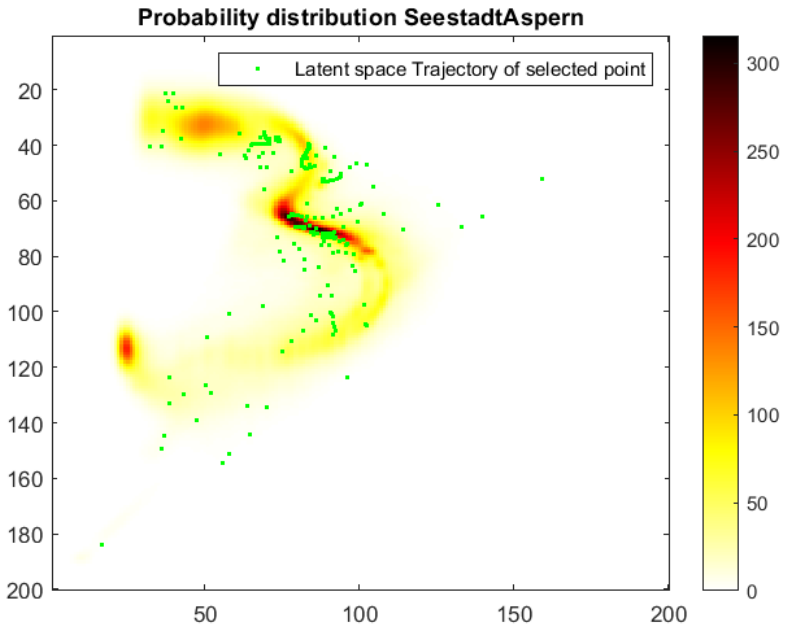


Figure 8.6: Outlier in PDF. Compared to a normal sample, outliers have many points in the latet space that lie in areas with a very low probability. These points with very low probabilities are indicators for outliers.

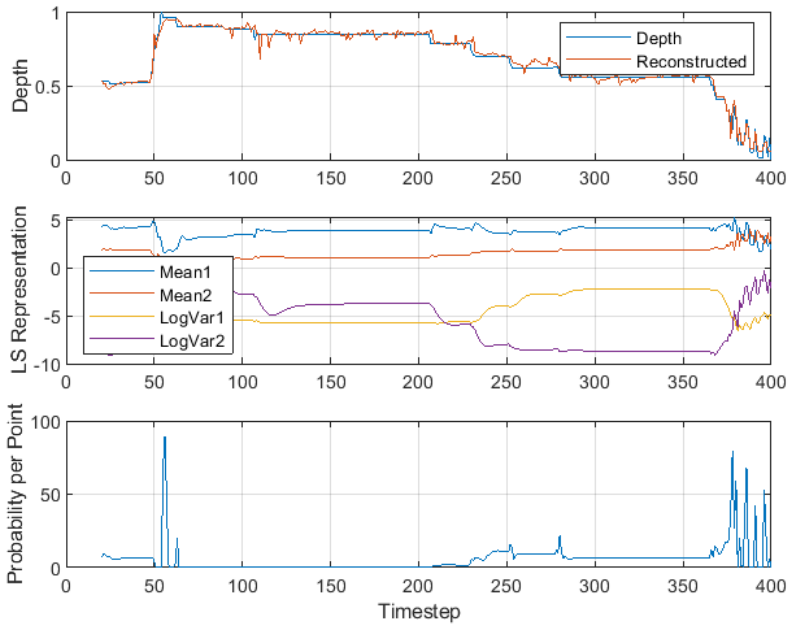


Figure 8.7: Time series plot of an outlier. The probability per point is very low here, which results in an overall low probability and indicates an outlier. Top: Exemplary time series data of an outlier as input and the reconstructed sequence, Middle: In a two dimensional space the latent space representation consists of two sequences of mean values and two sequences of variances, Bottom: For every timestep, the probability density was computed to evaluate the overall probability of a data sample. As one can see, the probabilities are low compared to the ones in figure 8.5.

An issue of comparing the datapoints of a sample with the PDF of a whole dataset is that the PDF doesn't take the PDFs of the different timesteps into account. It is possible that a timestep is in an area of high probability, although it is an outlier, because an outlier in a certain timestep can accidentally lie in an area where points of other timesteps lie very often. To fix this issue it is better practice to compute a PDF for every timestep and compare every timestep to the PDF it belongs to, like it's done in the computations of this chapter. This method led to more accurate results and it's possible to create an animation of the PDF over time. This helps to get a deeper understanding of the time dependant changes in the data and its latent space representation.

8.2 Outlier detection of a more dimensional input

The most important application of autoencoders is that that they are used to reduce the dimensionality of data while keeping as much important information as possible. The exemplary input data from the company Keller is a more dimensional time series data, as shown in chapter 6. We can take advantage of this and use this more dimensional data to detect outliers. As input, we have time series data that has more than one channel. Again, we use the exemplary data from Keller and we obtain time series data for depth, feed rate, pull down force, vibrator amperage, vibrator frequency, vibrator temperature, inclination in X and Y direction and the weight, so we have nine channels for every sample as input. If we are able to detect outliers in a two dimensional latent space, we would be able to compress the data to 22% of its initial size while keeping the information that is necessary to detect outliers in the data. A compression to 22% is quite good and would mean a massive reduction of required memory and time fore data transfer.

The latent space representations again look quite similar. An exemplary probability distribution function of the site Seestadt Aspern is shown in figure 8.8. The encoding of the depth data led to a quite comprehensible representation in the latent space and was a reflection of the overall shape of the depth data in a two dimensional latent space, whereas the probability distribution function of the whole input data is more complex.

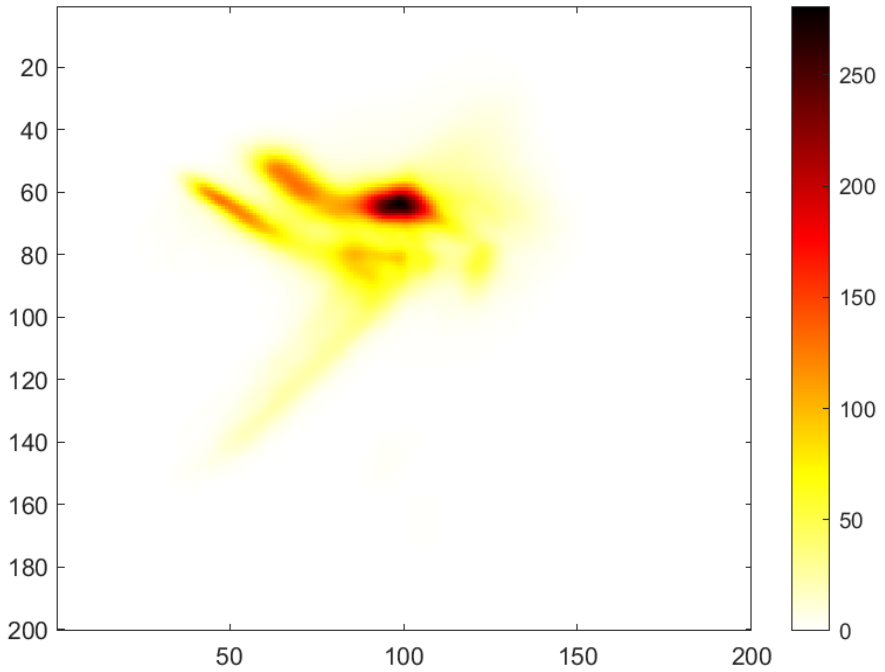
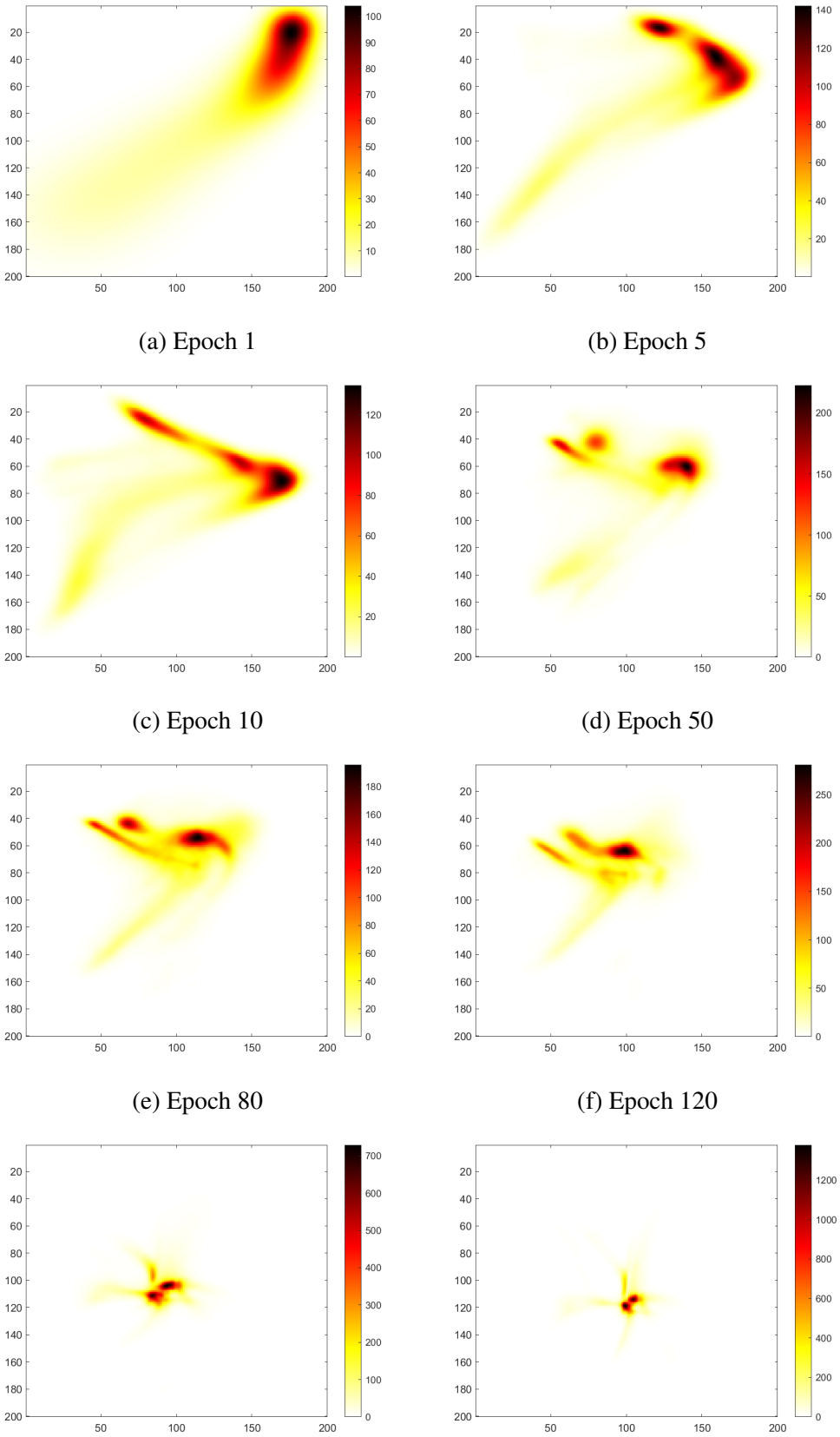


Figure 8.8: Two dimensional probability distribution function of input data with nine channels. As we increase the channels of the input data, the representations in the latent space get more abstract.

8.3 Influence of the hyperparameters on the latent space representation

The hyperparameters of our model have a huge influence on the quality of the results.

The most important parameters of an LSTM variational autoencoder are the number of training epochs, the number of hidden neurons in the encoding layer and the number of hidden neurons in the decoding layer. In figure 8.9 the influence of the number of training epochs on the probability distribution function is shown. After only a few epochs, the representation is very simple, because there haven't been much iterations to improve the weights of the model. With an increasing number of training epochs, the representations get more complex. It is interesting to see that the representations get smaller, as the number of epochs grows.



(a) Epoch 1

(b) Epoch 5

(c) Epoch 10

(d) Epoch 50

(e) Epoch 80

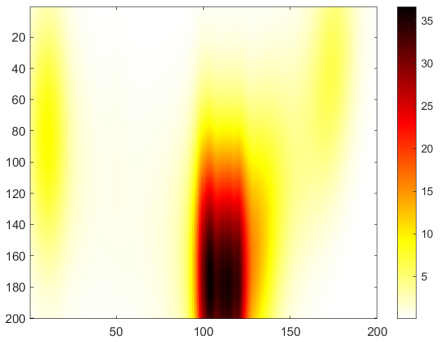
(f) Epoch 120

(g) Epoch 500

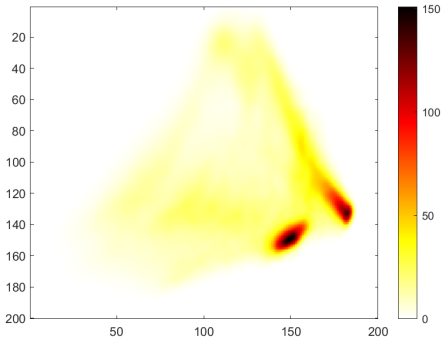
(h) Epoch 1000

Figure 8.9: Probability distribution in two dimensional latent space depending on the number of training epochs. The model was trained with 400 hidden neurons in the encoding layer and 2 hidden neurons in the decoding layer.

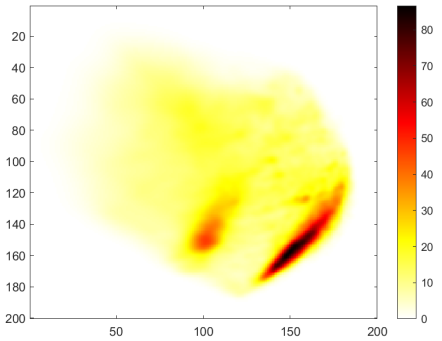
As mentioned before, the influence of the number of hidden layers in the encoder and decoder layers are also very important. The application of the models on the data showed, that it is important to have encoding layers with many hidden neurons and decoding layers with less hidden neurons. To to train a model to get complex and good representations in the latent space, it is important to give the encoder a high degree of freedom (many hidden layers) and to set the number of hidden neurons of the decoding layer to a small number. The influence of the layer parameters on the representation of the data in the latent space is shown in figure 8.10 and figure 8.11.



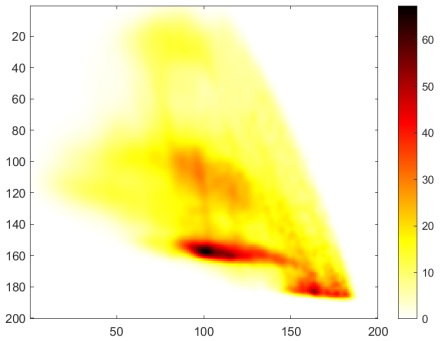
(a) Hidden neurons encoding layer: 2
Hidden neurons decoding layer: 1



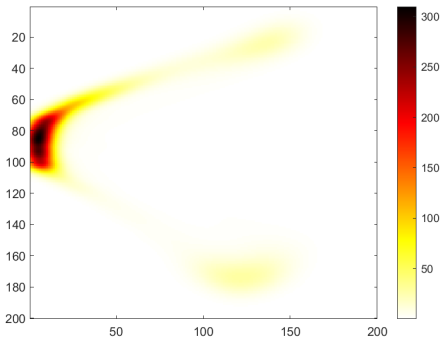
(b) Hidden neurons encoding layer: 2
Hidden neurons decoding layer: 2



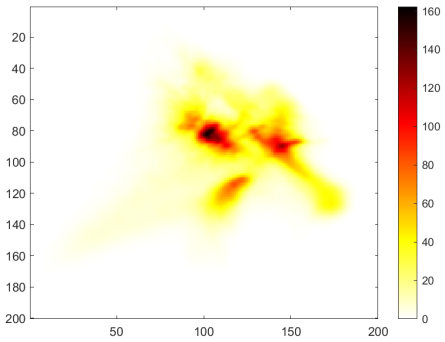
(c) Hidden neurons encoding layer: 2
Hidden neurons decoding layer: 50



(d) Hidden neurons encoding layer: 2
Hidden neurons decoding layer: 100

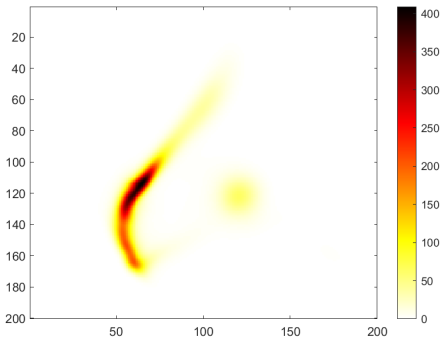


(e) Hidden neurons encoding layer: 10
Hidden neurons decoding layer: 1

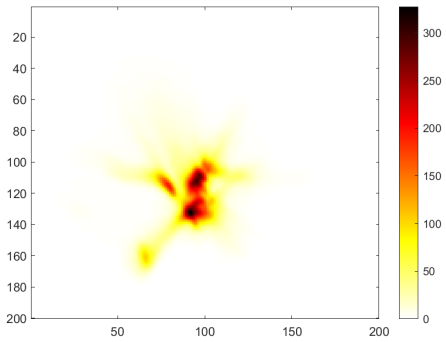


(f) Hidden neurons encoding layer: 10
Hidden neurons decoding layer: 2

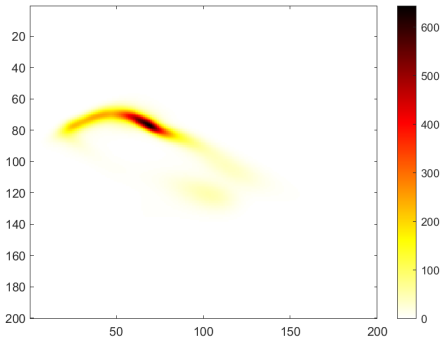
Figure 8.10: Probability distribution in two dimensional latent space depending on the number of hidden neurons in the encoding and decoding layers. A overall low number of hidden neurons leads to very bad representations (See figure a). As the number of hidden neurons increases, the representations get more complex. The model was trained for 500 epochs.



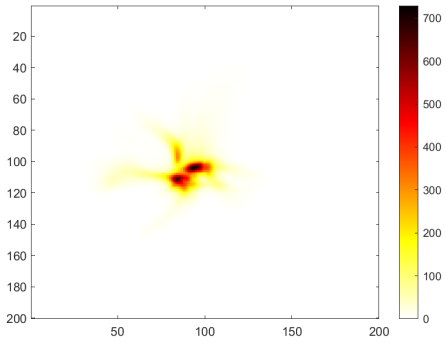
(a) Hidden neurons encoding layer: 100
Hidden neurons decoding layer: 1



(b) Hidden neurons encoding layer: 100
Hidden neurons decoding layer: 2



(c) Hidden neurons encoding layer: 400
Hidden neurons decoding layer: 1



(d) Hidden neurons encoding layer: 400
Hidden neurons decoding layer: 2

Figure 8.11: Probability distribution in two dimensional latent space depending on the number of hidden neurons in the encoding and decoding layers. A high number of hidden neurons in the encoding layer and a low number of hidden layers in the decoding layer forces the encoding layer to complex representations. The model was trained for 500 epochs.

8.4 Description of the Matlab implementations

In this section, the most important computations of the variational autoencoder computations are explained in detail. The full code can be seen in appendix A.

As already mentioned, a big amount of the time for writing the code of this computations is used for data preprocessing. For this part we assume that the data has the right input structure for the autoencoder to be able to focus on the machine learning part. At first we define a layer structure for the encoding an decoding layer. This is a very empirical task and requires much experience and many computations with different input parameters to optimize the algorithm. The influence of this parameters is already explained in chapter 3.

When all parameters are defined properly, the training of the neuronal nets can be started. For a certain number of epochs the training data gets passed trough the network

again and again. This training data is divided into smaller parts, the so called batches. After processing one of these parts, the error is computed and the network weights get updated. Thus the size of these batches has an influence on the learning performance. If the batch size is very small, the weights get updated using the information of a little amount of data, which can prevent the system from finding the minimum loss. This can be compensated by using a small learning rate. The very high batch size means a low number of weight updates and can lead to a very slow learning. Up to this point this wasn't very much new computations. The interesting computations is the sampling, the characteristic part of a variational autoencoder. The used function is shown below.

Listing 8.1: Sampling of the encoded latent space representations as input for the decoding layer.

```
function [zSampled, zMean, zLogvar] = sampling_LSTM(encoderNet, x)

%Purpose : This function performs an encoding to a given data and returns
%the mean and the variance of the VAE
%

% Input Parameters :
%   x: Matrix of the test data
%   encoderNet: the encoding network of the Variational Autoencoder
% Return Parameters :
%   zLogvar: Matrix of the variances of the test data in the latent space
%   zMean: Matrix of the mean values of the test data in the latent space
%   zSampled: Deep learning array that gets passed to the decoder during
%   training
% Author :
%    Stefan Herdy
%
% ------------------------------------------------
% (c) 2020, Stefan Herdy
% Chair of Automation, University of Leoben, Austria
% email: automation@unileoben.ac.at
% url: stefan.herdy@stud.unileoben.ac.at
% ------------------------------------------------

% Pass the input data through the encoding network
compressed = forward(encoderNet, x);

% Dividing the obtained matrix into two halfs (mean and variance matrix)
d = size(compressed,1)/2;
zMean = compressed(1:d,:,:);
zLogvar = compressed(1+d:end,:,:);
sz = size(zMean);
```

```
% Multiply every data point by a random number and the obtained standard
% devition.
epsilon = randn(sz);
sigma = exp(.5 * zLogvar);
z = epsilon .* sigma + zMean;
zSampled = dlarray(z, 'CBT');

end
```

The input data gets passed through the encoding neuronal net. The encoded data is a three dimensional hypermatrix with the dimensions [number of reduced channels (latent space dimension), number of batches, timesteps] and is split into two parts, the first half of the data is used as mean values and the second half of the data is used as standard deviation and gets multiplied by a random number to simulate a normal distribution. This sampled data gets passed to the decoder. The splitting of the data and adding of some noise is the main computation that makes an autoencoder different to a variational autoencoder. The learned latent space distributions are now used for visualization and outlier detection.

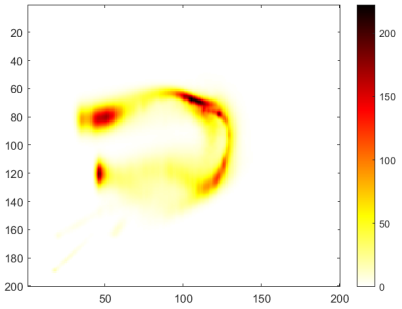
The encoded data gets passed through the decoder to compute the loss between the input and the output data and to update the weights. For visualization and to compute the probability distribution function in the latent space, the data is passed through the encoding layer and sampled again. After some data processing, again we obtain a sequence of means and a sequence of standard deviations for every dimension in the latent space. To be able to compare the timesteps the time series data gets resized before the training, so that every sequence has the same number of timesteps.

The data gets scaled from minimum to maximum, so that the mapped probability distribution function fits well into the meshgrid. A defined cell array contains a matrix of size [200, 200] for every timestep. So if the sequences got resized to a length of 500 timesteps, the cell array contains 500 matrices of size [200, 200]. For every timestep of every data sample, a probability distribution function (PDF) is computed and added to the PDF of the corresponding timestep. As a result we obtain 500 PDFs. Every single timestep of an input sequence of a geodrilling point can now be compared to this PDF to compute the probability as a measure for its outlierness.

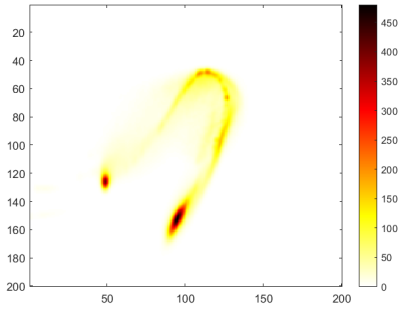
8.5 Reproducibility of the trained models and the results

An important question of this computations is the reproducibility. Due to the random initialization of the weights in the neuronal nets, the results can vary from one attempt to another. For every training the weights have different start values and the training process leads to slightly different results. The figures 8.12a and 8.12b show the results after two training attempts with the same layer parameters, the same input data and the

same number of training epochs. The representations are very similar but not the same and not reproducible.



(a) Latent Space representation A. The network was trained with 120 hidden neurons in the encoding layer and 2 hidden neurons in the decoding layer for 500 epochs.



(b) Latent Space representation B. Due to the random initialization of the weights, the latent space representation looks similar but is different to the representation A.

It is fundamental to produce results that are comprehensible and can be reproduced. The random initialization of the weights is less random than one might think. A computer can not produce absolute random numbers. It's always an algorithm that produces different numbers that appear to be random. This algorithms take a seed value to compute random numbers and if the seed value doesn't change, the so called pseudorandom numbers do not change too. In Matlab there is a built in function that makes it possible to set the random seed. This makes this pseudorandom numbers predictable and reproducible. Thus, the setting of the seed value is a very important step to make before the training of the autoencoder.

8.6 Results

The output of this computations is an added probability distribution for every sample as a measurement of weather it has anomalies or not. To evaluate how the machine learning models work, the results were compared to the analytical outlier computation. Via an inter quantile range (IQR) method, outliers in the data got computed from the defined key performance indicators (KPIs). The IQR computations were performed as follows. If we have a dataset of 2n (even) or 2n+1 (odd) values, we can compute the median of the first n values and the median of the last n values. This two medians, called quantiles, are defined as the quantiles Q1 and Q3. The interquantile range (IQR) is defined as,

$$IQR = Q_3 - Q_1. \tag{8.5}$$

The most common range for outlier detection is the $1.5 * IQR$. The lower bound for outliers is

$$Q_1 - 1.5 * IQR \tag{8.6}$$

and the upper bound is

$$Q_3 + 1.5 * IQR. \tag{8.7}$$

Every KPI of every geodrilling point was compared to the outlier boundaries to detect outliers in an analytical way. These analytical results were used to evaluate the performance of the machine learning models. To optimize the autoencoder models, a grid search was made with changing model parameters to identify the best model setting. As a measurement for the relation between the results of the autoencoder computations and the analytic computations, the correlation between both results was again computed as Pearson correlation coefficient.

$$\rho(\boldsymbol{a},\boldsymbol{b}) = \sum_{i=1}^{n} \frac{\boldsymbol{a_i} - \mu_a}{\sigma_a} \frac{\boldsymbol{a_i} - \mu_a}{\sigma_b} \tag{8.8}$$

The correlation between the optimized LSTM VAE model and the analytical results is -0.7055, which means a strong negative corellation. The PDF of the optimized model is shown in figure 8.13

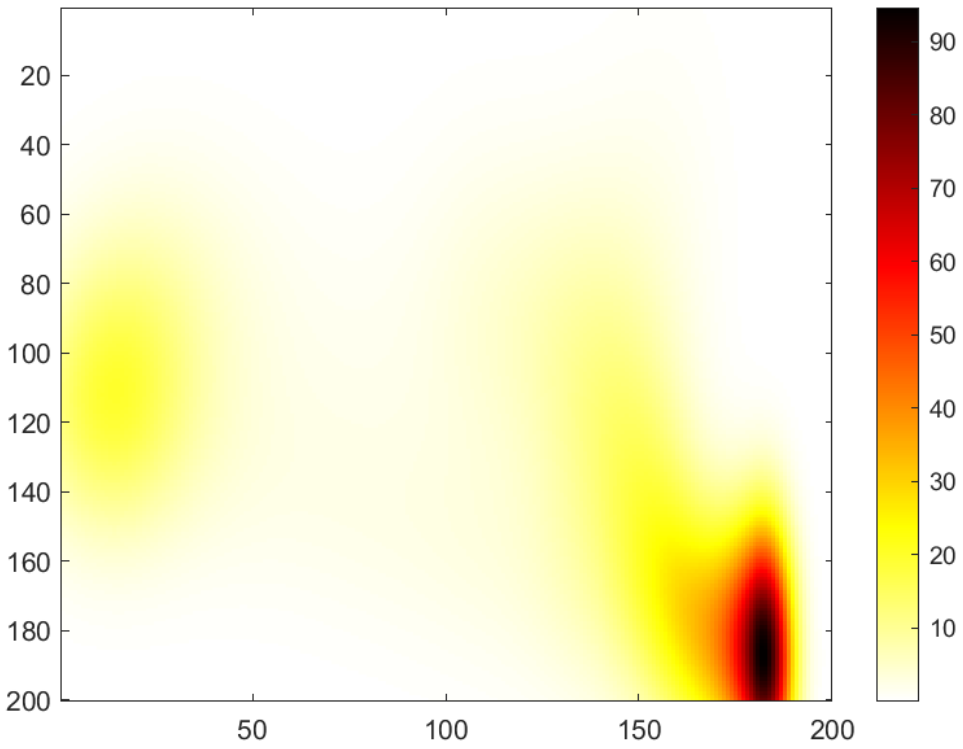


Figure 8.13: Latent Space representation of the optimized model. The main hyperparameters are: number of trained epochs = 120, number of hidden neurons in the encoding layer = 2, number of hidden neurons in the decoding layer = 2

A negative correlation may sound confusing at first, but the higher the analytical outlierness of a geodrilling point gets, the lower the probability of the point in the latent space is. This strong correlation is a proof, that it is possible to detect outliers with an LSTM variational autoencoder in the probability distribution in the latent space. The correlation between the two compared methods is visualized in figure 8.14.

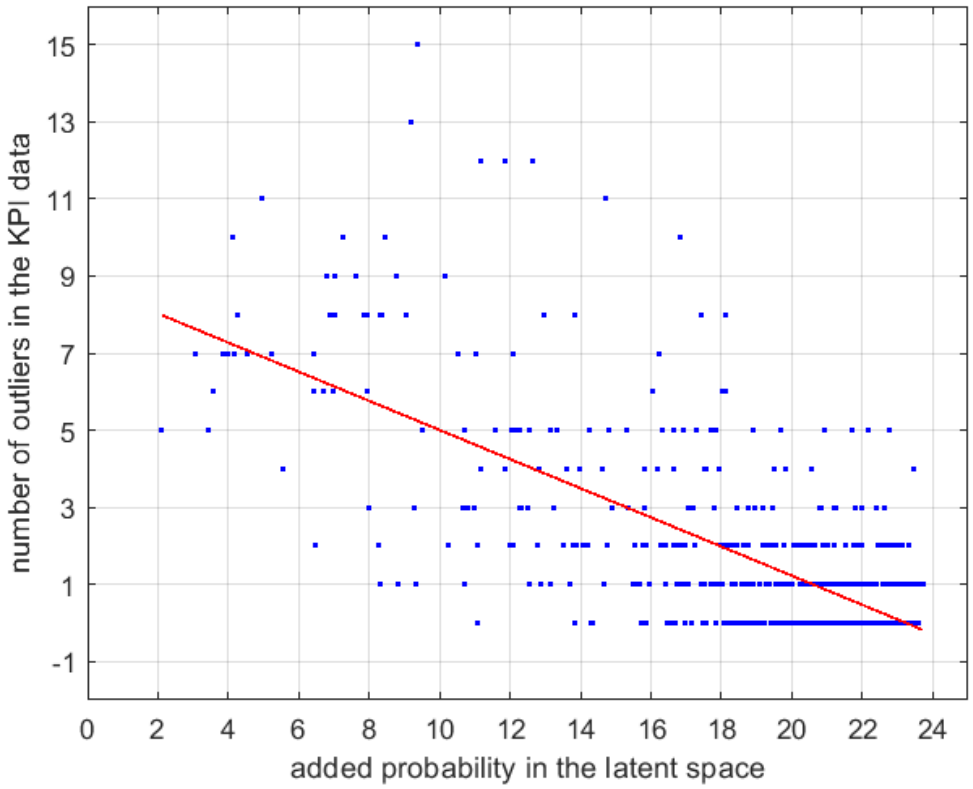


Figure 8.14: With an increasing outlierness in the KPIs, the probability is decreasing. On this figure, the correlation between the analytical outlier detection and the LSTM VAE method of the site Seestadt Aspern is shown.

Similar to the outlier detection via time series prediction, the main issue of this method is, that outliers are detected, but the model does not take into account, that different types of outliers can have very different meanings depending on the specific process. This issue is described in more detail in chapter 9.

Chapter 9

Conclusion

The main task of this work is to evaluate machine learning methods to detect outliers in time series data. A more general purpose of this thesis is to evaluate how machine learning models can be used in a reasonable way and to learn, where the limits of today's state of the art machine learning techniques lie. Machine Learning as a subdomain of artificial intelligence is promised to be one of the biggest innovations in the last few decades. There is a huge believe that machine learning and artificial intelligence in general will revolutionize the way we live fundamentally. Is this hype about machine learning justified? Is artificial intelligence really about to change our lives? Is machine learning able to fulfill tasks that humans are not able to do? This thesis gave some answers to the posed questions.

One of the most important things to clarify is, that today there is no artificial intelligence (AI) that is capable of doing what a human brain is able to do. AI, as a rather philosophical concept, became more and more an advertising name to market things better, but in reality we don't even know how we can transfer our intelligence to machines. I think that it is important to get a basic understanding of how our brain works before we start to develop intelligent machines. Machine Learning, on the other hand, takes a more technical approach and has more to do with data science. In this thesis we focused exclusively on different machine learning models and their applications. So what knowledge can be gained from the experiments and applications?

The main task was to build machine learning models that are able to detect outliers and faulty processes in time series data. This task is a typical example of something that humans are able to do with ease. A benefit of a program that is able to do that autonomously is, that a computer is able to process the data much quicker, so it would mean to save the time for checking hundreds or thousands of processes for their correctness. It was also important to develop machine learning models that are able to learn in an unsupervised way, which means that there is no need for a manual classification of a training data, because a manual classification of a part of the data to train the models would also require much time.

The two applied methods performed quite similar. Both methods were obviously able to detect anomalies in the data in an unsupervised way. There was a medium to strong correlation between the manual classification and the automated one. But there are important issues in this process.

Machine learning models need a certain data structure to be able to process it. In many machine learning tasks data preprocessing needs the most of the time. When using machine learning models to process data, the test data needs to have the same structure than the training data. To be able to use trained models over a long period of time, it is very important to ensure that the data is consistent. Well trained machine learning models can save much time of manual data analysis, but it is very important to think about the time for data preprocessing.

To get a good learning performance, sometimes it is necessary to change the data representation to improve the model performance. For a good outlier detection via time series prediction it was necessary to compute the regularized derivative and the discontinuity of the depth data to improve the model performance. The information is also in the depth data, but the model was not able to use it as good as after this computations.

The last issue is the most important. Unsupervised machine learning models are able to detect outliers with no need for labeling. The big problem is, that different types of outliers can have very different meanings. Some outliers may indicate no negative behaviour in a process, but others may indicate severe problems. In the geodrilling process, some outliers were only due to a pause of the operators. This may have some influence on the productivity, but not on the process itself. With the applied models it is not possible to detect context based outliers. It is possible in a supervised way, but we want to avoid the need to label the data. The analytical solution to detect outliers was through predefined key performance indicators (KPIs). When defining this KPIs one has to think, what the different KPIs indicate. A key performance indicator like "non productive time" is a measurement for the time, where a process was stopped. The ones who define this KPIs are humans that think of the process and of the different of abnormal occurrences. During this thesis, we did not find a way to take these contextual information into account in an unsupervised way. This last issue can be an advantage too. If we are not able to take contextual information from humans into account, the model is able to evaluate the information without the possibility of human's faults, because humans can be prejudiced by what they see and by the wanted outcome of an experiment. With a fully automated data analysis model, the evaluation of the data can not be influenced, but the interpretation of the results will always require humans which are able to set the information of the data analysis in relation to the actual process.

Appendix A

Matlab code

A.1 LSTM Time Series Prediction

A.1.1 Main Program

Listing A.1: Time Series Prediction Main Program

```
%% LSTM Prediction
%
% Description : This script is made to predict the future timesteps step by
% step. The computed loss is then used to have an "outlierness" of every
% point
% For every time stamp n  the next points n to n+t are predicted depending
% on a specific number of previous datapoints n-t, where t is an empirical
% number and the number of the previous time stamps that influence
% the prediction
%
% Author :
%    Stefan Herdy
%    m01610562
%
% Date: 30.04.2020
%  ------------------------------------------------
% (c) 2020, Stefan Herdy
%  Chair of Automation, University of Leoben, Austria
%  email: stefan.herdy@stud.unileoben.ac.at
%  ------------------------------------------------
%
%% Prepare Workspace
close all;
%clear;
% Add the path to the used functions
```

```
addpath(genpath(['..',filesep,'mcodeKellerLib']));

%% Load Data
% Ask user for site folder
%myDir = uigetdir( cd, 'Select the folder for the site');

myDir = 'C:\Users\stefa\Desktop\Masterarbeit\Code\sites\SeestadtAspern'



%% Settings
% LoadNet describes if the a trained net should be used or if the network
% should be trained on the training data
LoadNet = true;
% SaveNet describes if the trained network should be saved or not
SaveNet = false;


% Define how long the predicted sequences should be
TimeStep = 9;
% Define a Threshold for the maximum loss. If the max Loss for a point is

% above this Threshold, the data gets plotted for visual inspection
LossThreshDepth = 0;
LossThreshDisc = 0;
% Discontinuity defines wheter the discontinuity data should be used for the
% prediction or not
Discontinuity = true;
% The pahase defines wich phase should be analysed. 1 = compaction phase, 2 =
% penetration phase.
phase = 2;

if Discontinuity == true
LossThresh = LossThreshDisc;
else
LossThresh = LossThreshDepth;
end

% Call generatePredInput to load the train data
[XTrain, YTrain] = generatePredInputDisc(phase, myDir, TimeStep, Discontinuity)

%% Define LSTM Network Architecture
% An LSTM regression is used to predict a sequence of timesteps based on
% previous timesteps

inputSize = 1;
```

```
numHiddenUnits = 1500;
numResponses = 1;

layers = [ ...
sequenceInputLayer(inputSize)
bilstmLayer(numHiddenUnits)
fullyConnectedLayer(numResponses)
regressionLayer]

%% Specify the training options.

maxEpochs = 1;
miniBatchSize = 8;

options = trainingOptions('adam', ...
'MaxEpochs',maxEpochs, ...
'GradientThreshold',1, ...
'InitialLearnRate',0.005, ...
'MiniBatchSize',miniBatchSize, ...
'LearnRateSchedule','piecewise', ...
'LearnRateDropPeriod',10, ...
'LearnRateDropFactor',0.1, ...
'Verbose',0, ...
'Plots','training-progress');
%% Train the network
% Train the network based on the above defined settings
if LoadNet == false;
prednet = trainNetwork(XTrain,YTrain,layers,options);
end


s1 = int2str(TimeStep);
s2 = int2str(numHiddenUnits);
s3 = int2str(maxEpochs);

name = strcat('PredDisc_highdrop','_',s1,'_',s2,'_',s3);
name = convertCharsToStrings(name);
name = string(name);

if SaveNet == true;
save(name, 'prednet');

end

%% Test LSTM Network
% call makePrediction to test the trained or loaded LSTM network
makePredictionDisc(phase, myDir, TimeStep, prednet, LossThresh, ...
```

```
Discontinuity);

%makePrediction(phase, myDir, TimeStep, prednet, LossThresh);
```

A.1.2 Functions

Listing A.2: Generating the input for the time series prediction

```
function [XTrain,YTrain] = generatePredInput(phase,myDir, TimeStep, ...
Discontinuity)
% Purpose : Load a specified MAT file and compute the training and target
% sequences
%
% Syntax :
%
% Input Parameters :
%   phase: Defines wich phase should be analysed. 1 = compaction phase, 2 =
%   penetration phase.
%   myDir: Directory of the wanted site folder
%   TimeStep: number of TimeSteps that should be predicted
%
% Return Parameters :
%   XTrain: cell array of the training sequences
%   YTrain: cell array of the target sequences
%
% Description :
%   For every time stamp n  the next points n to n+t are predicted depending
%   on a specific number of previous datapoints n-t, where t is an empirical
%   number and the number of the previous time stamps that influence
%   the prediction
%
% Author :
%    Stefan Herdy
%
% History :
% \change{1.0}{01-Apr-2020}{Original}
%
% ------------------------------------------------
% (c) 2020, Stefan Herdy
% Chair of Automation, University of Leoben, Austria
% email: automation@unileoben.ac.at
% url: stefan.herdy@stud.unileoben.ac.at
% ------------------------------------------------
%
```

```
% Define cell arrays for output

% The timesteps n-t to n are saved in the cell array XTrain as training
% data and the timesteps n to n+t are saved in the cell array YTrain as
% target data wich is tried to get predicted

XTrain = {};
YTrain = {};

TimeStep = TimeStep;

path = fullfile(myDir, '\pointData\mat');

myFiles = dir(fullfile(path,'*.mat'));

segmentMetaData.PullDownForceMin = 5;
segmentMetaData.VibratorAmperageMin = 30;
segmentMetaData.startDepthMax = 0.5;

% Every file gets segmented file by file. For every segment, the future
% timesteps are predicted. A mean absolute error of the losss is computed
% as a measurement for the outlierness

%% Performing the Discontinuity detection
%
% In this example we wish to detect $C^1$ discontinuities.
% Additionally, the data set has a low sampling rate relative
% to the size of the features. This necessitates a short
% support length.
%
% _Define the support lengths_
lsLeft = 3;
lsRight = 3;
ls = [lsLeft, lsRight];
%%
% _Define the continuity degree_
%
% This defines the degree of continuity forced during the
% constrained approximation, e.g., $C^0$ continuity. This is
% suitable if we are trying to detect $C^1$ discontinuities.
%
continuity = 2;
%%
% _Define appropriate degrees for the polynomials_
%
% The continuity constraint plus 1 is the minimum polynomial
% degree required.
```

```
%
dLeft = continuity + 1 ;
dRight = dLeft ;
ds = [dLeft, dRight];

seq = taylorCnSequence( ls, continuity, ds );

%%
for k = 1:length(myFiles);
n = string(myFiles(k).name);
s = strcat(path,'\',n);

% Load the files
load(s);
file = load(s);

phases = segmentPhases( file.data, segmentMetaData );

values = file.data.Depth;

if Discontinuity == true;
values = conv( values, seq, 'same');
end

if phase == 1;
values = values(phases.penetrationStart:phases.compactionStart);
end
if phase == 2;
values = values(phases.compactionStart:phases.processEnd);
end

values = values';


% Padd the Depth data with zeros and the last value with the size of 'TimeStep'
% This padding is necessary to be able to predict the values from the
% first time step on

padd = zeros(1, TimeStep);
paddv(1:TimeStep) = values(end);

predval = [padd, values, paddv];

% Scale the data for a better learning performance
mu = mean(predval);
sig = std(predval);
```

```
predval = (predval - mu) / sig;


for k = 1:(length(predval)-2*TimeStep);

XTrain{end+1,1} = predval(k:k+TimeStep-1);
YTrain{end+1,1} = predval(k+TimeStep:k+2*TimeStep-1);

end

end
```

Listing A.3: Making the prediction and plotting the results

```
function  makePredictionDisc(phase, myDir,TimeStep, basenet, ...
 LossThresh, Discontinuity)
% Purpose : Load a specified MAT file and compute the test and target
% sequences
%
% Syntax :
%
% Input Parameters :
%   phase: Defines wich phase should be analysed. 1 = compaction phase, 2 =
%   penetration phase.
%   myDir: Directory of the wanted site folder
%   TimeStep: number of TimeSteps that should be predicted
%   basenet: LSTM network that should be used for the prediction
%   LossThresh: Threshold for printing the outliers
%
% Return Parameters :
%
% Description :
%   For every time stamp n  the next points n to n+t are predicted depending
%   on a specific number of previous datapoints n-t, where t is an empirical
%   number and the number of the previous time stamps that influence
%   the prediction
%
% Author :
%    Stefan Herdy
%
% History :
% \change{1.0}{01-Apr-2020}{Original}
%
% ------------------------------------------------
% (c) 2020, Stefan Herdy
% Chair of Automation, University of Leoben, Austria
```

```
% email: automation@unileoben.ac.at
% url: stefan.herdy@stud.unileoben.ac.at
% -------------------------------------------------
%
path = fullfile(myDir, '\pointData\mat');
myFiles = dir(fullfile(path,'*.mat'));


segmentMetaData.PullDownForceMin = 5;
segmentMetaData.VibratorAmperageMin = 30;
segmentMetaData.startDepthMax = 0.5;
% Get all .mat files in a struct

% Every file in the folder gets segmented file by file.
%For every segment, the future
% timesteps are predicted. A mean absolute error of the
%losss is computed
% as a measurement for the outlierness

%% Performing the Discontinuity detection
%
% In this example we wish to detect $C^1$ discontinuities.
% Additionally, the data set has a low sampling rate relative
% to the size of the features. This necessitates a short
% support length.
%
% _Define the support lengths_
lsLeft = 3;
lsRight = 3;
ls = [lsLeft, lsRight];
%%
% _Define the continuity degree_
%
% This defines the degree of continuity forced during the
% constrained approximation, e.g., $C^0$ continuity. This is
% suitable if we are trying to detect $C^1$ discontinuities.
%
continuity = 0;
%%
% _Define appropriate degrees for the polynomials_
%
% The continuity constraint plus 1 is the minimum polynomial
% degree required.
%
dLeft = continuity + 1 ;
dRight = dLeft ;
ds = [dLeft, dRight];
```

```
seq = taylorCnSequence( ls, continuity, ds );

Max = zeros(1,length(myFiles));
maxLossList = [];
outl = []
for j = 1:length(myFiles)
%for j = 1:5

n = string(myFiles(j).name);
s = strcat(path,'\',n);

% Load the files
file = load(s);


phases = segmentPhases( file.data, segmentMetaData );

values = file.data.Depth
time = file.data.Time;
if Discontinuity == true;
values = conv( values, seq, 'same');
depth = file.data.Depth;
if phase == 1;
depth = depth(phases.penetrationStart:phases.compactionStart);
end
if phase == 2;
depth = depth(phases.compactionStart:phases.processEnd);
end
end

if phase == 1;
values = values(phases.penetrationStart:phases.compactionStart);
time = time(phases.penetrationStart:phases.compactionStart);
end
if phase == 2;
values = values(phases.compactionStart:phases.processEnd);
time = time(phases.compactionStart:phases.processEnd);
end


values = values';
time = time';


% Padd the Depth data with zeros and the last value with the size of
% 'TimeStep'.
```

```
% This padding is necessary to be able to predict the values from the
% first time step on

padds(1:TimeStep) = values(1);
padde(1:TimeStep) = values(end);

predval = [padds, values, padde];

predval = [padds, values, padde];

XTest = {};
YTest = {};

mu = mean(predval);
sig = std(predval);

predval = (predval - mu) / sig;

for k = 1:(length(predval)-2*TimeStep);

XTest{end+1,1} = predval(k:k+TimeStep-1);
YTest{end+1,1} = predval(k+TimeStep:k+2*TimeStep-1);

end


% Predict on Test Data

%[net,YPred] = predictAndUpdateState(basenet,XTest);
YPred = predict(basenet,XTest);

loss = zeros(1, length(values));
pred = [];
for i = 1:length(YPred);
loss(i) = mean(abs(YTest{i,1}(1)-YPred{i,1}(1)));
pred(i) = YPred{i,1}(1);
end
% Remove the last five values from the loss vector to remove the
% influence of the loss at the end of the prozess
mloss = loss(1:end-5);
maxLoss = max(mloss)
if maxLoss>1.2;

% If the maximum mean absolute error of a prediction is above a
% certain Threshold, the data and the loss are plotted.
% The program then waits for an arbitrary input to go on with the
% computation to be able to look at the results for a desired time.
```

```
if Discontinuity == true;
x = linspace(1,length(values),length(values));
xd = linspace(1,length(depth),length(depth));
fig1 = figureGen(20,25);
subplot(2,1,1)
plot(time,values)

hold on
plot(time,loss)
plot(time,pred)
maxLossI = uint32(maxLoss*1000)
name = strcat('Discontinuity_Compaction_SeestadtAspern_High_Loss_',
int2str(j),'_MaxLoss_', int2str(maxLossI), '.png')
name = convertCharsToStrings(name);
name = string(name);
title(name)
xlabel('Time')
ylabel('Discontinuity/Loss')
legend('Discontinuity','Loss', 'Prediction','Location','southwest')
grid on
subplot(2,1,2)
plot(time,depth,'r')
xlabel('Time')
ylabel('Depth [m]')
legend('Depth')
grid on
%saveas(fig1, name)
close(fig1)
outlval = input('Outlier or not')
if outlval == 1
outl(1,j) = 1;
end
if outlval == 0
outl(1,j) = 0;
end
end

elseif maxLoss<1.2;

% If the maximum mean absolute error of a prediction is above a
% certain Threshold, the data and the loss are plotted.
% The program then waits for an arbitrary input to go on with the
% computation to be able to look at the results for a desired time.
if Discontinuity == true;
x = linspace(1,length(values),length(values));
xd = linspace(1,length(depth),length(depth));
fig1 = figureGen(20,25);
```

```
subplot(2,1,1)
plot(time,values,'k')

hold on
%plot(time,loss)
%plot(time,pred)
maxLossI = uint32(maxLoss*1000)
name = strcat('Discontinuity_CompactionPhase_SeestadtAspern_Low_Loss_', ...
 int2str(j), '_MaxLoss_', int2str(maxLossI), '.png')
name = convertCharsToStrings(name);
name = string(name);
title('Discontinuity of the depth data')
xlabel('Time')
ylabel('Discontinuity')
legend('Discontinuity','Loss', 'Prediction','Location','southwest')
grid on
subplot(2,1,2)
plot(time,depth,'k')
xlabel('Time')
ylabel('Depth [m]')
%legend('Depth')
grid on
outlval = input('Outlier or not')
if outlval == 1
outl(1,j) = 1;
end
if outlval == 0
outl(1,j) = 0;
end
%saveas(fig1,name)
close(fig1)
end
else

% If the maximum mean absolute error of a prediction is above a
% certain Threshold, the data and the loss are plotted.
% The program then waits for an arbitrary input to go on with the
% computation to be able to look at the results for a desired time.
if Discontinuity == true;
x = linspace(1,length(values),length(values));
xd = linspace(1,length(depth),length(depth));
fig1 = figureGen(20,25);
subplot(2,1,1)
plot(time,values)

hold on
plot(time,loss)
```

```
plot(time,pred)
maxLossI = uint32(maxLoss*1000)
name = strcat('Discontinuity_CompactionPhase_SeestadtAspern_Low_Loss_', ...
int2str(j), '_MaxLoss_', int2str(maxLossI), '.png')
name = convertCharsToStrings(name);
name = string(name);
title(name)
xlabel('Time')
ylabel('Discontinuity/Loss')
legend('Discontinuity','Loss', 'Prediction','Location','southwest')
grid on
subplot(2,1,2)
plot(time,depth,'k')
xlabel('Time')
ylabel('Depth [m]')
%legend('Depth')
grid on
outlval = input('Outlier or not')
if outlval == 1
outl(1,j) = 1;
end
if outlval == 0
outl(1,j) = 0;
end
saveas(fig1,name)
close(fig1)
end

end
if maxLoss < 3
Max(1,j) = maxLoss;
else
Max(1,j) = 1;
end
x = linspace(0,length(myFiles),length(myFiles));
maxLossList(j) = maxLoss;
end
save('maxLossL', 'maxLossList')
fig1 = figureGen();
plot(x,Max,'b.')
stem(x, Max, 'Marker', 'none')
grid on;
xlabel('Point Nr.')
ylabel('Maximum Prediction Error')
title('Maximum prediction loss for every point of site Seestadt Aspern')

for i=1:20
```

```
disp('Click on the wanted point for plotting')
g = ginput(1);
D = pdist2([x', Max'],g);
[~,ix] = min(D);

computePredictionDisc(phase, myDir,TimeStep, basenet, LossThresh, ...
 Discontinuity,ix)

end
end
```

A.2 Variational Autoencoder

A.2.1 Main Program

The main program of of the LSTM variational autoencoder is based on the variational autoencoder program that is provided by the MathWorks, Inc. [15]

Listing A.4: Variational Autoencoder Main Program

```
%% Variational Autoencoders
%
% Description : This script is made to apply a Variational Autoencoder to
% the Depth Data to find outliers in an unsupervised way.
%
% Author :
%    Stefan Herdy
%    m01610562
%
% Date: 13.05.2020
%  ------------------------------------------------
% (c) 2020, Stefan Herdy
%  Chair of Automation, University of Leoben, Austria
%  email: stefan.herdy@stud.unileoben.ac.at
%  ------------------------------------------------
%
%% Prepare Workspace
close all;
clear;
addpath(genpath(['..',filesep,'mcodeKellerLib']));

%% Initial Settings
% Visualization of the latent space
% If plotLatentPoints is set to true, the user can click on a point in the
% latent space and the corresponding depth data gets plotted. numPoints is
```

```
% the number of the points the user wants to plot.
plotLatentPoints = false;
numPoints = 20;

% Visualisation of the VAE reconstruction and the real depth data
% If plotHighLoss is set to true, all points with a maximum loss or a
% mean absolute loss above the defined thresholds are plotted
plotHighLoss = false;
MAEThresh = 0.1;
MaxThresh = 0.1;

% Visualization of the outliers in the Latent space
% If plotLatentOutliers is set to true, the outliers with an outlierness
% above LatentThresh in the latent space
% are plotted
plotLatentOutliers = false;
LatentThresh = 1;

% Define the phase that should be analysed.
% Is the phase is set to 1, the penetration phase is loaded. If the phase
% is set to 2, the compaction phase is loaded.
phase = 2;

% derivative: boolean value that defines wether the 1st derivative
% should be also imported or not
derivative = false;
% discont: boolean value that defines wether the nth discontinuity
% should be also imported or not
discont = false;
% CxCont: int value that defines the nth discontinuity
CxCont = 2;
% degree: degree of the Vandermonde matrix for local smoothing
degree = 3;
% kernelSize: size of the convolution matrix
kernelSize = 2;


TSLength = 400;


%% Load Data


[DataMatrix,Labels,Outlier] = generateVAEInputLSTM(phase, derivative, ...
discont, CxCont, degree, kernelSize, TSLength);
```

```
% Split into train and test data
TTsplit = 0.6;

[s1, s2] = size(DataMatrix);
split = TTsplit*s1;
split = round(split);


X_Train = DataMatrix(:,:);
X_Test = DataMatrix(:,:);



Y_Train = Labels(:,:);
Y_Test = Labels(:,:);



Outl_Test = Outlier(:,:);

YTrain = categorical(Y_Train);
YTest = categorical(Y_Test);

% Reshape the input to a 4D matrix


if derivative == true && discont == true
X_Train = reshape(X_Train', [TSLength,3, size(X_Train,1)]);
X_Test = reshape(X_Test', [TSLength,3, size(X_Test,1)]);

end

if derivative == true && discont == false
X_Train = reshape(X_Train', [TSLength,2, size(X_Train,1)]);
X_Test = reshape(X_Test', [TSLength,2, size(X_Test,1)]);
end

if derivative == false && discont == true
X_Train = reshape(X_Train', [TSLength,2, size(X_Train,1)]);
X_Test = reshape(X_Test', [TSLength,2, size(X_Test,1)]);
end

if derivative == false && discont == false
X_Train = reshape(X_Train', [TSLength,1, size(X_Train,1)]);
X_Test = reshape(X_Test', [TSLength,1, size(X_Test,1)]);
end
```

```
% Change the input to a dlarray
XTrain = dlarray(X_Train, 'TCB');
XTest = dlarray(X_Test, 'TCB');

%% Construct Network
% Autoencoders have two parts: the encoder and the decoder.
% The encoder takes an input and outputs a compressed
% representation in the latent space (the encoding),
%  which is a vector of size latent_dim.

latentDim = 2;


encoderLG = layerGraph([
sequenceInputLayer(1, 'Name', 'input1')
lstmLayer(100, 'Name', 'lstm1')
%lstmLayer(50, 'Name', 'lstm2')
%lstmLayer(100, 'Name', 'lstm3')
%lstmLayer(100, 'Name', 'lstm2', 'OutputMode', 'last')
fullyConnectedLayer(2*latentDim, 'Name', 'fc1')
]);

decoderLG = layerGraph([
sequenceInputLayer(latentDim, 'Name', 'input2')
%imageInputLayer([1 1 latentDim], 'Name', 'input2', 'Normalization','none')
lstmLayer(10, 'Name', 'lstm21')
%lstmLayer(50, 'Name', 'lstm22')
%lstmLayer(100, 'Name', 'lstm23')
%fullyConnectedLayer(2*latentDim, 'Name', 'fc2')
fullyConnectedLayer(1, 'Name', 'fc3')
]);



%%
% To train both networks with a custom training loop ,
% convert the layer graphs to |dlnetwork| objects.

encoderNet = dlnetwork(encoderLG);
decoderNet = dlnetwork(decoderLG);


%% Specify Training Options
% Train on a GPU

executionEnvironment = "auto";
```

```
%%
% Set the training options for the network.

numEpochs = 1;
miniBatchSize = 10;
lr = 1e-3;
numIterations = floor(split/miniBatchSize);
iteration = 0;

avgGradientsEncoder = [];
avgGradientsSquaredEncoder = [];
avgGradientsDecoder = [];
avgGradientsSquaredDecoder = [];
%% Train Model
% Train the model using a custom training loop.
%
% For each iteration in an epoch:

for epoch = 1:numEpochs
tic;
for i = 1:numIterations
iteration = iteration + 1;
idx = (i-1)*miniBatchSize+1:i*miniBatchSize;
XBatch = XTrain(:,idx,:);
XBatch = dlarray(single(XBatch), 'CBT');

if (executionEnvironment == "auto" && canUseGPU) || executionEnvironment ...
 == "gpu"
XBatch = gpuArray(XBatch);
end

[infGrad, genGrad] = dlfeval(...
@modelGradients, encoderNet, decoderNet, XBatch);

[decoderNet.Learnables, avgGradientsDecoder, avgGradientsSquaredDecoder] = ...
adamupdate(decoderNet.Learnables, ...
genGrad, avgGradientsDecoder, avgGradientsSquaredDecoder, iteration, lr);
[encoderNet.Learnables, avgGradientsEncoder, avgGradientsSquaredEncoder] = ...
adamupdate(encoderNet.Learnables, ...
infGrad, avgGradientsEncoder, avgGradientsSquaredEncoder, iteration, lr);
end
elapsedTime = toc;

[z, zMean, zLogvar] = sampling_LSTM(encoderNet, XTest);
xPred = sigmoid(forward(decoderNet, z));
%[x, y, z] = size(XTest);
%xPred = reshape(xPred, [x, y, z]);
```

```
%testpred = xPred(1,7,:);
elbo = ELBOloss(XTest, xPred, zMean, zLogvar);
elbo = mean(abs(elbo));
disp("Epoch : "+epoch+" Test loss = "+gather(extractdata(elbo))+...
". Time taken for epoch = "+ elapsedTime + "s")


end

%% Compute an visualize the Results
sizev = 300;
%MAEloss = computeLoss(xPred, XTest, plotHighLoss,MAEThresh,MaxThresh,sizev);



[zMean, zLogVar] = visualizeLatentSpace_LSTM_Clif(XTest, YTest, encoderNet, ...
 decoderNet, Labels, plotLatentPoints, TSLength);

%classifySVM(zMean,Outl_Test)
latentOutlierness = latentOutlier(XTest, encoderNet, ...
 plotLatentOutliers, LatentThresh,sizev);


%% Functions
% Compute the gradients of the loss with respect to the learnable paramaters
% of both networks by calling the |dlgradient| function.

function [infGrad, genGrad] = modelGradients(encoderNet, decoderNet, x)
[z, zMean, zLogvar] = sampling_LSTM(encoderNet, x);
xPred = sigmoid(forward(decoderNet, z));
%xPred = reshape(xPred, [1 10 300]);
loss = ELBOloss(x, xPred, zMean, zLogvar);
loss = mean(abs(loss));
[genGrad, infGrad] = dlgradient(loss, decoderNet.Learnables, ...
encoderNet.Learnables);
end

%%
% The |ELBOloss| function takes the encodings of the means and the variances
% returned by the |sampling| function, and uses them to compute the ELBO loss.

function elbo = ELBOloss(x, xPred, zMean, zLogvar)
squares = 0.5*(xPred-x).^2;
reconstructionLoss  = sum(squares, [1,2,3]);

KL = -.5 * sum(1 + zLogvar - zMean.^2 - exp(zLogvar), 1);
```

```
elbo = mean(reconstructionLoss + KL);
end
% Visualization Functions

function visualizeReconstruction(XTest,YTest, encoderNet, decoderNet)
f = figure;
figure(f)
title("Example ground truth image vs. reconstructed image")
for i = 1:2
for c=0:9
idx = iRandomIdxOfClass(YTest,c);
X = XTest(:,:,:,idx);

[z, ~, ~] = sampling(encoderNet, X);
XPred = sigmoid(forward(decoderNet, z));

X = gather(extractdata(X));
XPred = gather(extractdata(XPred));

comparison = [X, ones(size(X,1),1), XPred];
subplot(4,5,(i-1)*10+c+1), imshow(comparison,[]),
end
end
end

function idx = iRandomIdxOfClass(T,c)
idx = T == categorical(c);
idx = find(idx);
idx = idx(randi(numel(idx),1));
end


%% The |Generate| function tests the generative capabilities of the VAE.
% It initializes  a |dlarray| object containing 25 randomly
% generated encodings, passes them through the decoder network,
% and plots the outputs.

function generate(decoderNet, latentDim)
randomNoise = dlarray(randn(1,1,latentDim,25),'SSCB');
generatedImage = sigmoid(predict(decoderNet, randomNoise));
generatedImage = extractdata(generatedImage);

f3 = figure;
figure(f3)
imshow(imtile(generatedImage, "ThumbnailSize", [100,100]))
title("Generated samples of digits")
drawnow
```

```
end
```

A.2.2 Helper Functions

Listing A.5: Generating the input for the LSTM variational autoencoder

```
function [DataMatrix,Labels,Outlier] = generateVAEInputLSTM(phase, ...
derivative, discont, CxCont, degree, kernelSize, TSLength)
% Purpose : Load all MAT files from all site folders and compute the
% DataMatrix and the labels as input for the supervised LSTM outlier
% classification
%
% Syntax :
%
% Input Parameters :
%   phase: Defines wich phase should be analysed. 1 = compaction phase, 2 =
%   penetration phase.
%   derivative: boolean value that defines wether the 1st derivative
%   should be also imported or not
%   discont: boolean value that defines wether the nth ddiscontinuity
%   should be also imported or not
%   CxCont: int value that defines the nth discontinuity
%   degree: degree of the V-Matric for local smoothing
%   kernelSize: size of the convolution matrix
%
% Return Parameters :
%   DataMatrix: cell array containing the pont data for every point
%   Labels: array that contains the labels. The labels define the
%   "outlierness" of a point
%   Outlier: array that contains the labels. The labels define wether a
%   point is an outlier or not
%
% Author :
%    Stefan Herdy
%
% History :
%
%
% ------------------------------------------------
% (c) 2020, Stefan Herdy
% Chair of Automation, University of Leoben, Austria
% email: automation@unileoben.ac.at
% url: stefan.herdy@stud.unileoben.ac.at
% ------------------------------------------------
```

```
%% Initial Settings
% Define a Threshold for the labeling. If the number of KPIs that are
% outliers for one point are above this threshold the point is labeled as
% an outlier.
Thresh = 7;

%% Load The KPI tables

% Ask user for site folder
myDir = uigetdir( cd, 'Select the folder for the site'); %gets directory

% Load the KPIs for all points
overlapKPIpath = fullfile(myDir, '\KPI\overlapKPI');
penetrationKPIpath = fullfile(myDir, '\KPI\penetrationKPI');
pointKPIpath = fullfile(myDir, '\KPI\pointKPI');

overlap = load(overlapKPIpath);
penetration = load(penetrationKPIpath);
point = load(pointKPIpath);

%%
% Compute the IQR boundaries

% Delete the nrRoundTrips, because they are computed twice
penetration.penetrationKPIs('nrRoundTrips',:) = [];

KPIs = [overlap.overlapKPIs; penetration.penetrationKPIs; point.pointKPIs];

clear overlapKPIs penetrationKPIs pointKPIs

[r, c] = size(KPIs);
Rows = KPIs.Properties.RowNames;

lower_bounds = zeros(r,1);
upper_bounds = zeros(r,1);

for i=1:r;

values = KPIs(i,:);
values = table2array(values);
q3 = prctile(values,75,'all');
q1 = prctile(values,25,'all');

iqr = q3 - q1;
lower_bound = q1 -(1.5 * iqr);
```

```
upper_bound = q3 +(1.5 * iqr);

lower_bounds(i,1) = lower_bound;
upper_bounds(i,1) = upper_bound;



end

bounds = [lower_bounds, upper_bounds];
% Save the bounds for every KPI in a table
IQRbound = table( bounds, 'rowNames', Rows );

%% Compute the KPIs file by file and label the files

path = fullfile(myDir, '\pointData\mat');
KPIpath = fullfile(myDir, '\pointData\mat');


myFiles = dir(fullfile(path,'*.mat'));

DataMatrix = [];

Labels = [];
segmentMetaData.PullDownForceMin = 5;
segmentMetaData.VibratorAmperageMin = 30;
segmentMetaData.startDepthMax = 0.5;
% load all .mat files in struct
for k = 1:length(myFiles);
n = string(myFiles(k).name);
s = strcat(path,'\',n);

% Load the files
file = load(s);
%DataMatrix{k, 1}=file.data;

phases = segmentPhases( file.data, segmentMetaData );

values = file.data.Depth;

%    if phase == 1;
%    values = values(phases.penetrationStart:phases.compactionStart);
%    end
%    if phase == 2;
%    values = values(phases.compactionStart:phases.processEnd);
%    end
values = imresize(values, [TSLength 1], 'nearest');
```

```
%values = abs(discontinuity(values, CxCont));

depthdata = values;

if derivative == true
x = linspace(0,length(values),length(values));
yl = localSmooth(x, values,degree,kernelSize);
yl = diff(yl);
yl(1:5) = yl(6);
yl(end+1) = yl(end);
end

if discont == true
yc = discontinuity(values, CxCont);
yc(1:5) = yc(6);
end

if discont == true && derivative == false
values = [values; yc];
end
if derivative == true && discont == false
values = [values; yl];
end
if derivative == true && discont == true
error('Do not set discontunuity AND derivative to true');
end


%values = timetable2table(values,'ConvertRowTimes',false);
%values = table2array(values);
%    [s1, s2] = size(values);
%    len = 150-s1;
%    if s1>150
%        values  = values(1:150,:);
%    else
%        values = padarray(values,[len],'post');
%    end
DataMatrix(k,:) = values';
end

[s1 s2] = size(DataMatrix);

%if discont == false && derivative == false;
for i = 1:s1;
maxv = max(DataMatrix(i,:));
minv = min(DataMatrix(i,:));
```

```
for j = 1:s2;
DataMatrix(i,j) = DataMatrix(i,j)- minv;
DataMatrix(i,j) = DataMatrix(i,j)/(maxv-minv);
end
end
%end
Labels = zeros(c,1);
Outlier = zeros(c,1);

for i = 1:c;

cnt = 0;
for j = 1:r;
val = KPIs(j,i);
val = table2array(val);
if val<bounds(j,1) || val>bounds(j,2);
cnt=cnt+1;
end


end
Labels(i,1) = cnt;
if cnt > Thresh;
Outlier(i,1) = 1;
else
Outlier(i,1) = 0;
end
end
end
```

Listing A.6: computing and visualizing the results of the LSTM VAE

```
function [zMean, zLogvar, DataMtx] = visualizeLatentSpace_LSTM_Clif(XTest, ...
 encoderNet, decoderNet, TSLength)

% Purpose : The VisualizeLatentSpace function visualizes
%the latent space defined by the mean and the variance
matrices that form the output of the encoder network,
% and locates the clusters formed by the latent space
%representations of each digit.
%
% The function starts by extracting the mean and the variance
% matrices from the dlarra| objects. Then, it carries out
% a principal component analysis (PCA) on both  matrices.
% To visualize the latent space in two dimensions, the function keeps
% the first two principal components and plots them against each
```

```
% other. Finally,  the function colors the digit classes so
% that you can observe clusters.
%
% Syntax :
%
% Input Parameters :
%   XTest: Matrix of the test data
%   YTest: Matrix of the test labels
%   encoderNet: the encoding networc of the Variational Autoencoder
%   Labels: array that contains the labels. The labels define
%   the "outlierness" of a    point

% Return Parameters :
%   zLogvar: Matrix of the variances of the test data in the latent space
%   zMean: Matrix of the mean of the test data in the latent space
%
% Author :
%    Stefan Herdy
%
% History :
%
%
% ------------------------------------------------
% (c) 2020, Stefan Herdy
% Chair of Automation, University of Leoben, Austria
% email: automation@unileoben.ac.at
% url: stefan.herdy@stud.unileoben.ac.at
% ------------------------------------------------



[z, zMean, zLogvar] = sampling_LSTM(encoderNet, XTest);

[~, y, ~] = size(XTest);
%zMean = reshape(zMean, [x, y, z]);

%[x, y, z] = size(XTest);
%zLogvar = reshape(zLogvar, [x, y, z]);

zMean = stripdims(zMean);
zMean = gather(extractdata(zMean));

zLogvar = stripdims(zLogvar);
zLogvar = gather(extractdata(zLogvar));
```

```
x = linspace(1,TSLength,TSLength);

imsize = 200;

[~,l,~] = size(zMean);

DataMtx = {};

x1 = 1:200;
x2 = 1:200;
[X1,X2] = meshgrid(x1,x2);
X = [X1(:) X2(:)];


min1 = min(min(zMean(1,:,:)));
min2 = min(min(zMean(2,:,:)));

zMean(1,:,:) = zMean(1,:,:)-min1;
zMean(2,:,:) = zMean(2,:,:)-min2;

max1 = max(max(zMean(1,:,:)));
max2 = max(max(zMean(2,:,:)));

max1 = reshape(max1,[1,1]);
max2 = reshape(max2,[1,1]);

scalingFactor1 = (imsize-11)/max1;
scalingFactor2 = (imsize-11)/max2;


for i =1:l

meanv = zMean(:,i,:);
meanv = reshape(meanv, [2 TSLength]);
varz = zLogvar(:,i,:);
varz = reshape(varz, [2 TSLength]);
Mean1 = meanv(1,:);
Mean2 = meanv(2,:);
Mean1 = double(Mean1);
Mean2 = double(Mean2);
Var1 = varz(1,:);
Var2 = varz(2,:);
Var1 = double(Var1);
Var2 = double(Var2);
sigma1 = exp(0.5 .* Var1);
sigma2 = exp(0.5 .* Var2);
```

```
for k = 1:length(Mean1)
mu1 = Mean1(k)*scalingFactor1;
mu2 = Mean2(k)*scalingFactor2;
s1 = sigma1(k)*scalingFactor1;
s2 = sigma2(k)*scalingFactor2;
mu = [mu1 mu2];
sigma = [s1 0;0 s2];
y = mvnpdf(X,mu,sigma);
y = reshape(y,length(x2),length(x1));
%        DataMtx(round(mu1)+1:round(mu1)+11,round(mu2)+1:round(mu2)+11) ...
        = DataMtx(round(mu1)+1:round(mu1)+11,round(mu2)+1:round(mu2)+11)+y;
DataMtx{k} = DataMtx{k}+y;
end

%[PDFdata, xScale, yScale] = ...
%    PDF2D( Mean1, Mean2, [min(Mean1), max(Mean2), min(Mean1),...
%     max(Mean2)], [200, 200]);

end

%DataMtx(DataMtx == 0) = NaN;
fig2 = figure();
imagesc(DataMtx)
colormap(flipud(hot));
colorbar;
input('')

save('DataMatrix.mat', 'DataMtx');
save('encoderNet.mat', 'encoderNet');
save('decoderNet.mat', 'decoderNet');
save('XTest.mat', 'XTest');

end
```

Bibliography

[1] Amit Konar. *Machine Learning Techniques*. 1999.

[2] Kalidas Yeturu. *Machine learning algorithms, applications, and practices in data science*, volume 43. Elsevier B.V., 1 edition, 2020.

[3] Michel Denuit, Donatien Hainaut, and Julien Trufin. *Deep Neural Networks*. 2019.

[4] Sergios Theodoridis. *Neural Networks and Deep Learning*. 2020.

[5] Joseph Rocca. Understanding Variational Autoencoders (VAEs), 2019.

[6] Dimitar Ninevski and Paul O Leary. A Convolutional Method for the Detection of Derivative Discontinuities. 2020.

[7] Tammy Jiang, Jaimie L. Gradus, and Anthony J. Rosellini. Supervised machine learning: A brief primer. *Behavior Therapy*, 2020.

[8] Fabio Nelli. *Python Data Analytics*. Rome, 2 edition, 2018.

[9] Ludwig Fahrmeir. Regression. 2007.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[11] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. pages 1–9, 2014.

[12] Robert DiPietro and Gregory D. Hager. *Deep learning: RNNs and LSTM*. Elsevier Inc., 2019.

[13] Walter Hugo Lopez Pinaya, Sandra Vieira, Rafael Garcia-Dias, and Andrea Mechelli. *Autoencoders*. Elsevier Inc., 2020.

[14] Yang Liu, Eunice Jun, Qisheng Li, and Jeffrey Heer. Latent space cartography: Visual analysis of vector space embeddings. *Computer Graphics Forum*, 38(3):67–78, 2019.

[15] Inc. The MathWorks. Train Variational Autoencoder (VAE) to Generate Images, 2020.