Learning Representations from Crowdsourced Network Benchmarks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Telecommunications

eingereicht von

Lukas Eller, BSc.
Matrikelnummer 01325987

an der Fakultät für Elektrotechnik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Markus Rupp
Mitwirkung: Senior Scientist Dipl.-Ing. Dr.techn. Philipp Svoboda

Wien, 27. Juli 2020

Lukas Eller | Markus Rupp

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

institute of telecommunications

Learning Representations from Crowdsourced Network Benchmarks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Telecommunications

by

Lukas Eller, BSc.
Registration Number 01325987

to the Faculty of Electrical Engineering

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Markus Rupp
Assistance: Senior Scientist Dipl.-Ing. Dr.techn. Philipp Svoboda

Vienna, 27th July, 2020

Lukas Eller   Markus Rupp

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Lukas Eller, BSc.
[redacted]

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct — Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In— noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, 27. Juli 2020

Lukas Eller

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Zur Erstellung von Benchmarks von Mobilfunknetzen wird seit einigen Jahren verstärkt auf Crowdsourcing zurückgegriffen. Da Crowdsourcing per Definition nicht unter kontrollierten Bedingungen durchgeführt wird, ist es jedoch erforderlich den Kontext einer Messung zu berücksichtigen um **faire** Network-Benchmarks zu erhalten. Ausgehend von einer Sammlung an selbst-durchgeführten Messungen an einem Referenz LTE eNodeB, befasst sich diese Arbeit mit einem der kritischen Aspekte im Bereich der Kontextbestimmung - die Klassifizierung einzelner Messungen in tarif-limitert oder unlimitiert.

Dafür werden zunächst die relevanten Merkmale einzelner Messungen in einem Feature-Vector mit niedriger Dimension gebündelt. Es stellt sich heraus, dass diese Merkmale bereits eine annährend fehlerfreie Klassifizierung des Trainingsdatensatzes ermöglichen. Dieser Feature-Vector fungiert ferner als Grundlage für eine Klassifizierung basierend auf Label-Spreading. Als Semi-Supervised Algorithmus bietet Label-Spreading die Möglichkeit auch nicht-gelabelte Daten während des Trainingsprozess zu berücksichtigen. Durch die somit erhöhte Anzahl an Trainingsdaten kann eine Accuracy von 99% erreicht werden. Die Anschliessende Klassifizierung der Crowdsourcing-Daten ermöglicht die Entfernung tarif-limitierted Messungen — die übrigen Tests dienen somit als Grundlage für ein Netzbetreiber Ranking, das unabhängig von der jeweilgen Tarifstruktur ist.

Der zweite Teil dieser Arbeit beschäftigt sich mit der Verarbeitung von Crowdsourcing-Messungen mithilfe von Autoencodern. Die Verwendung von Deep-Learning-Techniken zur Verarbeitung von Network-Benchmarks in einem Unsupervised-Setup adressiert die begrenzte Verfügbarkeit von gelabelten Messungen. Die erhaltene Latent-Space Darstellung ermöglicht die Auswertung von hochdimensionalen Datensätzen und kann als Grundlage für nachfolgende Inference-Tasks dienen. Die Auswertung eines 2-D-Latent-Space zeigt, dass der Autoencoder eine Darstellung lernt, die die gesammelten Datensätze in limitierte und unlimitierte Tests unterteilt. Des Weiterene, hebt der Autoencoder bei Anwendung auf Crowdsourcing-Daten die Tarifstruktur verschiedener Mobilfunknetzbetreiber hervor.

Durch diesen zweistufigen Ansatz wird das Thema umfassend behandelt. Neben der Klassifizierung von Tarifflimitierungen in Crowdsourcing Network-Benchmarks dient die Arbeit somit auch als Fallstudie zur Durchführung von Regressions- oder Klassifizierungs-Aufgaben in einer Umgebung mit nur wenig gelabelten Datensätzen und einer begrenzter Verfügbarkeit von Parametern.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

In recent years there has been increased adoption of crowdsourced approaches in the realm of mobile network benchmarking. Compared to controlled drive tests, such approaches offer increased coverage — both in the spatial and temporal domain. However, obtaining fair network benchmarks from user measurements requires additional context information, as crowdsourced measurements are, by definition, not conducted under controlled conditions. Such context information might include tariff-limits, indoor/outdoor detection, or whether a user was static or moving during the measurement. The inference of such indicators tends to be challenging due to the limited availability of parameters and the tedious process of collecting labeled measurements.

Based on a data set I collect in a reference LTE eNodeB, this work tackles one of the critical aspects in the realm of context inference — the detection of tariff-limited measurements. This is achieved following a two-step approach: First, I process the raw measurements into a vector consisting of carefully selected features that allow for separation of the training data set almost without error. I further deploy a semi-supervised machine learning algorithm operating on this feature vector. This approach based on label spreading can also make use of unlabeled tests — thus tackling the limited availability of labeled measurements. Results show that the classifier achieves an accuracy of 99% when validated on a self-collected representative outdoor data set. After applying the classifier to a crowdsourced data set and removing the limited tests, I obtain an operator benchmark from the network view.

In a second step, I evaluate the application of autoencoders for representation learning in this field. Using deep learning techniques to process network measurements in an unsupervised setup, tackles the limited availability of labeled samples in a comprehensive way. The obtained latent space representation allows for large scale analysis of high-dimensional data sets and can act as the basis for a subsequent learning task. Evaluation of the 2D latent space shows that the autoencoder learns a representation that separates the collected data sets into limited and unlimited tests. When applied to crowdsourced data, the autoencoder highlights the tariff-structure of different mobile network operators.

By following this two-step approach, this work covers the topic comprehensively. Besides tackling the particular challenge of tariff-detection in crowdsourced network benchmark, it also acts as a case study on how to conduct inference in an environment with only a small number of labeled samples and a limited availability of parameters.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction & Motivation

Since the introduction of 1G in 1990 mobile communications has come a long way. What started as a system to handle voice calls and text messages, evolved into a network that powers a vast number of different applications [DH07]. Especially the rapid adoption of smartphones — kicked off by the launch of Apple's iPhone in 2007 — accelerated the notion of omnipresent connectivity. Nowadays, people see internet access as a basic necessity — on pair with water & electricity supply. Just a few decades ago, going online was a deliberate act. Now, in the age of smartphones and high throughput mobile communications, users simply expect to be connected at all times. This trend also fueled the rise of mobile broadband — internet access via portable modems — as an extension to established fixed broadband connections. The high number of mobile broadband subscribers reflects the user's wish for an internet connection that suits their mobile lifestyles — the access should be independent of a given location and provide increasing flexibility.

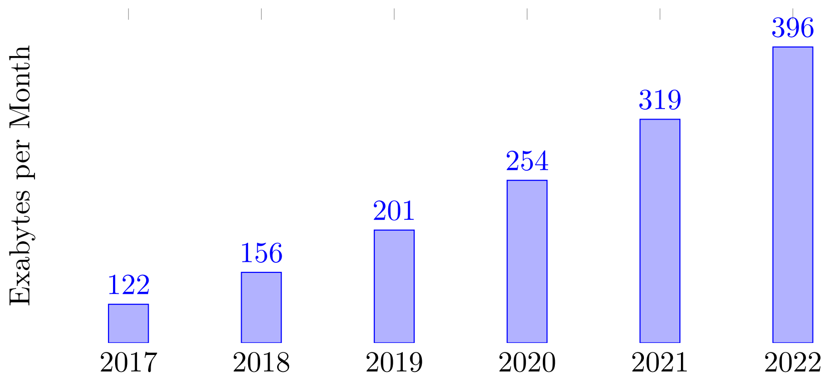


Figure 1.1: Global IP traffic growth.

The data in Figure 1.1 underpins this observation — the network supplier Cisco does not

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

expect the exponential growth of IP traffic to slow anytime soon [cis18]. A significant part of this growth will come from mobile devices — smartphones are estimated to be the second-fastest-growing device category until 2023 [cis20]. In general, mobile subscriptions are on the rise: The network supplier expects the total number of mobile subscribers to grow from 5.1 billion in 2018 to 5.7 billion in 2023 — then covering 71% of the global population. In the Central and Eastern Europe region, the predicted numbers are significantly higher. Here, Cisco expects the mobile network subscriptions to cover 81% of the population by 2023, while 78% of the regional population will have internet access.

Those figures show the extent to which our modern lifestyles are dependent on efficient and reliable communication networks. An increasing percentage of our economy is heavily reliant on a robust communication system — it is hard to imagine our intertwined world without the level of connectivity our mobile and fixed broadband networks are providing. The Covid19 pandemic in 2020 has exposed this ruthlessly — the subsequent global lockdown forced many organizations and institutions to introduce home office and remote work. Globally, a large number of people turned to video conferencing tools to replace social encounters and business trips alike. Reliable figures are not available at the time of writing. Still, several network providers report data volume increases of up to 50%. We can also expect the shift of working hours and the user migration due to the Covid19 response to have altered the utilization of network entry points and to have caused a change in the overall distribution of traffic over a given day.

Drawing generic conclusions from such exceptional circumstances is, of course, dangerous. Still, such an extreme situation acts as a window into the near future and provides valuable insights. Especially because the increase in user mobility and the growth of data volume seems to be a long-term trend anyway. To provide this additional capacity, continuous enforcement, and optimization of our existing mobile networks are required. In this context, effective benchmarking of mobile network operators ensures that potential bottlenecks and areas with insufficient service can be identified quickly and reliably.

Crowdsourcing in Mobile Network Benchmarking

In the past, benchmarking of mobile networks relied mainly on results from controlled drive- and walk tests. While those provide reliable results, the coverage of such benchmarking approaches — both in the spatial and the temporal domain — is low. This is why the use of crowdsourcing in this field has received increasing attention. In general, crowdsourcing refers to a scheme where a common problem is split into smaller subtasks, which are then individually solved by a large group of users. In the realm of mobile network benchmarking this means that individual users conduct measurements, which are then collected to provide the basis for comprehensive benchmarks. When interpreted correctly, crowdsourcing results do not only offer high-resolution coverage maps of an area of interest but can also facilitate the real-time detection of network disruptions [MS16]. In the time of omnipresent connectivity and mobile broadband connection, crowdsourcing offers another key advantage: Relying on measurements collected by the crowd ensures that the test locations do closely reflect the user behavior. Controlled drive & walk

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

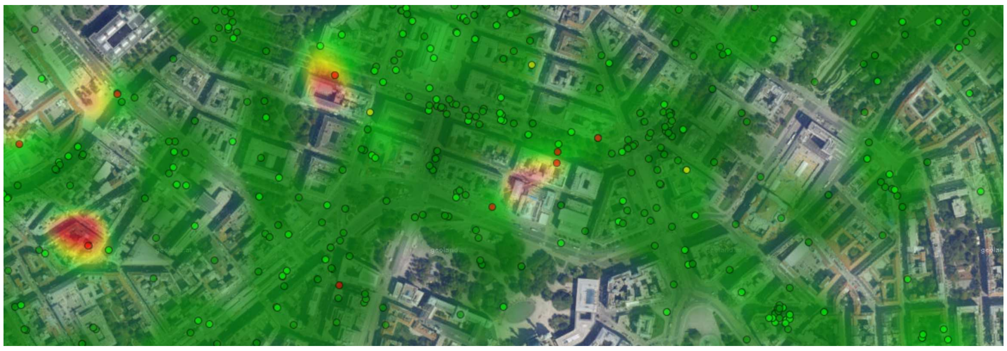


Figure 1.2: Crowdsourced Throughput Map.

tests seem out of touch with the increasing diversity of use cases — a problem which is has been addressed by industry observers and regulating authorities alike: In 2019 Connect [1], an industry observer in the DACH region started to use crowdsourced data in its benchmarks. Beforehand, the analysis was solely based on drive and walk tests. In Austria, the regulating authority RTR has also actively embraced crowdsourcing — see Chapter 2.3. It maintains RTR-Netztest, a mobile application where users can conduct their own measurements. All of those throughput-tests are then publicly available. Figure 1.2 shows a map of results from crowdsourced open data [2]. Here, each dot represents one specific LTE throughput measurement. The difference to a controlled walk and drive test is apparent — the center of Vienna is covered in a large number of tests, distributed in a way that matches representative user locations.

Can we trust the crowd?

When examining the results in Figure 1.2, we observe certain areas with low reported throughput — these areas are indicated by the red color. When looking at this depiction, it might be tempting to draw false conclusions. We might, for instance, assume that the red areas are caused by network outages, while an indoor environment with bad reception is most likely to blame. Also, we are not aware of whether the user was static or moving during the time of measurement, or whether tariff shaping altered the reported end rate. This shows that it is hard to assess whether those measurements are, in fact, representative. As opposed to controlled drive tests, we are not aware of the environmental conditions under which such measurements have been conducted.

In a nutshell, we can say that a crowdsourced test is only complete when additional context is provided or can be inferred. This observation is illustrated in Figure 1.3 — the crowdsourced tests have to be processed in order to be used as the basis for real network benchmarks. Consider, for instance, how unawareness of tariff limits can distort the results of operator rankings. In that case, operators are not credited for their network

[1] www.connect.de

[2] Results taken from www.netztest.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

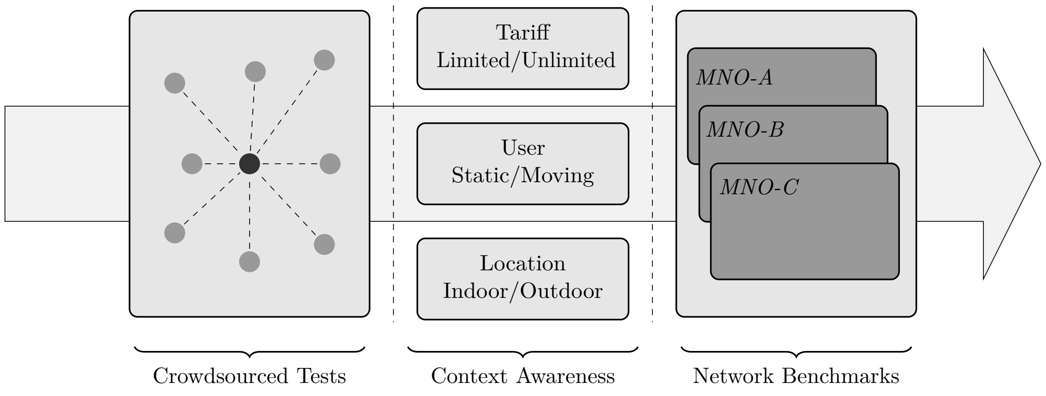


Figure 1.3: Awareness of context enables fair benchmarking.

infrastructure, but instead, for the tariff structure, they offer to customers [MS16]. Thus, tariff shaping detection is a requirement for **fair** operator benchmarking.

Inferring Context & Machine Learning

This shows that awareness of context is crucial to ensure fair and meaningful benchmarks. Still, it is often non-trivial to obtain. While GPS data allows for straightforward detection of movement — other context information usually has to be inferred indirectly. Due to the lack of meta and sensor data common in crowdsourced network benchmarking applications, this inference task can be challenging. [MS16] identified this lack of *availability of parameters* as one of core issue in this domain. To some extent, it is due to legitimate privacy concerns. Also, the collection of vast amounts of sensor-data might be contrary to the objective of crowdsourcing applications to be lightweight and non-intrusive — consider [LCZ$^+$13] and [RSR18]. Another reason is that mobile applications can often access only a subset of the available physical layer indicators. Most operating systems lack the APIs or do not provide them to developers.

All of those observations lead to the following question: ***To what extent can we infer the network view from the user view while dealing with the limited availability of parameters?***

Inference tasks with reduced availability of parameters but a vast number of available samples are a typical application of machine learning. This motivates a second question:

In what way can deep learning aid this process and what are the challenges when applying it to crowdsourced data sets?

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Outline of this Work

The objective of this work is now to address these questions in the following two ways:

1). First, by providing a robust algorithm for the classification of measurements as limited and unlimited following a machine learning approach. This scheme relies on the careful extraction of features from the raw test results to tackle this aspect of missing context.

2). Secondly, by analyzing the opportunities representation learning might bring to this field. Processing raw measurements into lower-dimensional feature vectors helps to simplify subsequent learning tasks. This can not only be of great benefit in the realm of context inference but can also act as a powerful tool for the large scale analysis of high-dimensional data sets. As such it can provide new insights and helps to reveal previously hidden connections in the crowdsourced data.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

State of the Art

2.1 Crowdsourced Mobile Network Benchmarks

Crowdsourcing, in general, has been exhaustively discussed in the literature. Several papers exist which provide a comprehensive overview of the topic and also layout some possible applications. [CKLZY12] does, for instance, discuss possible use cases and points out opportunities and challenges this relatively new approach brings with it.

[MS16] especially targets crowdsourcing in the realm of mobile network benchmarking. While the authors point out the obvious benefits of the *power of the crowd*: reduced cost, low organizational overhead, real-time monitoring, and increased coverage, they also identify possible issues that need to be addressed. Amongst others, they also bring up the problem of user measurements not necessarily qualifying as network benchmarks.

Effects of Measurement Context

Several papers also identified the three aspects of measurement context exemplary discussed in Chapter 1. In 2015 [MRB15] analyzed the effect of indoor/outdoor environments on mobile coverage benchmarks. The authors showed, that not taking the context of the measurement — indoor/outdoor —into account results in significant deviations of signal strengths. Such deviations in strengths would consequently also lead to throughput variations. The effects of movement on user reception have been measured in [JHC$^+$09] in a 3G context. To some extent, the issue of moving users is also a driving force behind non-intrusive short duration measurements, which are, for instance, discussed in [RSKR19]. The main observation here is, that a reduced measurement duration does simultaneously reduce the impact of movements — effectively reducing the distance traveled during a test. While [MS16] identified the issue of tariff limits, so far, it has not been exhaustively discussed in the literature. [RSR18] is one of the few papers covering the topic. Here the authors empirically show how the removal of tariff-limited tests changes the throughput ranking of Mobile Network Operators (MNOs). These results highlight the importance of considering tariff limits for fair operator benchmarking.

Context Inference

Several papers introduced different algorithms to infer context from measurements. Some of them are directly related to mobile network measurements while others — especially in the realm of indoor/outdoor detection — target a broader range of use cases.
An example of this is the algorithm in [IIT$^+$18], which infers the environment — indoor/outdoor — from a GPS location history. While the results seem promising, limited availability of parameters makes such approaches unsuitable for the case of mobile network benchmarks. The same is true for the scheme in [RKSM14]. Here, the authors evaluate the use of different sensors for indoor/outdoor classification. Again, access to those sensors is unlikely to be granted for network benchmarking applications. Even if this sensor information can be accessed, we can only benefit from those in future benchmarks. In the case of RTR-Netztest, there is already a significant number of measurements available. Therefore I will focus on features included in today's measurements — to extract meaningful information from those. The approach in [SMSV19] is interesting in this context — because it makes heavy use of physical layer indicators for LTE, such as Timing Advance, RSRP, RSRQ, and CQI. By using those indicators, the authors successfully separate indoor and outdoor environments.

For tariff limits, [RSR18] proposed a lightweight approach operating on RTR-Netztest throughput time series. They use a peak to average scheme, to detect the characteristic peak induced by tariff shaping — compare to Chapter 2.3. While this approach is similar to the one introduced in Chapter 3, a manual selection of the PAR threshold is required. We can assume this threshold to be dependent on the RSRP and the respective tariff limits. This work extends this scheme and makes use of machine learning techniques to tackle the problem in a comprehensive way. Moreover, the classification will rely on a set of carefully designed features that captures the time series's underlying shape. This exposed the difference between tariff shaping and effects caused by cell load or interference.

2.2 Machine Learning based Context Inference

It is interesting that most approaches, besides the one in [RSR18], utilize some kind of machine learning. The term machine learning describes a class of algorithms that learn directly from examples [Bis06]. Machine learning algorithms can be divided into three categories: supervised, semi-supervised & unsupervised learning. These subcategories differ concerning the experience of the learner.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Supervised Learning

In supervised learning, the training data consists of input samples $\mathbf{x}$ with respective targets y. Here, the learning objective is to learn the mapping from $\mathbf{x}$ to y. In [KAA+19], the authors use a supervised approach to infer the reported throughput rate of an RTR-Netztest measurement. For this prediction, they rely on a small number of samples from the beginning of the time series. Here, the objective is to reduce the overall measurement duration by allowing for a small deviation in the reported rate. This idea is directly related to other short-duration measurement schemes, such as the one in [RSKR19]. While the approach in [KAA+19] is clearly different from the use-case in this work, it still shows that there is inherent information in the throughput-time series, which can be extracted. This observation also motivates the learning of representations in Chapter 6. Another example of a supervised approach in this context is the one in [KMA+17]. Here, the authors developed a machine learning approach that can infer the respective MNO from an RTR-Netztest measurement. While this is a somewhat exotic use case — it again shows that machine learning algorithms can expose latent information from such measurements.

Semi-Supervised Learning

It is interesting to see that most of the approaches related to the inference of context rely on some semi-supervised scheme. Semi-supervised approaches differ from supervised ones by also including unlabelled data in the training set [Bis06]. This is useful when labeled examples are hard to come by — while unlabeled are available in huge numbers. Consider for instance the already discussed indoor/outdoor classifier [RKSM14], which utilizes techniques called *self-training* and *co-trainig*. A semi-supervised approach is also the basis for the classification in [SMSV19]. The success of semi-supervised approaches in this field is no surprise. Unlabelled data is, indeed, available in abundance — consider the extensive set of crowdsourced data that can be accessed at *www.netztest.at*. Obtaining labeled examples that cover a wide range of representative use cases is, however, non-trivial. Because of this, the approach in 3 will also include a semi-supervised algorithm.

Unsupervised & Representation Learning

Semi-supervised techniques can be interpreted as a hybrid of supervised and unsupervised approaches. The latter requires no labels at all — it processes the unlabeled data $\mathbf{x}$ to obtain new insights into its structure. This can take the form of clustering or also representation learning [Bis06]. The latter is commonly used for the processing of large publicly available data sets [QWD+16]. In a nutshell, representation learning simplifies subsequent learning tasks and helps to expose hidden connections in the data by reducing the dimensionality. In the literature, there are also multiple applications of representation learning on time series data. Consider for instance the following papers where representation learning operates on financial time series [BYR17] or on clinical temporal data in [YXS+19].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

To my knowledge, representation learning has not yet been applied to crowdsourced mobile network measurements. This is surprising — especially the vast amount of unlabeled data, combined with the relatively tedious process of collecting labeled examples renders its application promising in this field. Moreover, the combination of representation learning combined with a subsequent supervised learning task can be seen as an extension to semi-supervised approaches. Both process unlabeled samples and use the small number of labeled ones as the basis for the final classification. The primary difference is, however, that an expressive representation is beneficial by itself. It offers new insights into the data at hand. The success of semi-supervised approaches in the realm of crowdsourced mobile network benchmarks is clear from the literature. Thus, we can expect representation learning to be an interesting addition. Chapter 6 will exclusively focus on this idea.

2.3 Available Broadband Measurement Platforms

While a vast number of different broadband measurement platforms exists — most of them do not fulfill the requirements for this work. In particular, I aim for an application that has surpassed a critical mass of users. Only with a sufficient number of test results available can we reach a level of temporal and spatial coverage, under which the benefits of crowdsourcing become apparent. Speedtest by Ookla [1], does for sure, reach this critical number of users. In its respective category, the mobile broadband measurement application regularly ranks first in iOS and Android App Store downloads. The results of those measurements are, however, only available via a commercial API. To ensure that the methods presented in this work can be benchmarked against other approaches by an interested reader, I define open data access as another requirement for this work. Open access also ensures that anyone interested can adopt the results of this work and potentially build on top of them. RTR-Netztest fulfills both of those requirements.

RTR-Netztest

In Austria, RTR-Netztest is a well-known broadband measurement application. The fact that it is developed and promoted by the Austrian regulating authority RTR, ensures that it reaches the critical mass of users in its home market. Results from all conducted measurements are publically available — they get uploaded to the open data platform on *www.netztest.at*.

In a nutshell, it is an Android-, iOS-, Web-Application where users can conduct their own throughput measurements. The underlying technology is called RTR Multithreaded Broadband Test (RMBT) [Wim19]. While the throughput measurements are based on TCP and are thus generic over the underlying link layer protocols, the results include additional metadata and physical layer-specific information.

The meta-data provided by each measurement varies depending on the underlying technology. In the following listing, I will, therefore, only provide the features which are

[1] www.speedtest.net

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

most important in the course of this work. A detailed list of all parameters is available on the RTRs webpage [2].

- **TCP/IP**: For each throughput measurements, RMBT spawns a distinct number of parallel TCP connections — this number is three in the default configuration. Then, fixed-sized data chunks are transmitted via each of those threads. Note, that RMBT does not report the throughput time-series directly. Instead, it logs the time-stamps of arrival for each data chunk in the respective thread. From those time-stamps, I obtain the throughput time series for the downlink $\mathbf{d}$ and $\mathbf{u}$ uplink via a procedure referred to as resampling. Resampling is described in detail in [Rai17]. The author also identified the tendency of the resampling procedure to induce significant jitter onto the throughput time-series. Due to a lack of alternatives, I still rely on resampling in this work [3]. In addition to the throughput measurements, RMBT also conducts Ping measurements. The final results include a time-series of collected RTT values.

- **LTE**: While RMBT can be used to measure fixed broadband and mobile connections alike, this work does only cover mobile networks. Furthermore, I will limit the discussion to LTE measurements, as it is the underlying technology for the majority of conducted mobile tests. For the case of LTE, RTR-Netztest offers specific physical layer indicators — namely, LTE-RSRP and LTE-RSRQ. In LTE, RSRP reports the received power of the reference signal, while RSRQ can be interpreted as a cell load indicator under certain conditions [RLSR18]. Recently also timing advance has been added but is not available for all phone models. Furthermore, information on the serving LTE cell is provided. This includes the carrier frequency and additional information, such as the cell-id. Mobile Country- and Mobile Network Code are also collected — for the sim card and network alike.

- **Meta-Data**: Additionally, each RTR-Netztest includes meta-data not directly related to mobile networking. Besides the time and location of the measurement, the phone model, operating system, and other data are collected. Location data is provided in the form of a time-series — typically, we receive about 8 location updates during a single measurement. Those updates include the current speed, GPS coordinates, and altitude of the device. Netztest also reports the source of the location information — network vs. GPS — and estimates the accuracy of the respective updates.

The throughput time-series $\mathbf{d}$ plays an essential role throughout this work. In Chapter 3, it is, for instance, the shape of $\mathbf{d}$, which allows for robust classification in limited and unlimited tests. To understand how tariff shaping affects $\mathbf{d}$, it is crucial to be aware of the distinct phases of an RMBT measurement. In total RMBT consists of 7 distinct,

[2]www.netztest.at/en/OpenDataSpecification.html
[3]The resampling code used in this work is provided in Appendix A.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

nonoverlapping steps: 1) initialization; 2) downlink pretest; 3) latency test; 4) downlink RMBT; 5) uplink pretest; 6) uplink RMBT; and 7) finalization. Those phases are not apparent when only looking at the final test results reported by a distinct measurement. Still, the pretest does affect the primary downlink test — Figure 2.1 exposes this.

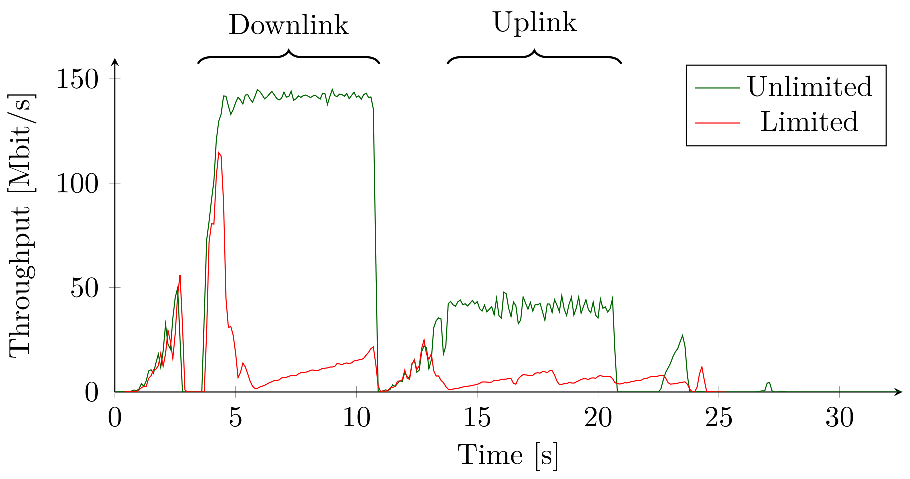


Figure 2.1: PCAP trace of RMBT test.

The throughput time series in 2.1, are reconstructed from PCAP traces of a complete RMBT measurement. This way, also the pretests are recorded. Usually, only the primary down-, and uplink-tests — highlighted in 2.1 — are reported, with lower temporal resolution. One of the throughput time series in Figure 2.1 was collected with an unlimited sim-card, while for the other one, there was a tariff limit of 20MBit/s in place. When looking at the unlimited series, we can clearly see the effect of the pretest. Here, the ramp-up phase of TCP is triggered so that the final throughput is reached as soon as the primary downlink test starts. During the pretest, limited and unlimited tests do not differ from each other. It is only after the beginning of the downlink test that the tariff shaping activates, and the series is throttled to the throughput specified by the respective tariff. The characteristic shape of a limited test — the induced peak in throughput — is the basis for the tariff detection in Chapter 3. Note that the pretest duration has a significant effect on the shape of the final **d**– a longer pretest would lead to a time-series without the abrupt decline in throughput. While the peak would still be present, it would not be recorded by RMBT — as it would occur during the pretest.

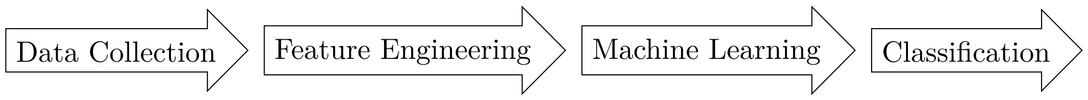
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Semi-Supervised Classification of Tariff Limits

The classification of measurements from RTR-Netztest into limited/unlimited tests has already been discussed in 2.1. In a broader sense, this chapter tackles one aspect of the effort of transforming user measurements into network benchmarks by inferring the context of a particular test result. In contrast to other environmental conditions — moving/static, indoor/outdoor —, which we can expect to be equally distributed amongst network operators, tariff shaping is directly determined by the contracts operators offer to their costumers. This renders tariff detection a key aspect in the realm of fair operator benchmarking — it ensures that operators are credited for their network layout and not their tariff structure.

In the following chapter, I will present a semi-supervised scheme for tariff-limit classification. Hereby, classification refers to a — generally machine learning — process where an input feature vector $\mathbf{x}$ is mapped to one of K discrete classes C_k [Bis06, Chapter 4]. In this particular case, we have $K = 2$ where C_0 describes the unlimited measurements, while C_1 refers to the limited tests. Overall the procedure follows 5 distinct steps:



First training and test data sets are collected — hereby, a reference cell in a lab ensures that measurements can be conducted automatically over a wide range over RSRP. In a second step the raw measurements are processed into a feature vector $\mathbf{x}$. This reduces the dimensionality of the problem. In the third step, the machine learning model — namely label spreading — operating on this feature vector $\mathbf{x}$ is trained and validated. In a final step, I apply this model to a crowdsourced data set to come up with an operator ranking from the network view.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

In the overall context of this work, this chapter also acts as a link from classical manual feature engineering to automated representation learning — see Chapter 6.

3.1 Data Collection

As mentioned in Chapter 2.1 any machine learning technique — besides unsupervised approaches — relies on labeled training data. As the crowdsourced data does not include any tariff shaping indicator, separate data collection is required. I did, therefore, conduct my own measurements with limited and unlimited sim-cards. Thereby, it is crucial to cover the same spectrum of environmental conditions included in the crowdsourced measurements. If the training data set does not closely reflect the crowdsourced data, the algorithm cannot generalize from the training samples. Thus, the final classification in 3.4 would fail.

To account for the broad spectrum of conditions, the collected data sets come from three different sources: Training data from a reference cell, one set of outdoor measurements under high cell load, and one under high interference. Each data set consists of $\{\mathbf{r}_n, y_n\}_n^N$, where N is the cardinality of the set. $\mathbf{r}$ are the raw measurement results and y their respective labels. Here, y is 1 if $\mathbf{r}$ comes from a limited test and 0 otherwise.

3.1.1 Reference Cell

As I do have access to a reference LTE eNodeB in a lab environment, the majority of the training data is collected there. The measurement setup is depicted in Figure 3.1.

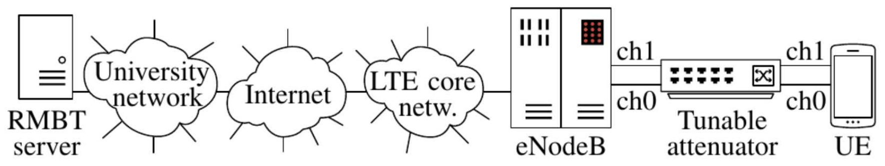


Figure 3.1: Measurement setup.

While the cell itself is fully controllable and only serves one active UE, the measurements still capture fluctuations induced by the core network of a major Austrian MNO, which the cell is connected to. To collect measurements over a representative range of receive power, the UEs' antennas are hard-wired to the eNodeB via a tunable attenuator. The attenuator itself can be triggered remotely, which allows for automated collection of samples. The server side RMBT application is hosted at the institute of telecommunications — it runs Open-RMBT an open-source variant of RMBT.

I denote this data set by $T_{lab} = \{\mathbf{r}_n, y_n\}_{n=1}^N$, here $N = 5166$. Half of the samples are limited, with a tariff limit of 10 Mbit/s in place. The other half is only limited by the receive power — this is due to the lack of any interference or additional load inside

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

the reference cell. Overall T_{lab} covers a RSRP range from -60 dbm to -130 dbm. The data set is visualized in Figure 3.2 — the plot shows throughput over RSRP. The limited and unlimited tests are clearly separated in this depiction. While the limited tests never exceed the 10Mbit/s limit — even for high RSRP values — the throughput increases for the unlimited ones before it saturates at 150 Mbit/s.

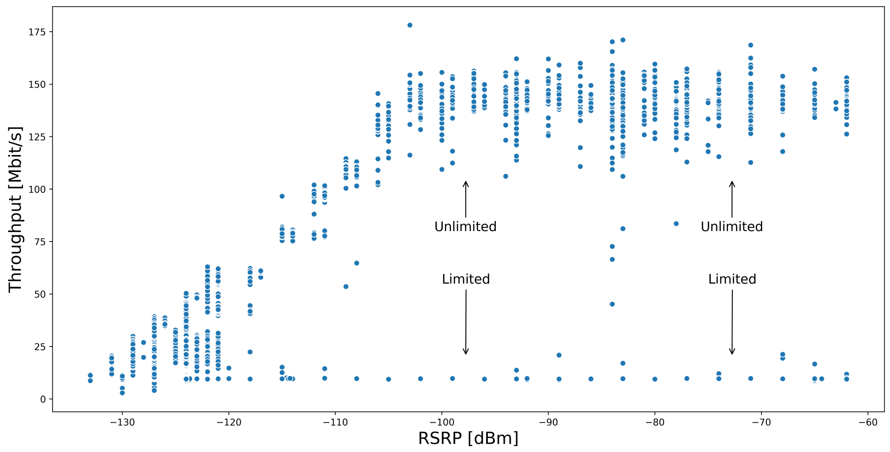


Figure 3.2: Throughtput over RSRP.

When looking at Figure 3.2, it might be tempting to classify the measurements according to some throughput over RSRP ratio. Such an approach would, however, not generalize well as it does not incorporate other factors such as cell load or interference. Thus, a classifier relying on those two features would only perform well on the training data set collected in a lab environment.

3.1.2 Operational Outdoor Cell

While the reference cell is ideal for collecting measurements over a wide range of RSRP, capturing the influence of cell load or interference is non-trivial in a lab setting. Therefore I collected additional outdoor measurements to validate the final classifier under non-lab conditions. All measurements were taken in the summer of 2019 in the city of Vienna.

To capture the influence of interference, I collected one set of measurements at the rooftop of the institute of telecommunications. Here, a significant number of interfering cells are present. This high-interference validation data set is denoted as

$$V_{\text{I10}} = \{\mathbf{r}_n, y_n\}_{n=1}^{M_{\text{I10}}} \text{ and } V_{\text{I20}} = \{\mathbf{r}_n, y_n\}_{n=1}^{M_{\text{I20}}}.$$

Here, M_{I10} and M_{I20} are both 200. I collected the second set of validation measurements at Viennas main station during rush hour, in order to capture the effects of high and varying cell load. This set consists of

$$V_{\text{CL10}} = \{\mathbf{r}_n, y_n\}_{n=1}^{M_{\text{CL10}}} \text{ and } V_{\text{CL20}} = \{\mathbf{r}_n, y_n\}_{n=1}^{M_{\text{CL20}}}.$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Here again, M_{CL10} and M_{CL20} are 200. All four measurement sets consist of 50% limited and unlimited tests — the tariff limits are either 10 Mbit/s or 20 Mbit/s. A complete overview of all data sets is given in table 3.1.

	Limit	**Data Set**	**Sample Size**	**Description**
Training	10 Mbit/s	T_{lab}	5166	reference cell, RSRP range
Validation	10 Mbit/s	V_{I10}	200	high interference, rooftop
		V_{CL10}	200	high cell load, train station
	20 Mbit/s	V_{I20}	200	high interference, rooftop
		V_{CL20}	200	high cell load, train station

Table 3.1: Overview of measurement sets.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

3.2 Feature Engineering

In machine learning, it is common practice not to use the raw data $\mathbf{r}$ directly, but to carefully select a subset of $\mathbf{r}$ and encode it into a so-called feature vector. This feature vector $\mathbf{x}$ is then used as the input to the algorithm. This process is referred to as feature selection — an area in the broader field of feature engineering [ZC18]. The objective of feature selection is generally to reduce the dimensionality of the input data, which increases the performance of many algorithms. Feature generation can be seen as an extension of this — here the features in $\mathbf{x}$ are not directly a subset of $\mathbf{r}$, but obtained via an additional processing step. Feature generation can further increase performance by representing the data in a form that simplifies subsequent learning tasks. The generation and selection as features also acts as a handle to guide the learner towards a specific direction — it determines the properties of the data set picked up by the algorithm.

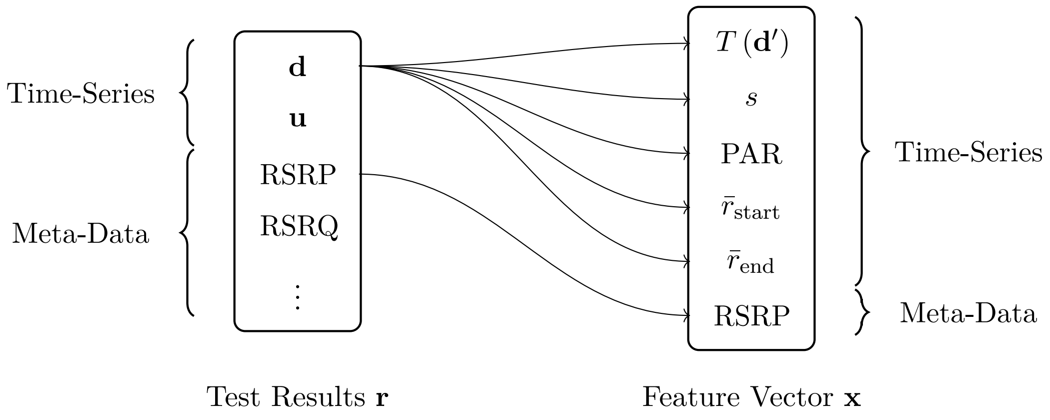


Figure 3.3: Feature Extraction.

Figure 3.3 depicts the feature engineering conducted in this specific use case. The raw test results $\mathbf{r}$ are encoded into the feature vector $\mathbf{x}$ [1]. Hereby, the RSRP is the only meta-data used in $\mathbf{x}$. The downlink time series $\mathbf{d}$ is encoded into 4 distinct features: rate at the beginning $\bar{r}_{\text{start}}$, end rate $\bar{r}_{\text{end}}$, PAR, skew s and the test statistic $T(\mathbf{d}')$. This reduces the dimensionality of $\mathbf{d}$ from 70 to 5.

Derived Feature 1: Test Statistic

The test statistic $T(\mathbf{d}')$ aims to extract the shape of a given downlink time series $\mathbf{d}$ and to encode it into a single scalar feature.

This intention is depicted in Figure 3.4. Besides the simple reduction of the dimensionality of $\mathbf{d}$, I also aim for a $T(\mathbf{d}')$, which clearly separates limited and unlimited time series. To achieve this, I make the assumption that $\mathbf{d}$ can be modeled as a multivariate gaussian with distinct distributions for the limited and unlimited case:

[1]The feature engineering code is provided in Appendix A

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

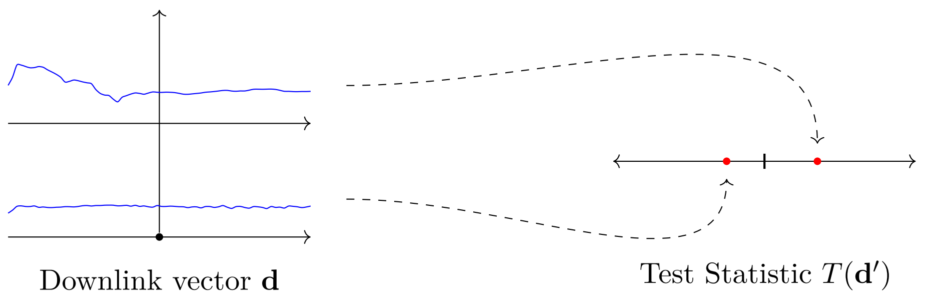


Figure 3.4: Test statistic intuition.

$$\begin{aligned} \mathcal{H}_1 : \quad & \mathbf{d} = \mathbf{d}_{\text{limited}} \sim \mathcal{N}\left(\boldsymbol{\mu}_l^{(\text{RSRP})}, C_l^{(\text{RSRP})}\right) \\ \mathcal{H}_0 : \quad & \mathbf{d} = \mathbf{d}_{\text{unlimited}} \sim \mathcal{N}\left(\boldsymbol{\mu}_u^{(\text{RSRP})}, C_u^{(\text{RSRP})}\right). \end{aligned} \tag{3.1}$$

Under this assumption I can define a log-likelihood ratio, which can be used as the basis for the test static $T(\mathbf{d}')$:

$$\ell(\mathbf{d}) = \log \frac{p(\mathbf{d}|\mathcal{H}_1)}{p(\mathbf{d}|\mathcal{H}_0)} \tag{3.2}$$

Note, that the moments $(\boldsymbol{\mu}_l, C_l)$ and $(\boldsymbol{\mu}_u, C_u)$ in Equation 3.1 are dependent on the RSRP. Thus, I would have to estimate the moments for each RSRP range and implement a lookup table for $T(\mathbf{d}')$. Empirical analysis of the data does, however, show that a simple normalization of the time series $\left(\mathbf{d}' = \frac{\mathbf{d}}{\max(\mathbf{d})}\right)$ reduces the RSRP dependency of the moments to such a level — where I can estimate them over the whole range of receive power. More precisely I estimate $(\boldsymbol{\mu}_l, C_l)$ and $(\boldsymbol{\mu}_u, C_u)$ over all of T_{lab}. The overall log-likelihood based test statistic is then given by the following expression:

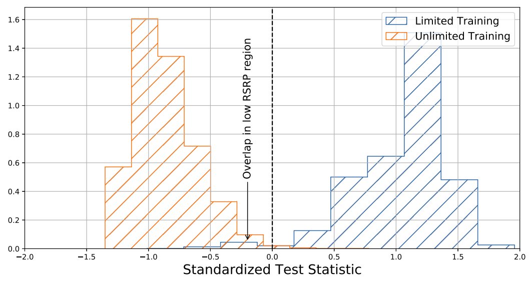
$$\begin{aligned} T(\mathbf{d}') = & \left(\mathbf{d}' - \boldsymbol{\mu}_u\right)^T C_u^{-1} \left(\mathbf{d}' - \boldsymbol{\mu}_u\right) \\ & - \left(\mathbf{d}' - \boldsymbol{\mu}_l\right)^T C_l^{-1} \left(\mathbf{d}' - \boldsymbol{\mu}_l\right) \\ & + \log\left(\det\left(C_u C_l^{-1}\right)\right) \end{aligned} \tag{3.3}$$

While the estimation of the moments over the complete RSRP range in T_{lab} is an approximation, validation results suggest that it is justified. This is depicted in Figure 3.5.

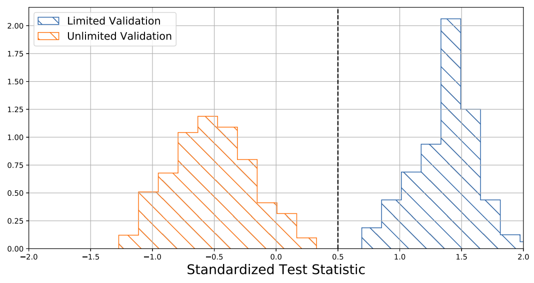
Figure 3.5a shows a normed histogram of the standardized test statistic applied to T_{lab}. Besides a small overlap — for low RSRP regions around -130dBm — $T(\mathbf{d}')$ seperates the data set perfectly. The overlap for low RSRP regions is not surprising — for low receive power the distinction between limited and unlimited tests is ambigous anyway.

While the test statistic is based on the moments of T_{lab}, it still generalizes to some extent. A histogram of $T(\mathbf{d}')$ applied to a validation data set consisting of V_{I10} & V_{CL10}

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) Training data set.



(b) Validation data set.

Figure 3.5: Standardized Test Statistic Histograms.

is depicted in Figure 3.5b. Here, the test statistic separates the two classes perfectly. This is most likely due to the very narrow range of RSRP values that the validation measurements were collected under. Note, however that the treshold differs from Figure 3.5a — for 3.5b the decision surface would have to be located around 0.5, while it is 0 for 3.5a. An explaination for this might be the influence of fluctuations induced by cell load and interference which are present in V_{I10} & V_{CL10}, but absent in the reference cell measurements from T_{lab}. This shows, that a classification reliant only on $T\left(\mathbf{d}'\right)$ is not sufficient — but additional features have to be considerd.

Derived Feature 2: Rate Start & End

I define $\bar{r}_{\mathrm{start}}$ and $\bar{r}_{\mathrm{end}}$ as the throughput at the beginning and the end of the downlink time series $\mathbf{d}_n$. They are given by the following equation:

$$\bar{r}_{\mathrm{start}} = \frac{1}{10}\sum_{k=1}^{10}\mathbf{d}_n[k] \qquad \bar{r}_{\mathrm{end}} = \frac{1}{50}\sum_{k=20}^{70}\mathbf{d}_n[k] \tag{3.4}$$

The use of this feature is motivated by the observation, that the token bucket tariff shaper does not affect the beginning of the time series — see Figure 2.1. I can therefore assume, that $\bar{r}_{\mathrm{start}}$ is only slightly affected by the tariff limit, while $\bar{r}_{\mathrm{end}}$ does exactly report the tariff limit — given that the RSRP is sufficient and cell load and interference are low.

Derived Feature 3: Peak to Average Ratio

The use of the PAR for tariff shaping detection is inspired by [RSR18]. It is based on the observation that tariff shaping via a token bucket algorithm leads to a significant peak at the beginning of a throughput series — given a sufficiently high theoretically achievable rate. This effect was already shown in Figure 2.1. Again, the peak is caused by the activation of tariff shaping after a specific data volume is transmitted, and the subsequent abrupt decline in throughput. The PAR is calculated as the ratio of maximum

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

and average throughput in $\mathbf{d}$:

$$\mathrm{PAR}_n = \frac{\max \mathbf{d}[k]_n}{\bar{d}_n}. \tag{3.5}$$

Note, however, that the PAR is not designed to capture the shape of a throughput measurement directly — as opposed to $T(\mathbf{d}')$. Therefore, it is not only sensitive to throughput declines induced by tariff limits, but also to sudden changes of cell load or interference. Without the use of additional features, such effects can not be distinguished by the classifier.

Derived Feature 4: Skew

In statistics, the skew is a well-known measure for the asymmetry of probability distributions. The skew s is 0 for the case of a symmetric probability density function while $|s| > 0$ describes an asymmetry. The orientation of the asymmetry is thereby indicated by the sign of s. Intuitively, a positive skew indicates a distribution where the right tail of the pdf is longer, and most mass is concentrated on the left, while the opposite is true for a negative skew. The skew is calculated as the ratio of central moments — of order 2 and 3 — of a scalar random variable.

$$s = \frac{m_3}{m_2^{3/2}} \tag{3.6}$$

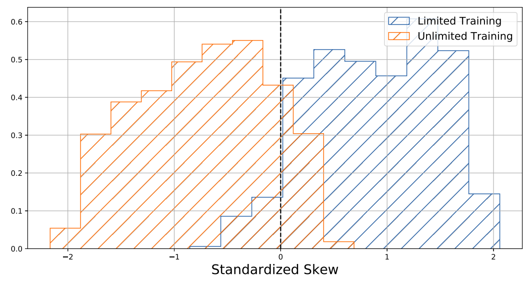
Using the skew as an indicator for throttled throughput series is motivated by the discussion in [RVS07]. For this, each sample $\mathbf{d}[k]$ of a throughput time series $\mathbf{d}$ is interpreted as a sample of an univariate random variable d. Then the skew can be calculated according to Equation 3.6. Note that, in practice, the calculation is based on the samples moments of $\mathbf{d}_n$, given by:

$$m_{j,n} = \frac{1}{K} \sum_{k=1}^{K} \left(\mathbf{d}_n[k] - \bar{d}_n \right)^j \tag{3.7}$$

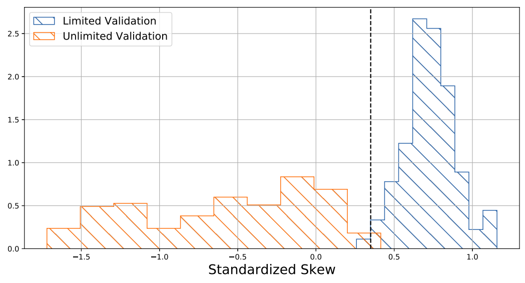
Here, the compensation term $\frac{\sqrt{K(K-1)}}{K-2}$ is ignored due to $K = 70$. Equation 3.6 can then be used as a throttling indicator. The idea here is that unlimited measurements randomly fluctuate around their average rate — while throttled ones are asymmetric due to the artificial hard limit introduced at rates above $\bar{d}$.

Figure 3.6 provides validation for this assumption — it shows histograms of the skew obtained for training and validation data sets.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



(a) Training data set.



(b) Validation data set.

Figure 3.6: Standardized Skew Histograms.

We can see that contrary to $T(\mathbf{d}')$, the skew does not allow for perfect separation of the data. Again we observe, that additional features need to be considered to set the thresholds correctly. It is also interesting to see that the histograms of unlimited and limited measurements in 3.6b differ notably in their shape. Most limited measurements seem to be concentrated in a smaller region of skew — while the unlimited ones are uniformly spread across a broader range.

All in all, the conducted feature engineering succeeds in reducing the dimensionality of the data. It captures the relevant properties of a raw test $\mathbf{r}$ and collects them in the feature vector $\mathbf{x}$. Still, it was shown that no single feature is sufficient to provide perfect classification. Multiple features — combined with effective generalization — are needed to correctly classify the test and the validation data sets. This observation lays out the objective of the subsequent learning task.

3.3 Machine Learning

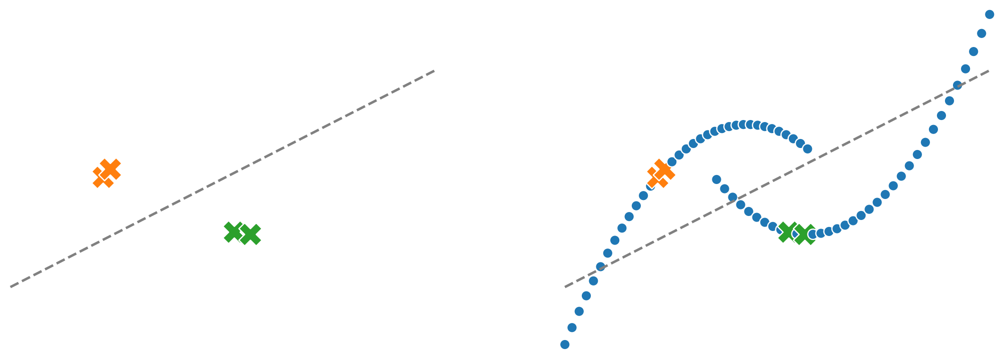
The main objective of the machine learning approach in this use case is, to provide classification in limited/unlimited measurements by operating on the feature vector $\mathbf{x}$. It was shown in the previous Chapter 3.2, that no single feature is sufficient to correctly classify all of the training and validation data. Thus I aim for a mapping of each $\mathbf{x}_n$ to a class label y_n — taken each of the features in $\mathbf{x}_n$ into account.

3.3.1 Semi Supervised Learning

As already discussed in Chapter 2.1, supervised, and unsupervised learning differ with regards to the labels $\mathbf{x}$ provided during training. Semi-supervised learning is a hybrid of the two. Here, labeled examples are available, but the majority of samples are unlabeled. In a nutshell, a semi-supervised algorithm does not only operate on labeled examples but can also learn from unlabeled ones.

This is beneficial in many use-cases. We often encounter situations where labeled examples are hard to come by or tedious to collect. Commonly, unlabeled samples are — at the same time — available in huge numbers. This also applies to this use-case: A practically

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) Supervised (b) Semi-Supervised

Figure 3.7: Unlabeled samples can improve accuracy.

endless number of unlabeled examples is available from the open data platform — while labeled examples have to be collected by hand (see Chapter 3.1).

Figure 3.7 provides some intuition on why unlabeled examples can be beneficial in the first place. It shows a binary classification problem with two labeled examples for each class. The supervised classifier in Figure 3.7a can only operate on the four labeled examples. Let us assume that the data comes from two natural clusters indicated in Figure 3.7b. If that is the case, a linear classifier not aware of the underlying distribution would not be able to obtain a correct decision surface. This is shown in Figure 3.7a. Here, we see that the linear classifier — the decision surface is indicated by the dotted line — misclassifies a significant number of samples. A semi-supervised learning algorithm, on the other hand, does also consider the unlabeled examples and can utilize an underlying density assumption. By doing so, it provides a decision surface, which correctly reflects the natural clusters.

In the following, I discuss two closely linked semi-supervised learning approaches: Label propagation and label spreading.

Label Propagation

Label propagation is a graph-based semi-supervised learning algorithm introduced by [BDLR06]. In a nutshell, each sample — represented by a vertex — does interactively exchange information with its neighbors. Those updates are weighted by some affinity metric which has to be defined for each edge. Let us consider two distinct training data sets, one of them labeled $T_l = \{\mathbf{x}_n, y_n\}_{n=1}^{N_l}$ and the other one unlabeled $T_u = \{\mathbf{x}_n\}_{n=1}^{N_u}$. For each of those samples we define soft labeles and collect them in $\hat{\mathbf{y}} = (\hat{\mathbf{y}}_l, \hat{\mathbf{y}}_u)$. Here, $\hat{\mathbf{y}}_l$ contains all the labeled samples — they are initialized by their original hard labels y_n and are from $\{-1, 1\}$. Meanwhile, each of the N_u unlabeled samples is initialized by 0.

Figure 3.8 shows an example of such a graph. It consists of three vertices $\mathbf{x}_0$, $\mathbf{x}_1$ and $\mathbf{x}_2$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

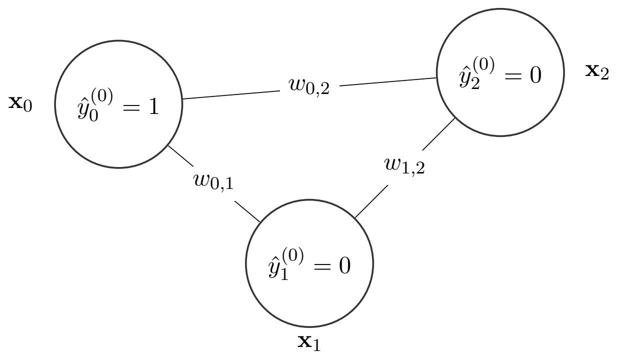
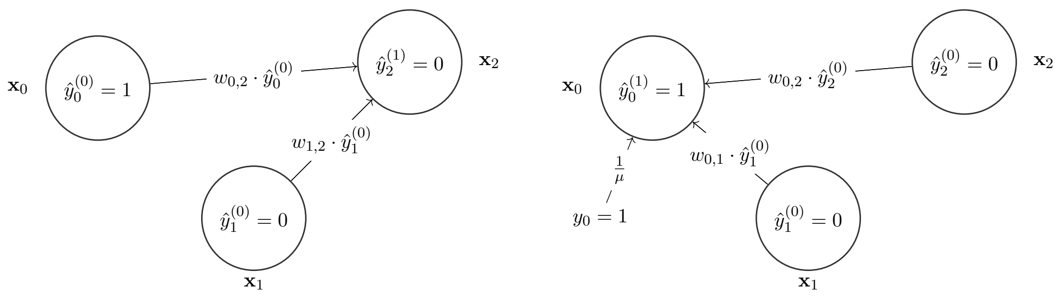


Figure 3.8: Initialization of Label Propagation.



(a) Labeled update. (b) Unlabeled update.

Figure 3.9: Label Propagation Graph.

Here, $\mathbf{x}_0$ is a labeled sample with $y_0 = 1$, while $\mathbf{x}_1$ and $\mathbf{x}_2$ are unlabeled.

In general, the edges in label propagation are calculated by some similarity measure and collected in a matrix $\mathbf{W}$ — a common choice is the RBF kernel:

$$\mathbf{W}_{nm} = e^{-\frac{\|\mathbf{x}_n - \mathbf{x}_m\|^2}{2\sigma^2}} \tag{3.8}$$

with hyperparameter σ. An alternative is kNN, where $\mathbf{W}_{nm} = 1$ if $\mathbf{x}_m$ is one of the k nearest neighbors of $\mathbf{x}_n$, otherwise it is 0. During each iteration of label propagation the soft labels of the nodes from $\hat{\mathbf{y}}_u$ are updated according to:

$$\hat{y}_n^{(t+1)} \leftarrow \frac{\sum_m \mathbf{W}_{nm} \hat{y}_m^{(t)}}{\sum_m \mathbf{W}_{nm} + \epsilon} \tag{3.9}$$

Here, ϵ is a hyperparameter added for numerical stability and should be assigned a small positive value. Equation 3.9 describes how nodes pass their updates along to their neighbors, multiplied by the edge weight. The same mechanism is shown in Figure 3.9a, where the normalization is omitted for readability.

For labeled examples, there is an additional term consisting of the original hard label of the sample. An example of this can be seen in Figure 3.9b. Here again, the normalization

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

term is omitted. The complete update step — including the normalization — is given by:

$$\hat{y}_n^{(t+1)} \leftarrow \frac{\sum_m \mathbf{W}_{nm}\hat{y}_m^{(t)} + \frac{1}{\mu}y_n}{\sum_m \mathbf{W}_{nm} + \frac{1}{\mu} + \epsilon} \tag{3.10}$$

In Equation 3.10 the hard label is multiplied by $\frac{1}{\mu}$ — here $\mu \in (0, +\infty)$. It is calculated from the hyperparameter $\alpha \in (0, 1)$ by $\mu \leftarrow \frac{\alpha}{1-\alpha}$. α controls, to what extend samples from $\hat{\mathbf{y}}_l$ are allowed to change during label propagation. An α close to 0 means that the hard label y_n is kept during each iteration of Equation 3.10. For an α close to 1, the hard labels are ignored. After convergence, each vertex is labeled by the sign of its soft label $\hat{\mathbf{y}}^{(\infty)}$. The complete algorithm of label propagation can be found in Algorithm 3.1.

Algorithm 3.1: Label Propagation.

```
1: Compute the affinity matrix $\mathbf{W}$ and set $\mathbf{W}_{nn} = 0$ ;
2: $\mathbf{D}_{nn} \leftarrow \sum_m W_{nm}$ ;
3: Select a parameter $\alpha \in (0, 1)$ and a small $\epsilon > 0$
4: $\mu \leftarrow \frac{\alpha}{1-\alpha} \in (0, +\infty)$
5: $\mathbf{A}_{nn} \leftarrow 1 + \mu\mathbf{D}_{nn} + \mu\epsilon$
6: Set $\hat{\mathbf{y}}^{(0)} \leftarrow (y_1, \dots, y_{N_l}, 0, 0, \dots, 0)$
7: **while** not converged to $\hat{\mathbf{y}}^{(\infty)}$ **do**
8:     $\hat{\mathbf{y}}^{(t+1)} \leftarrow \mathbf{A}^{-1}\left(\mu\mathbf{W}\hat{\mathbf{y}}^{(t)} + \hat{\mathbf{y}}^{(0)}\right)$
9: **end while**
10: Label vertices $x_n$ by sign of $\hat{\mathbf{y}}_n^{(\infty)}$
```

Label Spreading

Label spreading is an alternative graph-based semi-supervised algorithm. Here, again, samples are represented as vertices, and the edges indicate some affinity measure. Inherently, it is based on the same idea as label propagation — but it is formulated in a slightely different way. Namely, it operates directly on the normalized graph laplacian $\mathcal{L} \leftarrow \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$. Still, intuitively, label propagation and label spreading incorporate the same density assumption about the underlying data and pass updates iteratively to neighboring nodes.

The complete algorithm of label spreading is provided in Algorithm 3.2. Another relevant aspect of label spreading is that the loss function has regularization properties, which renders it more robust to noise on the input data [BDLR06]. This is also why I use it for the limited/unlimited classification task here. Generally, we can assume that there is noise on the input samples — induced by different fluctuations affecting the throughput time series. A comprehensive overview of label spreading is given in [ZBL+04].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Algorithm 3.2: Label Spreading.

```
1: Compute the affinity matrix $\mathbf{W}$ and set $\mathbf{W}_{nn} = 0$ ;
2: $\mathbf{D}_{nn} \leftarrow \sum_m W_{nm}$ ;
3: Select a parameter $\alpha \in (0, 1)$
4: $\mathcal{L} \leftarrow \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$
5: Set $\hat{\mathbf{y}}^{(0)} \leftarrow (y_1, \dots, y_{N_l}, 0, 0, \dots, 0)$
6: while not converged to $\hat{\mathbf{y}}^{(\infty)}$ do
7:     $\hat{\mathbf{y}}^{(t+1)} \leftarrow \alpha\mathcal{L}\hat{\mathbf{y}}^{(t)} + (1 - \alpha)\hat{\mathbf{y}}^{(0)}$
8: end while
9: Label vertices $x_n$ by sign of $\hat{\mathbf{y}}_n^{(\infty)}$
```

3.3.2 Model Training & Validation

For the final classification, I use label spreading with an RBF kernel, operating on a standardized feature vector $\mathbf{x}$ [2]. Standardized meaning, that each of the features x_i — $i \in [1, 5]$ — from $\mathbf{x}$ is transformed to have unit variance and zero mean. This is done for each sample $\mathbf{x}_n$ via the following processing step:

$$x_{i,n} = \frac{x_{i,n} - \mu_i}{\sigma_i} \tag{3.11}$$

Here μ_i and σ_i are mean and standard deviation of the distinct scalar feature, estimated over the training data set T_{lab}. The results of the label spreading validation can be found in Table 3.2.

Test Data Set	Approach	Accuracy	F1 Score
$V_{\mathbf{I10}}$ & $V_{\mathbf{CL10}}$	Supervised	98%	97%
	Semi-Supervised	**100%**	**100%**
$V_{\mathbf{I20}}$ & $V_{\mathbf{CL20}}$	Supervised	96%	96%
	Semi-Supervised	**99%**	**99%**

Table 3.2: Validation Results Label Spreading.

The results were obtained by the following procedure: For the semi-supervised case, I use the feature vectors of all of T_{lab} — including labels. Then I randomly split the respective validation data sets (V_{I10} & V_{CL10}) or (V_{I20} & V_{CL20}) into an unlabeled and a small labeled share. The unlabeled share is then provided to the algorithm, while the labeled one is used for benchmarking. I repeat this random selection 100 times — effectively cross-validation. The final accuracy and F1 scores are then the averages over all trials.

For comparison, the results from a supervised approach based on label spreading are also shown in Table 3.2. Here, the algorithm received all of T_{lab} with labels. Then I used all samples from the respective validation data sets to calculate the performance measures.

[2]The code used for classification can be found in Appendix A.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Table 3.2 shows the accurary and the F1 score for each of the experiments. Here, accuracy is defined as the percentage of correctly classified samples. In the case of an uneven distribution of labels in the test data set, this might be misleading. This is why the F1 score is often used as a measure that takes the underlying distribution of labels into account. In this use-case, however, F1 score and accuracy are effectively the same — as the test data set is evenly split into limited and unlimited samples.

All in all the results in Table 3.2 show, that the classification delivers promising results. The semi-supervised approach classifies all samples correctly. This is most likely due to the test statistic $T(\mathbf{d}')$ — which separates the data almost perfectly. We can also see, that the approach generalizes well to the 20 Mbit/s data sets — even though only labeled 10 Mbit/s were part of the training set. Furthermore, the semi-supervised approach outperforms the supervised one in each setting. It further increases accuracy from the already high baseline performance due to the successful feature extraction.

3.4 Classification Results & Operator Benchmarking

Finally, I apply the trained algorithm from Chapter 3.3.2 to an unlabeled data set from RTR Open Data. In total, this data set consists of 171.770 measurements — conducted during 2018. To ensure that all of the measurements are static — the users were not moving during measurements — all samples which have high fluctuations of RSRP were discarded here. The data set does, furthermore, only consist of LTE measurements.

	User view	**Limited**	**Tariff limited**	**Network view**
MNO-A	57 Mbit/s	**48%**	39 Mbit/s	**75 Mbit/s**
MNO-B	54 Mbit/s	**41%**	38 Mbit/s	**68 Mbit/s**
MNO-C	51 Mbit/s	**50%**	40 Mbit/s	**63 Mbit/s**

Table 3.3: Classification of RTR-OpenData.

The classification results are provided in Table 3.3. Each of the rows shows the results for one of the three major Austrian mobile network operators. Here, the user view is given by the reported raw results of each measurement. Out of those, the label spreading algorithm classifies between 40 and 50% as limited. It is interesting to see that the tariff shaping rate differs between operators. The average tariff limits — obtained by calculating the average end rate for all limited tests — are all closely around 40 Mbit/s. Finally, I obtain the network view by removal of the limited measurements. We can see that — while the operator benchmarking does not change for this specific use case — the operators are further apart in the network view. Unsurprisingly the network view reports higher rates for all three MNOs.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.5 Conclusion

Overall the results in this chapter show that the detection of tariff limits in crowdsourced network benchmarks is possible. This addresses one of the questions introduced in Chapter 1: ***...the inference of the network view from the user view while dealing with the limited availability of parameters***. In fact, the accuracy measures for the validation data indicate that the classification can be conducted almost without any error.

To achieve this, I require an extensive set of labeled measurements combined with a learning algorithm that offers effective generalization. Semi-Supervised approaches can provide this notion of generalization by also making use of unlabeled samples. Thus, they are well suited for this task. The algorithm in this chapter operates on publically RTR-Netztest data, and results can thus be compared to alternative implementations by any interested reader. Moreover, the approach was validated using a representative set of labeled outdoor measurements. This step is regularly missing in approaches presented in literature, due to the tedious process of data-collection. Most authors do also not have access to a reference eNodeB, which is really the foundation for the robust performance of the classifier in this work.

In the realm of context detection, the results from this chapter allow for the removal of tariff limited tests. Doing so transforms the user into the network view — which is an essential requirement for fair benchmarking. Still, I think that it might be interesting to examine whether additional information can be extracted from the tariff limited time-series. Instead of removing them from the analysis, one could, for instance, try to infer the "unlimited-rate" from a limited measurement. This might be a promising entry point for further research.

In the overall course of this work, this chapter also acts as a link to representation learning. Consider the feature engineering conducted in Section 3.2: The histogram for $T(\mathbf{d}')$ and for the *skew* show, how the feature vector $\mathbf{x}$ drastically reduces the complexity of separating limited/unlimited tests. In $\mathbf{x}$, they do already form their distinct clusters — the objective of the subsequent machine learning step is mainly to generalize those clusters to previously unseen data without labels. This is directly related to representation learning — which I apply to an RTR-Netztest data set in Chapter 6. To provide the necessary background, Chapter 4 will focus on the basics of deep learning, while 5 will introduce the idea of an autoencoder.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 4

Deep Learning Introduction

The term deep learning refers to a subdomain of machine learning, which is based on Artificial Neural Networks. In a Neural Network (NN), neurons are organized in layers, which are concatenated to form a deep network. Each neuron processes the inputs of the previous layer and applies a distinct activation function. During training the objective is to reduce a loss between the output of the network y and the labels in the training set $\mathbb{T}_{data} = \{\mathbf{x}_n, \mathbf{y}^*_n\}_{n=1}^N$. Training ends when a viable network parameter $\boldsymbol{\theta}$ configuration was found. NNs, are discussed in detail in the following section. Here, the focus is mostly on supervised-learning. In the second part, Chapter 4.2, I will introduce the notion of representation learning. Representation learning is also the basis for unsupervised NN architectures — like autoencoders which are discussed in 5.

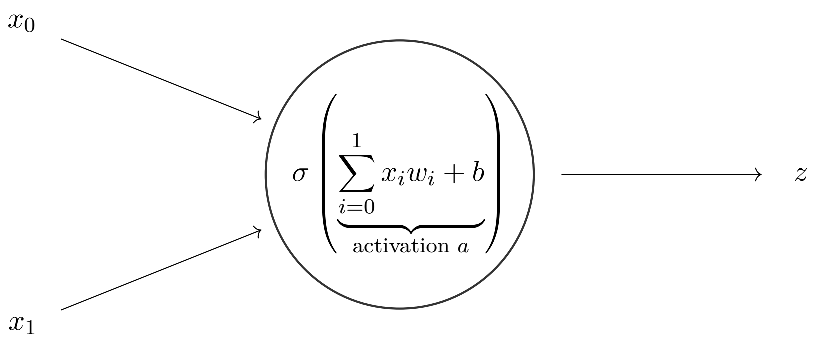

x_0
x_1
$\sigma\left(\underbrace{\sum_{i=0}^{1} x_i w_i + b}_{\text{activation } a}\right)$
z

Figure 4.1: Single perceptron with 2 inputs.

4.1 Neural Networks

NNs — especially deep NNs — are the main driving force behind the machine learning revival in the last decades. In general, they can be seen as universal function approximators, approximating some function f^*. Given pairs of inputs $\mathbf{x}$ and targets $y^* = f^*(\mathbf{x})$, a NN aims to learn a mapping $y = f(\mathbf{x}; \boldsymbol{\theta})$, which reduces some loss between y and y^*. Thereby, the network parameters $\boldsymbol{\theta}$ consist of weights and biases [BGC17, Chapter 6]. At the core of feedforward networks lies the idea of the perceptron.

Figure 4.1 shows such a perceptron with inputs x_0 and x_1. Those perceptrons form the basic building blocks of NNs. The perceptron in 4.1 perfoms a simple operation on its input vector $\mathbf{x} = [x_0, x_1]^\top$ in order to produce a scalar output z. First, the activation a is calculated as the sum of the scalar bias term b and the inner product of $\mathbf{x}$ with the neuron's weights $\mathbf{w} = [w_0, w_1]^\top$. After this linear operation, a function $\sigma(\cdot)$ is applied to the activation a to produce the final output z. We refer to $\sigma(\cdot)$ as the activation function — it is usually nonlinear. In a nutshell, a single neuron applies an affine transformation to it's inputs before the result is passed through a nonlinearity [Bis06, Chapter 5].

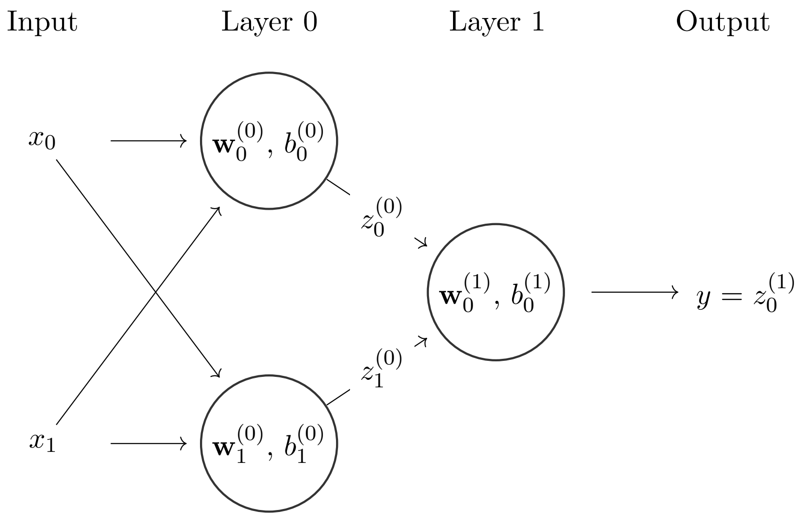

Input
Layer 0
Layer 1
Output
x_0
x_1
$\mathbf{w}_0^{(0)}, b_0^{(0)}$
$\mathbf{w}_1^{(0)}, b_1^{(0)}$
$z_0^{(0)}$
$z_1^{(0)}$
$\mathbf{w}_0^{(1)}, b_0^{(1)}$
$y = z_0^{(1)}$

Figure 4.2: Simple Two Layer MLP.

Concatenated layers of perceptrons form densely connected NNs, which are also called Multi-Layer Perceptrons. Here concatenation means, that the output of every single neuron in a given layer acts as one of the inputs to the neurons of the succeeding layer [BGC17, Chapter 6]. Thus, the mapping learned by the network is a composition of the individual functions $\mathbf{f}()^{(k)}$ of each layer $k = 0 \dots K-1$. The complete input-output relation of a network $\mathbf{f}()$ with $K = 2$ is then given by $\mathbf{f} = \mathbf{f}^{(1)}(\mathbf{f}^{(0)}(\mathbf{x}))$.

Figure 4.2 shows an example of such a network with 3 neurons in total. Equations 4.1 and 4.2 completely describes the input-output relation of this network — showcasing the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

underlying mechanisms of connecting multiple neurons into a layered network.

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}, \mathbf{w}_0^{(0)} = \begin{bmatrix} w_{0,0}^{(0)} \\ w_{1,0}^{(0)} \end{bmatrix}, \mathbf{w}_1^{(0)} = \begin{bmatrix} w_{0,1}^{(0)} \\ w_{1,1}^{(0)} \end{bmatrix}, \mathbf{w}_0^{(1)} = \begin{bmatrix} w_{0,0}^{(1)} \\ w_{1,0}^{(1)} \end{bmatrix} \tag{4.1}$$

$$\mathbf{z}^{(0)} = \begin{bmatrix} z_0^{(0)} = \sigma^{(0)}\left(\mathbf{x}^\top \mathbf{w}_0^{(0)} + b_0^{(0)}\right) \\ z_1^{(0)} = \sigma^{(0)}\left(\mathbf{x}^\top \mathbf{w}_1^{(0)} + b_1^{(0)}\right) \end{bmatrix} y \quad = z_0^{(1)} = \sigma^{(1)}\left(\mathbf{z}^{(0)\top} \mathbf{w}_0^{(1)} + b_0^{(1)}\right) \tag{4.2}$$

In general each neuron in some layer k receives the outputs $\mathbf{z}^{(k-1)}$ from the previous layer as inputs — except for $k = 0$ which receives $\mathbf{x}$ directly. Its outputs $\mathbf{z}^{(k)}$ are then again fed into the next layer. In this setup, neurons of the same layer do not interact with each other — the flow of information is strictly from lower to higher layers. We refer to such structures as feedforward networks [BGC17, Chapter 6].

Architectures, where this limitation does not apply, are called recurrent networks [BGC17, Chapter 10] — those are heavily used in Natural Language Processing — in this work, I will, however, focus on feedforward networks only. Thus, the term NN is used synonymously with feedforward networks. More precisely the scheme in 4.2 is denoted a dense feedforward network. Dense means that every neuron is connected with each of the neurons in the succeeding layer.

4.1.1 Training Neural Networks & Losses

Consider a typical supervised learning task, where the provided labeled data consists of a set $\mathbb{T}_{data} = \{\mathbf{x}_n, \mathbf{y}_n^*\}_{n=1}^N$. Therefore, $\mathbb{T}_{data}$ is a collection of N distinct pairs of inputs $\mathbf{x}$ and targets $\mathbf{y}^*$. Note that we assume $\mathbf{y}_n^* = \mathbf{f}(\mathbf{x}_n)$. Thus, the output of $\mathbf{f}(\cdot)$ is itself a vector — we denote the dimension of $\mathbf{y}$ as I_{out}. Again, the objective of the learning task is to find a mapping $\mathbf{y}_n = f(\mathbf{x}_n; \boldsymbol{\theta})$. More precisely, to learn a parameter vector $\boldsymbol{\theta}$, which reduces some error criteria $L(\mathbf{y}_n^*, \mathbf{y}_n)$. $L(\cdot)$ is referred to as the loss function — its choice typically depends on the given task [BGC17, Chapter 5].

A common choice for $L(\cdot)$ in case of a regression task is the Mean Squared Error (MSE). It is given by:

$$L_{\text{MSE}}(\mathbf{y}_n^*, \mathbf{y}_n) = \frac{1}{I_{out}} \|\mathbf{f}(\mathbf{x}_n) - \mathbf{y}_n^*\|^2 \tag{4.3}$$

For multiclass classification tasks where the output $\mathbf{y}$ models a probability mass function with I_{out} entries, categorical crossentropy is often selected [BGC17, Chapter 3].

$$L_{CrossEntropy}(\mathbf{y}_n^*, \mathbf{y}_n) = -\sum_{i=1}^{I_{\text{out}}} \mathbf{y}_n^*[i] \ln(\mathbf{f}(\mathbf{x}_n)[i]) \tag{4.4}$$

While $L(\cdot)$ is evaluated for each pair $\mathbf{x}_n$ and $\mathbf{y}_n^*$ individually, the objective is, in general, to minimize the accumulated loss over the complete training set $\mathbb{T}_{data}$. We denote this

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

loss by $J(\boldsymbol{\theta})$ — it is calculated according to Equation 4.5.

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N} L\left(\mathbf{y}_n^*, \mathbf{f}(\mathbf{x}_n)\right) \tag{4.5}$$

Gradient Descent & Backpropagation

NN optimization is typically based on gradient descent [BGC17, Chaper 5]. This means that, during each iteration the gradient $\mathbf{g} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is calculated — it provides the direction of steepest descent of $J(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. The networks parameters $\boldsymbol{\theta}$ are then updated according to Equation 4.6.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \epsilon \mathbf{g} \tag{4.6}$$

In Equation 4.6 the stepsize ϵ is a hyperparameter. The inherent structure of a feedforward neural network allows for a computationally efficient evaluation of the gradient, where the updates propagate from $J(\boldsymbol{\theta})$ back through the network. This way, each neuron receives the derivatives of the loss with respect to its own weights and the corresponding bias. [Bis06, Chapter 5] gives a detailed overview of backpropagation — therefore, in this work, I will only provide an illustrative example.

Consider again the network given in Figure 4.2. During a forward step of the optimization the output y and the loss $J(\boldsymbol{\theta})$ can be calculated by Equation 4.1 and 4.2. Figure 4.3 shows how the gradient information traverses the network in the opposite direction during backpropagation.

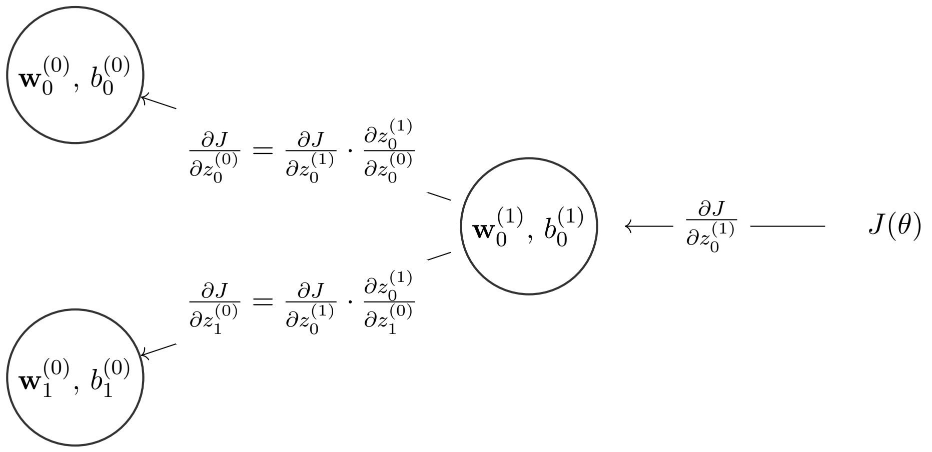

$\mathbf{w}_0^{(0)}, b_0^{(0)}$
$\frac{\partial J}{\partial z_0^{(0)}} = \frac{\partial J}{\partial z_0^{(1)}} \cdot \frac{\partial z_0^{(1)}}{\partial z_0^{(0)}}$
$\mathbf{w}_0^{(1)}, b_0^{(1)}$
$\frac{\partial J}{\partial z_0^{(1)}}$
$J(\theta)$
$\frac{\partial J}{\partial z_1^{(0)}} = \frac{\partial J}{\partial z_0^{(1)}} \cdot \frac{\partial z_0^{(1)}}{\partial z_1^{(0)}}$
$\mathbf{w}_1^{(0)}, b_1^{(0)}$

Figure 4.3: Backpropagation.

Without loss of generality, assume linear activation functions $\sigma(a) = a$ throughout the network. To simplify the notation, the loss $J(\boldsymbol{\theta})$ is here the squared error of a single training pair consisting of $\mathbf{x}$ and y. Thus, $J(\boldsymbol{\theta}) = (y - z_0^{(1)})^2$. Under those assumptions, the calculation of the derivatives for the weights of neuron $j = 0$ in layer $k = 0$ is as follows:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$
\begin{aligned}
\frac{\partial J(\boldsymbol{\theta})}{\partial w_{0,1}^{(0)}} &= \frac{\partial J}{\partial z_0^{(1)}} \cdot \frac{\partial z_0^{(1)}}{\partial a_0^{(1)}} \cdot \frac{\partial a_0^{(1)}}{\partial z_1^{(0)}} \cdot \frac{\partial z_1^{(0)}}{\partial a_1^{(0)}} \cdot \frac{\partial a_1^{(0)}}{\partial w_{0,1}^{(0)}} \\
&= \frac{\partial J}{\partial z_0^{(1)}} \cdot 1 \cdot \frac{\partial a_0^{(1)}}{\partial z_1^{(0)}} \cdot 1 \cdot \frac{\partial a_1^{(0)}}{\partial w_{0,1}^{(0)}} \\
&= 2(y - z_0^{(1)}) \cdot w_{0,1}^{(1)} \cdot x_0
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial J(\boldsymbol{\theta})}{\partial w_{0,0}^{(0)}} &= \frac{\partial J}{\partial z_0^{(1)}} \cdot 1 \cdot \frac{\partial a_0^{(1)}}{\partial z_1^{(0)}} \cdot 1 \cdot \frac{\partial a_1^{(0)}}{\partial w_{0,0}^{(0)}} \\
&= 2(y - z_0^{(1)}) \cdot w_{0,1}^{(1)} \cdot x_1
\end{aligned}
$$

This illustrates how backpropagation evaluates the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ in a recursive fashion — each neuron calculates the derivative with respect to its own weights and passes this information on to the neurons in the preceding layer. Note that this allows for an efficient implementation — only the remaining factors of the chain rule have to be evaluated. The derivatives of $w_{0,0}^{(0)}$ and $w_{0,1}^{(0)}$ do for instance only differ in the last factor.

This example covers the underlying idea of backpropagation — it does, however, not include a neuron with multiple outputs. In this case, the gradient is simply the sum of the updates received from all outputs.

As already mentioned, the objective is to minimize the loss over the whole training set $J(\boldsymbol{\theta})$ — not the individual losses per sample. The gradient $\mathbf{g} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is, therefore, also the sample mean over all gradients of the individual losses $L(\cdot)$ in $\mathbb{T}_{data}$.

$$
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} \left[L\left(\mathbf{y}_n^*, \mathbf{f}(\mathbf{x}_n)\right)\right] \tag{4.7}
$$

Implementing Equation 4.7 is unfeasible in practice — we are usually dealing with training sets with a cardinality of over 10.000 samples. Minimizing $J(\boldsymbol{\theta})$ would, therefore, require to evaluate the gradient for each input-output pair separately. This is why an approximation of gradient descent — Stochastic Gradient Descent (SGD) — is applied in practice. During training, the data set $\mathbb{T}_{data}$ is thereby split into smaller subsets with cardinality M each. Those subsets are referred to as batches. For each batch, the gradient $\mathbf{g}$ is then calculated according to Equation 4.8.

$$
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{M} \nabla_{\boldsymbol{\theta}} \left[\sum_{m=1}^{M} L\left(\mathbf{y}_m^*, \mathbf{f}(\mathbf{x}_m)\right)\right] \tag{4.8}
$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Note, the difference between equations 4.7 and 4.8. In SGD, the gradient itself is calculated over the averaged loss — as opposed to Equation 4.7 where the individual gradients are summed up. This means that the weights are updated only once per batch and not for each sample pair.

Typically, the training phase consists of multiple epochs. During each epoch the network is exposed to the complete training set $\mathbb{T}_{data}$, split into multiple batches.

4.1.2 Non Linearity

In 1989, [HSW+89] provided a proof that multilayer NNs are universal function approximators. The details of this argument are beyond the scope of this work — it is sufficient to state that *a single hidden layer with some squashing activation function can approximate any continuous function on a closed and bounded subset of R^n, with an arbitrary small non-zero error*. Squashing function does thereby refer to a class of functions for which $\lim_{a \to +\infty} \sigma(a) = 1$ and $\lim_{a \to -\infty} \sigma(a) = 0$ — the simplest example of this being the step function. Note, that [HSW+89] does not make any statements on how to optimize the parameters $\boldsymbol{\theta}$ to find an $f(\mathbf{x}; \boldsymbol{\theta})$ which approximates $f(\cdot)^*$. This means that we know that NNs can — given a sufficient number of neurons — represent any function, whether it is possible to learn this approximation is a different question.

In 1993, [LLPS93] extented this argument to all non-polynomial activation functions. This does also include step-wise linear functions like RELU [1], which are heavily used today.

The two proofs in [HSW+89] and [LLPS93] indicate that for its role as universal function approximators, the activation function $\sigma(\cdot)$ is crucial. Without it, the whole network $f(\cdot)$ would collapse into a linear model — it would not be able to approximate arbitrary nonlinear functions. Different classes of functions can be selected as $\sigma(\cdot)$ — the only hard constraint is that they must be differentiable, to allow for optimization via SGD. Still, the choice of $\sigma(\cdot)$ is not arbitrary but often customized to the specific learning objective. This is especially true for the last layers activation function $\sigma(\cdot)^{(K-1)}$. It transforms the activations $a^{(K-1)}$ into the output y, often enforcing specific constraints on the values of y. For binary classification problems, for instance, it is common to map a to the interval $[0, 1]$.

In the following different commonly used classes of activation functions are discussed. Those include squashing functions such as sigmoid and tanh, stepwise linear functions such as RELU and the softmax function, which is used for multiclass classification tasks. Each of those $\sigma(\cdot)$ have distinct use cases. In a final network configuration, activation functions are usually selected on a per-layer basis. In general, the choice of an activation function is often specific to the task, and there is still ongoing discussion regarding the properties of distinct activations [XWCL15].

[1] Stepwise linear functions such as RELU will be introduced in 4.1.2

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Squashing Functions

Selecting squashing functions as $\sigma(\cdot)$ is motivated by the idea that single neurons should have a hard threshold, above which they activate. The simplest possible squashing function — the step function — is not suitable for neural networks, as its derivative is zero for all $a \neq 0$. Thus, the flow of information during backpropagation is interrupted. *sigmoid* (Eq. 4.9) and *tanh* (Eq. 4.10) can be interpreted as continuous and differentiable versions of the step function. They both provide a region tightly around $a = 0$, where the derivatives are relatively large. This means that — in those regions — small changes of a result in significant deviations of z.

$$\sigma_{\text{sig}}(a) = \frac{1}{1 + e^{-a}} \tag{4.9}$$

$$\sigma_{\tanh}(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}} \tag{4.10}$$

Eq. 4.9 and 4.10 are closely linked — in fact, $\sigma_{\tanh}(a) = 2\sigma_{\text{sig}}(2a)) - 1$.

While squashing functions are commonly used, they suffer from some inherent issues — especially when used for hidden layers [LBOM12]. The fact that they saturate for high and low values of a means that the gradient is effectively zero in such regions. This can lead to a situation where the flow of gradient information in deep architectures is not sufficient, the so-called vanishing gradient problem [Hoc98].

The sigmoid function is often used as the last layer $\sigma(\cdot)^{(K-1)}$ — this way y can be interpreted as the probability of a Bernoulli distribution.

Stepwise Linear Functions

[LLPS93] showed that the universal function approximation property of NNs holds for any non-polynomial function. This motivates the use of stepwise linear functions such as the Rectified Linear Unit (RELU) for hidden layers [GBB11]. In recent years it has effectively become the default activation function in many applications. RELU is given by Equation 4.11.

$$\sigma(a) = \begin{cases} a, & \text{if } a \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.11}$$

Note, that Eq. 4.11 is non-differentiable at $a = 0$. In theory RELU would therefore not be a suitable candiate for an activation function. In practice, however, this singularity is ignored during implementation, and the derivative for $a = 0$ is set to 0. This simplification does not pose a problem to NN optimization. Also, because the probability of $a = 0$ is practically 0.

The fact that the gradient for RELU is either 0 or 1 does also mean that only active neurons do contribute to the learning process. This sparsity can be beneficial in many applications — still, it can introduce problems for some learning tasks. Here, Leaky-Relu provides an extension to RELU where and additional parameter α specifies the derivative of $\sigma(\cdot)$ for $a \leq 0$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Softmax

Softmax is an activation function which is used in multiclass classification tasks — it ensures that the output $\mathbf{z}^{(K-1)}$ of the final layer $\sigma(\mathbf{a})^{(K-1)}$ of a network does correctly model a probability distribution [BGC17]. The equation for Softmax is given by 4.12.

$$\sigma(\mathbf{a})_j = \frac{e^{a_j}}{\sum_{j=0}^{J-1} e^{a_j}} \tag{4.12}$$

Here, J specifies the total number of neurons in the given layer. Softmax applies a mapping from $\mathbb{R}^J \to \mathbb{R}^J$, where the individual outputs are in the interval $[0, 1]$, and $\sum_{j=0}^{J-1} \sigma(\mathbf{a})_j = 1$ is ensured. This way, the outputs $\sigma(\mathbf{a})_j$ can be interpreted as the probability of the input sample $\mathbf{x}$, belonging to one of the J classes. Softmax can also be seen as an extension of a sigmoid output to higher dimensions.

Networks with softmax outputs often utilize Cross-Entropy (see Eq. 4.4) with respect to some target distribution as a loss function.

Approximating logical AND with a simple neural network

While this chapter did already introduce the basic building blocks of NNs, consider the following example to give some intuition on how multi-layer perceptrons can model arbitrary functions — and especially the role that nonlinearities play in this context. Consider the task of learning the mapping from $\mathbf{x} = [x_0, x_1]$ to y, where x_0, x_1, y are from $\{0, 1\}$ and $y = x_0 \;\&\; x_1$. Here, & refers to the logical AND operation.

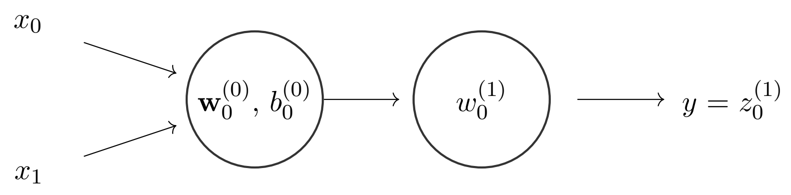


Figure 4.4: Logical AND function approximation.

Figure 4.4 shows a simple NN with 2 neurons. In total there are four trainable parameters $w_{0,0}^{(0)}, w_{1,0}^{(0)}, b_0^{(0)}$ and $w_{0,0}^{(1)}$ — note, that there is no bias $b_0^{(1)}$. Now let us consider two distinct cases, one where the activation function of the second layer $\sigma(\cdot)^{(1)}$ is linear and one where it is chosen to be RELU. Meanwhile, $\sigma(\cdot)^{(0)}$ is linear ($\sigma(a) = a$) in both cases. Thus, the activation of layer 1 is given by $a_0^{(1)} = w_{0,0}^{(1)} \left(x_0 w_{0,0}^{(0)} + x_1 w_{1,0}^{(0)} + b_0^{(0)} \right)$ for case one as well as case two.

We then choose a MSE loss for the target and optimize the networks parameters for each case[2]. For the RELU case $w_{0,0}^{(0)}, w_{1,0}^{(0)}$ are both 1, $b_0^{(0)} = -1$, while $w_{0,0}^{(1)} = 1$. The corresponding activations and outputs are shown in Table 4.1 — here the MSE is effectively

[2]The code for this example can be found in the Appendix A.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

0. This is not true for the linear case. Here, $w^{(0)}_{0,0}$ and $w^{(0)}_{1,0}$ are 0.76, $b^{(0)}_0 = -0.37$ and $w^{(1)}_{0,0} = 0.67$. It is not surprising that no parameter configuration achieves a MSE of 0 for the linear case here.

$\mathbf{x}^\top$	$a^{(0)}_0$	$z^{(0)}_0$	$a^{(1)}_0$	y
$[0,0]$	-1	-1	-1	0
$[0,1]$	0	0	0	0
$[1,0]$	0	0	0	0
$[1,1]$	1	1	1	1

Table 4.1: AND: RELU $\sigma(\cdot)^{(1)}$

$\mathbf{x}^\top$	$a^{(0)}_0$	$z^{(0)}_0$	$a^{(1)}_0$	y
$[0,0]$	-0.37	-0.37	-0.25	-0.25
$[0,1]$	0.37	0.37	0.25	0.25
$[1,0]$	0.37	0.37	0.25	0.25
$[1,1]$	1.13	1.13	1	0.75

Table 4.2: AND: linear $\sigma(\cdot)^{(1)}$

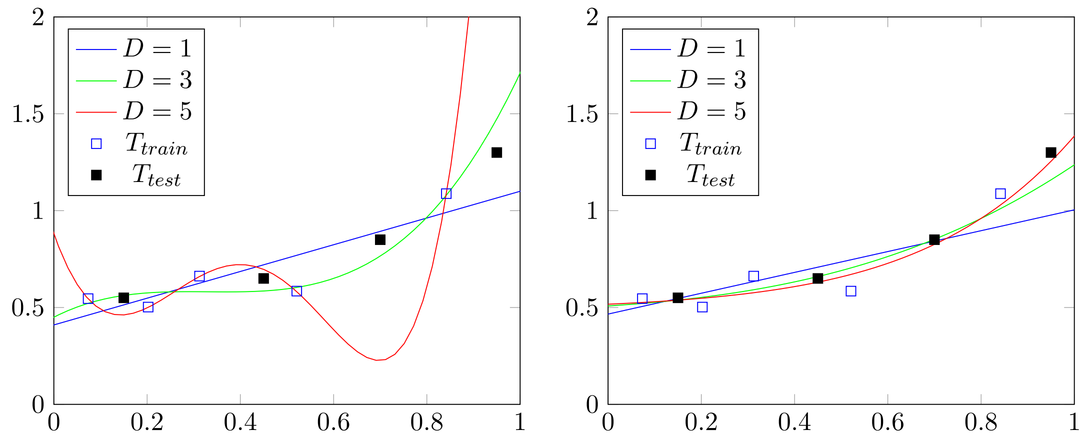
Note that this is, obviously, a toy example. It does, however, showcase how we can find a structure of concatenated perceptrons that models a given function. In practice, the real power lies in the universal applicability of this approach. Given an extensive training set, we can optimize a generic network structure — with multiple hidden layers and a sufficient number of neurons — and backpropagation automates the search over possible parametrizations $f(\mathbf{x};\boldsymbol{\theta})$.

Influence of depth

As already stated, neural networks with a sufficient number of neurons can, in theory, approximate any continuous function on a closed and bounded subset of R^n. In this proof, a network with only one hidden layer was considered. The problem here is that this layer can grow exponentially in size, which makes training unfeasible [Bar93]. It has been shown empirically that deeper models can reduce the overall number of neurons required to represent a given function. Thus, decreasing the number of parameters to optimize during training. Additionally, depth has also been shown to increase generalization & prevent overfitting to some extend [BGC17, Chapter 6]. The increasing depth of network architectures does, however, lead to problems of its own. Internal covariance shift [Iof17] or exploding gradients [PMB12] for instance. For reasonable network depths, approaches like batch-normalization and clipping can combat those issues.

Discussing the underlying reasons why depth is beneficial in most learning tasks is beyond the scope of this work — still, the following references provide some intuition. [BGC17, Chapter 6] stated that using deep architectures expressed a prior belief that the function $f(\cdot)^*$ to be learned is itself a composition of distinct elementary functions — a prior, which seems to be beneficial in many tasks. Moreover, in [SZT17], the influence of depth is discussed from an information theoretical viewpoint. Their findings indicate that deep neural networks optimize the information bottleneck tradeoff between reconstruction and compression for each layer separately.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



(a) Ordinary Least Squares.

(b) Ridge Regression.

Figure 4.5: Overfitting for OLS and Ridge Regression.

4.1.3 Generalization & Regularization

Even though NNs themselves are typically purely deterministic — exceptions are networks that learn moments of probability distributions, like Deep Gaussian Mixture Models or Variational Autoencoders [BGC17] — it is helpful to introduce the idea of a data generating distribution p_{data}. This way, the training $\mathbb{T}_{train}$ and test data set $\mathbb{T}_{test}$ can be interpreted as collections of samples from an unknown distribution p_{data}.

This approach is especially useful when dealing with over and underfitting. While we train our model with data from $\mathbb{T}_{train}$, we want it to perform well on all of p_{data} — also on samples it has not seen before. This concept is referred to as generalization. To ensure that a given model generalizes, the learning objective has to be extended from simply reducing $J(\boldsymbol{\theta})$ over $\mathbb{T}_{train}$. The concept of generalization has already been discussed in the context of semi-supervised approaches in 3.

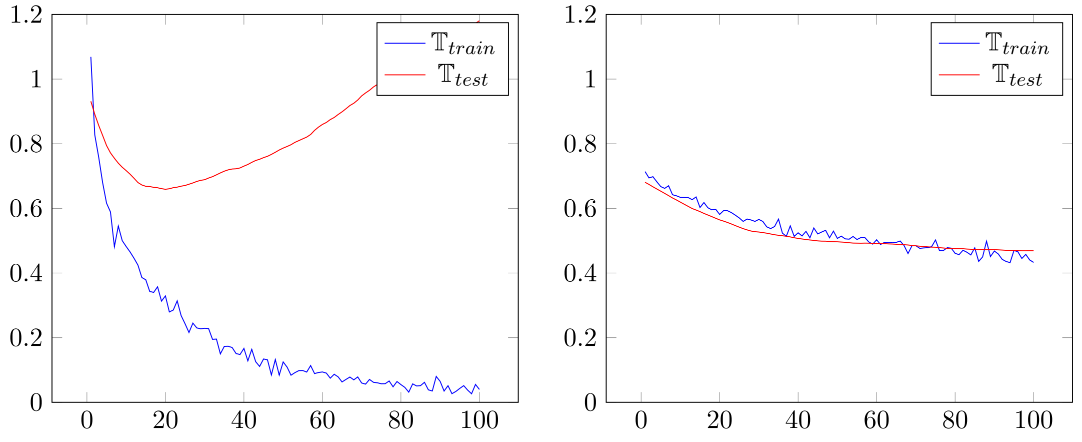
Generalization seems to be inherently linked to a model's capacity — capacity meaning, the complexity of the mapping the model can deploy. Without any adaption to the optimization process, a model with high capacity tends to overfit on the training data [Bis06, Chaper 1]. To illustrate this concept, consider a simple least-squares regression where p_{data} is such that for each pair x and y are drawn from the following distributions:

$$x \sim \mathcal{U}(0,1) \qquad y \sim \mathcal{N}(x^2 + \frac{1}{2}, 0.1)$$

Figure 4.5a shows an ordinary least squares fit with polynomial basis functions up to degrees 1, 3 and 5[3]. Here, the training data consists of 5 samples drawn from p_{data}. We can see, that for degree 5, the OLS approximates the samples perfectly — but it

[3]The code for this example can be found in Appendix A.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
Bibliothek, Your knowledge hub
TU WIEN



(a) No Regularization.

(b) Regularized Model.

Figure 4.6: Loss over Epoch for different degrees of regularization.

does not capture the unknown underlying distribution p_{data}. Figure 4.5b, on the other hand, shows a ridge regression fit of the same example. Ridge regression also optimizes a sum of squares error, but with an additional error term $\lambda \|\mathbf{w}\|^2$. Here, $\mathbf{w}$ are the coefficients of the given polynomial terms. This is a typical example of regularization — the additional term introduces a tradeoff between model complexity and the sum of squares interpolation error.

The concept of regularization is also heavily utilized in neural networks [BGC17, Chapter 7]. Typically the model's layout — its number of layers, neurons per layer — is fixed while some regularization parameter is tweaked to reach the desired degree of generalization. This means that the models representational capacity is kept the same — while its effective capacity is adjusted, dependent on the gains in error reduction.

Commonly used techniques are activity regularizers which add a loss component proportional to the activation $a = \mathbf{x}^\top \mathbf{w} + b$ of a neuron — here the regularization term is typically calculated jointly over all activations $\mathbf{a}^{(k)}$ of a layer k via some norm. This is also relevant in Chapter 5.2.1 — in the context of sparse autoencoders.

An alternative approach is dropout — which deactivates a certain number of randomly selected neurons during training. Here, deactivating means that those neurons are neither considered for the forward nor the backpropagation phase.

In practice, the regularization hyperparameters play a crucial role in overall model optimization. It is common practice to monitor the loss for $\mathbb{T}_{test}$ and $\mathbb{T}_{train}$ separately during training. The training progress is often displayed in the form of a learning curve — here, the loss is plotted over training epochs[4]. Figure 4.6 two examples of such learning curves. Learning curves provide a simple way of monitoring the generalization of a given

[4]The code for this example can be found in Appendix A.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

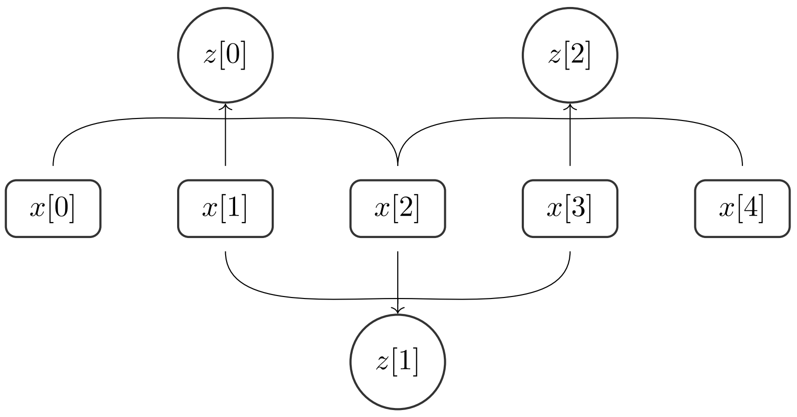


Figure 4.7: 1-D convolutional layer with 1 filter of size 3.

model. In Figure 4.6a, we see that while the training loss does steadily decrease, this is not the case for the test loss. It does increase after epoch 20 — a clear sign of overfitting. Here, the objective of reducing the loss of the training data does not lead to improved performance on the test samples. With increasing epochs, the model gets more and more specialized to features that are distinct to the training set.

The use of learning curves motivates an alternative approach for increasing generalization — early stopping. By simply halting the optimization process after epoch 20, the test loss is minimized for the given model. Figure 4.6b minimizes an objective which is far more representative for all of p_{data}. Even though the training loss does not reach values as low as in Figure 4.6a it clearly outperforms it on the test set. In most cases, the use of specific regularization layers is, however, necessary.

4.1.4 Convolutional Neural Networks & Weight Sharing

In many machine learning applications, there is an inherent structure in the input vector. However, dense neural networks do not directly incorporate any assumptions about the relation of neighboring inputs. Inputs might, for instance, represent measurements of a given quantity over time. In this case, it is reasonable to assume that there exists some correlation over multiple elements of the input vector. While dense networks can also discover such relations — the concept is not inherent to the network structure itself. Whenever it can be assumed that patterns over consecutive vector elements are relevant for the given task — convolutional layers should be considered [LGTB97]. To make this assumption obvious, I will denote such input vectors as series $\mathbf{x} = [x[0], x[1], \dots, x[L-1]]^\top$.

Note that convolutional networks are still feedforward networks. In fact, feedforward networks are often made up of a combination of convolutional and dense layers. Commonly, the convolution layers process the raw input — before the classification or regression task is carried out by dense layers. A detailed overview of convolutional networks is given in [BGC17, Chapter 9]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Figure 4.7 shows the underlying mechanism of a 1-dimensional convolutional layer. The layer processes a 5-dimensional input series $\mathbf{x}$ and produces an output series $\mathbf{z}$ with three elements. The input output relation is given in Equation 4.13.

$$\mathbf{z}_{(0)} = \begin{bmatrix} z[0] \\ z[1] \\ z[2] \end{bmatrix} = \begin{bmatrix} w_0 \cdot x[0] + w_1 \cdot x[1] + w_2 \cdot x[2] \\ w_0 \cdot x[1] + w_1 \cdot x[2] + w_2 \cdot x[3] \\ w_0 \cdot x[2] + w_1 \cdot x[3] + w_2 \cdot x[4] \end{bmatrix} = \begin{bmatrix} \langle \mathbf{w}, \mathbf{x}[0..2] \rangle \\ \langle \mathbf{w}, \mathbf{x}[1..3] \rangle \\ \langle \mathbf{w}, \mathbf{x}[2..4] \rangle \end{bmatrix} \tag{4.13}$$

Note that in 4.13 the weights $\mathbf{w} = [w_0, w_1, w_2]^\top$ are shared across all three neurons. We refer to this collection of weights as a filter — this specific one has size 3. Each convolutional layer learns coefficients of such a filter and slides it across the input domain to produce some output. Thus, each of the weights in $\mathbf{w}$ is applied to more than one input. Note that this so-called weight sharing does not pose any problem to SGD — the backpropagation information does simply include the contribution of each associated input.

Figure 4.7 provides a graphical representation of the mechanism in Equation 4.13. The single filter slides across the input and produces an output with length 3. In practical examples, it is, however, common to apply more than one filter. Also, the input might consist of multiple series. Consider, for instance, measurements from two distinct sensors collected at the same time stamps. We denote this as the input data having two channels. When a 1D convolutional layer is applied to a two-channel input, each filter slides across both channels jointly — with distinct filter coefficients for each of them. The weights are, however, shared across the time domain as in Eq. 4.13. The number of output channels of a layer is thus always the number of filters, no matter the number of channels of the layer input.

Consider the following example where we have $\mathbf{x}_{(0)}$ and $\mathbf{x}_{(1)}$ — the input data consists of 2 channels. The output of a layer with 3 filters is then given by matrix $\mathbf{Z}$ — it consists of 3 channels.

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_{(0)}^\top \\ \mathbf{z}_{(1)}^\top \\ \mathbf{z}_{(2)}^\top \end{bmatrix} = \begin{bmatrix} [z_{(0)}[0], z_{(0)}[1], z_{(0)}[2]] \\ [z_{(1)}[0], z_{(1)}[1], z_{(1)}[2]] \\ [z_{(2)}[0], z_{(2)}[1], z_{(2)}[2]] \end{bmatrix} \tag{4.14}$$

Each of the filters learns a distinct vector of weights for each input channel. Thus, the output of filter 1 ($\mathbf{z}_{(1)}$) at timestamp 1 is given by:

$$z_{(1)}[1] = \left\langle \mathbf{w}^{(1,0)}, \mathbf{x}_{(0)}[1..3] \right\rangle + \left\langle \mathbf{w}^{(1,1)}, \mathbf{x}_{(1)}[1..3] \right\rangle$$

The two inner products follow the same logic as in Equation 4.13. The complete equation is thus:

$$\begin{aligned} z_{(1)}[1] =& w_0^{(1,0)} \cdot x_{(0)}[1] + w_1^{(1,0)} \cdot x_{(0)}[2] + w_2^{(1,0)} \cdot x_{(0)}[3] \\ &+ w_0^{(1,1)} \cdot x_{(1)}[1] + w_1^{(1,1)} \cdot x_{(1)}[2] + w_2^{(1,1)} \cdot x_{(1)}[3] \end{aligned}$$

TU Bibliothek Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar. The approved original version of this thesis is available in print at TU Wien Bibliothek.

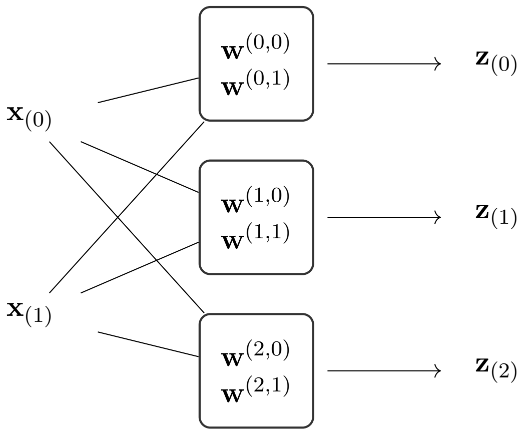


Figure 4.8: Convolutional layer with 2 channel input and 3 filters.

For the convolution along the time domain the same principles as in Figure 4.7 apply, the only difference is that there is now a weight vector $\mathbf{w}^{(i,j)}$ for each filter i and each input channel j.

Figure 4.8 gives an overview of the complete layer — note that this is an abstraction over multiple neurons, to showcase the mechanism behind the processing of distinct channels. As opposed to Figure 4.7, where the representation is — besides the weight sharing — equivalent with the one given in Figure 4.2.

In general, convolutional and dense layers are interoperable with each other. Convolutional layers often extract structural information from time series or image data, while consequent dense layers operate on their output. For this, multiple channel outputs have to be collected into a single vector. We denote this operation by flattening — all channels $\mathbf{Z}$ are concatenated in a 1-dimensional vector $\mathbf{z}$. Subsequent dense layers can then operate on $\mathbf{z}$. Note that the simple reordering of neurons in flattening is fully compatible with backpropagation.

The discussion of upsampling layers concludes the introduction into neural networks in this work. In a nutshell, they provide a powerful toolbox that can be utilized in many different applications. The different choices of activation functions, layers, losses, and network architectures provide the flexibility needed to adapt to a given problem. Still, all of those variants are united by the flow of gradient information via backpropagation, which allows for optimization of the model's parameters by using SGD. In the following Chapter 4.2 representation learning is introduced — an idea that heavily relies on neural networks in its implementations.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

4.2 Representation Learning

Representation learning is a field of machine learning that deals with the automated processing of unlabeled data to extract inherent information from it. More precisely, the objective is to derive representations, which capture the essential explanatory factors describing the data. As such, the setting for representation learning is different from typical supervised tasks. In a nutshell, we aim for a deeper understanding of the data at hand — not for function approximation. As Confuzius puts it this learning by reflection of the data is more subtle than imitating a certain known function:

By three methods we may learn wisdom: first, by reflection, which is the noblest; second, by imitation, which is the easiest; and third, by experience, which is the bitterest.
—Confuzius

Intuitively, representation learning is motivated by the observation that the complexity of a given problem is often closely linked to the representation of the respective data. Well known examples of this are the Laplace or Fourier transform, which are heavily used in engineering. The representation of a signal in the frequency domain, for instance, can drastically simplify certain operations. Calculation of convolution as the product of the frequency representations is just one example of this. In addition to reducing complexity, alternative representations of the same data set can also help to gain a deeper understanding of the data to be analyzed. Some relations are hard to discover in the raw data, while they become obvious as soon as the data is presented in a different space.

This idea is also prevalent in machine learning — preprocessing data before exposing it to a classification or regression algorithm is common practice in the field. Often the input data is standardized, ensuring zero mean and unit variance within each distinct feature. Complex tasks often require more advanced preprocessing steps — in natural language processing, it is, for instance, common to represent sentences in the form of a word embedding.

Those examples provide intuition on how crucial the inherent structure of the input data can be and in what way it might affect the outcome of a given learning task. The feature generation carried out in Chapter 3 can also be seen as an example of this. Here, the input time series $\mathbf{x}$ was mapped into a low dimension features space. Label spreading then operated in this feature space, instead of directly using the raw vectors $\mathbf{x}$. While this processing step increased accuracy, a lot of domain knowledge and manual analysis of the data is required to find the relevant mappings.

In representation learning, we aim to obtain such a mapping to a representation space in an automated fashion [BGC17, Chapter 15]. At the core of this lies the idea that there exists a small number of underlying explanatory factors — which completely describe the variation of $\mathbf{x}$ within p_{data}. Ideally, this new space $\mathbf{h}$ has a form that simplifies subsequent classification or regression tasks and allows for a more intuitive interpretation of the data.

More precisely, we want the encoding to be of such a kind that it offers meaningful insights. Not only for subsequent machine learners but also for human analysts. As

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

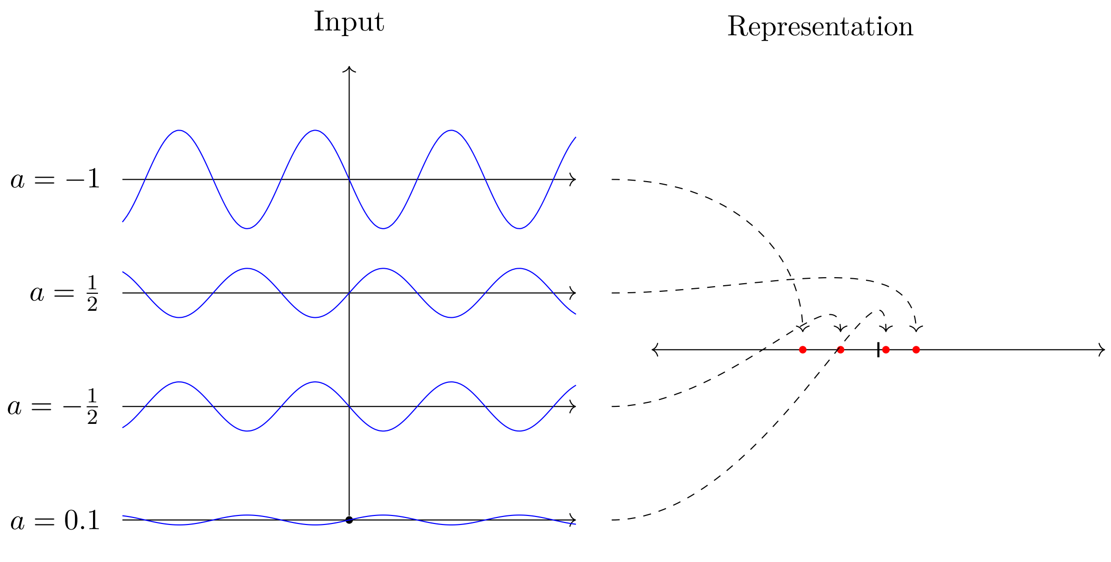


Figure 4.9: Representations learned from sine inputs.

such, we might aim for a representation where related input samples are close in the representation space — given that they share certain properties — even if those samples are far apart in the input space. Also, we might want a representation to naturally form clusters of related samples — and thus to expose hidden connections between samples. As such, the mapping can be described by Equation 4.15.

$$\mathbf{f} : \mathbb{R}^{d_x} \to \mathbb{R}^{d_h}, \qquad \mathbf{f}(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{h} \tag{4.15}$$

Note, that usually we require $d_h < d_x$. While Equation 4.15 might resemble compression, the objective of learning representation is different. Because, we do want the representations in $\mathbf{h}$ to provide deeper understanding — reducing d_h to an absolute minimum by a mapping, which offers no intuitive interpretation, is not sufficient.

Consider the example given in Figure 4.9. Here, p_{data} describes different sine functions $\mathbf{x} = a \cdot \sin(f \cdot \mathbf{t})$ — all with the same frequency f but different amplitudes a. So, the raw data is given in the form of vectors $\mathbf{x}$ from $\mathbb{R}^{d_x}$. Here, d_x is the dimension of $\mathbf{x}$. In the new space, the data can be represented by a single scalar variable h. A single scalar variable h is sufficient to encode all of the variation within the elements $\mathbf{x}$ of p_{data}. Thus, $d_h = 1$. Note, that $\mathbb{R}^{d_h}$ does only encode variations of the input data — common features of all of p_{data} are ignored. For this specific case setting $h = a$ and dropping the frequency f would be sufficient.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

4.2.1 What makes a representation good?

To some extend — the notion of a good representation is ambiguous. It is not straightforward to define what makes a representation good. Note again, that compression is usually not the primary objective of representation learning — but learning "useful" representations is. What makes a representation useful often depends on the specific use case. Thus it is hard to come up with a generic metric. Consider a human analyst trying to interpret the space **h** — how shall we quantify to what extend an analyst benefits from such a representation **h**? In a nutshell, the challenge is to quantify users' needs and then come up with a metric that can guide the learner in this direction. [BCV13] identified the following points as the two most critical aspects. Namely, a representation should:

1. Disentangle the underlying factors of variations.
2. Simplify consecutive learning tasks.

While those two aspects are crucial, it is not apparent how to guide a learner in this direction. Moreover, representation learning is typically done in an unsupervised way. Thus there are no direct clues — in the form of labels — to be used as a learning objective or for benchmarking. Even if the data set is partially labeled — how should we utilize those to gain a representation which is not only valuable for one distinct subsequent task?

This is why, in general, the objectives — in the form of losses — are more subtle for representation learning, and the models make heavy use of regularization schemes. In fact, learning is mostly driven by prior beliefs about the data and structure of the learner. Those prior beliefs motivate the design of the model and implicitly guide the learner.

The following section introduces those assumptions — amongst them are: Multiple explanatory factors, hierarchical organization of explanatory factors, shared factors across tasks, and sparsity. Note that I do mainly focus on aspects that are relevant for autoencoders — the complete list can be found in [BGC17, Chapter 15].

Multiple explanatory factors

A common assumption is that there is a distinct number of explanatory factors, which describe the input data entirely. Once the representation learner reveals those factors, successive algorithms can easily solve the remaining part of the problem. More specifically, this means that we assume that there is some **h**, which completely describes the variation of different **x**. Successive learning tasks can directly operate on **h**. Autoencoders incorporate this assumption — the latent layer models **h** directly.

Hierarchical organization of explanatory factors

In addition to the assumption that such explanatory factors exist, we might further assume those factors to be hierarchically organized. This assumption is implicitly part of each deep neural network. In fact, a NN can be interpreted as a concatenation of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

representation learners — the output of each layer is a representation of the predecessor. Thus, NNs provide a hierarchical set of representations by default. Additionally, those representations become more abstract with increasing depth [SZT17]. Note that the last layer in a regression task utilizes linear activation functions — thus, the last layer is a linear model operating on an abstract representation. As previous layers process the data before it reaches the final one, a linear model is sufficient to solve the learning task.

Shared factors across tasks

While most representation learning algorithms are unsupervised, some semi-supervised or supervised approaches exist. Often they have the form of multiple distinct supervised tasks operating on the same data. More specifically, there might be different label sets for each task — but all use the same $\mathbf{x}$ as the input. How can we now utilize labels of such form for representation learning?

The idea here is to learn those tasks jointly and, thus, come up with a common representation $\mathbf{h}$, which benefits all tasks. The underlying assumption here is, that each of the consecutive tasks can be solved by a subset of $\mathbf{h}$ and, more importantly, that those subsets are overlapping. Only when the subsets are overlapping, do the tasks benefit from a joint representation. Then, the joint learning of those tasks can outperform individual processing. Semi-supervised or supervised autoencoders implicitly incorporate those ideas [LPW18].

Sparsity

Some representation learners also build on a sparsity assumption of $\mathbf{h}$. The assumption here is that — of the underlying features collected in $\mathbf{h}$ — only a small subset is active for a given sample $\mathbf{x}_n$. Consider, for instance, an abstract feature, which uniquely describes a subset of p_{data}. Thus, the corresponding dimension in $\mathbf{h}$ should only be active if $\mathbf{x}$ does, in fact, share this feature. We can guide the learner in this direction by imposing some activity regularization on the latent layer output — this is the core idea of sparse autoencoders. Additionally, this constraint helps with disentangling features — sparsity penalizes the learner whenever different features in the representation space fire jointly.

The assumptions presented in this list lay out the basic objectives and requirements for any representation learner. Still, they are articulated in a generic way. In Chapter 5 I will introduce autoencoders as a practical example. There, some of the above assumptions are indirectly present, while others are explicitly enforced to obtain a representation with the desired properties.

4.2.2 Manifolds

Manifolds offer another way of thinking about representation learning. The idea of a manifold provides a geometric interpretation of the process of learning representations. In fact, most representation learning algorithms do implicitly build on top of a manifold

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

hypothesis [BCV13]. The manifold hypothesis refers to the idea that there is some kind of manifold $\mathcal{M}$ in the input data. This means that samples from p_{data} which are vectors with high dimensionality d_x are concentrated around a manifold $\mathcal{M}$ with dimension $d_{\mathcal{M}}$. If $d_x > d_{\mathcal{M}}$ learning the coordinate system of the manifold $\mathcal{M}$ gives a suitable representation **h**. An example of a linear manifold learning algorithm is Principal Component Analysis (PCA) [Bis06, Chapter 12]. Here, the input is linearly transformed into the direction of highest variation. Data manifolds for complex real-world domains are, however, expected to be strongly nonlinear. The power of deep learning is that it can obtain the necessary nonlinear projections onto the new coordinate systems. Consider again the example in 4.9. When plotting p_{data} in the frequency domain, we will observe that the input space is sparse. The samples will be concentrated along a a straight line representing the amplitude of the single frequency component present in p_{data}. The data can thereofore be described by a 1 dimensional manifold $\mathcal{M}$. In general those manifolds tend to be of

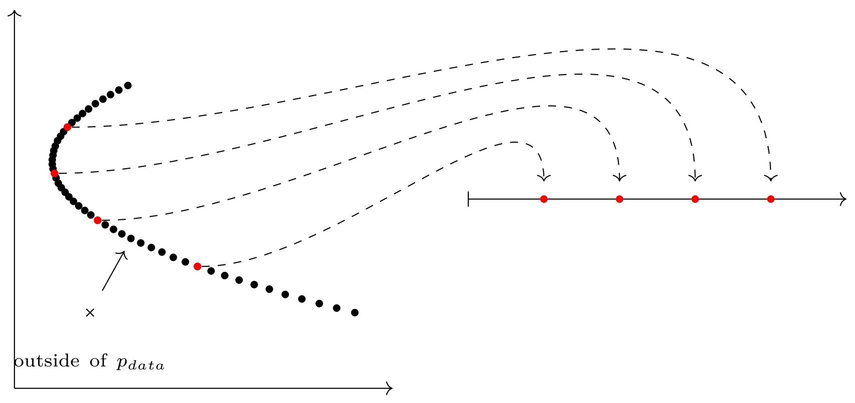


Figure 4.10: Parabolic manifold.

a more complex shape. Figure 4.10 shows an example of such a manifold. Here, p_{data} concentrates around a parabolic shape in $\mathbb{R}^{d_x}$. A manifold learner would, for instance, obtain a mapping into the 1 dimensional representation space $\mathbb{R}^{d_h}$ shown on the right. Note that points not lying on $\mathcal{M}$ are not considered in this mapping. As a consequence of this, the low-density regions of p_{data} are not optimized by the learner. Ideally the input sample from outside the manifold is mapped to a point given by the orthogonal projection onto the manifold. Figure 4.10 shows an example of such a sample. This means that the learner is invariant to changes in the input space which are orthogonal to the manifold.

All in all, learning representations– either via the manifold assumptions or one of the other aspects introduced above — is especially helpful when dealing with large amounts of high dimensional unlabeled data. For instance, it can be used to extract the essential features out of publicly available crowdsourced data sets. These representations can then

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

be used to gain additional insight into the data at hand or act as a preprocessing step for subsequent learners. The idea of representation learning plays a crucial role in this work: The following Chapter , introduces the concept of an autoencoder — an implementation of representation learning based on deep learning. Finally, in Chapter 6 representation learning is used to analyze crowdsourced network benchmarks.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Autoencoders

In a nutshell, an autoencoder is a NN which is used for representation learning — see Section 4.2. This means that an autoencoder maps a given input $\mathbf{x}$ into a representation space. Here, the core objective is to extract the underlying features of variations in the input data $\mathbf{x}$. This way, autoencoders can not only compress the input data but do also highlight the key characteristics of $\mathbf{x}$. Besides the use case for representation learning, autoencoders also fall in the realm of generative approaches — which means that they can generate new samples from the input distribution p_{data}.

In this chapter, I will first introduce the basic notion of an autoencoder. In the following, I discuss different autoencoder variations such as variational autoencoders, conditional autoencoders, as well as supervised and semi-supervised autoencoders. I subsequently provide different simulation results for toy data sets to showcase the discussed aspects in practice. Finally, I briefly mention different related techniques that help analyze the representation space — those include Kernel-Density-Estimation, Clustering, as well as approaches for visualization of high dimensional data sets.

5.1 Basics

An autoencoder consists of an encoder $\mathbf{e}(\mathbf{x})$ and a decoder $\mathbf{d}(\mathbf{h})$, which are trained jointly [BGC17, Chapter 14]. The encoder does thereby learn a representation into the so called latent space $\mathbf{h} = \mathbf{e}(\mathbf{x})$, while the decoder aims to reconstruct the original input from the representation $\hat{\mathbf{x}} = \mathbf{d}(\mathbf{h})$. I denote the dimension of $\mathbf{h}$ by d_h. Thus, the overall input output relation is $\hat{\mathbf{x}} = f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{d}(\mathbf{e}(\mathbf{x}))$.

Figure 5.1 shows such a scheme. Note that the encoder and decoder are normal feedforward networks. Commonly, it is beneficial to deploy specific layers for the decoder (Upsampling, ConvTranspose) — but dense layers are sufficient in most cases. The use case for such an NN architecture might not be apparent at first, but in the end, we are not interested

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

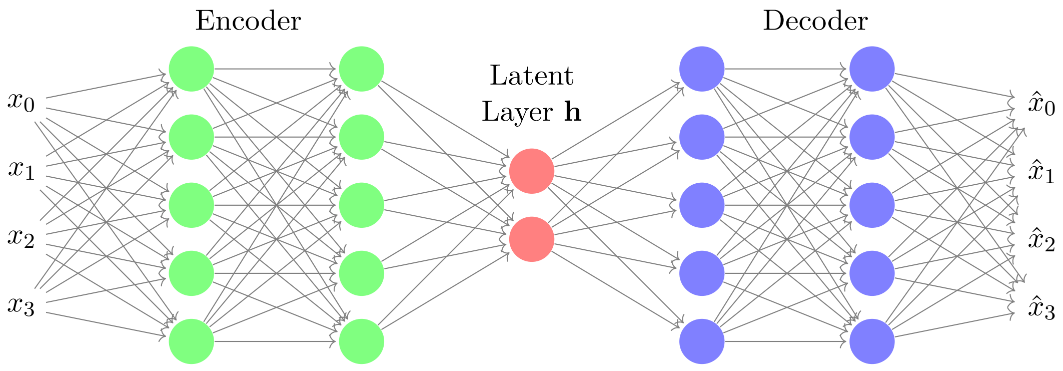


Figure 5.1: Basic autoencoder scheme.

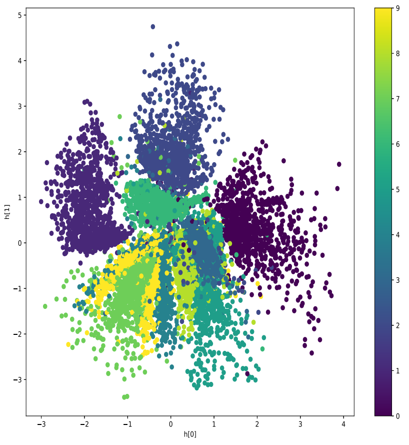
in the composition of the two networks, but in the individual components $\mathbf{e}(\cdot)$ and $\mathbf{d}(\cdot)$. Furthermore, the perfect reconstruction of $\mathbf{x}$ at the output is not the primary objective here — there is actually an inherent tradeoff between reconstruction and representation when dealing with autoencoders. A model with unlimited capacity — both for $\mathbf{e}(\cdot)$ and $\mathbf{d}(\cdot)$ — can theoretically learn a mapping which can encode all information about $\mathbf{x}$ into a single latent neuron. It effectively learns the identity function $\mathbf{x} = f(\mathbf{x};\boldsymbol{\theta}) = \mathbf{d}(\mathbf{e}(\mathbf{x}))$ [BGC17, Chapter 14].

This might be beneficial for compression tasks, but in representation learning, we aim for a network that provides a useful representation of $\mathbf{x}$. Simply reducing the reconstruction loss — for instance $L_{\text{MSE}}(\hat{\mathbf{x}}, \mathbf{x})$ — is not sufficient. We can only obtain a meaningful representation when the encoder has to extract the most relevant features of the input — and does not encode the complete $\mathbf{x}$ in an abstract scalar value [BGC17, Chapter 14]. To ensure this, the models capacity has to be limited artificially — but instead of changing the model's structure — number of neurons, number of layers — we usually deploy different regularization techniques. Some of them are quite generic to neural networks, while others are specifically designed to enforce structure in the representation space and to utilize the assumptions discussed in 4.2.

A common form of regularization is incorporated by the denoising autoencoder [VLL+10]. Here the input samples $\mathbf{x}$ are not passed to the autoencoder directly, instead a corrupted version $\mathbf{x} + \mathbf{n}$, where $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$, is used. The training objective is thus to learn the uncorrupted input from the noisy samples. This forces the encoder to extract the relevant features of $\mathbf{x}$ in $\mathbf{h}$ and prohibits it from learning the identity function. Besides offering a representation learner in the form of the encoder $\mathbf{e}(\cdot)$, autoencoders do also fall in the category of generative models. After training, the decoder $\mathbf{d}(\cdot)$ can be used separately from the encoder. It provides a way to generate new samples from p_{data} — also ones that were not part of the training set.

Figure 5.2 gives an example of an autoencoder processing the well known MNIST data set

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.


(a) Encoding

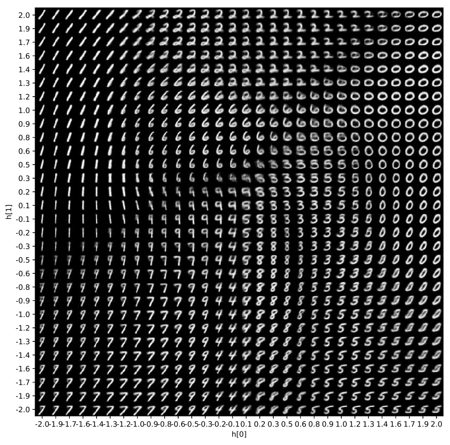

(b) Decoding

Figure 5.2: MNIST processed by variational auto encoder.

of handwritten digits [1]. Figure 5.2a does thereby show a two dimensional representation **h** learned by the encoder — each of the distinct points in 5.2a represents one image. The colors refer to the given labels — this shows that each digit class forms clusters in the latent space. The output of the decoder is given in Figure 5.2b. Here the latent space **h** is sampled equidistantly, and the corresponding output images are plotted. It is interesting to see how the latent space provides a smooth transition from digits that are related in their structure. The digits 1, 7, and 9, for instance, share similar features in **h**. The same is true for 0, 3, 5, and 8.

5.1.1 Network Layers for Decoding

In general, the layers used for autoencoders do not differ from the ones used typical classification or regression tasks. In some cases, it can, however, be beneficial to utilize specific layers for $\mathbf{d}(\cdot)$, which specifically take the requirements for decoding into account.

When we consider typical classification tasks, NN models usually consist of a high dimensional input vector $\mathbf{x}$, which is passed through thinner and thinner layers before the final layer produces a single output. In a sense, each of those layers introduces an additional level of abstraction, as it processes the outputs of the preceding neurons. This notion of increased abstraction is even more prominent for concatenated convolutional layers — compare Section 4.1.4. So, in a way, most network layers are inherently designed for an encoding use case — and not directly with decoding in mind.

[1] Results obtained with code from `https://github.com/keras-team/keras-io/blob/master/examples/generative/vae.py`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

The task of the decoder $\mathbf{d}(\cdot)$ is different from the encoder in a sense, as it upsamples its input $\mathbf{h}$ instead of abstracting over it. This observation motivates the introduction of several layers, specifically for decoding.

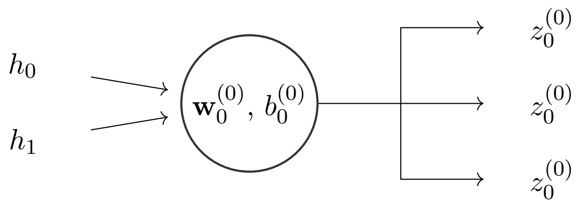


Figure 5.3: Upsampling layer.

In the simplest case, this can be done with the so called upsampling layer [2]. Figure 5.3 depicts such a layer with an upsampling factor of 3. The layer simply copies the output three times — it generates a three-dimensional output from its inputs. Of course, the naive implementation of upsampling does not make sense in every scenario. However, in some use cases, we might have an output vector where some neighboring elements are always the same.

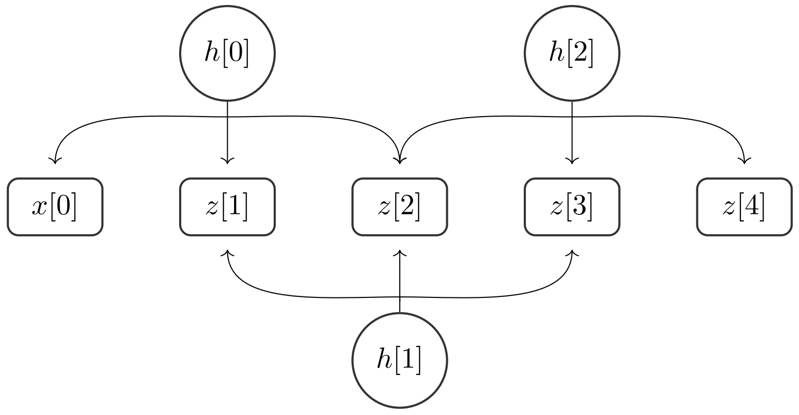


Figure 5.4: Transposed Convolutional Layer.

A more advanced approach to upsampling is the transposed convolutional layer [DV16] — it is depicted in Figure 5.4. This layer has one channel and a filter size of 3. In contrast to the classical convolutional layer in Figure 4.7 the positions of the inputs $\mathbf{h}$ and the outputs are inverted. Still, ConvTransposed is not the inverse operation of the convolution in a strict sense. It just applies the ideas of the convolutional operation to an upsampling procedure. Besides that, the operations are more or less equivalent. $z[2]$ is for instance given by the following expression:

$$z[2] = \sigma\left(w_2 \cdot h[0] + w_1 \cdot h[1] + w_0 \cdot h[2]\right)$$

[2]See `https://keras.io/api/layers/reshaping_layers/up_sampling1d/`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Note that for multiple filters and channels, similar relations as for the normal convolutional layers are in place. A comprehensive discussion of the topic can be found in [DV16].

5.2 Autoencoder Variants & Architectures

In order to obtain a "useful" representation, autoencoders must be prohibited from learning the identity function. Over the years, different autoencoder variants emerged, each tackling this issue in its own way. We denote the typical autoencoder variant shown in Figure 5.1 as a vanilla autoencoder. Without any regularization, such a model does only provide a naive framework for dealing with overcapacity. It solely relies on an encoder and decoder network with limited capacity. Such a model with limited capacity is referred to as undercomplete. Because of those shortcomings, vanilla autoencoders without regularization are not often used in practice.

The, already discussed, denoising autoencoder solves this issue by utilizing additive noise on the input samples. Contractive autoencoders follow a similar approach — a penalty on large derivates in the encoding **h** ensures that the autoencoder is less sensitive to small variations in the input **x**. A comprehensive overview of different autoencoder variants is provided in [BGC17, Chapter 14].

Besides the discussed variants of autoencoders — which deploy specific regularization strategies — there are also different autoencoder architectures. While the typical unsupervised architecture was already shown in Figure 5.1, also supervised extensions exist. The conditional autoencoder, on the other hand, offers a framework to include meta-data into the learning process. In general, those architectures can be implemented with any of the available autoencoder variants.

In the following, I will first introduce two commonly used variants in detail: Namely Sparse Autoencoders (SAEs) and the Variational Autoencoders (VAEs). Consequently, I discuss different architectures. Here, I start with the conditional autoencoder and subsequently introduce the semi-supervised / supervised architecture.

5.2.1 Sparse Vanilla Autoencoders

In a sense, th Sparse Autoencoder is the simplest approach to regularized autoencoders. In fact, its's structure does not differ at all from a generic vanilla autoencoder in Figure 5.1. In contrast to the VAE — which will be discussed in the following section — SAEs are entirely deterministic, and the latent layer is simply a typical network layer with a linear activation function. While the structure of the network mimics an undercomplete vanilla autoencoder — the loss function is different [N+11].

$$\mathcal{L}_{sae}(\mathbf{x}, \boldsymbol{\theta}) = \mathcal{L}_{rec} + \mathcal{L}_{reg} = \mathcal{L}\left(\mathbf{x}, \hat{\mathbf{x}}\right) + \lambda \sum_{i} |h_i| \tag{5.1}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Equation 5.1 defines the loss of a SAE as the sum of a reconstruction term $\mathcal{L}_{rec}$ and an regularization term $\mathcal{L}_{reg}$, which is enforced on the latent layer $\mathbf{h}$. Here, λ is a hyperparameter which controls the extend of the regularization. Effectively, the regularization term $\mathcal{L}_{reg}$ is equivalent to any other activity regularizer — see Chapter 4.1.3. In 5.1, the loss $\mathcal{L}_{reg}$ does, however, specifically utilize the l_1 norm. This means that $\mathcal{L}_{sae}(\mathbf{x}, \boldsymbol{\theta})$ encourages models with a sparse latent space $\mathbf{h}$. The motivation behind this sparseness assumption was already discussed in the representation learning Section 4.2.

While the use of l_1 norms for sparseness constraints is an common concept, it might not be apparent at first sight how absolute values induce sparseness. For this, consider the generic definition of the l_p norm in Equation 5.2 [FR17, Appendix A].

$$l_p = \|x\|_p := \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} \tag{5.2}$$

Here, $p \geq 1$. The most commonly used norm is probably the l_2 norm — mainly due to its relative simplicity in the subsequent optimization procedure. In general, smaller values of p lead to a higher penalty on non-sparse values of $\mathbf{x}$. To see why sparsity is related to the value of p, it is helpful to look at the edge cases. As such, l_∞ is the the maximum norm $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$. Setting $p = 0$ results in $\|x\|_0$ which counts the number of nonzero elements of $\mathbf{x}$. Note, that $\|x\|_0$ is not a norm — still, it is interesting that it acts as a direct measure of sparseness. Finally, choices of $0 < p < 1$ result in so called quasi-norms — for those the triangle inequality is not fulfilled [FR17, Appendix A].

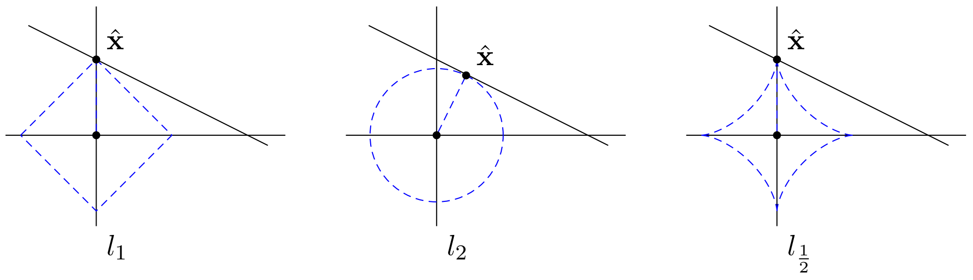


Figure 5.5: Contours of l_1, l_2 and the quasi-norm $l_{\frac{1}{2}}$.

Figure 5.5 depicts the contours of the norms l_1 and l_2 for $n = 2$ — for reference, the quasi-norm $l_{\frac{1}{2}}$ is shown. This means that the shapes in Figure 5.5 are described by the values of $\mathbf{x}$, which lead to a constant value of the respective expressions. Additionally, an arbitrary error surface is indicated by the straight lines. When minimizing this error objective and the regularization term at the same time, the result is given by the intersections indicated by $\hat{\mathbf{x}}$. Figure 5.5 clearly shows, that for $l_{\frac{1}{2}}$ and l_1 those intersections are likely to lead to sparse values of $\mathbf{x}$ [Bis06, Chapter 3].

By enforcing such a sparseness constraint onto the latent layer $\mathbf{h}$, SAEs introduce a tradeoff between the reconstruction $\mathcal{L}_{rec}$ and the regularization $\mathcal{L}_{reg}$ terms. This means

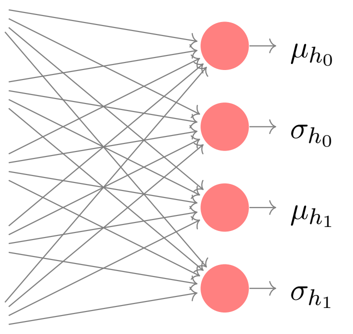
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

that an additional feature encoded into $\mathbf{h}$ needs to reduce $\mathcal{L}_{rec}$ significantly — otherwise, the overall loss increases. Besides this additional loss SAEs do not differ from generic vanilla autoencoders. VAEs, discussed in the next section, break with this structure, by introducing a probabilistic treatment of the latent layer $\mathbf{h}$.

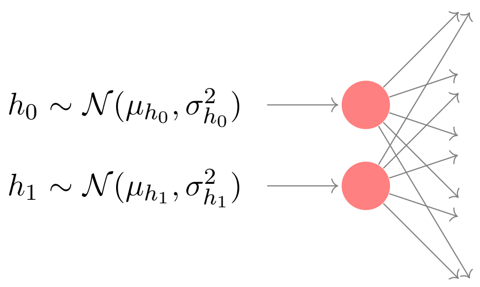
5.2.2 Variational Autoencoders

A Variational Autoencoder is a kind of autoencoder, that is often used for generative approaches [BGC17, Chapter 14]. This means that it provides a systematic way of generating samples, which are in p_{data}, but were not part of the training set. Thus, the output $\hat{\mathbf{x}}$ consists of variations of the original samples but comes from the same underlying distribution. In VAEs this is achieved by modeling the latent layer $\mathbf{h}$ as probabilistic. This probabilistic treatment is not only beneficial for generative approaches, but does also act as a regularization strategy. I discuss the benefits of this probabilistic treatment in detail in 5.2.2.

Instead of learning a deterministic mapping, the encoder in VAEs predicts moments of a probability distribution in the latent space for each given input sample $\mathbf{x}$. More specifically, the output of the encoder describes $p_\theta(\mathbf{h}; \mathbf{x}_n)$ via its moments collected in $\mathbf{h_m}$. During decoding, one or multiple samples $\mathbf{h}$ are drawn from the distribution described by $\mathbf{h_m}$. Those samples do subsequently act as the input for the decoder, which again predicts $\hat{\mathbf{x}}$. Thereby, the objective of reducing some reconstruction loss between $\mathbf{x}$ and $\hat{\mathbf{x}}$ is still in place. Note that besides the latent layer, all components of the VAE are still completely deterministic. This means that $\mathbf{e}(\cdot)$ and $\mathbf{d}(\cdot)$ are made up of classical dense or convolutional feedforward layers.



(a) Encoder


$h_0 \sim \mathcal{N}(\mu_{h_0}, \sigma^2_{h_0})$
$h_1 \sim \mathcal{N}(\mu_{h_1}, \sigma^2_{h_1})$

(b) Decoder

Figure 5.6: Variational Autoencoder Latent Layer.

Figure 5.6 shows the basic scheme for a two dimensional latent space $\mathbf{h}$. Note that the encoder has 4 outputs. Here $\mathbf{h}_m = [\mu_{h_0}, \sigma_{h_0}, \mu_{h_1}, \sigma_{h_1}]^\top$ collects the moments of the bivariate normal distribution $\mathcal{N}(\mu_{h_i}, \sigma^2_{h_i})$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

This probabilistical approach is also used during training — for each $\mathbf{x}$ the encoder predicts a moment vector describing the multivariate normal distribution. Consequently a sample vector $\mathbf{h}$ is drawn from the distribution $p_\theta(\mathbf{h}; \mathbf{x})$ and passed on to the decoder. $\mathbf{d}(\mathbf{h})$ predicts $\hat{\mathbf{x}}$ and some reconstruction loss $\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}})$ is calculated and backpropagated throughout the network.

As a consequence, also this probabilistic interpretation of the in the latent layer has to be differentiable. A reformulation of the sampling procedure — the so-called reparametrization trick can ensure this [BGC17, Chapter 14].

$$h = \mu + \sigma \cdot \epsilon \tag{5.3}$$

In a nutshell, the reparametrization trick in Equation 5.3 expresses the sampling from a normal distribution $\mathcal{N}(\mu, \sigma^2)$ as a the sum of μ and a sample $\epsilon \sim \mathcal{N}(0, 1)$ weighted by σ. This allows for the evaluation of the derivatives with respect to the encoder outputs μ and σ in the following way:

$$\frac{\partial h}{\partial \mu} = 1, \qquad \frac{\partial h}{\partial \sigma} = \epsilon \tag{5.4}$$

Formulated in this way, the sampling procedure does not pose any problems to backpropagation as ϵ is — from a backpropagation point of view — equivalent to the noise in denoising autoencoders. Thus, VAEs can be trained using SGD.

Again, consider the SAE introduced earlier. Here, the overall loss consisted of a reconstruction and a regularization loss. While $\mathcal{L}_{rec}$ was imposed on the output $\hat{\mathbf{x}}$, $\mathcal{L}_{reg}$ ensured that the model does not learn the identity function. VAEs also incorporate an additional regularization penalty on the latent layer. Here, the latent layer $\mathbf{h}$ is of a probabilistic nature. Thus the penalty needs to operate on distributions. In most implementations this regularization term is based on the Kulback Leibler Divergence (KLD) — thus KLD is explained in detail in the next section, before the complete VAE loss is derived.

Kulback Leibler Divergence

The KLD is a commonly used measure of similarity between two distinct probability density functions $p(x)$ and $q(x)$. At its core KLD is based on the difference in information content of the given distribution pair [Oda19].

$$\Delta I(x) = I_p(x) - I_q(x) = -\log p(x) + \log q(x) = \log\left(\frac{q(x)}{p(x)}\right) \tag{5.5}$$

Interpreting the log of a probability as a measure of information content is common in information theory — consider shannons definition of entropy. The KLD between $p(x)$ and $q(x)$ is now the expectation of $\Delta I(x)$ over $p(x)$.

$$D_{KL}(P\|Q) := \int_{-\infty}^{+\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx \tag{5.6}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Equation 5.6 takes on values in the interval $[0, \infty)$. This means that the KLD between $p(x)$ and $q(x)$ is only zero if the two pdfs are equivalent. Thus $D_{KL}(P\|Q) = 0$ if and only if $p(x) = q(x)$

Also note, that the KLD is not a metric. The reason for this is that $D_{KL}(P\|Q) \neq D_{KL}(Q\|P)$ — Equation 5.6 is not symmetric.

Variational Autoencoder Loss

For VAEs the KLD is used, to force the individual latent layer distributions $p_\theta(\mathbf{h}; \mathbf{x})$ to be close to a standard normal distribution $\mathcal{N}(0, 1)$. This means that $\mathbf{h}_m$ should consist of means close to 0 and variances close to 1. Without such a constraint, the VAE could effectively sidestep the probabilistic nature of the latent layer and simply set the variances to be 0 — thus, would render the overall network purely deterministic again.

This constraint is now incorporated in the overall loss of an VAE in form of the regularization term L_{reg}:

$$\mathcal{L}_{vae}(\mathbf{x}_n; \boldsymbol{\theta}) = \mathcal{L}_{reg} + \mathcal{L}_{rec} = d_{\mathrm{KL}}\left(p_\theta\left(\mathbf{h}; \mathbf{x}_n\right) \| p(\mathbf{h})\right) + \frac{1}{L} \sum_{l=1}^{L} \mathcal{L}_{MSE}\left(\hat{\mathbf{x}}_l, \mathbf{x}_n\right) \tag{5.7}$$

The reconstruction term in Equation 5.7 is simply the average MSE over L distinct decoded samples $\hat{\mathbf{x}}_l = \mathbf{d}(\mathbf{h}_l)$ drawn from the latent distribution $p_\theta(\mathbf{h}; \mathbf{x}_n)$. Meanwhile the regularization term $\mathcal{L}_{reg}$ is given by the KLD between the latent space distribution $p_\theta(\mathbf{h}; \mathbf{x}_n)$ and the prior $p(\mathbf{h})$.

When we assume independent multivariate normal distributions and define $p(\mathbf{h})$ to be of zero mean and unit variance the regularization term can be calculated as:

$$d_{\mathrm{KL}}\left(p_\theta\left(\mathbf{h}; \mathbf{x}_n\right) \| p(\mathbf{h})\right) = -\sum_{i=1}^{d_h} \frac{1}{2}\left[1 + \log\left(\sigma_i^2\right) - \sigma_i^2 - \mu_i^2\right] \tag{5.8}$$

Here, d_h describes the dimension of the latent space. The overall loss function is subsequently given by:

$$\mathcal{L}_{vae}(\mathbf{x}_n; \boldsymbol{\theta}) = -\sum_{i=1}^{d_h} \frac{1}{2}\left[1 + \log\left(\sigma_i^2\right) - \sigma_i^2 - \mu_i^2\right] + \frac{1}{L} \sum_{l=1}^{L} \mathcal{L}_{MSE}\left(\hat{\mathbf{x}}_l, \mathbf{x}_n\right) \tag{5.9}$$

While Equation 5.9 describes the complete loss function of the classical VAE, there is also the extension of the beta-VAE [HMP$^+$17]. It introduces an additional scaling term for L_{reg}.

$$\mathcal{L}_{vae} = \beta \cdot \mathcal{L}_{reg} + \mathcal{L}_{rec} \tag{5.10}$$

Here, β is a hyperparameter and controls the weight of the individual loss components. For $\beta = 1$ beta-VAE and VAE are equivalent.

I did already mention the primary motivation behind this probabilistic treatment of the latent layer — it acts as a regularization scheme and provides a framework for sampling unseen variations of the original p_{data} distribution. In the following, I will try to provide further intuition on how this is achieved.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Benefits of the probabilistic treatment?

The benefits of the probabilistic treatment of the latent layer might not be apparent immediately. A key aspect of the latent layer modeling the distribution $p_\theta(\mathbf{h}; \mathbf{x})$ is that the induced randomness acts as some kind of a regularization scheme — limiting the effective capacity of the network. Similar to the already discussed idea of the denoising autoencoders, the uncertainty introduced by the sampling layer limits the network from learning the identity function [BGC17, Chapter 14].

Additionally, the latent space of a VAE tends to be smooth [BGC17, Chapter 14]. More specifically, $d(\mathbf{h}) \approx \mathbf{h} + \epsilon \cdot \mathbf{u}$ — where $\mathbf{u}$ is a unit vector. This means that the immediate neighborhood of a sample in the latent space does again belong to the same class. This is a consequence of the sampling procedure. In fact, it is possible to generate multiple variations of a single input vector $\mathbf{x}$ by sampling the related distribution. Additionally, the zero mean constraint encourages the individual $p_\theta(\mathbf{h}; \mathbf{x})$ to overlap. Subsequently, the transitions between different samples tend to be smooth as well. This can also be observed in Figure 5.2b.

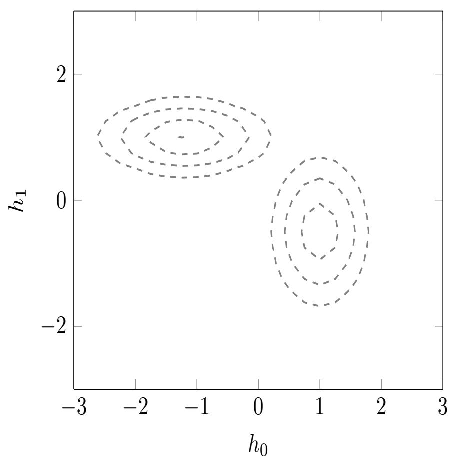


Figure 5.7: Distribution of two samples in latent space.

The probabilistic scheme is also beneficial to one of the core objectives in representation learning — the disentanglement of the distinct underlying explanatory factors [BHP+18]. Keep in mind, that the moments $\mathbf{h}_m$ learned by the encoder describe a multivariate normal distribution with independent elements. In Figure 5.7 two distinct $p_\theta(\mathbf{h}; \mathbf{x})$ are shown schematically — note that h_0 and h_1 are uncorrelated and thus independent. This independent assumption does, therefore, explicitly encode the disentanglement objective into the model's structure. The learner is encouraged to learn a representation where each element in $\mathbf{h}$ describes an independent feature of $\mathbf{x}$. Because all samples still share the same latent space and the individual distributions $p_\theta(\mathbf{h}; \mathbf{x})$ are overlapping those features are learned jointly.

The disentanglement of the underlying factors of variation can, again, be seen in Figure

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5.2b. It is apparent that h_0 and h_1 represent to distinct features of p_{data}. While on the interval of h_0 samples follow a transition from straight to round shapes, h_1 seems to encode a more ambiguous feature that spans from simple connected shapes to more complex combinations of strokes.

The β-VAE does further provide a way to control the desired degree of disentanglement — note that there is a tradeoff with the reconstruction objective. As the disentanglement is caused by the regularization term $\mathcal{L}_{reg}$, it is no surprise, that an increased β does, in general, lead to increased disentanglement [BHP+18].

Even though the two discussed variants of autoencoders SAEs and VAEs are different in their approach — they still follow similar objectives. Both incorporate some kind of regularization loss on their latent layer, in order to guide the learner towards a disentangled representation. The primary difference is, however, that the VAE offers a distinct framework for generative sampling, while the SAE does not. In the end, the decision on which autoencoder to use is often dependent on the data at hand at the specific use case.

The treatment of VAEs ends the discussion autoencoder variants in this work. In the following, I will introduce different architectures for autoencoders; in general, they can be used with any of the above discussed variants. Those architectures' primary objective is to extend the classical unsupervised scheme and to adapt to a broader range of use cases.

5.2.3 Conditional Autoencoders

The conditional autoencoder is an autoencoder architecture that extends the typical unsupervised scheme. It is motivated by the observation, that for most problems — while still being unsupervised — some metadata might be available. In the classical autoencoder scheme from Figure 5.1, there is no particular way of dealing with such metadata.

The only way is to include it into $\mathbf{x}$ directly. However, this is suboptimal in two ways: First, the encoder now has to encode the metadata into $\mathbf{h}$, which might require a higher dimensionality of the latent space. Often, we do, however, prefer lower dimensionality, which simplifies the visual representation of $\mathbf{h}$. The second problematic aspect is that the output $\hat{\mathbf{x}}$ does now also include the meta-data, and thus it is considered in the computation of the reconstruction loss. While the decoder can, most likely, deal with this, it increases complexity and does not serve any apparent benefit.

The core idea of the conditional autoencoders is now to provide a designated way of handling meta data [SLY15]. Therefore, the input $\mathbf{x} := \{\mathbf{x}', \mathbf{m}\}$ is split into a vector of metadata $\mathbf{m}$ and the remaining input $\mathbf{x}'$. The input output relations are then given by:

$$\mathbf{h} = \mathbf{e}(\mathbf{x}'; \mathbf{m}) \qquad \hat{\mathbf{x}}' = \mathbf{d}(\mathbf{h}; \mathbf{m}) \tag{5.11}$$

This means that for a conditional autoencoder, $\mathbf{e}(\cdot)$ and $\mathbf{d}(\cdot)$ are parametrized on the metadata — effectively there is a distinct encoder and decoder function for each

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

configuration of $\mathbf{m}$. The overall scheme is also depicted in Figure 5.8. The training procedure for a conditional and an unconditional autoencoder only differs in one aspect — the values of $\mathbf{m}$ also have to be fed into the network.

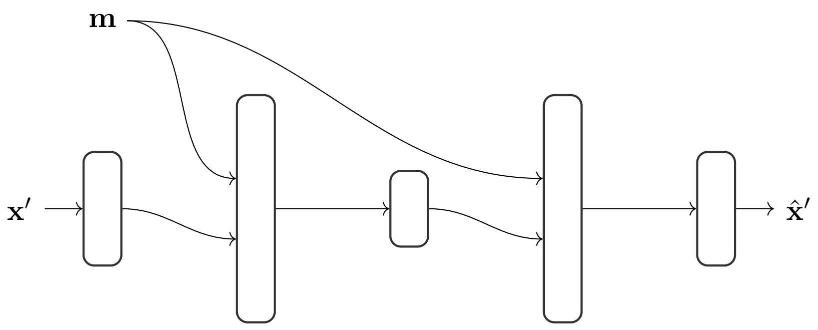


Figure 5.8: Conditional Autoencoder.

This idea of conditioning on the meta-data first sprung up in [SLY15] in the framework of VAEs. The architecture in Figure 5.8 is, however, not limited to VAEs, we are free in the selection of $\mathbf{e}(\cdot)$ and $\mathbf{d}(\cdot)$. This means that we can come up with a sparse-conditional autoencoder, simply by parametrizing the encoder on the meta data $\mathbf{m}$.

There are two aspects of conditional autoencoders which I will cover in detail: the independence of the latent space and the the implication the parameterization has for generative modeling.

Independent latent space

The parametrization on $\mathbf{m}$ in Equation 5.11 has an interesting effect: It encourages a representation $\mathbf{h}$ independent of $\mathbf{m}$. This is because encoding $\mathbf{m}$ into $\mathbf{h}$ would, most likely, lead to increased regularization loss. At the same time, it would not decrease the reconstruction loss, as $\mathbf{m}$ is available to the decoder anyway. Thus $\mathbf{h}$ tends to capture the variations which are not included in $\mathbf{m}$. More precisely, this means that the encoder $\mathbf{e}(\cdot)$ is encouraged to remove the influence of $\mathbf{m}$ from $\mathbf{x}'$.

Consider for instance the autoencoder in Figure 5.2 operating on the MNIST data set. It is clear from from 5.2a, that the latent space $\mathbf{h}$ includes information about the distinct digits shown in the images. While this is beneficial for the classification of digits, we might also develop a conditional scheme that includes the digit information into the meta-data vector $\mathbf{m}$. This way, we would obtain a latent space $\mathbf{h}$, which would not show clusters for different digits, but learn variations of the images over all digits jointly. This could, for instance, be used as a way of finding clusters of different styles of handwriting.

Conditional generative output

The second interesting aspect of the conditional autoencoder regards the decoding procedure. For this, consider the sampling over the latent space in Figure 5.2b. While

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

the digits form clusters and the latent space provides smooth transitions between those respective clusters, it is still nontrivial to obtain a distinct variation of a given digit. We can generate new samples of an arbitrary digit by sampling from the cluster in the latent space, but the control over the specific style of handwriting is, for instance, limited.

Again a conditional autoencoder with the digit information encoded into $\mathbf{m}$ can be used. Then, the decoder $\mathbf{d}(\mathbf{h};\mathbf{m})$ is paramterized on the desired digit, while different configurations of $\mathbf{h}$ incorporate the respective writing style. For this to work, we generally assume that the selected combinations of $\mathbf{m}$ and $\mathbf{h}$ have, in one way or another, been shown to the model during training. Unseen combinations are theoretically possible, but they do require effective regularization to provide reliable results.

5.2.4 Supervised Autoencoders

While autoencoders are in general an unsupervised technique, also supervised variants have been introduced in literature [LPW18]. This is motivated by the observation that most autoencoder applications are followed by some regression or classification task. Supervised autoencoders do now incorporate this subsequent learning task into the autoencoder itself.

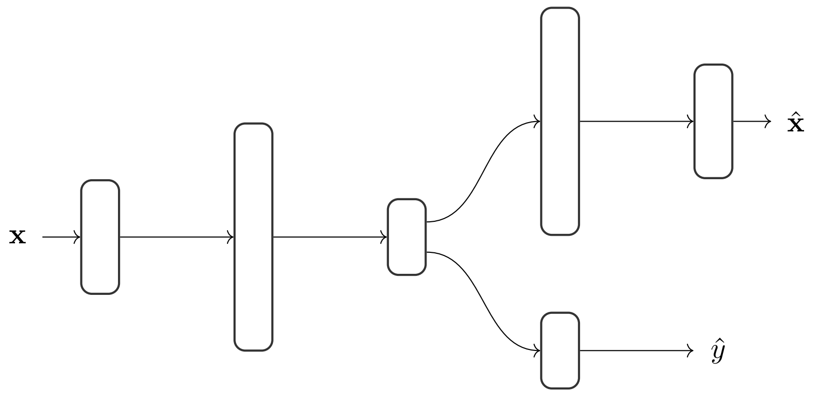


Figure 5.9: Supervised Autoencoder.

The overall structure of such a scheme is illustrated in Figure 5.9. The autoencoder structure is the same as before, but there is an additional branch which outputs an estimate of the label y. Note that this classifier does not operate on $\mathbf{x}$, but instead on the representation $\mathbf{h}$. Thus, it benefits from the entanglement of features learned by the encoder. The classification branch is reflected in the overall loss:

$$L_{super} = L_{reg} + L_{rec} + L_{class} \tag{5.12}$$

The additional term L_{class} in 5.12 introduces the objective of minimizing some missclassification penalty between y and $\hat{y}$. Note, that in Figure 5.9, $\hat{y}$ is obtained from $\mathbf{h}$ via an additional network layer. This layer processes the low dimensional latent space into the output $\hat{y}$ which is often scalar.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

It is essential to be aware of the effects this classification branch has on the overall representation. In a nutshell, the classification branch leads to a representation that highlights the features distinguishing the classes. As stated in Equation 5.12 the same regularization and reconstruction loss components as for the unsupervised autoencoders are activate.

Still, the classification loss affects the structure of the latent space **h** during backpropagation. The objective of decreasing L_{class} encourages a representation that reflects the classes described by y. This tendency is counteracted by the regularization and reconstruction term, which balances the two objectives of obtaining a "meaningful" representation and achieving effective classification.

Note that the complexity of the classification layer also affects the representation space. A layer with high capacity can learn a mapping from a generic representation, while a layer with low capacity requires a designated representation. This means that the capacity of this layer also controls the extent to which the representation highlights the features important to the classification tasks [LPW18].

In 2016 [GM16] proposed an approach to extend autoencoders to semi-supervised approaches. It is, inherently based on the same structure as the one shown in Figure 5.9. The loss is, however, given by:

$$L_{semi} = \begin{cases} L_{reg} + L_{rec} + L_{class} & \text{for labeled samples} \\ L_{reg} + L_{rec} & \text{else} \end{cases} \tag{5.13}$$

For each labeled example, the loss is consistent with Equation 5.12. The approach only differs for unlabeled examples, where the L_{class} is ignored.

The above discussion covered the key aspects of autoencoders. I discussed two different variants, the VAE and the SAE, which differ mainly in the way that they tackle regularization. Additionally, I introduced different autoencoder architectures, which help solve practical problems that might arise when applying autoencoders in a specific use case. In the following, I will provide some toy examples for autoencoders to showcase how the discussed schemes behave in practice.

5.3 Simulations

At the beginning of this chapter, I introduced Figure 5.2 as a motivational example for the use of autoencoders. It depicts a representation of the well known MNIST handwritten digits data set, as obtained by a variational autoencoder. The subsequent pages provide additional examples to illustrate the discussed aspects of autoencoders further.

The primary focus is hereby on concepts introduced above: the structure of the latent space **h**, reconstruction loss, and effects of regularization strategies. In this context, I compare the two autoencoder variants SAE and VAE — see 5.2.1 and 5.2.2. Additionally, I provide results for the conditional autoencoder architecture.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Whereas the example in Figure 5.2 operates on a realistic data set, the following experiments are toy examples. Here, the key objective is not to showcase the abilities of autoencoders but to illustrate the mechanisms discussed above. To analyze those aspects systematically, the first set of experiments operates on simple time-series consisting of rectangular pulses. In a second step, I extend the discussion to more complex inputs. For this, I generate a set of QAM inspired signals and compare the latent space **h** with the constellation in the IQ-plane.

5.3.1 Rectangular Function

The rectangular time series described in Equation 5.14 form the basis for the first set of experiments.

$$\mathbf{m} = \sum_{k=1}^{K} a_k \cdot \sqcap(t - kL; L) \qquad \sqcap(x) = \begin{cases} 1 & |t| \leq \frac{L}{2} \\ 0 & |t| > \frac{L}{2} \end{cases} \tag{5.14}$$

Here $\sqcap(\cdot)$ is the rectangular function. Equation 5.14 defines **m** as a series of concatenated rectangular pulses. Each of those pulses has a distinct amplitude described by a_k.

The primary motivation for a time-series of this kind is that the number of different rectangular pulses K can be adapted while keeping the overall time-series length fixed. This way, I can directly control the number of underlying factors. When the values for a_k are selected independently for each rectangular pulse, those K amplitudes do effectively capture all variation in the time-series. We generally want **h** to pick up those underlying factors, so this setup can be used to analyze the effect of different sizes of the latent layer. Additionally, the symmetry of this time-series ensures unambiguous representations in the latent space, which simplifies the systematic analysis.

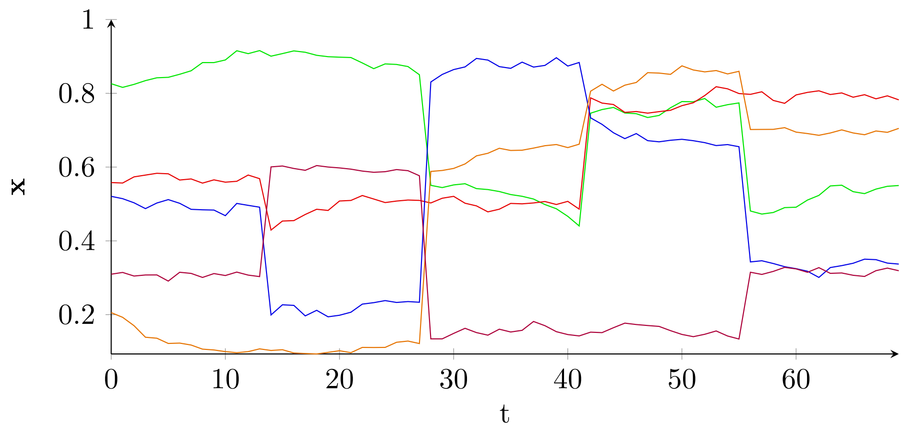


Figure 5.10: Multivariate gaussian with rectangular mean for $K = 5$.

Figure 5.10 makes this more clear — the 5 amplitudes do completely describe the time-series. Due to the symmetry of the data set, each amplitude contributes the same amount

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

of error to the overall reconstruction loss. This means that each of those factors is of equal importance in the representation.

Note, that the time-series in Figure 5.10 include additional variation. This is achieved by using Equation 5.14 as the mean vector for a multivariate Gaussian and by drawing multiple samples for each different a_k configuration. Overall the time series $\mathbf{x}$ in Figure 5.10 is given by $\mathbf{x} = \mathcal{N}(\mathbf{m}, C)$. The reasoning behind using multivariate Gaussians is to enlargen the data set artificially — otherwise, it would only consist of a small number of different signals. Note, however, that this scheme is not related to denoising autoencoders, as the autoencoder's objective is not to extract $\mathbf{m}$ from $\mathbf{x}$, but to reconstruct $\mathbf{x}$ itself.

In practice, I obtain $\mathbf{m}$ by generating combinations of different a_k linearly spaced from the interval of $[0.2, 0.8]$. The covariance matrix C is of such a form that it induces correlation over a distance of 5 time-stamps[3].

I now use two different kinds of autoencoder, a variational one and a sparse vanilla autoencoder — see Chapter 5.2.2 and 5.2.1. Plots of the used models can be found in Appendix A. For each of those, I run experiments with different sizes of latent layers — I denote the latent layer size by d_h. I also show results for conditional autoencoder architectures, which were introduced in 5.2.3.

Reconstruction

In most use cases, the reconstructed series $\hat{\mathbf{x}}$ itself is only a byproduct of the representation learning, whereas the real interest lies in the latent space $\mathbf{h}$. Analyzing the output $\hat{\mathbf{x}}$ can still be interesting, as it exposes the inner workings of the autoencoder — it reveals the features the encoder picked up.

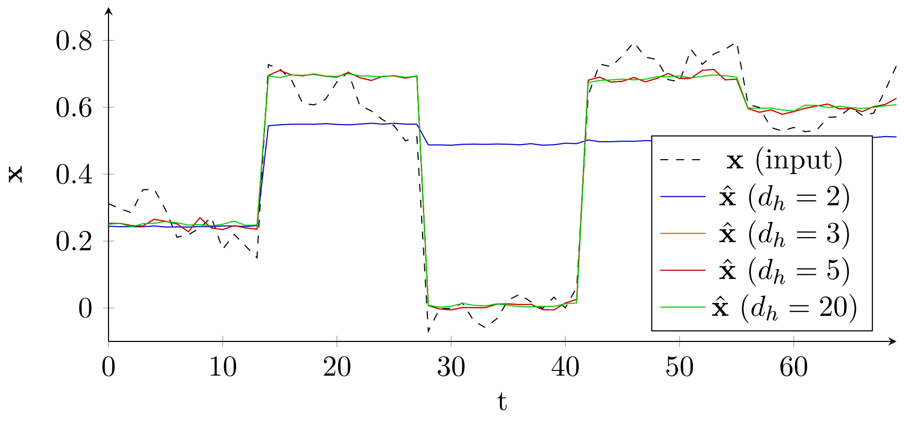


Figure 5.11: Reconstructed $\hat{\mathbf{x}}$ for different d_h.

Figure 5.11 shows the reconstructed input of a sparse vanilla autoencoder, for different sizes of the latent layer. For each latent layer size d_h, the autoencoder was trained using

[3] The code used to generate this data set is provided in A.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

rectangular series. Each with 5 different amplitudes a_k — thus $K = 5$.

The first observation is that the autoencoder extracts the mean function and does not reproduce the fluctuations introduced by the gaussian nature of the signal. Instead of merely learning the identity function, the autoencoder finds the underlying factors of variation.

The quality of the reconstruction in Figure 5.11 depends heavily on the latent layer size d_h. This is not surprising, as the rectangular series is specifically designed to include five distinct amplitudes a_k. To fully reconstruct $\mathbf{x}$ at the output, the latent layer $\mathbf{h}$ needs to account for those features. Generally, we can assume that a learner that tries to pick up those factors of variations needs a latent layer size d_h, which is larger or equal to the number of factors K. The comparison for the different latent sizes d_h shows that only the autoencoder with $d_h = 5$ can correctly reconstruct the input $\mathbf{x}$. Autoencoders with $d_h < 5$ only pick up a subset of the amplitudes a_k correctly, while they fail to approximate the rest. It is also interesting that the model with $d_h = 20$ does not deliver a better reconstruction than the one with $d_h = 5$. This aspect will be covered in a moment.

Note that without regularization in place, a model with sufficient capacity could, in theory, also learn a perfect reconstruction with a smaller d_h. The autoencoder would effectively learn a highly nonlinear mapping that reflects the identity function. This encoding of $\mathbf{x}$ in $\mathbf{h}$ can be expected to be highly abstract and would not offer any insights for human analysts — compare Chapter 4.2.

Activation

While a small d_h leads to increase reconstruction loss, a latent layer size, which is higher than the number of underlying factors K, does not pose any problems to reconstruction. The respective regularization strategies ensure that the learner does only pick up the most relevant features.

In the case of a sparse vanilla autoencoder this is enforced via the l_1 activity regularization on the latent layer $\mathbf{h}$.

Figure 5.12 shows the effect of this regularization strategy in practise. The figure depicts the absolute values of each component of $\mathbf{h}$ averaged over the data set with $K = 5$. The autoencoder it this experiment — here $d_h = 30$ — does only utilize 5 of those latent neurons. The regularization penalty guides the model towards a representation that captures the right number of underlying factors. This also offers an explanation for the reconstruction of the model with $d_h = 20$ from Figure 5.11. Note, that the regularization can be controlled via hyperparameters — consider λ for the SAE and β for the VAE. This means that the number of parameters that the model picks up depends not only on the data at hand but also on the choice of the respective hyperparameter λ and β. Indirectly, the regularization parameters can control the effective size of the representation. This has to be considered, as high dimensional latent spaces are typically hard to handle in subsequent learning tasks. When the objective of the representation

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

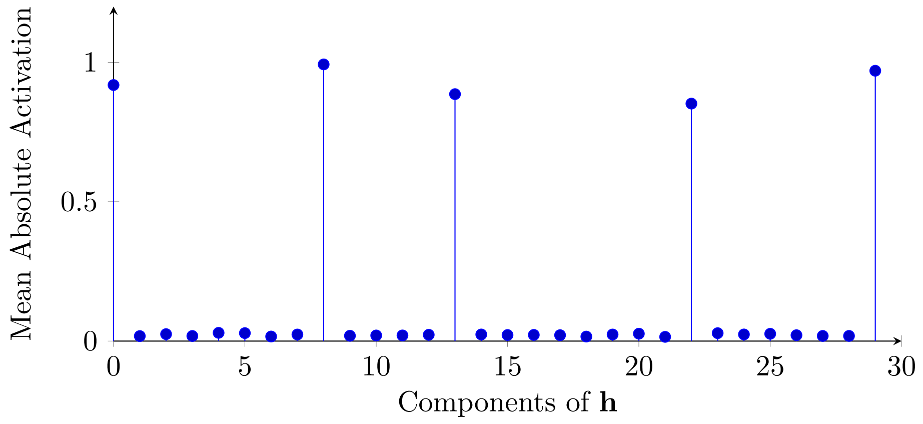


Figure 5.12: Sparse latent space activations.

learning is to analyze the obtained representations visually, this issue is, even more, pressing — Chapter 5.4 discusses this aspect in more detail.

Latent Space Representation

Naturally, the latent space is the primary interest when autoencoders are used for representation learning. The main objective in such use cases is to obtain a latent space **h**, which reflects the essential properties of the input **x**. Additionally, we require that the latent space h disentangles those key factors of variations. This means that each of the features picked up the autoencoder should be encoded along a distinct axis or direction in **h**.

For the case of the rectangular toy series, we have already identified the amplitudes a_k as the key factors of variations. In the following, I will thus analyze the latent space obtained by a SAE and VAE when trained with a rectangular time series[4]. Ideally, we will obtain a **h** where the different amplitudes a_k are directly encoded along one direction.

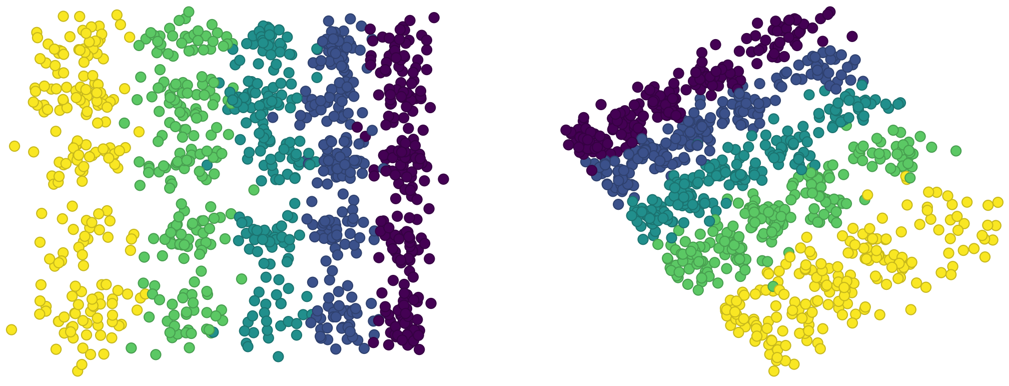
Figure 5.13 shows the latent space for two autoencoders both trained on rectangular time series with $K = 2$. Here, that mean function is given by the following equation:

$$\mathbf{m} = a_0 \cdot \sqcap(t; 35) + a_1 \cdot \sqcap(t - 35; 35) \tag{5.15}$$

The latent layer size d_h is 2 for both models. Figure 5.13a shows the results for a variational autoencoder, while Figure 5.13b depicts the representation of a sparse vanilla autoencoder. Note that, for variational autoencoders, the plots reflect the predicted means, while I omit the variances. In both plots, the color reflects the amplitude of the first rectangular component a_1. The choice of a_1 is arbitrary, plotting a_2 instead of a_1 would result in a 90 degree shifted color gradient.

[4]The code for the autoencoder used in this simulation is given in A.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub



(a) Variational autoencoder (b) Sparse autoencoder

Figure 5.13: Latent space representation of $\mathbf{x}$ with $K = 2$ with $d_h = 2$.

We can see that the amplitudes a_0 and a_1 from Equation 5.15 are directly reflected in 5.13. This means, that both autoencoders picked up the underlying factors of variation. The colors for 5.13a also indicate, that a_1 is directly reflected by one neuron in the latent layer. It is interesting to see that the sparsity constraint is directly visible in 5.13b — the shape of the latent space closely represents the surface of the l_1 norm shown in figure 5.5.

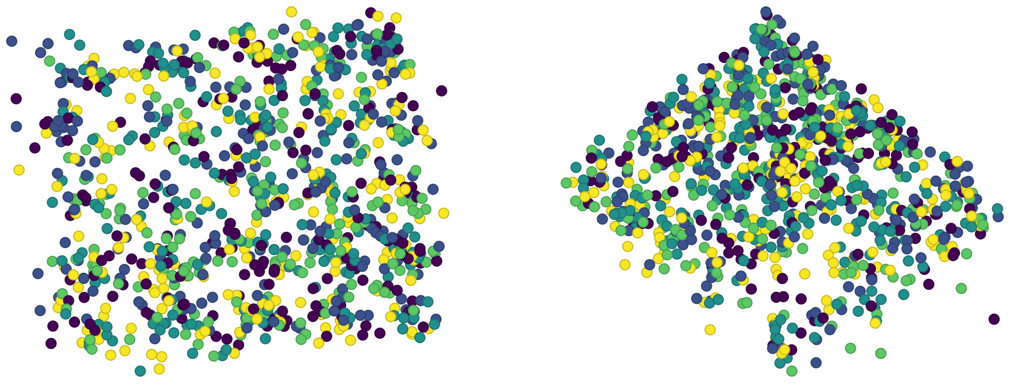
I showed in Figure 5.11 that the latent layer size d_h needs to be adapted to the underlying factors of variation K. Otherwise, the autoencoder is not able to encode all relevant properties of the input. Figure 5.14 illustrates the same observation from a latent layer perspective. Here, the same autoencoders with $d_h = 2$ were trained on a set of rectangular series with $K = 5$. The color in the scatter plots does again represent the amplitude of the first rectangular pulse a_1. Both figures 5.14a and 5.14b show, that a_1 is not directly reflected in the latent space — there is no consistent color gradient visible. This means that the autoencoders are not able to encode the five amplitudes a_k into a 2-dimensional latent space.

In general, the solution for this would be to increase the dimensionality of the latent space d_h. Often we are, however, interested in low dimensional representations in order to simplify a subsequent visual analysis. In some cases, the input data contains meta-data — by incorporating this directly into the training scheme and not forcing the autoencoder to encode it into $\mathbf{h}$ the effective dimension of the representation can be reduced.

Latent Layer of Conditional Autoencoder

Chapter 5.2.3 introduced conditional autoencoders as a way to incorporate available meta-data into the representation learning process. For this, each input sample additionally includes the meta-data vector $\mathbf{m}$. Figure 5.15 shows sparse autoencoders trained under the same setup as 5.14 — K=5 and $d_h = 2$. The difference is that here, the learners are

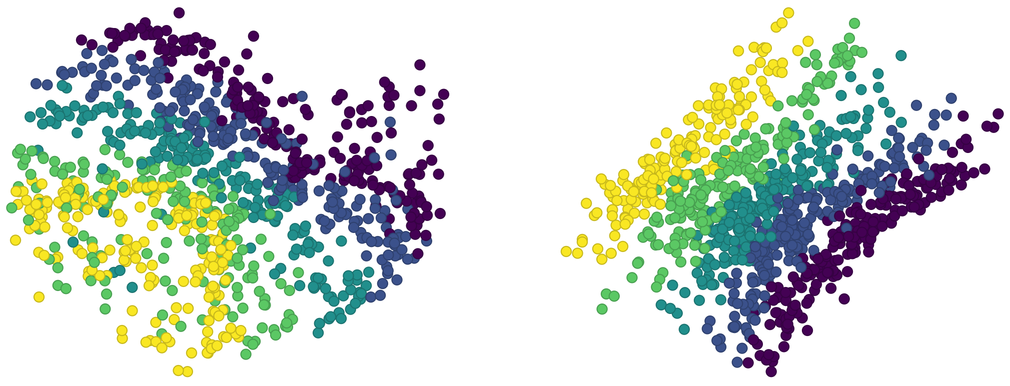
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



(a) Variational autoencoder (b) Sparse autoencoder

Figure 5.14: Latent space representation of $\mathbf{s}$ with $K = 5$ with $d_h = 2$.

based on a conditional architecture, and the remaining amplitudes a_k are encoded in the meta-data $\mathbf{m}$[5].



(a) $\mathbf{m} = [a_1, a_2]$ (b) $\mathbf{m} = [a_1, a_2, a_3]$

Figure 5.15: Sparse conditional autoencoder for $\mathbf{s}$ with $K = 5$ with $d_h = 2$.

For Figure 5.15a a_1 and a_2 are included in $\mathbf{m}$ — so only a_3, a_4, a_5 remain as degrees of freedom. In Figure 5.15b $\mathbf{m}$ encodes a_1, a_2, a_3, leaving only two factors of variation. For both figures, the color reflects a_5. The results show how conditional autoencoders can utilize the meta-data $\mathbf{m}$ in the learning process.

Figure 5.15a shows the same tendency, but here the encoder is not able to encode all three remaining amplitudes into the latent space. Still, visual analysis shows that the

[5]The code for the conditional autoencoder used in this simulation is given in A.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

color gradient does reflect the amplitude a_1 more closely than 5.14. The latent space in 5.15b is similar to the ones in 5.13b — the autoencoder does successfully capture the two remaining degrees of freedom.

In a sense, it represents a series with $K = 2$ in the latent space. The other three amplitudes a_1, a_2, a_3 are encoded in $\mathbf{m}$ anyway. Effectively, the encoder $e(\cdot)$ removes the already available meta-data from the input time series $\mathbf{s}$ and encodes the remaining variation. The decoder $d(\cdot)$ combines the information from latent space and $\mathbf{m}$ in order to generate the reconstructed signal $\hat{\mathbf{s}}$ at the output.

Outlier Detection

In Chapter 4.1.3 I introduced the basic aspects of regularization — and with it the concept of an underlying distribution p_{data}. In general, machine learning algorithms do only deliver predictable results when they receive samples from within p_{data}. The use of regularization allows for generalization, but only when the deviations from p_{data} are within boundaries.

Representation learning is no exception to this rule. Generally, it does only provide useful results when the samples do not deviate too far from the training data. This aspect is crucial and has to be considered in practice, as it limits the expressiveness of $\mathbf{h}$ for rare samples.

For this, consider the use case of outlier detection — utilizing autoencoders in this domain might be motivated by the fact that they provide a low dimensional representation of an input vector $\mathbf{x}$. Thus, $\mathbf{h}$ allows for a simplified separation of samples. Here the concept of p_{data} does, however, pose some problems.

To analyze the behavior of an autoencoder, which is exposed to an outlier, consider the following example[6]. Here I train a sparse vanilla autoencoder on a rectangular time-series data set with $K = 2$. Subsequently, I analyse the output of this autoencoder to a sample from outside of p_{data}. In this example the outlier is given by a sinusoid.

Figure 5.16 shows the input $\mathbf{x}$ and the reconstructed output $\hat{\mathbf{x}}$ for this experiment. The results might be surprising: we can see that while the input is sine signal with a constant offset, the output $\hat{\mathbf{x}}$ consists of a rectangular time-series with two distinct amplitudes. The discussion about manifolds in Chapter 4.2.2 provides an intuitive explanation for this. The encoder $\mathbf{e}(\cdot)$ maps $\mathbf{x}$ onto the manifold described by p_{data}. In a subsequent step, the decoder $\mathbf{d}(\cdot)$ provides the respective $\hat{\mathbf{x}}$ for the given manifold coordinates.

Note that the exact reason for why the autoencoder outputs precisely the series in Figure 5.16 is ambiguous — we only know that the output has to be from p_{data}. In a sense, we might find a constant signal a more appropriate output for sine with a fixed offset. The mappings obtained by $\mathbf{e}(\cdot)$ and $\mathbf{d}(\cdot)$ are, however, guided by an optimization process on the training data set and not unique. During training, we only care about a decrease

[6]This experiment is based on the SAE code in A.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

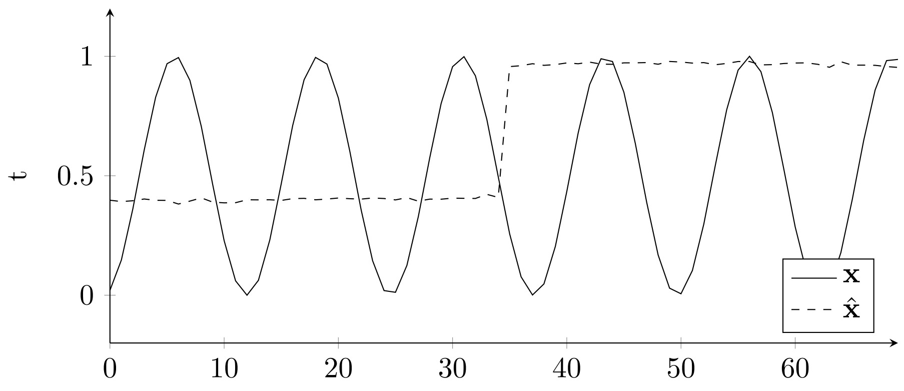


Figure 5.16: Reconstructed outlier.

in the loss function and not about the particular mapping — given that the mapping provides accurate results for samples from p_{data}.

This mapping onto p_{data} does naturally also have an effect on the representation $\mathbf{h}$ — outliers will not be apparent in the latent space. Figure 5.17 highlights this for the above example. Here, the latent space representation of the sinusoid outlier is plotted

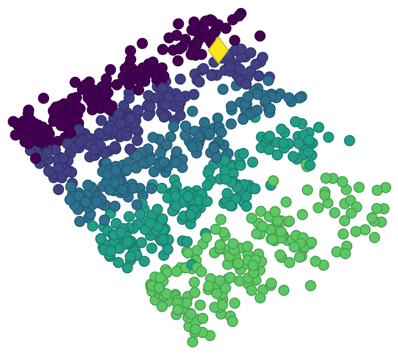


Figure 5.17: Outlier not distinguishable in $\mathbf{h}$.

in combination with a set of samples from p_{data}. It is clear from Figure 5.17, that the sinusoid can not be detected in $\mathbf{h}$ — its coordinates in the latent space describe the amplitudes of the reconstructed signal shown in 5.16.

Note, however, that outlier detection is still possible to some extend. Consider for instance a rectangular time-series with amplitudes a_k outside the range in p_{data}. An autoencoder that picked up the right features can correctly reassemble such an outlier at its output. Thus, the latent space does reflect the amplitudes a_k and subsequently exposes the outlier in $\mathbf{h}$. This means that outlier detection using autoencoders is possible, as long as the input shares the same basic structure as the samples from p_{data}.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5.3.2 QAM Representation Learning

While the above results covered the primary mechanism, the mapping from a rectangular series to the underlying amplitudes is not particularly challenging. Still, this toy input allowed for a systematic analysis of different autoencoder properties. To show that the same observations do also apply to more complex scenarios, consider the following QAM inspired input signal.

Quadratur Amplitude Modulation(QAM) is a widely adapted modulation scheme in telecommunications. In a nutshell, it provides a way to encode information into a time-series $s(t)$ to be transmitted [GG10]. Here, a bitstream $b[k]$ is mapped onto a symbol stream $a[k]$, each symbol encoding a distinct number of consecutive bits. All symbols $a[k]$ come from a complex symbol alphabet. In the case of 16-QAM the real and imaginary part of $a[k]$ can take the values of $\{-3, -1, 1, 3\}$. Thus the overall alphabet $\mathcal{A}$ is given by the 16 combinations $\mathcal{A} = \{-3 - 3j, -3 - 1j, ..., 3 + 3j\}$.

The real and imaginary part of $a[k]$ do then form the Inphase $I(t)$ an Quadratur $Q(t)$ component:

$$\begin{aligned} I(t) &= \sum_{k=1}^{K} \mathrm{Re}\{a[k]\} \cdot \sqcap(t - k \cdot T_s; T_s) \\ Q(t) &= \sum_{k=1}^{K} \mathrm{Im}\{a[k]\} \cdot \sqcap(t - k \cdot T_s; T_s) \end{aligned} \tag{5.16}$$

Here T_s is the duration of one symbol $a[k]$. Subsequently inphase and quadratur component are mixed to the carrier frequency ω_c and combined to form the final signal $s(t)$:

$$s(t) = I(t) \cdot \cos(\omega_c \cdot t) - Q(t) \cdot \sin(\omega_c \cdot t) \tag{5.17}$$

From this signal $s(t)$ the receiver can reconstruct the respective inphase and quadratur components and does consequently obtain $a[k]$.

The idea for this experiment is now as follows: I generate toy samples of $s(t)$ with the length of one symbol duration T_s. Those samples include all possible symbol values $a[k]$ and act as the input for a sparse vanilla autoencoder. Again I enlargen the data set — this time by adding zero-mean gaussian noise to the signal samples[7]. Subsequently, I analyze the latent space $\mathbf{h}$ and check whether the autoencoder picks up the 16 different symbols incorporated in the signal. Note that capturing all variation in a QAM signal would require samples of a duration larger than T_s. Still, this data set is a natural, more complex extension to the rectangular series.

Figure 5.18 shows different samples of $s(t)$ and there respective coordinates in the IQ-Plane. I use those samples — corrupted by additive white noise — as the input $\mathbf{x}$ for a sparse vanilla autoencoder[8]. Naturally, the autoencoder is only presented the time-series samples and not the IQ-representation.

[7]The code used to generate this data set is provided in A.
[8]A plot describing the models configuration is provided in A.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

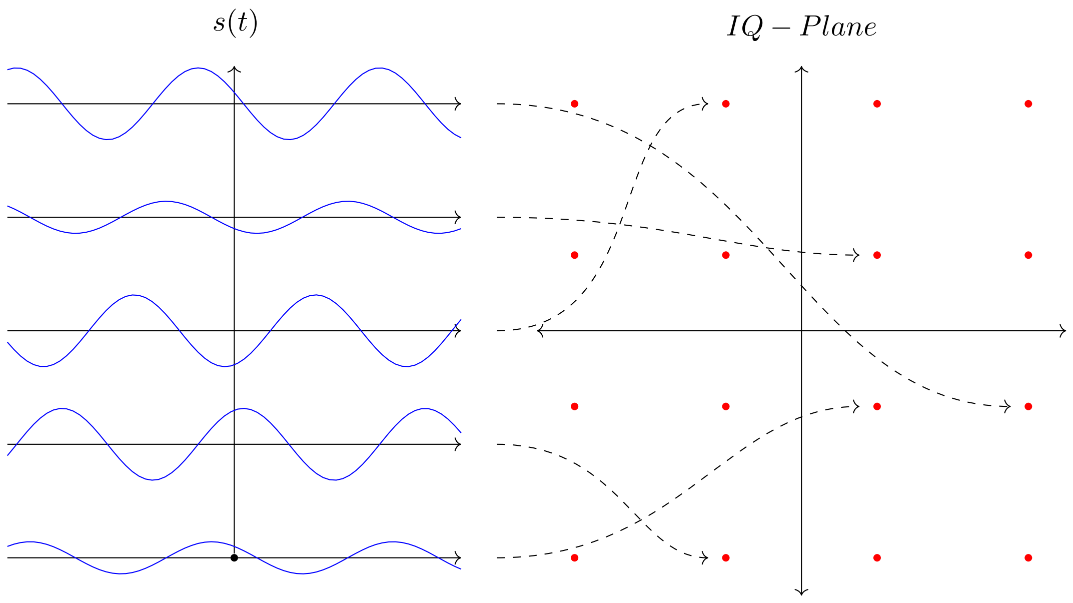
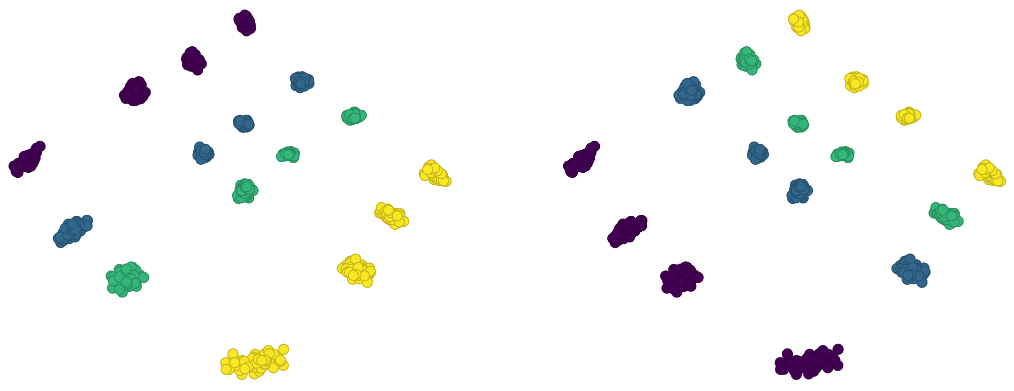


Figure 5.18: 16-QAM Constellation.

Figure 5.19 shows the representation obtained by the autoencoder. The 16 different symbols are clearly visible. It is interesting how closely the representation learned by the autoencoder mimics the commonly used IQ-Plane representation.



(a) I-Component (b) Q-Component

Figure 5.19: Learned 16-QAM representation.

The features in **h** do indeed represent the I and Q component of the signals. In Figure 5.19a the color indicates the real part of $a[k]$, while Figure 5.19b shows the same latent space but uses the color to highlight the imaginary component. We can see that besides

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

the rotation induced by the sparseness penalty, the latent space closely follows the IQ-Plane.

All in all, the simulations in this chapter show the basic mechanisms of autoencoders and how they can capture the underlying factors of variation in a given data set. The comparison of the rectangular mean series and the QAM inspired signal highlights that this extraction of features can be applied to different kinds of inputs. Interestingly, the latent space representation for the two different experiments does not differ much from one another. The latent layer does not reflect the signal's complexity but provides an abstract representation of the variation in the input data. Meanwhile, the encoder $\mathbf{e}(\cdot)$ removes the shared characteristics of p_{data}.

The results for the rectangular time-series also show how the respective regularization strategy guides the learning process. It ensures that the autoencoder picks up the most relevant underlying features and supports the disentanglement objective.

Note that for the above examples, I mostly limited the discussion to low dimensional latent spaces. However, for some problems, it might not be possible to encode all variation into a small d_h. Moreover, real-world input data is usually of a more complex nature. This means that also the representation space is, in general, not of such a structure as the toy examples above. An example of such a problem has already been discussed in the form of the MNIST representation in Figure 5.2a. Dealing with such complex and potentially high dimensional latent spaces is not straightforward. Thus, Section 5.4 will briefly mention some techniques for latent space analysis before Chapter 6 will finally apply autoencoders to network benchmarks.

5.4 Latent-Space Analysis Techniques

As already mentioned, most representation learning tasks are followed by the application of some additional learning algorithm. There are, however, use cases, where the visual exploration of the data set via the obtained representation $\mathbf{h}$ is the core objective. Different techniques for the analysis of the latent space $\mathbf{h}$ can be helpful in this context.

In general, we are often interested in finding natural clusters in $\mathbf{h}$ — consider for instance the 16 distinct clusters in Figure 5.19. Also, the structure of the latent space itself can provide new insights. For that purpose, clustering and kernel density estimation methods can be helpful — they drastically simplify the handling of complex $\mathbf{h}$.

For a latent size $d_h > 2$, the visualization of the latent space might be challenging. In this case, dimensionality reduction approaches designed explicitly for visualization can be used. In the following, I will briefly discuss approaches common in this field. Explaining them in detail is out of the scope of this work, but I will provide sources for further reading. This means that the following remarks are not an exhaustive introduction into the field, but act mainly as a motivation for why such approaches can be beneficial.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5.4.1 Kernel Density Estimation

Kernel Density Estimation (KDE) provides a way to obtain a probability density function from a given histogram of measurements. At its core, KDE is a nonparametric scheme operating on a set of samples $\{x\}_{i=1}^{N}$. The approximated density function is then obtained via Equation 5.18.

$$p_K(x) = \sum_{i=1}^{N} K\left(x - x_i; b\right) \tag{5.18}$$

Here b is a hyperparameter referred to as the bandwidth. In a nutshell, Equation 5.18 describes the superposition of individual pulses K with bandwidth b. Here, the number of pulses N is the same as the number of inputs x_i. This means that each x_i acts as the center for one pulse. K is referred to as the kernel — commonly K is based on a Gaussian.

$$K(x; b) \propto \exp\left(-\frac{x^2}{2b^2}\right) \tag{5.19}$$

I will not cover the details of hyperparameter selection for KDE — most schemes are based on cross-validation or Silvermans Rule of Thumb. A exhaustive introduction of KDE can be found in [Sco15]. Typical KDE implementations also utilize a scheme where neighboring pulses are merged based on some overall distortion measure. This means that in the end, $p_K(x)$ reflects a Gaussian mixture with distinct means and variances. Such schemes are closely related to clustering.

The use of KDE in the realm of autoencoders does mainly serve a practical purpose. When dealing with real-world data sets, the latent space structure might be challenging to handle. KDE can be used to obtain an analytical description of the latent space $\mathbf{h}$. More specifically, we can obtain a distribution $p_K(\mathbf{h})$ describing the arrangement of the latent space coordinates of all the input $\mathbf{x}$. Note that this is not the same as VAEs, which provide a statistical description for each individual sample.

Operating on $p_K(\mathbf{h})$ instead of directly using the individual samples $\mathbf{h}_i$ simplifies the handling of complex latent spaces. This is especially important for representations with $d_h > 2$. For instance, we can obtain a likelihood measure for a specific $\mathbf{h}$ to quantify how common samples get mapped to this part of the latent space. Additionally, $p_K(\mathbf{h})$ can be used to sample from the latent space systematically. Finally, I will also utilize KDE when plotting latent spaces in Chapter 6.

5.4.2 Clustering

Often the primary motivation for representation learning is to obtain natural clusters that are not directly visible in $\mathbf{x}$. Mapping the input $\mathbf{x}$ to $\mathbf{h}$ ideally exposes those clusters. An example of this is, for instance, given in the QAM mapping obtained by the SAE in Figure 5.19.

Clustering is a field of machine learning which deals with finding such underlying clusters in data. One of the most common clustering algorithms is Gaussian Mixture Clustering

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

(GMC), which can be seen as an extension of the well known k-Means [Bis06, Chapter 9]. In a nutshell, GMC models the input data as a superposition of multivariate gaussians.

$$p(\mathbf{x}) = \sum_{k=1}^{K} a_k \, \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, C_k) \tag{5.20}$$

Note that while this might look similar to KDE, the two approaches differ in crucial aspects. Here, the number of clusters K is a hyperparameter and has to be determined beforehand while it is given by the cardinality of the input set N for KDE. Additionally, each of the Gaussian components is weighted by a_k. Also note, that the components $\boldsymbol{\mu}_k$ and C_k are part of the optimization process and not predetermined by the samples x_i or set as a hyperparamter. The optimization of GMC itself is out of the scope of this work — details about the underlying EM-algorithm can be found in [Bis06, Chapter 9].

In the realm of representation learning, clustering techniques can again simplify the analysis of complex latent spaces. GMC does effectively provide a similar description of **h**, which is easier to handle than the raw samples. Note that as opposed to KDE, this description does reflect the underlying clusters and can be seen as a first step of a subsequent learning task.

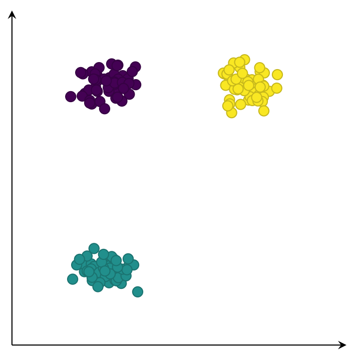
5.4.3 Visualization of High Dimensional Data

In general, the representation **h** obtained by autoencoders is of lower dimensionality than the original input **x**. While autoencoders have been shown to provide a robust framework to extract the underlying features of variation, the results from Section 5.3 indicate, that it is not always possible to obtain a meaningful representation with an arbitrary small d_h. Whenever $d_h > 2$, this poses a problem for the visualization of such latent spaces.

In such cases, it can be helpful to resort to dimensionality reduction techniques, specially designed for visualization. Note that such approaches should not be confused with representation learning techniques. While representation learning achieves a dimensionality reduction as a side-effect of the extraction of underlying features, the objective is different for such high-dimensional visualization techniques. They generally aim for a mapping towards a lower-dimensional space, which preserves most of the proximity information. This means that the main objective is that points close in the high-dimensional space are also close in the low-dimensional space.

A prominent example of such approaches is t-SNE introduced in [MH08]. At its core, this scheme is based on a stochastic similarity measure between different points. More precisely, it assigns each pair of points a similarity score, which is modeled as a conditional distribution based on a gaussian pdf. In the low dimensional space, also a similarity matrix is calculated, which is, however, based on a Student-t distribution. These similarity scores are then collected in two distinct similarity matrices. During the optimization process, the points in the lower dimensional space get rearranged until the two similarity matrices are close to each other. Here, the KLD acts as a measure of closeness between the similarity in the high and low dimensional spaces.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN


(a) 2D

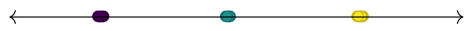

(b) 1D

Figure 5.20: t-SNE example for 2D Gaussian Mixture input.

Again, I will not cover the details of t-SNE in this work — all of the crucial aspects are covered in [MH08]. Instead, I provide a basic example given in Figure 5.20 to provide some intuition for the scheme. Figure 5.20a shows an exemplary 2D toy samples drawn from a multivariate gaussian with three distinct clusters. In 5.20b the 1D t-SNE visualization of the same data set is given. Note that the three distinct clusters are still visible after the dimensionality reduction. This shows that t-SNE can preserve the proximity information between the individual samples. Still, the results from t-SNE should be taken with a grain of salt. In more complex scenarios, the results may vary significantly for different choices of hyperparameters. Because of this, t-SNE is generally considered a tool for data exploration and visualization and should not be mixed up with dimensionality reduction techniques such as PCA.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

RTR Netztest Analysis

Chapter 1 identified the *limited availability of parameters* as well as the tedious process of *collecting labeled samples* as the key challenges in the realm of context inference. Such context information is, however, crucial to obtain **fair** network benchmarks from crowdsourced measurements. In Chapter 3, I introduced a semi-supervised approach that allows for classification of measurements as limited or unlimited. This approach offers a way to deal with the limited number of labeled samples.

data set	**Operator**	**Time-Range**	**Throughput**	**LTE-RSRP**
$D_{A,B,C}$	MNO-{A,B,C}	*16.11.19 - 13.02.20*	79 Mbit/s	-97 dBm
C_A	MNO-A	*28.12.19 - 30.03.20*	84 Mbit/s	-96 dBm
C_B	MNO-B	*24.11.19 - 29.03.20*	69 Mbit/s	-97 dBm
C_C	MNO-C	*30.11.19 - 30.03.20*	45 Mbit/s	-102 dBm

Table 6.1: Crowdsourced data sets.

The following chapter tackles the issue more comprehensively — by applying autoencoders to the crowdsourced data, the measurements are processed in a completely unsupervised way. The obtained representations allow for large scale analysis for the time-series data and can subsequently act as a basis for further inference tasks. This way, the field can benefit from the advances in deep learning without having to provide a large number of labeled data. In a sense, this can also be seen as an automated version of the manual feature engineering conducted in Chapter 3. We have seen that the mapping to the feature space allowed for robust classification of limited and unlimited measurements.

Motivated by that, I examine the representations obtained by a SAE in an unconditional and conditional setup and analyze whether those representations expose the context of measurements. Table 6.1 shows the crowdsourced data sets used for this analysis. Each

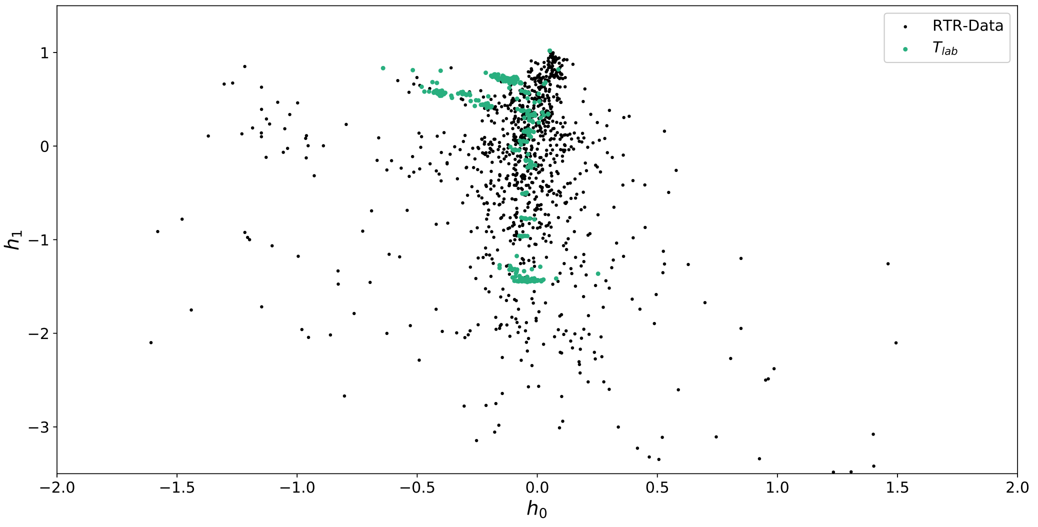


Figure 6.1: Latent Space for RTR Data.

of the sets consists of approximately 20.000 samples from RTR-Netztest. Thereby the first set $D_{A,B,C}$ is a collection of measurements that were conducted in the interval of 16.11.19 to 13.02.20. Meanwhile, the other sets C_A, C_B, C_C also contain measurements conducted in March 2020. Thus, the C data sets' timeframe coincides with the first two weeks of the COVID-19 related lockdown in Austria. Motivated by this, I will examine this period in more detail and look for signs of disruptions caused by the subsequent shift in user behavior.

For both experiments– unconditional and conditional– $D_{A,B,C}$ acts as the autoencoder training data set. Subsequently, I analyze the latent space representation for the remaining data, using the already trained model. Besides the crowdsourced data sets in 6.1, this also includes the manually collected measurements T_{lab}, V_I and V_{CL} from Chapter 3. Note that for both experiments, the latent layer size d_h is set to 2 to allow for a visual analysis of the results.

6.1 Unconditional Analysis

For the first set of experiments, I use a SAE in an unconditional setup. A detailed model description can be found in Appendix A. As already stated, $D_{A,B,C}$ acts as a training data set, while the remaining data sets are open for analysis.

Figure 6.1 provides a first overview of the obtained representations. Here, all samples from $D_{A,B,C}$ are plotted in black, whereas the latent space representation of T_{lab} is shown in green. Compared with the representation for the toy example in 5.13, the latent space $\mathbf{h}$ is of a more ambiguous structure here. Especially the samples from $D_{A,B,C}$ do not expose a distinct structure in $\mathbf{h}$. The data from T_{lab}, collected in a lab environment, does,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

however, follow a clear pattern. Figure 6.2 further highlights this structure in the sample distribution of the T_{lab} representations.

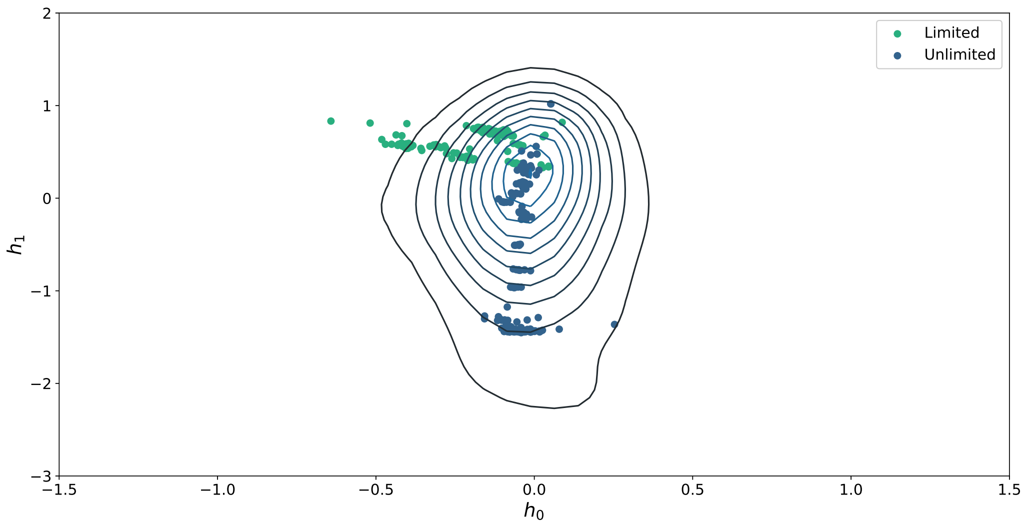


Figure 6.2: Limited and Unlimited T_{lab} within $D_{A,B,C}$.

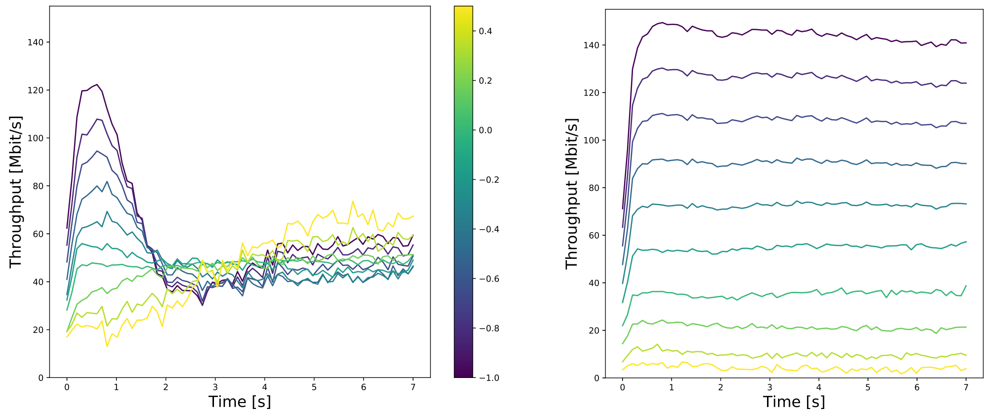
Here, limited and unlimited samples from T_{lab} are shown in different colors. As a reference $D_{A,B,C}$ is shown in the form of a density map obtained via KDE — compare 5.4.1. Interestingly the limited and unlimited samples from T_{lab} are naturally seperated in Figure 6.2. The autoencoder obtained a 2-dimensional representation that would allow for classification of T_{lab}. This is interesting, as I did not incorporate any specific scheme — like semi-supervised autoencoders — to guide the learner towards exposing tariff limitations. Also, the autoencoder's training data was given by $D_{A,B,C}$, and did not include T_{lab} itself. Moreover, it seems as if tariff limits are one of the major features the SAE obtained from the crowdsourced data of $D_{A,B,C}$.

Furthermore, we can notice that the limited and unlimited measurements are clustered along two distinct axes in 6.2. More precisely, the limited samples are spread across different values of h_0, while h_1 is more or less fixed. The opposite is true for the unlimited samples, which concentrate along the axis of h_1 with a fixed value of $h_0 \approx 0$.

Figure 6.3 allows for interpretation of those results. Here, I sampled the latent space along h_0 and h_1, in order to gain a deeper understanding of the encoded features. More precisely, I select distinct regions from the latent space and obtain the respective time-series for that particular choice of h_0 and h_1.

Hereby, Figure 6.3a shows the variation along h_0, with h_1 fixed at 0. The peak in throughput, already discussed in Figure 2.1 is clearly visible. Based on those results, we can interpret h_0 as an indicator for tariff limits — where the precise value of h_0 encodes the intensity of the peak. More specifically, this means that a low value of h_0 indicates a

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.



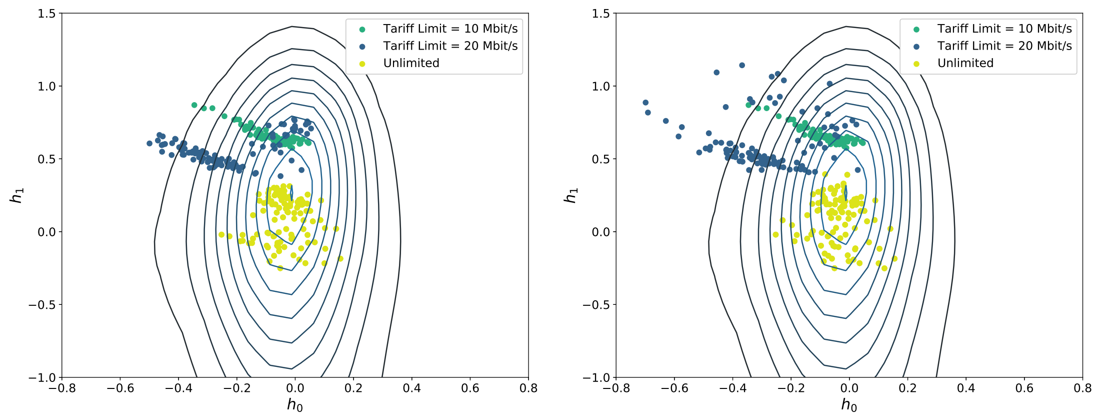
(a) Variation in h_0 for $h_1 = 0$.

(b) Variation in h_1 for $h_0 = 0$.

Figure 6.3: Variation along h_0 and h_1.

measurement environment where a high theoretical rate could have been achieved before the throttling activated.

The sampling along h_1 with $h_0 = 0$ in Figure 6.3b results in nearly constant throughput time-series with different means. This indicates that h_1 represents the throughput of the encoded measurement. Hereby, a smaller value of h_1 relates to a higher throughput time-series. Note that while the feature h_1 does mainly encode the series's mean, the overall encoding process itself is more complex. Consider, for instance, the ramp-up phase at the beginning of each series in 6.3b.



(a) High Interference data set - V_I.

(b) High Cell Load data set - V_{CL}.

Figure 6.4: Validation data sets in $\mathbf{h}$.

All in all, we can interpret the space in the following way: The value of h_1 is encoding the amplitude of the token-bucket induced peak, which correlates with the theoretically achievable rate without tariff shaping. At the same time, h_0 represents the overall offset

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

or, respectively, the time series's tail. For $h = 0$, the height of the peak and the tail coincide, which indicates an unlimited test.

As a consequence of this encoding, tariff limits are visible in the latent space. For that, consider the results shown in Figure 6.4. Here the representations for the validation data sets under high cell load and interference introduced in 3 are shown. We can see that the tests with limits of 10 and 20 Mbit/s form two distinct clusters, while the unlimited tests get mapped to the center. Again, it is interesting, that the autoencoder separates those natural clusters almost perfectly, without any guidance towards detecting tariff limitations.

Operator Comparison

Now that the interpretation of the latent space variables h_0 and h_1 is clear, the obtained representations can be used to compare the three different MNOs from Table 6.1.

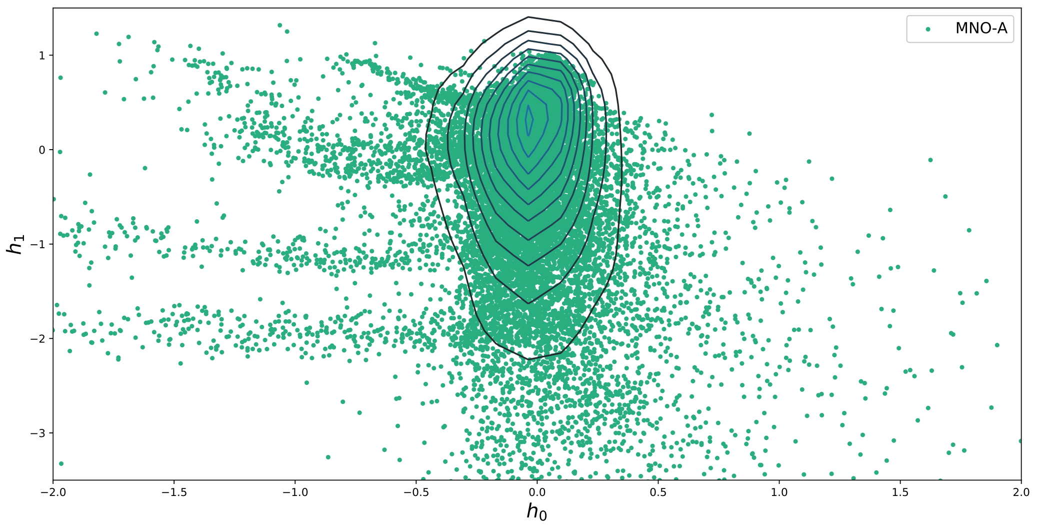
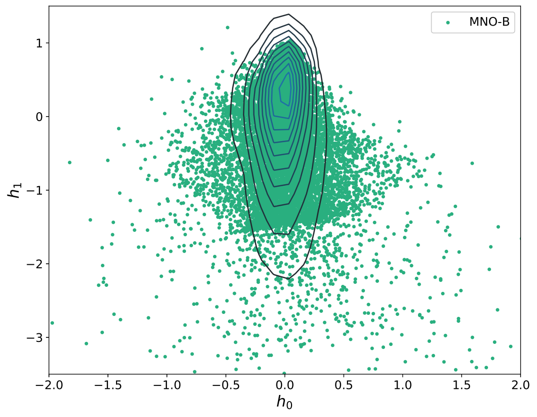


Figure 6.5: C_A latent space representation.

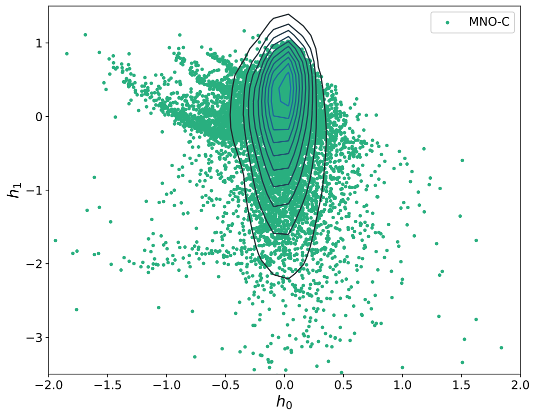
Figure 6.5 shows the obtained representation for the data set C_A. Again, I also show a KDE plot of the representation of $D_{A,B,C}$ for reference. On the left of 6.5, three to four horizontal clusters stand out. Figure 6.2 did already show, that horizontal clusters can be associated with tariff-limited tests. Analysis of those clusters — via sampling of the latent space — supports this interpretation. In fact, the obtained samples expose the respective tariff-limits for each of the clusters — 300, 150, 100, and 10 Mbit/s. Interestingly the first three limits are consistent with the tariffs that MNO-A is currently offering on its website.

The Figures 6.6 show the results for the other two MNOs. In 6.6b the results are similar to the ones in 6.5 — there is again a clear tariff limitation visible. However, the majority of limited tests seem to be throttled to a lower rate than for MNO-A. This suggests that

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) MNO-B



(b) MNO-C

Figure 6.6: MNO latent space representation.

MNO-C is selling fewer high rate tariffs than MNO-A. In theory, it could also mean that MNO-C customers hardly ever reach the throughput needed for the high rate tariff-limits to activate. This observation does also correspond to the metrics in 6.1 — where MNO-C lags far behind in throughput.

Interestingly, there are no clear tariff limits visible in Figure 6.6a. While the latent space suggests that peaks induced by tariff limits are present, they are not organized in distinct clusters. This is consistent with the results from Table 3.3, which also suggests that MNO-B has the smallest ratio of limited tests. The reason behind this is not entirely apparent. An explanation would be that MNO-B utilizes a different tariff limitation scheme than the other two operators.

Lockdown data set

The process of writing this thesis coincided with the COVID-19 induced lockdown in Austria. This motivates the analysis of the effects the lockdown had on the cellular networks. For that, I filter the data sets C_A, C_B, C_C for samples collected after 16.03.2020, while the training data does still consist of all of $D_{A,B,C}$.

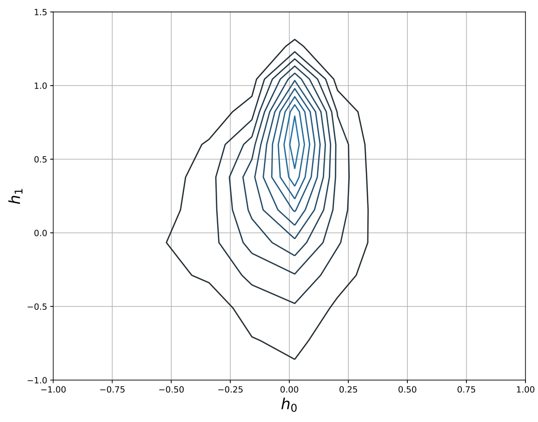
MNO	**Throughput**	**Change** [%]	**RSRP**	**Change** [dBm]
MNO-A	65 Mbit/s	-23%	-98 dBm	-2 dBm
MNO-B	34 Mbit/s	-51%	-101 dBm	-4 dBm
MNO-C	33 Mbit/s	-27%	-101 dBm	-1 dBm

Table 6.2: Lockdown data sets.

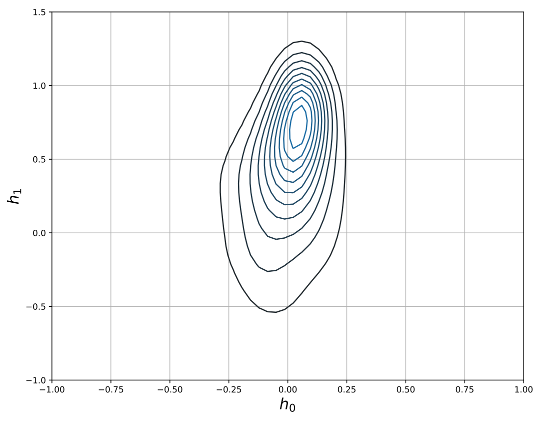
When looking at the metrics in Figure 6.2, we see that there has been an apparent reduction in throughput. However, this decrease in throughput can not be explained

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

solely by fluctuations in RSRP– note further that those fluctuations are most likely caused by the small sample size of two weeks. Analyzing the throughput time-series can provide further insights into the reasons behind this drop in throughput. Again we have to resort to the latent space representation of the time-series to pursue such an analysis on a large scale.

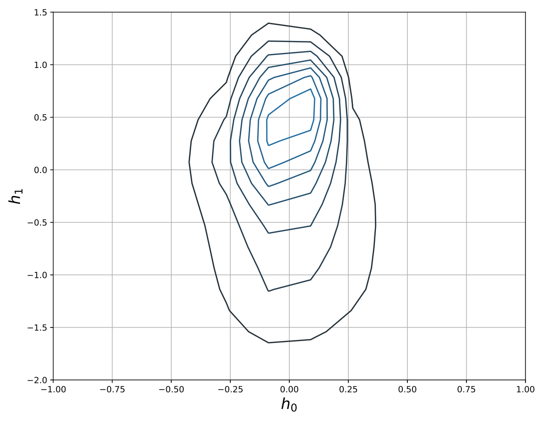


(a) All of C_C.

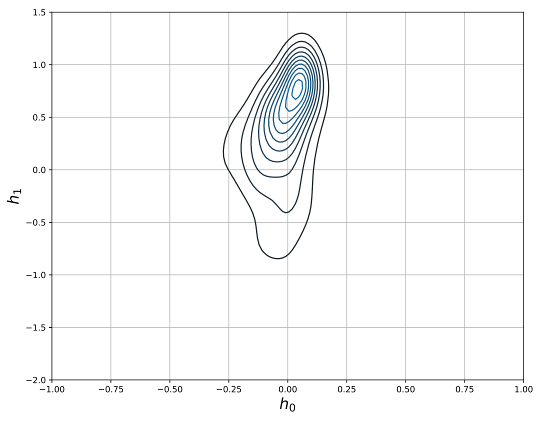


(b) Lockdown C_C.

Figure 6.7: Lockdown analysis MNO-C.



(a) All of C_B.

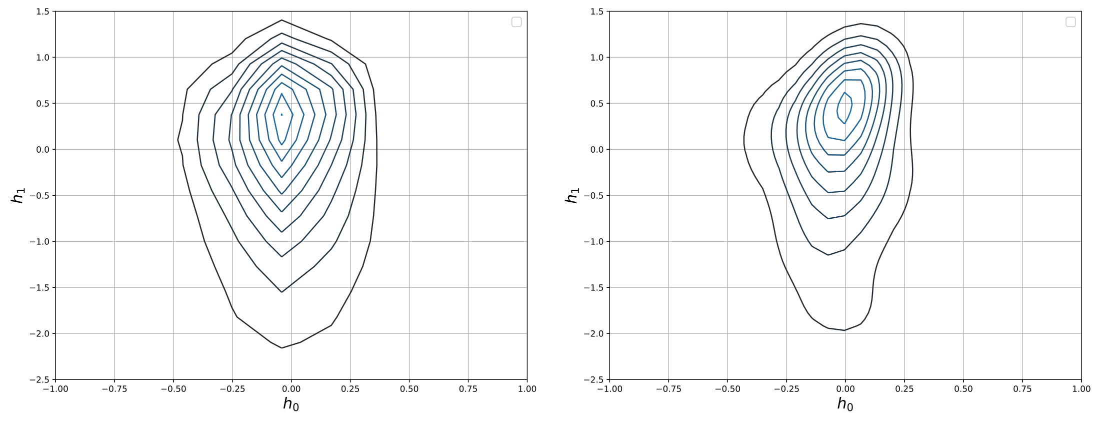


(b) Lockdown C_B.

Figure 6.8: Lockdown analysis MNO-B.

Figures 6.7, 6.8, 6.9 show a comparison of the representation from before and during the lockdown. While the results differ for each MNO, all lockdown data sets tend to have a reduced width in h_0 as compared to the pre-lockdown samples. This effect is quite prominent in Figure 6.7, where we can see a reduction of samples in the area representing tariff-limitations. At the same time, we can observe a reduction of high throughput measurements. Those findings suggest that there could have been increased cell-load during the lockdown — even though the RSRP is similar to pre-lockdown measurements,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



(a) All of C_A.　　(b) Lockdown C_B.

Figure 6.9: Lockdown analysis MNO-A.

users do not reach the same throughput levels. This increase in cell-load might also explain the lack of tariff-limited tests — users are simply not reaching the throughput necessary for the tariff shaping to activate.

While the latent space analysis offers some clues regarding the performance during the lockdown — we know that tariff-limits are not to blame — we still can not say, whether the measurements during lockdown are representative. By that I mean, that the lockdown caused a significant shift in user behavior. As an effect of this, it is hard to say whether the data set describes the network performance during a lockdown or is merely a consequence of a shift in the crowd's behavior concerning throughput measurements.

6.2 Conditional Analysis

Conditional autoencoders were introduced in Chapter 5.2.3. Remember, that they provide a way to encode metadata into a separate feature vector $\mathbf{m}$. This feature vector $\mathbf{m}$ acts as an additional input for the encoder $\mathbf{e}(\cdot)$ and decoder $\mathbf{d}(\cdot)$. As an effect of this the latent space $\mathbf{h}$ of a conditional autoencoder is parametrized on a distinct choice of $\mathbf{m}$.

In addition to the throughput time-series, RTR-Netztest offers several meta parameters. Those parameters were already discussed in Chapter 2.3. Encoding those features into $\mathbf{m}$ allows for comparison of different MNOs for a distinct choice of $\mathbf{m}$. An example of this would be the RSRP — Table 6.1 showed that it is varying slightly between different operators. Conditioning on the RSRP can highlight other performance aspects in cellular networks — cell load effects might, for instance, be more evident in such a setup.

For that, I discretize the RSRP into ten distinct intervals with uniform width and encode it directly into $\mathbf{m}$. In the following, I denote this intervals by $Level_{RSRP} = i$ where i is from $\{0, \dots, 9\}$. Without the discretization step, there would not be a sufficient number of samples for a given choice of RSRP.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Analysis of T_{lab}

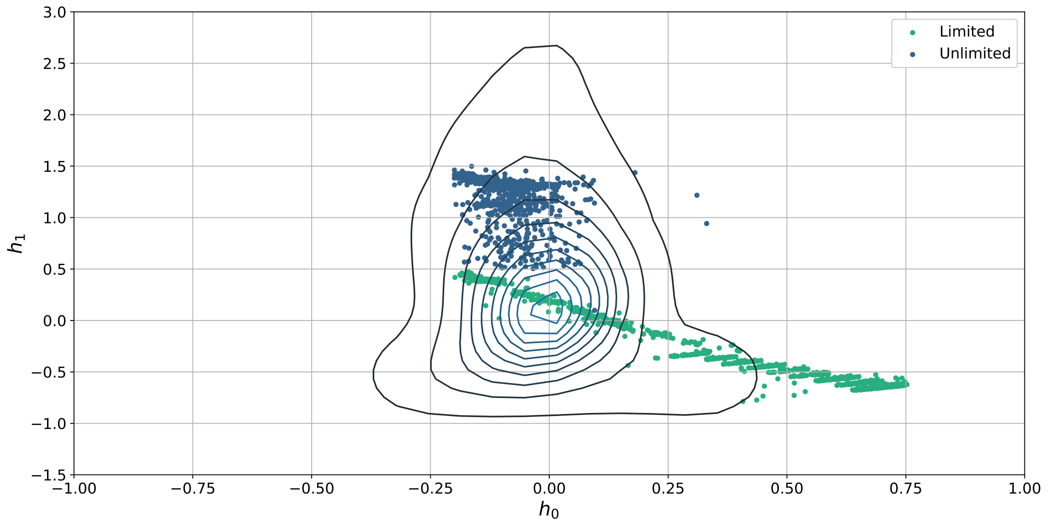


Figure 6.10: T_{lab} conditioned on RSRP ($Level_{RSRP} = 0, \ldots, 9$).

Figure 6.10 shows the latent space for such a conditional autoencoder. Here, I use a SAE — the precise model can be found in Appendix A. In Figure 6.10 the training data set $D_{A,B,C}$ is shown in form of a KDE, while the scatter plot shows the limited/unlimited samples of T_{lab}. Note also, that in this depiction all 10 $Level_{RSRP}$ are shown at the same time. Sampling of the latent space reveals, that h_0 does again encode the throughput, while h_1 reflects the peak induced by tariff shaping. Note, that this is similar to the results from Figure 6.3.

Still, there is a significant difference in the representation of the conditional autoencoder in 6.10 and its unconditional counterpart in Figure 6.2. Most notably, in 6.10, the unlimited samples are clustered along a small intervall of h_1 — besides samples for low RSRP. This is not the case for the unconditional autoencoder in 6.2, where the unlimited tests cover a wide range of h_1. The reason for this is that samples in 6.10 are conditioned on the RSRP. This means that h_1 does not directly encode throughput, but deviations from the expected throughput over RSRP.

Note that here, the training data set $D_{A,B,C}$ acts as the reference. As T_{lab} was collected under lab condition, the throughput over RSRP is not affected by cell load and interference — the data set is thus mapped to a distinct closed region. When only considering $Level_{RSRP} > 0$, the samples are even more densely packed.

Interpretation of the latent space

Figure 6.11 sums up the interpretation of the latent space for the trained conditional autoencoder. We can think of **h** to consist of 4 distinct regions, which describe different classes of tests.

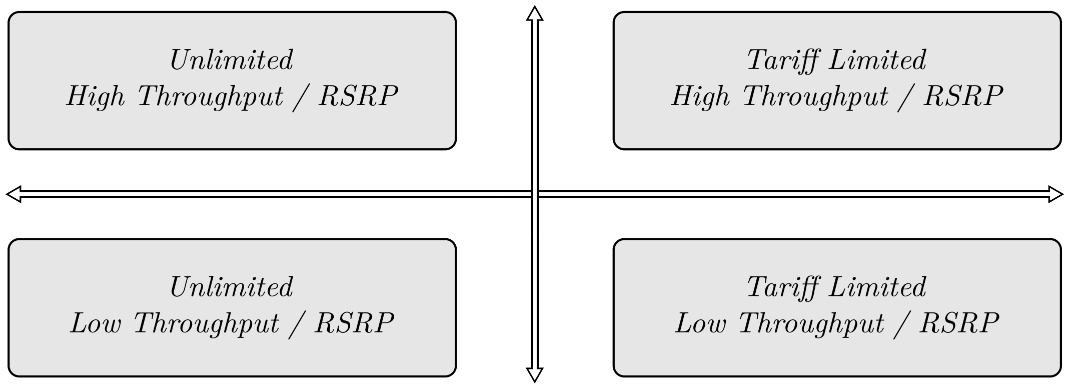


Figure 6.11: Interpretation of Latent Space.

The upper left corner of the latent space does, for instance, encode unlimited test with an above-average throughput over RSRP ratio. Unlimited samples from T_{lab} are also mapped to this part of **h** —the lack of cell load and interference renders those tests above average in terms of throughput.

Generally, test mapped to the lower half of **h** tend to have relatively low throughput for the given $Level_{RSRP}$. The distinct value of h_0 again splits this area into unlimited and limited tests. For the limited tests, the tariff limit is naturally the reason for the below-average throughput over RSRP. Meanwhile, the unlimited measurements in the lower-left corner include tests from high cell load or high interference scenarios. This area might also include measurements from limited sim-cards, which do not reach the throughput necessary for the tariff shaping to activate.

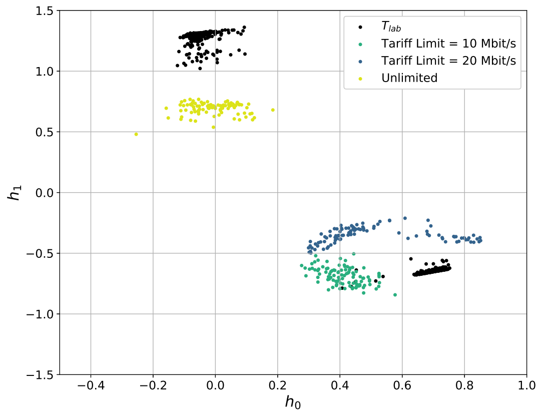
Finally, the right upper represents high throughput tests for a given RSRP region, which are limited at the same time. Such tests — high tariff limit and low cell load and interference — tend to be rare. This lack of tests can also be seen in the KDE plot in Figure 6.10.

All of this means that the conditional autoencoder offers a distinct area where cell-load and interference effects can be detected. In such a representation we can distinguish a network operator suffering from high cell load or interference from one which sells low throughput contracts. While the latter follows a legitimate business strategy, the first MNOs network might require further investment.

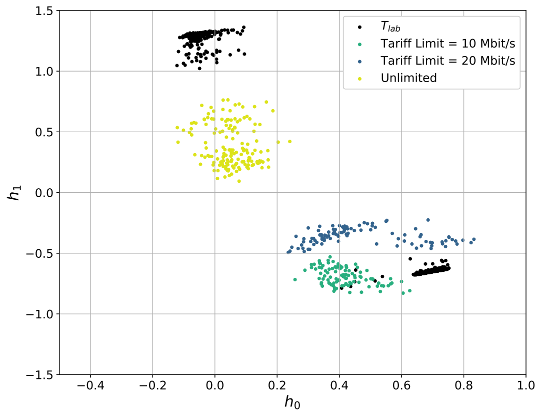
Validation data set analysis

Figures 6.12 further supports this interpretation of the latent space. Here, the representations for the validation data sets V_I (6.12a) and V_{CL} (6.12b) are shown. The plots can be seen as the conditional counterpart to 6.4. Due to the high RSRP environment in both data sets, the plot is only showing the region $Level_{RSRP} = 9$. As a reference, the limited & unlimited representations of T_{lab} for $Level_{RSRP} = 9$ are also depicted.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) High Interference data set V_I.



(b) High Cell-Load data set V_{CL}.

Figure 6.12: Latent Space Conditioned $Level_{RSRP} = 9$.

As expected, tariff-limited tests are mapped to the lower-right region for both plots. Furthermore, the different tariff-limits of 10 & 20 Mbit/s are separated by different values of h_1. At the same time, the 10 Mbit/s limited tests of validation data and T_{lab} do share the same h_1 value. On the other hand, the unlimited tests get mapped to the upper-left part of **h**. This suggests that the throughput over RSRP is still above-average for V_{CL} and V_I. However, both unlimited clusters are well below the T_{lab} samples, which were collected in a lab environment without cell-load or interference effects. It is also interesting to see that the unlimited test in 6.12b are spread across a larger interval of h_1 than the ones in 6.12a. This effect is most likely caused by variations in cell-load throughout the measurement process. Note that this behavior was not directly visible in the unconditional counterpart in Figure 6.4b.

6.3 Conclusion

All in all, the results above show that autoencoders can indeed be used for large scale analysis of high-dimensional data. Without any guidance towards exposing tariff-limitations, the learner obtained a representation that naturally separates limited and unlimited tests. Additionally, it maps each tariff-limit to a distinct cluster, which uncovers the tariff structure of different MNOs. The results from the extension to conditional autoencoders suggest that the removal of the RSRP influence leads to a representation that further highlights cell-load and interference effects.

All of this is achieved under a relatively generic setup that does not require extensive fine-tuning to adapt to the data at hand. Exposing the QAM-Constellation in 5.3.2 and learning representations from RTR-Netztest is, for instance, achieved with the same autoencoder configuration[1].

[1]The model configuration is provided in A

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

While the results for the lab and validation data sets are promising, the representations obtained for the crowdsourced data are more ambiguous. This shows that the representation learning process is still highly sensitive to the quality of the input data. The operator comparison did offer some insights into the distribution of the throughput time-series and exposed the underlying tariff-structure of the MNOs. Still, I did not obtain natural clusters for different classes of throughput time-series — in fact, most of the MNOs had a generic distribution in $\mathbf{h}$. This shows that while RTR-Netztest offers a vast number of samples, it still lacks a sufficient number of diverse users to really capture all environmental conditions. Especially when limiting the evaluation of the data to a smaller region and time-interval, we can see, that a majority of samples is most likely coming from a small number of devices.

Still, I find the application of representation learning to such network benchmarking results promising. We have seen that the representations clearly separate limited & unlimited tests. This could be extended to other use-cases. Higher-dimensional representations of the measurement results could, for instance, act as the basis for other context inference tasks. Among those are, for instance, indoor/outdoor detection or the inference of cell-load effects. Also semi-supervised architectures seem promising in this domain — provided that a sufficient number of labeled samples is available.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 7

Conclusion

Throughout this work, I used machine learning to address one of the primary challenges in the field of crowdsourced network benchmarks — the missing context of user measurements. The key objective was thereby to extract information inherent to the throughput time-series in order to expose the environmental conditions under which measurements were conducted. In particular, I focused on detecting tariff-limitations, which has been identified as a critical requirement for **fair** operator benchmarking. For all of this work, the extensive set of several thousand ground truth measurements I collected proved to be essential for training and validating the used algorithms.

In summary, I tackled the two primary challenges of context inference — namely the *limited availability of parameters* and the tedious process of collecting *labeled measurements*– in two ways:

The first step, as discussed in Chapter 3, involved introducing a semi-supervised method for classifying limited and unlimited measurements. This addresses the question put forward in Chapter 1 — regarding the extent to which we can ***infer the network view from the user view while dealing with the limited availability of parameters***. As such, I conducted manual feature engineering to extract the critical features from an RTR-Netztest measurement. Transforming the collected raw measurements into the feature space shows, that the carefully selected features separate limited and unlimited tests almost without error. Subsequently, I trained a classifier based on Label-Spreading that operates on the generated feature vector. The semi-supervised nature of Label-Spreading allows unlabeled data to be included in the process, which is a step to ensure effective generalization from the training samples. Benchmarking based on an outdoor validation data set confirmed that the unlabeled data does indeed improve accuracy. Overall, the classifier achieves an accuracy of 99% when benchmarked on the unseen 20Mbit/s data set. Subsequently, I applied this classifier to a crowdsourced data set from RTR-Netztest to obtain an operator benchmark from the network view. Although the ranking itself did not change in this case, the results differed significantly from the user view. Note that the outdoor ground truth data set proved to be essential to guarantee dependable validation of the overall approach — a step that is often missing in approaches in the literature. In general, the manual data collection I conducted is crucial for training and validating the algorithm — having access to a reference LTE eNodeB proved to be fundamental to achieve the reliable classification results.

Although the classifier in Chapter 3 offers robust and accurate results, the manual feature engineering conducted is specific to the use case and requires domain knowledge. As such, the second part of this work evaluated the application of representation learning in this field. As a mainly unsupervised approach, it tackles the issue of limited labeled examples comprehensively. As such, the benchmark analysis can benefit from deep learning techniques without having to provide a large amount of labeled data. This addresses the second question in this work — namely, ***in what way deep learning can aid this process and what are the challenges when applying it to crowdsourced data sets?***. For that, I applied representation learning with autoencoders to the crowdsourced data sets. The latent space analysis showed that the obtained 2D representation encoded the overall throughput of the time-series as well as the peak induced by the tariff-limits. As an effect of this, limited and unlimited tests were each mapped to distinct clusters. This is remarkable, as I did not deploy any specific scheme to guide the learner towards detecting tariff limits. When comparing the latent space representation of measurements from different MNOs, the distinct tariff limit structures of the respective MNOs were clearly visible. In a second step, I evaluated measurements from during the Covid-19 induced lockdown in Austria. The analysis of the meta-parameters revealed that throughput significantly decreased, but the data did not provide any further insight into its cause. Thus, I conducted a large-scale analysis of the throughput time series with the help of the latent space representation of this lockdown data set. The results offered a more detailed insight into the data set; however, it is still hard to say to what extent the data represents the performance during the lockdown. The quality of the crowdsourced data is, in my opinion, a general issue in this field. We have seen that, while the representations of the self-collected lab and validation data are meaningful, this is not always the case for the crowdsourced data sets — this further highlights the importance of reliable ground truth data. Additionally, the latent space interpretation proved to be challenging in some cases, as unsupervised autoencoders do not offer a direct way of controlling which features get picked up. I think that supervised and semi-supervised autoencoders are particularly promising to tackle those issues.

Overall, this work successfully addressed the challenges discussed in Chapter 1. In my opinion, it can also act as a case study on how to conduct inference in an environment with only a small number of labeled samples and a limited availability of parameters. The semi-supervised approach, which generalizes from a self-collected data set by using unlabeled samples, allowed for robust classification of limited and unlimited measurements. I think that the generalization from a self-collected data set by using unlabeled samples could also be beneficial in other problems regarding the inference of context in mobile network benchmarking. The problem of tariff-limit detection does, for instance, exhibit

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

similar challenges than indoor/outdoor classification. Both problems share the *limited availability of parameters* and lack sufficient numbers of *labeled data*. Representation learning does have an even broader scope of possible applications. We have seen that it offers techniques to make sense of large data sets with high-dimensional samples. As such, it is a powerful tool for data exploration.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

APPENDIX A

Used Code & Models

The following Listings provide the key code sections used throughout this work. In the following, the implementation of the feature generation process from chapter 3 is provided.

Listing A.1: Feature Generation

```
import pandas as pd
import numpy as np
from sklearn.covariance import GraphicalLassoCV
import scipy

t_lab = pd.read_pickle("t_lab_data.pb")

t_lab_series_normalized = t_lab.series.div(
        t_lab.series.max(axis=1), axis=0
)
unlimited = t_lab_series_normalized[t_lab.meta.label == 0].values
limited = t_lab_series_normalized[t_lab.meta.label == 1].values

'''
Test Statistic
'''

def estimate_moments(series):
    C = GraphicalLassoCV(cv=5).fit(series).covariance_
    m = series.mean()
    return C, m
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
def test_statistic(x, m0, m1, C0, C1):
    x_0 = x - m0
    x_1 = x - m1

    C0_inv = np.linalg.inv(C0)
    C1_inv = np.linalg.inv(C1)
    deter = np.linalg.det(C0) * np.linalg.det(C1_inv)

    x_0_term = np.matmul(x_0, C0_inv) @ x_0.transpose()
    x_1_term = np.matmul(x_1, C1_inv) @ x_1.transpose()

    return (x_0_term - x_1_term) + np.log(deter)

def get_statistic_vector(series, m0, m1, C0, C1):
    test_statistics = []
    for _, x in series.iterrows():
        t_x = testStatistic(x.values, m0, m1, C0, C1)
        test_statistics.append(t_x)
    return test_statistics

m_unlimited, C_unlimited = estimate_moments(unlimited)
m_limited, C_limited = estimate_moments(limited)

t_lab['meta', 'statistic'] = get_statistic_vector(
    t_lab_series_normalized,
    m_unlimited, m_limited,
    C_unlimited, C_limited
)

'''
Skew Statistic
'''

t_lab['meta', 'skew'] = scipy.stats.skew(
    t_lab_series_normalized, axis=1, bias=False
)

'''
PAR
'''

t_lab['meta', 'par'] = 1 / np.mean(
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
    t_lab_series_normalized, axis=1
)

'''
Rate Beginning & End
'''

t_lab['meta', 'r_start'] = t_lab.series.iloc[:, :10].mean(axis=1)
t_lab['meta', 'r_end'] = t_lab.series.iloc[:, 20:].mean(axis=1)
```

The below listing provides the code used for LabelSpreading used in 3. Hereby, I ressort to Sklearn [BLB+13] for a high performance implementation. Additionally, I use Hyperopt [BKE+15] for hyperparameter optimization.

Listing A.2: Label Spreading

```
from sklearn.preprocessing import StandardScaler
from sklearn.semi_supervised import LabelSpreading
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, f1_score
import pandas as pd
import numpy as np
from hyperopt import fmin, tpe, hp, STATUS_OK

training_data = pd.read_pickle("t_lab_data.pb")
validation_data = pd.read_pickle("vcl_vi_combined.pb")
rtr_data = pd.read_pickle("rtr_data.pb")

'''
Standardize Dataset for Label Propagation
'''

scaler = StandardScaler()
scaler.fit(training_data.features)

input_training = scaler.transform(training_data.features)
input_gt = scaler.transform(validation_data.features)
input_rtr = scaler.transform(rtr_data.features)

input_training = pd.DataFrame(input_training)
input_gt = pd.DataFrame(input_gt)
input_rtr = pd.DataFrame(input_rtr)
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
'''
Label Spreading Parameter Optimization
'''

def objective(params):
    def loss_fn(params):
        ls = LabelSpreading(
            gamma=params['gamma'],
            alpha=params['alpha']
        )
        ls.fit(
            input_training.iloc[:, :-1],
            training_data.meta.label
        )
        return -ls.score(
            input_gt,
            validation_data.meta.label
        )
    return {
        'loss': loss_fn(params),
        'status': STATUS_OK
    }

hp_space = {'gamma': hp.loguniform('gamma', 1e-3, 5),
            'alpha': hp.loguniform('alpha', 1e-3, 5)}

optimized_hyperparams = fmin(
    objective,
    space=hp_space,
    algo=tpe.suggest,
    max_evals=1000
)

gamma = np.log(optimized_hyperparams['gamma'])
alpha = np.log(optimized_hyperparams['alpha'])

'''
Label Spreading & Cross Validation
'''

NUM_ITERATIONS = 100
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
accuracy_semi = []
score_semi = []
accuracy_super = []
score_super = []

for _ in range(0, NUM_ITERATIONS):
    gt_tr, gt_te, _, gt_tar_te = train_test_split(
        input_gt,
        validation_data.meta.label,
        test_size=0.2,
        shuffle=True
    )

    #Prepare Training Data Set
    gt_tr['label'] = -1
    input_training['label'] = training_data.meta.label
    combined_semi = pd.concat([input_data, gt_tr])

    #Super
    lab_prop_semi = LabelSpreading(
        gamma=gamma, alpha=alpha
    )
    lab_prop_semi.fit(
        combined_semi.iloc[:, :-1],
        combined_semi['label']
    )

    lab_prop_super = LabelSpreading(
        gamma=gamma, alpha=alpha
    )
    lab_prop_super.fit(
        input_training.iloc[:, :-1],
        training_data.meta.label
    )

    accuracy_semi.append(
        lab_prop_semi.score(
            gt_te,
            gt_tar_te
        )
    )
    score_semi = f1_score(
        lab_prop_semi.score(
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
                    gt_te,
                    gt_tar_te
                )
            )

            accuracy_super.append(
                lab_prop_super.score(
                    input_gt,
                    validation_data.meta.label
                )
            )
            score_super = f1_score(
                lab_prop_super.score(
                    input_gt,
                    validation_data.meta.label
                )
            )

        print(f"Accuracy Semi-Supervised : {np.median(accuracy_semi)}")
        print(f"F1 Semi-Supervised : {np.median(score_semi)}")

        print(f"Accuracy Supervised : {np.median(accuracy_super)}")
        print(f"F1 Supervised : {np.median(score_super)}")

        '''
        MNO-Comparison
        '''

        mno_a = pd.read_pickle("mno_a.pb")
        mno_a_features = scaler.transform(mno_a.features)
        mno_a_features = pd.DataFrame(mno_a_features)

        mno_b = pd.read_pickle("mno_b.pb")
        mno_b_features = scaler.transform(mno_b.features)
        mno_b_features = pd.DataFrame(mno_b_features)

        mno_c = pd.read_pickle("mno_c.pb")
        mno_c_features = scaler.transform(mno_c.features)
        mno_c_features = pd.DataFrame(mno_c_features)

        mno_classifier = LabelSpreading(
            gamma=gamma, alpha=alpha
        )
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
    mno_classifier.fit(
        pd.concat([input_data, input_gt])
        pd.concat(
            [training_data.meta.label,
            validation_data.meta.label]
        )
    )

    labels_a = mno_classifier.predict(mno_a_features)
    labels_b = mno_classifier.predict(mno_b_features)
    labels_c = mno_classifier.predict(mno_c_features)
```

The below listing provides the code for the RMBT class, which incorporates the resampling procedure used throughout the work. The *set_raw_data* method is hereby used to process data from RTR-Netztest.

Listing A.3: Resampling

```
import pandas as pd
import numpy as np

class RMBT:
    def __init__(self, open_test_uuid):
        self._open_test_uuid = open_test_uuid

        #Time Series Parameters
        self._resolution = 0.1
        self._total_duration = 7

    def set_raw_data(self, overview, threads, signals):
        #Data From RTR-Netztest
        self._signal = signals
        self._threads = threads
        self._overview = overview

    def __str__(self):
        #Also add the downlink and uplink information
        return str(self._open_test_uuid)

    '''
    Time Series Processing
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
'''

def _resample_datarate(self, input_df):
    resolution = self._resolution
    total_duration = self._total_duration

    resampled_time = np.arange(
            0,
            total_duration,
            resolution
    )
    accumulated = np.zeros(len(resampled_time))

    for thread in input_df.thread.unique():
        per_frame_df = input_df[input_df.thread == thread]

        time_series = list(
            per_frame_df.time_elapsed_ns * 1e-9
        )
        byte_series = list(per_frame_df.bytes_total)

        time_series.insert(0, 0)
        byte_series.insert(0, 0)

        accumulated += np.interp(
            resampled_time,
            time_series,
            byte_series
        )

    return pd.Series(
        list(
            np.gradient(accumulated)
            * 1e-5 * 8 * 0.1/resolution
        ),
        resampled_time
    )

'''
Public Methods and Properties
'''

@property
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
    def downlink_series(self):
        #Resample Downlink Series
        return self._resample_datarate(
            self._threads[ self._threads.type=='dl']
        )

    @property
    def uplink_series(self):
        #Resample Uplink Series
        return self._resample_datarate(
            self._threads[ self._threads.type=='ul']
        )


rmbt = RMBT('open_test_uuid_placeholder')
rmbt.downlink_series
rmbt.uplink_series
```

The below code was used to approximate the logical AND in chapter 4. Hereby, I ressort to [C+15] for an efficient implementation of neural networks.

Listing A.4: Approximating AND

```
from tensorflow import keras
from tensorflow.keras.layers import Dense
import numpy as np
import pandas as pd

size = 1000
x0 = np.random.choice([0, 1], size=(size,), p=[1./2, 1./2])
x1 = np.random.choice([0, 1], size=(size,), p=[1./2, 1./2])

data = pd.DataFrame([x0, x1]).transpose()
data['output'] = data[0] & data[1]
data['output'] = data['output'].apply(int)

'''
Linear Model
'''

model = keras.Sequential([
    Dense(1,
        activation="linear",
        input_shape=(2,)
    ),
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
        Dense(1,
            activation="linear",
            use_bias=False
        )
    ])

    model.compile(
        optimizer='adam',
        loss='mse'
    )

    model.fit(
        x=data.iloc[:,:-1],
        y=data['output'],
        batch_size=50,
        epochs=500
    )

    model.get_weights()
    model.predict(data.iloc[:, :-1])

    '''
    Model with Nonlinearity
    '''

    model = keras.Sequential([
        Dense(1,
            activation="linear",
            input_shape=(2,)
        ),
        Dense(1,
            activation="relu",
            use_bias=False
        )
    ])

    model.compile(
        optimizer='adam',
        loss='mse'
    )

    model.fit(
        x=data.iloc[:,:-1],
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
        y=data['output'],
        batch_size=50,
        epochs=200
    )

    model.get_weights()
    model.predict(data.iloc[:, :-1])
```

I used the below code to generate the plots for the overfitting example in chapter 4.

Listing A.5: Ridge Regression & OLS

```
import pandas as pd
import numpy as np

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.pipeline import Pipeline

x_true = np.linspace(0,1,100)
x = np.random.uniform(high=1, size=5)
y = np.array([
    np.random.normal(
        loc**2+0.5, scale=0.1)
        for loc in x
])

data = pd.DataFrame([x, y]).transpose()

ordinary_ls_dict = {}
ridge_dict = {}

print_ols_dict = {}
print_ridge_dict = {}

for degree in [1, 3, 5]:
    model = Pipeline([
        ('poly', PolynomialFeatures(degree=degree)),
        ('linear', LinearRegression(fit_intercept=False))
    ])

    model = model.fit(x[:, np.newaxis], y)
    y_pred = model.predict(x_true[:, np.newaxis])
    ordinary_ls_dict[degree] = model.named_steps['linear'].coef_
    print_ols_dict[degree] = y_pred
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
    model = Pipeline([
        ('poly', PolynomialFeatures(degree=degree)),
        ('ridge', Ridge(alpha=.1))
    ])
    model = model.fit(x[:, np.newaxis], y)
    y_pred = model.predict(x_true[:, np.newaxis])
    ridge_dict[degree] = model.named_steps['ridge'].coef_
    print_ridge_dict[degree] = y_pred
```

The below code was used to generate the plots regarding regularization in neural networks in chapter 4. Here I used the dataset from .

Listing A.6: Overfitting & Regularization

```
import pandas as pd
from tensorflow import keras
import matplotlib.pyplot as plt

train = pd.read_csv("./train.csv")
x_train, y_train = train.iloc[:, 1:], train.iloc[:, 1]

test = pd.read_csv("./test.csv")
x_test, y_test = test.iloc[:, 1:], test.iloc[:, 1]

'''
Overfitting Model
'''

model = keras.Sequential()
model.add(keras.layers.Dense(
        300, activation="relu",
        input_dim=300
))
model.add(keras.layers.Dense(128, activation="relu"))
model.add(keras.layers.Dense(64, activation="relu"))
model.add(keras.layers.Dense(1, activation="sigmoid"))
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
    x_train, y_train,
    batch_size=32,
    epochs=100,
    validation_split=0.2,
    shuffle=True
)

loss = history.history['accuracy']
val_loss = history.history['val_accuracy']

plt.figure()
plt.title("No Regularization")
plt.plot(loss, label="Accuracy")
plt.plot(val_loss, label="Validation Accuracy")
plt.legend()
plt.xlabel("Epochs")
plt.show()

'''
Regularized Model
'''

model = keras.Sequential()
model.add(keras.layers.Dense(16, activation="relu", input_dim=10))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(8, activation="relu"))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(1, activation="sigmoid"))
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

history_reg = model.fit(
    x_train_new, y_train,
    batch_size=32,
    epochs=100,
    validation_split=0.2,
    shuffle=True
)

loss = history_reg.history['loss']
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
val_loss = history_reg.history['val_loss']

figure = plt.figure()
plt.plot(loss, label="Training")
plt.plot(val_loss, label="Test")
plt.legend()
plt.ylabel("Loss")
plt.ylim(0, 1.25)
plt.xlabel("Epochs")
plt.show()
```

The below code provides the basic structure for the implementation of a vanilla autoencoder used throughout chapter 5.

Listing A.7: Vanilla Autoencoder Basic Structure

```
from tensorflow.python.keras.layers import Dense, Input
from tensorflow.python.keras.models import Model
from tensorflow.python.keras.regularizers import l1

'''
Basic Structure of a Vanilla Autoencoder
'''

def encoding_network(input, dimension):
    x = Dense(...)(input)
    #...
    x = Dense(dimension, activation='relu')(x)
    return x

def decoding_network(latent_input, dimension):
    x = Dense(...)(latent_input)
    #...
    x = Dense(dimension, activation='relu')(x)
    return x

ORIGINAL_DIMENSION = 70
LATENT_DIMENSION = 2

'''
Generate the Model
'''

main_input = Input(
    shape=(ORIGINAL_DIMENSION,),
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
    name="Main-Input"
)

latent_input = Input(
    shape=(LATENT_DIMENSION,),
    name="Latent-Input"
)

latent_output = encoding_network(
    main_input,
    LATENT_DIMENSION
)
main_output = decoding_network(
    latent_input,
    ORIGINAL_DIMENSION
)

encoder = Model(
    main_input,
    latent_output,
    name="Encoder"
)

decoder = Model(
    latent_input,
    main_output,
    name="Decoder"
)

autoencoder = Model(
    main_input,
    decoder(encoder(main_input)),
    name="Autoencoder"
)


autoencoder.compile(loss="...", optimizer="...")
autoencoder.fit(...)

'''
Prediction
'''
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
z = encoder.predict(x)
x_prime = decoder.predict(z)
x_prime = autoencoder.predict(x)
```

Implementing the sparse autoencoder from chapter 5 requires changes to the encoding_network function.

Listing A.8: Sparse Autoencoder

```
from tensorflow.python.keras.layers import Dense, Input
from tensorflow.python.keras.models import Model
from tensorflow.python.keras.regularizers import l1

'''
Sparse Autoencoder - Activity regularization for Latent Layer
'''

def encoding_network(input, dimension):
    x = Dense(...)(input)
    #...
    x = Dense(
        dimension,
        activation="relu",
        activity_regularizer=l1(SPARSITY_PENALTY),
    )(x)
    return x

def decoding_network(latent_input, dimension):
    x = Dense(...)(latent_input)
    #...
    x = Dense(dimension, activation='relu')(x)
    return x
```

Below the code for the variational autoencoder from chapter 5 is provided.

Listing A.9: Variational Autoencoder

```
from tensorflow.python.keras.layers import Dense, Input
from tensorflow.python.keras.models import Model
from tensorflow.python.keras.layers.core import Lambda
from tensorflow.python.keras import backend as K
from tensorflow.python.keras.losses import mse

'''
Basic Structure of a Variational Autoencoder
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
'''

def sampling_procedure(args):
    z_mean, z_log_var = args
    batch = K.shape(z_mean)[0]
    dim = K.int_shape(z_mean)[1]

    epsilon = K.random_normal(shape=(batch, dim))
    return z_mean + K.exp(0.5 * z_log_var) * epsilon

def encoding_network(input, dimension):
    x = Dense(16)(input)
    #...
    z_mean = Dense(
        dimension,
        activation="linear",
        name='z_mean'
    )(x)
    z_log_var = Dense(
        dimension,
        activation="linear",
        name='z_log_var'
    )(x)

    #Latent Layer Sampling Output
    z = Lambda(
        sampling_procedure,
        output_shape=(dimension,), name='z'
    )([z_mean, z_log_var])

    return (z_mean, z_log_var, z)

def decoding_network(latent_input, dimension):
    x = Dense(16)(latent_input)
    #...
    x = Dense(dimension, activation='relu')(x)
    return x

ORIGINAL_DIMENSION = 70
LATENT_DIMENSION = 2
BETA = 1 #For BETA-VAE

'''
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
Generate the Model
'''

main_input = Input(
    shape=(ORIGINAL_DIMENSION,),
    name="Main-Input"
)

latent_input = Input(
    shape=(LATENT_DIMENSION,),
    name="Latent-Input"
)

latent_output = encoding_network(
    main_input,
    LATENT_DIMENSION
)

main_output = decoding_network(
    latent_input,
    ORIGINAL_DIMENSION
)

encoder = Model(
    main_input,
    latent_output,
    name="Encoder"
)

decoder = Model(
    latent_input,
    main_output,
    name="Decoder"
)

autoencoder = Model(
    main_input,
    #Only use the sampled values as output
    decoder(encoder(main_input)[2]),
    name="Autoencoder"
)

def get_loss(dimension, beta):
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
    #Reconstruction Loss
    reconstruction_loss = mse(main_input, main_output)

    #Kullback-Leibler Divergence
    (z_mean, z_log_var, _) = latent_output
    kl_loss = 1 + z_log_var - K.square(z_mean) - K.exp(
        z_log_var
    )
    kl_loss = K.sum(kl_loss, axis=-1)
    kl_loss *= -0.5


    return beta * kl_loss + reconstruction_loss

autoencoder.add_loss(get_loss(LATENT_DIMENSION, BETA))
autoencoder.compile(optimizer='...')
autoencoder.fit(...)

'''
Prediction
'''

z_mean, z_log_var, z = encoder.predict(x)
x_prime = decoder.predict(z)
x_prime = autoencoder.predict(x)
```

Below the code for a conditional autoencoder — see 5 — is provided.

Listing A.10: Conditional Autoencoder Architecture

```
from tensorflow.python.keras.layers import Dense, Input
from tensorflow.python.keras.layers import concatenate, Flatten
from tensorflow.python.keras.models import Model

def encoding_network(original_input, meta_input):
    meta = Dense(ORIGINAL_DIMENSION)(meta_input)
    #...
    series = Dense(16)(original_input)
    #...

    x = concatenate([
        meta,
        series
    ], name="combined_start_encoder")
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
        flatten_layer = Flatten()
        x = flatten_layer(x)

        #Encoding Network
        x = Dense(16)(x)
        #...
        x = Dense(LATENT_SIZE)(x)
        return x

    def decoding_network(original_input, meta_input):
        x = Dense(16)(latent_input)
        #...
        x = Dense(LATENT_SIZE, activation='relu')(x)
        return x

    META_INPUT_DIMENSION = 1
    ORIGINAL_DIMENSION = 70
    LATENT_SIZE = 2

    meta_input = Input(
        shape=(META_INPUT_DIMENSION, ),
        name="Meta-Input"
    )

    latent_input = Input(
        shape=(LATENT_SIZE, ),
        name="Latent-Input"
    )

    main_input = Input(
        shape=(ORIGINAL_DIMENSION, ),
        name="Main-Input"
    )

    latent_output = encoding_network(
        main_input,
        meta_input
    )

    main_output = decoding_network(
        latent_input,
        meta_input
    )
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
encoder = Model(
    [main_input, meta_input],
    latent_output,
    name="Encoder"
)

decoder = Model(
    [latent_input, meta_input],
    main_output,
    name="Decoder"
)

overall_output = decoder([
    encoder([
        main_input,
        meta_input,
    ]),
    meta_input
])

autoencoder = Model(
    [main_input, meta_input],
    overall_output,
    name="Autoencoder"
)

def get_loss():
    reconstruction_loss = mse(main_input, overall_output)
    return reconstruction_loss

autoencoder.add_loss(get_loss())
autoencoder.compile(optimizer='...')
autoencoder.fit(...)

'''
Predict
'''

z = encoder.predict([x, m])
x_prime = decoder.predict([z, m])
x_prime = autoencoder.predict([x, m])
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

The following caption provides the code for the generation of the rectangular series used in 5.

Listing A.11: Rect-Series Generator

```
from numpy.random import multivariate_normal
from scipy.sparse import diags
import numpy as np
import pandas as pd

class TwoMeanSymmetric():
    @property
    def _mean_functions(self):
        result = []
        for mean1 in np.linspace(0.2, 0.8, 5):
            for mean2 in np.linspace(0.2, 0.8, 5):
                series = np.concatenate(
                    (np.repeat(mean1, 35), np.repeat(mean2, 35))
                )
                result.append(
                    series
                )
        return result

class FiveMeanSymmetric():
    @property
    def _mean_functions(self):
        result = []
        for mean1 in np.linspace(0.2, 0.8, 5):
            for mean2 in np.linspace(0.2, 0.8, 5):
                for mean3 in np.linspace(0.2, 0.8, 5):
                    for mean4 in np.linspace(0.2, 0.8, 5):
                        for mean5 in np.linspace(0.2, 0.8, 5):
                            series = np.concatenate(
                                (
                                    np.repeat(mean1, 14),
                                    np.repeat(mean2, 14),
                                    np.repeat(mean3, 14),
                                    np.repeat(mean4, 14),
                                    np.repeat(mean5, 14)
                                )
                            )
                            result.append(
                                series
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
                              )
        return result

class MultivariateGaussianGenerator():
    def __init__(
            self,
            covariance_scaling=0.2,
            correlation_length=30,
            batch_size=100
        ):
        self._covariance_scaling = covariance_scaling
        self._correlation_length = correlation_length
        self._length = 70

        self._data = self._generate_data(batch_size)

    '''
    Public
    '''

    def get_data(self):
        return self._data

    '''
    Private
    '''

    def _generate_data(self, batch_size):
        size = self._correlation_length
        length = self._length

        correlation = np.concatenate(
            (
                np.linspace(0, 1, size),
                np.linspace(1, 0, size)[1:]
            )
        )
        indices = np.arange(1-size, size)

        x_data = []
        for mean_vector in self._mean_functions:
            cov_matrix = diags(
                correlation,
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
                    indices,
                    shape=(length, length)
                ).toarray()

                x = np.random.multivariate_normal(
                    mean_vector,
                    cov_matrix * self._covariance_scaling,
                    batch_size
                )

                x_data.append(x)

            df = pd.DataFrame(
                np.concatenate(x_data)
            ).sample(frac=1)

            return df

class TwoMeanGaussian(
            MultivariateGaussianGenerator,
            TwoMeanSymmetric
        ):
        pass

class FiveMeanGaussian(
            MultivariateGaussianGenerator,
            FiveMeanSymmetric
        ):
        pass

FiveMeanGaussian().get_data()
TwoMeanGaussian().get_data()
```

In the following, the code used to generate the QAM series from chapter 5 is provided.

Listing A.12: QAM-Series-Generator

```
import pandas as pd
import numpy as np

SAMPLE_SIZE = 10

CONSTELLATION = {
    '0000' : (-3, -3),
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
    '0001' : (-3, -1),
    '0010' : (-3, 3),
    '0011' : (-3, 1),
    '0100' : (-1, -3),

    '0101' : (-1, -1),
    '0110' : (-1, 3),
    '0111' : (-1, 1),
    '1000' : (3, -3),
    '1001' : (3, -1),

    '1010' : (3, 3),
    '1011' : (3, 1),
    '1100' : (1, -3),
    '1101' : (1, -1),
    '1110' : (1, 3),
    '1111' : (1, 1)
}

def modulate(symbol):
    f = 5
    w = 2 * np.pi * f
    T_s = 0.5

    I, Q = CONSTELLATION[symbol]
    t = np.linspace(0, T_s, 70)
    return I * np.cos(w*t) - Q * np.sin(w*t)

data = []
for key, item in constellation.items():
    I, Q = item
    signal = modulate(key)
    df = pd.DataFrame(
        signal + np.random.normal(
            scale=0.5,
            size=(
                SAMPLE_SIZE,
                len(signal)
            )
        )
    )
    data.append(df)
```

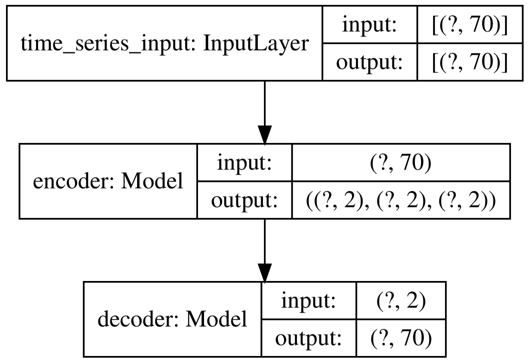
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
data = pd.concat(data, ignore_index=True).transpose()
```

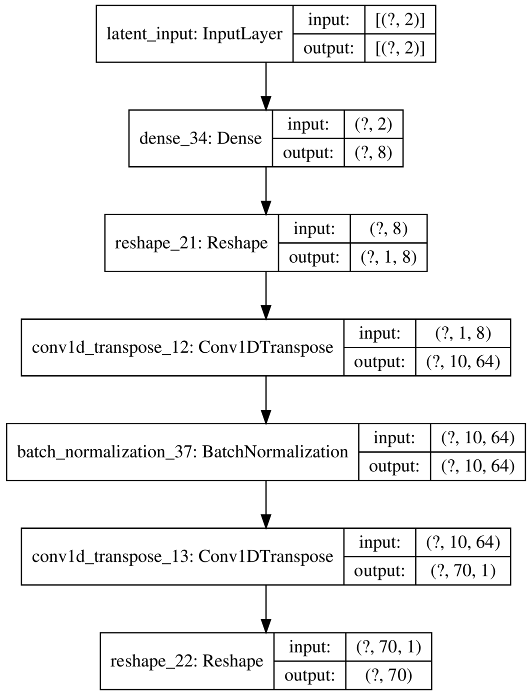
The below figures describe the network layout used for the experiments in 5 and 6. The complete code can be obtained by implementing the given networks into the encoding_network and decoding_network functions from above.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

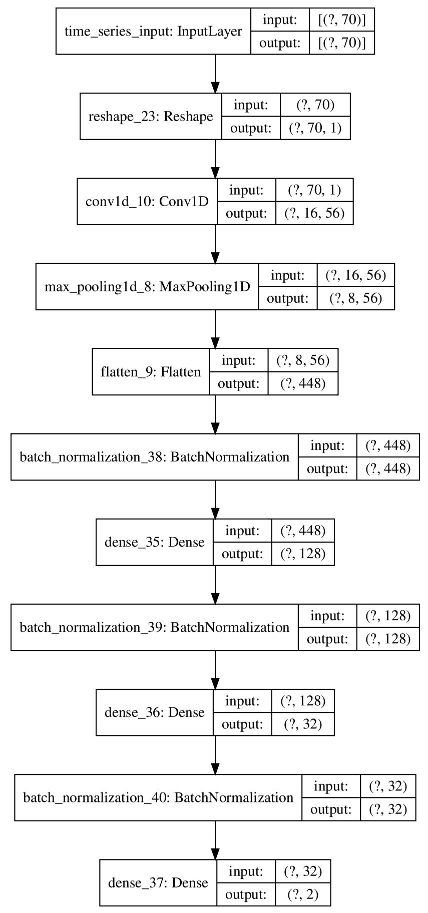


(a) Complete Autoencoder

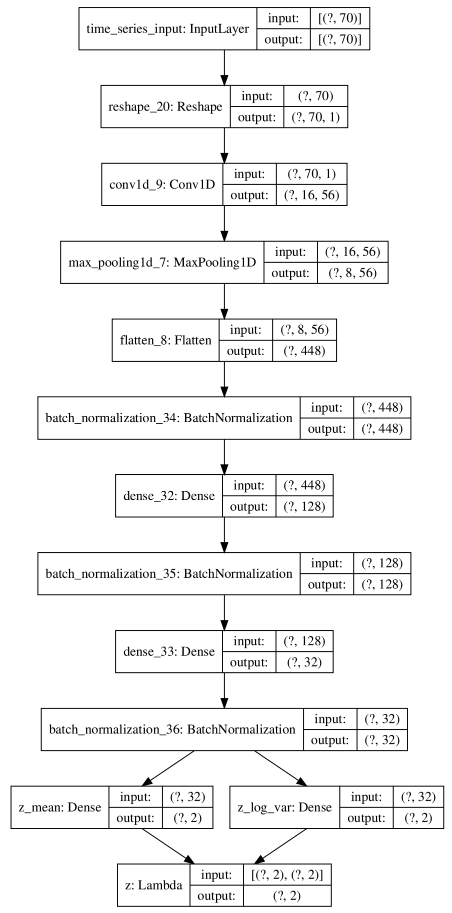


(b) Decoder

Figure A.1: Unconditional Model


time_series_input: InputLayer | input: [(?, 70)] | output: [(?, 70)]
reshape_23: Reshape | input: (?, 70) | output: (?, 70, 1)
conv1d_10: Conv1D | input: (?, 70, 1) | output: (?, 16, 56)
max_pooling1d_8: MaxPooling1D | input: (?, 16, 56) | output: (?, 8, 56)
flatten_9: Flatten | input: (?, 8, 56) | output: (?, 448)
batch_normalization_38: BatchNormalization | input: (?, 448) | output: (?, 448)
dense_35: Dense | input: (?, 448) | output: (?, 128)
batch_normalization_39: BatchNormalization | input: (?, 128) | output: (?, 128)
dense_36: Dense | input: (?, 128) | output: (?, 32)
batch_normalization_40: BatchNormalization | input: (?, 32) | output: (?, 32)
dense_37: Dense | input: (?, 32) | output: (?, 2)

(a) Vanilla

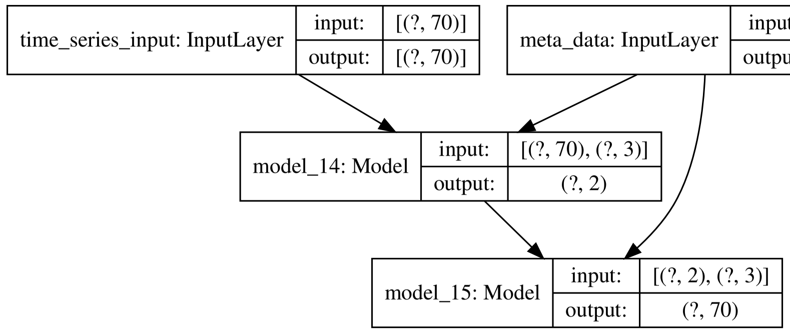

time_series_input: InputLayer | input: [(?, 70)] | output: [(?, 70)]
reshape_20: Reshape | input: (?, 70) | output: (?, 70, 1)
conv1d_9: Conv1D | input: (?, 70, 1) | output: (?, 16, 56)
max_pooling1d_7: MaxPooling1D | input: (?, 16, 56) | output: (?, 8, 56)
flatten_8: Flatten | input: (?, 8, 56) | output: (?, 448)
batch_normalization_34: BatchNormalization | input: (?, 448) | output: (?, 448)
dense_32: Dense | input: (?, 448) | output: (?, 128)
batch_normalization_35: BatchNormalization | input: (?, 128) | output: (?, 128)
dense_33: Dense | input: (?, 128) | output: (?, 32)
batch_normalization_36: BatchNormalization | input: (?, 32) | output: (?, 32)
z_mean: Dense | input: (?, 32) | output: (?, 2)
z_log_var: Dense | input: (?, 32) | output: (?, 2)
z: Lambda | input: [(?, 2), (?, 2)] | output: (?, 2)

(b) Variational

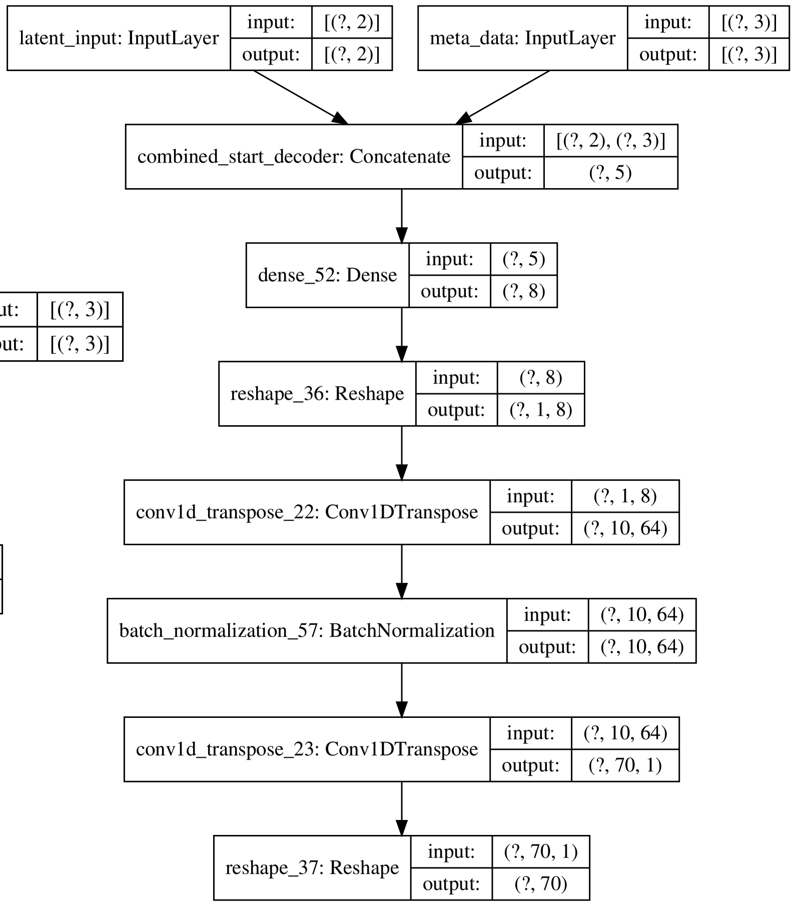
Figure A.2: Unconditional Encoders

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

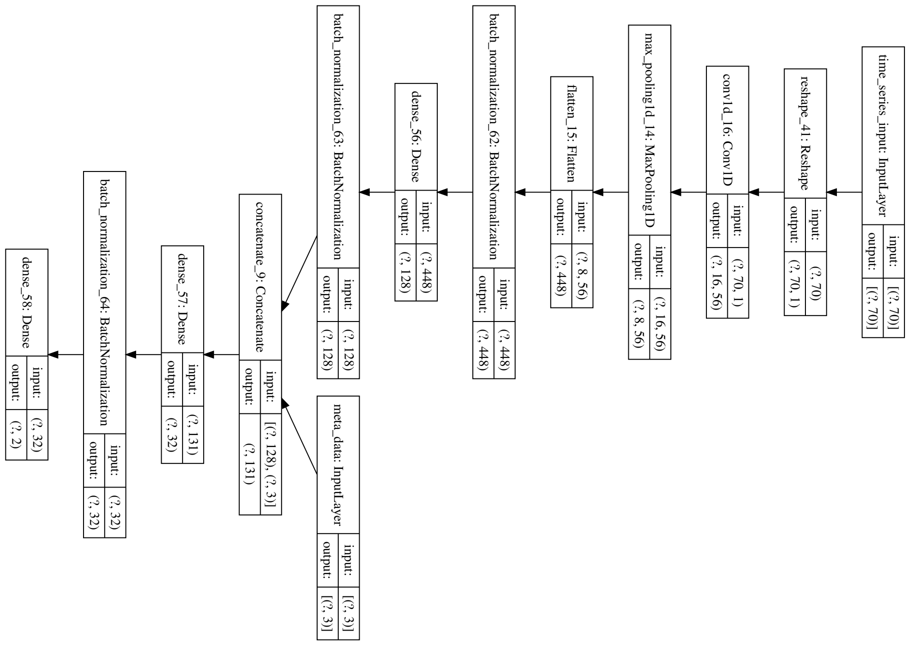


(a) Conditional Model

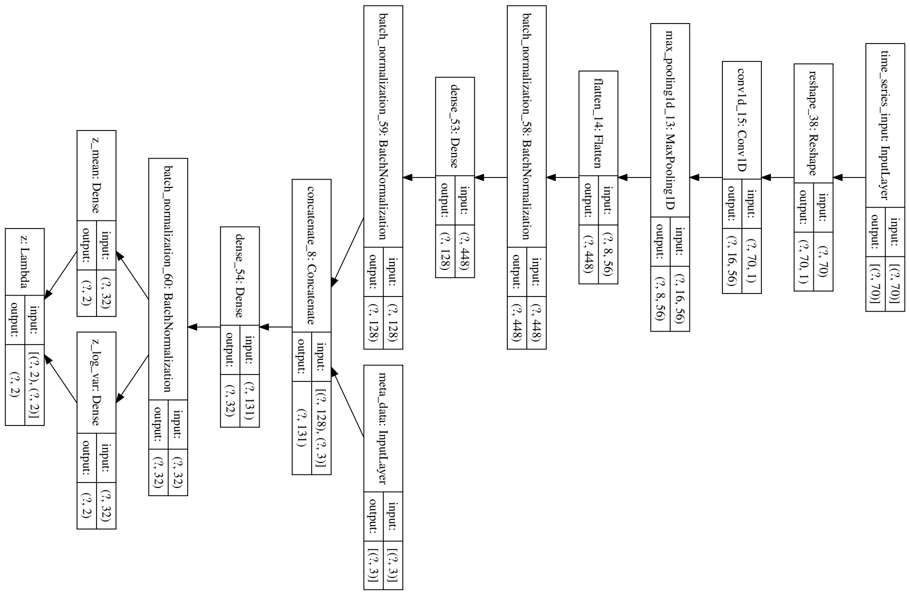


(b) Conditional Decoder

Figure A.3: Conditional Model & Decoder



(a) Vanilla



(b) Variational

Figure A.4: Conditional Encoders

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

List of Tables

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Algorithms

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[Bar93] Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

[BCV13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[BDLR06] Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. 11 label propagation and quadratic criterion. 2006.

[BGC17] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. MIT press, 2017.

[BHP+18] Christopher P Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-vae. *arXiv preprint arXiv:1804.03599*, 2018.

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[BKE+15] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, 2015.

[BLB+13] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, et al. Api design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.

[BYR17] Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, 12(7), 2017.

[C+15] François Chollet et al. Keras. `https://keras.io`, 2015.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[cis18] Cisco visual networking index 2017-2022. Technical report, CISCO, 2018.

[cis20] Cisco annual internet report 2018-2023. Technical report, CISCO, 2020.

[CKLZY12] Georgios Chatzimilioudis, Andreas Konstantinidis, Christos Laoudias, and Demetrios Zeinalipour-Yazti. Crowdsourcing with smartphones. *IEEE Internet Computing*, 16(5):36–44, 2012.

[DH07] Theo Dunnewijk and Staffan Hultén. A brief history of mobile communication in europe. *Telematics and Informatics*, 24(3):164–179, 2007.

[DV16] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[FR17] Simon Foucart and Holger Rauhut. A mathematical introduction to compressive sensing. *Bull. Am. Math*, 54:151–165, 2017.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[GG10] Ian Glover and Peter M Grant. *Digital communications*. Pearson Education, 2010.

[GM16] Anupriya Gogna and Angshul Majumdar. Semi supervised autoencoder. In *International Conference on Neural Information Processing*, pages 82–89. Springer, 2016.

[HMP$^+$17] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *Iclr*, 2(5):6, 2017.

[Hoc98] Sepp Hochreiter. Recurrent neural net learning and vanishing gradient. *International Journal Of Uncertainity, Fuzziness and Knowledge-Based Systems*, 6(2):107–116, 1998.

[HSW$^+$89] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[IIT$^+$18] Sae Iwata, Kazuaki Ishikawa, Toshinori Takayama, Masao Yanagisawa, and Nozomu Togawa. Robust indoor/outdoor detection method based on sparse gps positioning information. In *2018 IEEE 8th International Conference on Consumer Electronics-Berlin (ICCE-Berlin)*, pages 1–4. IEEE, 2018.

[Iof17] Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Advances in neural information processing systems*, pages 1945–1953, 2017.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

[JHC$^+$09] Keon Jang, Mongnam Han, Soohyun Cho, Hyung-Keun Ryu, Jaehwa Lee, Yeongseok Lee, and Sue B Moon. 3g and 3.5 g wireless network performance measured from moving cars and high-speed trains. In *Proceedings of the 1st ACM workshop on Mobile internet through cellular networks*, pages 19–24, 2009.

[KAA$^+$19] Konstantinos Kousias, Özgü Alay, Antonios Argyriou, Andra Lutu, and Michael Riegler. Estimating downlink throughput from end-user measurements in mobile broadband networks. In *2019 IEEE 20th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pages 1–10. IEEE, 2019.

[KMA$^+$17] Konstantinos Kousias, Cise Midoglu, Ozgu Alay, Andra Lutu, Antonios Argyriou, and Michael Riegler. The same, only different: Contrasting mobile operator behavior from crowdsourced dataset. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–6. IEEE, 2017.

[LBOM12] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[LCZ$^+$13] Nicholas D Lane, Yohan Chon, Lin Zhou, Yongzhe Zhang, Fan Li, Dongwon Kim, Guanzhong Ding, Feng Zhao, and Hojung Cha. Piggyback crowdsensing (pcs) energy efficient crowdsourcing of mobile sensor data by exploiting smartphone app opportunities. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, pages 1–14, 2013.

[LGTB97] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

[LLPS93] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[LPW18] Lei Le, Andrew Patterson, and Martha White. Supervised autoencoders: Improving generalization performance with unsupervised regularizers. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 107–117. Curran Associates, Inc., 2018.

[MH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[MRB15] Mahesh K Marina, Valentin Radu, and Konstantinos Balampekos. Impact of indoor-outdoor context on crowdsourcing based mobile coverage analysis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, pages 45–50, 2015.

[MS16] Cise Midoglu and Philipp Svoboda. Opportunities and challenges of using crowdsourced measurements for mobile network benchmarking a case study on rtr open data. In *2016 SAI Computing Conference (SAI)*, pages 996–1005. IEEE, 2016.

[N$^+$11] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[Oda19] Stephen Odaibo. Tutorial: Deriving the standard variational autoencoder (vae) loss function. *arXiv preprint arXiv:1907.08956*, 2019.

[PMB12] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2:417, 2012.

[QWD$^+$16] Junfei Qiu, Qihui Wu, Guoru Ding, Yuhua Xu, and Shuo Feng. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016(1):67, 2016.

[Rai17] Vaclav Raida. Deriving a network perspective of cellular mobile networks based on crowd sourced benchmark tests. Master's thesis, Technische Universität Wien, 2017. Diplomarbeit.

[RKSM14] Valentin Radu, Panagiota Katsikouli, Rik Sarkar, and Mahesh K Marina. A semi-supervised learning approach for robust indoor-outdoor detection with smartphones. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 280–294, 2014.

[RLSR18] Vaclav Raida, Martin Lerch, Philipp Svoboda, and Markus Rupp. Deriving cell load from rsrq measurements. In *2018 Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–6. IEEE, 2018.

[RSKR19] Vaclav Raida, Philipp Svoboda, Martin Kruschke, and Markus Rupp. Constant rate ultra short probing (crusp): Measurements in live lte networks. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[RSR18] Vaclav Raida, Philipp Svoboda, and Markus Rupp. Lightweight detection of tariff limits in cellular mobile networks. In *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–7. IEEE, 2018.

[RVS07] Fabio Ricciato, Francesco Vacirca, and Philipp Svoboda. Diagnosis of capacity bottlenecks via passive monitoring in 3g networks: An empirical analysis. *Computer Networks*, 51(4):1205–1231, 2007.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

[Sco15] David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.

[SLY15] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in neural information processing systems*, pages 3483–3491, 2015.

[SMSV19] Illyyne Saffar, Marie Line Alberi Morel, Kamal Deep Singh, and César Viho. Machine learning with partially labeled data for indoor outdoor detection. In *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–8. IEEE, 2019.

[SZT17] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[VLL+10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.

[Wim19] Leonhard Wimmer. Platform for measuring mobile broadband performance analysis and implementation. Master's thesis, Technische Universität Wien, 2019.

[XWCL15] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[YXS+19] Ye Yuan, Guangxu Xun, Qiuling Suo, Kebin Jia, and Aidong Zhang. Wave2vec: Deep representation learning for clinical temporal data. *Neurocomputing*, 324:31–42, 2019.

[ZBL+04] Dengyong Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Advances in neural information processing systems*, pages 321–328, 2004.

[ZC18] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc.", 2018.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.