



Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften  
Department für Informatik

Masterstudiengang Informatik

Masterthesis

# **Evaluation eines Meta-Learning Ansatzes für Deep Autoencoder im Kontext der Anomalieerkennung in multivariaten Zeitreihen**

vorgelegt von

**Torge Wolff**

Gutachter:

**Prof. Dr. Sebastian Lehnhoff**  
**Dr. Martin Tröschel**

Oldenburg, 14. April 2020



---

## Geschlechtergerechte Sprache

Aus Gründen der Lesbarkeit wird in der vorliegenden Arbeit bei Personenbezeichnungen das generische Maskulinum verwendet.

An dieser Stelle sei jedoch darauf hingewiesen, dass dabei alle Geschlechter mit eingeschlossen sind und eine Benachteiligung der Geschlechter nicht intendiert ist.



---

## Abstract

Complex systems are omnipresent in modern industry. Monitoring the behaviour of these systems generates a considerable amount of multivariate time series data, such as the measured values of sensors distributed in a power plant. The detection of anomalies in these time series data is often a central part of monitoring. Due to their characteristics, machine learning models such as autoencoders are often used in practice for this purpose. However, the operational use of these models poses different challenges that need to be addressed from a machine learning perspective. One of these challenges is the problem of changing distributions in the data, which can lead to a so-called concept drift. If a model is exposed to a strong concept drift after a short operational time, a re-training of the model is necessary. However, complex models require a large amount of training data for retraining, which may not be available after a short time. An approach of learning with few data is meta-learning, which is used especially for the training of artificial neural networks for image recognition.

In the context of the present work it should be investigated whether the adaptation of models for anomaly detection with only a few data is possible by using meta learning algorithms. For this purpose, two models  $M_1$  and  $M_2$  were initially trained in the course of an experiment. The model  $M_2$  was trained using classical learning algorithms such as backpropagation. The model  $M_1$  was trained using the meta-learning algorithm FOMAML. Subsequently, both models were examined on a new data set  $\mathbf{X}_{drifted,anormal}$  with regard to a concept drift. In the next phase of the experiment, both models were adapted with only a few data from the new data set  $\mathbf{X}_{drifted,anormal}$  and then re-evaluated with respect to a concept drift.

The evaluation of the data of the experiment showed that the models for anomaly detection in time series data, which were trained by means of meta-learning, could not be trained correctly. Thus, during the training using meta-learning algorithms, different tasks have to be defined, on the basis of which a model can be evaluated and adapted. The evaluation and training with different tasks makes a model sensitive for new tasks. Since it was not possible to define different tasks for the anomaly detection in time series data, the model trained with meta learning algorithms could not learn the sensitivity, so that it could not be improved in the context of the adaptation to the new data.



## Zusammenfassung

Komplexe Systeme sind in der modernen Industrie allgegenwärtig. Die Überwachung des Verhaltens dieser Systeme generiert dabei eine erhebliche Menge multivariater Zeitreihendaten, wie z.B. die Messwerte der in einem Kraftwerk verteilten Sensoren. Die Erkennung von Anomalien in diesen Zeitreihendaten ist dabei häufig ein zentraler Bestandteil der Überwachung. Aufgrund ihrer Eigenschaften werden in der Praxis dafür oftmals Modelle des *Machine Learnings*, wie z.B. *Autoencoder*, verwendet. Bei dem operativen Einsatz dieser Modelle entstehen jedoch unterschiedliche Herausforderungen, die es aus Sicht des *Machine Learnings* zu behandeln gilt. Eine dieser Herausforderungen ist das Problem sich ändernder Verteilungen in den Daten, was zu einem sog. *Concept Drift* führen kann. Ist ein Modell schon nach kurzer operativer Zeit einem starken *Concept Drift* ausgesetzt, ist ein Re-Training des Modells notwendig. Komplexe Modelle benötigen dabei eine große Anzahl an Trainingsdaten für ein Re-Training, die aber u.U. nach einer kurzen Zeit nicht zur Verfügung stehen. Ein Ansatz des Lernens mit wenigen Daten ist dann das *Meta-Learning*, welches insbesondere bei dem Training von künstlichen neuronalen Netzen zur Bilderkennung verwendet wird.

Im Rahmen der vorliegenden Arbeit sollte untersucht werden, ob die Adaption von Modellen zur Anomalieerkennung mit nur wenigen Daten durch die Verwendung von *Meta-Learning*-Algorithmen möglich ist. Hierfür wurden im Rahmen der Durchführung eines Experimentes zunächst zwei Modelle  $M_1$  und  $M_2$  trainiert. Das Modell  $M_2$  wurde dabei anhand klassischer Lernalgorithmen wie *Back-propagation* trainiert. Das Modell  $M_1$  wurde hingegen mittels des *Meta-Learning*-Algorithmus FO-MAML trainiert. Anschließend wurden beide Modelle auf einem neuen Datensatz  $\mathbf{X}_{drifted,anormal}$  hinsichtlich eines *Concept Drifts* untersucht. In der nächsten Phase des Experimentes wurden beide Modelle mit nur wenigen Daten des neuen Datensatzes  $\mathbf{X}_{drifted,anormal}$  adaptiert und anschließend erneut hinsichtlich eines *Concept Drifts* evaluiert.

Bei der Auswertung der Daten des Experimentes zeigte sich, dass die mittels *Meta-Learning* trainierten Modelle für die Anomalieerkennung in Zeitreihendaten nicht korrekt trainiert werden konnten. So sind bei dem Training mittels *Meta-Learning*-Algorithmen verschiedene Aufgaben zu definieren, anhand derer ein Modell evaluiert und adaptiert werden kann. Die Evaluation und das Training anhand unterschiedlicher Aufgaben lässt ein Modell dabei sensitiv für neue Aufgaben werden. Dadurch, dass es nicht möglich war für die Anomalieerkennung in Zeitreihendaten unterschiedliche Aufgaben zu definieren, konnte das mittels *Meta-Learning*-Algorithmen trainierte Modell die Sensitivität nicht lernen, wodurch es im Rahmen der Adaption an die neuen Daten nicht verbessert werden konnte.



# Inhalt

<b>1. Motivation</b>	<b>1</b>
1.1. Problemstellung . . . . .	2
1.2. Eigener Ansatz . . . . .	4
1.3. Verwandte Arbeiten . . . . .	5
1.4. Aufbau der Arbeit . . . . .	6
<b>2. Grundlagen</b>	<b>7</b>
2.1. Machine Learning . . . . .	7
2.2. <i>Autoencoder</i> . . . . .	22
2.3. Anomalieerkennung . . . . .	25
2.4. <i>Meta-Learning</i> . . . . .	32
2.5. Evaluation von Modellen . . . . .	42
<b>3. Konzeptionierung</b>	<b>47</b>
3.1. Analyse der Zielstellung . . . . .	47
3.2. Versuchsplanung . . . . .	49
3.3. Vorstudie . . . . .	52
3.4. Experiment . . . . .	56
3.5. Technologien und Implementierung . . . . .	59
<b>4. Vorstudie</b>	<b>65</b>
4.1. Motivation und Zielstellung . . . . .	65
4.2. Beschreibung des Datensatzes . . . . .	65
4.3. Preprocessing . . . . .	69
4.4. Durchführung . . . . .	70
4.5. Erkenntnisse . . . . .	72
<b>5. Implementierung</b>	<b>75</b>
5.1. Implementierung des Datengenerators . . . . .	75
5.2. Simulationsprozess zur Erstellung der Daten . . . . .	81
5.3. Implementierung der Anomalieerkennung . . . . .	82
5.4. Implementierung des <i>Meta-Learnings</i> . . . . .	87
5.5. Wrapper-Klassen für die Durchführung der Experimente . . . . .	94
<b>6. Experiment</b>	<b>97</b>
6.1. Motivation und Zielstellung . . . . .	97
6.2. Versuchsaufbau & -ablauf . . . . .	97
6.3. Statistische Versuchsplanung . . . . .	98
6.4. Durchführung . . . . .	102
6.5. Auswertung . . . . .	109

6.6. Evaluation . . . . .	114
<b>7. Fazit &amp; Ausblick</b>	<b>117</b>
7.1. Fazit . . . . .	117
7.2. Ausblick . . . . .	118
<b>A. Abbildungen</b>	<b>121</b>
<b>B. Code-Listings</b>	<b>129</b>
<b>C. Tabellen</b>	<b>139</b>
<b>Glossar</b>	<b>151</b>
<b>Abkürzungen</b>	<b>153</b>
<b>Symbolverzeichnis</b>	<b>155</b>
<b>Abbildungsverzeichnis</b>	<b>157</b>
<b>Tabellenverzeichnis</b>	<b>161</b>
<b>Literatur</b>	<b>163</b>
<b>Index</b>	<b>169</b>

# 1. Motivation

Die Erkennung von Anomalien ist ein wichtiges Problem, das in verschiedenen Forschungsbereichen eingehend untersucht wurde und in vielen Anwendungsbereichen eine wichtige Rolle spielt. Sei es die Erkennung eines Kreditkartenbetruges, die Erkennung von Auffälligkeiten in medizinischen Daten, die Erkennung besonderer Events in Sensordaten oder die Identifikation von Angriffen auf ein Netzwerk. Hawkins definierte eine Anomalie in [Haw80] wie folgt: „*Ein Ausreißer<sup>1</sup> ist eine Beobachtung, die so sehr von den anderen Beobachtungen abweicht, dass sie den Verdacht aufkommen lässt, dass sie durch einen anderen Mechanismus hervorgerufen wurde.*“ In den meisten Anwendungen werden die Daten durch einen oder mehrere generierende Prozesse erstellt, die entweder die Aktivität im System oder die über Entitäten gesammelten Beobachtungen widerspiegeln können. Wenn sich dieser Prozess ungewöhnlich verhält, werden Anomalien erzeugt. Daher enthält eine Anomalie häufig nützliche Informationen zu abnormalen Merkmalen oder einem anomalen Verhalten des Systems oder der Entitäten, die sich auf den Datengenerierungsprozess auswirken. Das Erkennen derartiger ungewöhnlicher Merkmale liefert daher nützliche anwendungsspezifische Erkenntnisse, wie z.B. die o.g. Erkennung eines Kreditkartenbetruges. [Agg17]

Zur Identifizierung von Anomalien werden häufig auf *Machine Learning* (ML) basierende Verfahren verwendet, da sich diese als sehr praktikabel erwiesen haben [Agg17]. Dabei werden Modelle anhand der Daten eines spezifischen Systems trainiert, sodass das Modell anschließend in der Lage ist, anomale Daten als solche zu erkennen. Im Gegensatz zum bekannten Setup der Klassifizierung, wobei Trainingsdaten zum Trainieren eines Klassifikators verwendet werden und anhand von Testdaten anschließend die Leistung des Modells bewertet wird, sind beim Erkennen von Anomalien mehrere Setups, wie die *Supervised Anomaly Detection*, *Semi-Supervised Anomaly Detection* oder *Unsupervised Anomaly Detection*, möglich [CC19; CBK09].

*Supervised Models* weisen im Gegensatz zu *unsupervised Models* i.d.R. eine besserer Performance auf, da sie anhand von gelabelten Daten trainiert wurden. Das Problem des supervised Ansatzes ist jedoch, dass gelabelte Daten in der Realität selten zur Verfügung stehen. Insbesondere für komplexe Anwendungsszenarien mit multivariaten Daten sind die Datensätze selten gelabelt. Darüber hinaus können supervised Modelle meist nur die Anomalien als solche erkennen, die sie während des Trainings gesehen haben. Dies setzt also voraus, dass der Trainingsdatensatz möglichst alle Arten von Anomalien für einen Anwendungsfall beinhaltet, was in der Praxis ebenfalls unrealistisch ist. [CC19]

Aufgrund dieser Eigenschaften werden in der Praxis häufig unsupervised oder semi-supervised Verfahren verwendet, wobei sich *Autoencoder* (AE) als sehr praktikabel erwiesen haben. Für komplexe Anwendungen, wie der Anomalieerkennung in multivariaten Zeitreihen oder multidimensionalen Datenströmen, können *Autoencoder* z.B. verwendet werden, um einen Anomaliescore vorherzusagen. [Agg17]

Auf *Deep Learning* basierende Algorithmen zur Anomalieerkennung haben in den letzten Jahren immer mehr an Popularität gewonnen und werden daher heute für eine Vielzahl von Anwendungsbereichen eingesetzt. Verschiedene Studien haben gezeigt, dass für viele Anwendungsbereiche *Deep Learning*-Modelle traditionelle Methoden des ML weit übertreffen. Ein Grund für den Bedarf an *Deep Anomaly Detection* (DAD) Modellen liegt dabei an der immer weiter steigenden Komplexität der Systeme und der daher wachsenden Datenmengen. [CC19]

---

<sup>1</sup> Neben Anomalie wird in der statistischen Literatur häufig auch der Begriff Ausreißer verwendet.

Komplexe Systeme sind in der modernen Industrie allgegenwärtig. Die Überwachung des Verhaltens dieser Systeme generiert eine erhebliche Menge multivariater Zeitreihendaten, beispielsweise die Messwerte der in einem Kraftwerk verteilten vernetzten Sensoren (z.B. Temperatur oder Druck) oder der angeschlossenen Komponenten (z.B. CPU-Auslastung oder Netzwerkdaten) in einem IT-System. Eine wichtige Aufgabe bei dem Betrieb dieser Systeme besteht darin, in den Zeitreihen Anomalien zu erkennen, damit die Operatoren weitere Maßnahmen zur Behebung der zugrunde liegenden Probleme ergreifen können. Die Erkennung von Anomalien in multivariaten Zeitreihen stellt jedoch ein komplexes Problem dar, da nicht nur die zeitliche Abhängigkeit in jeder Zeitreihe erfasst werden muss, sondern auch die Wechselbeziehungen zwischen verschiedenen Paaren von Zeitreihen codiert werden müssen. In Abbildung 1.1 ist ein Beispiel für die Anomalieerkennung in multivariaten Zeitreihen für ein Kraftwerk dargestellt. Im unteren Bereich (a) der Abbildung ist die Erzeugung multivariater Zeitreihen durch die Verbindung eines Kraftwerks mit IT-Systemen sowie die Identifizierung von Anomalien in diesen Zeitreihen dargestellt. Im oberen Bereich (b) der Abbildung sind beispielhaft die sog. *System Signature Matrices*  $M_{normal}$  und  $M_{abnormal}$  zur Visualisierung des Zustandes der Zeitreihen dargestellt. [Zha+18]

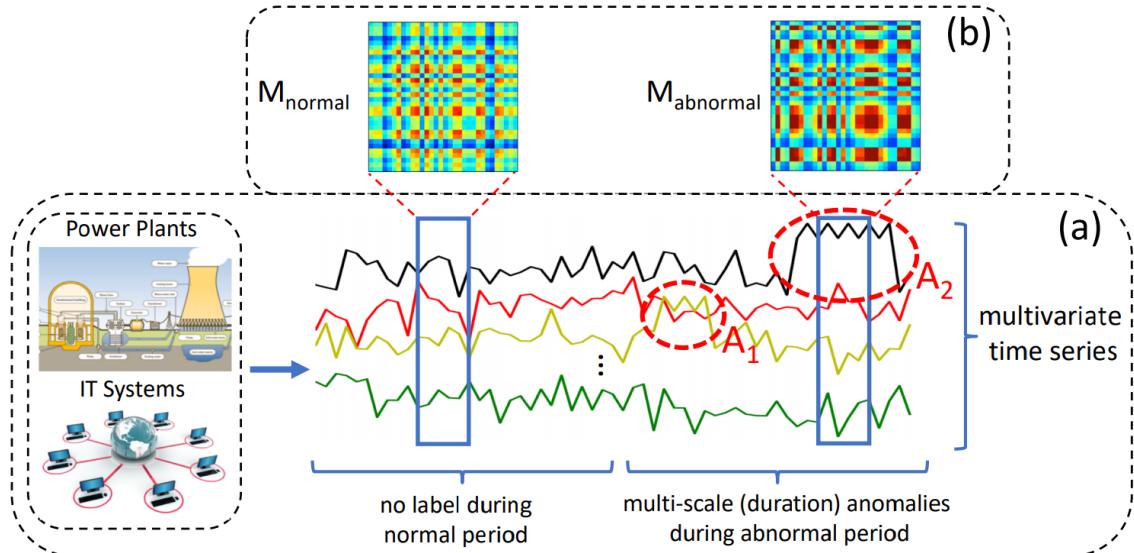


Abbildung 1.1.: (a) *Unsupervised Anomalieerkennung in multivariaten Zeitreihen für Smart Grids.* (b) *Unterschiedliche System Signature Matrices für normalen und abnormalen Perioden.* [Zha+18]

Herkömmliche Anomalieerkennungs-Algorithmen, die auf Distanz- oder Dichtemetriken basieren, können keine periodischen und saisonabhängigen Punkt- und Kontextanomalien in Zeitreihen erkennen, die jedoch häufig in multivariaten Zeitreihen auftreten [Mun+19]. Deshalb werden für die Erkennung von Anomalien in multivariaten Zeitreihen häufig DAD-Modelle, wie *Deep AE*, verwendet [CC19].

## 1.1. Problemstellung

Bei dem operativen Einsatz von *Deep AE* in der Anomalieerkennung entstehen unterschiedliche Herausforderungen, die es aus ML-Sicht zu behandeln gilt. Eine dieser Herausforderungen ist das Pro-

blem sich ändernder Verteilungen in den Daten, dem sog. *Concept Drift*, und ein daher benötigtes Re-Training der Modelle.

Daten, die durch einen generierenden Prozess erzeugt werden, können sich im Laufe der Zeit ändern. In einem SCADA System können z.B. zusätzliche Systeme/Komponenten angeschlossen werden oder sich das zugrundeliegende Verhalten bei der Benutzung des Systems ändern. Dies kann bei ML-Modellen, die eine statistische Beziehung zwischen *Features* und *Labels* lernen, zu einer sich verschlechternden Performance führen. Dieses Problem der sich ändernden zugrunde liegenden Beziehungen in den Daten wird im Kontext des MLs als *Concept Drift* bezeichnet. [Tsy04]

Die von den Modellen gelernte Beziehung zwischen den *Features*  $x$  und den *Labels*  $y$  entspricht der Approximation einer Funktion  $y = f(x)$ . Häufig wird davon ausgegangen, dass dieses *Mapping* statisch ist. Dies bedeutet, dass das aus den historischen Daten gelernte *Mapping* für neue Daten in Zukunft genauso gültig ist und sich die Beziehungen zwischen *Features* und *Label* nicht ändern. Für die meisten praktischen Anwendungen ist dies jedoch nicht der Fall, da sich im Laufe der Zeit nahezu immer Änderungen in den Daten ergeben. [Gam+14]

Das Problem des *Concept Drifts* spielt für Modelle zur Identifizierung von Anomalien eine besondere Rolle. Ändern sich die zugrunde liegenden Beziehungen in den Daten über die Zeit, ist das Modell nicht mehr in der Lage, Anomalien als solche zu erkennen [CW17]. Aus einer anwendungsspezifischen Sicht heißt dies, dass in den Daten z.B. ein Angriff auf ein Netzwerk oder ein Kreditkartenbetrug erkannt wird, obwohl es sich um ein normales Ereignis in den Daten handelt. Konkret bedeutet dies, dass durch den *Concept Drift* die *False Positive Rate*, also die Anzahl der fälschlicherweise als Anomalie erkannten Events, des Modells steigt [Das+19].

Die einfachste und gängigste Methode, um ein Modell vor einem *Concept Drift* zu schützen, ist ein periodisches Re-Training des Modells mit neuen Daten [Mor+13]. Je nachdem wie schnell sich die zugrunde liegenden Beziehungen in den Daten ändern, ist ein Re-Training des Modells früher oder später notwendig. Verschiedene Methoden können dabei helfen, den optimalen Zeitpunkt für ein Re-Training zu bestimmen. In der Praxis werden hierfür die Abstände zwischen der Verteilungen der Trainings- und operativen Daten mittels Metriken, wie z.B. der *Kullback-Leibler-Divergenz*, bestimmt, um zu messen, ob sich die Daten im operativen Betrieb eines Modells stark von den Trainingsdaten unterscheiden. Ist die Distanz zwischen den Trainings- und aktuellen Daten zu groß, ist ein Re-Training notwendig. [Gam+14]

Dies führt jedoch zu dem Problem, dass zu einem Zeitpunkt, zu dem ein Re-Training notwendig ist, evtl. nicht genügend neue (gelabelte) Daten vorliegen. Gerade für Modelle, die komplexe nichtlineare Zusammenhänge aus den Daten lernen, stellt dies ein Problem dar, da diese i.d.R. mehr Daten als ein klassisches ML-Modell benötigen [Mit19]. Im Kontext des ML wird dieses Problem als *k-shot* oder *few-shot Learning* bezeichnet. *Few-shot Learning* ist die Fähigkeit eines Modells, aus wenigen Samples lernen zu können [SSZ17]. Der Mensch ist z.B. sehr gut dazu in der Lage, anhand weniger Beispiele neue Dinge zu lernen. So sind schon kleine Kinder dazu in der Lage, eine Giraffe in der realen Welt zu erkennen, nachdem sie nur ein einzelnes Bild gesehen haben. Für ML-Modelle gilt dies jedoch nicht. Angenommen, ein Modell  $M$  wurde für eine Aufgabe  $A$  trainiert, dann ist dieses Modell für die Aufgabe  $A$  ggf. sehr gut geeignet und unter Umständen sogar besser als der Mensch. Dabei kann die Aufgabe  $A$  z.B. das Erkennen von Katzen in einem Bild sein. Nun gibt es eine zweite Aufgabe  $B$ , z.B. das Erkennen von Hunden in einem Bild, welche stark mit der Aufgabe  $A$  in Beziehung steht. Würde das Modell  $M$  nun für die Aufgabe  $B$  verwendet werden, würde es keine gute Performance aufweisen, da es nur für die Aufgabe  $A$  trainiert wurde. [Rav18]

*Meta-Learning* ist ein Ansatz, um das *few-shot Learning*s Problem zu lösen. *Meta-Learning*, auch als „Lernen zu Lernen“ bezeichnet, zielt darauf ab, Modelle zu entwerfen, mit denen anhand einiger weniger Trainingsbeispiele neue Fähigkeiten erlernt und Modelle schnell an neue Umgebungen angepasst werden können [Van19]. Ein *Meta-Learning*-Algorithmus nimmt eine Verteilung von Aufgaben, wobei jede Aufgabe ein Lernproblem darstellt, und erstellt anhand dieser Verteilung von Aufgaben einen „Quick Learner“ [NAS18]. Der *Model Agnostic Meta-Learning* (MAML) Algorithmus ist ein kürzlich von Finn, Abbeel und Levine eingeführter und in der Wissenschaft häufig verwendeter *Meta-Learning*-Algorithmus. MAML agiert dabei als Meta-Modell, welches eine Gewichtsinitialisierung  $\theta$  für ein zugrunde liegendes Modell lernt. Anhand dieser Gewichtsinitialisierung  $\theta$  kann das zugrunde liegende Modell schnell und mit nur wenigen *Samples* an neue Aufgaben adaptiert werden [FAL17].

## 1.2. Eigener Ansatz

Zuvor wurde bereits das *k-shot Learning* Problem sowie ein möglicher Lösungsansatz mittels *Meta-Learning* beschrieben. Im Rahmen der Masterarbeit soll weiter untersucht werden, ob und wie mittels *Meta-Learning*-Algorithmen die Auswirkungen des *Concept Drift* auf *Autoencoder*, die für die Anomalieerkennung in multivariaten Zeitreihen verwendet werden, verhindert bzw. minimiert werden können. Im Rahmen der Masterarbeit sollen daher die folgenden Forschungsfragen bearbeitet werden:

**RQ1:** Können die Auswirkungen des *Concept Drifts* auf die Performance von Deep *Autoencodern* zur Anomalieerkennung in multivariaten Zeitreihen mittels *Meta-Learning*-Algorithmen minimiert bzw. verhindert werden?

**RQ2:** Welche Auswirkung hat das *Meta-Learning* auf die Performance des *Autoencoders*?

Zur Bearbeitung dieser Forschungsfragen wurde ein grobes Vorgehen entwickelt:

**1. Benchmarking von Autoencodern auf realen Daten:** Zunächst soll ein *Autoencoder* auf einem realen Datensatz trainiert werden, um die Performance des AE mit anderen Anomalieerkennungsmodellen, die in der Literatur häufig verwendet werden, zu vergleichen. Hierfür ist, neben der Auswahl der zu vergleichenden Modelle, ein geeigneter Datensatz auszuwählen, der im Kontext der Anomalieerkennung in multivariaten Zeitreihen oft in der Literatur verwendet wird. Für die Auswahl des Datensatzes ist vorab eine Recherche notwendig. Da die spätere Evaluation des *Meta-Learning*-Ansatzes auf multivariaten Zeitreihendaten durchgeführt werden soll, ist ein Datensatz mit Zeitreihen auszuwählen. Die Modelle werden dann anhand gängiger Metriken, wie der *Accuracy*, *False Positive Rate* oder *Specificity*, auf dem Datensatz miteinander verglichen. Dieser erste Vergleich ermöglicht eine Bewertung, inwiefern die gewählte AE Topologie zur Erkennung von Anomalien in multivariaten Zeitreihen geeignet ist. Die Auswirkungen des *Concept Drifts* auf den AE werden in diesem Schritt noch nicht untersucht. Das Ergebnis dieses Schrittes ist die Bewertung der Eignung eines AE zur Anomalieerkennung in multivariaten Zeitreihen sowie die Auswahl einer geeigneten AE Topologie.

**2. Evaluation des *Meta-Learning*-Ansatzes auf synthetischen Daten:** Im zweiten Schritt sollen die Erkenntnisse bzgl. der Topologie von AE zur Anomalieerkennung genutzt werden, um einen *Meta-Learning*-Ansatz auf synthetischen Daten zu evaluieren. Die synthetischen Daten werden dabei von einem *Data Generator* erzeugt, der zunächst zu implementieren ist. Die synthetischen Daten wer-

den Zeitreihendaten, ähnlich wie in dem zuvor verwendeten Datensatz, sein. Die Verwendung eines *Data Generators* ermöglicht die vollkommen flexible Generierung aller Arten von Anomalien sowie eine flexible Steuerung der zeitlichen Abstände zwischen den Anomalien. Um die Auswirkungen des *Meta-Learnings* zu untersuchen, werden anhand der synthetisch erzeugten Daten zwei Modelle trainiert. Modell  $M_1$  wird dabei mittels eines *Meta-Learning*-Algorithmus trainiert. Modell  $M_2$  wird hingegen klassisch mittels eines gängigen Lernalgorithmus trainiert. Anschließend werden beide Modelle auf verschiedenen Datensätzen mit Anomalien und *Concept Drift* hinsichtlich ihrer Performance evaluiert. Beide Modelle werden darüber hinaus mit nur wenigen *Samples* an den Datensatz mit *Concept Drift* adaptiert. Sollte Modell  $M_1$ , welches mittels eines *Meta-Learning*-Algorithmus trainiert wurde, besser an den neuen Datensatz adaptiert werden können, ist dies ein Indiz dafür, dass das *Meta-Learning* einen positiven Effekt auf die Adaptierbarkeit des Modells an neue Aufgaben aufweist.

### 1.3. Verwandte Arbeiten

In der Literatur gibt es eine Vielzahl an Büchern und Papern, die sich entweder dem Thema *Meta-Learning*, Anomalieerkennung oder Autoencoder widmen. Während der Literaturrecherche konnte jedoch nur wenig Literatur gefunden werden, welche sich mit dem Thema *Meta-Learning* für *Autoencoder* beschäftigt.

Das mit am häufigsten zitierte Paper für MAML ist das Paper [FAL17]. Hierin stellen Finn, Abbeel und Levine den MAML-Algorithmus als Meta-Learning-Algorithmus vor. Ein weiteres wichtiges Paper ist [NAS18]. Hierin stellen die Autoren Nichol, Achiam und Schulman einen *First-Order Meta-Learning*-Algorithmus vor, der sich stark an MAML orientiert.

Bekannte Literatur zum Thema Concept Drift sind u.a. die Paper [CBK09], [Gam+14] oder [WSK10]. In jedem der Paper stellen die Autoren jeweils das Problem des Concept Drifts vor und beschreiben entweder Methoden zur Identifizierung oder Lösung des Problems.

Ein bekanntes Werk für die Anomalieerkennung mittels statistischer und ML basierender Methoden ist das Buch [AS17], welches einen umfassenden Blick auf die Anomalieerkennung wirft. Darüber hinaus gibt es eine Vielzahl von Literatur, welche sich mit der anwendungsspezifischen Anomalieerkennung beschäftigt. Als Beispiel seien hier die Paper von Andrysiak, Saganowski und Kiedrowski oder Borghesi et al. genannt. In [ASK17] stellen die Autoren einen statischen Ansatz vor, mittels dessen Anomalien in einer Smart Meter Infrastruktur identifiziert werden können. In [Bor+18] hingegen stellen die Autoren einen Ansatz vor, um mittels Autoencoder Anomalien in einem High Performance Computing System zu erkennen.

Besonders relevant für die Masterarbeit sind die Werke [Mun+19], [Zha+18] und [Guo+18]. In [Mun+19] stellen Munir et al. einen Ansatz zur Erkennung von Anomalien in Zeitreihen mittels *Deep Learning* Methoden vor. In [Zha+18] stellen Zhang et al. ein Vorgehen zur Erkennung von Anomalien in multivariaten Zeitreihen vor. Der Fokus der Arbeit liegt dabei neben der Erkennung einer Anomalie auch auf der *Root Cause Detection*, also der Analyse der Ursache für eine Anomalie. Als Anwendungskontext werden multivariate Zeitreihen eines Kraftwerks verwendet. In [Guo+18] stellen Guo et al. ein auf *Gated Recurrent Units Autoencoder* (GRU-AE) basierendes Modell zur Erkennung von Anomalien in Zeitreihen vor.

Einen umfangreichen Blick auf Autoencoder wirft das Werk [Bal12]. Darin beschreibt Baldi die grundlegenden Konzepte und Einsatzmöglichkeiten von Autoencodern. Darüber hinaus gibt es viele Paper, die sich mit der Anwendung von AE auf spezielle Probleme beschäftigen.

#### **1.4. Aufbau der Arbeit**

Die vorliegende Arbeit ist folgendermaßen gegliedert: In Kapitel 2 werden die Grundlagen der verwendeten Methoden und Algorithmen beschrieben. Im darauf folgenden Kapitel 3 wird die Konzeptionierung der vorliegenden Arbeit beschrieben. Dafür werden anhand der Forschungsfragen entsprechende untersuchbare Thesen und Hypothesen aufgestellt, sowie die Planung der Vorstudie und des Experiments anhand statistischer Versuchspläne beschrieben. In Kapitel 4 werden die Durchführung und Erkenntnisse der Vorstudie dokumentiert. In dem darauf folgenden Kapitel 5 wird die Implementierung der notwendigen Komponenten sowie ML-Modelle beschrieben. Anschließend wird in Kapitel 6 das Experiment zur Untersuchung der Hypothesen beschrieben und evaluiert. Abgeschlossen wird die vorliegende Arbeit mit einem Fazit und Ausblick in Kapitel 7.

## 2. Grundlagen

In den folgenden Abschnitten werden die für das Verständnis der vorliegenden Arbeit notwendigen Grundlagen vermittelt. Hierzu werden zu Beginn in Abschnitt 2.1 die Grundlagen des *Machine Learnings* erläutert. Im nächsten Abschnitt 2.2 werden die Grundlagen von Autoencodern vermittelt. Anschließend wird in Abschnitt 2.3 der Begriff „Anomalie“ definiert sowie die Grundlagen der Anomalieerkennung dargestellt. Im darauf folgenden Abschnitt 2.4 wird *Meta Learning* als Begriff eingeführt und die Grundlagen dargestellt. Abschließend werden in Abschnitt 2.5 die grundlegenden Konzepte zur Evaluation von Modellen im Kontext der Anomalieerkennung erläutert.

### 2.1. Machine Learning

*Machine Learning* (ML) hat in den letzten Jahren in vielen Anwendungsbereichen große Fortschritte erzielt. Auch bei der Erkennung von Anomalien in verschiedenen Domänen werden heute unterschiedliche Modelle des *Machine Learnings* verwendet [Ass17]. In den folgenden Unterabschnitten soll daher zunächst der Begriff des ML definiert werden, um anschließend alle relevanten Grundlagen des ML im Kontext der vorliegenden Arbeit zu vermitteln. Hierzu wird in Unterabschnitt 2.1.1 zunächst der Begriff des Lernens im Kontext des ML eingeführt. Anschließend werden in Unterabschnitt 2.1.2 die Grundlagen künstlicher neuronaler Netze dargelegt. In Unterabschnitt 2.1.3 wird dann der Begriff „Deep Learning“ eingeführt. Abschließend wird in Unterabschnitt 2.1.4 das Problem des *Concept Drifts* im Kontext der Anomalieerkennung erläutert.

ML kann als eine Reihe von Methoden definiert werden, die automatisch Muster in Daten erkennen und diese erkannten Muster dann verwenden, um zukünftige Daten vorherzusagen oder andere Arten von Entscheidungen unter Unsicherheit zu treffen [Mur12]. In [Sha14] definiert Shai Shalev-Shwartz ML hingegen als die Aufgabe Computer so zu programmieren, dass sie aus den ihnen zur Verfügung stehenden Informationen lernen können. Abstrakt betrachtet ist Lernen dabei der Prozess der Umwandlung von Erfahrung in Wissen. Der Input für einen Lernalgorithmus sind Trainingsdaten, die Erfahrung repräsentieren. Der Output ist eine gewisse Expertise, die anhand der gelernten Muster aus dem Input entsteht.

#### 2.1.1. Lernen

Im Allgemeinen ist ein Lernalgorithmus ein Algorithmus, welcher in der Lage ist aus Daten zu lernen. Der Begriff „Lernen“ wird von Mitchell in [Mit97] wie folgt definiert: „*Ein Computerprogramm lernt aus Erfahrungen E in Bezug auf eine bestimmte Klasse von Aufgaben T und Leistungskennzahlen P, wenn sich eine Leistung der Aufgaben T, gemessen an P, mit der Erfahrung E verbessert.*“

#### Die Aufgabe $T$

Durch ML können Aufgaben gelöst werden, die zu komplex sind, um sie mit von Menschen programmierte Software zu lösen. Dabei ist die Aufgabe selbst nicht das Lernen, sondern Lernen ist das Mittel, um die Fähigkeit zu erlangen, die Aufgabe auszuführen. Wenn ein Roboter beispielsweise in der Lage sein soll zu Laufen, dann ist Laufen die Aufgabe. Der Roboter kann nun so programmiert

werden, dass er das Laufen lernt oder es kann versucht werden, das Programm zum Laufen direkt zu entwickeln. [GBC16]

Aufgaben des ML werden in der Regel dahingehend beschrieben, wie der Lernalgorithmus ein *Sample* verarbeiten soll. Ein *Sample* ist eine Sammlung von *Features*, die quantitativ von einem Objekt oder Ereignis gemessen wurden. Typischerweise wird ein *Sample* als Vektor  $\mathbf{x} \in \mathbb{R}^n$  beschrieben, wobei jedes Element  $x_i$  des Vektors ein Feature darstellt [GBC16]. Im Rahmen des ML kann zwischen verschiedenen Aufgabenklassen unterschieden werden. Gängige Aufgabenklassen sind dabei die Klassifikation, Regression, Anomalieerkennung oder das Entfernen von Rauschen (*Denoising*):

- **Klassifikation:** Bei dieser Aufgabenklasse wird ein Algorithmus dazu verwendet, um zu bestimmen, zu welcher der  $k$  Klassen ein *Sample*  $\mathbf{x}$  gehört. Um diese Aufgabe zu lösen, lernt der Algorithmus eine Funktion  $f : \mathbb{R}^n \rightarrow \{1, \dots, k\}$  zu approximieren. Wenn  $y = f(\mathbf{x})$  ist, dann ordnet das Modell eine durch den Vektor  $\mathbf{x}$  gegebene Eingabe einer durch den einen numerischen Code  $y$  identifizierten Kategorie zu. Darüber hinaus gibt es weitere Varianten der Klassifikation, bei denen  $f$  z.B. eine Wahrscheinlichkeitsverteilung über Klassen ausgibt. [GBC16]
- **Regression:** Hierbei wird ein Lernalgorithmus verwendet, um einen numerischen Wert anhand einer gegebenen Eingabe vorherzusagen. Um dies zu lösen, lernt der Algorithmus eine Funktion  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  zu approximieren. Diese Art der Aufgabe ist sehr ähnlich zur Klassifikation, mit dem Unterschied, dass sich die Art des *Outputs* unterscheidet. [GBC16]
- **Anomalieerkennung:** Bei dieser Art von Aufgabenklasse wird ein Algorithmus verwendet, um eine Reihe von Ereignissen oder Objekten zu durchsuchen und einige davon ggf. als ungewöhnlich oder untypisch zu markieren. Ein gängiges Beispiel der Anomalieerkennung ist die Erkennung von Kreditkartenbetrügen, wobei ein Algorithmus verwendet wird, um den Diebstahl oder die falsche Verwendung von Kreditkarten zu identifizieren. [GBC16]
- **Denoising:** Beim *Denoising* (oder auch Entrauschen) wird ein ML Algorithmus verwendet, um anhand eines gegebenen korrupten *Samples*  $\tilde{\mathbf{x}} \in \mathbb{R}^n$ , welches durch einen unbekannten korrupten Prozess erzeugt wurde, ein sauberes *Sample*  $\mathbf{x} \in \mathbb{R}^n$  zu erzeugen. Der Algorithmus muss dafür das saubere *Sample*  $\mathbf{x}$  anhand des korrupten *Samples*  $\tilde{\mathbf{x}}$  erzeugen, bzw. die bedingte Wahrscheinlichkeitsverteilung  $p(\mathbf{x}|\tilde{\mathbf{x}})$  vorhersagen. [GBC16]

Neben den o.g. Aufgabenklassen gibt es im ML noch weitere Aufgabenklassen, wie z.B. die Transkription von Texten oder Videos, die Übersetzung von Texten, die Imputation fehlender Werte oder die Schätzung von Dichten, die im Kontext der vorliegenden Arbeit jedoch weniger von Bedeutung sind [GBC16].

## Die Leistungskennzahl $P$

Um die Fähigkeiten eines ML Algorithmus zu bewerten, müssen quantitative Metriken  $P$  definiert werden, welche üblicherweise auf die vom Algorithmus auszuführende Aufgabe  $T$  abgestimmt sind. Für Aufgaben wie der Klassifikation werden beispielsweise Metriken wie die *Accuracy* oder *Error Rate* verwendet. Um zu bestimmen, wie gut die Performance eines Algorithmus tatsächlich ist, wird häufig ein sog. *Test Set* verwendet. Ein *Test Set* ist eine Sammlung von *Samples*  $\mathbf{x}$  und Features  $y$ , welche nicht während des Trainings von dem Modell verwendet wurden. Für die Aufgabenklasse der Regression werden hingegen Metriken wie der (*Root*) *Mean Squared Error* oder *Mean Absolute Error* verwendet. Die Wahl einer geeigneten Metrik scheint einfach und objektiv zu sein, in der Praxis ist

es jedoch of schwierig, ein Leistungsmaß zu wählen, das dem gewünschten Verhalten des Systems entspricht. [GBC16]

## Die Erfahrung $E$

Die Erfahrung  $E$ , die einem ML Algorithmus während des Trainings zur Verfügung steht, besteht i.d.R. aus einem Datensatz  $\mathbf{X}$ , wobei  $\mathbf{X}$  wiederum aus einer Sammlung von *Samples*  $\mathbf{x}$  besteht. Eine gängige Art, einen Datensatz zu beschreiben, ist die Verwendung einer Designmatrix, welche in jeder Zeile unterschiedliche *Samples*  $\mathbf{x}$  enthält. Jede Spalte der Designmatrix entspricht dabei einem *Feature*. So kann ein Datensatz als Matrix  $\mathbf{X} \in \mathbb{R}^{m \times n}$  dargestellt werden, wobei  $m$  die Anzahl der unterschiedlichen *Samples* und  $n$  die Anzahl der unterschiedlichen *Features* darstellt. Jede Zeile der Matrix  $\mathbf{X}$  besteht aus einem Vektor, wobei alle Vektoren über die Matrix verteilt die gleiche Länge aufweisen. [GBC16]

Die Art, wie ML Algorithmen lernen, kann in *Unsupervised Learning*, *Supervised Learning*, *Semi-Supervised Learning* und *Reinfocement Learning* unterteilt werden, entsprechend der Art, welche Erfahrungen sie während des Trainings verwendet haben [GBC16]:

- ***Unsupervised Learning*:** Bei dieser Art des Lernens werden ML Algorithmen verwendet, um hilfreiche Eigenschaften in Datensätzen zu erkennen. Dazu bekommen sie während des Trainings den Datensatz  $\mathbf{X}$  präsentiert, in dem die Algorithmen dann z.B. die Verteilungsfunktion der zugrunde liegenden Daten lernen. Das Ziel des *unsupervised Learning* ist es, die zugrunde liegende Struktur oder Verteilung der Daten zu modellieren, um mehr über die Daten zu erfahren. Verfahren aus dieser Klasse des Lernens werden als *unsupervised* bezeichnet, da es im Gegensatz zum *supervised Learning* keine richtigen Antworten und keinen Lehrer gibt. Die Algorithmen sind bei der Identifizierung von Strukturen durch ihre eigenen Möglichkeiten begrenzt. Ein klassisches Beispiel für einen Algorithmus dieser Art ist *k-means* zum *Clustering* von Daten. Dabei wird ein Datensatz in  $k$  möglichst heterogene Cluster unterteilt, wobei die einzelnen Punkte in einem Cluster möglichst homogen sind. [GBC16]
- ***Supervised Learning*:** Beim *supervised Learning* wird einem Lernalgorithmus während des Trainings ein Datensatz  $\mathbf{X} \in \mathbb{R}^{m \times n}$  sowie ein Vektor dazugehöriger *Label*  $\mathbf{y} \in \mathbb{R}^m$  präsentiert. Der Algorithmus versucht anhand der Daten eine Funktion  $f(\mathbf{x}) = y$  zu approximieren. Die vom Algorithmus vorhergesagten Werte werden als  $\hat{y}$  gekennzeichnet, falls es sich um einen Vektor von Vorhersagen handelt, sonst als  $\hat{y}$ . Die tatsächlich vom Algorithmus gelernte Funktion wird mit  $f^*(\mathbf{x}) = \hat{y}$  beschrieben. Verfahren dieser Art des Lernens werden als *supervised* bezeichnet, da der Prozess eines Algorithmus, der aus dem Trainingsdatensatz mit *Labels* lernt, als „Lehrer“ betrachtet werden kann, der den Lernprozess überwacht. Die richtigen Antworten sind (durch die *Label*) bekannt, der Algorithmus berechnet während des Trainings iterativ Vorhersagen und wird bei Fehlern durch den externen „Lehrer“ korrigiert. Der Lernprozess stoppt, wenn der Algorithmus eine akzeptable Leistungskennzahl erreicht hat. [GBC16]
- ***Semi-Supervised Learning*:** Diese Art des Lernens ist eine Mischung aus *unsupervised* und *supervised Learning*, da für das Training sowohl gelabelte als auch ungelabelte Daten verwendet werden. Im Datensatz befinden sich jedoch wesentlich mehr ungelabelte, als gelabelte Daten. Viele der in der Praxis verwendeten Modelle sind *semi-supervised*, da häufig keine *Label* vorhanden sind oder der Prozess zur Erstellung von *Labels* sehr zeitintensiv und teuer ist. [Goe+14; Ruf+19]

- **Reinforcement Learning:** Einige Algorithmen des ML verwenden für das Training keinen Datensatz, sondern stehen während des Trainings mit einer Umwelt in Interaktion. Der Lernalgorithmus ist dabei häufig teil eines Agenten. Dabei gibt es einen *Feedback Loop* zwischen dem Agenten und seiner Umwelt. Der Agent hat das Ziel anhand der Interaktion mit seiner Umwelt und der so erlangten Erfahrung eine vorgegebene Metrik zu maximieren. [Ric18]

Abstrakt betrachtet beinhaltet *unsupervised Learning* das Beobachten mehrere Beispiele eines zufälligen Vektors  $\mathbf{x}$  und den Versuch, die Wahrscheinlichkeitsverteilung  $p(\mathbf{x})$  oder einige interessante Eigenschaften dieser Verteilung zu erlernen. Beim *supervised Learning* hingegen werden mehrere Beispiele eines zufälligen Vektors  $\mathbf{x}$  und eines zugehörigen Labels  $y$  betrachtet und anhand dieser gelernt  $y$  anhand von  $\mathbf{x}$  durch das Schätzen von  $p(y|\mathbf{x})$  vorherzusagen. *Unsupervised* und *supervised Learning* sind keine formal definierten Begriffe. Die Grenzen zwischen den beiden Begriffen sind oft verschwommen, was u.a. daran zu erkennen ist, dass viele Verfahren des ML sowohl für das *unsupervised* als auch für das *supervised Learning* verwendet werden können. [GBC16]

### 2.1.2. Artificial Neural Networks

*Artificial Neural Networks* (ANN) sind informationsverarbeitende Modelle, deren Struktur und Funktionsprinzipien vom Nervensystem und dem Gehirn von Tieren und Menschen inspiriert sind. Sie bestehen aus einer Vielzahl von relativ einfachen Einheiten, den sogenannten Neuronen, die parallel arbeiten. Diese Neuronen kommunizieren, indem sie Informationen in Form von Aktivierungssignalen entlang gerichteter Verbindungen zueinander senden. [Kru+13]

ANN werden aus verschiedenen Gründen in unterschiedlichen Disziplinen untersucht. In der Neurobiologie und Neurophysiologie, aber auch in der Psychologie, interessieren sich Forscher v.a. für die Ähnlichkeit mit biologischen Nervensystemen. In diesen Bereichen werden ANN häufig als Rechenmodelle und Simulationen verwendet, um die Mechanismen von Nerven- und Gehirnfunktionen zu verstehen. Insbesondere in der Informatik, aber auch in anderen Ingenieurswissenschaften, wird versucht, bestimmte kognitive Fähigkeiten des Menschen (wie z.B. die Lernfähigkeit) durch funktionale Elemente des Nervensystems und des Gehirns nachzuahmen. In der Physik werden bestimmte mathematische Modelle durch ANN abgebildet, um physikalische Phänomene zu beschreiben. Aufgrund der unterschiedlichen Interessen und Einflüsse gilt die Forschung an ANN als stark interdisziplinäres Forschungsgebiet mit Einflüssen aus verschiedenen Disziplinen, wie eben der Biologie, Psychologie, Informatik oder der Robotik und Linguistik. [Kru+13]

#### Funktionsweise biologischer Neuronen

Wie eingangs bereits erwähnt, sind ANN ein von der Natur inspiriertes Verfahren. Bevor die abstrakte Funktionsweise der Neuronen erläutert wird, soll für ein besseres Verständnis zunächst kurz die biologische Funktionsweise von Neuronen erläutert werden. Hierzu ist in Abbildung 2.1 die Skizze eines biologischen Neurons inkl. der Beschriftungen der Bestandteile dargestellt. Ein Neuron ist eine elektrische erregbare Zelle, die mit Hilfe von elektrischen und chemischen Signalen Informationen aufnimmt, verarbeitet und weitergibt. Der Aufbau eines Neurons kann in drei Teile unterteilt werden: den Zellkörper samt Zellkern, die Dendriten und das Axon. Der Grundbaustein der Funktion des Nervensystems ist die synaptische Signalübertragung, die teils chemisch, teils elektrisch stattfindet. Sobald ein elektrisches Signal die Synapse erreicht, werden spezielle Übertragungsstoffe, die sog.

Neurotransmitter, von dem Neuron freigesetzt. Diese Neurotransmitter stimulieren dann ein weiteres Neuron und lösen damit eine neue Welle von elektrischen Impulsen aus. Die sich baumartig verzweigenden Dendriten sammeln dabei die Informationsanteile der präsynaptischen Seite und leiten diese an den Zellkern weiter. Nachdem über die Dendriten sowohl aktivierende als auch inhibierende Signale beim Zellkern angekommen sind, werden diese vom Soma kumuliert. Sobald dieser kumulierte Wert einen gewissen Schwellwert überschreitet, löst der Neuronenzellkern seinerseits einen elektrischen Impuls aus, der dann über das Axon zu weiteren Neuronen geleitet wird. [Kri07]

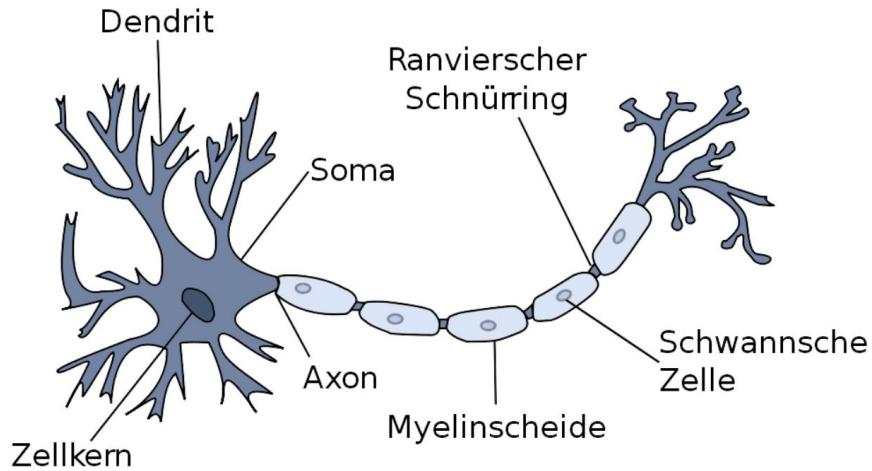


Abbildung 2.1.: Skizze eines biologischen Neurons mit Beschriftungen der Bestandteile  
[Kri07]

### Funktionsweise eines Perzeptron

In Abbildung 2.2 ist die Abstraktion eines biologischen Neurons in Form eines künstlichen Neurons, wie es für ANN verwendet wird, dargestellt. Verschiedene Eingaben  $x_1$  bis  $x_n$  werden jeweils durch Multiplikation mit einem Gewicht  $w_1$  bis  $w_n$  gewichtet. Die gewichteten Eingaben werden anschließend von einer Übertragungsfunktion summiert [Kru+13]. Mathematisch entspricht dies der Funktion:

$$z = \sum_{i=1}^n x_i w_i \quad (2.1)$$

Dabei ist  $n$  die Anzahl der Eingaben,  $x_i$  die jeweilige Eingabe  $i$  und  $w_i$  das Gewicht der  $i$ -ten Eingabe. Sowohl die Eingaben  $\mathbf{x} = (x_1, \dots, x_n)$ , als auch die Gewichte  $\mathbf{w} = (w_1, \dots, w_n)$  eines Neurons, können als Vektoren dargestellt werden. Entsprechend kann auch ihre Multiplikation als Multiplikation zweier Vektoren beschrieben werden:  $z = \mathbf{w}\mathbf{x}$ . Der aus der Addition resultierende Wert  $z$  wird anschließend weiter in einer Aktivierungsfunktion  $\varphi$  verarbeitet [Kri07]:

$$a_j = \varphi(z) \quad (2.2)$$

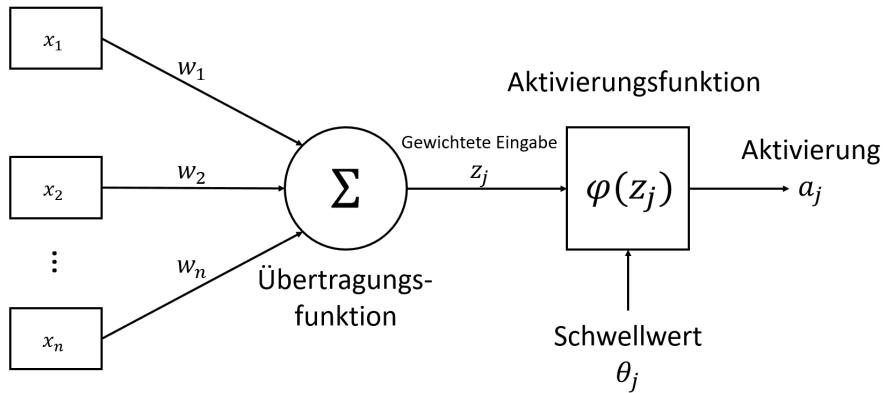


Abbildung 2.2.: Darstellung eines künstlichen Neurons mit seinen Elementen (eigene Darstellung nach [Wik19])

Die Ausgabe  $a_j$  der Aktivierungsfunktion  $\varphi$  kann anschließend wieder als gewichtete Eingabe für weitere Neuronen dienen.

Mittels eines einzelnen Perzeptrons können einfache logische Funktionen, wie eine einfache Schwellwertfunktion, dargestellt werden. Die Funktion ist dabei wie folgt definiert, wobei  $\theta$  einen vorab gewählten Schwellwert darstellt:

$$y = \begin{cases} 1 & \sum_{i=1}^n w_i x_i \geq \theta \\ 0 & \text{sonst} \end{cases} \quad (2.3)$$

Anhand dieser einfachen Schwellwertfunktion können verschiedene logische Funktionen, wie die Konjunktion, Implikation oder komplexere logische Terme, wie z.B.  $(x_1 \wedge \neg x_2) \vee (x_1 \wedge x_3) \vee (\neg x_2 \wedge x_3)$ , abgebildet werden. Ein Beispiel für den gerade genannten Term inkl. einer Wahrheitstwerttabelle ist in Abbildung 2.3 dargestellt. Das Neuron hat dabei drei Eingaben  $x_1, x_2$  und  $x_3$  mit den Gewichten  $w_1 = 2, w_2 = -2$  und  $w_3 = 2$ . Der Schwellwert  $\theta$  des Neurons beträgt 1. Die Eingaben des Neurons werden entsprechend mit ihren Gewichten multipliziert. Sofern das Ergebnis der Multiplikation den Schwellwert übersteigt, hat das Neuron den Output 1, sonst 0. Auf der rechten Seite der Abbildung ist die Wahrheitstwerttabelle mit den entsprechenden Ausgaben des Neurons dargestellt.

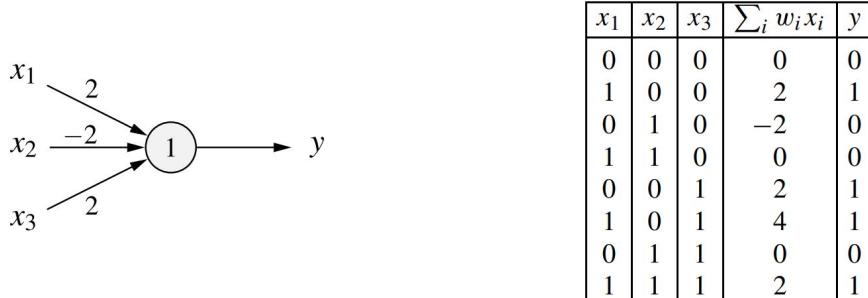


Abbildung 2.3.: Perzepron für den log. Term  $(x_1 \wedge \neg x_2) \vee (x_1 \wedge x_3) \vee (\neg x_2 \wedge x_3)$  [Kru+13]

Das zuvor genannte Beispiel des logischen Terms könnte zu der Annahme führen, dass ein einzelnes Perzeptron mit einer simplen Schwellwertfunktion ein ziemlich leistungsfähiges Modell ist. Leider sind einstufige Logikbausteine, wie ein einzelnes Perzeptron, in ihrer Ausdrucks- und Rechenleistung jedoch stark limitiert. Einzelne Perzeptrons sind nur in der Lage, Funktionen darzustellen, die linear trennbar sind, d.h. Funktionen, die durch eine Linie oder Ebene dargestellt werden können [MP69]. In Abbildung 2.4 ist eine Wahrheitstabelle der Exklusiv-Nicht-Oder-Funktion (XNOR) dargestellt. Die Ausgabe der Funktion nimmt nur den Wert 1 an, falls beide Eingaben  $x_1$  und  $x_2$  jeweils den Wert 1 oder 0 annehmen, entsprechend der Funktion  $y = \overline{x_1 \vee x_2}$ . Auf der rechten Seite der Abbildung ist eine grafische Darstellung der Funktion dargestellt. Es ist zu sehen, dass es keine Linie oder lineare Ebene gibt, mit der es möglich ist, die weißen und schwarzen Punkte voneinander zu trennen.

$x_1$	$x_2$	$y$
0	0	1
1	0	0
0	1	0
1	1	1

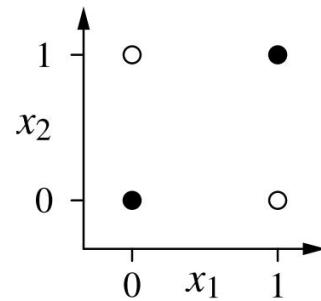


Abbildung 2.4.: *XNOR-Funktion: Es gibt keine lineare Sparierbarkeit der weißen und schwarzen Punkte [Kru+13]*

### Multi-Layer Perceptron

Die gerade genannte Limitation eines einzelnen Neurons kann jedoch eliminiert werden, indem mehrere Neuronen kombiniert werden. Das resultierende ANN wird dann als *Multi-Layer Perceptron* (MLP) bezeichnet. In Abbildung 2.5 ist die Lösung einer XNOR Funktion mittels mehrerer zu einem MLP kombinierter Neuronen dargestellt. Durch die Kombination mehrerer Neuronen in Schichten können so komplexerer Funktionen, wie die XNOR oder XOR Funktion gelöst werden.

In der Abbildung 2.5 ist ebenfalls das Konzept der sog. Bias-Neuronen in Form eines Neurons mit einer 1 dargestellt, welche benötigt werden um die XNOR Funktion zu lösen. Ein Bias Neuron (manchmal auch *on-Neuron* genannt) ist ein Neuron mit dem konstanten Eingang 1. In der Regel hat jeder Layer (außer des *Input Layers*) ein zusätzliches Bias Neuron. Die Ausgaben des Neurons werden entsprechend durch die Multiplikation mit den jeweiligen Gewichten und der Konstanten 1 berechnet. An dieser Stelle soll nicht tiefer auf die Bedeutung des Bias für ANN eingegangen werden. Als Stichwort zur weiteren Recherche soll jedoch auf das sog. Bias-Varianz-Dilemma hingewiesen werden, welches insbesondere im ML und der Statistik von Bedeutung ist. Eine detaillierte Beschreibung bzgl. des Bias Neurons und des Bias-Varianz-Dilemmas ist in [Mur12] von Murphy zu finden.

In Abbildung 2.6 ist der schematische Aufbau eines MLP dargestellt. Grundsätzlich können drei Arten von Schichten (im Englischen auch *Layer*) unterschieden werden. Durch den sog. *Input Layer* werden die Eingaben  $x$  in das Modell gebracht, die dann anschließend in den sog. *Hidden Layern* verarbeitet werden. Dabei kann es mehr als nur einen *Hidden Layer* geben und auch die Anzahl der

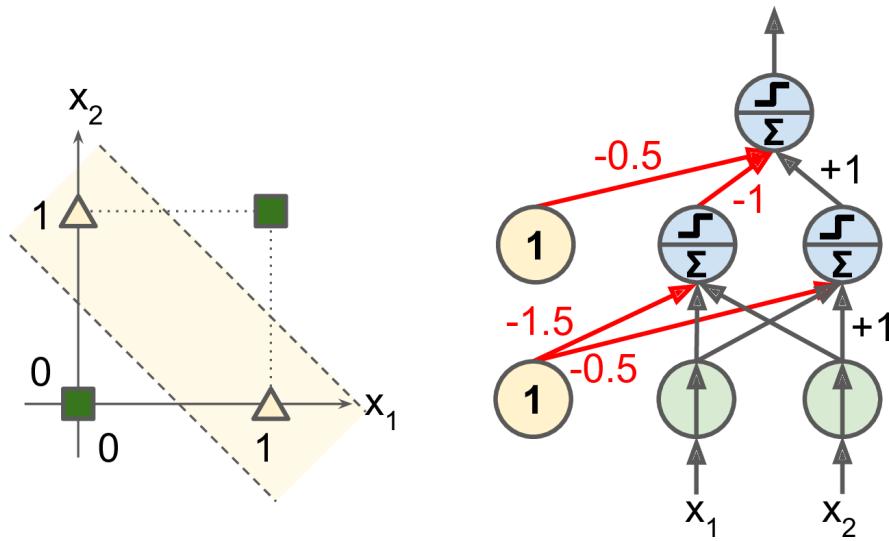


Abbildung 2.5.: Lösen der XNOR Funktion durch Kombination mehrerer Neuronen [Gér18]

Neuronen in einem *Hidden Layer* kann variieren. Im *Output Layer* werden die zuvor verarbeiteten Eingaben als Ergebnis des Modells verfügbar gemacht.

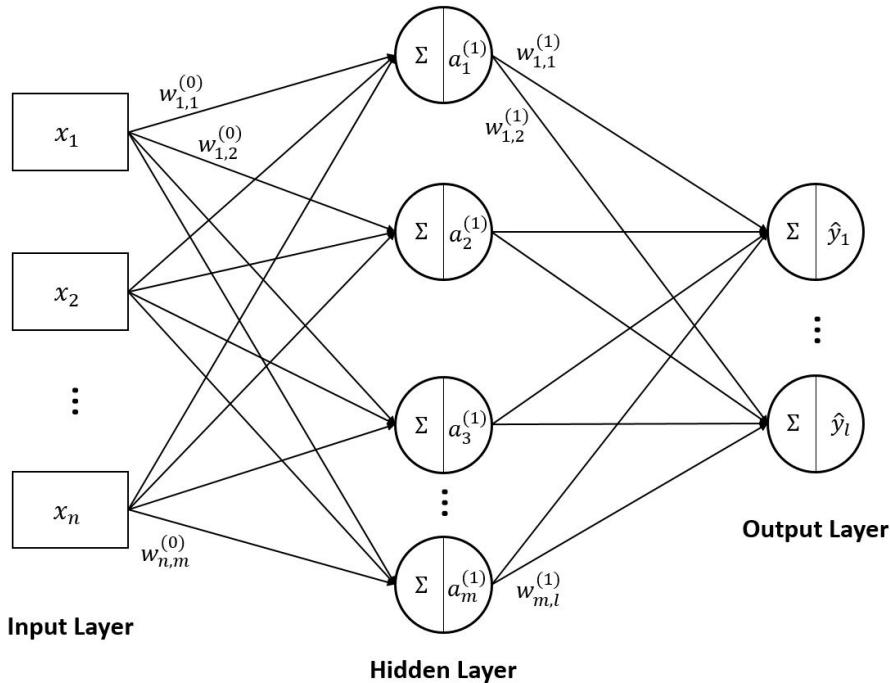


Abbildung 2.6.: Multi-Layer Perceptron mit einem Input, Hidden und Output Layer (eigene Abbildung nach [Haz+11])

Die Informationen durchlaufen das Netz dabei (je nach Darstellung) von links nach rechts, weshalb einfache MLP auch als *Feedforward Networks* bezeichnet werden [GBC16]. Jede Eingabe aus dem *Input Layer* wird zunächst zu jedem Neuron der nachfolgenden Schicht weitergeleitet. Das erste Ge-

wicht der ersten Eingabe aus dem *Input Layer* zum ersten Neuron der nachfolgenden Schicht kann dabei durch  $w_{1,1}^{(0)}$  identifiziert werden. Die Notation ist dabei  $w_{n,m}^{(l)}$ , wobei  $l$  den ausgehenden *Layer* kennzeichnet, beginnend bei 0 für den *Input Layer*,  $n$  kennzeichnet das entsprechende Neuron vom ausgehenden *Layer*, beginnend bei 1 als oberstes Neuron und  $m$  kennzeichnet das Ziel-Neuron im nachfolgenden *Layer*  $l + 1$ . Die schichtweise Verarbeitung der Eingaben in einen *Layer*, multipliziert mit den Gewichten, kann als Matrixoperation ausgedrückt werden. Für den ersten *Layer* im *Hidden Layer*, welcher als Eingabe die reinen *Inputs*  $\mathbf{x} = (x_1, \dots, x_n)$  erhält, sieht dies wie folgt aus:

$$\mathbf{z} = \mathbf{W} \cdot \mathbf{x} + \mathbf{b} \quad (2.4)$$

Dabei ist  $\mathbf{x} \in \mathbb{R}^n$  der Input aus dem *Input Layer*,  $\mathbf{W} \in \mathbb{R}^{k \times n}$  die Gewichtsmatrix des *Layers* und  $\mathbf{b} \in \mathbb{R}^n$  der Bias-Vektor des *Layers*. Mit  $n$  Neuronen im ausgehenden *Layer* und  $k$  Neuronen im nächsten *Layer* hat  $\mathbf{W}$   $k \cdot n$  Elemente. Für jedes Neuron in dem *Layer* werden alle *Inputs*  $\mathbf{x} = x_1, \dots, x_n$  mit all seinen Gewichten multipliziert, z.B.:  $(w_{1,1}x_1 + b_1) + (w_{1,2}x_2 + b_2) + (w_{1,3}x_3 + b_3) + \dots + (w_{1,n}x_n + b_n)$ . Daraus ergibt sich das Matrix-Vektor-Produkt:

$$\begin{aligned} \mathbf{z} &= \mathbf{W} \cdot \mathbf{x} + \mathbf{b} \\ &= \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1} & w_{k,2} & \cdots & w_{k,n} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \\ &= \begin{bmatrix} (w_{1,1}x_1 + b_1) + (w_{1,2}x_2 + b_2) + \dots + (w_{1,n}x_n + b_n) \\ (w_{2,1}x_1 + b_1) + (w_{2,2}x_2 + b_2) + \dots + (w_{2,n}x_n + b_n) \\ \vdots \\ (w_{k,1}x_1 + b_1) + (w_{k,2}x_2 + b_2) + \dots + (w_{k,n}x_n + b_n) \end{bmatrix} \end{aligned} \quad (2.5)$$

## Aktivierungsfunktionen

Das Ergebnis dieser Multiplikation ist ein  $k$  dimensionaler Vektor  $\mathbf{z}$ , dessen Elemente anschließend elementweise von einer Aktivierungsfunktion  $\varphi(\mathbf{z})$  verarbeitet werden. Aktivierungsfunktionen sind wichtige Bausteine eines ANN. Durch ihre Eigenschaften bringen Aktivierungsfunktionen eine Nichtlinearität in ein ANN, wodurch es erst in der Lage ist, komplexe Zusammenhänge zwischen Ein- und Ausgaben zu lernen. Abstrakt betrachtet wandeln Aktivierungsfunktionen ein Eingangssignal  $z$  eines Neurons in ein Ausgabesignal  $a$  um, welches wiederum als Eingangssignal in einer nachfolgenden Schicht verwendet wird. Für ANN gibt es heute eine Vielzahl unterschiedlicher Aktivierungsfunktionen mit unterschiedlichen Eigenschaften. Häufig verwendete Funktionen sind dabei die *Sigmoid*-, *Tangens hyperbolicus*- (TanH), *Schwellwert*- sowie *Rectified Linear Unit* (ReLU)-Funktion. [Mur12]

Im Folgenden sollen die zuvor genannten Aktivierungsfunktionen kurz definiert sowie deren Eigenschaften beschrieben werden:

- **Sigmoidfunktion:** Diese Aktivierungsfunktion ordnet die Eingaben  $z$  einem Intervall  $[0, 1]$  zu und ist wie folgt definiert:

$$\varphi(z) = \frac{1}{1 + e^{-z}} \quad (2.6)$$

Die Sigmoidfunktion weist alle relevanten Eigenschaften von Aktivierungsfunktionen auf: Die Funktion ist nichtlinear, kontinuierlich differenzierbar, monoton, hat einen festen Bereich für die Ausgaben und die Gradienten der Funktion sind für das Training mittels *Backpropagation* geeignet. Für einige Fälle ist die Sigmoidfunktion jedoch weniger geeignet, da sie das Problem der *Vanishing Gradients* verstärkt. Darüber hinaus ist die Ausgabe nicht um 0 zentriert, was dazu führen kann, dass die Gradienten zur Gewichtsaktualisierung während des Trainings zu sehr in eine falsche Richtung gehen. Die Sigmoidfunktion wird häufig in ANN für Klassifikationsprobleme genutzt. Eine Darstellung der Funktion ist in Abbildung 2.7 in Bereich (a) zu sehen.

- **Tangens hyperbolicus:** TanH ordnet die Eingaben  $z$  einem Wertebereich  $[-1, 1]$  zu. Die Funktion ist nichtlinear, aber im Gegensatz zur Sigmoidfunktion ist die Ausgabe nullzentriert. Die Funktion ist wie folgt definiert:

$$\varphi(z) = 1 - \frac{2}{e^{2z} + 1} \quad (2.7)$$

Neben den gleichen Eigenschaften der Funktion wie die der Sigmoidfunktion, sind die Gradienten der Funktion wesentlich stärker, was beim Training zu einer schnelleren Konvergenz führen kann. Eine Darstellung der Funktion ist in Abbildung 2.7 in Bereich (b) zu sehen.

- **Schwellwertfunktion:** Diese Art der Aktivierungsfunktion wird im *Output Layer* eines ANN verwendet, wenn Eingaben binär klassifiziert werden sollen. Die Definition der Funktion lautet:

$$\varphi(z) = \begin{cases} 1 & z \geq \theta \\ 0 & \text{sonst} \end{cases} \quad (2.8)$$

Dabei ist  $\theta$  ein zuvor definierter Schwellwert. Aufgrund ihrer Eigenschaften wird die Schwellwertfunktion selten in der Praxis angewendet. Eine Darstellung der Funktion ist in Abbildung 2.7 in Bereich (c) zu sehen.

- **ReLU:** Die *Rectified Linear Unit*-Funktion ist wie folgt definiert:

$$\varphi(z) = \max(0, z) \quad (2.9)$$

Entgegen ihres Namens ist die Funktion nichtlinear und weist daher die gleichen positiven Eigenschaften wie die Sigmoidfunktion, jedoch mit einer besseren *Performance* auf. Aufgrund ihrer einfachen Definition ist die Berechnung der Funktion weniger rechenintensiv. Darüber hinaus verhindert sie aufgrund ihrer Eigenschaften das *Vanishing Gradients* Problem. Eine Visualisierung der Funktion ist in Abbildung 2.7 in Bereich (d) dargestellt.

Neben den gerade genannten Aktivierungsfunktionen gibt es noch eine Vielzahl weiterer Funktionen, wie z.B. die *Leaky ReLU*, *ArcTan*, *Softmax* oder die Identität der Eingabe. Jede der Funktionen hat

unterschiedliche Eigenschaften und somit auch unterschiedliche Effekte auf ein ANN. Die Wahl einer entsprechenden Aktivierungsfunktion in einem *Layer* hängt daher immer stark von dem Kontext ab, für den ein ANN verwendet wird. [GBC16]

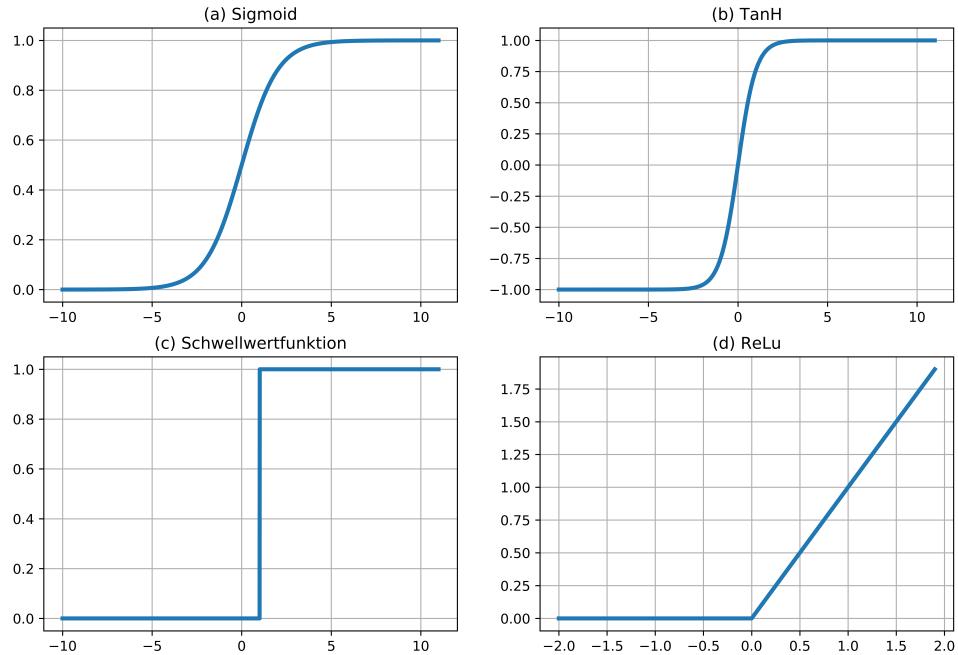


Abbildung 2.7.: Verschiedene Aktivierungsfunktionen: (a) Sigmoid, (b) Tangens Hyperbolicus, (c) Schwellwertfunktion, (d) ReLU

## Training künstlicher neuronaler Netze

Abstrakt betrachtet lernt ein ANN während des Trainings das Mapping  $f^* : \mathbf{x} \mapsto \hat{\mathbf{y}}$  der Eingaben  $\mathbf{x}$  zu den Ausgaben  $\hat{\mathbf{y}}$  des Modells  $f^*$ , indem die Parameter  $\theta$  des Modells angepasst werden. Modelle des *supervised Learning* nutzen hierfür historische Daten, um  $\theta$  anzupassen. Vereinfacht ausgedrückt ist das Lernen daher ein der Parameter, respektive der Gewichte. Ein häufig für das Training von ANN genutzter Algorithmus heißt *Backpropagation*. Der Algorithmus führt einen Gradientenabstieg auf den Gewichten durch, um den Fehler einer *Loss Funktion*  $\mathcal{L}(\mathbf{x}, \theta)$  zu minimieren. Ziel des Trainingsprozesses ist es, die Kombination der Parameter  $\theta$  zu finden, für die  $\mathcal{L}(\mathbf{x}, \theta)$  einen möglichst geringen Fehler aufweist. Abstrakt betrachtet führt *Backpropagation* dafür eine Modifikation der Gewichte durch, abhängig davon, wie groß der Einfluss des Gewichts auf den Fehler des ANN ist. Der Fehler wird anschließend rückwärts durch das Netz propagiert und die Gewichte entsprechend der Auswirkung ihres Fehlers angepasst. In Abbildung 2.8 ist das abstrakte Prinzip von *Backpropagation* anhand eines einzelnen Gewichtes  $\theta$  dargestellt. Ausgehend von einer initialen Parameterbelegung  $\theta_{start}$  führt *Backpropagation* einen Gradientenabstieg (dargestellt durch die schwarzen Pfeile) durch, um den Punkt  $\mathcal{L}_{min}(\theta)$  der Funktion  $\mathcal{L}(\theta)$  zu erreichen. [GBC16; Sch14]

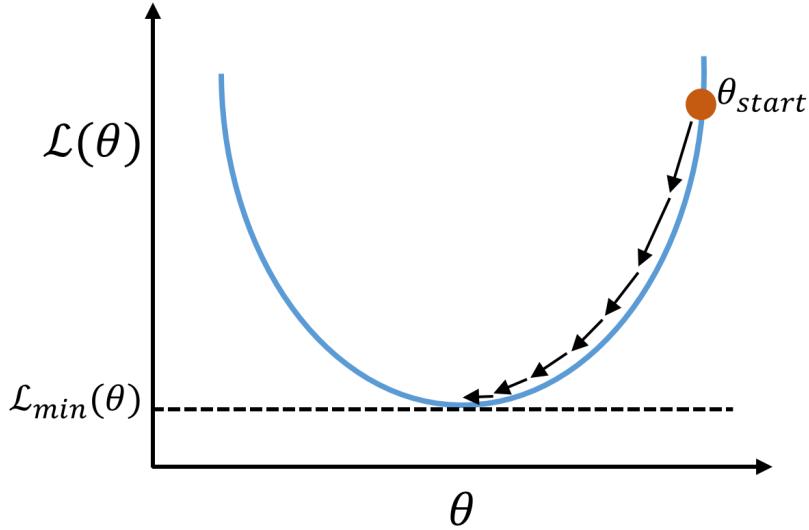


Abbildung 2.8.: Gradientenabstieg zur Minimierung einer Kostenfunktion  $\mathcal{L}(\theta)$  (eigene Darstellung nach [Mur12])

Eine Iteration von *Backpropagation* kann in sechs Schritte unterteilt werden. In Abbildung 2.9 ist der Ablauf einer solchen Iteration schematisch dargestellt. Zu Beginn des Trainings werden die Gewichte des Modells zunächst entweder zufällig oder anhand eines Algorithmus wie der *Xavier Initialisierung* initialisiert. Im zweiten Schritt wird eine sog. *Forward Propagation* durchgeführt. Dabei werden die Eingaben  $x_t$  durch das ANN propagiert, um die Ausgaben  $\hat{y}_t$  des Modells zu erhalten. Die Ausgabe eines ANN mit  $L$  Schichten kann wie folgt berechnet werden:

$$\hat{y} = a_n^L = [\varphi(\sum_m w_{n,m}^L [\dots [\varphi(\sum_j w_{k,j}^{(2)} [\varphi(\sum_i w_{j,i}^{(1)} x_i + b_j^{(1)})] + b_k^{(2)})] \dots]_m + b_n^L)]_n \quad (2.10)$$

Anhand der berechneten Ausgabe und einem Label  $y_t$  kann anschließend mittels einer *Loss Function* im dritten Schritt der Fehler  $\delta_t$  des Modells bestimmt werden. Je nach Problemklasse werden unterschiedliche *Loss Functions* verwendet. Eine gängige Funktion für die Regression ist beispielsweise der *Root-Mean-Square Error* (RMSE). Der Fehler für  $n$  Sample kann dabei wie folgt bestimmt werden:

$$\delta = \sqrt{\frac{1}{n} \sum_{j=1}^n (y_i - \hat{y}_j)^2} \quad (2.11)$$

In den anschließenden Schritten vier und fünf werden zunächst die Fehler der einzelnen Neuronen berechnet und diese anschließend durch das Netz zurückgeführt. Im letzten, dem sechsten, Schritt werden anschließend die Gewichte des Netzes aktualisiert. Die neuen Gewichte  $w'$  werden wie folgt aktualisiert:

$$w' = w + \Delta w \quad \text{mit} \quad \Delta w = -\eta \nabla L(w) \quad (2.12)$$

Der Gradient einer *Loss Function*  $\nabla \mathcal{L}(\mathbf{w})$  am Punkt  $\mathbf{w} \in \mathbb{R}^{k^*}$  ist ein  $k^*$  dimensionaler Vektor partieller Ableitungen  $\frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_i}$  für jedes Gewicht  $w_i$  mit  $i = 1, \dots, k^*$ :

$$\nabla \mathcal{L}(\mathbf{w}) = \begin{bmatrix} \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_1} \\ \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_2} \\ \vdots \\ \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_{k^*}} \end{bmatrix} \quad (2.13)$$

Konkret wird die Gewichtsänderung  $\Delta w_{i,j}$  eines Gewichtes anhand der hebbischen Lernregel wie folgt berechnet [Heb02]:

$$\Delta w_{i,j} = -\eta \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_{i,j}} = -\eta \delta_j a_j \quad (2.14)$$

Dabei ist  $\delta_j$  das Fehlersignal eines Neurons  $j$ ,  $a_j$  die Aktivierungen des Neurons,  $\eta$  die Lernrate bzw. Schrittweite des Gradientenabstiegs und  $\Delta w_{i,j}$  das Gewichtsupdate des Gewichtes  $w_{i,j}$  zwischen dem Neuron  $i$  und  $j$ . Das Minuszeichen vor dem Term sorgt entsprechend dafür, dass das Gewichtsupdate in die entgegengesetzte Richtung des Gradienten durchgeführt wird, sodass ein Abstieg stattfindet. Die Berechnung des Fehlersignals  $\delta_j$  für ein Neuron  $j$  kann berechnet werden anhand:

$$\delta_j = \begin{cases} \varphi'(z_j)(a_j - y_j) & \text{Falls } j \text{ ein Ausgabeneuron ist} \\ \varphi'(z_j) \sum_k \delta_k w_{k,j} & \text{Falls } j \text{ ein Neuron im Hidden Layer ist} \end{cases} \quad (2.15)$$

Dabei ist  $\varphi'$  die abgeleitete Aktivierungsfunktion eines Neurons  $j$ ,  $z_j$  die gewichteten Eingaben des Neurons,  $a_j$  die Ausgaben des Neurons und  $\sum_k \delta_k w_{k,j}$  das gewichtete Fehlersignal der folgenden Neuronen von  $j$ . [Stu12]

Das in Abbildung 2.9 beschriebene Prozedere zur Aktualisierung der Gewichte wird solange wiederholt, bis ein Abbruchkriterium erfüllt ist oder ein globales Minimum der *Loss Function* gefunden wurde.

Verfahren zur Gewichtsadaption gehören zur Klasse der Optimierungsverfahren und werden daher auch *Solver* genannt. Die zuvor beschriebene Variante von *Backpropagation* heißt *Stochastic Gradient Descent* (SGD). Dabei werden die Gewichtsänderungen nach jedem *Sample* aus dem Trainingsdatensatz durchgeführt. Weitere bekannte Verfahren sind u.a. *Adam*, Quasi-Newton Verfahren (z.B. LBFGS) oder *Nesterov Momentum*. Auf Momentum basierende *Solver* nutzen häufig einen *Moving Average* der Gradienten, um eine Glättung der Gradienten zu erzielen und den Gradientenabstieg so zu optimieren. [GBC16]

### 2.1.3. Deep Learning

*Deep Learning* (DL) ist ein Teilbereich des *Machine Learnings* und ermöglicht Modelle, die aus vielen Schichten bestehen, um Darstellungen von Daten mit mehreren Abstraktionsebenen zu lernen. So

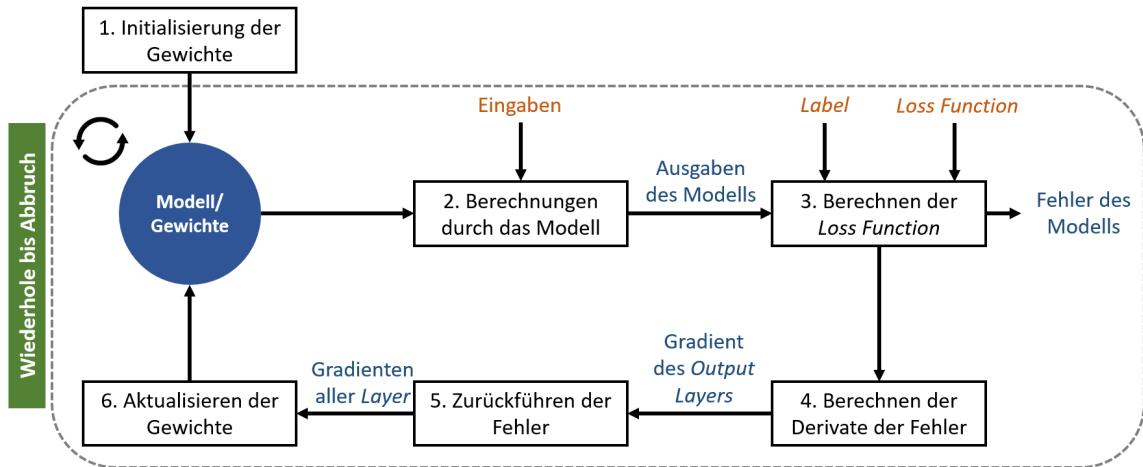


Abbildung 2.9.: Abstrakte Funktionsweise von Backpropagation (eigene Darstellung nach [GBC16; Mur12])

können komplexe Strukturen in großen Datensätzen entdeckt werden. Dies führt dazu, dass diese Modelle den Stand der Technik in den Bereichen der Spracherkennung, visuellen Objekterkennung und vielen anderen Bereichen deutlich verbessert haben und weiterhin verbessern. Die Entwicklung von DL wurde zum Teil durch die Limitierung klassischer ML Algorithmen motiviert. So haben klassische ML Algorithmen beispielsweise Probleme mit Daten hoher Dimensionalität<sup>1</sup>. Wohingegen konventionelle ML Algorithmen durch ihre Fähigkeit beschränkt nur reine Daten verarbeiten können, lernen komplexe DL Modelle hingegen eine Repräsentation der Daten, die es ihnen ermöglicht komplexe Aufgaben, wie die Identifizierung von Objekten in Bildern, zu bewältigen. [LBH15]

DL Modelle verwenden mehrere Schichten, um schrittweise Merkmale höherer Ebene aus den Rohdaten zu extrahieren. So können beispielsweise Modelle zur Bildverarbeitung, sog. *Convolutional Neural Network* (CNN), in unteren Schichten Kanten identifizieren, während in höheren Schichten die für den Menschen relevante Konzepte wie Ziffern, Buchstaben oder Flächen identifizieren werden können. [GBC16; Sch14]

Bekannte und heute häufig verwendete Architekturen von tiefen KNN sind u.a. *Recurrent Neural Networks* (RNN), *Convolutional Neural Nets* (CNN) oder *Generative Neural Nets* (GAN). [GBC16]

#### 2.1.4. Concept Drift

Daten, die durch einen generierenden Prozess erzeugt werden, können sich im Laufe der Zeit ändern. In einem SCADA System können z.B. zusätzliche Systeme/Komponenten angeschlossen werden oder sich das zugrundeliegende Verhalten bei der Benutzung des Systems ändern. Dies kann bei ML-Modellen, die eine statistische Beziehung zwischen *Features* und *Labels* lernen, zu einer sich verschlechternden Performance führen. Dieses Problem der sich ändernden zugrunde liegenden Beziehungen in den Daten wird im Kontext des ML als *Concept Drift* bezeichnet. [Tsy04]

<sup>1</sup> Dieses Problem ist auch bekannt unter dem Namen „Fluch der Dimensionalität“.

In Abbildung 2.10 sind vier Arten von Änderungen in Datenströmen dargestellt. Grundsätzlich kann dabei zwischen plötzlichen (a), schrittweisen (b), inkrementellen (c) und wiederkehrenden (d) Änderungen unterschieden werden [CW17]. Die Analyse zur Erkennung solcher Änderungen in Datenströmen wird in der Statistik auch als *Change Point Analyse* bezeichnet [AC16].

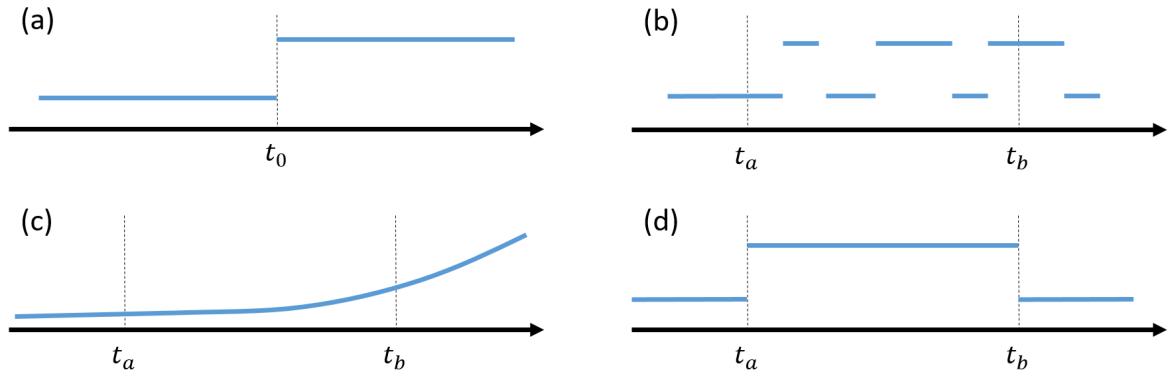


Abbildung 2.10.: Arten von Änderungen in Datenströmen: (a) plötzlich, (b) schrittweise, (c) inkrementell, (d) wiederkehrend (eigene Abbildung nach [CW17])

Die von den Modellen gelernte Beziehung zwischen den *Features*  $x$  und den *Labels*  $y$  entspricht einer Funktion  $\hat{y} = f^*(x)$ . Häufig wird davon ausgegangen, dass dieses *Mapping* statisch ist. Dies bedeutet, dass das aus den historischen Daten gelernte *Mapping* für neue Daten in Zukunft genauso gültig ist und sich die Beziehungen zwischen *Features* und *Label* nicht ändern. Für die meisten praktischen Anwendungen ist dies jedoch nicht der Fall, da sich im Laufe der Zeit nahezu immer Änderungen in den Daten ergeben. [Gam+14]

In Abbildung 2.11 ist das Konzept des *Concept Drifts* am Beispiel einer Klassifikation dargestellt. Ein Modell lernt in einem Zeitpunkt  $t_1$  eine *Data Decision Boundary*, um die Klassen A und B voneinander zu separieren (linke Seite der Abbildung). Mit der Zeit ändern sich die zugrunde liegenden Beziehungen in den Daten. In einem Zeitpunkt  $t_2$  ist das Modell dann nicht mehr in der Lage, alle Daten korrekt zu klassifizieren, was zu einer erhöhten Fehlerquote des Modells führt (rechte Seite der Abbildung).

Das Problem des *Concept Drifts* spielt für Modelle zur Identifizierung von Anomalien eine besondere Rolle. Ändern sich die zugrunde liegenden Beziehungen in den Daten über die Zeit, ist das Modell nicht mehr in der Lage, Anomalien als solche zu erkennen [CW17]. Aus einer anwendungsspezifischen Sicht heißt dies, dass in den Daten z.B. ein Angriff auf ein Netzwerk oder ein Kreditkartenbezug erkannt wird, obwohl es sich um ein normales Ereignis in den Daten handelt. Konkret bedeutet dies, dass durch den *Concept Drift* die *False Positive Rate*, also die Anzahl der fälschlicherweise als Anomalie erkannten Events, des Modells steigt [Das+19].

Die einfachste und gängigste Methode, um ein Modell vor einem *Concept Drift* zu schützen, ist ein periodisches Re-Training des Modells mit neuen Daten [Mor+13]. Je nachdem wie schnell sich die zugrunde liegenden Beziehungen in den Daten ändern, ist ein Re-Training des Modells früher oder später notwendig. Verschiedene Methoden können dabei helfen, den optimalen Zeitpunkt für ein Re-Training zu bestimmen. In der Praxis werden hierfür die Abstände zwischen der Verteilungen der Trainings- und operativen Daten mittels Metriken, wie z.B. der *Kullback-Leibler-Divergenz*, bestimmt, um zu messen, ob sich die Daten im operativen Betrieb eines Modells stark von den Trai-

ningsdaten unterscheiden. Ist die Distanz zwischen den Trainings- und aktuellen Daten zu groß, ist ein Re-Training notwendig. [Gam+14]

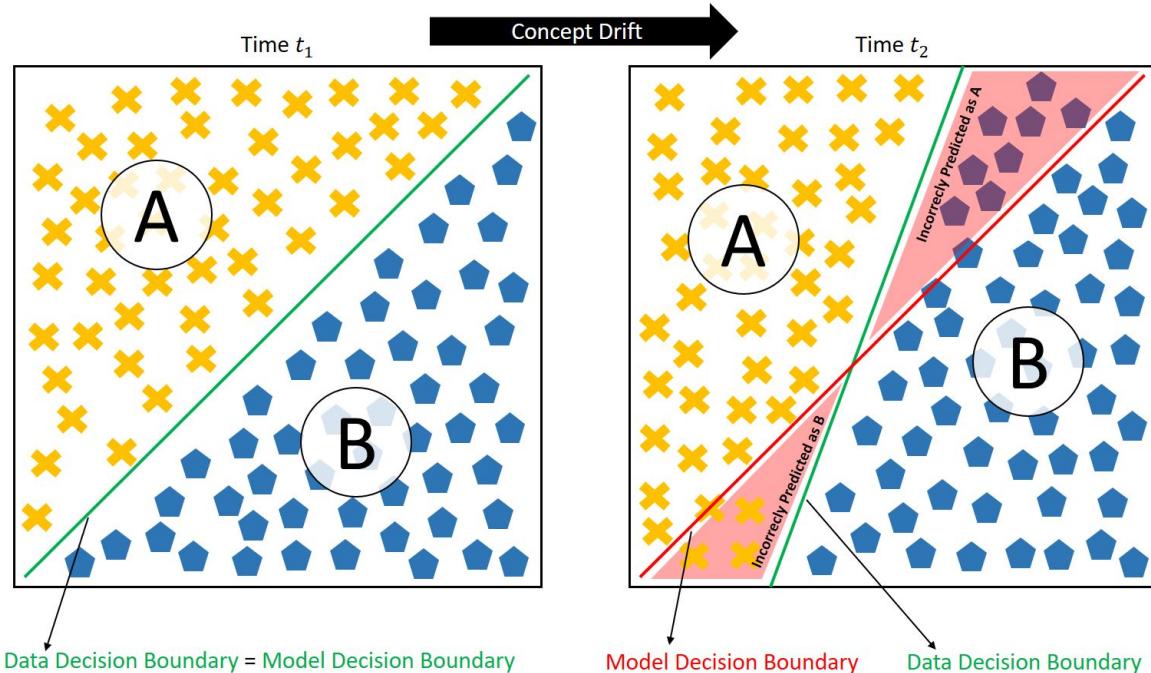


Abbildung 2.11.: Beispiel des Concept Drifts für eine Klassifikation (eigene Abbildung nach [Chi19])

## 2.2. Autoencoder

Autoencoder (AE) sind eine *unsupervised* Art von ANN, die darauf trainiert werden, ihren *Input*  $x$  als *Output*  $x'$  zu reproduzieren. In Abbildung 2.12 ist das Schema eines AE dargestellt. Im Grunde bestehen AE aus zwei Teilen. Zum einen aus dem *Encoder*, auf der linken Seite der Abbildung 2.12, und einem *Decoder*, auf der rechten Seite der Abbildung. Der *Encoder* codiert zunächst den *Input*  $x$ , sodass dieser im *Hidden Layer* als *Code*  $h$  vorliegt. Die *Encoder*-Funktion kann dabei wie folgt beschrieben werden:  $h = f(x)$ . Anhand des *Codes*  $h$  aus dem *Hidden Layer* versucht der *Decoder* anschließend den *Input* wieder zu rekonstruieren, was durch die entsprechende *Decoder*-Funktion  $x' = g(h)$  beschrieben wird. [GBC16]

Gelingt es einem AE einfach  $g(f(x)) = x$  zu lernen, ist er nicht besonders nützlich. Stattdessen werden AE so konzipiert, dass sie nicht in der Lage sind, perfekt zu kopieren, sondern nur annähernd die Eingaben kopieren, die den Trainingsdaten ähneln. Da das Modell gezwungen wird zu priorisieren, welche Daten für die Rekonstruktion relevant sind, können durch AE häufig relevante Eigenschaften der Daten identifiziert werden. [GBC16]

Ein AE mit nur einem *Hidden Layer* hat einen *Encoder*, wie in Gleichung 2.16 dargestellt, und einen *Decoder*, wie in Gleichung 2.17 dargestellt. Dabei ist  $W_{xh}$  die Gewichtsmatrix des *Encoders*,  $W_{hx}$  die Gewichtsmatrix des *Decoders* und  $b_{xh}$  entsprechend der Bias-Vektor des *Encoders* bzw.  $b_{hx}$  der Bias-Vektor des *Decoders* und  $\varphi$  eine nichtlineare Aktivierungsfunktion.

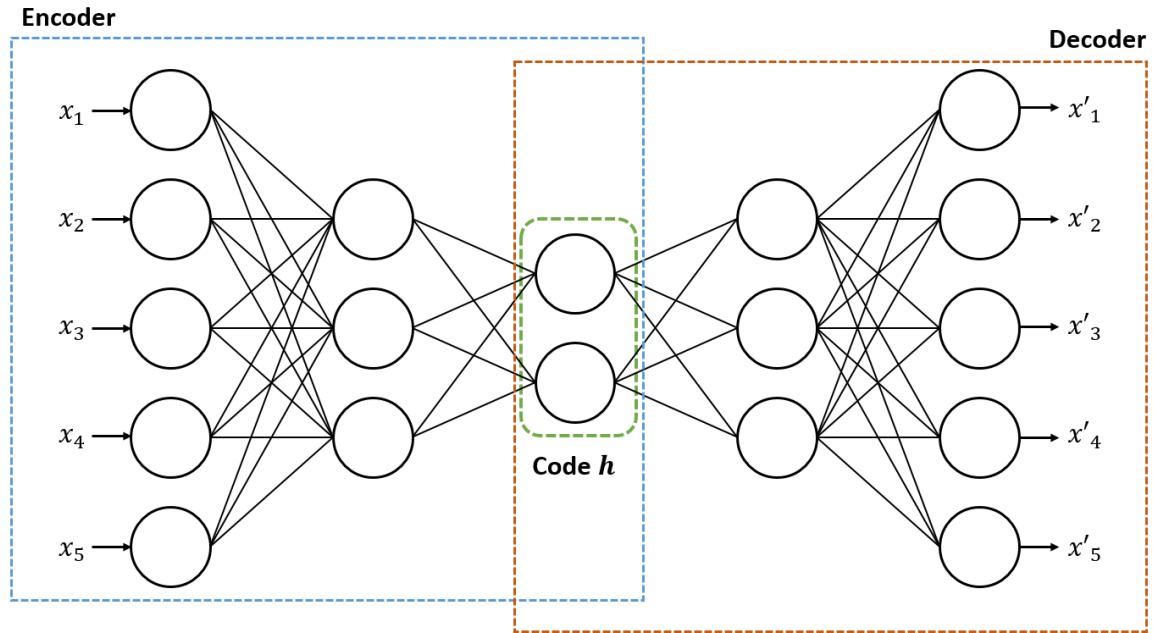


Abbildung 2.12.: Schema eines Autoencoders (eigene Abbildung nach [GBC16])

$$\mathbf{h} = \varphi(\mathbf{W}_{xh}\mathbf{x} + \mathbf{b}_{xh}) = f(\mathbf{x}) \quad (2.16)$$

$$\mathbf{x}' = \varphi(\mathbf{W}_{hx}\mathbf{h} + \mathbf{b}_{hx}) = g(\mathbf{h}) \quad (2.17)$$

$$\mathbf{x}' = \varphi(\mathbf{W}_{hx}\varphi(\mathbf{W}_{xh}\mathbf{x} + \mathbf{b}_{xh}) + \mathbf{b}_{hx}) = g(f(\mathbf{x})) \quad (2.18)$$

$$\epsilon = \|\mathbf{x} - \mathbf{x}'\| \quad (2.19)$$

Der Fehler des AE kann mittels des sog. *Reconstruction Loss*  $\epsilon$ , wie in Gleichung 2.19 dargestellt, bestimmt werden. Dabei wird die Distanz zwischen dem *Input*  $\mathbf{x}$  und dem *Output*  $\mathbf{x}'$  als Fehlermaß verwendet. Als Metrik werden dabei, je nach Anwendungsfall, unterschiedliche Funktionen angewandt. Während des Trainings eines AE wird mittels eines Trainingsalgorithmus, wie z.B. *Backpropagation*, der Fehler  $\epsilon$  minimiert. Da zur Bestimmung des Fehlers kein explizites *Label*, sondern nur der *Output* sowie der *Input* des AE, verwendet wird, gehören AE zur Kategorie der *unsupervised ANN*. [GBC16] In Listing 1 ist der Pseudocode für das Training eines AE dargestellt.

---

**Algorithm 1** Training eines Autoencoders nach [AC15]

---

**INPUT:** Dataset:  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}$

**OUTPUT:** Encoder  $f_\phi$ , decoder  $g_\theta$

$\phi, \theta \leftarrow$  Initialize parameters

**repeat**

$\epsilon = \sum_{i=1}^N \|\mathbf{x}^{(i)} - g_\theta(f_\phi(\mathbf{x}^{(i)}))\|$  Calculate sum of reconstruction error

$\phi, \theta \leftarrow$  Update parameters using gradients of  $\epsilon$  (z.B. with *SGD*)

**until** convergence of parameters  $\phi, \theta$

---

### 2.2.1. Arten von Autoencodern

Grundsätzlich kann zwischen *undercomplete* und *overcomplete* AE unterschieden werden. Beide Arten unterscheiden sich dabei darin, wie viele Neuronen im *Hidden Layer* verwendet werden. [GBC16]

#### *Undercomplete* AE

Den *Input* eines AE als *Output* zu kopieren mag auf den ersten Blick nutzlos erscheinen. Typischerweise liegt der Fokus dabei jedoch nicht auf dem eigentlichen *Output* des AE. Vielmehr wird der AE dazu genutzt, nützliche Eigenschaften der Daten im *Code h* zu identifizieren. Damit der AE hierzu in der Lage ist, wird für den *Code h* eine kleinere Dimension als für den *Input x* verwendet. AE, deren Dimension im *Code* geringer als die Eingabedimension ist, werden daher als *undercomplete* bezeichnet. Das Erlernen einer möglichst guten Repräsentation des *Inputs x'* zwingt den *undercomplete* AE dazu, nützliche *Feature* aus den Daten zu lernen. Bei dem in Abbildung 2.12 bereits vorgestellten Schema eines AE handelt es sich ebenfalls um einen *undercomplete* AE, da die Anzahl der Neuronen im mittleren *Hidden Layer* geringer ist, als die Anzahl der *Input* Neuronen. [GBC16]

Der in Formel 2.20 dargestellte Term beschreibt das Training eines *undercomplete* AE als Optimierungsproblem. Das Ziel des Trainings ist dabei die Minimierung der Funktion  $\mathcal{L}$ :

$$\min_{\theta_{xh}, \theta_{hx} \in \mathbb{R}^n} \mathcal{L}(x, g(f(x, \theta_{xh}), \theta_{hx})) \quad (2.20)$$

Dabei ist  $\mathcal{L}$  eine *Loss Function*, die die Distanz zwischen dem *Input x* und dem *Output g(f(x, θ<sub>xh</sub>), θ<sub>hx</sub>)* des AE, z.B. mittels dem *Mean Squared Error* (MSE), misst. Wenn der AE linear ist, also nur lineare Aktivierungsfunktionen verwendet werden, und als *Loss Function* der MSE verwendet wird, entspricht dieser AE in der Funktion und dem Ergebnis einer *Principal Component Analysis* (PCA), welche häufig zur Dimensionsreduktion verwendet wird. Nichtlineare AE können hingegen aufgrund ihrer größeren Kapazität eine mächtigere Repräsentation der Daten lernen. Ist die Kapazität eines AE jedoch zu groß, kann der AE das einfache Kopieren des *Inputs* lernen, ohne dabei relevante Eigenschaften aus den Daten lernen zu müssen. [GBC16]

#### *Overcomplete* AE

*Overcomplete* AE zeichnen sich dadurch aus, dass sie im *Hidden Layer* eine größere Dimension als im *Input* bzw. *Output Layer* aufweisen, wodurch sie zu einer Vielzahl von Zwecken verwendet werden können. Aufgrund der geringen Relevanz für das Verständnis der vorliegenden Arbeit, soll jedoch nicht detailliert auf die Funktionsweise eingegangen werden. Eine detaillierte Beschreibung verschiedener Arten und Funktionsweisen von *overcomplete* AE ist u.a. in [GBC16] oder [Bal12] zu finden.

### 2.2.2. Autoencoder zur Anomalieerkennung

Die AE-basierte Erkennung von Anomalien ist eine abweichungsbasierte Anomalieerkennungsme-thode unter Verwendung des *Semi-Supervised Learnings*. Dabei wird der *Reconstruction Loss* des AE als Metrik zur Anomalieerkennung verwendet. Datenpunkte mit einem hohen *Reconstruction*

*Loss* gelten als Anomalie, wohingegen Datenpunkte mit einem geringen *Reconstruction Loss* als normal betrachtet werden. Während des Trainings des AE werden nur normale *Samples* verwendet. Dies führt dazu, dass der AE normale Daten später mit einem sehr geringen Fehler rekonstruieren kann, da er gelernt hat, die relevanten Eigenschaften aus den normalen Daten zu extrahieren. Die Eigenschaften von unbekannten bzw. anormalen Daten konnten vom AE während des Trainings nicht gelernt werden, was so zu einem hohen *Reconstruction Loss* führt. [AC15]

In 2 ist der Pseudocode eines AE-basierten Algorithmus zur Erkennung von Anomalien dargestellt. Dabei ist zu sehen, wie der *Reconstruction Loss*  $\epsilon$  verwendet wird, um zu entscheiden, ob es sich bei einem *Sample* um einen normale oder anormalen Datenpunkt handelt. Für diese Entscheidung wird ein zuvor definierter Schwellwert  $\lambda$  verwendet. [AC15; Bor+18]

---

**Algorithm 2** AE-basierter Algorithmus zur Erkennung von Anomalien nach [AC15]

---

**INPUT:** Normal dataset  $X$ , Anomalous dataset  $x^{(i)}$  with  $i = 1, \dots, N$ , threshold  $\lambda$

**OUTPUT:** Reconstruction error  $\epsilon = \|x - x'\|$

```

 $\phi, \theta \leftarrow$  train an autoencoder using the normal dataset  $X$ 
for  $i = 0, N$  do
     $\epsilon(i) = \|x^{(i)} - g_\theta(f_\phi(x^{(i)}))\|$ 
    if  $\epsilon(i) > \lambda$  then
         $x^{(i)}$  is an anomaly
    else
         $x^{(i)}$  is not an anomaly
    end if
end for

```

---

Mittels des Schwellwertes  $\lambda$  kann gesteuert werden, ab wann ein Datensatz als Anomalie gilt, oder ob es sich bei einem Datensatz noch um natürliches Rauschen aus den Daten handelt. In der Praxis muss dieser Parameter sorgfältig ausgewählt werden, damit das Modell noch eine gute Performance aufweist. [AC15]

## 2.3. Anomalieerkennung

Die Erkennung von Anomalien ist ein wichtiges Problem, welches in verschiedenen Forschungs- und Anwendungsbereichen bereits umfassend untersucht wurde. Die Anomalieerkennung beschäftigt sich mit dem Problem, Muster in Daten zu finden, die nicht dem erwarteten Verhalten entsprechen [CBK09]. Die Bedeutung der Erkennung von Anomalien beruht auf der Tatsache, dass Anomalien in Daten in einer Vielzahl von Anwendungsbereichen in signifikante und oft kritische Informationen umgewandelt werden können. Beispielsweise könnte ein anormales Verhalten in einem Computer-Netzwerk darauf hindeuten, dass ein gehackter Computer vertrauliche Informationen an ein nicht autorisiertes Ziel sendet. Die Erkennung von Anomalien in Daten wurde in der Statistik bereits im 19. Jahrhundert untersucht. Im Laufe der Zeit wurden in der Forschung eine Vielzahl von Techniken zur Erkennung von Anomalien entwickelt. Viele dieser Techniken wurden speziell für bestimmte Anwendungsbereiche entwickelt, wohingegen andere allgemeiner sind.

In den folgenden Unterabschnitten soll daher zunächst in Unterabschnitt 2.3.1 definiert werden, was eine Anomalie ist, um anschließend in Unterabschnitt 2.3.2 verschiedene Arten von Anomalien zu

betrachten. In Unterabschnitt 2.3.3 werden dann speziell Anomalien in Zeitreihen betrachtet. Anschließend werden in Unterabschnitt 2.3.4 Arten der Anomalieerkennung im Kontext der verfügbaren Label beschrieben.

### 2.3.1. Definition Anomalie

Eine allgemeine Definition des Begriffs Anomalie stammt von Hawkins aus [Haw80]: „*Ein Ausreißer ist eine Beobachtung, die so sehr von den anderen Beobachtungen abweicht, dass sie den Verdacht aufkommen lässt, dass sie durch einen anderen Mechanismus hervorgerufen wurde.*“ Neben dem Begriff „Anomalie“ werden in der Literatur häufig die Begriffe „Abnormalität“, „Deviation“ oder „Ausreißer“ synonym verwendet. Abstrakt betrachtet wird ein Objekt aus einem Datensatz als Anomalie bezeichnet, wenn:

- es stark von dem normalen/bekannten Verhalten der Daten abweicht,
- es Werte annimmt, die weit von den erwarteten/durchschnittlichen Werten entfernt sind, oder
- es in seinen Eigenschaften mit keinem anderen Objekt verbunden oder vergleichbar ist.

In Abbildung 2.13 ist ein simples Beispiel für Anomalien in einem zweidimensionalen Datensatz dargestellt. Der Datensatz umfasst zwei normale Regionen,  $N_1$  und  $N_2$ , in denen die meisten Datenpunkte liegen. Datenpunkte, die weit von diesen Regionen entfernt sind, wie z.B. die Punkte  $o_1$  oder  $o_2$  sowie Punkte in der Region  $O_3$ , sind aufgrund ihrer Eigenschaften Anomalien.

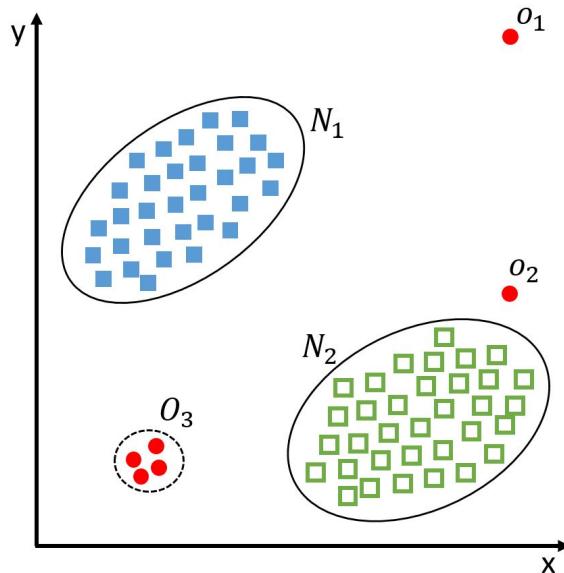


Abbildung 2.13.: Ein einfaches Beispiel für Anomalien in einem zweidimensionalen Datensatz (eigene Darstellung nach [CBK09])

Es ist oft schwer zu unterscheiden, ob es sich um eine „ausreichende“ Abweichung handelt, damit ein Punkt als Anomalie betrachtet wird. In realen Anwendungen können die Daten z.B. von einer erheblichen Menge an Rauschen umgeben sein. Rauschen kann als ein Phänomen in Daten definiert werden, welches für Analysten nicht von Interesse ist, die Datenanalyse jedoch behindert. Die Rauschentfer-

nung wird durch die Notwendigkeit motiviert, die unerwünschten Objekte zu entfernen, bevor eine Datenanalyse an den Daten durchgeführt wird. In Abbildung 2.14 ist eine Anomalie in einem zweidimensionalen Raum dargestellt. Auf der linken Seite (a) der Abbildung 2.14 sind zwei Punktwolken sowie eine in der Mitte liegende Anomalie  $o_1$  dargestellt. Auf der rechten Seite (b) der Abbildung sind die identischen Punktwolken sowie die Anomalie  $o_1$  inkl. Rauschen dargestellt. Die Beurteilung, bei welchen Punkten es sich auf der rechten Seite (b) in der Abbildung um eine Anomalie handelt oder nicht, ist nicht mehr ohne Weiteres möglich. [SU12]

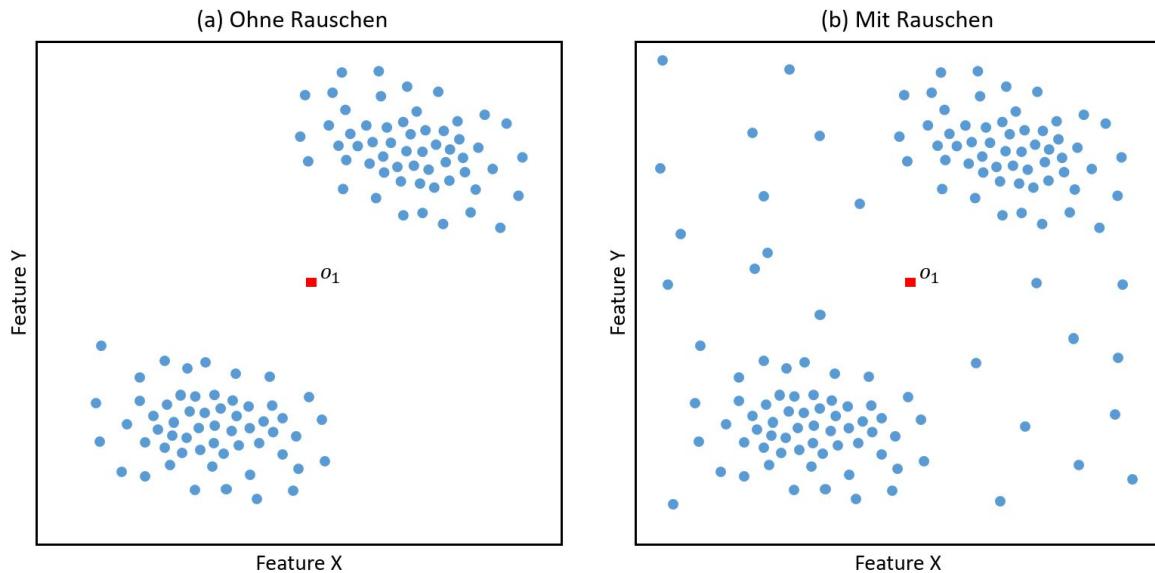


Abbildung 2.14.: *Anomalie in (a) Daten ohne Rauschen und in (b) Daten mit Rauschen*  
(eigene Darstellung nach [AS17])

In der Literatur wird häufig eine Unterscheidung für die Bedeutung der Begriffe *Outlier* und *Anomaly* getroffen. So bezeichnet ein *Outlier* einen Datenpunkt, welcher sowohl als *Anomaly* als auch als Rauschen betrachtet werden kann. Ein Datenpunkt wird hingegen als *Anomaly* bezeichnet, wenn es sich der Definition nach um eine Anomalie handelt, aber der Datenpunkt kein Rauschen darstellt. [AS17]

Die Ausgaben von Algorithmen zur Identifizierung von Anomalien können in zwei Arten von Ausgaben unterteilt werden. Die meisten Algorithmen bestimmen einen *Outlier Score*, der den Grad der Stärke eines Ausreißers für jeden Datenpunkt quantifiziert. Dieser Score kann z.B. genutzt werden, um einzelne Datenpunkte entsprechend ihrer Stärke zu sortieren. In Abbildung 2.15 ist auf einer horizontalen Achse die Einteilung von normalen Daten, Rauschen und Anomalien dargestellt. Der *Outlier Score* eines Datenpunkts nimmt dabei zu, je weiter er sich von den normalen Daten entfernt. Rauschen wird in diesem Kontext häufig als schwacher Ausreißer und Anomalien als starke Ausreißer bezeichnet. Die zweite Art von Ausgaben von Algorithmen zur Erkennung von Anomalien sind sog. *Binary Label*, welche angeben, ob es sich bei einem Datenpunkt um eine Anomalie handelt oder nicht. In der Praxis können auch *Outlier Scores* in *Binary Label* umgewandelt werden, indem ein Schwellwert festgelegt wird. Ein *Binary Label* enthält jedoch weniger Informationen, als ein *Outlier Score*, da die Information über den Grad des Ausreißers verloren geht. [AS17]

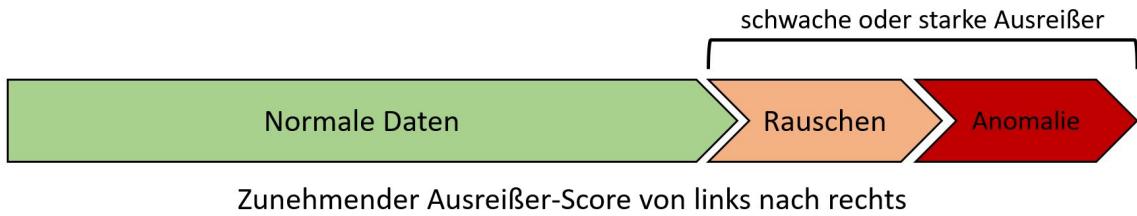


Abbildung 2.15.: *Grad der Anomalie von Daten nach [AS17]*

### 2.3.2. Arten von Anomalien

Ein wichtiger Aspekt bei der Auswahl von Algorithmen zur Identifizierung von Anomalien sind die unterschiedlichen Arten der Anomalien. Dabei können Anomalien nach Chandola, Banerjee und Kumar in drei unterschiedliche Arten unterteilt werden:

#### Point Anomalies

Ein Datenpunkt wird als *Point Anomaly* bezeichnet, wenn er zu weit vom Rest der Daten entfernt ist. Dies ist die simpelste Art einer Anomalie und steht im Mittelpunkt vieler Forschungen, insbesondere bei statistischen Verfahren. In Abbildung 2.13 liegen die Punkte  $o_1$  und  $o_2$  sowie die Punkte in der Region  $O_3$  außerhalb der Regionen  $N_1$  und  $N_2$ . Sie unterscheiden sich aufgrund ihrer Entfernung zu den normalen Daten ( $N_1$  und  $N_2$ ) und werden daher als *Point Anomalies* bezeichnet. [CBK09]

#### Contextual Anomalies

Wenn ein Datenpunkt nur in einem bestimmten Kontext anormal ist, wird er als *Contextual Anomaly* oder kontextbezogene Anomalie bezeichnet. Diese Art von Anomalie kommt häufig in Zeitreihendaten vor, wo der Kontext durch die Zeit gegeben ist. In Abbildung 2.16 ist ein Beispiel für eine kontextbezogene Anomalie dargestellt. Die Temperatur ist zu den Zeitpunkten  $t_1$  und  $t_2$  jeweils gleich. Durch den Kontext der Jahreszeit bzw. den Monat Juni wird der Messwert im Zeitpunkt  $t_2$  zu einer Anomalie. Der Datenpunkt kann dabei aufgrund von zwei Eigenschaften als kontextbezogene Anomalie identifiziert werden. Zum einen ist eine so geringe Temperatur für den Monat Juni im Vergleich zur Durchschnittstemperatur eine Anomalie. Zum anderen kann der Datenpunkt aufgrund der Nachbarschaftsbeziehungen als Anomalie erkannt werden, da die benachbarten Messwerte von  $t_2$  wesentlich höher sind. [Son+07]

#### Collective Anomalies

Wenn eine Anzahl zusammengehöriger Datenpunkte in Bezug auf den gesamten Datensatz anormal ist, werden diese zusammengehörigen Datenpunkte als *Collective Anomaly* oder kollektive Anomalie bezeichnet. Die Datenpunkte in einer kollektiven Anomalie sind einzeln möglicherweise keine Anomalie, aber ihr gemeinsames Auftreten als Sammlung gilt als anormal. In Abbildung 2.17 ist eine kollektive Anomalie in Form einer Extrasystole<sup>2</sup> in einem menschlichen Elektrokardiogramm (EKG)

<sup>2</sup> Eine Extrasystole ist eine Herzrhythmusstörung, bei der ein zusätzlicher Herzschlag außerhalb des normalen physiologischen Herzrhythmus auftritt.

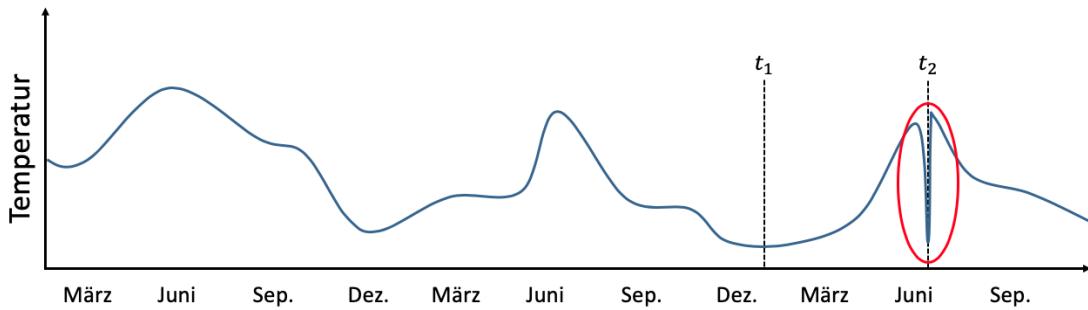


Abbildung 2.16.: Beispiel für eine kontextbezogene Anomalie nach [AS17]

dargestellt. Der rot hervorgehobene Bereich stellt dabei die Anomalie dar. Für sich genommen stellt keiner der Messwerte innerhalb dieses Bereiches eine Anomalie dar, da diese Messwerte innerhalb eines normalen Sinusrhythmus normal sind. Treten diese Messwerte jedoch als Kollektiv auf, gilt dies als anormal.

Kollektive Anomalien treten insbesondere in sequentiellen Daten (z.B. Zeitreihen), Graph-Daten oder räumlichen Daten auf. Wohingegen *Point Anomalies* in jeder Art von Daten auftreten können, können kollektive Anomalien nur in Datensätzen auftreten, in denen einzelne Datenpunkte in Relation zueinander stehen. Im Gegensatz dazu hängt das Auftreten von kontextbezogenen Anomalien von der Verfügbarkeit von Kontextattributen in den Daten ab. Sowohl eine *Point Anomaly* als auch *Collective Anomalies* können kontextbezogene Anomalien sein, sofern sie in Bezug auf einen Kontext analysiert werden. [CBK09]

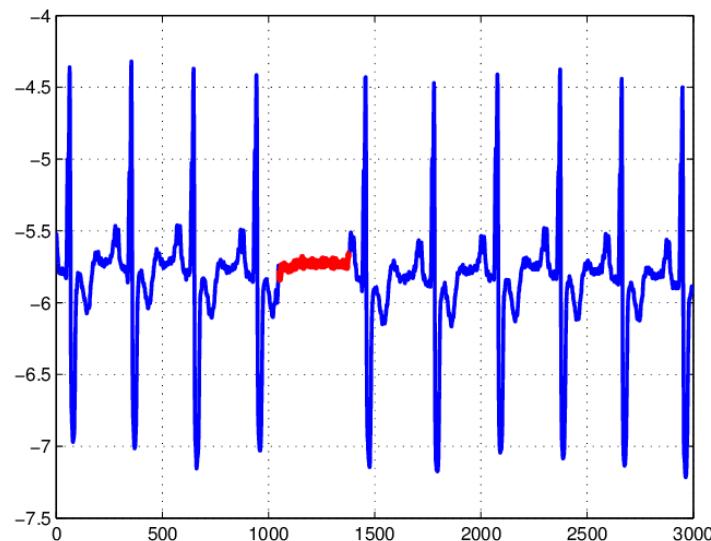


Abbildung 2.17.: Beispiel einer kollektiven Anomalie in Form einer Extrasystole (vorzeitige Kontraktion des Vorhofes) in einem menschlichen Elektrokardiogramm [CBK09]

### 2.3.3. Anomalien in Zeitreihen

Wie im vorherigen Unterabschnitt bereits erwähnt, treten kontextbezogene und kollektive Anomalien insbesondere in sequentiellen Daten, wie Zeitreihen, auf. Da Zeitreihen im Rahmen der vorliegenden Arbeit eine besondere Rolle spielen, soll im Folgenden weiter auf Anomalien in Zeitreihen eingegangen werden.

Ein wichtiger Aspekt jedes Algorithmus zur Erkennung von Anomalien ist die Art der Eingabedaten. Der *Input* ist im Allgemeinen eine Sammlung von Dateninstanzen, die auch als *Event*, *Sample*, *Observation*, *Vector* oder *Record* bezeichnet werden. Im Folgenden wird hierfür der Begriff *Sample* verwendet. Jedes *Sample* kann wiederum durch eine Reihe von Attributen (oder auch *Feature* genannt) beschrieben werden. Die einzelnen *Features* können unterschiedlichen Typs, z.B. binär, kategorisch oder kontinuierlich, sein. Jedes *Sample* kann aus nur einem *Feature* (univariat) oder mehreren *Features* (multivariat) bestehen. Bei multivariaten *Samples* können alle *Feature* vom gleichen Datentyp sein oder aus unterschiedlichen Datentypen bestehen. Die Auswahl eines geeigneten Modells zur Identifizierung von Anomalien wird u.a. durch die Art des Inputs bestimmt. Der Input eines Modells kann auch basierend auf den Beziehungen zwischen den *Samples* eines Datensatzes beschrieben werden. Die meisten Modelle befassen sich nur mit Punktdaten, bei denen es keine Beziehungen zwischen den einzelnen *Samples* gibt. Bei Zeitreihendaten gibt es jedoch eine Beziehung zwischen einzelnen *Samples* durch deren zeitlich lineare Ordnung. [SMA19]

Bei der Erkennung von Anomalien spielt die Annahme der zeitlichen Kontinuität eine wichtige Rolle. Die zeitliche Kontinuität bezieht sich auf die Tatsache, dass sich die Muster in den Daten voraussichtlich nicht abrupt ändern, es sei denn, es treten abnormale Prozesse auf. Anomalien in Zeitreihen werden als kontextbezogen beschrieben, wenn sich die Werte zu bestimmten Zeitpunkten plötzlich in Bezug auf ihre zeitlich benachbarten Werte ändern. Kollektive Anomalien in Zeitreihen sind hingegen Anomalien, bei denen ganze Zeitreihen oder große Teile einer Zeitreihe anormale Formen annehmen. [AS17]

### 2.3.4. Arten der Anomalieerkennung

Die Art der Anomalieerkennung und die damit einhergehende Auswahl geeigneter Modelle wird maßgeblich von der Verfügbarkeit der Daten bestimmt. *Label* in den Daten können angeben, ob es sich bei einem *Sample* um eine Anomalie handelt oder nicht. Gelabelte Daten sind in der Praxis jedoch sehr selten, da sie häufig manuell von Menschen erzeugt werden müssen, was zu hohen Kosten führt. Darüber hinaus ist das anormale Verhalten häufig dynamischer Natur, da z.B. neue Anomalien auftreten können, für die keine gelabelten Daten vorliegen [CBK09]. Basierend auf der Verfügbarkeit von *Labels* können drei unterschiedliche Vorgehensweisen bzw. Setups unterschieden werden.

1. *Supervised Anomaly Detection*: Beschreibt den Aufbau, bei dem die Daten vollständig gelabelte Trainings- und Testdatensätze umfassen. Ein gewöhnlicher Klassifikator kann zuerst trainiert und anschließend angewendet werden. Dieses Szenario ist der herkömmlichen Klassifikation sehr ähnlich, mit der Ausnahme, dass die Klassen in der Regel stark unausgeglichen sind. Nicht alle Algorithmen eignen sich daher für diese Aufgabe. Beispielsweise können Entscheidungsbäume wie C4.5 nicht gut mit unausgeglichenen Daten umgehen, wohingegen *Support Vector Machines* (SVM) oder *Artificial Neural Networks* (ANN) eine bessere Performance aufweisen. [Laz+03; CC19]

2. *Semi-Supervised Anomaly Detection*: Es wird auch ein Trainings- und Testdatensatz verwendet, wobei die Trainingsdaten nur normale Daten beinhalten. Im Testdatensatz können dann auch Anomalien enthalten sein, um die Performance des Modells zu bewerten. Die Grundidee ist, dass ein Modell verwendet wird, welches die Eigenschaften der normalen Klasse lernt und so in der Lage ist, Abweichungen von den normalen Daten als Anomalien zu erkennen. Dies wird auch als *One-Class-Klassifikation* bezeichnet. Hierfür häufig verwendete Algorithmen sind z.B. *One-Class SVMs* oder *Autoencoder* (AE). Darüber hinaus können auch Dichteschätzungsverfahren, wie z.B. *Gaussian Mixture Models* oder *Kernel Density Estimation*, verwendet werden, um die Wahrscheinlichkeitsdichtefunktion der normalen Klassen zu modellieren. [Laz+03; CC19]
3. *Unsupervised Anomaly Detection*: Für dieses Setup sind keine Label in den Daten notwendig. Darüber hinaus gibt es auch keinen Unterschied zwischen einem Trainings- und einem Testdatensatz. Die grundlegende Idee ist, dass ein unsupervised Anomalieerkennungsalgorithmus die Daten ausschließlich auf den intrinsischen Eigenschaften des Datensatzes bewertet. Typischerweise werden dafür Abstände oder Dichten verwendet, um zu schätzen, was normal und was anormal ist. [Laz+03; CC19]

### 2.3.5. Problemcharakteristiken der Anomalieerkennung

Ein einfacher Ansatz zur Erkennung von Anomalien ist es, einen Bereich zu definieren, der das normale Verhalten darstellt. Jede Beobachtung, die dann nicht innerhalb dieses Bereichs liegt, stellt eine Anomalie dar. Mehrere Faktoren machen diesen scheinbar einfachen Ansatz jedoch zu einer großen Herausforderung [CBK09]:

- Es ist sehr schwierig, einen normalen Bereich zu definieren, der alle möglichen normalen Verhaltensweisen umfasst. Die Grenze zwischen normalem und anormalen Verhalten ist oft ungenau. So kann eine anormale Beobachtung, die nahe an der Grenze liegt, tatsächlich normal oder anormal sein.
- Die Verfügbarkeit von gelabelten Daten zum Training der Modelle ist in der Praxis häufig ein Problem, da nicht ausreichend Daten vorliegen.
- Oft enthalten die Daten Rauschen, das den tatsächlichen Anomalien ähnelt und daher schwer zu unterscheiden und zu entfernen ist.
- Wenn Anomalien z.B. das Ergebnis böswilliger Handlungen, wie einem Cyberangriff, sind, passen sich die Angreifer häufig an, um die anormalen Beobachtungen als normal erscheinen zu lassen, wodurch die Aufgabe, normales Verhalten zu definieren, erschwert wird.
- Die genaue Definition einer Anomalie ist je nach Anwendungsbereich unterschiedlich. Beispielsweise kann in der Medizin eine geringe Abweichung von der Norm (z.B. der Körpertemperatur) eine Anomalie darstellen, wohingegen eine ähnliche Abweichung im Börsenbereich (z.B. die Schwankung des Wertes einer Aktie) als normal gilt. Daher ist es nicht einfach möglich, ein für einen Bereich entwickeltes Verfahren auf einen anderen Bereich zu adaptieren.

Aufgrund dieser Herausforderungen ist das Problem der Erkennung von Anomalien in einer allgemeinen Form nicht einfach zu lösen. Tatsächlich lösen die meisten der vorhandenen Anomalieerkennungsverfahren eine spezifische Formulierung des Problems, welche gut durch Problemcharakteristiken beschreiben werden können. In Abbildung 2.18 sind die Kernkomponenten der Problemcharak-

terisierung der Anomalieerkennung nach Chandola, Banerjee und Kumar inkl. der Einflussfaktoren Forschung und Anwendung dargestellt. In der Anomalieerkennung wird ein Problem durch die vier Faktoren Natur bzw. Art der Daten, die Verfügbarkeit von *Labels*, die Art der Anomalie und den Output der Modelle charakterisiert. Motiviert wird diese Charakterisierung durch verschiedene Anwendungsbereiche, wie die Erkennung von Betrügen oder Angriffen auf ein Computernetzwerk. Je nach Anwendungsbereich ergeben sich unterschiedliche Problemcharakteristiken. Anhand der konkreten Charakteristik eines Problems lassen sich unterschiedliche Arten der Anomalieerkennung, wie die *Supervised Anomaly Detection*, *Semi-Supervised Anomaly Detection* oder *Unsupervised Anomaly Detection* unterscheiden. Die verschiedenen Forschungsbereiche, wie das *Machine Learning* oder die Statistik haben wiederum einen großen Einfluss auf die Verfügbarkeit unterschiedlicher Verfahren und Modelle.

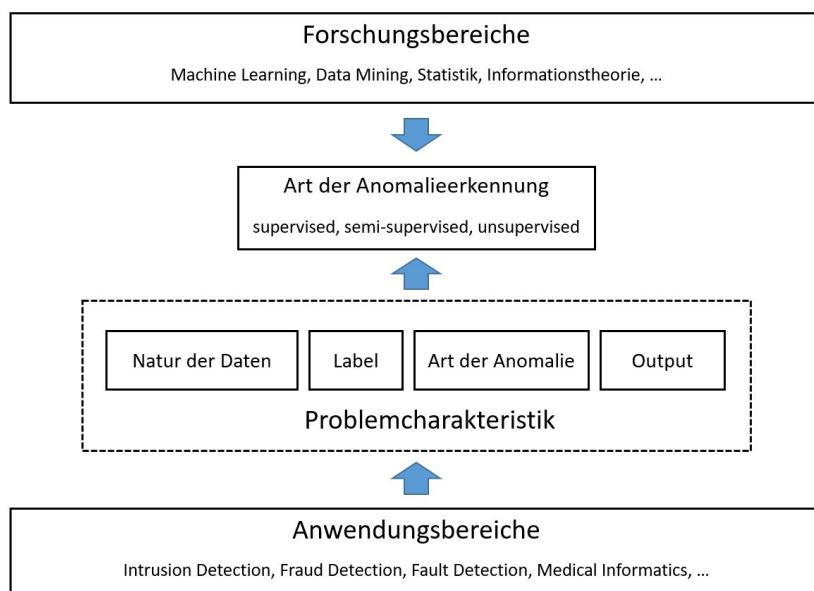


Abbildung 2.18.: *Problemcharakteristik der Anomalieerkennung nach [CBK09]*

## 2.4. Meta-Learning

Wenn ein Mensch neue Fähigkeiten lernt, beginnt er selten diese Fähigkeiten komplett neu zu lernen. Der Mensch ist in der Lage, Wissen von früher erworbenen Fähigkeiten wiederzuverwenden und kann somit neue Fähigkeiten schneller erlernen. Dabei verwendet er Wiederverwendungsmechanismen, die insbesondere Fähigkeiten nutzen, die sich aufgrund der Erfahrungen wahrscheinlich lohnen. Mit jeder erlernten Fertigkeit wird das Erlernen neuer Fähigkeiten so einfacher, da weniger Beispiele und weniger *Trail-and-Error*-Versuche benötigt werden. [Van19]

Ein grundlegendes Problem der KI ist, dass ML-Algorithmen nicht so effizient lernen können, wie Menschen. Einige *Deep Learning*-Modelle sind in ihrer Leistung heute schon besser als der Mensch. Verglichen mit dem Menschen, haben diese Modelle jedoch zwei zentrale Nachteile [Rav18]:

- **Sample efficiency:** Deep Learning-Modelle benötigen eine große Anzahl an *Samples* für ihr Training. So sind für die Erkennung von handschriftlich geschriebenen Ziffern ca. 6.000 *Samples* pro Ziffer notwendig.
- **Übertragbarkeit:** Die Modelle sind auf eine konkrete Aufgabe spezialisiert und sind nicht in der Lage, aus zuvor erlangtem Wissen zu lernen.

Das folgende Beispiel verdeutlicht diese Nachteile: Wird einem Kind ein Bild einer Giraffe gezeigt, obwohl es noch nie eine Giraffe in der Realität gesehen hat, wird das Kind von nun an nie wieder vergessen, wie eine Giraffe aussieht. Wird ein *ResNet*<sup>3</sup> mit nur einem Bild einer Giraffe trainiert, ist das ANN danach nicht in der Lage, Giraffen auf Bildern zu erkennen. Dieses Problem, ML-Modelle anhand weniger *Samples* zu trainieren, wird auch als *few-shot* oder *k-shot Learning* bezeichnet. [LST15]

*Meta-Learning* (häufig auch als „Lernen zu Lernen“ bezeichnet) ist ein Teilgebiet des ML, in dem verschiedene Ansätze entwickelt wurden, die sich mit dem Problem des *k-shot Learnings* befassen [Van19]. *Meta-Learning* hat in den letzten Jahren in der KI-Forschung an Bedeutung gewonnen, da es ein wichtiger Baustein auf dem Weg zu einer starken KI darstellt [Rav18]. Dabei haben sich fünf unterschiedliche *Meta-Learning* Ansätze herausgebildet [Fin17]:

- **Recurrent Model Meta-Learning:** Diese Art des *Meta-Learnings* basiert auf RNN, wie z.B. LSTMs. Dabei trainiert ein *Meta-Learning* Algorithmus ein RNN, welches einen Datensatz sequenziell verarbeitet und neue Eingaben einer Aufgabe verarbeitet. Ein Beispiel für diesen Ansatz ist das *Meta-Reinforcement-Learning*. [Van19]
- **Metric Meta-Learning:** Ziel dieses Ansatzes ist es, einen metrischen Raum zu bestimmen, in dem das Lernen besonders effizient ist. Dieser Ansatz wird häufig im Kontext des *few-shot Learnings* verwendet, wenn nur wenige Daten zur Verfügung stehen. [Rav18]
- **Optimizer Meta-Learning:** Beim *Optimizer Meta-Learning* Ansatz wird versucht ein ANN mittels eines weiteren ANNs zu optimieren, sodass das erste ANN eine bestimmte Aufgabe besser erfüllen kann. Dabei wird das zweite ANN genutzt, um eine Optimierung der Hyperparameter des ersten ANN durchzuführen. [Rav18]
- **Few Shots Meta-Learning:** Die Idee dieses Ansatzes ist es, DL-Modelle zu schaffen, die aus minimalen Datensätzen lernen können. Die Modelle ahnen dabei nach, wie z.B. Babys lernen, indem sie nur ein oder zwei Bilder während des Trainings sehen. Zwei Techniken dieses Ansatzes sind *Memory Augmented Neural Networks* oder *One-Shot Generative Models*. [Rav18]
- **Learning Initializations:** Dieser Ansatz hat in den letzten Jahren aufgrund seiner Erfolge an großer Bedeutung gewonnen. Dabei werden Algorithmen verwendet, um die Parameter eines zugrundeliegenden Modells so anzupassen, dass das Modell mittels einer initialen Parameterkombination mit möglichst wenigen Lernschritten neue Aufgaben lernen kann. Bekannte Algorithmen sind hierbei *Model-Agnostic Meta-Learning* (MAML) oder *Reptile*. [Fin17]

In dem folgenden Unterabschnitt 2.4.1 sollen zunächst die Grundlagen des *k-shot Learnings* weiter vertieft werden. Im darauf folgenden Unterabschnitt 2.4.2 wird der *Model-Agnostic Meta-Learning* Algorithmus von Finn, Abbeel und Levine als *Meta-Learning* Modell eingeführt. Abschließend wird

---

<sup>3</sup> Das *ResNet* ist ein 2015 von He et al. vorgestelltes CNN, welches im top-5 Benchmark mit einer Fehlerrate von 3.57 % einen geringeren Fehler, als ein Mensch aufwies.

in 2.4.3 *Reptile*, eingeführt von Nichol, Achiam und Schulman, als eine Adaption des MAML Algorithmus erläutert.

### 2.4.1. *k-shot Learning*

Wie o.g. beschreibt *k-shot Learning* ein Problem im ML, bei dem Modelle anhand weniger *Samples* neue Aufgaben lernen müssen. In Anlehnung an die Definition des Lernens aus Unterabschnitt 2.1.1 von Mitchell, kann *k-shot Learning* wie folgt definiert werden: *k-shot Learning* ist eine Klasse von ML-Problemen, spezifiziert durch Erfahrungen  $E$ , Aufgaben  $T$  und Leistungskennzahlen  $P$ , wobei  $E$  wenig Informationen für die Aufgabe  $T$  enthält. [Wan+19]

Eine bestimmte Lernaufgabe  $\mathcal{T}$  vom Typ  $(N, k)$ , d.h. mit  $N$  Klassen und je  $k$  *Samples*, kann wie folgt beschrieben werden: Für das Training von  $f_\theta$  werden Tupel der Form  $(x_i, y_i)$  verwendet, wobei  $i = 1 \dots N \cdot k$ ,  $x_i \in \mathcal{X}$  und  $y_i \in \mathcal{Y}$  ist. Dabei gibt es für jedes  $y_i$  genau  $k$  *Samples*. Die Klassifikation von  $N$  Klassen und je  $k$  *Samples* wird auch als  $N$ -way *k-shot* Klassifikation bezeichnet. Die stärkste Variante des *k-shot Learnings* stellt das sog. *zero-shot Learning*, also  $k = 0$ , dar. Dabei wird ein ANN ohne ein entsprechendes *Label* trainiert, stattdessen werden dafür Meta-Informationen für das Training verwendet. [Van19]

Eine Besonderheit des *k-shot Learnings* ist die Art und Weise, wie die Daten  $\mathcal{D}$  für das Training verwendet werden. Üblicherweise wird  $\mathcal{D} = \langle \mathcal{S}, \mathcal{Q} \rangle$  in zwei Teile, in ein *Support Set*  $\mathcal{S}$  für das Training und in ein *Query Set*  $\mathcal{Q}$  für das Testen, aufgeteilt. In Abbildung 2.19 ist ein Beispiel für ein *Support* und ein *Query Set* für die Klassifikation von Bildern dargestellt. Bei dem Beispiel handelt es sich um ein 2-shot 3-way Learning Problem. Das *Support Set* besteht aus  $N = 3$  Klassen (Labrador, Bernhardiner und Mops) mit je  $k = 2$  Bildern. Anhand des *Support Sets*  $\mathcal{S}$  wird ein ANN trainiert. Mittels des *Query Sets*  $\mathcal{Q}$  wird anschließend die Performance des ANN gemessen. Dabei besteht  $\mathcal{Q}$  aus  $N$  Bildern, die während des Trainings nicht im *Support Set* verwendet wurden. [Fin17]

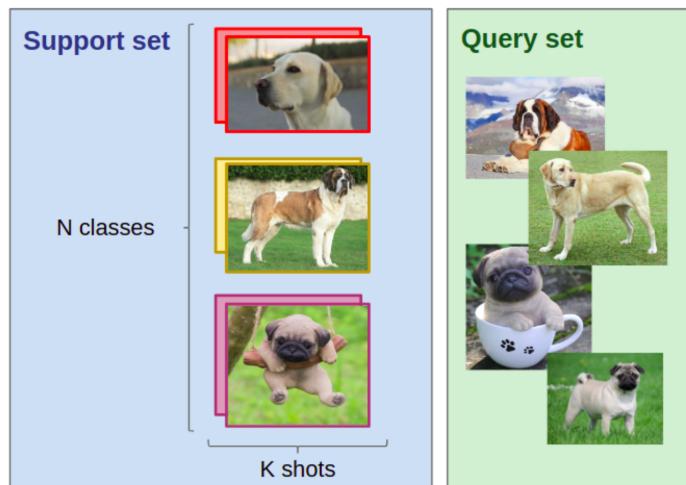


Abbildung 2.19.: Beispiel der Verwendung eines *Support Sets*  $\mathcal{S}$  und eines *Query Sets*  $\mathcal{Q}$  für das Meta-Learning anhand einer Klassifikation von Bildern [RL17]

### 2.4.2. Model-Agnostic Meta-Learning

Der *Model-Agnostic Meta Learning* (MAML) Algorithmus, vorgestellt von Finn, Abbeel und Levine in [FAL17], ist ein häufig verwendeter Algorithmus im Kontext des *Meta-Learnings*. Die Grundidee von MAML ist es, initiale Parameter zu finden, so dass das Modell mittels dieser Anfangsparameter schnell mit wenigen Gradientenschritten neue Aufgaben lernen kann. Der Begriff *Model-Agnostic* stammt daher, dass MAML keinerlei Einschränkungen für die zugrundeliegende Modellarchitektur oder der *Loss Function* mit sich bringt. MAML kann als Optimierungsalgorithmus für jedes ANN, welches mittels *Backpropagation* trainiert wird, verwendet werden. [Rav18]

Die Grundidee von MAML ist einfach: MAML optimiert ein Set von Parametern so, dass, wenn ein Gradientenschritt in Bezug auf eine bestimmte Aufgabe  $\mathcal{T}_i$  durchgeführt wird, die Parameter  $\theta$  nahe an den optimalen Parametern  $\theta_i^*$  für die Aufgabe  $\mathcal{T}_i$  liegen. In dem Diagramm in Abbildung 2.20 entspricht  $\theta$  den Modellparametern und die durchgezogene, dicke graue Linie entspricht der *Meta-Learning* Phase. Für z.B. drei unterschiedliche Aufgaben 1, 2 und 3 werden jeweils die Gradienten  $\nabla \mathcal{L}_1$ ,  $\nabla \mathcal{L}_2$  und  $\nabla \mathcal{L}_3$  (dünne graue Linien) bestimmt. Die Modellparameter  $\theta$  werden anschließend anhand der Gradienten  $\nabla \mathcal{L}_1$ ,  $\nabla \mathcal{L}_2$  und  $\nabla \mathcal{L}_3$  so angepasst, dass das Modell schnell an neue Aufgaben angepasst werden kann. Dadurch kann das Modell anschließend mit einer kleinen Adaption der Parameter  $\theta_1^*$ ,  $\theta_2^*$  oder  $\theta_3^*$  an eine neue Aufgabe angepasst werden (gestrichelte graue Linie). [FAL17]

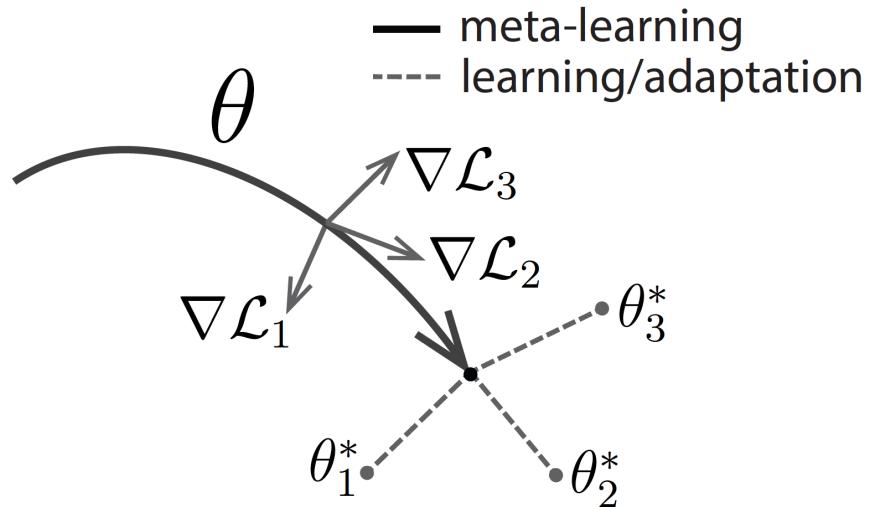


Abbildung 2.20.: Diagramm der abstrakten Funktionsweise des MAML Algorithmus [FAL17]

Finn, Abbeel und Levine definieren in [FAL17] eine Aufgabe  $\mathcal{T}$  als Viertupel der Form<sup>4</sup>:

$$\mathcal{T} = \{\mathcal{L}(\mathbf{x}_1, \mathbf{a}_1, \mathbf{x}_2, \mathbf{a}_2, \dots, \mathbf{x}_H, \mathbf{a}_H), q(\mathbf{x}_1), q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{a}_t), H\} \quad (2.21)$$

Eine Aufgabe  $\mathcal{T}$  besteht dabei aus einer *Loss*-Funktion  $\mathcal{L}$ , einer Verteilung aus initialen Beobachtungen  $q(\mathbf{x}_1)$ , einer Transitionsverteilung  $q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{a}_t)$  sowie der Episodenlänge  $H$ , wobei für unabhängig identisch verteilte *supervised-learning* Probleme  $H = 1$  gilt.

<sup>4</sup> Siehe hierzu auch 2.4.1

Formal betrachtet ist das Ziel von MAML die Optimierung eines Modells  $f_\theta$  durch Anpassung seiner Modellparameter  $\theta$ :

$$\min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})}) \quad (2.22)$$

Dabei ist  $\mathcal{T}_i$  eine Aufgabe, welche aus einer Verteilung von Aufgaben  $p(\mathcal{T})$  gezogen wird,  $\mathcal{L}_{\mathcal{T}_i}$  ist der *Loss* einer bestimmten Aufgabe  $\mathcal{T}_i$  und  $\theta'_i$  sind die adaptierten Parameter für eine bestimmte Aufgabe.

Der MAML Algorithmus kann in die zwei Phasen *Inner Loop Learning* und *Outer Loop Learning* unterteilt werden. Um die Optimierung des Modells zu erzielen werden zunächst die Parameter  $\theta$  des Modells  $f_\theta$  zufällig initialisiert. Anschließend wird ein Satz von Aufgaben  $\mathcal{T}_i$  aus der Verteilung  $p(\mathcal{T})$  gezogen, also  $\mathcal{T}_i \sim p(\mathcal{T})$ . Angenommen es werden drei Aufgaben  $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3\}$  gezogen, dann werden für jede Aufgabe  $\mathcal{T}_i$  zunächst  $k$  *Samples* gezogen und anhand dieser *Samples* die Parameter  $\theta_i$  angepasst.

Die Adaption von  $\theta_i$  entspricht dabei dem *Inner Loop Learning*. Sie wird mittels *Backpropagation* durchgeführt, um die optimalen Parameter  $\theta'_i$  zu finden, die einen minimalen *Loss* für  $\mathcal{T}_i$  aufweisen. Die Aktualisierung der Parameter geschieht wie folgt, wobei  $\alpha$  ein *Hyperparameter* des MAML Algorithmus ist:

$$\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta}) \quad (2.23)$$

Nachdem für alle drei Aufgaben  $\mathcal{T}_i$  die Parameter mittels eines Gradientenabstieges aktualisiert wurden, existieren drei Parameter, die jeweils einen minimalen Fehler für die zugehörige Aufgabe aufweisen:  $\theta' = \{\theta'_1, \theta'_2, \theta'_3\}$ . Bevor nun die nächsten Aufgaben aus der Verteilung  $p(\mathcal{T})$  gezogen werden, wird zunächst das *Meta-Update* durchgeführt, was dem *Outer Loop Learning* entspricht. Dafür werden die zuvor zufällig initialisierten Parameter  $\theta$  in Bezug auf die optimalen Parameter  $\theta'$  aktualisiert. Dies führt dazu, dass die zuvor zufällig initialisierten Parameter  $\theta$  sich in Richtung einer optimalen Position bewegen, sodass beim Training der nächsten Reihe von Aufgaben nicht viele Gradientenschritte notwendig sind, um optimale Parameter zu erreichen. Dieser Schritt wird als *Meta-Training*, *Meta-Update* oder als *Meta-Optimierung* bezeichnet und entspricht der folgenden Gleichung, wobei  $\beta$  der *Meta Step Size* Hyperparameter ist:

$$\theta = \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) \quad (2.24)$$

Es sei anzumerken, dass  $\theta'_i$  keinen zusätzlichen Satz von Parametern darstellt. Vielmehr wird  $\theta'_i$  während des *Inner Loop Learnings*, dargestellt in Formel 2.23, berechnet, indem von  $\theta$  aus ein oder mehrere Gradientenschritte in Richtung einer speziellen Aufgabe  $\mathcal{T}_i$  durchgeführt werden. Das in Formel 2.22 dargestellte Optimierungsproblem entspricht hingegen dem *Outer Loop Learning* und wird mit den Parametern  $\theta'_i$  durchgeführt. Abstrakt betrachtet entspricht das *Inner Loop Learning* der Anpassung von  $\theta$  in Bezug auf eine bestimmte Aufgabe  $\mathcal{T}_i$ , wohingegen das *Outer Loop Learning*

der Optimierung der Modellparameter entspricht, sodass das Modell gut für alle Aufgaben  $\mathcal{T}_i \in \mathcal{T}$  abschneidet. [FAL17]

Das *Meta-Update* umfasst die Bestimmung eines Gradienten anhand eines Gradienten. Rechnerisch erfordert dies einen zusätzlichen *Backward Pass* durch  $f$ , um das Hessische Vektorprodukt zu berechnen. Da dies sehr rechenintensiv ist, haben Finn, Abbeel und Levine einige Experimente mit der Approximation von MAML erster Ordnung für *supervised Learning* Szenarien vorgestellt, bei denen die zweiten Ableitungen weggelassen werden. [FAL17]

In der Darstellung 3 ist der MAML Algorithmus anhand von Pseudocode exemplarisch nach Finn, Abbeel und Levine dargestellt. Zu Beginn werden neben einer Verteilung von Aufgaben  $p(\mathcal{T})$  noch die Hyperparameter  $\alpha$  und  $\beta$  benötigt. Anschließend werden die Modellparameter  $\theta$  zufällig initialisiert und das *Inner Loop* und *Outer Loop Learning* ausgeführt. Dabei werden die Modellparameter nach jedem Durchlauf der inneren Schleife aktualisiert. Der Algorithmus terminiert nach einem definierten Abbruchkriterium, wie z.B. das Erreichen maximaler Durchläufe der äußeren Schleife.

---

**Algorithm 3** Model-Agnostic Meta-Learning nach [FAL17]

---

**Require:**  $p(\mathcal{T})$  distribution over tasks

**Require:**  $\alpha, \beta$  step size hyperparameters

randomly initialize  $\theta$

**while** not done **do**

    Sample batch of tasks  $\mathcal{T}_i \sim p(\mathcal{T})$

**for all**  $\mathcal{T}_i$  **do**

        Evaluate  $\nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$  with respect to  $k$  examples

        Compute adapted parameters with gradient descent:  $\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$

**end for**

    Update  $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$

**end while**

---

In [AES18] haben Antoniou, Edwards und Storkey einige Nachteile von MAML und eine eigene Erweiterung namens „MAML++“ vorgestellt. So haben Antoniou, Edwards und Storkey herausgefunden, dass das Training mit MAML je nach Netzarchitektur des verwendeten ANN und der gewählten Hyperparameter instabil sein kann. In einem Beispiel konnten sie zeigen, dass das einfache Ersetzen eines *Max-Pooling Layers* durch einen *Strited Convolutional Layer* das Training mittels MAML instabil machte. Die Ergebnisse des Experiments über mehrere Epochen sind in Abbildung 2.21 dargestellt. Die Autoren vermuten, dass die Instabilität in tiefen ANN dadurch bedingt ist, dass bei dem Training mittels des klassischen MAML nur die finalen Gewichte des *Inner Loop Learnings* verwendet werden, wodurch es bei tiefen ANN zum *Vanishing and Exploding Gradients Problem* kommen kann, sodass die Gradienten mittels *Backpropagation* nicht mehr adäquat durch das Netz propagiert werden können. [AES18]

Eine mögliche Lösung für das zuvor beschriebe Problem von MAML ist die *Multi-Step Loss Optimization* (MSL). Dabei wird der Fehler der äußeren Schleife nach jedem Durchlauf der inneren Schleife berechnet. Entsprechend ergibt sich die Berechnung der Modellparameter  $\theta$  wie in Formel 2.25 dargestellt.

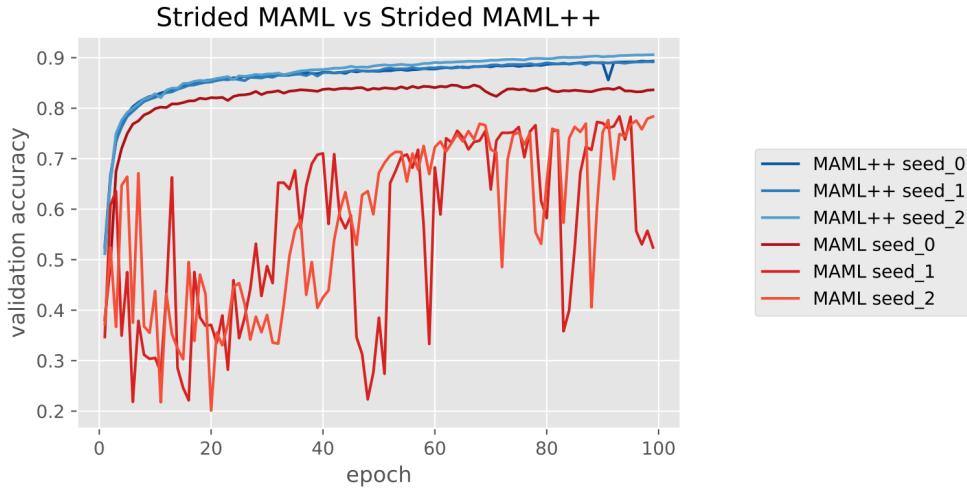


Abbildung 2.21.: Instabilität während des Trainings mittels MAML gegenüber MAML++ [AES18]

$$\theta = \theta - \beta \nabla_{\theta} \sum_{i=1}^B \sum_{j=1}^N w_j \mathcal{L}_{T_i}(f_{\theta_j^i}) \quad (2.25)$$

Dabei ist  $\beta$  eine Lernrate,  $\mathcal{L}_{T_i}(f_{\theta_j^i})$  ist der Fehler des *Outer Loop Learnings* von Aufgabe  $i$ , unter Verwendung der Basis-Netzgewichte  $\theta$  nach dem  $j$ -ten Durchlauf der inneren Schleife und  $w_j$  bezeichnet das Gewicht der äußeren Schleife im Durchlauf  $j$ . [AES18]

#### 2.4.3. First-Order Model-Agnostic Meta-Learning

Wie im vorherigen Unterabschnitt 2.4.2 bereits erwähnt, hat MAML zwei Nachteile. Zum einen ist das Training mittels MAML je nach Netztopologie und Hyperparameter instabil und zum anderen ist die Berechnung der zweiten Ableitungen für das *Meta-Update* sehr rechen- und speicherintensiv. Der *First-Order Model-Agnostic Meta-Learning* (FOMAML) Algorithmus ist eine Erweiterung des MAML Algorithmus, wobei vermieden wird, zweite Ableitungen zu verwenden. [Bis18]

Zur Erläuterung des FOMAML Algorithmus werden in der folgenden Gleichung  $k$  Gradientenschritte, mit  $k \geq 1$  und initialen Modellparametern  $\theta_{meta}$  durchgeführt:

$$\theta_0 = \theta_{meta} \quad (2.26)$$

$$\theta_1 = \theta_0 - \alpha \nabla_{\theta} \mathcal{L}^{(0)}(\theta_0) \quad (2.27)$$

$$\theta_2 = \theta_1 - \alpha \nabla_{\theta} \mathcal{L}^{(0)}(\theta_1) \quad (2.28)$$

$$\dots \quad (2.29)$$

$$\theta_k = \theta_{k-1} - \alpha \nabla_{\theta} \mathcal{L}^{(0)}(\theta_{k-1}) \quad (2.30)$$

In der äußeren Schleife, dem *Outer Loop Learning*, werden dann die Modellparameter wie folgt aktualisiert:

$$\theta_{\text{meta}} \leftarrow \theta_{\text{meta}} - \beta g_{\text{MAML}} \quad ; \text{Meta-Update} \quad (2.31)$$

Dabei ist

$$g_{\text{MAML}} = \nabla_{\theta} \mathcal{L}^{(1)}(\theta_k) \quad (2.32)$$

$$= \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k) \cdot (\nabla_{\theta_{k-1}} \theta_k) \dots (\nabla_{\theta_0} \theta_1) \cdot (\nabla_{\theta} \theta_0) \quad ; \text{Kettenregel} \quad (2.33)$$

$$= \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k) \cdot \prod_{i=1}^k \nabla_{\theta_{i-1}} \theta_i \quad (2.34)$$

$$= \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k) \cdot \prod_{i=1}^k \nabla_{\theta_{i-1}} (\theta_{i-1} - \alpha \nabla_{\theta} \mathcal{L}^{(0)}(\theta_{i-1})) \quad (2.35)$$

$$= \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k) \cdot \prod_{i=1}^k (I - \alpha \nabla_{\theta_{i-1}} (\nabla_{\theta} \mathcal{L}^{(0)}(\theta_{i-1}))) \quad (2.36)$$

Der MAML Gradient  $g_{\text{MAML}}$  ist dann:

$$g_{\text{MAML}} = \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k) \cdot \prod_{i=1}^k (I - \alpha \boxed{\nabla_{\theta_{i-1}} (\nabla_{\theta} \mathcal{L}^{(0)}(\theta_{i-1}))}) \quad (2.37)$$

FOMAML ignoriert hingegen den zweiten Ableitungsteil (in Formel 2.37 in der Box dargestellt). Es wird wie folgt vereinfacht, was der Ableitung des letzten Ergebnisses der Aktualisierung des inneren Gradienten entspricht:

$$g_{\text{FOMAML}} = \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k) \quad (2.38)$$

Finn, Abbeel und Levine konnten in [FAL17] zeigen, dass die Verwendung nur erster Ableitungen im Kontext der Klassifikation keine wesentlich schlechtere Performance als die Verwendung des klassischen MAML Algorithmus bewirkt.

#### 2.4.4. Reptile

*Reptile* ist ein von Nichol, Achiam und Schulman in [NAS18] vorgestellter Algorithmus des *Meta-Learnings*. Er basiert auf der Grundidee von FOMAML und ist in vielerlei Hinsicht ähnlich zu MAML, da er ebenfalls auf einer Meta-Optimierung mittels eines Gradientenabstieges basiert, ebenfalls eine Parameterinitialisierung lernt und ebenfalls Modell-agnostisch ist, unterscheidet sich jedoch in der Art von MAML, dass keine zweiten Ableitungen für das *Meta-Update* verwendet werden.

*Reptile* greift die Idee von MAML auf und führt diese weiter. So wird für eine spezielle Aufgabe  $\mathcal{T}_i$  zunächst SGD über ein paar Iterationen ausgeführt. Anschließend werden die Modellparameter,

welche später zur Initialisierung des Modells dienen, in Richtung der zuvor mittels SGD berechneten Parameter verschoben. Der *Reptile* Algorithmus in der *batch* Variante ist in 4 als Pseudocode dargestellt. Dabei führt  $SGD(\mathcal{L}_{\mathcal{T}_i}, \theta, k)$  eine stochastische Gradientenaktualisierung mittels SGD für  $k$  Schritte, für den *Loss*  $\mathcal{L}_{\mathcal{T}_i}$  und beginnend mit den Parametern  $\theta$  durch und liefert als Ergebnis einen Parametervektor  $W_i$ . In der *batch* Variante von *Reptile* werden mehrere Aufgaben  $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n$  statt nur einer Aufgabe pro Iteration verwendet. Der *Reptile* Gradient ist definiert als  $(\theta - W)/\alpha$ , wobei  $\alpha$  die von SGD verwendete Schrittweite ist. [NAS18]

---

**Algorithm 4** *Reptile, batched version* nach [NS18]

---

```

Initialize  $\theta$ 
for iteration= 1,2,... do
    Sample tasks  $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n$ 
    for  $i = 1, 2, \dots, n$  do
        Compute  $W_i = SGD(\mathcal{L}_{\mathcal{T}_i}, \theta, k)$ 
    end for
    Update  $\theta \leftarrow \theta + \beta \frac{1}{n} \sum_{i=1}^n (W_i - \theta)$ 
end for

```

---

Auf dem ersten Blick ähnelt der in 4 vorgestellte Algorithmus dem normalen SGD Algorithmus. Da die aufgabenspezifische Optimierung jedoch mehr als einen Schritt dauern kann, weicht das Ergebnis von SGD  $SGD(\mathbb{E}_{\mathcal{T}}[\mathcal{L}_{\mathcal{T}}], \theta, k)$  von  $\mathbb{E}_{\mathcal{T}}[SGD(\mathcal{L}_{\mathcal{T}}, \theta, k)]$  ab, wenn  $k > 1$ . Wenn die einzelnen Gradientenschritte für die Aufgaben  $\mathcal{T}_i$  jedoch nur mit einem *Sample*, also  $k = 1$ , durchgeführt werden, entspricht *Reptile* der Minimierung des erwarteten *Loss*, sodass gilt:

$$\mathbb{E}_{\mathcal{T}}[\nabla_{\theta} \mathcal{L}_{\mathcal{T}}(f_{\theta})] = \nabla_{\theta} \mathbb{E}_{\mathcal{T}}[\mathcal{L}_{\mathcal{T}}(f_{\theta})] \quad (2.39)$$

Werden jedoch mehrere Gradientenschritte durchgeführt, dann ist die durchschnittliche Aktualisierung nicht gleich zu der Aktualisierung mittels SGD. Die Gleichung

$$\mathbb{E}_{\mathcal{T}}[SGD(\mathcal{L}_{\mathcal{T}}, \theta, k)] \stackrel{?}{=} SGD(\mathbb{E}_{\mathcal{T}}[\mathcal{L}_{\mathcal{T}}], \theta, k) \quad (2.40)$$

gilt nur für  $k = 1$ . Für  $k > 1$  hängt der Erwartungswert der Aktualisierung (linke Seite der Gleichung 2.40) von den Ableitungen höherer Ordnung von  $\mathcal{L}_{\mathcal{T}}$  ab. Aus diesem Grund konvergiert *Reptile* zu einer Lösung, die sich stark von der Minimierung des erwarteten *Loss*  $\mathbb{E}_{\mathcal{T}}[\mathcal{L}_{\mathcal{T}}]$  unterscheidet. [NAS18]

Angenommen eine Aufgabe  $\mathcal{T}_i p(\mathcal{T})$  hat eine Vielzahl von optimalen Parameterkonfigurationen  $W_{\mathcal{T}}^*$ , dann erzielt das Modell  $f_{\theta}$  die beste Leistung für die Aufgabe  $\mathcal{T}_i$ , wenn  $\theta$  auf der Oberfläche von  $W_{\mathcal{T}}^*$  liegt. Um eine aufgabenübergreifende Lösung zu finden, soll aber eine Parameterkombination gefunden werden, die in der Nähe aller Aufgaben liegt:

$$\theta^* = \arg \min_{\theta} \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})} \left[ \frac{1}{2} \text{dist}(\theta, \mathcal{W}_{\mathcal{T}}^*)^2 \right] \quad (2.41)$$

Dabei wird als  $dist(\cdot)$  die  $L2$  Norm verwendet und die Distanz zwischen einem Punkt  $\theta$  und einem Set  $W^*_{\mathcal{T}}$  entspricht der Distanz zwischen  $\theta$  und einem Punkt  $W^*_{\mathcal{T}}(\theta)$ , der am nächsten zu  $\theta$  ist:

$$dist(\theta, \mathcal{W}_{\mathcal{T}}^*) = dist(\theta, W_{\mathcal{T}}^*(\theta)), \text{ where } W_{\mathcal{T}}^*(\theta) = \arg \min_{W \in \mathcal{W}_{\mathcal{T}}^*} dist(\theta, W) \quad (2.42)$$

Der Gradient der quadrierten euklidischen Distanz ist dabei:

$$\nabla_{\theta} \left[ \frac{1}{2} dist(\theta, \mathcal{W}_{\mathcal{T}_i}^*)^2 \right] = \nabla_{\theta} \left[ \frac{1}{2} dist(\theta, W_{\mathcal{T}_i}^*(\theta))^2 \right] \quad (2.43)$$

$$= \nabla_{\theta} \left[ \frac{1}{2} (\theta - W_{\mathcal{T}_i}^*(\theta))^2 \right] \quad (2.44)$$

$$= \theta - W_{\mathcal{T}_i}^*(\theta) \quad (2.45)$$

Daraus ergibt sich die Aktualisierungsregel von SGD nach einem Gradientenschritt wie folgt:

$$\theta = \theta - \alpha \nabla_{\theta} \left[ \frac{1}{2} dist(\theta, \mathcal{W}_{\mathcal{T}_i}^*)^2 \right] = \theta - \alpha (\theta - W_{\mathcal{T}_i}^*(\theta)) = (1 - \alpha)\theta + \alpha W_{\mathcal{T}_i}^*(\theta) \quad (2.46)$$

Der nächstgelegene optimale Punkt  $W_{\mathcal{T}_i}^*$  auf der Oberfläche für eine Aufgabe  $\mathcal{T}_i$  kann nicht genau berechnet werden. *Reptile* approximiert ihn jedoch unter Verwendung von  $SGD(\mathcal{L}_{\mathcal{T}}, \theta, k)$ . In Abbildung 2.22 ist die abstrakte Funktionsweise von *Reptile* dargestellt. Der Abbildung kann entnommen werden, dass *Reptile* versucht, einen optimalen Punkt im Raum mittels SGD zu finden, der möglichst nahe an allen Oberflächen für alle Aufgaben liegt. [NAS18]

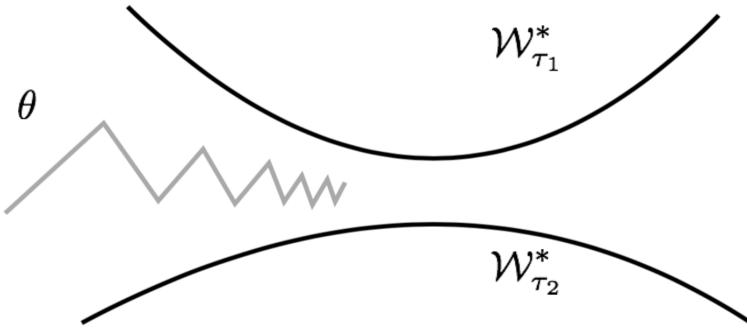


Abbildung 2.22.: Der *Reptile* Algorithmus aktualisiert die Modellparameter so, dass sie möglichst nahe an den Oberflächen aller Aufgaben liegen [NAS18]

#### 2.4.5. *Reptile* vs. FOMAML

Wie in Unterabschnitt 2.4.4 bereits erwähnt, sind sich *Reptile* und FOMAML sehr ähnlich, da beide Algorithmen, im Gegensatz zu MAML auf die Verwendung zweiter Ableitungen verzichten. Ein abstrakter Vergleich von FOMAML und *Reptile* ist in Abbildung 2.23 dargestellt. [Wen18]

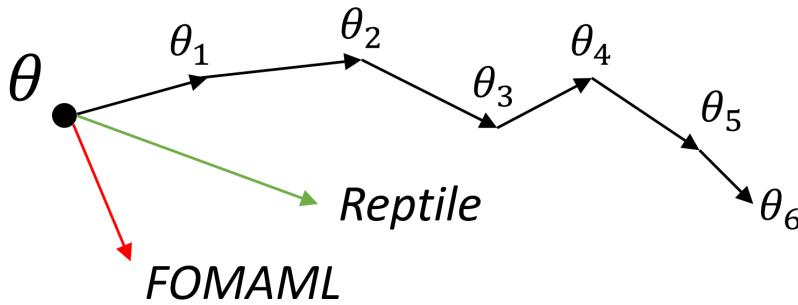


Abbildung 2.23.: *Reptile* vs. *FOMAML* im Durchlauf der Meta-Optimierung (eigene Darstellung nach [Wen18])

Algorithmus	1-shot 5-way	5-shot 5-way
MAML + Transduction	$48.70\% \pm 1.84\%$	$63.11\% \pm 0.92\%$
FOMAML + Transduction	$48.07\% \pm 1.75\%$	$63.15\% \pm 0.91\%$
<i>Reptile</i>	$47.07\% \pm 0.26\%$	$62.74\% \pm 0.37\%$
<i>Reptile</i> + Transduction	$49.97\% \pm 0.32\%$	$65.99\% \pm 0.58\%$

Tabelle 2.1.: Ergebnisse von MAML und Reptile auf dem Mini-ImageNet Datensatz nach [NAS18]

In [NAS18] konnten Nichol, Achiam und Schulman zeigen, dass *Reptile* eine bessere Leistung als MAML und FOMAML aufweist. In Tabelle 2.1 sind die Ergebnisse von MAML gegenüber *Reptile* für ein *1-shot 5-way* sowie *5-shot 5-way Meta-Learning* Problem dargestellt.

Zusammenfassend ist festzuhalten, dass sowohl MAML, FOMAML als auch *Reptile* darauf abzielen, eine bessere Aufgabenleistung und eine bessere Generalisierung zu erzielen. An dieser Stelle sei noch auf eine detaillierte Herleitung der Unterschiede und Gemeinsamkeiten von MAML, FOMAML sowie *Reptile* in [Wen18] und [NAS18] hingewiesen. In diesen Werken liefern Weng bzw. Nichol, Achiam und Schulman eine umfassende mathematische Herleitung der Verfahren.

## 2.5. Evaluation von Modellen

Es stellt sich die Frage, wie Algorithmen zur Erkennung von Anomalien evaluiert werden können. In der Praxis ist dies kein einfach zu lösendes Problem, da Ausreißer per Definition selten sind. Dies bedeutet, dass die Kennzeichnungen von Datenpunkten als Ausreißer oder normaler Datenpunkt als *Ground Truth* nicht zur Bewertung zur Verfügung stehen. Dies gilt insbesondere für Algorithmen des *unsupervised Learnings*, da, wenn eine *Ground Truth* zur Evaluation zur Verfügung stehen würde, diese für das Training eines mächtigeren *supervised* Algorithmus hätte genutzt werden können. [Agg17]

Bei der Evaluation von *unsupervised* Problemen, wie z.B. dem *Clustering*, werden daher häufig modellinterne Validierungen zur Bestimmung der Güte eines Modells verwendet. Ein häufig verwendetes Maß für die Güte eines *Clustering* Algorithmus ist beispielsweise der Radius des mittleren Quadrats eines *Clusters*. Das Problem bei diesem Ansatz ist jedoch, dass solche Validierungen nur eine Idee

über die Güte eines Modells liefern können. Es gibt also kein Modell, um die tatsächliche Güte eines *unsupervised* Algorithmus zu beurteilen. Das Paradoxe daran ist, dass, sollte ein solches Modell zur korrekten Bewertung eines Modells existieren, sollte dieses Modell eher zur Lösung des Problems, als zur Evaluierung eines schwächeren Modells genutzt werden. [Agg17]

Bei der Erkennung von Anomalien besteht ein (wenn auch unvollständiger) Ansatz darin, externe Qualitätsmaße zu verwenden. In einigen Fällen können Verfahren zur Bewertung von unausgeglichenen Klassifizierungsproblemen angewandt werden. Dabei stellt sich die Frage, wie die *Ground Truth* zur Bewertung des Modells herangezogen werden kann. Die meisten Algorithmen zur Erkennung von Anomalien geben einen *Outlier Score* aus, welcher mittels eines Schwellwertes in ein Ausreißer-Label konvertiert werden kann. Ist der Schwellwert dabei zu restriktiv gewählt, sinkt die Anzahl der als Ausreißer deklarierten Werte, was evtl. zur Erhöhung der fälschlicherweise als normal klassifizierten Werte (*False Negative*) führt. Ist der Schwellwert auf der anderen Seite zu klein gewählt, führt dies evtl. zu einer Erhöhung der fälschlicherweise als Anomalie klassifizierten Werte (*False Positive*). Anhand dieser beiden Werte können verschiedene Kennzahlen bestimmt werden, welche verschiedene Eigenschaften des Modells bewerten. [Agg17]

Die Bedeutung und Definitionen dieser Metriken sowie eine mögliche Art der Visualisierung sind in Unterabschnitt 2.5.1 sowie 2.5.2 dargestellt.

### 2.5.1. Metriken

Zur Beurteilung von Klassifikatoren wird häufig eine sog. *Confusion Matrix* verwendet, anhand derer Kennzahlen zur Bewertung der Eigenschaften des Modells bestimmt werden können [HKP17]. In [Agg17] hat Aggarwal das Konzept der *Confusion Matrix* aufgegriffen und anhand dessen Kennzahlen zur Beurteilung von Modellen zur Anomalieerkennung vorgestellt. Für eine Erläuterung der klassischen Kennzahlen der *Confusion Matrix* sei auf [HKP17] verwiesen.

Für einen gegebenen Schwellwert  $\lambda$  eines *Outlier Scores* wird der Satz von als Ausreißer deklarierter Daten mit  $S(\lambda)$  und der Satz von als normal deklarierten Daten mit  $N(\lambda)$  bezeichnet. Ändert sich  $\lambda$ , so ändert sich auch die Anzahl der zu  $S(\lambda)$  und  $N(\lambda)$  gehörigen Daten.  $G$  entspricht der *Ground Truth*, also der tatsächlichen Ausreißer in den Daten.  $A$  umfasst alle Daten, also sowohl Anomalien, als auch normale Daten. Für einen gegebenen Schwellwert  $\lambda$  ist die *Precision* eines Modells schließlich definiert als der prozentuale Anteil an korrekt erkannten Ausreißer deklarierter Werte:

$$\text{Precision}(\lambda) = 100 \cdot \frac{|S(\lambda) \cap G|}{|S(\lambda)|} \quad (2.47)$$

Der Wert der *Precision* ist in  $\lambda$  nicht unbedingt monoton, da sich sowohl der Zähler als auch der Nenner mit  $\lambda$  unterschiedlich ändern können. Der *Recall* ist dementsprechend definiert als der prozentuale Anteil der für einen Schwellwert  $\lambda$  als korrekt gekennzeichneten Daten  $S(\lambda)$  an der *Ground Truth*  $G$ :

$$\text{Recall}(\lambda) = 100 \cdot \frac{|S(\lambda) \cap G|}{|G|} \quad (2.48)$$

Durch Variation des Schwellwertes  $\lambda$  ist es möglich, eine Kurve zwischen der *Precision* und dem *Recall* zu zeichnen. Diese Kurve wird auch als *Precision-Recall Curve* (PRC) bezeichnet. Ein Beispiel ist in Abbildung 2.24 dargestellt. In der Praxis haben die Modelle häufig nur einen hohen Wert für eine der Kennzahlen und einen schlechten Wert für die jeweils andere.

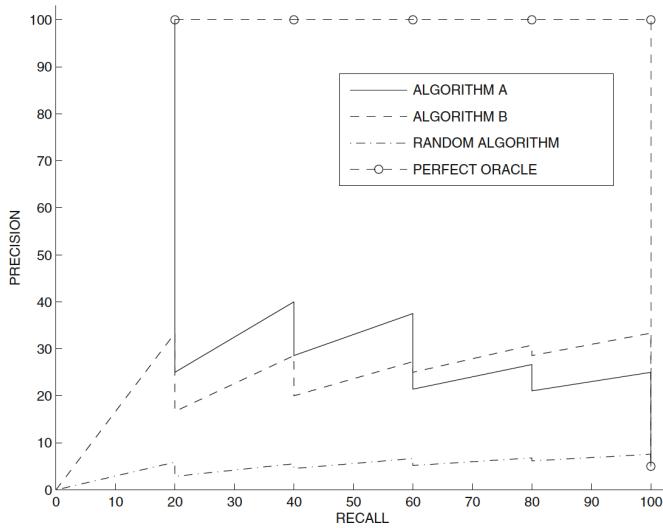


Abbildung 2.24.: Beispiel einer Precision-Recall Curve [Agg17]

Eine weitere häufig verwendete Kennzahl zur Bewertung eines Modells ist die *Accuracy*. Wohingegen die *Precision* die Performance eines Modells der Prognose aller positiven Klassen (Anomalien) misst, bestimmt die *Accuracy* die Gesamt-Performance des Modells über alle Klassen.

$$\text{Accuracy}(\lambda) = 100 \cdot \frac{|(S(\lambda) \cap G) \cup (N(\lambda) \cap (A \setminus B))|}{|A|} \quad (2.49)$$

## 2.5.2. Visualisierung

Die *Receiver Operating Characteristic Curve* (ROC) ist eng mit der PRC verwandt, manchmal visual jedoch intuitiver. Dabei wird die *True Positive Rate* (*Recall*) für einen Schwellwert  $\lambda$  gegenüber der *False Positive Rate* (*FPR*) dargestellt. Die FPR ist dabei der prozentuale Anteil der fälschlicherweise als Ausreißer deklarierten Werte an der negativen *Ground Truth*. Umgangssprachlich wird die FPR auch als *BadRecall* bezeichnet. Für einen Datensatz  $D$  mit einer *Ground Truth*  $G$  sind die Kennzahlen daher wie folgt definiert:

$$TPR(\lambda) = Recall(\lambda) = 100 \cdot \frac{|S(\lambda) \cap G|}{|G|} \quad (2.50)$$

$$FPR(\lambda) = BadRecall(\lambda) = 100 \cdot \frac{|S(\lambda) - G|}{|D - G|} \quad (2.51)$$

Ein Beispiel einer ROC ist in 2.25 dargestellt. Bei der ROC wird die  $FPR(\lambda)$  auf der X-Achse und die  $TPR(\lambda)$  entsprechend auf der Y-Achse dargestellt. Es ist zu beachten, dass sowohl die  $FPR(\lambda)$  als auch die  $TPR(\lambda)$  mit größeren Schwellwerten  $\lambda$  monoton ansteigen, da mehr Daten als Ausreißer klassifiziert werden. Daher liegen die Endpunkte der ROC immer bei  $(0, 0)$  und  $(100, 100)$ , und es wird erwartet, dass ein zufälliges Raten eine Leistung entlang einer Diagonalen dieser Punkte aufweist. Die Fläche oberhalb dieser Diagonalen entspricht dann einem Modell, das eine bessere Leistung als das reine Raten aufweist. [Agg17]

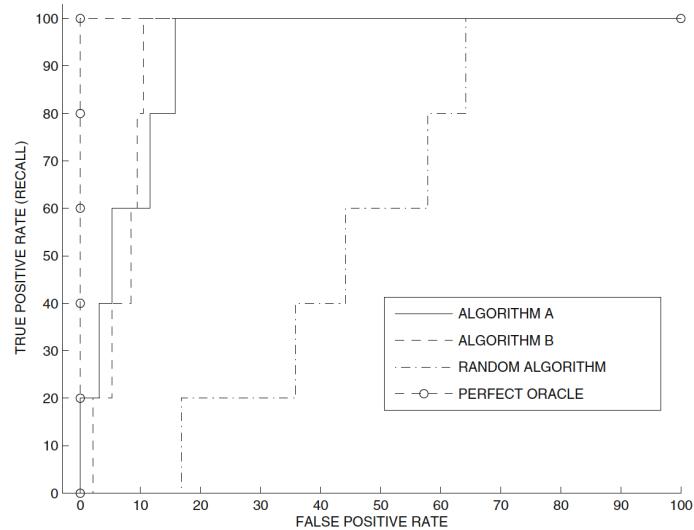


Abbildung 2.25.: Beispiel einer Receiver Operating Characteristic Curve [Agg17]

Als Kennzahl zur Bewertung der Güte eines Modells anhand der ROC wird die *Area Under Curve* (AUC) bestimmt. Wie der Name bereits suggeriert, handelt es sich dabei um die Fläche unter der ROC. Ein hoher AUC Wert eines Modells sagt aus, dass das Modell einen hohen *Recall* Wert und einen kleinen *FPR* für eine Anzahl möglicher  $\lambda$  aufweist. Die AUC kann auch als *Accuracy* des Modells interpretiert werden und liegt immer zwischen 0 und 1, wobei ein Modell mit einer *Accuracy* einem perfekten Modell entsprechen würde. [HKP17]



## 3. Konzeptionierung

Das Bundesministerium des Inneren, für Bau und Heimat definiert in ihrem Organisationshandbuch 2019 die Konzeptionierung als „die Entwicklung von umsetzungsfähigen Ansätzen zu den aufgedeckten Schwachstellen und Problembereichen. [...] In der Phase der Konzeption wird ein gezielter Weg zur Entwicklung der Lösungsansätze durchlaufen. Zu den ermittelten Schwachstellen werden Lösungsansätze erarbeitet.“ [Ref19] Adaptiert auf den Kontext der vorliegenden Arbeit sollen in den folgenden Abschnitten daher die Ansätze zur Untersuchung der in Abschnitt 1.2 definierten Forschungsfragen dargelegt werden. Zwar wurde in Abschnitt 1.2 bereits ein abstraktes eigenes Vorgehen vorgestellt, das Ziel dieses Kapitels ist jedoch die Entwicklung eines konkreten Plans zur Bearbeitung der Forschungsfragen. Dies beinhaltet zum einen die Planung zur Lösung der identifizierten Probleme durch ML-Ansätze und zum anderen die Planung des Versuchsaufbaus und -ablaufs sowie eine Auswahl an Metriken zur Evaluation des Versuchs. Ein Hauptbestandteil besteht daher in der Auswahl verschiedener Vorgehensweisen, Modelle und Metriken. So wird in Abschnitt 3.1 zunächst eine Analyse der Zielstellung der vorliegenden Arbeit vorgenommen. Anschließend wird in Abschnitt 3.2 die generelle Versuchsplanung dargelegt. Im darauffolgenden Abschnitt 3.3 wird die Durchführung einer Vorstudie beschrieben, im Rahmen derer eine Auswahl für eine Modellart zur Anomalieerkennung sowie für einen Datensatz getroffen wird. In Abschnitt 3.4 wird das Hauptexperiment zur Untersuchung der Forschungsfragen konzeptioniert. Dabei wird das Design des *Data Generators* sowie die Auswahl eines *Meta-Learning*-Algorithmus als Bestandteil des Experiments beschrieben. Abschließend wird in Abschnitt 3.5 die Auswahl der zu nutzenden Programmiersprache und Bibliotheken zur Durchführung des Experimentes elaboriert.

### 3.1. Analyse der Zielstellung

Anhand der in Abschnitt 1.2 definierten Forschungsfragen sollen in diesem Abschnitt zunächst die Thesen und anschließend überprüfbare Hypothesen aufgestellt werden, welche dann im weiteren Verlauf der vorliegenden Arbeit untersucht werden. Hierzu werden in Unterabschnitt 3.1.1 zunächst noch einmal die grundlegenden Forschungsfragen dargestellt. Anschließend werden in Unterabschnitt 3.1.2 notwendige Annahmen getroffen, um dann in Unterabschnitt 3.1.3 Thesen anhand der Forschungsfragen aufzustellen, welche dann abschließend in Unterabschnitt 3.1.4 in Hypothesen umgewandelt werden. Bei dem Aufbau bzw. der Definitionen der Begriffe „These“ und „Hypothese“ wurde [KR17] von Karmasin und Ribing zugrunde gelegt.

#### 3.1.1. Forschungsfragen

Für eine bessere Übersichtlichkeit sollen an dieser Stelle noch einmal die im Rahmen der vorliegenden Arbeit zu untersuchenden Forschungsfragen dargestellt werden. Die hier gezeigten Forschungsfragen entsprechen dabei den initial in Abschnitt 1.2 erläuterten Forschungsfragen.

**RQ1:** Können die Auswirkungen des *Concept Drifts* auf die Performance von Deep Autoencodern zur Anomalieerkennung in multivariaten Zeitreihen mittels *Meta-Learning*-Algorithmen minimiert bzw. verhindert werden?

**RQ2:** Welche Auswirkung hat das *Meta-Learning* auf die Performance des Autoencoders?

### 3.1.2. Annahmen

Nach Karmasin und Ribing bilden Annahmen gewissermaßen den Argumentationsrahmen, um Thesen und Hypothesen zu formulieren [KR17]. Mit Annahmen lassen sich Thesen und Hypothesen also erst präzise aufstellen. Während im Rahmen dieser Arbeit Thesen und Hypothesen aufgestellt und kritisch analysiert werden, bilden die Annahmen das Fundament dieser Prüfung. Im Folgenden werden daher zunächst die notwendigen Annahmen formuliert und grundlegend begründet.

**A1:** *Autoencoder können zur Identifizierung von Anomalien verwendet werden.*

Damit die Forschungsfragen untersucht werden können, muss angenommen werden, dass *Autoencoder* als ML-Modelle dafür geeignet sind, Anomalien in Zeitreihendaten zu erkennen. Im Allgemeinen kann diese Aussage angenommen werden, da in der Literatur eine Vielzahl von Ansätzen beschrieben wurden, die *Autoencoder* zur Anomalieerkennung verwenden. Als Beispiel sei hier z.B. [AC15] von An und Cho, [Fen+17] von Fengming et al. oder [Lee17] von Lee genannt. In allen drei Arbeiten konnten die Autoren zeigen, dass *Autoencoder* für die Identifizierung von Anomalien in verschiedenen Domänen, sowohl für Zeitreihen, als auch andere Daten, verwendet werden können.

**A2:** *Der Concept Drift hat negative Auswirkungen auf die Performance von Modellen zur Anomalieerkennung.*

Diese Annahme stellt die zweite zentrale Annahme zur Bearbeitung der Forschungsfragen dar. Es wird angenommen, dass ML-Modelle, welche *offline* trainiert wurden und anschließend für den Zweck der Vorhersage eingesetzt werden, einem *Concept Drift* unterliegen, welcher negative Auswirkungen auf die Leistung dieser Modelle hat. Diese Annahme kann durch die Arbeiten [Sau+18] von Saurav et al., [Chi19] von Chilakapati sowie [Tsy04] von Tsymbol begründet werden. In allen Arbeiten haben die Autoren gezeigt, dass ML-Modelle nach dem Training degenerieren, da sie einem *Concept Drift* unterliegen.

### 3.1.3. Thesen

Das Ziel von Thesen ist es, Behauptungen aufzustellen, welche anschließend mittels wissenschaftlicher Methoden falsifiziert werden können [KR17]. Im Folgenden sollen daher die für diese Arbeit relevanten Thesen aufgestellt und kurz erläutert werden.

**T1:** *Die negativen Auswirkungen des Concept Drifts auf die Performance von ML-Modellen können mittels Meta-Learning-Algorithmen minimiert werden.*

Die These T1 stellt den zentralen Untersuchungsgegenstand der vorliegenden Arbeit dar. Dabei soll untersucht werden, ob *Meta-Learning-Algorithmen* den durch A2 begründeten Effekt minimieren können.

**T2:** *Es ist möglich, die Effekte des Concept Drifts sowie die Auswirkungen des Meta-Learnings auf ein ML-Modell in einem Experiment zu simulieren und zu messen.*

Auf einer Metaebene soll untersucht werden, ob der generell geplante Versuchsaufbau in Form eines Experiments dafür geeignet ist, die Ursachen und Wirkungen zu simulieren und zu messen. Eine mögliche Falsifizierung von T2 durch ein Scheitern des Experiments, kann dabei nur streng auf den Kontext der vorliegenden Arbeit bezogen werden, da ein Scheitern von Faktoren, wie Zeit, Wissen oder Ressourcen, abhängt. Eine Falsifizierung der These T2 führt nicht zwangsläufig zu einer Falsifi-

zierung von  $T1$ . So können beide Thesen im Rahmen der vorliegenden Arbeit gemeinsam wahr sein, es ist jedoch nicht möglich, dass  $T1$  wahr und  $T2$  falsch ist, wodurch sich eine Implikation der Form  $T1 \Rightarrow T2$  ergibt. Darüber hinaus kann  $T2$  als notwendige Voraussetzung für  $T1$  gesehen werden, um im Rahmen der vorliegenden Arbeit  $T1$  zu untersuchen.

### 3.1.4. Hypothese

Hypothesen sind eine Unterform von Thesen und stellen einen Zusammenhang zwischen mindestens zwei Faktoren dar. Dabei liegt der Fokus mehr auf einer Ursache-Wirkung-Beziehung, welche in Form von „Wenn-Dann“- oder „Je-Desto“-Formulierungen ausgedrückt werden kann [KR17]. Im Folgenden werden die aufgestellten Thesen als überprüfbare Hypothesen formuliert.

***H1: Wenn Meta-Learning einen positiven Einfluss auf die Performance eines Modells hat, dann ist der Fehler  $\epsilon_{M_1}$  eines Modells  $M_1$ , welches mittels Meta-Learning adaptiert wird, kleiner als der Fehler  $\epsilon_{M_2}$  eines Modells  $M_2$ , welches nicht mittels Meta-Learning adaptiert wird, also  $\epsilon_{M_1} < \epsilon_{M_2}$ .***

Die Hypothese  $H1$  leitet sich von der These  $T1$  ab, welche als Voraussetzung dient. Im Rahmen verschiedener Experimente soll  $H1$  dahingegen analysiert werden, ob ein Modell, welches mittels *Meta-Learning* an einen *Concept Drift* adaptiert wird, eine bessere *Performance* aufweist, als ein Modell, welches nach einem ersten Training nicht weiter verändert wird.

## 3.2. Versuchsplanung

Die Grundidee zur Bearbeitung der Forschungsfragen  $RQ1$  und  $RQ2$ , der Thesen  $T1$  und  $T2$  sowie der Hypothese  $H1$  ist die Durchführung eines Experiments und die anschließende Auswertung der Messungen zur Ableitung von Ergebnissen. Die Durchführung eines Experimentes ermöglicht es, die Effekte des *Concept Drifts* und *Meta-Learnings* zu messen und zu bewerten. Ziel der folgenden Unterabschnitte ist die Erläuterung der allgemeinen Versuchsplanung. Dazu wird in Unterabschnitt 3.2.1 zunächst der allgemeine Ablauf eines Experimentes inkl. der relevanten Phasen erläutert. Im anschließenden Unterabschnitt 3.2.2 wird dann der allgemeine Versuchsablauf beschrieben. In 3.2.3 wird der Aufbau des Versuchs erläutert. Im abschließenden Unterabschnitt 3.2.4 werden Metriken zur Evaluation ausgewählt sowie mögliche Zeitpunkte zur Messung der Ergebnisse beschrieben.

### 3.2.1. Experimentieren

Die Durchführung von Experimenten hat insbesondere in den Bereichen des wissenschaftlichen Rechnens und der angewandten Mathematik in den letzten Jahren deutlich an Bedeutung gewonnen, da aufgestellte Hypothesen im Rahmen von Experimenten strukturiert untersucht werden können [Feh+16]. Der generelle Ablauf eines Experimentes in der Informatik nach Steinkamp ist in Abbildung 3.1 dargestellt. Als Ausgangssituation dienen Hypothesen und Fragestellungen, wie sie in  $RQ1$  und  $RQ2$  definiert sind. Davon ausgehend wird in den folgenden Phasen zunächst das Experiment vorbereitet und anschließend durchgeführt. Die Ergebnisse eines Experimentes werden ausgewertet und können in zukünftigen Experimenten wiederum in Hypothesen und Fragestellungen mit einfließen. Die Vorbereitung eines Experimentes umfasst verschiedene Phasen, wie den Aufbau, die Konfiguration und die Auswahl der Beobachtungsaspekte. Die Planung dieser Phasen wird in den folgenden Unterab-

schnitten dieses Kapitels beschrieben. Die Phasen des eigentlichen Experimentes, die Ablaufsteuerung bzw. Interaktion sowie die Beobachtung, welche während der Durchführung eines Experimentes stattfinden, werden ebenfalls im Rahmen dieses Kapitels geplant und beschrieben.

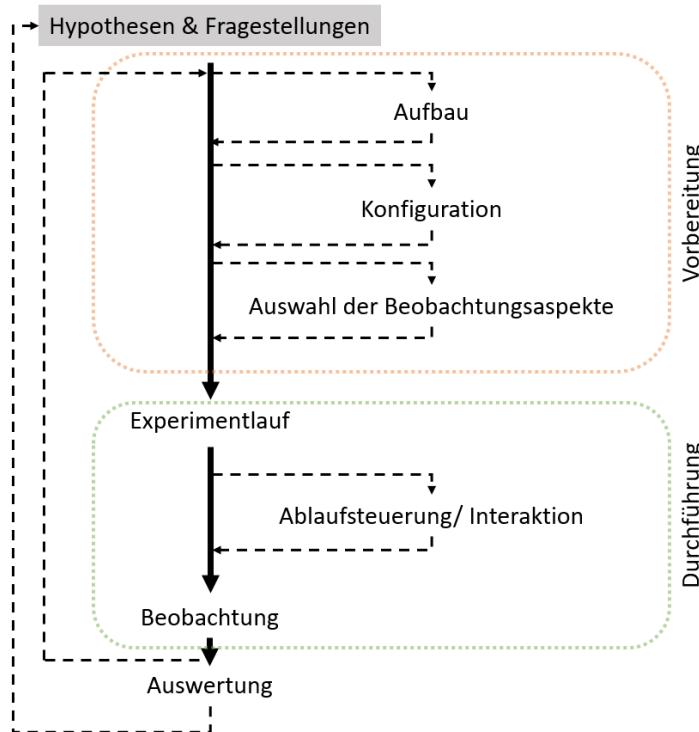


Abbildung 3.1.: *Aktivitätszyklus beim Experimentieren in der Informatik (eigene Darstellung nach [Ste99])*

### 3.2.2. Versuchsablauf

Die Durchführung des Versuchs besteht im Wesentlichen aus den zwei Phasen der Vorstudie sowie der Ausführung des Experimentes. Im Rahmen der Vorstudie sollen erste Erkenntnisse gesammelt werden, um den Raum möglicher Kombinationen für die zweite Phase, dem eigentlichen Experiment, einzuschränken. Dies umfasst beispielsweise die Auswahl einer möglichen Topologie eines *Autoencoders* oder die Anzahl an Datensätzen zwischen zwei *Meta-Learning* Phasen. In der zweiten Phase, dem eigentlichen Experiment, wird dann mittels synthetischer Daten der Effekt des *Concept Drifts* auf *Autoencoder* simuliert und gemessen. Anschließend wird untersucht, ob mittels *Meta-Learning* der Effekt des *Concept Drifts* minimiert werden kann. Der geplante Versuchsaufbau ist in Abbildung 3.2 als Aktivitätsdiagramm dargestellt. Dabei wird zwischen den drei Phasen der Konzeptionierung, der Vorstudie sowie des eigentlichen Experimentes unterschieden. In jeder der Phasen werden verschiedene Tätigkeiten als Vorbereitung für die darauffolgende Phase durchgeführt und dokumentiert.

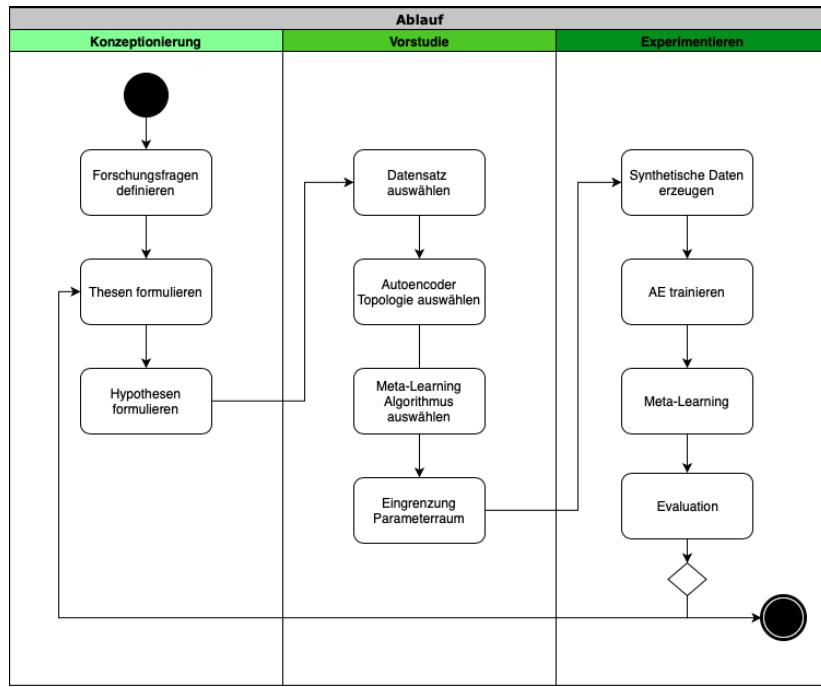


Abbildung 3.2.: Aktivitätsdiagramm des gesamten Ablaufes inkl. Konzeptionierung, Vorstudie und Experiment (eigene Darstellung)

### 3.2.3. Versuchsaufbau

In Abbildung 3.3 ist der schematische Versuchsaufbau zur Evaluierung des *Meta-Learning*-Ansatzes dargestellt. Im Rahmen eines ersten Trainings werden zwei Modelle  $M_1$  und  $M_2$  auf einem Trainingsdatensatz  $\mathbf{X}_{train}$  trainiert.  $M_1$  wird dabei mit einem *Meta-Learning*-Algorithmus trainiert,  $M_2$  wird hingegen auf gewöhnliche Weise mit einem normalem Trainingsalgorithmus trainiert. Anschließend können in  $t_1$  beide Modelle hinsichtlich ihrer Performance auf dem Datensatz  $\mathbf{X}_{train}$  evaluiert werden. Beide Modelle werden darauffolgend auf einem Datensatz  $\mathbf{X}_{drifted}$ , welcher mittels des *Data Generators* so manipuliert wurde, dass ein *Concept Drift* entsteht, in Zeitpunkt  $t_2$  evaluiert. Beide Modelle  $M_1$  und  $M_2$  werden nachfolgend mittels  $k$  *Samples* an  $\mathbf{X}_{drifted}$  adaptiert und in  $t_3$  erneut evaluiert. Die Adaption der Modelle an  $\mathbf{X}_{drifted}$  mittels der  $k$  *Samples* sollte dafür sorgen, dass die Modelle weniger *Samples* fälschlicherweise als Anomalie klassifizieren. Auf einem dritten Datensatz  $\mathbf{X}_{drifted,ano}$ , welcher neben einem *Concept Drift* auch Anomalien umfasst, werden beide Modelle in  $t_4$  abschließend erneut evaluiert. Das Vorgehen bei dem verschiedene Datensätze zur Adaption und Evaluation verwendet werden sowie zwei Modelle, welche initial unterschiedlich trainiert wurden, ermöglicht es, die Auswirkungen des *Concept Drifts* und des *Meta-Learnings* zu untersuchen.

### 3.2.4. Evaluation

Die Ergebnisse der Evaluation stellen ein zentrales Ergebnis der vorliegenden Arbeit dar. Daher ist die Planung der Evaluation ein wichtiger Bestandteil der Konzeptionierung. Bei der Planung sind dabei zwei Aspekte zu berücksichtigen: zum einen sind geeignete Metriken auszuwählen, anhand derer verschiedene Modelle sinnvoll bewertet werden können. Zum anderen spielt der Zeitpunkt der

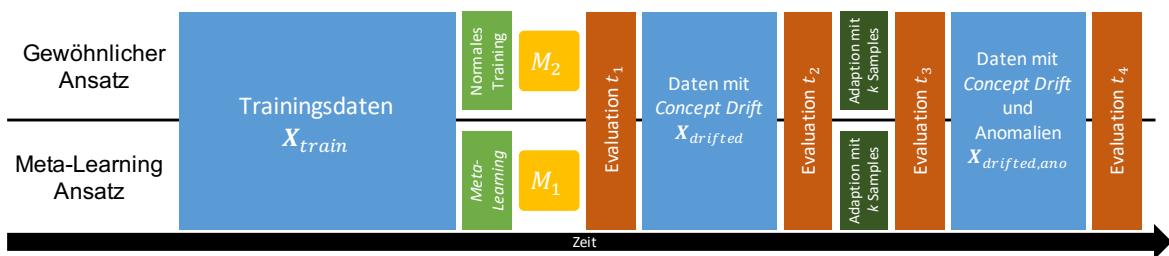


Abbildung 3.3.: Versuchsaufbau zur Evaluierung des Meta-Learning-Ansatzes gegenüber keinem Meta-Learning (eigene Darstellung)

Messung eine wichtige Rolle. Neben der Auswahl geeigneter Metriken und Messzeitpunkte stellt die Nachvollziehbarkeit einen wichtigen Aspekt dar. So ist es im Rahmen der Evaluation wichtig, dass zu jedem Zeitpunkt sowohl die Ergebnisse der Modelle, als auch die Berechnungen der Metriken transparent dargestellt werden.

## Metriken

Bei der Evaluation der Modelle bzw. Experimente sollen die in Abschnitt 2.5 eingeführten Metriken verwendet werden. Dabei handelt es sich insbesondere um die *Accuracy*, *Precision*, *Specificity* sowie die *Sensitivity* und den *F1-Score*. Zur Visualisierung der Ergebnisse können neben der *Confusion Matrix* auch die sog. *Precision-Recall Curve* und *Receiver Operation Characteristic Curve* verwendet werden. Da sowohl bei der Vorstudie, als auch bei der Durchführung der Experimente die *Label* bekannt sind, können diese Kennzahlen ohne Probleme bestimmt werden. Diese Art der Evaluation von ML-Modellen zur Anomalieerkennung stellt ein gängiges Vorgehen dar, wie u.a. von Gulenko et al. in [Gul+16] dargestellt wurde.

## Messzeitpunkte

Die Auswahl geeigneter Messzeitpunkt richtet sich nach dem in Abbildung 3.3 beschriebenen Ablauf. So wird in Zeitpunkt  $t_1$  unmittelbar nach dem Training die Performance der Modelle  $M_1$  und  $M_2$  hinsichtlich ihrer Leistung auf den Trainingsdaten  $X_{train}$  bewertet. In  $t_2$  werden beide Modelle dann auf einem Datensatz  $X_{drifted}$  bezüglich eines *Concept Drifts* untersucht. Im darauf folgenden Zeitpunkt  $t_3$ , nachdem beide Modelle mit wenigen *Samples* an  $X_{drifted}$  adaptiert wurden, wird ihre Performance erneut evaluiert. In  $t_4$  werden beide Modelle auf einem Datensatz  $X_{drifted,anomal}$ , welcher sowohl einen *Concept Drift*, als auch Anomalien umfasst, abschließend bewertet.

### 3.3. Vorstudie

Das Ziel der Vorstudie ist die Vorauswahl möglicher Parameter für die Durchführung der anschließenden Experimente. Im Rahmen der Vorstudie sollen daher verschiedene Parameterkombinationen, wie die Auswahl einer geeigneten AE-Topologie oder Messzeitpunkte zur Bestimmung der Metriken, evaluiert werden. Hierzu wird in Unterabschnitt 3.3.1 zunächst ein geeigneter Datensatz ausgewählt, auf dem die Einschränkung der Parameterkombinationen vorgenommen wird. Im darauffolgenden Unterabschnitt 3.3.2 wird die Problemcharakteristik der Anomalieerkennung für die Vorstudie dar-

gestellt. Im abschließenden Unterabschnitt 3.3.3 wird eine Vorauswahl geeigneter AE-Topologien diskutiert.

### 3.3.1. Auswahl eines geeigneten Datensatzes

In der Literatur wird eine Vielzahl unterschiedlicher Datensätze für die Evaluation und den Vergleich von Modellen zur Anomalieerkennung verwendet. Um die Auswahl einzuschränken, sollen daher zunächst Kriterien vorgestellt, beschrieben und gewichtet werden. Anschließend werden verschiedene potenzielle Datensätze vorgestellt, bevor abschließend anhand der Gewichte eine Auswahl für einen Datensatz vorgenommen wird.

#### Art der Daten

Damit im Rahmen der Vorstudie bereits eine geeignete AE-Topologie evaluiert und ausgewählt wird, welche anschließend im Rahmen der Experimente zur Erkennung von Anomalien in multivariaten Zeitreihen genutzt werden kann, sollte der Datensatz demnach möglichst multivariate Zeitreihen umfassen.

#### Bekanntheit

Die Bekanntheit des Datensatzes spielt eine Rolle, da eine große Bekanntheit des Datensatzes viele mögliche Ideen für unterschiedliche ML-Modelle und Evaluationsmöglichkeiten bietet und die Datensätze in vielen Publikationen verwendet werden. Darüber hinaus bietet eine große Bekanntheit die Möglichkeit, die im Rahmen der Vorstudie erstellten Modelle mit anderen Modellen aus der Literatur zu vergleichen.

#### Dokumentation

Eine gute Dokumentation des Datensatzes umfasst beispielsweise die Beschreibung der einzelnen *Features* sowohl aus semantischer, als auch aus statistischer Sicht.

#### Verfügbarkeit/ Lizenz

Im Rahmen der Nachvollziehbarkeit und Reproduzierbarkeit sollte der Datensatz möglichst leicht verfügbar sein. Dies umfasst auch die Lizenz, unter der der Datensatz veröffentlicht wurde, sodass der Datensatz im Rahmen der vorliegenden Arbeit überhaupt verwendet werden darf.

#### Größe des Datensatzes

Da im Rahmen der vorliegenden Arbeit Modelle zur Erkennung von Anomalien aus dem Bereich des *Deep Learnings* untersucht werden sollen, muss der Datensatz eine gewisse Größe (Anzahl an *Samples* und *Features*) aufweisen, damit die Modelle mit ausreichend Daten trainiert werden können.

Im Rahmen einer durchgeführten Präferenzanalyse<sup>1</sup> wurden die oben beschriebenen Kriterien jeweils gegeneinander abgewogen. In Abbildung 3.4 ist das Ergebnis der Gewichtung der einzelnen Kriterien dargestellt. Dabei ist zu sehen, dass die Art der Daten den größten Einfluss bei der Auswahl eines Datensatzes hat, gefolgt von der Größe des Datensatzes, der Dokumentation, Verfügbarkeit sowie Bekanntheit der Daten.

Im Rahmen einer Recherche wurden mehrere potenzielle Datensätze identifiziert. Bei der Vorauswahl wurden nur Datensätze berücksichtigt, die den Rahmenbedingungen entsprechen, also z.B. auch im Rahmen der vorliegenden Arbeit verwendet werden dürfen.

<sup>1</sup> Die Präferenzanalyse ist Teil der Nutzwertanalyse. Dabei kann anhand mehrerer Kriterien eine Präferenz bzw. Gewichtung der Kriterien erstellt werden, indem alle Kriterien gegeneinander bewertet werden. [KKH02]

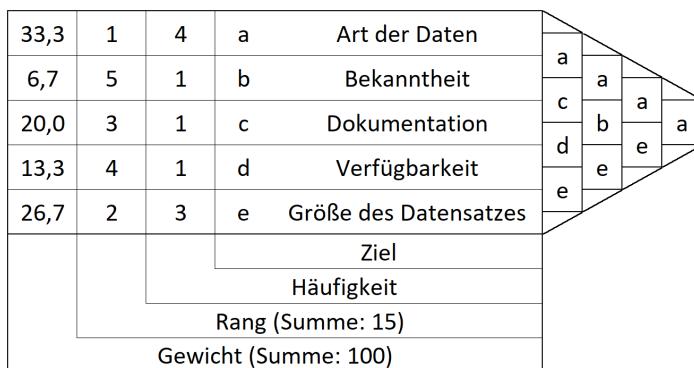


Abbildung 3.4.: Bestimmung der Gewichte der fünf Kriterien als Ergebnis einer Nutzwertanalyse (eigene Darstellung)

### GHL Datensatz

In [FLV16] haben Filonov, Lavrentyev und Vorontsov einen Datensatz einer Anlage zur Erhitzung und dem Transport von Gasöl mit dem Namen *GHL dataset* vorgestellt.<sup>2</sup> Den Datensatz haben Filonov, Lavrentyev und Vorontsov dabei anhand eines Modells einer Anlage erstellt. Anschließend haben die Autoren Anomalien in Form von unautorisierten Änderungen in der Anlage (z.B. das Öffnen eines Ventils) erzeugt. Das Ergebnis ist eine große Anzahl unterschiedlicher Zeitreihen von Messwerten, wie z.B. der Leitungs- oder Öl-Temperatur.

### SWaT Datensatz

Goh et al. haben in [Goh+17] die Erstellung eines Datensatzes einer Wasseraufbereitungsanlage vorgestellt. Dabei haben die Autoren eine reale Anlage mit Sensorik und einem Kommunikationsnetz ausgestattet. Die Anlage wurde dann über einen Zeitraum betrieben und die Daten aufgezeichnet. Anomalien wurden durch verschiedene Events, wie z.B. Cyber-Angriffen auf die Kommunikationsinfrastruktur erzeugt. So besteht der Datensatz aus insgesamt 51 *Features* mit 946.722 *Samples*, welche über einen Zeitraum von mehreren Tagen aufgezeichnet wurden.

### Power System Datensatz

Bei dem *Power System* Datensatz handelt es sich um einen von Hink et al. in [Hin+14] verwendeten Datensatz. Dieser wurde von Hink et al. mithilfe eines einfachen Energiesystemmodells, bestehend aus zwei Generatoren und mehreren Schaltern und Intelligent Electronic Devices (IED), erstellt. Insgesamt beinhaltet der Datensatz 128 Messwerte aus dem Energiesystem und 37 unterschiedliche Arten von Events, wie z.B. Ausfälle oder Angriffe.

In der Tabelle 3.1 ist das Ergebnis der Nutzwertanalyse dargestellt. Im Rahmen der Auswahl wurde den einzelnen Datensätzen pro Kriterium jeweils ein Wert zwischen 0 und 5 zugewiesen. Anschließend wurden die verteilten Punkte mit der jeweiligen Gewichtung des Kriteriums multipliziert und der Wert aller gewichteten Kriterien pro Datensatz als Score summiert. Das Ergebnis der Nutzwertanalyse zeigt, dass der *Power System* Datensatz mit 4,40 Punkten am besten abgeschnitten hat, gefolgt von dem GHL Datensatz mit 3,70 und dem SWaT Datensatz mit 3,53 Punkten. Im Rahmen der Vorstudie soll daher der *Power System* Datensatz verwendet werden.

<sup>2</sup> im Original *gasoil heating loop dataset*

<b>Kriterium</b>	<b>Gewichtung</b>	<b>GHL</b>		<b>SWaT</b>		<b>Power Systems</b>	
		ungew.	gew.	ungew.	gewichtet	ungew.	gew.
Art der Daten	0,333	4	1,332	4	1,332	5	1,665
Bekanntheit	0,067	3	0,201	3	0,201	3	0,201
Dokumentation	0,2	4	0,8	2	0,4	4	0,8
Verfügbarkeit	0,133	4	0,532	4	0,532	5	0,665
Größe	0,267	3	0,801	4	1,068	4	1,068
<b>Summe</b>	<b>1</b>	<b>18</b>	<b>3.70</b>	<b>17</b>	<b>3,53</b>	<b>21</b>	<b>4,40</b>

Tabelle 3.1.: *Ergebnisse der Auswahl eines Datensatzes im Rahmen der Nutzwertanalyse*

### 3.3.2. Beschreibung der Problemcharakteristik

Im Folgenden wird die Problemcharakteristik der Anomalieerkennung nach Chandola, Banerjee und Kumar aus [CBK09] beschrieben. Hierfür wird das Problem der Anomalieerkennung im Rahmen der Vorstudie anhand der Kriterien des Schemas nach Chandola, Banerjee und Kumar bewertet, damit anschließend anhand dieser Bewertung ein geeignetes Modell zur Anomalieerkennung gewählt werden kann.

#### Natur der Daten

Bei den Daten handelt es sich um multivariate Zeitreihen in Form von Messwerten und Zustandsbeschreibungen aus einem Modell eines Energiesystems. Die Simulation des Modells wurde dabei über einen Zeitraum von mehreren Wochen durchgeführt und die Messwerte aufgezeichnet. Während der Simulation wurden darüber hinaus verschiedene Angriffe, wie z.B. *Data Injections* oder fehlerhafte Fernauslösebefehle, durchgeführt.

#### Label

Die Datensätze sind jeweils für jeden Zeitpunkt mit einem *Label* versehen. Dabei handelt es sich um einen numerischen Wert zwischen 0 und 37, der bestimmt, welche Art der Anomalie ein *Sample* darstellt.

#### Art der Anomalie

In dem Datensatz sind alle der in 2.3.2 definierten Anomalien enthalten, da unterschiedliche Ursachen bzw. Effekte unterschiedliche Arten von Anomalien verursachen. So sind beispielsweise einzelne *Point Anomalies* durch fehlerhafte Messwerte eines Sensors oder *Collective Anomalies* durch *Data Injection* Angriffe in dem Datensatz enthalten.

#### Output

Der *Output* des Modells wird maßgeblich von der Art des Modells bestimmt. Da im Rahmen der Vorstudie u.a. *Autoencoder* evaluiert werden sollen, handelt es sich bei den Outputs um sog. *Outlier Scores*, die den Grad einer Anomalie bestimmen.

### 3.3.3. Auswahl eines Modells zur Anomalieerkennung

Da im Rahmen der Vorstudie und der Experimente Anomalien in multivariaten Zeitreihen identifiziert werden sollen, bietet es sich an, auf rekurrenten neuronalen Netzen basierende *Autoencoder* zu

verwenden. Dabei werden häufig auf LSTM oder GRU basierende AE verwendet, da diese besonders geeignet sind, Langzeitbeziehungen in Zeitreihen zu lernen [GBC16]. Die Vorauswahl einer Basis für ein AE Modell ist jedoch selbst Ziel der Vorstudie und kann daher an dieser Stelle nicht final beantwortet werden.

### 3.4. Experiment

Wurden im Rahmen der Vorstudie erste Parameter evaluiert, um eine valide Vorauswahl für die Experimente zu treffen, ist das Ziel des Experimentierens die Untersuchung der definierten Forschungsfragen. Im Rahmen des Experimentes sollen dafür die aufgestellten Hypothesen analysiert werden. Im Folgenden werden die für die Durchführung der Experimente relevanten Planungen erläutert. In Unterabschnitt 3.4.1 wird hierfür zunächst die Planung des generellen Experimentablaufes beschrieben. Anschließend wird in Unterabschnitt 3.4.2 der Rahmen zur Beschreibung einer Konfiguration für ein Experiment erläutert. Im darauffolgenden Unterabschnitt 3.4.3 wird die Beschreibung der Problemcharakteristik in der Anomalieerkennung dargelegt. Zuletzt wird in Unterabschnitt 3.4.4 das Design des *Data Generators* erläutert.

#### 3.4.1. Planung der Experimente

Die Planung eines Experimentierlaufes wurde bereits in Abbildung 3.3 erläutert. Zusätzlich zu dem dort genannten Ablauf sind vor einem Durchlauf zunächst einige Vorbereitungen durchzuführen. Dies umfasst beispielsweise die Konfiguration eines Experimentes durch Definition der entsprechenden Parameter, welche im folgenden Unterabschnitt weiter beschrieben werden.

#### 3.4.2. Konfiguration eines Experimentes

Ein Durchlauf eines Experimentes wird von verschiedenen Parametern bestimmt, welche im Rahmen der Vorbereitung definiert werden. Die Menge der konkreten Parameterbelegungen bestimmt dann die Konfiguration eines Experimentes. Im Folgenden werden die unterschiedlichen Parameter kurz aufgelistet und erläutert.

- ***Meta-Learning-Algorithmus***

Auswahl eines zugrundeliegenden *Meta-Learning*-Algorithmus, wie z.B. MAML, FOMAML oder *Reptile*.

- ***Hyperparameter des Meta-Learning-Algorithmus***

Dies umfasst alle möglichen *Hyperparameter* des *Meta-Learning*-Algorithmus wie z.B. Lernraten oder Anzahl an *Samples*, die für einen *Meta-Learning* Schritt verwendet werden.

- ***Metriken zur Evaluation der Modelle***

Auswahl der Metriken, die für eine Messung berechnet werden und anhand derer die Modelle miteinander verglichen werden sollen.

- ***Art und Stärke des Concept Drifts***

Anzahl der Änderungen sowie Stärke der Änderungen in den Daten, die zu einem *Concept Drift* führen.

- **Zeit zwischen Messungen**

Anzahl an *Samples* zwischen zwei Messzeitpunkten  $t_n$  und  $t_{n+1}$ .

- **Anzahl der Messungen**

Legt fest, wie oft Kennzahlen zur Evaluation berechnet werden sollen. Zusammen mit der Zeit zwischen zwei Messungen und der Anzahl der Messungen ergibt sich die Dauer eines Experiments.

### 3.4.3. Beschreibung der Problemcharakteristik

Bei den Experimenten handelt es sich um ein anderes ML-Problem, als das im Rahmen der Vorstudie definierte Problem. Da die Vorstudie jedoch zur Vorauswahl und Einschränkung der Parameter durchgeführt wird und sich daher an den Experimenten orientiert, sind beide Problemcharakteristiken identisch. Daher sei an dieser Stelle nur auf die Beschreibung der Problemcharakteristik in 3.3.2 hingewiesen.

### 3.4.4. Design des *Data Generators*

Der Datengenerator ist ein wichtiger Bestandteil für die Durchführung der Experimente. Mittels des Generators sollen Daten erzeugt werden, die für ein Training eines Modells zur Erkennung von Anomalien verwendet werden. Darüber hinaus sollen mithilfe des Datengenerators die Daten so verändert werden können, dass ein *Concept Drift* entsteht. Im Folgenden werden zunächst die Anforderungen an den Datengenerator dargestellt, bevor abschließend das eigentliche Design erläutert wird.

#### Anforderungen

Die Anforderungen an den Datengenerator können in funktionale (FR) und nicht funktionale Anforderungen (NFR) unterteilt werden. FR stellen dabei direkt Anforderungen bzgl. der Funktionalität des Datengenerators, wohingegen NFR Anforderungen an die Qualitätseigenschaften stellen.

##### Funktionale Anforderungen:

- *FR1*: Es müssen multivariate Zeitreihen erstellt werden können.
- *FR2*: Es muss möglich sein, einen *Concept Drift* zu erzeugen.
- *FR3*: Die Art des *Concept Drifts* muss gesteuert werden können.
- *FR4*: Die Stärke des *Concept Drifts* muss gesteuert werden.
- *FR5*: Es muss möglich sein, alle Arten der in 2.3.2 vorgestellten Anomalien in die Daten zu bringen.
- *FR6*: Die Erzeugung der Daten muss nachvollziehbar und reproduzierbar sein.
- *FR7*: Die Daten müssen nach der Erstellung gelabelt sein, d.h. die Art, die Dauer und der Zeitpunkt einer Anomalie muss bekannt sein.
- *FR8*: Die Daten müssen in einem durch *Python* gut einlesbaren Format gesichert werden.
- *FR9*: Die Größe des zu erzeugenden Datensatzes muss flexibel gesteuert werden können.

### Nicht funktionale Anforderungen:

- *NFR1*: Die Erstellung der Daten darf nicht länger als 10 Minuten dauern.
- *NFR2*: Der Datengenerator muss unter *macOS Catalina 10.15.1* und *Ubuntu Linux 18.04.3 LTS* lauffähig sein.

### Konzept

Die Grundidee bzgl. des Datengenerators ist, anhand von Zeitreihen, bestehend aus Lasten und Erzeugern, Messwerte in einem Stromnetz für verschiedene Zeitpunkte zu messen. Durch Änderungen an der zugrundeliegenden Topologie des Stromnetzes können so Änderungen in den Daten verursacht werden, die zu einem *Concept Drift* führen. Durch saisonale Effekte in den Zeitreihen (z.B. nur Zeitreihen aus dem Winter: geringere PV-Einspeisung) kann ebenfalls ein *Concept Drift* erzeugt werden, sofern ein Anomalieerkennungsmodell nur auf saisonalen Daten trainiert wurde.

In Abbildung 3.5 ist das aus den *FR1* bis *FR9* sowie *NFR1* und *NFR2* resultierende Konzept des Datengenerators inkl. aller relevanten Komponenten dargestellt. Im Kern baut der Generator auf *PandaPower*<sup>3</sup> auf. Zusätzlich werden die drei Komponenten *Executor*, *AnomalyGenerator* und *NetworkTopologyChanger* implementiert.

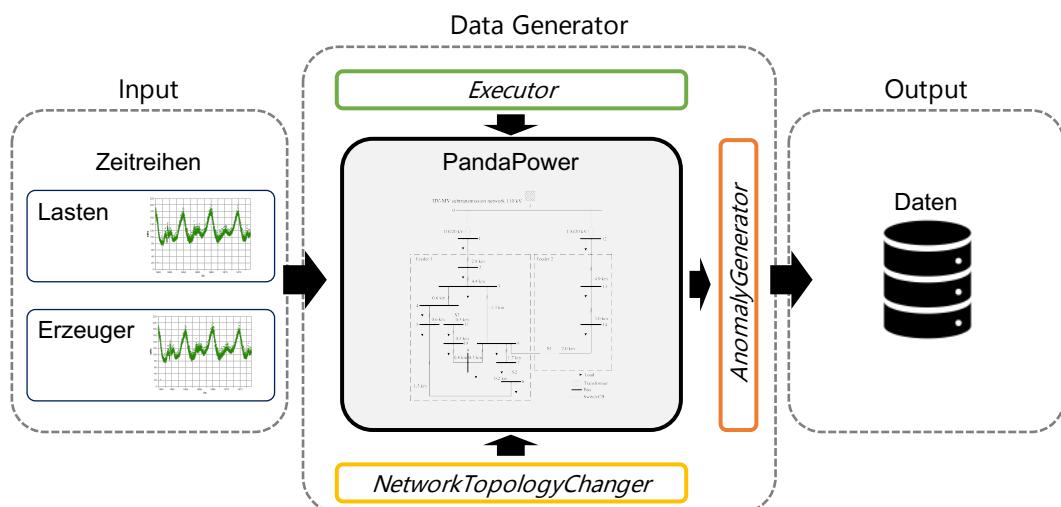


Abbildung 3.5.: Konzept des Datengenerators mit allen Komponenten sowie dem Input und Output (eigene Darstellung)

Zunächst wird das Mittelspannungsverteilungsnetz *CIGRE* in *PandaPower* modelliert. Die Netztopologie ist bereits in *PandaPower* integriert, es müssen jedoch noch die entsprechenden Lasten und Erzeuger an die Knoten im Netz modelliert werden. Als Lasten und Erzeuger werden reale Daten von Haushalten, Industrieanlagen, Photovoltaik- sowie Windkraftanlagen verwendet. Da *PandaPower* keine dynamische Berechnung der Lastflüsse ermöglicht, sondern die Berechnungen immer nur für

<sup>3</sup> *PandaPower* ist eine vom Fraunhofer EE Institut und der Universität Kassel entwickelte Bibliothek zur Lastflussrechnung und Optimierung von Stromnetzen. Für mehr Informationen siehe: <https://pandapower.readthedocs.io/en/v2.1.0/>

einen Zeitpunkt durchführt, ist ein *Executor* zu implementieren, welcher eine automatische Berechnung der Lastflüsse und Netzkennzahlen über mehrere Zeitpunkte hinweg ermöglicht.

Damit durch den Datengenerators ein *Concept Drift* möglich wird, ist eine Komponente zur Änderung der Netztopologie notwendig. Mittels der Komponente *NetworkTopologyChanger* können Erzeuger und Lasten beispielsweise aus der Netztopologie entfernt oder hinzugefügt werden. Neben Änderungen an der Netztopologie durch Manipulation der Lasten und Erzeuger, können Änderungen in den Daten ebenfalls verursacht werden, indem durch die Komponente z.B. Steuerbefehle, wie das Öffnen eines Schalters, ausgeführt werden.

Die dritte und letzte Komponente ist der sog. *AnomalyGenerator*. Dieser wird verwendet, um Anomalien in den Daten zu erzeugen. Die Betrachtung von Anomalien kann in diesem Fall von zwei Perspektiven ausgehen. Aus konzeptueller Sicht können die Anomalien in unterschiedliche Arten, wie in 2.3.2 dargestellt, unterteilt werden. Aus semantischer Sicht des Stromnetzes können die Anomalien darin unterschieden werden, wodurch sie ausgelöst wurden, z.B. durch fehlerhafte Messwerte aus dem Stromnetz, dem Ausfall von Erzeugern, Angriffen auf das Stromnetz oder ähnliches. Die Aufgabe des *AnomalyGenerators* ist die Verbindung beider Sichtweisen. So sollen möglichst alle Arten von Anomalien in den Daten enthalten sein, diese müssen jedoch durch nachvollziehbare und realistische Aktionen aus dem Stromnetz verursacht werden.

Durch Kombination der drei Komponenten und *PandaPower* ist die Erzeugung und Manipulation von Daten möglich, welche für ein Training eines Modells zur Identifizierung von Anomalien verwendet werden kann. Durch die Steuerung der Zeitpunkte und Arten der Anomalien kann der Datensatz während der Erstellung mit *Labels* versehen werden, wodurch eine umfassende Evaluation der Modelle und Algorithmen möglich ist.

## 3.5. Technologien und Implementierung

In den folgenden Unterabschnitten werden die grundlegenden Entscheidungen bzgl. der Technologie und Implementierung getroffen. Dazu wird in Unterabschnitt 3.5.1 zunächst die Wahl der Programmiersprache vorgestellt und begründet. In Unterabschnitt 3.5.2 werden anschließend die verwendeten Bibliotheken aufgezählt. Im darauffolgenden Unterabschnitt 3.5.3 wird die verwendete Entwicklungsumgebung für das Training von ANNs beschrieben. Abschließend werden in Unterabschnitt 3.5.4 die Maßnahmen zur Nachvollziehbarkeit und Reproduzierbarkeit als wichtige Bestandteile der vorliegenden Arbeit vorgestellt.

### 3.5.1. Programmiersprache

Als Programmiersprache wird *Python*<sup>4</sup> in der Versionsnummer 3.8.0 verwendet. Der Hauptgrund hierfür liegt darin begründet, dass die meisten Bibliotheken, die im Rahmen der vorliegenden Arbeit verwendet werden, für Python verfügbar sind. Darüber hinaus ist *Python* eine geeignete Programmiersprache für Projekte im Kontext des *Machine Learnings*, was u.a. daran zu erkennen ist, dass *Python* in vielen wissenschaftlichen Publikationen verwendet wird [Inn+18]. Der Nachteil, dass *Python* eine geringere *Performance* im Vergleich zu hardware-näheren Programmiersprachen wie *C* aufweist, spielt im Kontext der vorliegenden Arbeit keine Rolle, da durch die Verwendung von speziell in *C* ge-

<sup>4</sup> <https://docs.python.org/3/>

schriebenen und für die exakte CPU-Architektur kompilierte Bibliotheken kein Performanceproblem entstehen sollte.

### 3.5.2. Bibliotheken

Die Wahl der Bibliotheken kann in die Kategorien ML-Bibliothek, *Meta-Learning*-Bibliothek sowie Test-Bibliothek unterteilt werden. Dabei gibt es je nach Kategorie unterschiedliche Anforderungen an die auszuwählende Bibliothek. Im Folgenden werden daher für jede Kategorie kurz die Anforderungen sowie gängige Bibliotheken aus der Theorie und Praxis dargestellt, um anschließend jeweils eine Auswahl zu treffen und diese kurz darzustellen.

#### ML-Bibliothek

An die zu verwendende ML-Bibliothek sind mehrere Anforderungen zu stellen. So ist es u.a. wichtig, dass die Möglichkeit besteht, einen *Autoencoder* mit unterschiedlichen Topologien abzubilden. Des Weiteren sollte die Bibliothek ein Training von KNN mittels GPUs unterstützen, um das Training bei Bedarf beschleunigen zu können. Eine weitere wichtige Anforderung ist die Möglichkeit, Gewichte eines KNN extern manipulieren zu können, da *Meta-Learning*-Algorithmen diese Möglichkeit während des Trainings benötigen.

Die drei am meisten verwendeten ML-Bibliotheken im Jahr 2019 sind *TensorFlow*<sup>5</sup> entwickelt von *Google*, *PyTorch*<sup>6</sup> entwickelt von *Facebook* sowie *Keras*<sup>7</sup> erstmals im Jahr 2015 von François Chollet veröffentlicht [Hal19].

Generell erfüllen alle drei genannten Bibliotheken die beschriebenen Anforderungen. Untersuchungen zeigen, dass insbesondere *TensorFlow* und *PyTorch* in der letzten Zeit an großer Bedeutung gewonnen haben. In Abbildung 3.6 ist ein Vergleich zwischen *TensorFlow* und *PyTorch* dargestellt. Der Abbildung ist zu entnehmen, dass *PyTorch* in den letzten Jahren in wissenschaftlichen Publikationen im Vergleich zu *TensorFlow* auf höhere Popularität stößt.

Aufgrund der zunehmenden Bedeutung von *PyTorch* und die damit einhergehende Verfügbarkeit von Dokumentationen sowie Unterstützung durch die *Community* und die Möglichkeit, einfach die Gewichte eines trainierten KNN zu manipulieren, wird im Rahmen der vorliegenden Arbeit *PyTorch* als ML-Bibliothek ausgewählt.

#### *Meta-Learning*-Bibliothek

Eine Anforderung an die *Meta-Learning*-Bibliothek ist, dass sie möglichst alle in Kapitel 2.4 vorgestellten *Meta-Learning*-Algorithmen unterstützt, da zum Zeitpunkt der Konzeptionierung noch kein Algorithmus ausgewählt werden soll. Eine kurze Recherche im Rahmen der vorliegen Arbeit ergab, dass derzeit nur wenige Bibliotheken veröffentlicht sind, die *Meta-Learning*-Algorithmen beinhalten. Eine dieser Bibliotheken ist *TorchMeta*<sup>8</sup>, welche *PyTorch* um einige *Meta-Learning*-Algorithmen er-

---

<sup>5</sup> <https://www.tensorflow.org/>

<sup>6</sup> <https://pytorch.org/>

<sup>7</sup> <https://keras.io/>

<sup>8</sup> <https://github.com/tristandeleu/pytorch-meta>

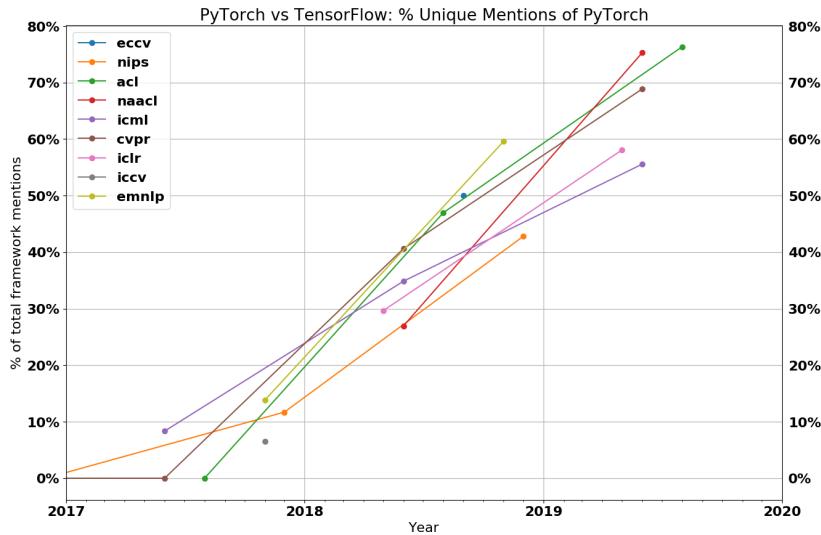


Abbildung 3.6.: Verwendung von PyTorch und TensorFlow in wissenschaftlichen Publikationen [He19]

weitert. Eine weitere Bibliothek ist *learn2learn* (*l2l*)<sup>9</sup>, ebenfalls eine Erweiterung von *PyTorch*, welche *PyTorch* um Algorithmen wie MAML oder FOMAML erweitert. Kurz vor Beginn der Implementierung der *Meta-Learning*-Algorithmen in *PyTorch* wurde sowohl *TorchMeta*, als auch *l2l* auf dem *Global PyTorch Summer Hackathon 2019* mit dem Preis für die beste und einflussreichste Erweiterung ausgezeichnet [Del+19].

An dieser Stelle kann noch keine der Bibliotheken final ausgewählt werden, da die Bibliotheken jeweils nur einen Teil der notwendigen *Meta-Learning*-Algorithmen implementieren. Aufgrund der Flexibilität und dem Umfang der Implementierungen soll im Rahmen der Experimente zunächst *l2l* für die Implementierung des *Meta-Learnings* verwendet werden. So bietet *l2l* einerseits eine API, um bereits implementierte *Meta-Learning*-Algorithmen zu verwenden und andererseits eine Schnittstelle, um eigene Implementierungen von *Meta-Learning*-Algorithmen zu verwenden. Darüber hinaus bietet *l2l* mit FOMAML, MAML und MetaSGD die größte Anzahl an bereits implementierten Algorithmen. Sollte sich während der Implementierung oder der Durchführung der Experimente herausstellen, dass *l2l* den Anforderungen nicht genügt, besteht die Möglichkeit *l2l* um die eigenen Anforderungen zu erweitern, da der Quellcode offen verfügbar ist. Darüber hinaus gibt es für den MAML-Algorithmus von Finn, Abbeel und Levine eine online verfügbare Implementierung<sup>10</sup>, welche ebenfalls an die konkrete Problemstellung der vorliegenden Arbeit angepasst werden könnte.

## Test-Bibliothek

Zwar handelt es sich bei dem *Data Generator* nicht um ein zentrales Ergebnis der vorliegenden Arbeit, jedoch soll durch die Durchführung von Tests ein gewisses Maß an Qualitätssicherung während der Entwicklung sichergestellt werden. Aufgrund von Vorerfahrungen sollen daher *Unit Tests* geschrieben und ausgeführt werden. Eine Anforderung an die Test-Bibliothek ist dabei lediglich eine

<sup>9</sup> <http://learn2learn.net/>

<sup>10</sup> <https://github.com/cbfinn/maml>

gute Integration in die verwendete Entwicklungsumgebung. Die Bibliothek *unittest*<sup>11</sup> ist eine direkt in *Python 3.8.0* integrierte Bibliothek für *Unit Tests*. Aufgrund von bereits vorhandenem Wissen sowie der guten Integration von *unittest* in alle gängigen Entwicklungsumgebungen wird daher *unittest* als Test-Bibliothek ausgewählt.

### 3.5.3. Entwicklungsumgebung

Die Entwicklungsumgebung besteht aus mehreren Komponenten, welche ein effizientes und effektives Arbeiten ermöglichen sollen. Für die Installation und Versionierung von *Python* Bibliotheken wird der *Python* Paketmanager *pip*<sup>12</sup> verwendet. Für eine schnelle Analyse und Darstellung von Daten wird *Jupyter*<sup>13</sup> bzw. *Jupyter Notebooks* verwendet. Für die Entwicklung von Modellen und komplexeren Artefakten, wie z.B. dem *Data Generator* wird *PyCharm*<sup>14</sup> von *JetBrains* als IDE verwendet. Sollte das Training der KNN zu komplex bzw. rechenintensiv werden, besteht die Möglichkeit auf einen *DGX-1*<sup>15</sup> Server der Marke *nvidia* zurückzugreifen. Dabei handelt es sich um einen Server, welcher mit acht GPUs bestückt ist, die für das Training komplexer KNN genutzt werden können.

### 3.5.4. Nachvollziehbarkeit und Reproduzierbarkeit

Nach Fehr et al. sind Nachvollziehbarkeit und Reproduzierbarkeit zentrale Bestandteile wissenschaftlichen Arbeitens bei der Durchführung von Computer-Experimenten [Feh+16]. Im Rahmen der vorliegenden Arbeit sollen daher folgend die Maßnahmen, welche der Nachvollziehbarkeit und Reproduzierbarkeit dienen, erläutert werden.

Die Verfügbarkeit des Quellcodes selbst ist für die Nachvollziehbarkeit erforderlich und für die Reproduzierbarkeit unbedingt notwendig. Daher wird sämtlicher Code, der als Ergebnis der vorliegenden Arbeit entspringt, neben der eigentlichen Arbeit bereitgestellt. Da der Quellcode alleine mögliche Interessierte nicht dazu befähigt den Code auszuführen, ist ebenfalls eine rudimentäre Dokumentation bereitzustellen. Diese umfasst neben der Dokumentation des Codes zum einen eine *README* Datei, welche neben dem Veröffentlichungsdatum und einer Versionsnummer, kurz die Motivation und grundlegende Funktionalität des Softwareartefaktes erläutert. Zum anderen umfasst die Dokumentation eine *RUNME* Datei, welche eine Anleitung zur Ausführung des Codes beinhaltet. Darüber hinaus sind alle notwendigen Informationen, wie z.B. die verwendeten Software-Bibliotheken, deren Versionsnummern sowie eine Anleitung zur Installation der Bibliotheken anzugeben, damit der Code ausgeführt werden kann. Da als Paketverwaltungstool *pip* verwendet wird, kann dies mittels sog. *requirements.txt* Dateien umgesetzt werden.

Darüber hinaus sind alle ML-relevanten Informationen und Artefakte bereitzustellen, welche zur Nachvollziehbarkeit und Reproduzierbarkeit notwendig sind. So sind beispielsweise alle Daten, sowohl in unverarbeiteter als auch in aufbereiteter Form, bereitzustellen. Zusätzlich ist anzugeben, welches Experiment mit welchen Daten durchgeführt wurde. Da das Training von KNN aufgrund der

---

<sup>11</sup> <https://docs.python.org/3/library/unittest.html>

<sup>12</sup> <https://pypi.org/project/pip/>

<sup>13</sup> <https://jupyter.org/>

<sup>14</sup> <https://www.jetbrains.com/de-de/pycharm/>

<sup>15</sup> <https://www.nvidia.com/de-de/data-center/dgx-1/>

Verwendung zufälliger Initialisierungen ggf. nicht deterministisch<sup>16</sup> ist, werden darüber hinaus alle trainierten KNN in einer geeigneten Form bereitgestellt.

Die Bereitstellung aller relevanten Informationen für die Entwicklung und das Training der KNN sowie eine Anleitung zur Anwendung ermöglichen es Dritten, die Experimente selbstständig auszuführen und zu bewerten. Zusammen mit den Daten sowie den trainierten KNN ist es Interessenten darüber hinaus möglich, die Artefakte auszuführen und die Ergebnisse eigenständig zu evaluieren. Neben den bereits vorgestellten Maßnahmen zur Nachvollziehbarkeit und Reproduzierbarkeit bietet es sich an, alle relevanten Informationen und Artefakte in einem *Docker Container*<sup>17</sup> bereitzustellen. Das *Docker* Format ermöglicht es Interessierten, mit nur wenig Aufwand und unabhängig von den zugrundeliegenden Software- und Hardware-Ressourcen, eine eigene Evaluation vorzunehmen. Ein Nachteil an dieser Möglichkeit ist das notwendige Wissen bzgl. der *Docker* Technologie. Deshalb kommt die Bereitstellung aller Informationen und Artefakte in einem *Docker Container* nur zusätzlich zu einer einfach zugänglichen Bereitstellung in Frage.

---

<sup>16</sup> Funktionalitäten der Software-Bibliotheken wie z.B. die Verwendung von sog. *Seeds* ermöglichen es, einen gewissen Determinismus während des Trainings zu erzeugen.

<sup>17</sup> <https://www.docker.com/>



## 4. Vorstudie

Das Ziel der Vorstudie ist ein erster Erkenntnisgewinn bzgl. der Technologien und Eigenschaften verschiedener ML-Modelle, welche im Rahmen der auf die Vorstudie folgenden Experimente verwendet werden können. Das Vorgehen sowie die Ergebnisse der Vorstudie werden in dem folgenden Kapitel dargestellt. Hierzu wird in Abschnitt 4.1 zunächst die Motivation sowie die Zielstellung der Vorstudie detailliert dargelegt. Im anschließenden Abschnitt 4.2 wird der verwendete Datensatz aus statistischer und ML-Sicht beschrieben. Anschließend werden in Abschnitt 4.3 die Maßnahmen zur Vorbereitung des Datensatzes für das Training erläutert. Darauf folgend wird in Abschnitt 4.4 die Durchführung der Vorstudie beschrieben. Im abschließenden Abschnitt 4.5 werden die Erkenntnisse der Vorstudie zusammenfassend dargestellt.

### 4.1. Motivation und Zielstellung

Die Hauptmotivation zur Durchführung der Vorstudie liegt in dem Erkenntnisgewinn für die darauf-folgenden Experimente. So dient die Vorstudie insbesondere dazu, den Raum möglicher Parameterkombinationen während der Experimente einzuschränken. Dies umfasst beispielsweise die Auswahl einer möglichen Topologie für *Autoencoder* oder geeigneter Kennzahlen zur Evaluation der Modelle. Darüber hinaus kann im Rahmen der Vorstudie eine gewisse Expertise bzgl. der verwendeten ML-Bibliothek *PyTorch* aufgebaut werden, da diese im Vorlauf der Experimente nicht umfassend vorhanden war. Der Erkenntnisgewinn durch die Vorstudie zahlt zwar nicht direkt auf die Beantwortung der Forschungsfragen der vorliegenden Arbeit ein, dient aber dennoch als wichtige Voraussetzung zur Durchführung der Experimente. Die folgende Aufzählung fasst alle Ziele der Vorstudie stichpunktartig zusammen:

- Erlangung einer Expertise bzgl. der ML-Bibliothek *PyTorch*
- Sammeln von Erfahrungen mit unterschiedlichen Topologien für *Autoencoder*
- Auswahl einer geeigneten Methodik zur Evaluation der Modelle
- Auswahl geeigneter Metriken zur Evaluation der Modelle
- Evaluation von *Autoencodern* gegenüber weiteren Verfahren des ML für die Bewertung der Eignung zur Anomalieerkennung
- Implementierung erster *Python*-Funktionen, welche im Rahmen der Experimente wiederverwendet werden können

Die obige Aufzählung der Ziele der Vorstudie kann für eine Bewertung des Erfolgs bzw. des Fortschrittes der Vorstudie verwendet werden.

### 4.2. Beschreibung des Datensatzes

In dem folgenden Abschnitt soll der im Rahmen der Vorstudie verwendete Datensatz beschrieben werden. Hierzu wird in Unterabschnitt 4.2.1 zunächst der zugrundeliegende Datengenerierungsprozess, im Rahmen dessen der Datensatz erstellt wurde, erläutert. Im darauffolgenden Unterabschnitt 4.2.2 wird der Datensatz aus einer statistischen Perspektive bzgl. der Eignung für die Anomalieerken-

nung untersucht. Abschließend wird in Unterabschnitt 4.2.2 eine Übersicht über verwandte Arbeiten gegeben, im Rahmen derer der Datensatz verwendet wurde.

#### 4.2.1. Datengenerierungsprozess

Der Datensatz wurde von Adhikari et al. im *Oak Ridge National Laboratory* an der *Mississippi State University* erstellt und umfasst Messwerte eines elektrischen Übertragungssystems mit Störungen, Kontrolleingriffen und Cyberangriffen. Die Messwerte umfassen zeitsynchronisierte Messungen aus dem Stromnetz, Zustandsmessungen von *Snort*<sup>1</sup> sowie Befehle eines simulierten Bedienfeldes und Zustandsmessungen mehrerer Relais. In Abbildung 4.1 ist die Konfiguration des Stromnetzes inkl. aller relevanten Komponenten und Akteure dargestellt. Im Stromnetz existieren die zwei Generatoren *G1* sowie *G2*. *R1* bis *R4* sind *IEDs*, welche die Schalter *BR1* bis *BR4* steuern können. Jedes *IED* steuert jeweils einen Schalter (*R1* steuert *BR1*, *R2* steuert *BR2*, etc.). Darüber hinaus existieren die zwei Leitungen *L1* (zwischen *BR1* und *BR2*) sowie *L2* (zwischen *BR3* und *BR4*).

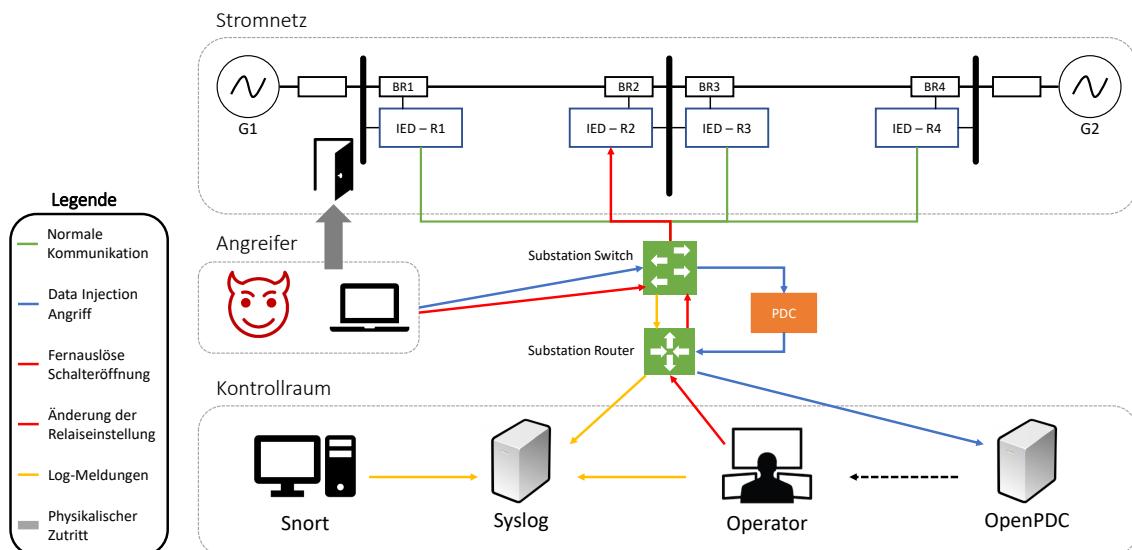


Abbildung 4.1.: *Übersicht des Szenarios zur Generierung der Daten (eigene Darstellung nach [Adh+14])*

Die *IEDs* verwenden ein Distanzschutzschemata, welches den Schalter bei einem erkannten Fehler auslöst, unabhängig davon, ob der Fehler tatsächlich gültig ist oder gefälscht wurde, da die *IEDs* keine interne Validierung der Fehler durchführen. Operator können auch manuell Befehle an die *IEDs* senden, um die Schalter auszulösen. Die manuelle Schaltung wird beispielsweise bei der Wartung an den Leitungen oder anderen Netzkomponenten angewandt. In den *IEDs* ist jeweils eine sog. *Phasor Measurement Unit* (PMU) verbaut. Eine PMU ist ein zeitsynchronisiertes Zeigermessgerät, welches die komplexe Amplitude von Strom und Spannung zu einem bestimmten Zeitpunkt misst. Insgesamt

<sup>1</sup> *Snort* ist eine freie Software für die Identifikation von Angriffen auf ein Computernetzwerk (ein sog. *Intrusion Detection System*, kurz: IDS) [Cas+03]

sind vier *PMUs* verbaut, die je 29 Messwerte, wie z.B. die Frequenz  $f_{Hz}$ , den Phasenwinkel  $\phi_t$ , die Stromspannung  $U$ , die Stromstärke  $I$ , die Impedanz  $Z$ , den Frequenzgang oder den Status eines Relais, aufzeichnen. Daraus ergibt sich insgesamt eine Aufzeichnung von 116 Messwerten durch die vier *PMUs*. Zusätzlich wurden pro Aufzeichnung 12 Statusflags von *Snort* und den *IEDs* gespeichert, sodass pro Zeitschritt 128 Messwerte aufgezeichnet werden. Bezuglich der Auflösung der Zeit werden in der Beschreibung des Datensatzes keine Angaben gemacht.

In dem Kontrollraum gibt es ein simuliertes Bedienfeld, über welches Steuerbefehle an die Komponenten im Stromnetz gesendet werden können. Neben *Snort* wurde *Syslog* verwendet, um Log-Meldungen von *Snort* und den Komponenten aus dem Stromnetz zu sammeln (dargestellt durch die orangenen Pfeile in Abbildung 4.1). *openPDC*<sup>2</sup> wurde verwendet, um den *Stream* der Messwert-Zeitreihen zu sammeln und dem Operator bereitzustellen.

Der Angreifer hat drei unterschiedliche Möglichkeiten eines Angriffes, welche in der nachfolgenden Auflistung als Angriff dargestellt werden. Darüber hinaus hat der Angreifer die Möglichkeit, über einen physischen Zutritt zu Komponenten des Stromnetzes, diese zu beeinflussen. Für beide Angriffsarten muss der Angreifer bereits die Schutzmaßnahmen auf Netzwerk- bzw. physischer Ebene überwunden haben.

In dem Datensatz wurden fünf unterschiedliche Arten von Szenarien erzeugt, welche in der folgenden Auflistung dargestellt sind:

- **Kurzschlussfehler:** Ein Kurzschluss in einer der Stromleitungen  $L1$  oder  $L2$ , welcher an verschiedenen Stellen auftreten kann. Die Stelle des Auftrittes wird durch den prozentualen Bereich indiziert.
- **Leitungswartung:** Mindestens ein Relais wird für eine bestimmte Leitung deaktiviert, damit Leitungswartungen durchgeführt werden können.
- **Injektion eines Fernauslösekommandos (Angriff):** Bei diesem Angriff wird ein unautorisierte Befehl an ein Relais gesendet, der z.B. das Öffnen eines Relais bewirkt (dargestellt durch die roten Pfeile ausgehend vom Angreifer).
- **Änderung der Relaiseinstellungen (Angriff):** Das Distanzschutzschemata der Relais wird so konfiguriert, dass ein Relais bei einem gültigen Fehler oder Befehl nicht auslöst (dargestellt durch die roten Pfeile ausgehend vom Angreifer).
- **Data Injection (Angriff):** Für Parameter wie beispielsweise die Stromstärke, oder -spannung werden fehlerhafte Werte durch einen Angreifer während der Übertragung von den Komponenten des Stromnetzes zu den Komponenten des Kontrollraums injiziert (dargestellt durch die blauen Pfeile ausgehend vom Angreifer). Dieser Angriff kann dazu führen, dass der Operator aufgrund der gefälschten Messwerte beispielsweise eine neue Relaiskonfiguration vornimmt, welche dann zu einem Stromaussfall führt (dargestellt durch die roten Pfeile ausgehend vom Operator).

### 4.2.2. Statistische Beschreibung

Der Datensatz umfasst insgesamt 128 *Features* (Messwerte) sowie ein *Label*, welches ein *Sample* jeweils als „Angriff“ oder „kein Angriff“ markiert. Insgesamt besteht der Datensatz aus 72.072 *Samples*, wovon 20.628 (28,62 %) als „kein Angriff“ und 51.444 (71,38 %) als „Angriff“ markiert sind.

<sup>2</sup> <https://github.com/GridProtectionAlliance/openPDC/wiki>

Im Rahmen der statistischen Untersuchung des Datensatzes wurden für jedes *Feature* der Mittelwert, die Standardabweichung, der Minimal- und Maximalwert sowie die Quartile berechnet. Zusätzlich wurde die Zeitreihe jedes *Features* als Plot dargestellt sowie ein Boxplot und Histogramm pro *Feature* erstellt.

Die Visualisierung der Daten hat bereits erste Eigenschaften der Daten offenbart. So könnten einzelne Ausreißer in den Daten bereits auf einen *Data Injection* Angriff oder einen Netzausfall hindeuten. In Abbildung 4.2 ist exemplarisch die Frequenz an Relais 1 dargestellt. Diese pendelt, wie in einem amerikanischen Stromnetz üblich, um ihren Normwert von 60 Hz. Einzelne *Samples* weichen jedoch stark von der Norm ab. Dies lässt sich auch anhand der statistischen Kennzahlen darstellen. Der Mittelwert liegt für die Frequenz an Relais 1 bei 59,9928 Hz, mit einer Standardabweichung von 0,61 Hz. Der minimale Wert der Messung beträgt 0 Hz, der maximale Messwert beträgt 66,01 Hz.

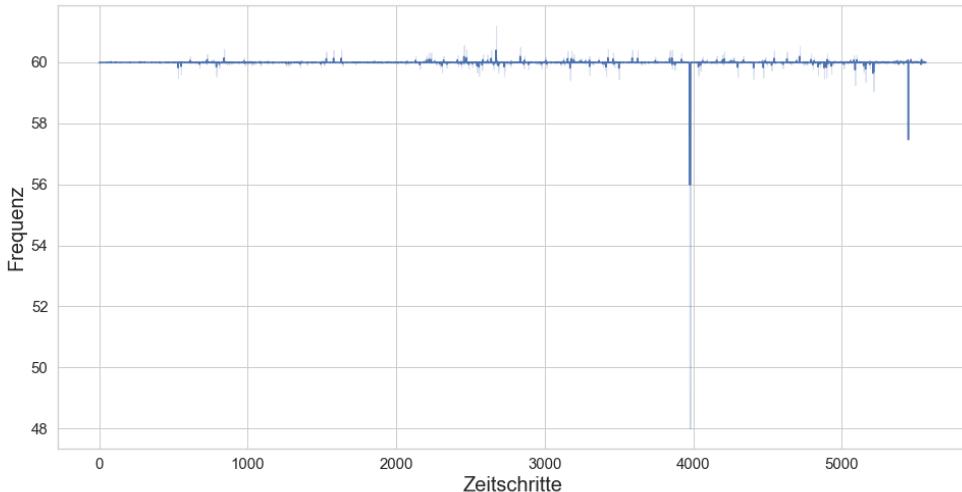


Abbildung 4.2.: *Plot der Messungen der Frequenz an Relais 1 (eigene Darstellung)*

#### 4.2.3. Verwandte Arbeiten

Adhikari et al. referenzieren in [Adh+14] vier ihrer Paper, die den *Power System Attack* Datensatz verwenden. Im Folgenden werden die vier relevanten Paper kurz genannt und zusammengefasst.

In [PMA15c] haben Pan, Morris und Adhikari ein IDS mittels klassischer *Data Mining* Methoden entwickelt. Dazu haben die Autoren ein sog. *Common Path Mining* verwendet. Anschließend wurde das Modell mittels des Datensatzes evaluiert, indem eine *Confusion Matrix* erstellt und die entsprechenden Kennzahlen berechnet wurden. Insgesamt konnten die Autoren über alle Szenarien hinweg eine *Accuracy* von 73,43 % erreichen.

In [PMA15b] haben Pan, Morris und Adhikari ebenfalls einen *Common Path Mining* Ansatz implementiert, welcher auf [PMA15c] aufbaut. Insgesamt haben Pan, Morris und Adhikari drei unterschiedliche Experimente durchgeführt, wobei jeweils unterschiedliche Szenarien in den Daten erzeugt wurden. Die *Accuracy* liegt bei den Experimenten zwischen 90 % und 95 %.

In [PMA15a] haben Pan, Morris und Adhikari ein Bayes'sches Netzwerk trainiert, um die Kausalzusammenhänge zwischen den verfügbaren Informationen grafisch zu codieren und Muster mit zeitlichen Zustandsübergängen zu erstellen. Die gelernten zeitlichen Zustandsübergänge werden anschließend als IDS *Framework* verwendet. Die Autoren machen über die *Accuracy* auf dem Datensatz keine Angaben.

In [Hin+14] haben Hink et al. sieben unterschiedliche, auf ML basierende Verfahren implementiert und evaluiert. Dabei haben die Autoren keine ANN, sondern statistische Verfahren, wie SVMs, *Random Forests* oder *AdaBoost* verwendet. Die beste *Accuracy* von etwa 90 % haben dabei Baum-basierte Verfahren mit *AdaBoost* erreicht.

Insgesamt konnten die Autoren in ihren Arbeiten zeigen, dass auf dem Datensatz die Entwicklung einer Anomalieerkennung mit einer *Accuracy* zwischen 70 % und 95 % möglich ist. In allen der referenzierten Arbeiten wurden klassische ML-Algorithmen, wie das *Common Path Mining* oder Baum-basierte Verfahren wie *AdaBoost* verwendet, sodass über die Performance von *Autoencodern* für den Datensatz keine Aussage getroffen werden kann.

### 4.3. Preprocessing

Im Rahmen des *Preprocessings* waren nur wenige Tätigkeiten notwendig, um den Datensatz für das Training vorzubereiten, da dieser im Rahmen einer Simulation erstellt wurde und eine aufwändige Bereinigung des Datensatzes daher nicht notwendig war.

Initial besteht der Datensatz aus 15 einzelnen Dateien. Diese wurden zu einem großen Datensatz zusammengefügt. Anschließend wurden Duplikate aus dem Datensatz entfernt. So wurde der Datensatz von 78.377 auf 78.368 *Samples* reduziert. Im Prinzip wäre die Entfernung von Duplikaten jedoch nicht notwendig gewesen, da im Rahmen der Vorstudie ein Verfahren des *unsupervised Learnings* verwendet wird und somit nicht die Gefahr besteht, Informationen aus dem Training während der Evaluation zu verwenden. Eine Bereinigung von Ausreißern darf nicht vorgenommen werden, da dies evtl. *Samples* eines *Data Injection* Angriffes oder andere Anomalien entfernen würde, die für das Training relevant sind.

Anschließend mussten noch Werte, welche von dem ML-Modell nicht als Zahl interpretiert werden können, eliminiert werden. So wurde der Datensatz weiter auf 72.072 *Samples* reduziert. Zwar war die Entfernung der *NaN*- und *Inf*-Werte notwendig, damit ein Modell trainiert werden konnte, jedoch ist dieser Schritt kritisch zu betrachten. So könnten *NaN*- oder *Inf*-Werte auch das Ergebnis eines *Data Injection*-Angriffes darstellen. Eine Alternative zur Eliminierung der Werte wäre die Ersetzung dieser gewesen. Da in der Beschreibung des Datensatzes jedoch keine Angaben über die *NaN*- und *Inf*-Werte gemacht wurden, wurden die Werte der Einfachheit halber entfernt.

Die *Features* *snort log1*, *control panel log3* und *control panel log4* wiesen über alle 72.072 *Samples* den gleichen Wert 0 aus. Da diese *Features* daher keine Informationen tragen, die für das Training relevant sind, wurden diese *Features* entfernt, sodass der Gesamt-Datensatz einer Matrix der Dimension  $72.072 \times 125$  entspricht.

Da die Wertebereiche der *Features* sehr unterschiedlich waren, wurden im Rahmen des *Preprocessings* alle *Features* auf einen Wertebereich  $[-1, 1]$  skaliert, um durch das Training bessere Ergebnisse zu erzielen [GBC16].

Abschließend wurde der Datensatz noch in zwei Datensätze aufgeteilt, wobei ein Datensatz alle *Samples* mit Angriffen und der zweite Datensatz alle *Samples* ohne Angriffe umfasst.

#### 4.4. Durchführung

Im Rahmen der Vorstudie wurden zwei unterschiedliche AE trainiert. Zunächst wurde ein linearer und anschließend ein komplexerer nichtlinearer AE trainiert. Für beide Modelle musste nach dem Training im Rahmen der Evaluation jeweils ein geeigneter Schwellwert  $\lambda$  gesucht werden.

Für den linearen AE wurden unterschiedliche Topologien evaluiert. Die beste Topologie des linearen AE hat im *Input- und Output-Layer* entsprechend der Anzahl der *Features* 125 Neuronen. Als *Hidden-Layer* wurde nur der *Latent-Space* mit 32 Neuronen verwendet. Als Optimierungsverfahren zum Lernen wurde *Adam* verwendet. Trainiert wurde in insgesamt 6 Epochen.

Der nichtlineare AE hat ebenfalls 125 Neuronen im *Input- und Output-Layer*. Der *Latent-Space* umfasst 96 Neuronen. Zusätzlich wurde jeweils vor und nach dem *Latent-Space* ein zusätzlicher *Hidden-Layer* mit je 110 Neuronen verwendet. Als Aktivierungsfunktionen wurde der *Tanh* sowie *ReLU* verwendet, da diese gute Eigenschaften für das Training aufweisen (s.u.). Als Lernalgorithmus wurde ebenfalls *Adam* verwendet. Trainiert wurde der AE mittels 18.566 normalen Samples in 100 Epochen. In Abbildung 4.3 ist eine schematische Darstellung des nichtlinearen AE dargestellt.

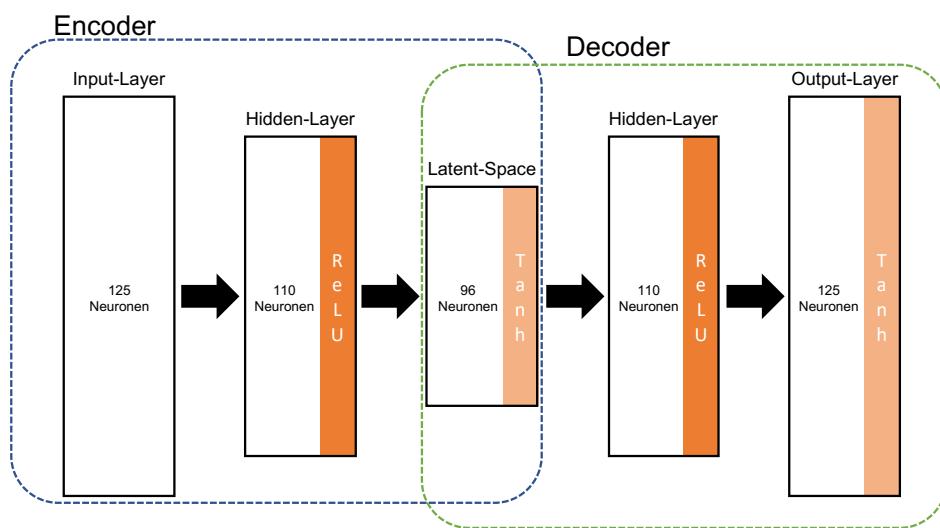


Abbildung 4.3.: Schematische Darstellung des nichtlinearen AE (eigene Darstellung)

Bei der Wahl der Anzahl der Neuronen in den *Hidden-Layer* und dem *Latent-Space* wurde die von Heaton eingeführte Formel verwendet. In [Hea08] schlägt Heaton folgende Formel vor:

$$n_k = \lfloor n_{in} + \frac{n_{out} - n_{in}}{hl + 1} \cdot k \rfloor, \text{ mit } hl > 0 \quad (4.1)$$

Nach dieser Empfehlung wird die Anzahl der Neuronen  $n_k$  in einem *Hidden-Layer*  $k$  in Abhängigkeit der Anzahl der Eingangsneuronen  $n_{in}$ , der Anzahl der Ausgangsneuronen  $n_{out}$  und der Anzahl der

*Hidden-Layer*  $hl$  berechnet, wobei  $hl > 0$  gilt. Dabei ist  $n_{out}$  nicht die Anzahl der Neuronen im *Output-Layer*, sondern die Anzahl der Neuronen im *Latent-Space*.

Die Auswahl der weiteren Parameter wurde anhand einer Literaturrecherche vorgenommen. So wurde die Wahl und Anordnung der Aktivierungsfunktionen anhand der Erkenntnisse von Ellison in [Ell18] vorgenommen. Die Anzahl der Epochen wurde evaluiert, indem während des Trainings untersucht wurde, ob der Fehler des ANN bei einer größeren Anzahl von Epochen noch weiter reduziert werden kann. Ein weiterer wichtiger Parameter der Topologie eines AE ist die Anzahl der Neuronen im *Latent-Space*. Dieser stellt im Verhältnis zu der Anzahl der *Input*-Neuronen den Komprimierungsfaktor des *Encoders* dar. In der Literatur gibt es bisher keine umfassenden Untersuchungen dazu, wie die Anzahl der Neuronen im *Latent-Space* auszuwählen sei. In [Amo+18] zeigen Amodio et al., dass die Anzahl der Neuronen im *Latent-Space* stark von der weiteren Topologie des AE abhängt. So können komplexere *Encoder* komplexere Eigenschaften der Daten lernen und daher einen größeren Komprimierungsfaktor erzielen. Entsprechend der Erkenntnisse aus [Bor+18] wurden im *Latent-Space* 96 Neuronen verwendet, was einem Komprimierungsfaktor von 1,3 entspricht.

In Code-Listing 4.1 ist beispielhaft die Definition des nichtlinearen AE in *PyTorch* dargestellt.

```

1 import torch
2 from torch.nn import Sequential, Linear, ReLU, Module, Tanh
3 from torch.nn.functional import mse_loss
4
5 class AutoEncoder(torch.nn.Module):
6     def __init__(self, num_inputs, val_lambda=666):
7         super(AutoEncoder, self).__init__()
8         self.val_lambda = val_lambda
9
10        self.encoder = Sequential(
11            Linear(num_inputs, 110),
12            ReLU(True),
13            Linear(110, 96),
14            Tanh()
15        )
16
17        self.decoder = Sequential(
18            Linear(96, 110),
19            ReLU(True),
20            Linear(110, num_inputs),
21            Tanh()
22        )
23
24    def forward(self, x):
25        x = self.encoder(x)
26        x = self.decoder(x)
27        return x

```

Listing 4.1: *Definition des nichtlinearen AE*

In Zeile 10 bis 15 ist die Definition des *Encoders* des AE mit den ersten beiden *Layern* dargestellt. In Zeile 17 bis 22 ist entsprechend der *Decoder* mit seinen Schichten und Aktivierungsfunktionen dargestellt. In Zeile 24 bis 27 ist die Funktion für einen *Forward-Pass* des AE dargestellt. So wird eine Eingabe  $x$  zunächst im *Encoder* encodiert (Zeile 25) und anschließend im *Decoder* in Zeile 26 dekodiert und als Ausgabe des ANN zurückgegeben.

Im Anschluss an das Training muss noch ein geeigneter Schwellwert  $\lambda$  definiert werden, damit der *Reconstruction-Error* des AE in ein binäres *Label* umgewandelt werden kann, welches ein *Input-Sample* als „normal“ oder „anormal“ kennzeichnet. Hierzu gibt es in der Literatur unterschiedliche Ansätze. In [KTP18] beschreiben Kiran, Thomas und Parakkal das Problem der Auswahl eines Schwellwerts in Abhängigkeit der Ziele (z.B. Maximierung der *FPR* oder *TPR*). Eine Strategie zur Auswahl eines Schwellwerts könnte daher die Optimierung einer Kennzahl sein. Malhotra et al. wählen den Schwellwert in [Mal+16] hingegen abhängig von der Verteilung der *Reconstruction-Error* während des Trainings. Sie empfehlen beispielsweise einen Schwellwert entsprechend der Formel  $\lambda = \bar{re} - 1,5 \cdot \sigma_{re}$ . Wobei  $\bar{re}$  der mittlere *Reconstruction-Error* des Trainings und  $\sigma_{re}$  seine Standardabweichung ist.

Im Rahmen der Vorstudie wurde die Maximierung der *Accuracy* als Auswahlstrategie eines Schwellwerts gewählt, da diese die Gesamt-Genauigkeit des Modells bewertet und zunächst einen guten Kompromiss darstellt. Um einen passenden Schwellwert zu suchen, wurde für ein definiertes Intervall an Schwellwerten  $[min, max]$  mit einer Schrittweite  $k$  die *Confusion Matrix* aufgestellt und die dazugehörigen Kennzahlen berechnet. Anschließend konnte der Schwellwert mit der größten *Accuracy* herausgesucht werden. Dieses Vorgehen war möglich, da die verwendeten Daten gelabelt waren, sodass die Vorhersagen eines Modells mit einem Schwellwert mithilfe der *Label* überprüft werden konnten. Durchgeführt wurde die Suche des Schwellwerts auf einer unabhängigen Test-Menge  $X_{test}$ , welche insgesamt 4.124 Samples umfasst, wovon je 50 % normale und 50 % anormale *Samples* sind.

## 4.5. Erkenntnisse

Bevor die eigentlichen Erkenntnisse der Vorstudie dargelegt und interpretiert werden, sollen zunächst die Ergebnisse der Evaluation der beiden zuvor trainierten AE beschrieben werden. In Tabelle 4.1 sind die Kennzahlen der *Confusion Matrix* beider AE dargestellt. Wie bereits im vorherigen Unterabschnitt 4.4 beschrieben, sind die jeweiligen Schwellwerte anhand der besten *Accuracy* ausgewählt worden<sup>3</sup>.

Kennzahl	linearer AE	nichtlinearer AE
Schwellwert	0,25	0,02
<i>Accuracy</i>	50,00 %	50,92 %
<i>Precision</i>	50,00 %	50,57 %
<i>Specivity</i>	0,00 %	2,81 %
<i>Sensitivity</i>	100,00 %	98,98 %

Tabelle 4.1.: Kennzahlen beider AutoEncoder

Im Vergleich zu den Modellen, welche in Unterabschnitt 4.2.3 beschrieben wurden, weisen beide AE eine wesentlich schlechtere *Accuracy* auf. Die *Accuracy* des linearen AE entspricht mit 50 % genau dem Verhältnis der normalen zu anormalen *Samples* in den Testdaten  $X_{test}$ . Eine *Sensitivity* von 100 % sowie eine *Specivity* von 0 % zeigen, dass der lineare AE alle *Samples* als Anomalie gekennzeichnet hat, wodurch auch die *Accuracy* von 50 % zu erklären ist. Der nichtlineare AE weist sehr ähnliche Kennzahlen wie der lineare AE auf, scheint hingegen mit einer *Specivity* von 2,81 % und einer *Accuracy* von 50,92 % zumindest einige Anomalien besser als solche erkannt zu haben.

<sup>3</sup> Die Schwellwerte beziehen sich auf die auf  $[-1, 1]$  skalierten Daten

Es war zu erwarten, dass die AE aufgrund der Art des *unsupervised* Trainings eine geringere *Performance* als die *supervised* Modelle, welche in Unterabschnitt 4.4 beschrieben wurden, aufweisen. Mit einer solchen *Performance* wären die AE in der Praxis jedoch nicht geeignet. Eine im Anschluss an das Training durchgeführte Untersuchung hat bereits einige Gründe für die geringe *Performance* der AE zeigen können. In Abbildung 4.4 ist ein Histogramm der Test-Losse (hier Synonym für *Reconstruction-Error*) der normalen (grün) und anormalen (rot) *Samples* des nichtlinearen AE dargestellt.

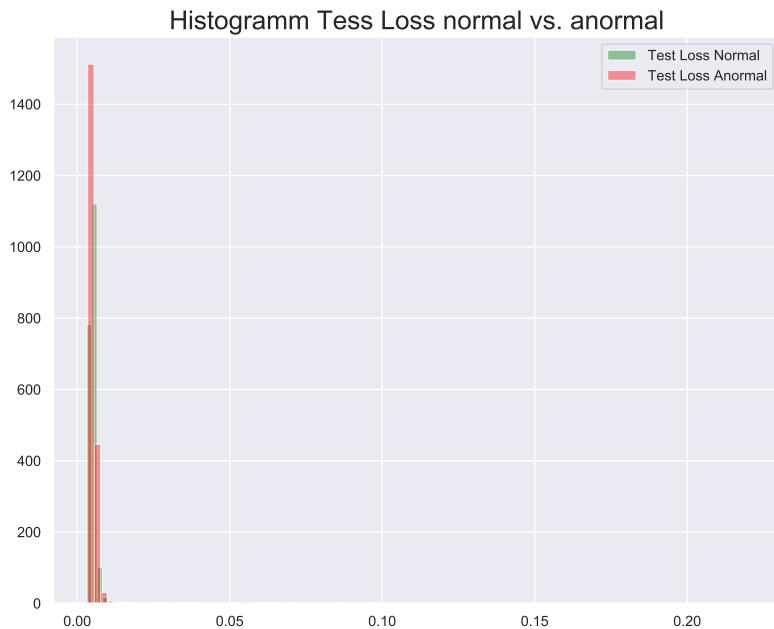
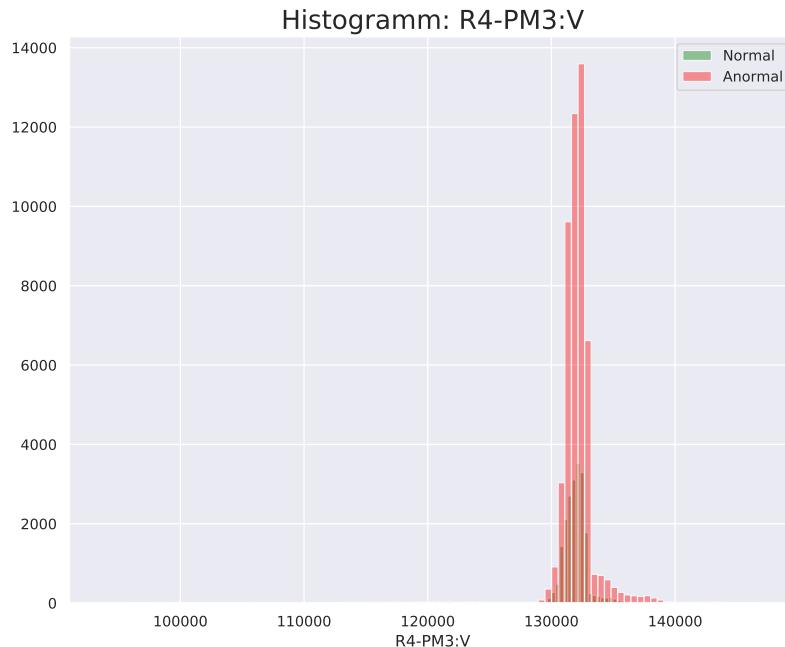


Abbildung 4.4.: *Histogramm der Test Loss für normale und anormale Samples (eigene Darstellung)*

Der Abbildung 4.4 kann entnommen werden, dass sich die *Reconstruction-Error* für normale und anormale *Samples* kaum unterscheiden. Dies ist ein Zeichen dafür, dass sich die anormalen Daten unwesentlich von den normalen Daten unterscheiden oder dass der AE während des Trainings die Eigenschaften der Daten nicht ausreichend lernen konnte. Eine weitere Untersuchung der Daten konnte dies größtenteils bestätigen. In Abbildung 4.5 ist exemplarisch das Histogramm des *Features „R4-PM3:V“* dargestellt. Wie in der vorherigen Abbildung entsprechen die roten Balken der Verteilung der anormalen Daten und die grünen Balken der Verteilung der normalen Daten.

Anhand der Abbildung 4.5 ist zu sehen, dass sich die Verteilung der normalen und anormalen Daten kaum unterscheidet. Jedoch gibt es einige wenige anormale *Samples*, welche kleinere Werte annehmen. Eine weitere Untersuchung mittels unterschiedlicher statistischer Verfahren, wie z.B. Boxplots, konnte ebenfalls zeigen, dass sich die normalen und anormalen Daten sehr ähnlich sind. Da das Hauptziel der vorliegenden Arbeit nicht die eigentliche Evaluation von AE zur Anomalieerkennung ist, wird von einer Verbesserung der *Performance* der AE an dieser Stelle jedoch abgesehen.



## 5. Implementierung

In den folgenden Abschnitten wird die Implementierung der Komponenten und ML-Modelle zur Durchführung der Experimente beschrieben. Zunächst wird in Abschnitt 5.1 die Implementierung des *DataGenerators* erläutert. In 5.2 wird der Simulationsprozess mithilfe des *DataGenerators* zur Erzeugung der Daten beschrieben. Anschließend wird in Abschnitt 5.3 die Implementierung des Modells zur Anomalieerkennung dargelegt. In Abschnitt 5.4 wird die Implementierung des *Meta-Learnings* in *PyTorch* gezeigt, bevor abschließend in Abschnitt 5.5 die Implementierung der Wrapper-Klassen zur Durchführung der Experimente erläutert wird.

### 5.1. Implementierung des Datengenerators

Ziel des Datengenerators ist es, synthetische Daten zur Untersuchung der Hypothesen im Rahmen der Experimente zu erzeugen. Dazu wurden in Unterabschnitt 3.4.4 bereits funktionale sowie nicht-funktionale Anforderungen an den Datengenerator gestellt, die bei der Implementierung berücksichtigt werden sollen.

Entgegen der ursprünglichen Idee, den Datengenerator als eigenständig lauffähiges Programm zu entwickeln, besteht der Datengenerator nun aus einzelnen aufeinander aufbauenden *Python* Klassen. Dieser Ansatz wurde verfolgt, da die Entwicklung eines großen Programms einen zu hohen Entwicklungsaufwand verursacht hätte, der keinen zusätzlichen Nutzen bringt. Insgesamt besteht der Datengenerator aus den drei Komponenten *Executor*, *AnomalyGenerator* sowie *NetworkTopologyChanger*, welche im Folgenden näher beschrieben werden. Die Komponenten wurden in der Regel als eigene *Python* Klasse implementiert, welche alle notwendigen Funktionen bereitstellt. Die Ausführung der Funktionen der Komponenten wurde anschließend jeweils in einem *Jupyter Notebook* durchgeführt, in welchem die Funktionen der unterschiedlichen Klassen genutzt wurden.

#### 5.1.1. Executor

Mittels des *Executors* wird zunächst ein initialer Datensatz erstellt, welcher anschließend für die Simulation in *PandaPower* verwendet wird. Anhand eines gegebenen Start- und Enddatums wird zunächst ein Datensatz mit normierten Standardlastprofilen (SLP) bzw. Standardeinspeiseprofilen (SEP) in 15-minütiger Auflösung und in Abhängigkeit zur Jahreszeit und Tagesart erzeugt. Die Werte der SLP sind dabei nach Vorschrift des Bundesverbandes der Energie- und Wasserwirtschaft (BDEW) auf 1.000 kWh p.a. normiert. Bei den SEP handelt es sich bereits um synthetisierte Profile der Stadtwerke Emmendingen, welche ebenfalls auf 1.000 kWh p.a. normiert, aber in ihrer Eigenschaft bereits an die geografischen sowie netzspezifischen Gegebenheiten der Stadtwerke angepasst sind. Als Vorbereitung zur Verwendung in *PandaPower* wurden alle Profile auf MW normiert und den SEP wurde ein negatives Vorzeichen vorangestellt, da *PandaPower* im Verbraucherzeigerdiagramm rechnet. Die folgende Tabelle 5.1 beinhaltet alle verwendeten SLP und SEP, eine kurze Bezeichnung sowie eine Referenz auf die entsprechenden Quellen.

Profil	Art	Beschreibung	Quelle
H0	SLP	Haushalt, Privatverbrauch, ggf. geringfügig gewerblicher Bedarf	[FF02]
L0	SLP	Landwirtschaftsbetriebe allg., gewichteter Mittelwert der Profile L1 und L2	[FF02]
G0	SLP	Gewerbe allg., gewichteter Mittelwert der Profile G1 bis G6	[FF02]
EV0	SEP	Kleine bis mittlere Photovoltaikanlage	[Sta20]
E29	SEP	Kleine bis mittlere Biomasse-/Biogas-Anlage	[Ene19]
E27	SEP	Onshore-Windenergieanlagen	[Ene19]

Tabelle 5.1.: Verwendete SEP und SLP zur Modellierung in *PandaPower*

Anschließend wurden mittels der SLP und SEP die Lasten und Einspeiser in *PandaPower* modelliert. Als zugrundeliegende Stromnetztopologie wurde das CIGRE<sup>1</sup> Benchmark Mittelspannungsverteilernetz gewählt. Dieses bietet sich für die Simulation an, da das Netz entwickelt wurde, um eine Analyse und Validierung neuer Methoden und Techniken zu ermöglichen, die darauf abzielen eine wirtschaftliche, robuste und umweltverträgliche Integration von DER (*Distributed Energy Ressources*) zu ermöglichen [Str+14]. Ein weiterer Grund für die Auswahl des CIGRE Mittelspannungsnetzes ist die gute Implementierung des Referenzmodells in *PandaPower*, welche einfach an die Zwecke der Simulation angepasst werden kann. In Anhang A ist in Abbildung A.1 das Referenzmodell inkl. der Leitungen, Transformatoren, Schalter und Busse dargestellt.

Entsprechend der Empfehlungen des CIGRE wurden die Profile in dem Mittelspannungsnetz modelliert. In Tabelle 5.2 ist die Zuordnung der Lasten zu dem jeweiligen Bus im Netz dargestellt. Die Zahl in der Spalte eines Profils gibt an, wie oft das Profil an dem entsprechenden Bus als Last modelliert wurde.

Entsprechend der Faktoren aus Tabelle 5.2 konnte mittels einer Vektor-Vektor-Multiplikation pro Bus  $i$  für jeden Zeitschritt  $t$  die Wirkleistung  $P$  sowie anhand des Phasenwinkels  $\phi(t)$  die Blindleistung  $Q$  berechnet werden. In Formel 5.1 ist die Berechnung der Wirkleistung  $p_{i,t}$  für einen Bus  $i \in \{i | 1 \leq i \leq 14\}$  zu einem Zeitpunkt  $t$  dargestellt.

$$p_{i,t} = \mathbf{v}_i \cdot \mathbf{x}_t^T \quad (5.1)$$

Dabei ist  $p_{i,t}$  die konkrete Wirkleistung zu einem Zeitpunkt  $t$  für ein Bus  $i$  und ein Skalar als Ergebnis der Multiplikation des Zeilenvektors  $\mathbf{v}_i$  (Faktoren eines bestimmten Busses  $i$  aus Tabelle 5.2) und des Spaltenvektors  $\mathbf{x}_t^T$  (Werte der Wirkleistung der Profile aus Tabelle 5.1 in einem Zeitpunkt  $t$ ).

Für einen gegebenen Phasenwinkel  $\phi(t)$  in einem Zeitpunkt  $t$  und einer gegebenen Wirkleistung  $p_{i,t}$  kann die Blindleistung  $q_{i,t}$  an einem Bus  $i$  entsprechend der Darstellung in Formel 5.2 berechnet werden.

$$q_{i,t} = p_{i,t} \cdot \tan(\cos^{-1}(\phi(t))) \quad (5.2)$$

<sup>1</sup> CIGRE ist die Abkürzung für „Conseil International des Grands Réseaux Électriques“ und bezeichnet eine internationale technisch-wissenschaftliche Organisation für den Informationsaustausch von Fachleuten im Bereich elektrische Energieübertragung und -Versorgung.

Bus	Profile					
	H0	G0	L0	EV0	E27	E29
<b>1:</b>	2	1	0	0	0	1
<b>2:</b>	4	2	1	1	0	1
<b>3:</b>	2	0	0	0	1	0
<b>4:</b>	0	1	0	0	0	0
<b>5:</b>	3	0	0	2	0	0
<b>6:</b>	1	1	0	0	0	0
<b>7:</b>	1	2	0	0	0	0
<b>8:</b>	1	0	0	1	0	0
<b>9:</b>	3	0	0	0	0	0
<b>10:</b>	0	0	0	0	0	0
<b>11:</b>	2	0	0	0	0	0
<b>12:</b>	0	0	0	0	0	0
<b>13:</b>	2	0	2	0	2	0
<b>14:</b>	1	0	1	0	1	0

Tabelle 5.2.: Zuordnung der Lasten im CIGRE Benchmark-Referenzmodell in PandaPower

Dabei ist tan die Funktion zur Berechnung des Tangens und  $\cos^{-1}$  die Funktion zur Bestimmung des trigonometrisch inversen Kosinus anhand eines gegebenen Phasenwinkels  $\phi(t)$ . Bei der Implementierung wurde für alle Zeitpunkte ein konstanter Phasenwinkel von  $\phi(t) = 0,9$  gewählt. In Anhang B ist in Listing B.1 ein Code-Beispiel zur Berechnung der  $P$  und  $Q$  Werte anhand der in 5.1 und 5.2 gezeigten Formeln dargestellt.

Der nun erstellte Datensatz mit den  $P$  und  $Q$  Werten pro Bus konnte anschließend genutzt werden, um die Simulation in *PandaPower* durchzuführen. Hierzu wurde mittels der Funktion `pandapower.create_load()` zunächst pro Bus in dem CIGRE Netz eine Last erstellt. An jeder Last wird anschließend die Summe der Lasten und Erzeuger modelliert. In Anhang B ist in Listing B.2 die Initialisierung des leeren CIGRE Mittelspannungsnetzes sowie die Modellierung der Lasten und Erzeuger pro Bus dargestellt. Die Lasten wurden dabei initial nur pro Bus ohne Werte modelliert. Die Annotation der Lasten mit Werten und die Ausführung der Lastflussrechnung wird in der anschließenden Simulation vorgenommen.

Die Durchführung der Simulation in *PandaPower* besteht im Wesentlichen aus einer Schleife, die über den gesamten Simulationszeitraum, definiert durch ein Start- und Enddatum und einer Schrittweite von 15 Minuten, läuft. Pro Schleifendurchlauf wird ein Simulationsschritt ausgeführt, welcher aus drei Phasen besteht:

**1. Wirk- und Blindleistungswerte im Netz aktualisieren:** Anhand des zuvor erstellten Datensatzes mit den Wirk- und Blindleistungswerten sowie der Modellierung der Lasten im CIGRE Netz, können in jedem Zeitschritt  $t$  die Werte der Lasten im Netz aktualisiert werden. Hierfür bietet *PandaPower* die Funktion `grid.load.update()`. Der Funktion wird ein *Pandas DataFrame* übergeben, welche pro Bus je einen  $P$  sowie  $Q$  Wert beinhaltet. In Anhang A ist in Abbildung A.2 beispielhaft eine Matrix zur Aktualisierung der Werte in einem Zeitpunkt dargestellt.

**2. Lastflussrechnung durchführen:** Die Ausführung der Lastflussrechnung in *PandaPower* wird durch einen einzelnen Aufruf der entsprechenden Funktion `pp.runpp()` ausgeführt. Als Algorithmus wurde Newton-Raphson verwendet, da dieser zum einen einen guten Kompromiss aus Robustheit und Geschwindigkeit für das CIGRE Netz darstellt und zum anderen eine Implementierung des Algorithmus in *numba*<sup>2</sup> vorhanden ist [Gal00].

**3. Messwerte sichern:** Zuletzt werden für jeden Zeitschritt  $t$  die Messwerte aus dem CIGRE Netz gesichert und nach dem Durchlauf der Simulation als Datei gespeichert, um anschließend für das Training des ML-Modells verwendet werden zu können. Insgesamt wurden 85 Messwerte pro Zeitschritt gesichert. Dabei wurden verschiedene Messwerte, wie die prozentuale Auslastung der Transformatoren und Leitungen, Wirk- und Blindleistungswerte der Leitungen und Busse sowie den Status der Leitungen und Schalter, gemessen. Eine vollständige Auflistung der Messwerte inkl. einer kurzen Beschreibung ist in Anhang C in Tabelle C.1 dargestellt.

In Anhang B ist in Listing B.3 ein Auszug des Codes der Simulation dargestellt. Für eine Simulation von vier Jahren ist das Ergebnis der Simulation eine Matrix  $\mathbf{X}^{m \times n}$ , wobei  $m$  die Anzahl der Zeitschritte ist, also für die Jahre von 2020 bis 2023 entsprechend  $m = (366 + 365 + 365 + 365) \times 96 = 140.256$ , und  $n$  die Anzahl der Messwerte, also  $n = 85$  ist. Die *Executor* Komponente wurde als *Python* Klasse mit dem Namen `Executor` implementiert, welche alle Funktionen für die Durchführung der Simulation umfasst.

### 5.1.2. *AnomalyGenerator*

Mittels der Komponente *AnomalyGenerator* können beliebig viele Anomalien in den Daten erzeugt werden. Dabei ermöglicht der *AnomalyGenerator* die Erstellung der drei unterschiedlichen Arten von Anomalien: Punkt- und Kollektiv anomalien sowie ein Verrauschen von Werten über ein Intervall<sup>3</sup>. Für jede Art von Anomalie kann außerdem mittels eines Faktors die Stärke der Anomalie bestimmt werden. Ein anomaler Wert  $x'_i$  in Form einer Punktanomalie für einen normalen Wert  $x_i$  wird wie folgt bestimmt. In Formel 5.3 wird zunächst die Stärke  $z$  der Anomalie mittels des Faktors  $s$  bestimmt. Die Stärke der Anomalie ist definiert als prozentuale Abweichung des anormalen Wertes  $x'_i$  zu seinem Vorgängerwert  $x_{i-1}$ . Der Skalar  $r \in \{-1, 1\}$  wird verwendet, um eine positive oder negative Veränderung des Wertes  $x_i$  zu erzeugen und wird zufällig gewählt. Ein anomaler Wert  $x'_i$  wird schließlich durch die in 5.4 dargestellte Gleichung bestimmt.

$$z = \frac{x_{i-1}}{100} \cdot s \quad (5.3)$$

$$x'_i = x_i + r \cdot z \quad (5.4)$$

Eine kollektive Anomalie wird analog zu dem gerade beschriebenen Vorgehen erzeugt. Dafür muss zusätzlich ein Intervall  $[i_{start}, i_{ende}]$  definiert werden, über welches die Anomalie erzeugt werden soll. Die Stärke der Anomalie wird anschließend analog als prozentuale Abweichung zu  $x_{i_{start}-1}$  berechnet. In Abbildung 5.1 ist ein Auszug des *Features* „prozentuale Auslastung Trafo 0“ dargestellt.

<sup>2</sup> *numba* ist ein Just-in-time-Compiler für Python, welcher numerische Funktionen in Maschinencode übersetzt, was zu einer Beschleunigung der Ausführung dieser Funktionen führt. Darüber hinaus ermöglicht es die parallele Verarbeitung von *NumPy*-Objekten auf mehreren Kernen des Prozessors wie auch der Grafikkarte.

<sup>3</sup> Siehe hierzu auch Abschnitt 2.3

Die kollektive Anomalie ist im Intervall zwischen den Werten 35 und 45 gut als Verschiebung zu den normalen Werten zu erkennen. Zusätzlich zu der einfachen Verschiebung, kann einer kollektiven Anomalie noch ein Rauschen hinzugefügt werden.

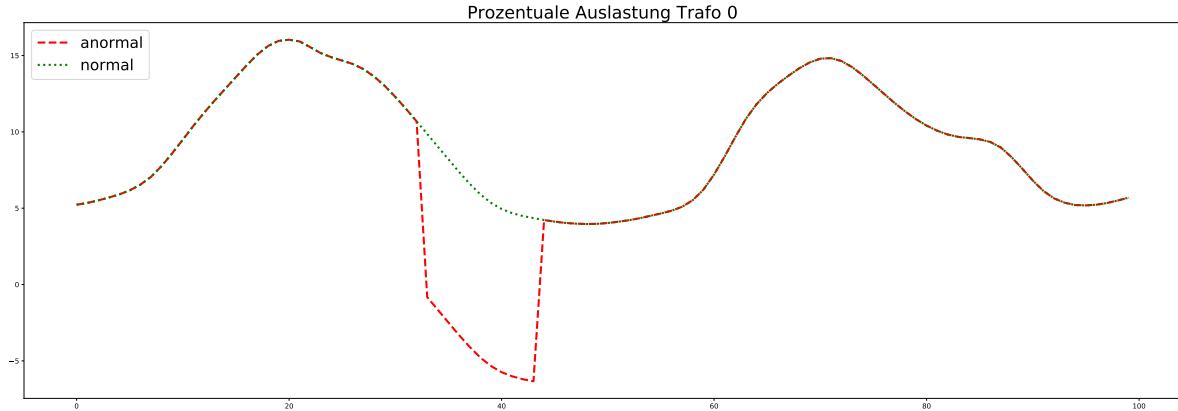


Abbildung 5.1.: Normale und anormale Daten für das Feature „prozentuale Auslastung Trafo 0“ (eigene Darstellung)

Um  $n$  normale Werte  $\mathbf{x} \in \mathbb{R}^n$  mit einem Rauschen zu versehen, werden aus einer Normalverteilung mit dem Erwartungswert  $\mu = 1$  und einer Standardabweichung  $\sigma = z$  zunächst  $n$  Werte gezogen, so dass ein Vektor  $\mathbf{u} \in \mathbb{R}^n$  entsteht, welcher anschließend auf  $\mathbf{x}$  addiert werden kann, sodass  $\mathbf{x}' = \mathbf{x} + \mathbf{u}$ . Der Skalar  $z \in \mathbb{N}^*$  bestimmt dabei, äquivalent zu der Erzeugung der Punkt- und Kollektivanomalien, die Stärke des Rauschens.

Das Ergebnis der *AnomalyGenerator* Komponente ist ein Datensatz  $\mathbf{X}'^{m \times n}$  mit gelabelten Anomalien, welcher anschließend verwendet werden kann, um einen AE zu trainieren und anhand seiner *Performance* zu evaluieren. Jedes *Sample* im Datensatz ist annotiert mit 0 = keine Anomalie, 1 = Punktanomalie, 2 = Kollektivanomalie oder 3 = Rauschen. In Anhang B sind, für ein besseres Verständnis, Auszüge zur Erstellung der verschiedenen Anomalien sowie der Definition der Anzahl, Stärke und Länge der Anomalien in dem Listing B.4 dargestellt.

### 5.1.3. NetworkTopologyChanger

Die Komponente des *NetworkTopologyChanger* umfasst Funktionen, um Änderungen an dem zugrundeliegenden CIGRE Mittelspannungsverteilernetz vorzunehmen, sodass ein *Concept Drift* entsteht. Im Unterabschnitt 2.1.4 wurden bereits die Grundlagen des *Concept Drifts* dargelegt. Dort wurde der *Concept Drift* definiert als: „Wenn sich die statistischen Eigenschaften der Zielvariablen ändern, ändert sich auch das Konzept dessen, was vorhergesagt werden soll. Beispielsweise könnte sich die Definition dessen, was als Anomalie im Stromnetz gilt, im Laufe der Zeit ändern, wenn natürliche Änderungen, wie Neuinstallationen oder Netzeingriffe, im Stromnetz vorgenommen werden. Diese Art von Änderungen führt zu einem Concept Drift.“.

Um also einen *Concept Drift* zu erzeugen, müssen die Änderungen zu Veränderungen in den zugrundeliegenden Daten führen, welche jedoch nicht als Anomalie gelten. Es wurden folgende Arten von Änderungen identifiziert und anschließend implementiert. Die Ideen der Manipulation des Stromnetzes (*CD1* bis *CD5*) stammen dabei hauptsächlich aus dem Diskussionspapier der Bundesnetzagentur

(BNetzA) [Bun10] und dem Bericht zum Thema Netzausbau in Deutschland, herausgegeben von der Konrad Adenauer Stiftung [Kon14].

### Manipulation des Stromnetzes:

- *CD1*: Öffnen und Schließen von Schaltern im CIGRE Netz
- *CD2*: Änderungen der Zuordnung von SEP und SLP an verschiedenen Bussen
- *CD3*: Modifikation von Profilen
- *CD4*: Hinzufügen eines Speichers
- *CD5*: Veränderung des Phasenwinkels  $\phi(t)$  einzelner Anlagen

### Statistische Manipulationen:

- *CD6*: Hinzufügen eines leichten Rauschens
- *CD7*: Statistische Modifikation einzelner Zeitreihen

Im Folgenden wird die Implementierung und Ausführung der *NetworkTopologyChanger* Komponente beispielhaft anhand der Manipulation der Schalter (*CD1*) sowie der Änderung der Zuordnung der SEP und SLP an den verschiedenen Bussen (*CD2*) dargestellt.

Wie auch bei der Definition der Art und Stelle der Anomalien in der *AnomalyGenerator* Komponente, werden die Art und Stelle der Maßnahmen zur Erzeugung eines *Concept Drifts* in Form eines Python *Dictionary* definiert. Eine solche Maßnahme wird im weiteren Verlauf als *Event* bezeichnet. Ein Beispiel für die Definition mehrerer *Events* ist in Listing B.5 in Anhang B dargestellt.

Die Implementierung der *NetworkTopologyChanger* Komponente wurde als *Python* Klasse mit dem Namen `Drifter` vorgenommen. Die Klasse implementiert alle notwendigen Funktionen, um Änderungen an dem *PandaPower CIGRE* Netz durchzuführen. Einen Auszug der Implementierung der Klasse ist in Anhang B in Listing B.6 dargestellt.

Für ein *Event* zum Öffnen oder Schließen eines bestimmten Schalters im *PandaPower* Netz, muss zunächst, anhand der in Listing B.5 dargestellten Struktur, definiert werden, für welchen Schalter und zu welchem Zeitpunkt das Event durchgeführt werden soll. Bei der Ausführung der Lastflussrechnungen durch die *Executor* Komponente, prüft der *NetworkTopologyChanger* zu jedem Zeitpunkt, ob ein *Event* ausgeführt werden muss. Ist dies der Fall, wird das jeweilige Event bearbeitet, indem die definierten Änderungen (bei einem *CD1 Event* entsprechend das Ändern des Schalter-Zustandes in *PandaPower*) in dem *PandaPower CIGRE* Netz durchgeführt werden. Die Auswirkungen auf die Messwerte sind in Abbildung A.3 in Anhang A dargestellt.

In Abbildung A.4 in Anhang A sind die Auswirkungen der Veränderung des *Load Mappings* (*CD2*) an einem Bus auf die Wirk- und Blindleistungswerte dargestellt. In diesem Beispiel wurde zusätzlich zu dem bestehendem *Load Mapping* aus zwei H0-SLP und einem Wind-SEP, noch ein weiteres G0-SLP verwendet, was für einen Zeitraum von fünf Tagen zu einer deutlich höheren Last an dem entsprechenden Bus 3 führt.

## 5.2. Simulationsprozess zur Erstellung der Daten

In Abbildung 5.2 ist das Konzept zur Erstellung der Daten inkl. aller relevanten Phasen und Komponenten im Rahmen der Simulation dargestellt. Insgesamt wird die Simulation über einen Zeitraum von vier Jahren (2020 bis 2023) durchgeführt. In den ersten beiden Jahren 2020 und 2021 wird im Rahmen der Simulation nur die Lastflussrechnung mittels der *Executor* Komponente durchgeführt. Die Messdaten der zwei Jahre werden anschließend genutzt, um ein Modell zur Anomalieerkennung zu trainieren.

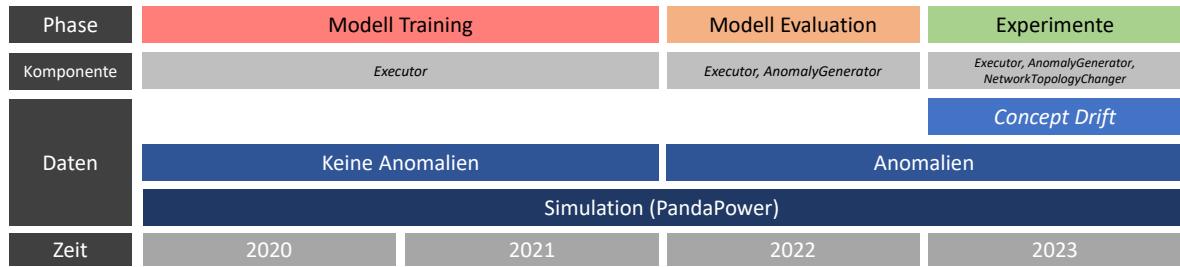


Abbildung 5.2.: Konzept zur Erstellung der Daten inkl. aller relevanten Phasen und Komponenten (eigene Darstellung)

Im dritten Jahr 2022 wird zunächst die Lastflussrechnung zur Erzeugung der Messdaten ausgeführt. Anschließend werden mittels der *AnomalyGenerator* Komponente Anomalien in den Daten erzeugt. Diese Daten werden anschließend verwendet, um die Güte des Anomalieerkennungsmodells zu evaluieren. Während der Simulation im vierten Jahr 2023, wird mittels der *NetworkTopologyChanger* Komponente ein *Concept Drift* und mithilfe der *AnomalyGenerator* Komponente Anomalien erzeugt. Die Daten aus dem Jahr 2023, werden anschließend für die Experimente genutzt.

Wie in Unterabschnitt 5.1.1 bereits beschrieben, werden im Rahmen der Simulation mittels *PandaPower* pro Zeitschritt 85 Messwerte erhoben. In Tabelle 5.3 ist eine Übersicht über die Anzahl der *Samples* und vorgenommenen Manipulation der Daten pro Jahr dargestellt. Da das Jahr 2020 ein Schaltjahr ist, sind im Jahr 2020 96 (entspricht einem Tag) mehr *Samples* als in den anschließenden Jahren.

Jahr	Anzahl Samples	Manipulation der Daten	Verwendung
2020	35.136	keine	Training
2021	35.040	keine	Training
2022	35.040	Anomalien	Evaluation der Modelle
2023	35.040	Anomalien, <i>Concept Drift</i>	Experimente

Tabelle 5.3.: Übersicht über die Anzahl der Samples und Manipulation der Daten pro Jahr

In Abbildung 5.3 sind drei  $20 \times 20$  *Self-Organizing Maps* (SOM)<sup>4</sup> abgebildet, welche die drei unterschiedlichen Datensätze darstellen. Alle drei SOM wurden jeweils mit den gleichen Gewichten initialisiert und 10.000 Epochen mit einer konstanten Lernrate von 0.01 trainiert.

<sup>4</sup> Eine SOM ist eine Art ANN, welches verwendet wird, um eine niedrigdimensionale (meist zweidimensionale) diskretisierte Darstellung des Eingaberaums der Trainingsmuster zu erzeugen, eine sog. Karte. SOM gehören daher zu den Algorithmen der Dimensionsreduktion. Für weitere Informationen zu SOM siehe [Qia+19]

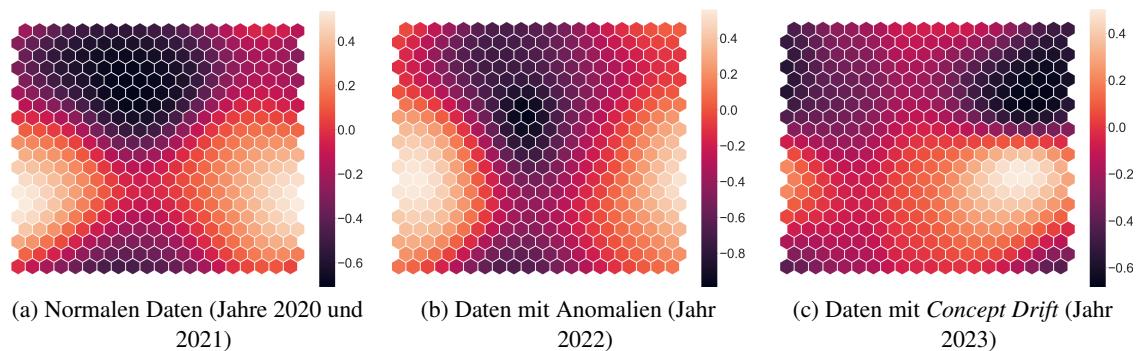


Abbildung 5.3.: Drei  $20 \times 20$  SOMs zur Darstellung der Unterschiedlichkeit der drei Datensätze.

Der Abbildung 5.3 kann entnommen werden, dass sich die normalen Daten (5.3a) und anormalen Daten (5.3b) kaum unterscheiden, wohingegen die Daten (5.3a), welche mittels des *NetworkTopologyChangers* stark manipuliert wurden, sich stark von den beiden vorherigen Daten unterscheiden.

### 5.3. Implementierung der Anomalieerkennung

Ein AE zur Anomalieerkennung in den zuvor erstellten Daten, dient als notwendige Voraussetzung für die Evaluation des *Meta-Learning* Ansatzes im Rahmen der Experimente. Erste Experimente sowie die Erkenntnisse aus der Vorstudie haben jedoch gezeigt, dass das Training sowie das Suchen und Finden einer passenden AE-Topologie eine große Herausforderung (im Rahmen der Anomalieerkennung) darstellt [SM02]. Da der Fokus der vorliegenden Arbeit nicht auf der Evaluation von AE zur Anomalieerkennung, sondern auf der Evaluation von *Meta-Learning*-Algorithmen zur Minimierung des *Concept Drifts* liegt, wurde beschlossen, das Problem der Anomalieerkennung zu minimieren. So wurde der in Abschnitt 5.2 beschriebene Datensatz von 85 auf 17 repräsentative *Features* reduziert. Dazu wurden im weiteren Verlauf nur die *Features* der prozentualen Auslastung der Leitungen und Transformatoren verwendet. Eine Auflistung der 17 *Features* ist im Anhang C in Tabelle C.2 dargestellt.

Mittels der daraus resultierenden Datensätze, wurde zunächst ein AE zur Anomalieerkennung trainiert und anschließend evaluiert. Der aus der Implementierung folgende AE, dient in den anschließenden Experimenten als Modell  $M_2$ , welches mittels klassischer Trainingsverfahren auf einem Datensatz zur Anomalieerkennung trainiert wurde und im Rahmen der Evaluationen auf den Datensätzen  $\mathbf{X}_{drifted}$  und  $\mathbf{X}_{drifted,anormal}$  hinsichtlich des *Concept Drifts* untersucht wird.

Für die Wahl der Hyperparameter des Modells wurden insbesondere die Erkenntnisse der Vorstudie aus Unterabschnitt 4.5 herangezogen. Im Folgenden wird in Unterabschnitt 5.3.1 zunächst die Implementierung sowie das Training des AE beschrieben. Im darauf folgenden Unterabschnitt 5.3.2 wird der AE evaluiert sowie die Effekte des *Concept Drifts* auf den AE untersucht.

### 5.3.1. Implementierung

Im Anhang A ist in Abbildung A.5 die schematische Topologie des verwendeten AE dargestellt. Der AE verwendet im *Input-* sowie *Output-Layer*, entsprechend der Anzahl der *Features*, 17 Neuronen mit Tanh-Aktivierungsfunktionen. In den beiden *Hidden-Layern* werden sowohl im *Encoder*, als auch im *Decoder* jeweils 12 Neuronen mit einer ReLU-Aktivierungsfunktion verwendet. Der *Latent-Space* umfasst 8 Neuronen und verwendet ebenfalls Tanh-Aktivierungsfunktionen.

Als *Loss*-Funktion wurde der MSE verwendet. Adam wurde mit einer *Batch-Size* von 128 *Samples* und einer initialen Lernrate von 0.01 als *Optimizer* verwendet. Trainiert wurde zunächst in 64 Epochen. Da nach ca. 30 Epochen der *Loss* nicht weiter gesunken ist, wurde bei späteren Trainingsläufen in 32 Epochen trainiert. Die Gewichte des AE wurden initialisiert, indem aus einer stetigen Gleichverteilung  $\mathcal{U}(-\sqrt{k}, \sqrt{k})$  mit  $k = \frac{1}{in\_features}$  entsprechend der Anzahl der Gewichte  $n$  initiale Gewichte gezogen wurden. Dabei wurde mit `torch.manual_seed(42)` ein *Seed* verwendet, damit die Gewichte bei mehreren Trainingsdurchläufen jeweils gleich initialisiert werden und die trainierten AE so besser vergleichbar sind.

In Abbildung 5.4 ist der *Trainings-Loss* des AE über die 32 Epochen dargestellt. Der Abbildung kann entnommen werden, dass der AE während des Trainings konvergiert, indem der Fehler (*Loss*) des AE während des Trainings minimiert wird.



Abbildung 5.4.: *Trainings-Loss* des AutoEncoders über 32 Epochen mit Adam (eigene Darstellung)

Im Rahmen der Vorstudie wurden bereits unterschiedliche Methoden zur Suche eines geeigneten Schwellwerts, welcher den *Reconstruction Error* des AE in ein binäres *Label* umwandelt, diskutiert. Alle der diskutierten Methoden hatten jedoch den Nachteil, dass die Suche sehr lange dauert. Daher wurde im Rahmen der Implementierung ein weiteres Verfahren evaluiert. Da sowohl die Anomalien, als auch die Events für den *Concept Drift* synthetisch erzeugt werden, können die Daten auch mit *Labels* versehen werden. So konnte eine logistische Regression trainiert werden, welche als einzigen *Input* den *Reconstruction Error* des AE erhält und als *Output* ein binäres Label bestimmt. In Abbildung 5.5 ist der gesamte Prozess zur Bestimmung eines binären *Labels* anhand eines *Samples* dargestellt.

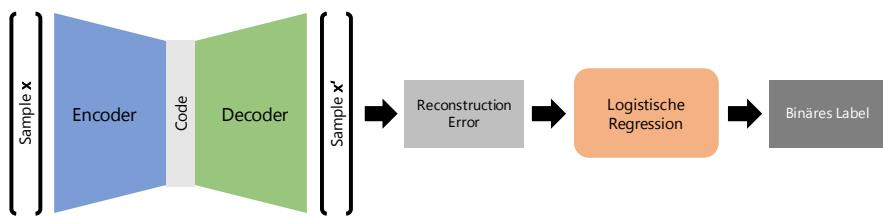


Abbildung 5.5.: Schema des Prognoseprozesses vom AutoEncoder über die logistische Regression bis zum binären Label (eigene Darstellung)

Für die logistische Regression wurde der *LIBLINEAR* Solver verwendet, da dieser einen guten Kompromiss aus Trainings- und Prognosezeit für große Datensätze sowie einer guten Performance für die Klassifikation liefert [Ped+11]. Zusätzlich wurde der Klasse der Anomalien ein höheres Gewicht gegeben, damit diese besser prognostiziert werden können. Implementiert wurde die logistische Regression in dem ML-Framework *scikit-learn*.

### 5.3.2. Ergebnisse

Im Folgenden sollen die Ergebnisse in Form der Kennzahlen des Modells  $M_2$  nach dem Training dargestellt werden. Hierzu wird das Modell zunächst bzgl. seiner allgemeinen Performance auf den drei verwendeten Datensätzen untersucht. Anschließend wird noch eine detailliertere Analyse der Performance des Modells hinsichtlich der Performance, je nach Art der Anomalie und nach Art des *Concept Drift Events*, pro Datensatz vorgenommen.

Zur Evaluation des trainierten AE wurden anhand der Prognosen auf drei unterschiedlichen Datensätzen, jeweils die Kennzahlen des AE bestimmt. Alle drei Datensätze  $X$  umfassen ein Jahr an Daten (entspricht 35.040 Samples). Der Datensatz  $X_{test}$  wurde mithilfe der *AnomalyGenerator* Komponente so manipuliert, dass dieser eine Reihe von Anomalien beinhaltet. Der Datensatz  $X_{drifted}$  beinhaltet keine Anomalien, jedoch wurden mittels der *NetworkTopologyChanger* Komponente Änderungen an dem Datensatz vorgenommen, sodass ein *Concept Drift* provoziert wird. Da  $X_{drifted}$  keine Anomalien umfasst, sind auch die Kennzahlen *Precision*, *Sensitivity* und *F1-Score* für den Datensatz entsprechend 0.0. Der Datensatz  $X_{drifted,anormal}$  stellt eine Kombination aus  $X_{test}$  und  $X_{drifted}$  dar. So wurde  $X_{drifted,anormal}$  mit den gleichen Events wie  $X_{drifted}$  manipuliert und mit den gleichen Anomalien wie  $X_{test}$  versehen. Die Kennzahlen sind in Tabelle 5.4 dargestellt.

Kennzahl	Datensätze		
	$X_{test}$	$X_{drifted}$	$X_{drifted,anormal}$
<b>Accuracy:</b>	<b>99.70 %</b>	90.51 %	90.28 %
<b>Precision:</b>	<b>99.99 %</b>	0.0 %	42.05 %
<b>Specificity:</b>	93.15 %	<b>99.99 %</b>	92.40 %
<b>Sensitivity:</b>	95.79 %	0.0 %	<b>96.28 %</b>
<b>F1-Score:</b>	<b>97.85 %</b>	0.0 %	58.53 %

Tabelle 5.4.: Kennzahlen zur Performance des AutoEncoders auf drei unterschiedlichen Datensätzen zur Evaluation. Die jeweils beste Kennzahl ist fett hervorgehoben.

Zusätzlich zu den Kennzahlen in Tabelle 5.4 sind in Abbildung A.6 in Anhang A, die drei *Confusion Matrizen* des AE auf den jeweiligen Datensätzen dargestellt. Anhand der Kennzahlen aus Tabelle 5.4 und den *Confusion Matrices*, lassen sich die folgenden Schlüsse ziehen:

- Der AE hat auf dem Datensatz  $\mathbf{X}_{test}$  insgesamt die beste *Accuracy*, *Precision* und *F1-Score*.
- Auf dem Datensatz  $\mathbf{X}_{drifted,anormal}$  hat der AE eine *Precision* von nur 42.05 %, da die Anzahl der als Anomalie prognostizierten *Samples*, die aber keine Anomalie (FP) sind, mit 3313 sehr hoch ist. Im Vergleich zu  $\mathbf{X}_{test}$  hat der AE auf  $\mathbf{X}_{drifted,anormal}$  mit 3313 *Samples* wesentlich mehr *Samples* fälschlicherweise als Anomalie prognostiziert (FPR). Die Ursache dafür liegt vermutlich an dem künstlich erzeugten *Concept Drift*.
- Eine Analyse der 3324 *Samples* in der Menge der FP auf  $\mathbf{X}_{drifted}$  hat gezeigt, dass alle darin enthaltenen *Samples* ein *Concept Drift Label* tragen. 3000 *Samples* in FP tragen dabei das *Drift Label 1* (Öffnen- bzw. Schließen eines Schalters), 300 *Samples* tragen das *Label 2* (Veränderung der Lastprofile) und nur 24 *Samples* das *Label 3* (Veränderung des Phasenwinkels).
- Gleichzeitig hat der AE auf  $\mathbf{X}_{drifted,anormal}$  mit 96.28 % seine insgesamt beste *Sensitivity*. Eine Analyse der  $TP_{\mathbf{X}_{drifted,anormal}}$  von  $\mathbf{X}_{drifted,anormal}$  hat ergeben, dass die Differenz zu  $TP_{\mathbf{X}_{test}}$ , also  $TP_{diff} = TP_{\mathbf{X}_{drifted,anormal}} \setminus TP_{\mathbf{X}_{test}}$ , insbesondere jene *Samples* beinhaltet, welche auf  $\mathbf{X}_{test}$  in der Menge der FN sind und gleichzeitig im Rahmen der Erzeugung eines *Concept Drifts*, neben einer Anomalie, mit einem *Concept Drift* versehen wurden.
- Die auf  $\mathbf{X}_{drifted}$  höchste *Specificity* von 99.99 % lässt sich dadurch erklären, dass der AE auf diesem Datensatz alle *Samples*, die keine Anomalie sind, auch als keine Anomalie prognostiziert hat. Dazu sei jedoch noch einmal angemerkt, dass  $\mathbf{X}_{drifted}$  keine Anomalien beinhaltet.

Neben den allgemeinen Kennzahlen aus Tabelle 5.4 wurde noch eine detailliertere Analyse des Modells  $M_2$  durchgeführt. Dazu wurde die *Performance* des Modells jeweils pro Art der Anomalie (Tabelle 5.8) sowie Art des *Concept Drifts* (Tabelle 5.6) untersucht. Im Folgenden werden die Erkenntnisse der Analyse dargestellt.

In Tabelle 5.5 sind die Kennzahlen des AE pro Art der Anomalie (0: keine Anomalie, 1: Punktanomalie, 2: Kollektive Anomalie, 3: Rauschen) und Datensatz  $\mathbf{X}_{test}$  und  $\mathbf{X}_{drifted,anormal}$  dargestellt. Die jeweils beste Kennzahl pro Zeile ist fett hervorgehoben. Der Tabelle kann entnommen werden, dass der AE auf beiden Datensätzen jeweils die Anomalieart 2 mit einer *Accuracy* von 99.07 % bzw. 98.11 % am besten erkennt. Die beiden anderen Anomaliearten 1 und 3 erkennt der AE dabei im Vergleich eher schlecht. Dabei ist jedoch zu berücksichtigen, dass ein leichtes Verrauschen der Daten (Anomalieart 3) ggf. nicht als Anomalie erkannt werden soll (siehe hierzu Abschnitt 2.3). Ein Grund für das bessere Identifizieren von kollektiven Anomalien könnte die Verwendung von rekurrenten ANN darstellen, da diese Informationen aus der Zeitreihe selber lernen können. Da jedoch keine rekurrenten ANN verwendet wurden und daher auch keine Nachbarschaftsinformationen aus Zeitreihen gelernt werden können, ist eine einfache Erklärung ohne Weiteres nicht möglich. Die hohe *Accuracy* von 99.99 % der Daten ohne Anomalie auf  $\mathbf{X}_{test}$  ist dadurch zu erklären, dass der Datensatz keine Anomalien beinhaltet und der AE alle *Samples* korrekterweise als negativ, also keine Anomalie, klassifiziert hat. Da in den Daten der Klasse 0 zu  $\mathbf{X}_{test}$  eben keine *Samples* mit Anomalien enthalten sind, sind auch die Kennzahlen *Precision*, *Sensitivity* und der *F1-Score* 0.00 %.

In Tabelle 5.6 sind die Kennzahlen des *AutoEncoders*  $M_2$  für die drei unterschiedlichen Arten zur Erzeugung eines *Concept Drifts* auf dem Datensatz  $\mathbf{X}_{drifted,anormal}$  dargestellt. Die jeweils beste

<b>Art der Anomalie:</b>	$\mathbf{X}_{test}$				$\mathbf{X}_{drifted,anormal}$			
	0	1	2	3	0	1	2	3
Anzahl Samples:	32543	20	2433	44	32543	20	2433	44
Accuracy (%):	<b>99.99</b>	14.99	98.07	6.82	89.82	19.99	98.11	27.27
Precision (%):	0.00	<b>99.99</b>	<b>99.99</b>	<b>99.99</b>	0.00	<b>99.99</b>	<b>99.99</b>	<b>99.99</b>
Specificity (%):	<b>99.99</b>	0.00	0.00	0.00	<b>99.99</b>	0.00	0.00	0.00
Sensitivity (%):	0.00	14.99	98.07	6.82	0.00	19.99	<b>98.11</b>	27.27
F1-Score (%):	0.00	26.09	99.02	12.77	0.00	33.33	<b>99.05</b>	42.86

Tabelle 5.5.: Kennzahlen des AutoEncoders  $M_2$  pro Art der Anomalie (0: keine Anomalie, 1: Punktanomalie, 2: Kollektive Anomalie, 3: Rauschen) und Datensatz  $\mathbf{X}_{test}$  und  $\mathbf{X}_{drifted,anormal}$

Kennzahl pro Zeile ist fett hervorgehoben. Im unteren Teil der Tabelle sind zusätzlich die prozentualen Anteile der Werte der *Confusion Matritzen* zu der Anzahl der *Samples* pro *Concept Drift* Event-Art dargestellt. Das Modell  $M_2$  hat für das Öffnen und Schließen eines Schalters (1) mit nur 0.35 % eine sehr schlechte *Accuracy*. Die niedrige *Precision* lässt sich dadurch erklären, dass das Modell alle *Samples*, die ein *Concept Drift Label* 1 tragen, als Anomalie klassifiziert hat, was sich auch in der hohen *Sensitivity* von 99.99 % widerspiegelt. Die *Accuracy* des Modells für die Manipulation des *Load Mappings* sowie die Veränderung des Phasenwinkels sind mit 98.85 bzw. 99.51 % im Vergleich zu den Kennzahlen auf  $\mathbf{X}_{test}$  auf einem Niveau. Die insgesamt schlechtere *Performance* des Modells  $M_2$  auf  $\mathbf{X}_{drifted,anormal}$  im Vergleich zu  $\mathbf{X}_{test}$  ist anhand der Kennzahlen durch die schlechte *Performance* des Modells für die *Concept Drift* Art 2 zu erklären. An dieser Stelle sei noch darauf hingewiesen, dass die Interpretation der prozentualen Anteile der Werte der *Confusion Matrices* pro Event-Art nur in Verbindung mit den Kennzahlen vorgenommen werden sollte. So klingt ein prozentualer Anteil von 0.90 % an TP zunächst sehr wenig. Jedoch machen die Anomalien mit 26 von 1112 *Samples* auch nur 2.33 % aus.

Zusammenfassend kann festgestellt werden, dass im Rahmen der Implementierung ein AE (Modell  $M_2$ ) trainiert werden konnte, der für die Erkennung von Anomalien auf einem normalen Datensatz  $\mathbf{X}_{test}$  eine insgesamt gute *Performance* aufweist und im Rahmen der Experimente für den Vergleich mit dem Meta-Modell  $M_1$  verwendet werden kann. Diese Feststellung bekräftigt die in Unterabschnitt 3.1.2 als Bedingung vorausgesetzte Annahme **A1**. Des Weiteren konnte gezeigt werden, dass der AE auf einem Datensatz  $\mathbf{X}_{drifted}$  eine höhere Rate von FP aufweist, was auf den künstlich erzeugten *Concept Drift* durch Manipulation der Daten zurückzuführen ist. Dies bestärkt die in Unterabschnitt 3.1.2 als Bedingung aufgestellte Annahme **A2**, sodass für beide Annahmen **A1** und **A2** Effekte im Rahmen der Implementierungen nachgewiesen werden konnten. Dariüber hinaus konnte festgestellt werden, dass der AE auf einem Datensatz  $\mathbf{X}_{drifted,anormal}$  eine bessere TP-Rate als auf  $\mathbf{X}_{test}$  aufweist, da der AE jene Anomalien erkennt, die zusätzlich mit einem *Concept Drift* manipuliert wurden. Bei der detaillierten Analyse der *Performance* des AE konnte außerdem gezeigt werden, dass nicht alle geplanten Events zur Erzeugung des *Concept Drifts* auch zu einem höheren FP-Rate des AE führen, sodass diese Events im Rahmen des *Meta-Learnings* ggf. nicht geeignet sind oder die Events verstärkt werden sollten. Des Weiteren haben die Arten des *Concept Drifts* jeweils stark unterschiedliche Auswirkungen auf die *Performance* des Modells. So konnte gezeigt werden, dass die Veränderung des *Load Mappings* sowie des Phasenwinkels nur minimale Auswirkungen auf die Kennzahlen des

<b>Kennzahl</b>	<b>Concept Drift Art</b>			
	0: Normal	1: Switch	2: Load Mapping	3: Cos Phi
<i>Anzahl Samples:</i>	1112	3132	17855	12941
<i>Anzahl Anomalien:</i>	26	11	2100	360
<i>Accuracy (%):</i>	98.47	0.35	98.85	<b>99.51</b>
<i>Precision (%):</i>	<b>99.99</b>	0.35	91.57	<b>99.99</b>
<i>Specificity (%):</i>	<b>99.18</b>	0.00	88.18	97.69
<i>Sensitivity (%):</i>	34.62	<b>99.99</b>	99.33	82.50
<i>F1-Score (%):</i>	51.43	0.69	<b>95.29</b>	90.41
<i>False Positives (%):</i>	0.00	99.65	1.06	0.00
<i>False Negatives (%):</i>	1.53	0.00	0.08	0.49
<i>True Positives (%):</i>	0.81	0.35	11.63	2.30
<i>True Negatives (%):</i>	97.66	0.00	87.23	97.21

Tabelle 5.6.: Kennzahlen des AutoEncoders  $M_2$  für die drei unterschiedlichen Arten zur Erzeugung eines Concept Drifts auf dem Datensatz  $X_{drifted,anormal}$

AEs haben, das Öffnen und Schließen eines Schalters jedoch stark negative Auswirkungen auf die Performance hat.

## 5.4. Implementierung des *Meta-Learnings*

Das *Meta-Learnings* umfasst zwei Phasen. Zunächst wird ein Modell  $M_1$  auf einem Datensatz  $X_{train}$  mittels eines *Meta-Learning*-Algorithmus so trainiert, dass es anhand der initial gelernten Parameter  $\theta_{f_{MAML}}$  einfach an neue Aufgaben adaptiert werden kann. Diese Phase wird von Finn, Abbeel und Levine als *Meta-Training* bezeichnet. Anschließend kann das zuvor trainierte Modell  $M_1$  in der *fine-tune* Phase mithilfe weniger  $k$  *Samples* an neue Aufgaben adaptiert werden. Für die Umsetzung im Rahmen dieser Arbeit bedeutet dies, dass in der *Meta-Training* Phase hingegen die gelernten Gewichte  $\theta$  des Modells zunächst zufällig initialisiert und der Datensatz  $X_{train}$  ohne *Concept Drift* verwendet werden. In der *fine-tune* Phase hingegen werden die gelernten Gewichte  $\theta_{f_{MAML}}$  aus der vorherigen Phase sowie ein Datensatz  $X_{drifted}$  verwendet.

Im Folgenden wird die Implementierung des *Meta-Learnings* mittels *l2l* für die beiden zuvor genannten Phasen beschrieben. In Unterabschnitt 5.4.1 wird zunächst die Implementierung des *Meta-Trainings* beschrieben. Anschließend wird in Unterabschnitt 5.4.2 das *Fine-tuning* des in 5.4.1 trainierten Modells  $M_1$  beschrieben. In Unterabschnitt 5.4.3 werden abschließend die Ergebnisse der Implementierung zusammengefasst und interpretiert.

### 5.4.1. *Meta-Training* Phase

Die Implementierung des *Meta-Trainings* besteht im Kern aus zwei Teilen. Bevor das eigentliche *Meta-Learning* durchgeführt werden kann, müssen die Daten zunächst für *l2l* vorbereitet werden. Dazu wird anhand zweier *PyTorch TensorDatasets* (in Listing 5.1 als Variablen `train_tensor_X`

und `test_tensor_X` dargestellt) je ein `121.data.MetaDataset` erstellt. Das `TensorDataset` umfasst dabei zum einen alle *Samples* (Variable `train_tensor_X`) des Datensatzes  $X_{train}$  und zum anderen *Labels* (Variable `train_tensor_y`), wobei im Rahmen der vorliegenden Arbeit keine *Labels* verwendet wurden, da im Rahmen des *Meta-Trainings* nicht zwischen unterschiedlichen Aufgaben  $\mathcal{T}$  unterschieden werden kann. Der Tensor für die *Labels* besteht daher aus einem Vektor, in dem jedes Element den Wert 1 trägt.

Mittels der erstellten `MetaDatasets` `train_metadata` und `test_metadata` können anschließend `TaskDatasets` erstellt werden. Die Verwendung der `TaskDatasets` `train` sowie `eval` ermöglicht das Sampeln und Aufteilen des Datensatzes  $X_{train}$  in *Tasks*, welche jeweils aus einem *n-way k-shot* Datensatz  $A$  zur Adaption an neue *Tasks* dienen sowie aus einem Evaluations-Datensatz  $E$  zur Adaption des gesamten Modells und zur Evaluation des Modells bzgl. der neuen Aufgaben, also  $X_{train} = \{A \cup E\}$ . Bei der Initialisierung eines `TaskDatasets` wird der Klasse die Liste `task_transforms` übergeben, welche die `121.data.transform` Klassen `NWays` und `KShots` mit Informationen bzgl. des *Samplings* der Daten beinhalten. Im Rahmen der Implementierung wird zunächst ein *1-way 5-shot Sampling* umgesetzt, so werden pro Task  $\mathcal{T}$  je fünf *Samples* im Rahmen der Adaption des Modells an eine neue Aufgaben  $\mathcal{T}_i$  verwendet. Es werden also pro Training Task  $\mathcal{T}_i$  zufällig fünf *Samples* aus  $A$  und ein *Sample* aus  $E$  gezogen. Die Reihenfolge der *Sample* spielt dabei keine Rolle. Die Mengen der *Samples* in  $A$  und  $E$  sollten jedoch disjunkt sein.

```

1 import learn2learn as l2l
2
3 train_tensor_X = torch.from_numpy(X_train).type(FloatTensor)
4 train_tensor_y = torch.from_numpy(y_train.reshape(len(y_train),1)).type(
5     FloatTensor)
6
7 test_tensor_X = torch.from_numpy(X_test).type(FloatTensor)
8 test_tensor_y = torch.from_numpy(y_test.reshape(len(y_test),1)).type(FloatTensor)
9
10 train_dataset = TensorDataset(train_tensor_X, train_tensor_y)
11 test_dataset = TensorDataset(test_tensor_X, test_tensor_y)
12
13 train_metadata = MetaDataset(train_dataset)
14 test_metadata = MetaDataset(test_dataset)
15
16 tf_train = [
17     NWays(train_metadata, n=1),
18     KShots(train_metadata, k=5),
19     LoadData(train_metadata),
20     num_tasks=128
21 ]
22
23 tf_eval = [
24     NWays(test_metadata, n=1),
25     KShots(test_metadata, k=1),
26     LoadData(test_metadata),
27     num_tasks=32
28 ]
29
30 train = TaskDataset(train_metadata, task_transforms=tf_train, num_tasks=128)
31 eval = TaskDataset(test_metadata, task_transforms=tf_eval, num_tasks=32)

```

Listing 5.1: Vorbereitung der Daten für *learn2learn*

Der zweite Teil der Implementierung des *Meta-Learnings* mittels *l2l* wird nach der Vorbereitung der Daten als TaskDatasets ausgeführt. Die Implementierung ist in Form der Klasse `Learner` umgesetzt, welche bei der Initialisierung alle Hyper-Parameter des *Meta-Learnings*, den AE und ein *Optimizer* sowie die notwendigen TaskDatasets zur Adaption und Evaluation erhält. Ein Auszug der Implementierung der Klasse ist im Anhang B in Listing B.7 dargestellt. Die Umsetzung in Form einer gekapselten Klasse ermöglicht es, schnell und einfach Experimente durchzuführen, da mehrere Instanzen der Klasse mit unterschiedlichen Parametern initialisiert werden können. Ein Auszug der *Meta-Learning* Implementierung mit *l2l* ist in Listing 5.2 im Form der Methode `start_learning_phase` dargestellt<sup>5</sup>. Zunächst werden die Gewichte des Modells mittels der *Xavier-Initialisierung*<sup>6</sup> initialisiert (Zeile 1). In der darauffolgenden Zeile 2 wird der in PyTorch entwickelte AE als *l2l* MAML Modell mit einer Lernrate von 0.001 und als FOMAML Version (Zeile 2 im Listing: `first_order = True`) initialisiert. Anschließend wird in Zeile 3 *Adam* als *Optimizer* mit den Parametern des Meta-Modells und einer Lernrate von 0.004 initialisiert<sup>7</sup>. Die beiden Werte der Lernraten wurden entsprechend der Empfehlung der Entwickler der *l2l* Bibliothek gewählt [Arn+19].

```

1 model = model.apply(SimpleAutoEncoder.weight_init)
2 meta_model = l2l.algorithms.MAML(model, lr=1e-3, first_order=True)
3 opt = Adam(meta_model.parameters(), lr=4e-3)
4
5 def start_learning_phase(self):
6     self.meta_model.module.train()
7
8     for _ in self.nun_iterations:
9         iteration_error = 0.0
10
11     # For Loop samples a number of tasks from the train data!
12     for task in self.train_task_data:
13         learner = self.meta_model.clone()
14
15         # Split data for adaption and evaluation
16         adaption_data = task[0]
17         adaption_data = adaption_data.to(self.device)
18
19         eval_data = self.eval_task_data.sample()[0]
20         eval_data = eval_data.to(self.device)
21
22         # Fast Adaption
23         for step in range(self.adaption_steps):
24             train_error = mse_loss(learner(adaption_data), adaption_data)
25             self.adaption_loss.append(train_error.item())
26             learner.adapt(train_error)
27
28         # Compute validation loss
29         predictions = learner(eval_data)
30         valid_error = mse_loss(predictions, eval_data)
31         valid_error /= len(eval_data)
32         iteration_error += valid_error

```

<sup>5</sup> Für ergänzende Informationen siehe Pseudocodedarstellung 3 in Unterabschnitt 2.4.2

<sup>6</sup> Die *Xavier-Initialisierung* ist neben der He-Initialisierung eine beliebte Art der Initialisierung der Gewichte eines ANN im Rahmen des DL. Für weitere Informationen zur *Xavier-Initialisierung* siehe: [GBC16]

<sup>7</sup> Die Initialisierung des Meta-Modelles sowie des Optimizers ist in Listing 5.2 nur schematisch dargestellt. Die eigentliche Initialisierung ist im Konstruktor der `Learner` Klasse implementiert.

```

33         self.evaluation_loss.append(valid_error.item())
34
35     # Take the meta-learning step
36     self.iteration_error_list.append(iteration_error.item())
37     self.optimizer.zero_grad()
38     iteration_error.backward()
39     self.optimizer.step()
40
41     # Save Weights for Analysis
42     self._update_weight_matrices_of_meta_model()

```

Listing 5.2: *Implementierung des Meta-Learnings mittels learn2learn in der Klasse Learner*

Zeile 5 bis 39 stellen den eigentlichen Teil der *Meta-Training* Phase dar. Ab Zeile 8 wird über eine definierte Anzahl an Schleifendurchläufen gelaufen. Die Schleife dient gleichzeitig als Abbruchkriterium des *Meta-Learnings*. In Zeile 13 wird das aktuelle Modell zunächst kopiert, sodass das kopierte Modell anschließend nur an die spezifischen Tasks aus dem Schleifendurchlauf adaptiert wird. In Zeile 16 bis 20 werden die Daten mittels der zuvor beschriebenen TaskDatasets gesampelt. In Zeile 23 bis 26 findet dann die Adaption des kopierten Modells an die spezifischen *Tasks* des Schleifendurchlaufes statt. Anschließend wird in Zeile 29 bis 33 der *Validation Loss* des zuvor adaptierten Modells anhand unabhängiger Evaluationsdaten dargestellt. In den Zeilen 36 bis 39 wird das eigentliche Meta-Modell dann anhand des Fehlers der unabhängigen Evaluationsdaten adaptiert. In der Zeile 42 werden vor Abschluss eines jeden Schleifendurchlaufes noch die Gewichte des adaptierten Meta-Modells für Analysezwecke gesichert.

In Abbildung 5.6 ist der *Evaluation Loss* des Meta-Modells während des *Meta-Learnings* über 400 Iterationen mittels FOMAML dargestellt. Der Abbildung kann entnommen werden, dass der Fehler des Meta-Modells  $M_1$  durch das *Meta-Learning* im Rahmen der *Meta-Training* Phase minimiert werden kann.

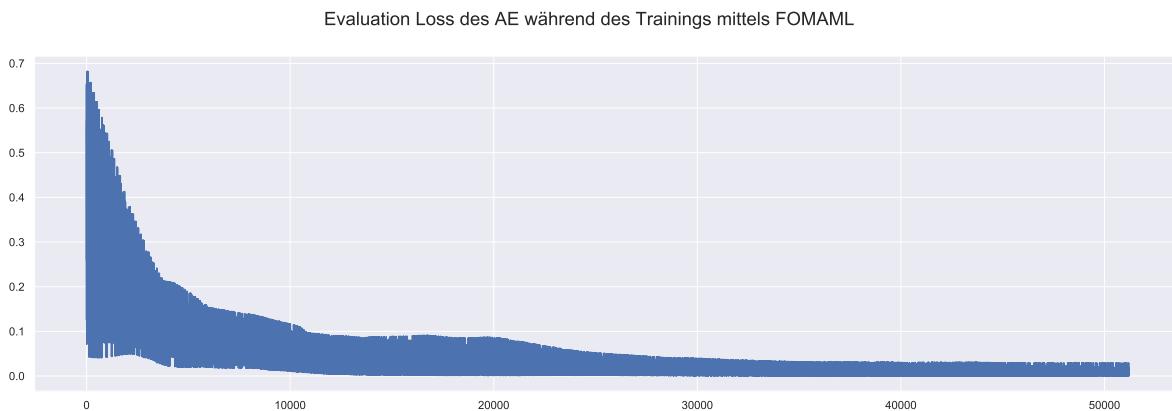


Abbildung 5.6.: *Evaluation Loss des Meta-Modells während des Meta-Learnings über 400 Iterationen mittels FOMAML (eigene Darstellung)*

Für die *Meta-Training* Phase wurde eine *Batch-Size* von 128 verwendet, sodass pro Iteration 128 *Tasks* aus  $\mathbf{A}$  gezogen werden. Da ein *1-way 5-shot Learning* implementiert wurde, besteht jeder der Tasks aus  $\mathbf{A}$  entsprechend aus fünf *Samples*. Die Phase des *Fast Adaptions* in der ein Modell an die gezogenen *Tasks* adaptiert wird, besteht entsprechend der Empfehlung des *l2l Frameworks* aus

fünf *Fast Adaption Steps* [Arn+19]. Die *Batch-Size* zur Evaluation und Adaption des Meta-Modells an die gezogenen Aufgaben in einer Iteration besteht aus 32 Aufgaben. Da für die Adaption ein *1-way 1-shot Learning* implementiert wurde, wird das Modell  $M_1$  pro Iteration also anhand von 32 *Samples* adaptiert. Die Werte zur Größe der *Batch-Sizes* für die Adaption und das *Meta-Training* wurden ebenfalls entsprechend der Empfehlungen der Entwickler der *l2l* Bibliothek gewählt. Nach dem Training des Modells  $M_1$  wurde nach gleichem Vorgehen, wie in Abbildung 5.5 beschrieben, eine Logistische Regression zur Klassifikation der Anomalien trainiert.

#### 5.4.2. *Fine-tune Phase*

Die zweite Phase des *Meta-Learnings* besteht aus dem *Fine-tuning*. Im Rahmen dieser Phase wird das zuvor mittels FOMAML trainierte Modell  $M_1$  mit den Gewichten  $\theta_{fMAML}$  aus der vorherigen Phase durch das Training mit wenigen  $k$  Samples an neue Aufgaben angepasst. Als Lernverfahren werden gewöhnliche, auf *Backpropagation*-basierende Algorithmen wie Adam oder SGD verwendet. [FAL17]

Die Funktionalität des *Fine-Tunings* wurde in der Klasse *FineTuner* implementiert. Dies ermöglicht, wie auch die Implementierung des *Meta-Trainings* in der Klasse *Learner*, die unkomplizierte und schnelle Durchführung mehrerer *fine-tuning* Durchläufe. Bei der Initialisierung der *FineTuner* Klasse wird dem Objekt zunächst ein PyTorch *TensorDataset*, die Anzahl der zu verwendenden *Samples*  $k$  als *Integer*, das PyTorch Modell mit den Gewichten  $\theta_{fMAML}$ , ein *String*, der angibt welcher Lernalgorithmus verwendet werden soll, eine Liste *drift\_classes*, mittels derer aus dem *TensorDataset* bestimmte *Concept Drift Events* anhand der *Label* gefiltert werden können sowie ein *Integer* *num\_iterations*, welcher bestimmt mit wie vielen Gradientenschritten im Rahmen des *Fine-Tunings* adaptiert werden soll, übergeben. In Anhang B ist in Listing B.8 der Quellcode der gesamten *FineTuner* Klasse dargestellt. In Listing 5.3 ist die Implementierung des *Fine-Tuning* als Auszug aus der *FineTuner* Klasse in Form der Methode *fine\_tune* dargestellt. Der Ablauf des *Fine-Tunings* entspricht im Prinzip der in Abschnitt 5.3 vorgestellten Implementierung des Trainings des AE (Modell  $M_2$ ) mittels Adam.

```

1 def fine_tune(self):
2     self._sample_fine_tune_data()
3     self.model.train()
4
5     for _ in range(self.num_iterations):
6         inner_loss = 0.0
7         avg_inner_loss = 0.0
8         for i, (x, y) in enumerate(zip(self.X, self.drift_label)):
9             x = torch.autograd.Variable(x)
10            self.optimizer.zero_grad()
11
12            pred = self.model(x)
13            loss = mse_loss(pred, x)
14            self.all_losses_fine_tune.append(loss.item())
15
16            inner_loss += loss.item()
17            avg_inner_loss = inner_loss / self.k
18
19            # Backpropagation
20            loss.backward()
21            self.optimizer.step()
```

```

22
23     self.avg_losses_fine_tune.append(avg_inner_loss)

```

Listing 5.3: Auszug der Implementierung des Fine-Tunings mittels Adam in der Klasse Learner

Zunächst wird in Zeile 2 anhand des TensorDatasets, der Anzahl der zu verwendenden *Samples*  $k$  sowie der Liste *drift\_classes* ein Datensatz (in der Klasse zugreifbar durch *self.X*) für das Fine-Tuning erstellt. In der Zeile 3 wird das Modell  $M_1$  in den Trainingsmodus gesetzt, sodass Gradientenschritte durch *Adam* oder SGD auch Veränderungen an den Gewichten bewirken können. Von Zeile 5 bis 23 wird die eigentliche Adaption durchgeführt. Diese wird durch eine äußere Schleife in Zeile 5 gesteuert, welche bestimmt, über wie viele Iterationen die Adaption durchgeführt werden soll. Für jeden Schleifendurchlauf werden anschließend in den Zeilen 6 und 7 die Variablen *inner\_loss* und *avg\_inner\_loss* mit 0.0 initialisiert. Die Variablen werden verwendet, um den Fehler sowie den durchschnittlichen Fehler des Modells pro Iteration zu speichern. Der weitere Ablauf in der Schleife in den Zeilen 8 bis 15 sowie 20 und 21 entspricht dem bereits bekanntem klassischen Training eines Modells in PyTorch. Für ein erstes Fine-Tuning wurde *Adam* als Lernalgorithmus mit einer Lernrate von 0.001,  $k = 5$  sowie eine Anzahl an 1000 Iterationen verwendet. Außerdem wurden nur *Samples* aus  $\mathbf{X}_{\text{drifted}}$  genutzt, welche mittels der *NetworkTopologyChanger* Komponente so manipuliert wurden, dass sie zu einem *Concept Drift* führen, also das Label 1, 2 oder 3 tragen. Wie auch im Rahmen der Phase des *Meta-Trainings* wurden die Werte der Parameter des Fine-Tunings entsprechend der Empfehlungen der Entwickler der *l2l* Bibliothek gewählt.

#### 5.4.3. Ergebnisse

Im Folgenden sollen die Ergebnisse in Form der Kennzahlen des Modells  $M_1$  nach dem *Meta-Training* dargestellt werden. Hierzu wird das Modell zunächst bzgl. seiner allgemeinen Performance auf den drei verwendeten Datensätzen untersucht. Anschließend wird noch eine detailliertere Analyse der Performance des Modells hinsichtlich der Performance je nach Art der Anomalie und nach Art des *Concept Drift Events* pro Datensatz vorgenommen.

In Tabelle 5.7 sind die allgemeinen Kennzahlen des Modells  $M_1$  nach dem Training mittels des *Meta-Learning*-Algorithmus FOMAML und den in Unterabschnitt 5.4.1 beschriebenen Hyperparametern dargestellt. Zusätzlich sind in Abbildung A.7 im Anhang A die drei *Confusion Matrices* des Modells  $M_1$  dargestellt. Der Tabelle kann entnommen werden, dass das Modell  $M_1$  auf dem Datensatz  $\mathbf{X}_{\text{test}}$  seine mit Abstand beste Performance aufweist. Die hohe *Specificity* von 99.99 % auf  $\mathbf{X}_{\text{drifted}}$  lässt sich dadurch erklären, dass in dem Datensatz keine Anomalien vorhanden sind. Die insgesamt schlechteste Performance hat  $M_1$  auf  $\mathbf{X}_{\text{drifted}, \text{anormal}}$ . Dies lässt sich vermutlich durch den erzeugten *Concept Drift* erklären. So konnte das Modell nur noch 41.18 % aller Anomalien korrekt als solche erkennen. Darüber hinaus hat das Modell auf  $\mathbf{X}_{\text{drifted}}$  wesentlich mehr (3313 zu 0) FP prognostiziert, als auf  $\mathbf{X}_{\text{test}}$ , was ebenfalls ein Indiz für einen *Concept Drift* darstellt. Die Zahl der FN auf  $\mathbf{X}_{\text{drifted}, \text{anormal}}$  ist mit 178 zu 166 *Samples* hingegen nur unwesentlich höher als auf  $\mathbf{X}_{\text{test}}$ .

In Tabelle 5.8 sind die Kennzahlen des Modells  $M_1$  pro Art der Anomalie und Datensatz dargestellt. Die jeweils beste Kennzahl ist pro Zeile fett hervorgehoben. Wie auch das Modell  $M_2$ , dessen Kennzahlen in der Tabelle 5.5 auf die gleiche Art analysiert wurden, erkennt das mittels FOMAML trainierte Modell  $M_1$  kollektive Anomalien wesentlich besser als Punktanomalien oder Rauschen. Wohingegen das Modell  $M_2$  kollektive Anomalien in  $\mathbf{X}_{\text{drifted}, \text{anormal}}$  minimal besser als in  $\mathbf{X}_{\text{test}}$

<b>Kennzahl</b>	<b>Datensätze</b>		
	$X_{test}$	$X_{drifted}$	$X_{drifted,anormal}$
<b>Accuracy:</b>	<b>99.53 %</b>	90.51 %	90.04 %
<b>Precision:</b>	<b>99.99 %</b>	0.0 %	41.18 %
<b>Specificity:</b>	93.32 %	<b>99.99 %</b>	92.65 %
<b>Sensitivity:</b>	<b>93.35 %</b>	0.0 %	92.87 %
<b>F1-Score:</b>	<b>96.56 %</b>	0.0 %	57.05 %

Tabelle 5.7.: Kennzahlen zur Performance des Meta-Modell AutoEncoders  $M_1$  auf drei unterschiedlichen Datensätzen zur Evaluation. Die jeweils beste Kennzahl ist fett hervorgehoben.

erkennen konnte, trifft dies für das  $M_1$  nicht zu. So erkennt  $M_1$  kollektive Anomalien in  $X_{test}$  minimal (+0.78 %) besser. Insgesamt hat das Modell  $M_1$  auf  $X_{test}$  eine bessere Performance, was vermutlich auf den künstlich erzeugten *Concept Drift* zurückzuführen ist.

<b>Art der Anomalie:</b>	$X_{test}$				$X_{drifted,anormal}$			
	0	1	2	3	0	1	2	3
Anzahl Samples:	32543	20	2433	44	32543	20	2433	44
Accuracy (%):	<b>99.99</b>	9.99	95.48	13.64	89.82	14.99	94.70	27.27
Precision (%):	0.00	<b>99.99</b>	<b>99.99</b>	<b>99.99</b>	0.00	<b>99.99</b>	<b>99.99</b>	<b>99.99</b>
Specificity (%):	<b>99.99</b>	0.00	0.00	0.00	<b>99.99</b>	0.00	0.00	0.00
Sensitivity (%):	0.00	9.99	<b>95.48</b>	13.64	0.00	14.99	94.70	27.27
F1-Score (%):	0.00	18.18	<b>97.69</b>	23.99	0.00	26.09	97.28	42.86

Tabelle 5.8.: Kennzahlen des AutoEncoders  $M_1$  pro Art der Anomalie (0: keine Anomalie, 1: Punktanomalie, 2: Kollektive Anomalie, 3: Rauschen) und Datensatz  $X_{test}$  und  $X_{drifted,anormal}$  vor dem Fine-Tuning

In Tabelle 5.9 sind die Kennzahlen sowie die Werte der *Confusion Matrices* des AutoEncoders  $M_1$  pro *Concept Drift* Event-Art (0: keine Manipulation, 1: Schalter Öffnen/Schließen, 2: Manipulation des *Load Mappings*, 3: Manipulation des Phasenwinkels) auf dem Datensatz  $X_{drifted,anormal}$  dargestellt. Die jeweils beste Kennzahl pro Zeile ist fett hervorgehoben. Im unteren Teil der Tabelle sind zusätzlich die prozentualen Anteile der Werte der *Confusion Matrices* zu der Anzahl der *Samples* pro *Concept Drift* Event-Art erkennbar. Es ist zu sehen, dass das Modell für die Event-Art 3 die insgesamt beste *Accuaracy* aufweist. Die Manipulation der Daten mittels des *Concept Drift* Events 1: „Öffnen bzw. Schließen eines Schalters“ scheint den größten negativen Effekt auf das Modell zu haben, da die Kennzahlen für diese Event-Art insgesamt am niedrigsten sind. Diese Annahme wird wiederum durch die hohe Anzahl der FP von 99.65 % gestützt. Das Modell hat demnach nahezu alle der 3132 *Samples*, die das *Label* der Event-Art 1 tragen, fälschlicherweise als Anomalie klassifiziert. Die Veränderung der Daten durch eine Veränderung des *Load Mappings* (2) oder des Phasenwinkels (3) scheint jedoch so gut wie keine Auswirkungen auf das Modell  $M_1$  zu haben. So weicht die *Accuracy* mit 98.74 bzw. 98.99 % nur minimal (+0.18 bzw. +0.43) von den *Samples* ohne Veränderung (0) ab bzw. hat sogar eine positive Auswirkung auf die Genauigkeit des Modells. Dies ist wieder-

um auch an dem geringen prozentualen Anteil der falsch klassifizierten *Samples* (FP bzw. FN) pro *Concept Drift* Event-Art 2 und 3 zu sehen. An dieser Stelle sei noch darauf hingewiesen, dass die Interpretation der prozentualen Anteile der Werte der *Confusion Matrices* pro Event-Art nur in Verbindung mit den Kennzahlen vorgenommen werden sollte. So klingt ein prozentualer Anteil von 0.90 % an TP zunächst sehr wenig. Jedoch machen die Anomalien mit 26 von 1112 *Samples* auch nur 2.33 % aus.

<b>Kennzahl</b>	<b>Concept Drift Art</b>			
	0: Normal	1: Switch	2: Load Mapping	3: Cos Phi
Anzahl Samples:	1112	3132	17855	12941
Anzahl Anomalien:	26	11	2100	360
Accuracy (%):	98.56	0.35	98.74	<b>98.99</b>
Precision (%):	<b>99.99</b>	0.35	91.50	<b>99.99</b>
Specificity (%):	<b>99.09</b>	0.00	88.27	98.20
Sensitivity (%):	34.46	<b>99.99</b>	98.48	63.89
F1-Score (%):	55.55	0.69	<b>94.86</b>	77.97
False Positives (%):	0.00	99.65	1.08	0.00
False Negatives (%):	1.43	0.00	0.08	1.00
True Positives (%):	0.90	0.35	11.68	1.78
True Negatives (%):	97.67	0.00	87.16	96.22

Tabelle 5.9.: Kennzahlen des AutoEncoders  $M_1$  für die drei unterschiedlichen Arten zur Erzeugung eines Concept Drifts auf dem Datensatz  $X_{drifted,anormal}$

Das Ergebnis der *Meta-Training* Phase ist ein Modell  $M_1$ , welches mittels des *Meta-Learning*-Algorithmus FOMAML trainiert wurde und im Rahmen einer *fine-tune* Phase einfach an neue Aufgaben adaptiert werden kann. Alle dafür benötigten Komponenten wurden ebenfalls im Rahmen dieses Abschnitts gezeigt und erläutert. In Bezug auf die Analyse der Ergebnisse konnte gezeigt werden, dass ein initial mittels FOMAML trainiertes Modell  $M_1$  eine ähnlich gute Performance bei der Identifizierung von Anomalien aufweist, wie das zuvor beschriebene Modell  $M_2$ . Auch die in Unterabschnitt 5.3.2 bereits festgestellten Bekräftigungen der Annahmen durch  $M_2$  treffen für das Modell  $M_1$  zu. Insgesamt sind sich beide Modelle hinsichtlich der Kennzahlen sehr ähnlich. Dies liegt vermutlich darin begründet, dass für beide Modelle die gleiche Initialisierung der Parameter sowie die identische Netztopologie verwendet wurden. Dennoch unterscheiden sich beide Modelle hinsichtlich ihrer gelernten Gewichtsmatrizen. Dies konnte im Rahmen einer kurzen Analyse, bei der die Differenz der vier Matrizen berechnet wurde, gezeigt werden.

## 5.5. Wrapper-Klassen für die Durchführung der Experimente

Damit im Rahmen der Durchführung der Experimente möglichst unkompliziert viele Durchläufe mit unterschiedlichen Parameterkombinationen gestartet und ausgewertet werden können, wurden jeweils für die Phase des *Meta-Trainings* sowie des Fine-Tunings die zwei Wrapper-Klassen *MetaTrainingExperiment* sowie *FineTuneExperiment* implementiert. Während die beiden Klassen *Learner* und *FineTuner* die eigentliche Funktionalität des *Meta-Trainings* bzw. Fine-Tunings implementieren, bilden die beiden Wrapper-Klassen durch die Steuerung des Ablaufs einen Rahmen

um die Klassen. Das abstrakte Schema beider Wrapper-Klassen inkl. der Parameter ist in Abbildung 5.7 dargestellt.

Die Klasse `MetaTrainingExperiment` bildet einen Rahmen für die `Learner` Klasse, mittels derer das *Meta-Training* durchgeführt werden kann. Die Wrapper-Klasse wird zunächst mit allen notwendigen Parametern für einen Durchlauf eines Experimentes initialisiert. Der gesamte Durchlauf wird anschließend von der Wrapper-Klasse gesteuert. So werden anhand der übergebenen Parameter zunächst die `121.data.MetaDatasets` sowie das `121.algorithms.MAML` Modell vorbereitet. Anschließend wird ein Modell anhand des in Unterabschnitt 5.4.1 beschriebenen Ablaufs trainiert und auf den Datensätzen  $X_{test}$  sowie  $X_{drifted,anormal}$  evaluiert. Alle Artefakte eines Durchlaufes, wie das trainierte Modell und alle Kennzahlen der Evaluation, werden anschließend eindeutig identifizierbar gespeichert.

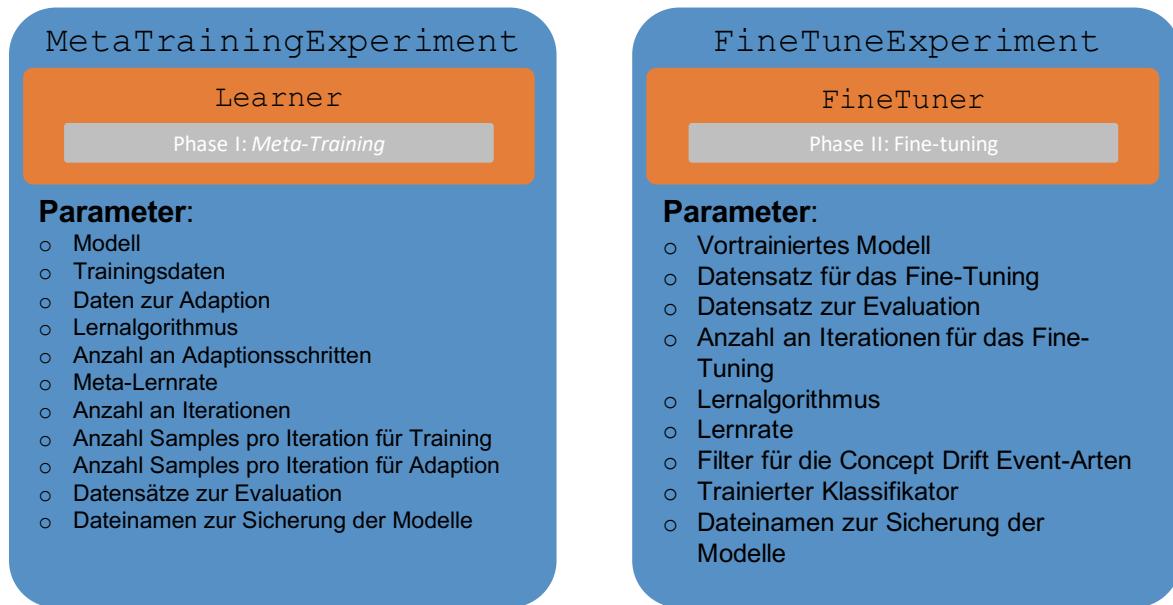


Abbildung 5.7.: *Schema der Wrapper-Klassen `MetaTrainExperiment` und `FineTuneExperiment` (eigene Darstellung)*

Die Klasse `FineTuneExperiment` bildet hingegen einen Rahmen für die Klasse `FineTuner` und steuert den Ablauf des Fine-Tunings, welcher in Unterabschnitt 5.4.2 beschrieben wurde. Auch diese Wrapper-Klasse wird zunächst mit allen notwendigen Parametern initialisiert. Anschließend kann über den Aufruf der Funktion `run()` der Lauf eines Experimentes gestartet werden.



## 6. Experiment

Das Wort Experiment stammt von dem lateinischen Begriff *experimentum* ab, was soviel bedeutet wie „das in Erfahrung gebrachte, der Versuch, die Probe, der Beweis“. Im Sinne der Wissenschaft ist ein Experiment eine methodisch angelegte Untersuchung zur empirischen Gewinnung von Informationen [Pre+95].

In den folgenden Abschnitten werden die Durchführung sowie die Ergebnisse und Erkenntnisse des Experimentes zur Beantwortung der Forschungsfragen beschrieben. Hierzu wird in Abschnitt 6.1 kurz die Motivation und Zielstellung des Experimentes erläutert. Anschließend wird in Abschnitt 6.2 der Versuchsaufbau und -ablauf des Experimentes dargestellt. In Abschnitt 6.3 wird die statistische Versuchsplanung inkl. der Erstellung und Einschränkung der Versuchspläne dargelegt. Im darauf folgenden Abschnitt 6.4 wird die Durchführung des Experimentes beschrieben. In Abschnitt 6.5 werden die Ergebnisse anschließend ausgewertet und interpretiert. Abschließend wird in Abschnitt 6.6 die Evaluation durch Überprüfung der Hypothese 1 anhand der in Abschnitt 6.5 gewonnenen Erkenntnisse vorgenommen.

### 6.1. Motivation und Zielstellung

Wie im bisherigen Verlauf der vorliegenden Arbeit bereits gezeigt wurde, kann ein *Concept Drift* negative Auswirkungen auf die Leistung eines ML-Modells haben<sup>1</sup>. Im Rahmen der vorliegenden Arbeit soll daher untersucht werden, ob *Meta-Learning* Ansätze die negativen Auswirkungen auf die Leistung eines ML-Modells minimieren können. Im Rahmen der Untersuchung wurden in Abschnitt 3.1 zunächst Forschungsfragen definiert, anhand derer anschließend eine untersuchbare Hypothese **H1** abgeleitet wurde. Das Ziel der in Kapitel 6 beschriebenen Tätigkeiten ist die Durchführung eines Experimentes, zur Untersuchung der zuvor definierten Hypothese.

### 6.2. Versuchsaufbau & -ablauf

Die Idee zur Untersuchung der Hypothese im Rahmen eines Experimentes wurde in der vorliegenden Arbeit bereits an mehreren Stellen skizziert. Im Folgenden soll noch einmal die Grundidee zur Untersuchung der Forschungsfragen sowie der generelle Ablauf erläutert werden. Hierzu ist in Abbildung 6.1 der Versuchsaufbau inkl. der zwei relevanten Phasen I und II sowie der Messzeitpunkte  $t_i$  zur Evaluation der Modelle dargestellt. Die Grundidee ist dabei folgende: In der Phase I werden Modelle mittels zweier unterschiedlicher Ansätzen auf dem gleichen Datensatz  $\mathbf{X}_{train}$  trainiert. Das Modell  $M_2$  beschreibt dabei jenes Modell, welches mittels eines klassischen Lernalgorithmus auf  $\mathbf{X}_{train}$  trainiert wurde. Im Rahmen dieser Phase wird nur ein Modell  $M_2$  erstellt, welches eine möglichst gute Performance aufweist. Für den *Meta-Learning* Ansatz werden hingegen eine Vielzahl unterschiedlicher Modelle mit unterschiedlichen Parameterkombinationen trainiert. Das Training mehrerer Modelle  $M_2$  wird deshalb durchgeführt, da einerseits noch wenig Erfahrung mit dem Training von *Meta-Learning*-Modellen vorhanden war und da andererseits angenommen wird, dass die Fähigkeit eines *Meta-Learning*-Modells einfach an neue Aufgaben adaptierbar zu sein, maßgeblich von den Parametern des *Meta-Learnings* abhängig ist. Durch ein Training mehrerer verschiedener Modelle

---

<sup>1</sup> Siehe hierzu beispielsweise Unterabschnitt 2.1.4

mit jeweils unterschiedlichen Parametern soll insbesondere verhindert werden, dass das Experiment durch falsch gewählte *Meta-Learning*-Parameter beeinflusst wird. Die Vielzahl der Modelle des *Meta-Learning* Ansatzes werden zur Vereinfachung weiterhin mit  $M_2$  gekennzeichnet, wenngleich damit mehrere Modelle identifiziert werden sollen. Anschließend an das Training werden die Modelle in  $t_1$  hinsichtlich ihrer Performance auf den Trainingsdaten  $X_{train}$  evaluiert. Um die Auswirkungen des *Concept Drifts* zu evaluieren, werden in  $t_2$  beide Modelle jeweils auf einem Datensatz  $X_{drifted}$  evaluiert. Dieser Datensatz umfasst keine Anomalien, aber verschiedene Events zur Erzeugung eines *Concept Drifts*. Durch die Messung der FP, also der fälschlicherweise als Anomalie klassifizierten *Samples*, kann auf die Stärke des *Concept Drifts* geschlossen werden. Die Evaluation der Modelle in  $t_2$  stellt den Abschluss der Phase I dar. In der darauf folgenden Phase II werden die Modelle jeweils mit nur wenigen *Samples* aus  $X_{drifted}$  an den Datensatz adaptiert und in Messzeitpunkt  $t_3$  evaluiert. Im Rahmen der zweiten Phase wird darüber hinaus in  $t_4$  eine Evaluation der Modelle auf dem Datensatz  $X_{drifted,ano}$  durchgeführt. Dieser Datensatz umfasst neben den gleichen *Concept Drift* Events wie  $X_{drifted}$  zusätzlich Anomalien. Sollte das Modell  $M_1$  im Rahmen der zweiten Phase hinsichtlich seiner Performance besser abschneiden als  $M_2$ , ist dies ein Indiz für die im Rahmen der vorliegenden Arbeit aufgestellten Hypothese **H1**.

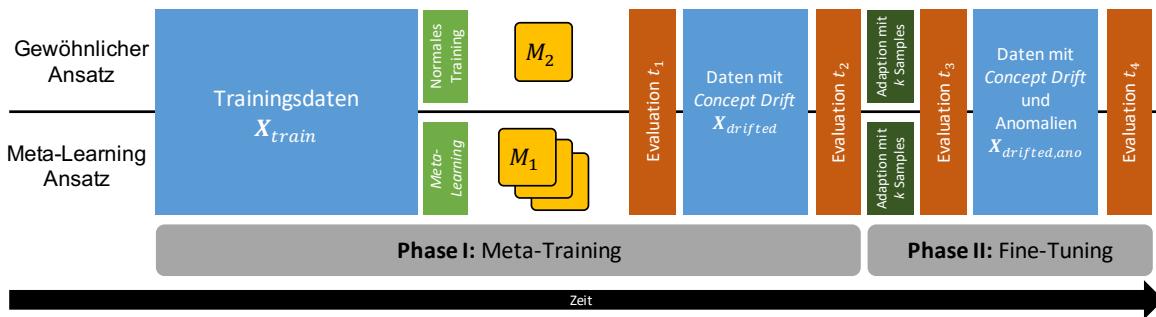


Abbildung 6.1.: Versuchsablauf inkl. der relevanten Phasen, Modelle und Messzeitpunkte (eigene Darstellung)

### 6.3. Statistische Versuchsplanung

Nach Siebertz, Bebber und Hochkirchen ist die statistische Versuchsplanung eine Methode zur effizienten Planung und Auswertung von Versuchsreihen [SBH10]. Neben der Strukturierung des Vorgehens umfasst die statistische Versuchsplanung auch weitere Methoden, wie z.B. zur Bestimmung der notwendigen Stichprobengröße oder zur Interpretation des Einflusses und den Zusammenhängen verschiedener Faktoren durch Beschreibungsmodelle. Darüber hinaus bietet die statistische Versuchsplanung hilfreiche Werkzeuge zur Erstellung von Versuchsplänen.

Die Sammlung der Parameter sowie die Auswahl von Faktoren und die Erstellung eines Versuchsplanes für ein Experiment werden folgend anhand der Grundideen verschiedener Methoden der statistischen Versuchsplanung vorgenommen. Dabei ist das Ziel nicht die vollständige und korrekte Anwendung dieser Methoden, sondern vielmehr die Strukturierung und Steigerung der Nachvollziehbarkeit während des Experiments. Im Rahmen der Durchführung der statistischen Versuchsplanung werden zunächst alle Parameter des Experiments zusammengetragen und, sofern notwendig, erläutert. Anschließend wird ein vollfaktorieller Versuchsplan anhand der Faktoren erstellt. Da die Durchführung

des Experimentes unter Berücksichtigung aller Parameterkombinationen aus verschiedenen Gründen nicht möglich ist, wird anschließend mittels einer eigenen Methodik ein eigener teilstatischer Versuchsplan erstellt. Bei der Verwendung der verschiedenen Grundbegriffe werden die von Siebertz, Bebber und Hochkirchen in [SBH10] vorgenommenen Definitionen verwendet.

### 6.3.1. Sammlung aller Parameter

Die Sammlung der Parameter wird folgend jeweils getrennt für die beiden Phasen I und II des Experimentes vorgenommen. Für jeden Parameter werden die möglichen Werteausprägungen inkl. der Quelle der Empfehlungen für diese Werteausprägungen genannt. Im Rahmen der Phase I werden die in Tabelle 6.1 dargestellten Parameter verwendet. Die Sammlung umfasst insgesamt sieben unterschiedliche Parameter. Bei den Parametern handelt es sich um jene Parameter, welche im Rahmen des *Meta-Trainings* einen Einfluss auf die Modelle  $M_2$  haben. Die Parameter des AE zur Anomalieerkennung werden im Rahmen der Experimente nicht berücksichtigt, da diese bereits in der im Vorhinein durchgeführten Vorstudie evaluiert wurden. Daher weisen sowohl das Modell  $M_2$  als auch die Modelle  $M_2$  die identische AE-Topologie auf. An dieser Stelle sei jedoch angemerkt, dass der kontinuierliche Verwendung der gleichen AE-Topologie für  $M_1$  und  $M_2$  die Annahme zugrunde liegt, dass die AE-Topologie (insbesondere in Form der Anzahl der Neuronen in den *Hidden-Layern*) keinen nennenswerten Einfluss auf die Fähigkeit eines Modells, einfach an neue Aufgaben adaptierbar zu sein, hat. Eine kurze Literaturrecherche konnte dies weder belegen noch widerlegen. Dennoch sollte diese Annahme durch eine detailliertere Analyse kritisch betrachtet werden.

Parameter	Ausprägung	Quelle der Empfehlung
Lernalgorithmus	{Adam, SGD}	Erfahrung Vorstudie
Anzahl an Adaptionsschritten	{5, 10, 15}	[FAL17]
$k$	{5, 10, 20}	[FAL17]
<i>Samples</i> pro It. (Training)	{32, 64, 128}	[Arn+19]
<i>Samples</i> pro It. (Evaluation)	{16, 32}	[Arn+19]
Anzahl an Iterationen	{100, 300, 500}	[Arn+19]
Meta-Lernrate	{0.001, 0.01, 0.1}	[Arn+19]

Tabelle 6.1.: *Parameter der Phase I inkl. möglicher Werteausprägungen sowie der Quelle für die Empfehlung der Werteausprägungen*

In Tabelle 6.2 ist die Sammlung aller Parameter der Phase II des Experimentes dargestellt. Insgesamt bestimmen sechs Parameter die Phase II des Experimentes. Der Parameter „Anzahl der Modelle aus Phase I“ ist nur für die Modelle  $M_1$  und nicht für  $M_2$  von Relevanz. Die Modelle  $f_1^{min}$ ,  $f_2^{min}$  und  $f_3^{min}$  entsprechen dabei den drei Modellen mit der niedrigsten *Precision* auf  $X_{drifted,anormal}$  in Phase I. Die Modelle  $f_4^{max}$ ,  $f_5^{max}$  und  $f_6^{max}$  entsprechen hingegen den drei Modellen mit der höchsten *Precision* auf  $X_{drifted,anormal}$  in Phase I. Die Betrachtung der drei jeweils schlechtesten und besten Modelle soll zwei unterschiedliche Blickwinkel bei der Untersuchung ermöglichen. Die Beschreibung  $P(\{1, 2, 3\}) \setminus \emptyset = |7|$  des Wertebereichs von Parameter Filter *Concept Drift* Events bezeichnet die Potenzmenge der drei unterschiedlichen *Concept Drift* Event-Typen ohne die leere Menge  $\emptyset$ , sodass insgesamt sieben unterschiedliche Kombinationen aus  $P$  resultieren. Die Parameter

der Lernrate, der Anzahl der Iterationen sowie der Anzahl an Samples zur Adaption ( $k$ ) im Rahmen des Fine-Tunings wurden entsprechend der Empfehlungen aus [Arn+19] bzw. [FAL17] gewählt.

Parameter	Ausprägung	Quelle der Empfehlung
Lernalgorithmus	{Adam, SGD}	Erfahrung Vorstudie
Filter <i>Concept Drift</i> Events	$\{P(\{1, 2, 3\}) \setminus \emptyset\}$	Erfahrung Implementierung
Modelle $M_1$ aus Phase I	$\{f_1^{\min}, f_2^{\min}, f_3^{\min}, f_4^{\max}, f_5^{\max}, f_6^{\max}\}$	Erfahrung Implementierung
Samples zur Adaption ( $k$ )	{5, 10, 20}	[FAL17]
Anzahl Iterationen	{1, 100, 500}	[Arn+19]
Lernrate	{0.001, 0.01, 0.1}	[Arn+19]

Tabelle 6.2.: Parameter der Phase II inkl. möglicher Werteausprägungen sowie der Quelle für die Empfehlung der Werteausprägungen

### 6.3.2. Versuchspläne

Anhand der zuvor gesammelten Parameter sollen für die Phasen I und II nun Versuchspläne erstellt werden. Insgesamt müssen dafür drei unterschiedliche Versuchspläne erstellt werden. Entsprechend ein Versuchspran für das Training der Modelle  $M_1$  in Phase I und je ein Versuchspran für  $M_1$  und  $M_2$  für Phase II. Hierfür werden zunächst die vollfaktoriellen Versuchspläne erstellt und anschließend eine Minimierung dieser durch Erstellung teilst faktorieller Versuchspläne motiviert und durchgeführt. Der Versuchspran der Phase I für die Modelle  $M_1$  wird dabei als  $VP_{M_1}^I$ , der Versuchspran für  $M_2$  in Phase II als  $VP_{M_2}^{II}$  und der Versuchspran für  $M_1$  in Phase II entsprechend als  $VP_{M_1}^{II}$  gekennzeichnet. Die Präfixe  $v$  und  $t$  kennzeichnen die Versuchspläne zusätzlich als voll- bzw. teilst faktoriell.  $tVP_{M_1}^{II}$  identifiziert beispielsweise den teilst faktoriellen Versuchspran in Phase II der Modelle  $M_1$ . Im Kontext der statistischen Versuchspranung bezeichnet der Begriff Parameter die Menge aller Eingangsgrößen. Die im Versuchspran enthaltenen Parameter werden als Faktoren bezeichnet [SBH10]. Da bei der Durchführung des Experiments alle Parameter verwendet werden, können beide Begriffe im Folgenden synonym verwendet werden.

#### Vollfaktorielle Versuchspläne

Bei einem vollfaktoriellen Versuchspran (auch Vollfaktorplan genannt) werden alle Kombinationen aus den Faktoren getestet. Der Versuchsaufwand  $n_r$  ergibt sich aus der Zahl der Faktoren  $n_f$  und der Zahl der Stufen  $n_l$ , also:  $n_r = n_l^{n_f}$  [SBH10]. Bei sieben Faktoren auf zwei Stufen ergeben sich entsprechend 128 Kombinationen. Das Hinzufügen eines weiteren Faktors sorgt dabei für einen exponentiellen Anstieg der Größe des Versuchsprans. So gibt es für acht Faktoren auf zwei Stufen bereits 256 Kombinationen. Um die Anzahl der Kombinationen der Faktoren mit einer unterschiedlichen Anzahl an Stufen zu bestimmen, lässt sich einfach das kartesische Produkt aus den Mengen der Werteausprägungen der Faktoren bestimmen. Für die Phase II ergibt sich der vollfaktorielle Versuchspran demnach folgendermaßen:

$$\begin{aligned} vVP_{M_1}^{II} = \{\text{Adam, SGD}\} \times \{P(\{1, 2, 3\}) \setminus \emptyset\} \times & \{f_1^{\min}, f_2^{\min}, f_3^{\min}, f_4^{\max}, f_5^{\max}, f_6^{\max}\} \\ & \times \{5, 10, 20\} \times \{1, 100, 500\} \times \{0.001, 0.01, 0.1\} \end{aligned} \quad (6.1)$$

Die Größe der Versuchspläne lässt sich anschließend durch die Berechnung der Mächtigkeit der Mengen  $vVP_{M_1}^I$ ,  $vVP_{M_1}^{II}$  und  $vVP_{M_2}^{II}$  bestimmen:

$$|vVP_{M_1}^I| = 972 \quad (6.2)$$

$$|vVP_{M_1}^{II}| = 2268 \quad (6.3)$$

$$|vVP_{M_2}^{II}| = 378 \quad (6.4)$$

Die Durchführung von insgesamt  $3618 = 972 + 2268 + 378$  Versuchsläufen ist im Rahmen der vorliegenden Arbeit nicht durchführbar. Das Training eines Modells  $M_1$  im Rahmen des *Meta-Trainings* hat bei ersten Tests während der Implementierung im Schnitt 15 Minuten benötigt. Alleine das vollständige Training aller 972 Modelle  $M_1$  in Phase I würde demnach bereits ca. zehn Tage benötigen. Die Adaption eines Modells in Phase II hat während der Implementierung im Schnitt zwei Minuten benötigt. Die Adaption aller 2268 Modelle würde ca. drei Tage benötigen. Da beide Phasen nur nacheinander durchzuführen sind, ergäbe sich so eine Gesamtdauer von  $\approx 13$  Tagen für die Phasen I und II.

## Teilfaktorielle Versuchspläne

Die statistische Versuchsplanung bietet mit ihren Methoden verschiedene Ansätze um den Versuchsaufwand zu verringern, wobei dennoch viele Faktoren und nichtlineare Zusammenhänge mit vertretbarem Aufwand untersucht werden können [SBH10]. Viele dieser Methoden zielen dabei darauf ab, einen vollfaktoriellen Versuchsplan so zu minimieren, dass in einem resultierenden teilstarktionalen Versuchsplan dennoch eine ausreichende aussagekräftige Teilmenge des Vollfaktorplans enthalten ist. Methoden zur Erstellung teilstarktionaler Versuchspläne anhand eines Vollfaktorplans werden im Allgemeinen auch als *Sampling*-Methoden bezeichnet. Siebertz, Beber und Hochkirchen stellen in [SBH10] verschiedene Methoden für das *Sampling* vor. Das Ziel ist dabei das möglichst raumfüllende *Sampling* aus einem Vollfaktorplan. Bekannte und häufig angewandte Methoden des *Samplings* sind dabei *Monte-Carlo*, *Latin-Hypercube* oder die Verwendung der *Halton*-Sequenz [SBH10].

Wie in dieser Arbeit bereits gezeigt wurde, ist die Durchführung eines Vollfaktorplans aus zeitlichen Gründen schwer möglich. Die Anwendung der Methoden des *Samplings* der statistischen Versuchsplanung ist aus einem methodischen Blickwinkel sinnvoll. Dennoch erfordert die korrekte Anwendung dieser Methoden einerseits einen gewissen Grad an Wissen bzgl. der Methoden und andererseits einen Aufwand zur Implementierung des *Samplings*. Im Rahmen der vorliegenden Arbeit wurde daher bewusst von der Verwendung einer der o.g. *Sampling*-Strategien abgesehen. Stattdessen wurde ein eigener Ansatz zur Erzeugung der teilstarktionalen Versuchspläne entwickelt, welcher sich an den Ideen und Konzepten der o.g. *Sampling*-Methoden orientiert und im Folgenden vorgestellt wird.

Das Ziel bei der Erstellung der teilstarktionalen Versuchspläne ist die Reduzierung des Versuchsaufwands auf ein zeitlich vertretbares Maß bei einer gleichzeitig noch aussagekräftigen Größe des Versuchsplans. Die Bestimmung der Größe einer Population, damit diese aussagekräftig ist, stellt ein

eigenes Gebiet der Statistik dar. Das zeitlich vertretbare Maß für die Durchführung eines Versuchsplanes hängt wiederum einerseits von externen Rahmenparametern wie Abgabedatum oder der zeitlichen Verfügbarkeit von Rechenkapazitäten und andererseits von Parametern wie der zugrundeliegenden Hardware des Systems ab. Für die Aussagekraft der Versuchspläne wurde eine Vereinfachung vorgenommen. So gilt ein Versuchsplan im Rahmen dieser Arbeit als aussagekräftig, wenn: a.) die Verteilung aller Faktoren jeweils in etwa einer Gleichverteilung entspricht und b.) jeder Faktor mit jeder Ausprägung mindestens einmal im Versuchsplan enthalten ist.

Durch eine explorative Analyse, bei der mehrere Versuchspläne erstellt und hinsichtlich der Aussagekraft sowie der Rechendauer untersucht wurden, wurde herausgefunden, dass die Reduzierung eines Vollfaktorplans auf 10 % einen guten Kompromiss aus Rechendauer und Aussagekraft darstellt. Die 10 % werden dabei zufällig aus einem Vollfaktorplan gezogen, sodass  $tVP_{M_2}^{II}$  entsprechend 10 % der Kombinationen aus  $vVP_{M_2}^{II}$  enthält. In Abbildung 6.2 sind die Histogramme der Faktoren des Versuchsplans  $tVP_{M_1}^I$  dargestellt.

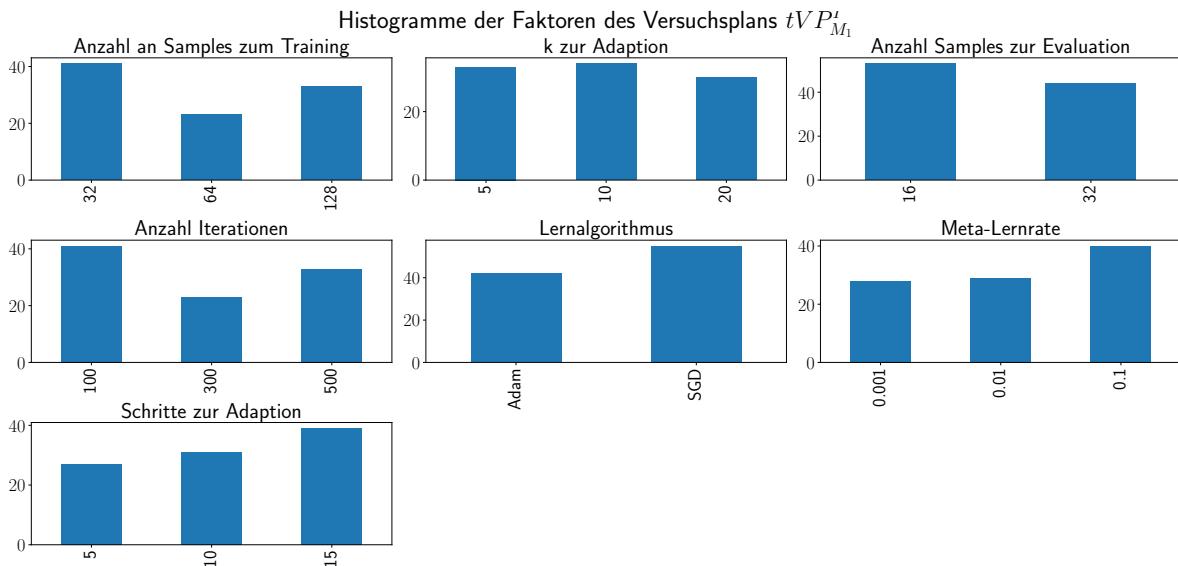


Abbildung 6.2.: Histogramme der Faktoren des Versuchsplans  $tVP_{M_1}^I$  (eigene Darstellung)

Der Abbildung 6.2 kann entnommen werden, dass beide Voraussetzungen zur Erfüllung der Aussagekraft erfüllt sind. Zum einen ist keine Werteausprägung eines Faktors stark über- oder unterrepräsentiert und zum anderen ist jede Werteausprägung mindestens einmal vorhanden. In Anhang A sind in den Abbildungen A.8 und A.9 jeweils die Histogramme der weiteren teilstellfaktoriellen Versuchspläne  $tVP_{M_1}^{II}$  bzw.  $tVP_{M_2}^{II}$  dargestellt. Durch die Reduzierung der  $tVP$  auf 10 % wurden die Versuchspläne, wie in Tabelle 6.3 dargestellt, reduziert.

## 6.4. Durchführung

In den folgenden Unterabschnitten wird die Durchführung des Experiments zur Beantwortung der Forschungsfragen und die Untersuchung der Hypothese dokumentiert. Hierfür werden die Ergebnisse der Durchführung der zuvor definierten teilstellfaktoriellen Versuchspläne  $tVP_{M_1}^I$ ,  $tVP_{M_1}^{II}$  und  $tVP_{M_2}^{II}$  beschrieben.

Versuchsplan	Größe	Zeit der Durchführung (h)
$tVP_{M_1}^I$	97	$\approx 24.25$
$tVP_{M_1}^{II}$	226	$\approx 7.20$
$tVP_{M_2}^{II}$	37	$\approx 1.23$

Tabelle 6.3.: GröÙe und geschätzte Zeit der Durchführung der drei unterschiedlichen teilstudiellen Versuchspläne

#### 6.4.1. Meta-Training

In Abbildung 6.3 sind die Histogramme der Kennzahlen der 97 Modelle des Versuchsplans  $tVP_{M_1}^I$  jeweils auf den Datensätzen  $X_{test}$  (6.3a) und  $X_{drifted,anormal}$  (6.3b) dargestellt.

In Tabelle 6.4 sind zusätzlich zur Abbildung 6.3 die statistischen Kennzahlen der Kennzahlen der 97 Modelle des Versuchsplans  $tVP_{M_1}^I$  jeweils auf  $X_{test}$  und  $X_{drifted,anormal}$  aus Phase I dargestellt. Der Tabelle kann entnommen werden, dass sich die Werte pro Kennzahl nicht stark unterscheiden, was insbesondere durch die sehr geringe Standardabweichung oder den Interquartilsabstand  $IQR$  durch Subtraktion der Minima und Maxima dargestellt wird.

Kennzahl:	$X_{test}$				$X_{drifted,anormal}$			
	Acc.	Prec.	Spec.	Sens.	Acc.	Prec.	Spec.	Sens.
Mittelwert:	99.70	100.00	93.16	95.73	90.25	41.96	92.43	95.62
Standardabweichung:	0.05	$3.42 \cdot 10^{-12}$	0.05	0.76	0.06	0.23	0.06	0.89
Min. Wert:	99.36	100.00	93.09	91.03	89.92	40.73	92.36	91.19
1. Quartil:	99.69	100.00	93.13	95.59	90.23	41.89	92.39	91.19
2. Quartil:	99.70	100.00	93.15	95.84	90.27	42.00	92.41	96.12
3. Quartil:	99.73	100.00	93.17	96.16	90.29	42.09	92.450	96.44
Max. Wert:	99.77	100.00	93.47	96.72	90.34	42.27	92.77	97.16

Tabelle 6.4.: Statistische Kennzahlen der Kennzahlen der 97 Modelle des Versuchsplans  $tVP_{M_1}^I$  jeweils auf  $X_{test}$  und  $X_{drifted,anormal}$

Im Rahmen der Durchführung des Versuchsplans  $tVP_{M_1}^I$  wurden 97 Modelle anhand des in Unterabschnitt 5.4.1 beschriebenen Vorgehens trainiert. Der vollständige teilstudielle Versuchsplan  $tVP_{M_1}^I$  inkl. der Accuracy für  $X_{test}$  und  $X_{drifted,anormal}$  pro Versuch ist in Anhang C in Tabelle C.3 dargestellt. Die gesamte Durchführung des Versuchsplans  $tVP_{M_1}^I$  benötigte 22 Stunden und 41 Minuten und entspricht damit in etwa der geschätzten Dauer der Durchführung aus Tabelle 6.3.

#### 6.4.2. Fine-Tuning

Im Rahmen der zweiten Phase wird das Fine-Tuning jeweils für die Modelle  $M_1$  und  $M_2$  aus der ersten Phase I nach den teilstudiellen Versuchsplänen  $tVP_{M_1}^{II}$  und  $tVP_{M_2}^{II}$  durchgeführt. Wie auch für die erste Phase I, werden zur Dokumentation der zweiten Phase II zunächst die Histogramme

beider Versuchspläne dargestellt. Anschließend werden die Kennzahlen der jeweiligen Durchführung beschrieben.

### Fine-Tuning $M_1$

Die Histogramme des teilstatistischen Versuchsplans  $tVP_{M_1}^{II}$  sind in Anhang A in Abbildung A.8 abgebildet. Der vollständige teilstatistische Versuchsplan des Fine-Tunings der Modelle  $M_1$  ist zusätzlich in Anhang C in Tabelle C.4 dargestellt. Während der Durchführung des Versuchsplans  $tVP_{M_1}^{II}$  wurden insgesamt 216 Fine-Tuning Adaptionen durchgeführt. Die durchschnittliche Zeit für eine Adaption betrug dabei 40 Sekunden, sodass die Gesamtdauer von in etwa zwei Stunden und 30 Minuten wesentlich geringer als die in Tabelle 6.3 geschätzten sieben Stunden und 12 Minuten ausfiel.

Im Rahmen der vorherigen Phase I wurden bei der Durchführung des Versuchsplans  $tVP_{M_1}^I$  insgesamt 97 unterschiedliche Modelle trainiert und evaluiert. Die Verwendung aller Modelle in der Phase II würde wiederum zu einem großen Versuchsplan  $tVP_{M_1}^{II}$  führen. Daher wurden für die Durchführung des Fine-Tunings in Phase II jeweils nur die drei Modelle mit der niedrigsten und die drei Modelle mit der höchsten *Precision*, also in Summe sechs Modelle, verwendet. Die Auswahl der jeweils besten und schlechtesten Modelle bzgl. der Erkennung von Anomalien folgt dabei der intuitiven Annahme, dass Modelle, welche eine schlechte Performance bzgl. der Identifizierung von Anomalien aufweisen, ein größeres Potential bei der Optimierung aufweisen. Die Auswahl der drei besten Modelle beruht dabei auf der gegenteiligen Annahme und soll einen weiteren Blickwinkel ermöglichen.

In Abbildung 6.4 sind die Histogramme Kennzahlen der Modelle  $M_2$  aus Phase II jeweils auf  $\mathbf{X}_{test}$  (6.4a) und  $\mathbf{X}_{drifted,anormal}$  (6.4b) dargestellt.

In Tabelle 6.5 sind zusätzlich zu den Histogrammen aus Abbildung 6.4 die statistischen Kennzahlen der Kennzahlen der Modelle aus Phase II dargestellt.

<b>Kennzahl:</b>	$\mathbf{X}_{test}$				$\mathbf{X}_{drifted,anormal}$			
	Acc.	Prec.	Spec.	Sens.	Acc.	Prec.	Spec.	Sens.
<i>Mittelwert:</i>	61.43	27.41	84.24	98.32	55.89	17.99	82.94	98.43
<i>Standardabweichung:</i>	25.81	26.76	13.58	1.74	23.23	10.43	13.90	1.65
<i>Min. Wert:</i>	7.12	7.12	0.00	91.30	7.12	7.12	0.00	91.39
<i>1. Quartil:</i>	44.16	11.25	83.95	97.65	40.28	10.63	82.46	97.80
<i>2. Quartil:</i>	61.90	15.63	88.63	98.76	56.48	13.93	87.55	98.76
<i>3. Quartil:</i>	83.03	29.38	91.55	99.52	75.54	22.35	90.73	99.56
<i>Max. Wert:</i>	99.79	100.00	93.45	100.00	90.37	42.36	92.76	100.00

Tabelle 6.5.: *Statistische Kennzahlen der Kennzahlen der 97 Modelle des Versuchsplans  $tVP_{M_1}^{II}$  jeweils auf  $\mathbf{X}_{test}$  und  $\mathbf{X}_{drifted,anormal}$*

### Fine-Tuning $M_2$

Im Rahmen der Durchführung des teilstatistischen Versuchsplans  $tVP_{M_2}^{II}$  wurden 97 Fine-Tuning-Adaptionen durchgeführt. Jedes aus einer Adaption resultierende Modell sowie das Ergebnis eines Durchlaufs wurde anschließend jeweils als Datei gesichert. Jede Adaption dauerte im Schnitt 90 Se-

kunden. Daraus ergibt sich eine Gesamtdauer der Durchführung von ca. 55 Minuten. Damit liegt auch die Dauer der Durchführung der Adaption des Modells  $M_2$  etwas niedriger, als die in Tabelle 6.3 geschätzte eine Stunde und 14 Minuten. Die aus dem Versuchsplan  $tVP_{M_2}^{II}$  resultierenden Modelle können anschließend verwendet werden, um die Modelle  $M_1$  und  $M_2$  hinsichtlich ihrer Adaptierbarkeit für neue Aufgaben hin zu untersuchen. In Abbildung 6.5 sind die Histogramme der Kennzahlen der aus  $tVP_{M_2}^{II}$  resultierenden 37 Modelle dargestellt. In Tabelle 6.6 sind zusätzlich die statistischen Kennzahlen als Ergänzung zu Abbildung 6.5 aufgelistet.

<b>Kennzahl:</b>	$\mathbf{X}_{test}$				$\mathbf{X}_{drifted,anormal}$			
	Acc.	Prec.	Spec.	Sens.	Acc.	Prec.	Spec.	Sens.
<i>Mittelwert:</i>	47.42	30.05	77.93	98.94	43.33	16.88	76.37	98.96
<i>Standardabweichung:</i>	30.91	37.20	15.04	1.51	27.73	13.15	15.26	1.43
<i>Min. Wert:</i>	9.98	7.33	28.72	95.15	9.87	7.33	27.81	95.39
<i>1. Quartil:</i>	29.82	9.19	76.21	99.52	27.14	8.89	73.82	99.19
<i>2. Quartil:</i>	35.95	9.98	80.21	99.68	33.72	9.68	78.93	99.72
<i>3. Quartil:</i>	50.74	12.59	86.02	99.84	46.05	11.63	84.60	99.72
<i>Max. Wert:</i>	99.81	100.00	93.20	99.96	90.37	42.37	92.47	100.00

Tabelle 6.6.: *Statistische Kennzahlen der Kennzahlen der 37 Modelle des Versuchsplans  $tVP_{M_2}^{II}$  jeweils auf  $\mathbf{X}_{test}$  und  $\mathbf{X}_{drifted,anormal}$*

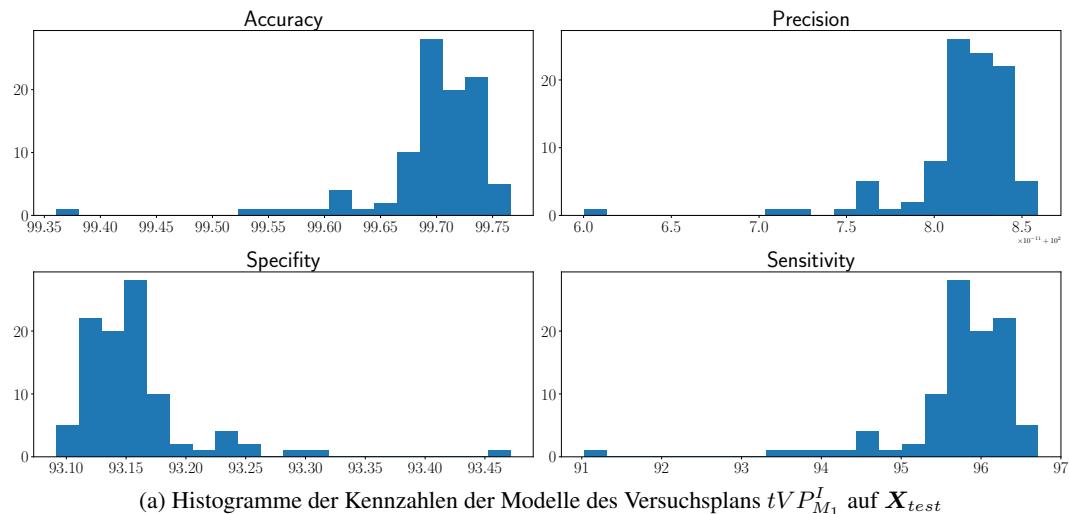
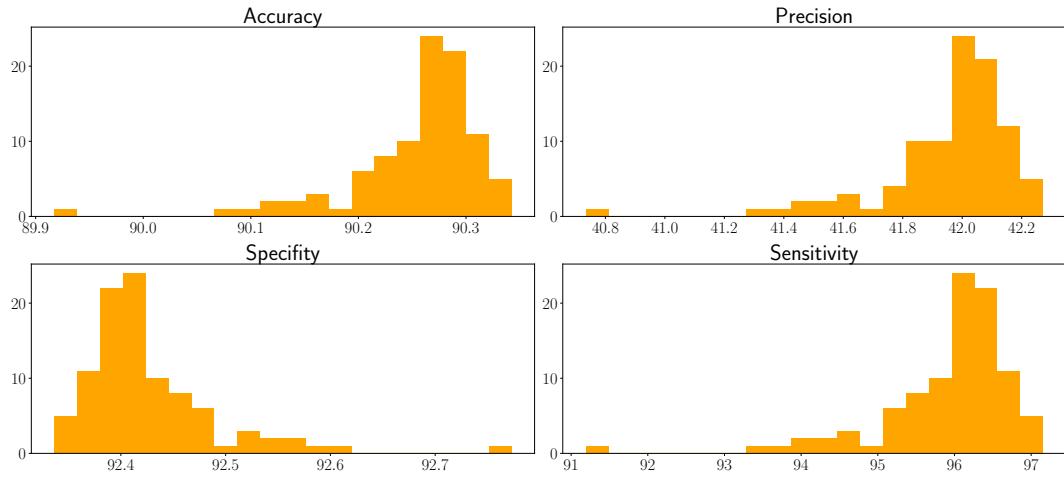
(a) Histogramme der Kennzahlen der Modelle des Versuchsplans  $tVP_{M_1}^I$  auf  $\mathbf{X}_{test}$ (b) Histogramme der Kennzahlen der Modelle des Versuchsplans  $tVP_{M_1}^I$  auf  $\mathbf{X}_{drifted,anormal}$ 

Abbildung 6.3.: *Histogramme der Kennzahlen Accuracy, Precision, Specificity und Sensitivity der Modelle des Versuchsplans  $tVP_{M_1}^I$  auf den Datensätzen  $\mathbf{X}_{test}$  und  $\mathbf{X}_{drifted,anormal}$*

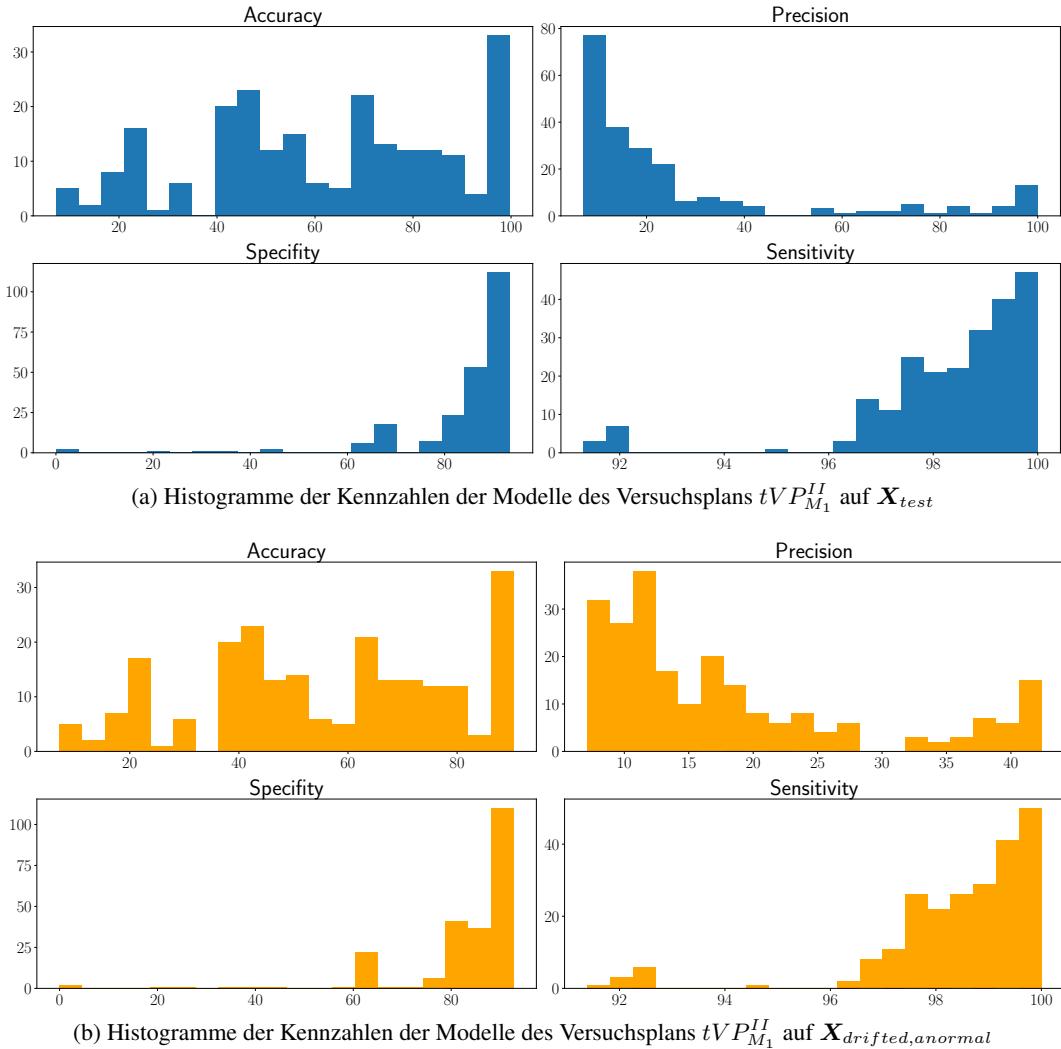


Abbildung 6.4.: *Histogramme der Kennzahlen Accuracy, Precision, Specificity und Sensitivity der Modelle des Versuchsplans  $tVP_{M_1}^{II}$  auf den Datensätzen  $X_{test}$  und  $X_{drifted,anormal}$*

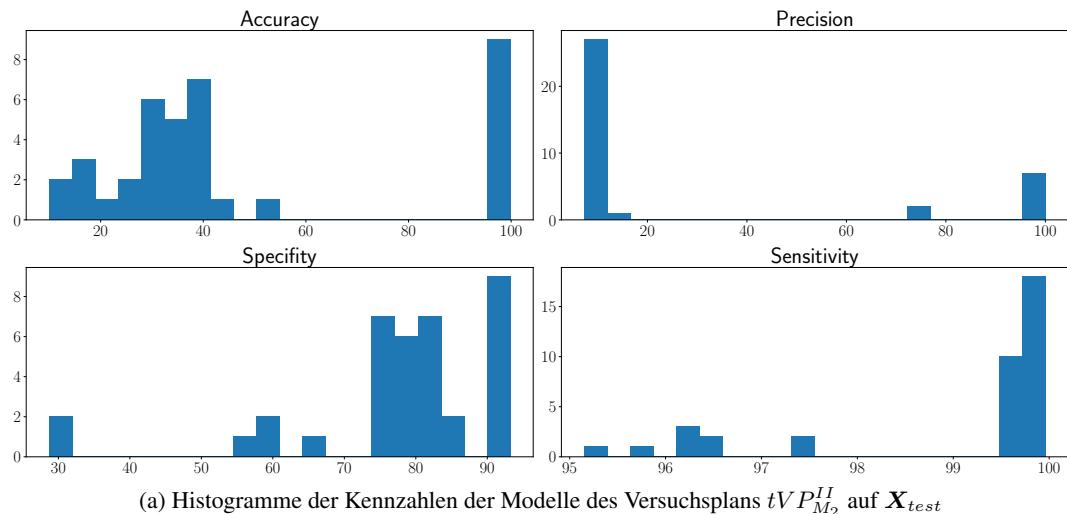
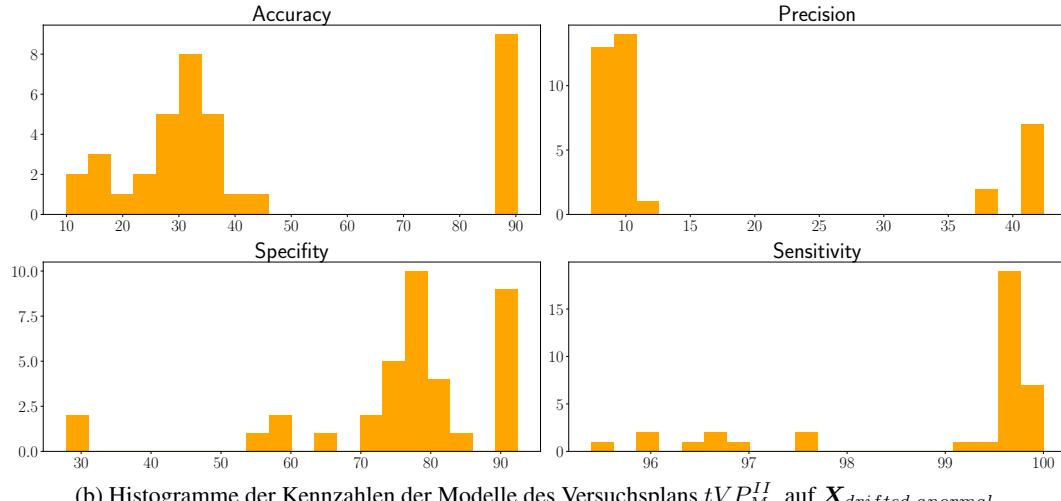
(a) Histogramme der Kennzahlen der Modelle des Versuchsplans  $tVP_{M_2}^{II}$  auf  $X_{test}$ (b) Histogramme der Kennzahlen der Modelle des Versuchsplans  $tVP_{M_2}^{II}$  auf  $X_{drifted,anormal}$ 

Abbildung 6.5.: *Histogramme der Kennzahlen Accuracy, Precision, Specificity und Sensitivity der Modelle des Versuchsplans  $tVP_{M_2}^{II}$  auf den Datensätzen  $X_{test}$  und  $X_{drifted,anormal}$*

## 6.5. Auswertung

In den folgenden Unterabschnitten sollen die zentralen Erkenntnisse aus der Durchführung des Experiments ausgewertet werden. Hierfür wird in Unterabschnitt 6.5.1 zunächst der durch die Durchführung des Experiments gesammelte Datenbestand aufgezählt. Anschließend wird in Unterabschnitt 6.5.2 ein Vergleich der trainierten Modelle aus den Phasen I und II anhand definierter Fragestellungen vorgenommen.

### 6.5.1. Datenbestand

Das Ergebnis der Durchführung des Experiments (in Form der Ausführung der drei teilstatischen Versuchspläne  $tVP_{M_1}^I$ ,  $tVP_{M_2}^I$  und  $tVP_{M_2}^{II}$ ) sind drei Listen mit den Kennzahlen *Accuracy*, *Precision*, *Specificity* und *Sensitivity* sowie den Werten der *Confusion Matrix* pro Versuch eines Versuchsplans jeweils für den Datensatz  $X_{test}$  und  $X_{drifted,anormal}$ . Die Anzahl der Einträge pro Liste entspricht dabei der Anzahl der Versuche pro Versuchsplan, also 97 für  $tVP_{M_1}^I$ , 226 für  $tVP_{M_1}^{II}$  und 37 für  $tVP_{M_2}^{II}$ .

### 6.5.2. Modellvergleiche

Grundsätzlich lassen sich anhand der Daten, welche im Rahmen der Durchführung des Experiments erhoben wurden, eine Vielzahl spannender Fragestellungen untersuchen und beantworten. Das Ziel dieser Arbeit ist jedoch die Beantwortung fest definierter Forschungsfragen sowie die Untersuchung einer zuvor spezifizierten Hypothese. Die Definition verschiedener Fragestellungen und Beantwortung dieser anhand der erhobenen Daten soll deshalb hauptsächlich auf die Beantwortung der Forschungsfragen **RQ1** und **RQ2** und die Untersuchung der Hypothese **H1** einzahlen. Alle der im Folgenden bearbeiteten Fragestellungen **Q1** bis **Q5** wurden daher so gewählt, dass sie einen umfangreichen Blickwinkel zur Untersuchung ermöglichen.

#### **Q1:** Wurde die Modellqualität der Modelle $M_1$ auf $X_{drifted,anormal}$ durch das Fine-Tuning verbessert?

Zur Beantwortung dieser Frage werden insbesondere die Kennzahlen der *Accuracy* (Gesamterkennungsrate eines Modells) und *Precision* (Erkennungsrate der Anomalien) auf  $X_{drifted,anormal}$  betrachtet. Dabei werden die Kennzahlen jeweils vor und nach dem Fine-Tuning untersucht. Anhand der Abweichung lässt sich so ein Eindruck der Veränderung erzeugen. In Abbildung 6.6 sind dafür die Boxplots zur Darstellung der Kennzahl *Accuracy* der Modelle  $M_1$  jeweils vor dem Fine-Tuning ( $tVP_{M_1}^I$ ) und danach ( $tVP_{M_1}^{II}$ ) für den Datensatz  $X_{drifted,anormal}$  dargestellt. Während die *Accuracy* der Modelle vor dem Fine-Tuning im Mittel bei 90.25 % mit einer Standardabweichung von 0.06 lag, liegt sie nach der Durchführung des Fine-Tunings im Mittel bei 55.89 % mit einer Standardabweichung von 23.23. Die geringste *Accuracy* vor dem Fine-Tuning betrug dabei 89.92 % und die höchste 90.34 %. Die niedrigste *Accuracy* nach dem Fine-Tuning beträgt nun 7.12 %, die höchste 90.37 %.

Während der Durchführung des Versuchsplans  $tVP_{M_1}^{II}$  wurden jeweils die drei Modelle mit der niedrigsten *Precision* ( $f_1, f_2$  und  $f_3$ ) und die drei Modelle mit der höchsten *Precision* ( $f_4, f_5$  und  $f_6$ ) aus der Durchführung der Phase II verwendet. In Tabelle 6.7 sind die Kennzahlen der *Accuracy* und

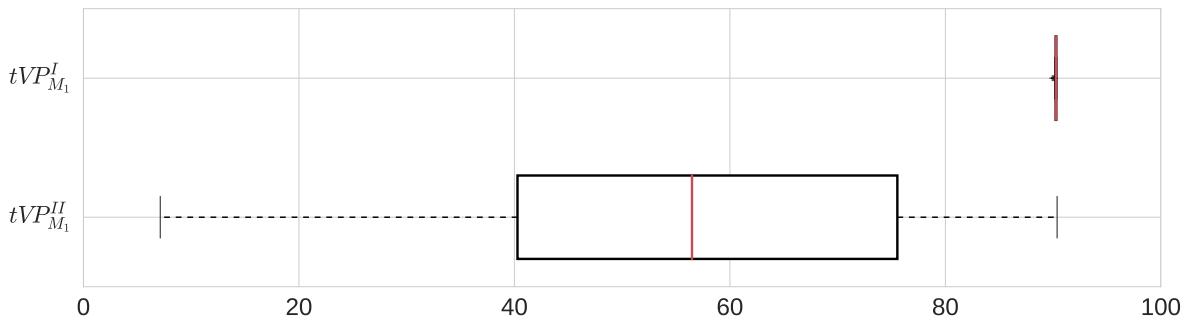


Abbildung 6.6.: Boxplot der Accuracy der Modelle  $M_1$  der Versuchspläne  $tVP_{M_1}^I$  und  $tVP_{M_1}^{II}$  auf  $X_{drifted,anormal}$  (eigene Darstellung)

*Precision* pro Modell auf  $X_{drifted,anormal}$  jeweils vor und nach dem Fine-Tuning inkl. der prozentualen Abweichung dargestellt. Die Kennzahlen aus  $tVP_{M_1}^I$  sind dabei die Kennzahlen der jeweiligen Modelle nach Phase I und die Kennzahlen aus  $tVP_{M_1}^{II}$  sind jeweils der Mittelwert aus allen Versuchsläufen pro Modell.

<b>Modell</b>	$tVP_{M_1}^I$		$tVP_{M_1}^{II}$		Abweichung in %	
	Accuracy (%)	Precision (%)	Accuracy (%)	Precision (%)	Accuracy	Precision
$f_1:$	89.92	40.73	68.46	22.71	-23, 87	-44, 24
$f_2:$	90.07	41.29	57.95	16.36	-35, 66	-60, 38
$f_3:$	90.11	41.43	48.17	13.83	-46, 54	-66, 62
$f_4:$	90.33	42.23	43.07	14.67	-52, 32	-65, 26
$f_5:$	90.33	42.24	53.57	18.51	-40, 70	-56, 18
$f_6:$	90.34	42.27	66.72	22.67	-26, 14	-46, 37

Tabelle 6.7.: Accuracy und Precision pro Modell auf  $X_{drifted,anormal}$  pro Versuchsplan  $tVP_{M_1}^I$  und  $tVP_{M_1}^{II}$

**A1:** Wie Abbildung 6.6 und Tabelle 6.7 zu entnehmen ist, konnte die Modellqualität für keines der 97 Modelle aus Phase II im Rahmen des Fine-Tunings verbessert werden. Vielmehr wurde die Modellqualität stark minimiert. Wie in Tabelle 6.7 beispielhaft für die sechs zugrundeliegenden Modelle in Phase II dargestellt ist, wurde die *Accuracy* im besten Fall um 23.87 % und im schlechtesten Fall um 52.32 % minimiert, was einer durchschnittlichen Verschlechterung um 37.54 % entspricht. Die *Precision* wurde im Schnitt um 56.51 % verschlechtert. Die *Accuracy* für alle Modelle, ohne Gruppierung nach zugrundeliegendem Modell, wurde von Durchschnittlich 90.25 % auf 55.89 % minimiert, was einer Verschlechterung um 38.07 % entspricht. Auch die *Specificity* und *Sensitivity* wurden durch das Fine-Tuning minimiert.

**Q2:** Konnte der *Concept Drift* durch das Fine-Tuning minimiert werden?

Der *Concept Drift* wurde im Kontext der Anomalieerkennung in der vorliegenden Arbeit v.a. als die Veränderung der Anzahl der FP auf einem Datensatz gemessen. In Abbildung 6.7 sind für die Untersuchung der Veränderung des *Concept Drifts* die beiden *Confusion Matrices* der Modelle  $M_1$  nach den beiden Phasen I und II auf dem Datensatz  $X_{drifted,anormal}$  dargestellt. Die Werte der *Confusion*

Matrices ergeben sich dabei jeweils aus den Mittelwerten aller Confusion Matrices der  $M_1$  Modelle.

		Confusion Matrix $M_1$ nach Phase I				Confusion Matrix $M_1$ nach Phase II			
		Tatsächlich		Prognostiziert		Tatsächlich		Prognostiziert	
Tatsächlich	Anomalie	k. Anomalie		Anomalie		Anomalie	k. Anomalie	Anomalie	
		29230	3313	102	2395			17128	15415
		k. Anomalie	Anomalie	Anomalie	Prognostiziert		k. Anomalie	Anomalie	Prognostiziert

Abbildung 6.7.: Confusion Matrices der Modelle  $M_1$  nach Phase I und Phase II auf  $\mathbf{X}_{drifted,anormal}$  (eigene Darstellung)

Der Abbildung 6.7 kann entnommen werden, dass die Anzahl an FP nach Phase I von 3313 auf durchschnittlich 15415 gestiegen ist. Dies entspricht einem Zuwachs um 365.29 %. Die Anzahl der FN konnte hingegen von 102 nach Phase I auf 39 nach Phase II minimiert werden (-61.77 %). Die Anzahl der TN ist jedoch von 29230 um 12102 Samples auf 17128 gesunken (-41.40 %). Die Anzahl der korrekterweise als Anomalie identifizierten Samples (TP) ist von 2395 um 63 Samples auf 2458 gestiegen (+2.64 %).

**A2:** Wie die bereits gezeigten durchschnittlichen Veränderungen in den Confusion Matrices andeuten, konnten die Auswirkungen des *Concept Drifts* auf die Modelle  $M_1$  durch das Fine-Tuning in Phase II nicht minimiert werden. Die Fähigkeit der Modelle  $M_1$  ein Sample, welches keine Anomalie darstellt, auch als solches zu erkennen, hat sich durch das Fine-Tuning stark verschlechtert.

**Q3:** Wie hat sich für  $M_1$  die Modellqualität pro Anomalie-Art verändert?

In Tabelle 6.8 ist die Accuracy der Modelle  $M_1$  der Versuchspläne  $tVP_{M_1}^I$  (nach Phase I) und  $tVP_{M_1}^{II}$  (nach Phase II) auf  $\mathbf{X}_{drifted,anormal}$  pro Anomalie Art dargestellt.

Anomalie Art:	$tVP_{M_1}^I$				$tVP_{M_1}^{II}$			
	0	1	2	3	0	1	2	3
Mittelwert:	89.82	15.72	97.81	27.74	52.63	57.21	99.15	76.64
Standardabweichung:	0.00	1.77	0.90	0.92	25.11	24.13	1.29	19.31
Min. Wert:	89.82	15.00	92.97	27.27	0.00	15.00	93.18	27.27
Max. Wert:	89.82	20.00	99.01	29.55	89.82	100.00	100.00	100.00

Tabelle 6.8.: Accuracy der Modelle  $M_1$  der Versuchspläne  $tVP_{M_1}^I$  und  $tVP_{M_1}^{II}$  auf  $\mathbf{X}_{drifted,anormal}$  pro Anomalie Art

Die Tabelle 6.8 stellt eine interessante Beobachtung der *Accuracy* im Vergleich zur allgemeinen Veränderung der Modellqualität aus Fragestellung **Q1** dar. Während die Modellperformance insgesamt durch das Fine-Tuning deutlich verschlechtert wurde, ist die Erkennungsrate für die Anomalie Arten 1 (Punktanomalie), 2 (kollektive Anomalie) und 3 (Rauschen) deutlich verbessert worden. Nach Durchführung des Versuchsplans  $tVP_{M_1}^I$  konnten die Modelle  $M_1$  beispielsweise Punktanomalien im Mittel nur zu 15.72 % erkennen. Nach dem Fine-Tuning beträgt die Erkennungsrate 57.21 %, was einer Verbesserung um 263.93 % entspricht. Die Klassifikation von *Samples*, welche keine Anomalie darstellen (Anomalie Art 0), konnten die Modelle  $M_1$  vor dem Fine-Tuning zu durchschnittlich 89.82 % richtig klassifizieren. Nach dem Fine-Tuning beträgt die Genauigkeit nur noch 52.63 %. Hierdurch lässt sich auch die negative Veränderung der gesamten Modellqualität erklären. In dem Datensatz  $\mathbf{X}_{drifted,anormal}$  sind insgesamt nur 2497 *Samples* mit Anomalien, aber 32543 *Samples* ohne Anomalie enthalten. Die Verschlechterung der Erkennungsrate von *Samples* ohne Anomalie hat daher einen wesentlich größeren Einfluss auf die Veränderung der Modellqualität, als die Verbesserung der Erkennungsrate der 2497 *Samples* mit Anomalie. Diese Feststellung lässt sich ebenfalls in der Veränderung der TN, FP, TP und FN in den *Confusion Matrices* in Abbildung 6.7 beobachten.

**A3:** Die Erkennungsrate der unterschiedlichen Anomalie Arten hat sich durch das Fine-Tuning deutlich verbessert. Da die Anzahl der *Samples* mit Anomalie mit 7.13 % in den Daten nur einen sehr geringen Anteil aufweisen, hat die Verbesserung der Erkennung der Anomalien jedoch keinen Einfluss auf die gesamte Modellperformance, da die Anzahl der FP durch das Fine-Tuning deutlich gestiegen ist.

**Q4:** Wie hat sich für  $M_1$  die Modellqualität pro *Concept Drift Event Art* verändert?

In Tabelle 6.9 sind die Kennzahlen der Modelle  $M_1$  pro *Concept Drift Event Art* jeweils nach Durchführung des Versuchsplans  $tVP_{M_1}^I$  und  $tVP_{M_1}^{II}$  auf dem Datensatz  $\mathbf{X}_{drifted,anormal}$  dargestellt. Insgesamt wird zwischen den drei unterschiedlichen *Concept Drift Event Arten* 1: öffnen oder schließen eines Schalters, 2: Veränderung eines Lastprofils und 3: Veränderung des Phasenwinkels unterschieden. Wenn ein *Sample* nicht durch ein Event manipuliert wurde, ist dieses mit 0 gekennzeichnet.

Event Art	Kennzahlen	$tVP_{M_1}^I$				$tVP_{M_1}^{II}$			
		Acc.	Prec.	Spec.	Sen.	Acc.	Prec.	Spec.	Sen.
0	Mittelwert:	98.03	100.00	99.62	15.90	55.82	11.43	94.07	66.75
	Std. $\sigma$ :	0.16	0.00	0.16	6.97	25.90	24.22	11.71	32.39
1	Mittelwert:	0.35	0.35	0.00	100.00	0.35	0.35	0.00	100.00
	Std. $\sigma$ :	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	Mittelwert:	98.81	91.55	88.21	99.06	62.60	33.92	76.51	99.56
	Std. $\sigma$ :	0.05	0.03	0.04	0.40	24.18	23.68	14.85	0.47
3	Mittelwert:	99.53	100.00	97.67	83.27	60.10	17.84	92.55	94.06
	Std. $\sigma$ :	0.12	0.00	0.12	4.46	27.44	26.55	11.11	7.30

Tabelle 6.9.: Kennzahlen der Modelle  $M_1$  für die Versuchspläne  $tVP_{M_1}^I$  und  $tVP_{M_1}^{II}$  auf  $\mathbf{X}_{drifted,anormal}$  pro *Concept Drift Event Art*

Wie der Tabelle 6.9 zu entnehmen ist, hat sich die *Accuracy*, wie auch die *Precision*, für alle Arten der *Concept Drift* Events deutlich verschlechtert. Die *Specificity* hat sich hingegen weniger stark verringert. In Tabelle 6.10 sind zusätzlich die prozentualen Veränderungen der Kennzahlen zwischen  $tVP_{M_1}^I$  und  $tVP_{M_1}^{II}$  auf  $X_{drifted,anormal}$  pro *Concept Drift* Event Art dargestellt. Der Tabelle kann entnommen werden, dass sich die *Accuracy* der Modelle  $M_1$  für alle Event Arten in etwa gleich stark verringert hat. Die *Sensitivity* hat sich als einzige Kennzahl hingegen deutlich erhöht. Besonders auffällig ist dabei die starke Veränderung für die *Samples* ohne *Concept Drift* Event, welche sich durch die deutlich gesteigerte Anzahl an TN (siehe Abbildung 6.7) erklären lässt.

<i>Concept Drift</i> Event-Art	<i>Accuracy</i>	<i>Precision</i>	<i>Specificity</i>	<i>Sensitivity</i>
0:	-43.06	-88.57	-5.57	+319.81
1:	0.00	0.00	0.00	0.00
2:	-36.65	-62.95	-13.26	+0.50
3:	-39.61	-82.16	-5.24	+12.96
<b>Mittelwert:</b>	<b>-29.83</b>	<b>-58.42</b>	<b>-6.02</b>	<b>+83.32</b>

Tabelle 6.10.: Prozentuale Veränderung der Kennzahlen der Modelle  $M_1$  zwischen den Versuchsplänen  $tVP_{M_1}^I$  und  $tVP_{M_1}^{II}$  auf  $X_{drifted,anormal}$  pro *Concept Drift* Event Art

**A4:** Wie in den Antworten der Fragen **Q1** bis **Q3** bereits gezeigt wurde, hat sich die Modellperformance insgesamt deutlich verschlechtert. Dies lässt sich auch bei der Betrachtung der Kennzahlen pro *Concept Drift* Event Art beobachten. Die *Sensitivity* ist dabei die einzige Kennzahl, welche durch das Fine-Tuning deutlich (im Mittel um +83.32 %) verbessert werden konnte. Dies lässt sich durch die nach Phase II deutlich gesteigerte Anzahl an TN erklären, welche so zu einer hohen *Sensitivity* führt.

**Q5:** Gibt es bei der Veränderung der Modellqualität einen Unterschied zwischen den Modellen  $M_1$  und  $M_2$ ?

Im Rahmen der Betrachtung der vorherigen Fragestellungen **Q1** bis **Q4** wurde der Fokus insbesondere auf die Modellqualität der Modelle  $M_1$  gelegt. Im Folgenden soll daher abschließend noch ein Vergleich der Modelle  $M_1$  und  $M_2$  jeweils für beide Phasen des Experiments vorgenommen werden. In Tabelle 6.11 sind daher die Kennzahlen der Modelle  $M_1$  und  $M_2$  auf  $X_{drifted,anormal}$  jeweils nach Phase I und Phase II dargestellt. Als Kennzahlen der Modelle  $M_1$  wurde dabei jeweils der Mittelwert aller Modelle verwendet. Der Tabelle kann entnommen werden, dass beide Modelle  $M_1$  und  $M_2$  nach dem initialen Training in Phase I eine sehr ähnliche Performance aufweisen. Die Veränderung der Kennzahlen zwischen beiden Phasen betrifft beide Modelle ebenfalls gleich stark.

**A5:** Die Beantwortung der Fragestellung **Q5** lässt sich anhand der Kennzahlen aus Tabelle 6.11 relativ simpel beantworten. So weisen beide Modelle sowohl nach dem ersten Training in Phase I als auch nach dem Fine-Tuning in Phase II eine sehr ähnliche Performance auf, wobei die Modelle  $M_1$  im Mittel eine minimal bessere Performance aufweisen. Dabei ist jedoch zu berücksichtigen, dass im Rahmen der Phase II insgesamt 226 Modelle  $M_1$  und nur 37 Modelle  $M_2$  trainiert wurden. Es kann daher nicht ausgeschlossen werden, dass die bessere Performance von  $M_1$  nach dem Fine-Tuning

nicht einfach durch die Minimierung der Varianz durch eine größere Anzahl an Modellen verursacht wird.

<b>Kennzahlen</b>	<b>Phase I</b>		<b>Phase II</b>	
	$M_1$	$M_2$	$M_1$	$M_2$
<i>Accuracy:</i>	90.25	90.28	55.89	27.73
<i>Precision:</i>	41.96	42.05	17.99	16.88
<i>Specificity:</i>	92.43	92.40	82.94	76.37
<i>Sensitivity:</i>	95.62	96.28	98.43	98.96

Tabelle 6.11.: *Kennzahlen der Modelle  $M_1$  und  $M_2$  auf  $X_{drifted,anormal}$  jeweils nach Phase I und Phase II*

## 6.6. Evaluation

In den folgenden Unterabschnitten wird die Evaluation der Erkenntnisse der Durchführung des Experiments durchgeführt. Hierzu werden in den Unterabschnitten 6.6.1 und 6.6.2 zunächst die Annahmen **A1** und **A2** daraufhin untersucht, ob sie im Rahmen der vorliegenden Arbeit tatsächlich zutreffen. Anschließend werden in den Unterabschnitten 6.6.3 und 6.6.4 die Thesen **T1** und **T2** untersucht. Abschließend wird in Unterabschnitt 6.6.5 die zentrale Hypothese **H1** evaluiert.

### 6.6.1. Überprüfung der Annahme A1

Die in Unterabschnitt 3.1.2 definierte Annahme **A1** konnte im Rahmen des Experiments bestätigt werden. So haben die mittels *Meta-Learning*-Algorithmen trainierten Modelle  $M_1$  auf dem Datensatz  $X_{test}$  eine *Precision* von durchschnittlich 99.99 %. Auch die Modelle  $M_2$ , welche anhand klassischer Lernalgorithmen trainiert wurden, haben auf  $X_{test}$  mit ebenfalls 99.99 % eine hohe *Precision*. Daher kann angenommen werden, dass AE für den Zweck der Anomalieerkennung im Kontext der vorliegenden Arbeit geeignet sind.

### 6.6.2. Überprüfung der Annahme A2

Die Annahme **A2** konnte im Rahmen des Experiments ebenfalls bestätigt werden. Die Modelle  $M_1$  und  $M_2$  weisen auf  $X_{test}$  jeweils eine *Precision* von 99.99 % auf. Auf dem Datensatz  $X_{drifted,anormal}$  weisen die Modelle  $M_1$  mit durchschnittlich 41.96 % hingegen eine deutlich geringere *Precision* auf. Die *Precision* der Modelle  $M_2$  auf  $X_{drifted,anormal}$  ist mit 42.05 % ebenfalls deutlich geringer als auf  $X_{test}$ . Die Minimierung der *Precision* lässt sich insbesondere durch die gestiegene Anzahl an FP erklären, was wiederum ein Indiz für die Präsenz eines *Concept Drifts* darstellt. Es kann also angenommen werden, dass der durch die Simulation erzeugte *Concept Drift* im Kontext dieser Arbeit negative Auswirkungen auf die Qualität der Modelle  $M_1$  und  $M_2$  hat.

### 6.6.3. Überprüfung der These T1

Die These **T1** kann im Rahmen der vorliegenden Arbeit nicht bestärkt werden, da der Versuch der Minimierung der Auswirkungen des *Concept Drifts* auf die Modelle im Rahmen des Experiments nicht gelungen ist. So konnte die *Sensitivity* der Modelle  $M_1$  durch das Fine-Tuning in Phase II für unterschiedliche *Concept Drift* Event-Arten zwar deutlich verbessert werden. Dieser Effekt lässt sich jedoch durch die deutlich gestiegene Anzahl an TP und FP erklären, da die Modelle  $M_1$  ein *Sample* nach dem Fine-Tuning eher als Anomalie klassifizieren. Die Falsifizierung der These im Rahmen der vorliegenden Arbeit ist jedoch nicht möglich. Die Gründe hierfür werden in Unterabschnitt 6.6.5 detailliert diskutiert.

### 6.6.4. Überprüfung der These T2

Die Erzeugung eines *Concept Drifts* sowie die Messung dieses im Rahmen des Experiments war im Kontext dieser Arbeit erfolgreich. Wie in Unterabschnitt 6.6.2 bereits gezeigt wurde, hat die Manipulation der Daten mittels verschiedener Events einen negativen Effekt auf die Modellperformance in Form einer Erhöhung der FP-Rate. Die Messung dieses Effekts ist über die Berechnung der Kennzahlen der *Confusion Matrix* sowie der FP, TP, FN, FP Raten ebenfalls möglich. So kann die These **T2** im Rahmen der vorliegenden Arbeit nicht falsifiziert werden.

### 6.6.5. Überprüfung der Hypothese H1

Die in Unterabschnitt 3.1.4 definierte Hypothese **H1** stellt den zentralen Untersuchungsgegenstand der vorliegenden Arbeit dar. Die Hypothese wurde dabei in Form einer Implikation durch Formulierung als „Wenn-Dann“ Satz aufgestellt, wobei (entsprechend der Definition einer Implikation) der linke Term (Wenn-Teil) die Voraussetzung und der rechte Term (Dann-Teil) die Behauptung darstellt. Die Voraussetzung ist dabei äquivalent zu der These **T1**, welche in Unterabschnitt 6.6.3 bereits kurz untersucht wurde.

Im Folgenden soll die Hypothese zunächst aus einer rein logischen Perspektive analysiert werden. Bei einer rein auf Fakten basierenden Betrachtung der Voraussetzung  $A$  folgt, dass diese *falsch* ist, da das *Meta-Learning* keinen positiven Effekt auf die Performance des Modells  $M_1$  hat. Die Behauptung  $B$  ist ebenfalls *falsch*, da der Fehler  $\epsilon_{M_1}$  des Modells  $M_1$  nach dem Fine-Tuning nicht kleiner ist, als der Fehler  $\epsilon_{M_2}$  des Modells  $M_2$ , welches nicht adaptiert wurde. Daraus folgt  $falsch \Rightarrow falsch$ . Entsprechend der logischen Funktion einer Implikation folgt daraus, dass die Hypothese *wahr* sein muss.

Die Untersuchung der Hypothese 1 kann im Kontext der vorliegenden Arbeit dennoch nicht abschließend vorgenommen werden. So wurde in Unterabschnitt 6.6.3 bereits gezeigt, dass die Voraussetzung von **H1** nicht bewertet werden kann, wodurch eine Beurteilung der Hypothese nicht mehr möglich ist. Im Folgenden soll daher weiter analysiert werden, weshalb die Beurteilung der Voraussetzung der Hypothese 1 nicht möglich ist.

Die valide Untersuchung der Ursachen für das Scheitern der Beurteilung der These 1 bzw. der Voraussetzung  $A$  bedingt eine tiefgehende Analyse der Modelle bzgl. ihrer Adoptionsfähigkeit an neue Aufgaben. Die Untersuchung dieser Adoptionsfähigkeit stellt jedoch ein komplexes Gebiet des *Meta-Learnings* dar, welches im Rahmen dieser Arbeit aus zeitlichen Gründen leider nicht möglich ist.

Über die Ursachen können daher im Folgenden nur Vermutungen angestellt werden, welche, sofern möglich, mit Fakten unterlegt werden.

Die Fähigkeit eines Modells, schnell an neue Aufgaben adaptiert werden zu können, wird im Rahmen des *Meta-Trainings* dadurch erreicht, dass ein Modell möglichst sensitiv gegenüber unterschiedlichen Aufgaben trainiert wird. Daher wird während des *Meta-Trainings* zwischen dem *Outer-Loop*- und *Inner-Loop*-Learning unterschieden, wobei die Modellparameter jeweils auf unterschiedlichen Aufgaben  $\mathcal{T}$  evaluiert und adaptiert werden. Im Rahmen der Implementierung des *Meta-Trainings*, beschrieben in Unterabschnitt 5.4.1, konnten für das Training jedoch keine unterschiedlichen Aufgaben definiert werden. Daher könnte vermutet werden, dass das Modell während des *Meta-Trainings* die Sensitivität für unterschiedliche Aufgaben nicht lernen konnte. Diese Vermutung wird durch die Kennzahlen der Modelle  $M_1$  und  $M_2$  nach der Phase II gestützt. So haben sich die Kennzahlen beider Modelle durch das Fine-Tuning in Phase II in gleicher Art und Weise verschlechtert, was vermuten lassen könnte, dass sich beide Modelle hinsichtlich der Eigenschaften ihrer Gewichte ähnlich sind und das *Meta-Training* der Modelle  $M_1$  keinen Einfluss auf die Adoptionsfähigkeit der Modelle hatte.

In Unterabschnitt 5.3.1 wurde die Idee der Ersetzung eines Schwellwerts durch eine log. Regression beschrieben. Dabei wurde eine log. Regression anhand der Werte der *Reconstruction Error* eines AE und der tatsächlichen *Label* des Datensatzes  $\mathbf{X}_{test}$  trainiert, um binäre Anomalie-*Label* vorherzusagen. Durch Verwendung dieses zusätzlichen Modells konnte die Prognosequalität der AE deutlich verbessert werden. Dabei wurde angenommen, dass die Verwendung der log. Regression als binärer Klassifikator der vorherigen Intuition des Lernen eines Schwellwerts entspricht. Bei der Evaluation der Modelle nach dem Fine-Tuning auf  $\mathbf{X}_{drifted,anormal}$  wurden jedoch weiterhin die auf den Daten  $\mathbf{X}_{test}$  trainierten log. Regressionen verwendet. Für eine Analyse der Rolle der log. Regression wurde daher für zehn zufällig gewählte Modelle aus  $M_2$ , welche im Rahmen der Phase II mittels Fine-Tuning adaptiert wurden, jeweils eine neue log. Regression trainiert. Für das Training der log. Regression wurde zunächst der *Reconstruction Error* der adaptierten Modelle auf  $\mathbf{X}_{test}$  berechnet. Anschließend wurden die log. Regressionen erneut trainiert. Bei der Evaluation der zehn Modelle konnte anschließend festgestellt werden, dass die Modellperformance jeweils deutlich verbessert wurde. So betrug die *Accuracy* des schlechtesten Modells der zehn Modelle auf  $\mathbf{X}_{drifted,anormal}$  nach dem Fine-Tuning 7.12 %. Durch das Training einer neuen logistischen Regression konnte die *Accuracy* auf 89.72 % erhöht werden, sodass die Genauigkeit des Modells deutlich verbessert wurde. Jedoch betrug die *Accuracy* mit 89.72 % dennoch minimal ( $-0.62\%$ ) weniger als die *Accuracy* des Modells vor der Adaption durch das Fine-Tuning. Das Training einer neuen log. Regression führt demnach zu einer Auflösung der Verschlechterung der Modelle durch das Fine-Tuning, da die Modelle durch die neue log. Regression eine ähnliche Performance, wie vor dem Fine-Tuning erreichen.

## 7. Fazit & Ausblick

In den beiden folgenden Unterabschnitten sollen die Ergebnisse dieser Arbeit resümiert werden. Hierzu wird in Unterabschnitt 7.1 zunächst ein Fazit gezogen. In Unterabschnitt 7.2 wird abschließend ein Ausblick gegeben.

### 7.1. Fazit

Im Rahmen dieser Arbeit wurde der Einfluss des *Meta-Learnings* auf die Minimierung des *Concept Drifts* von Modellen zur Anomalieerkennung untersucht. Dafür wurden im Rahmen der Konzeptierung zunächst Annahmen und Thesen definiert, um anschließend eine Hypothese aufzustellen. Die Untersuchung der Hypothese wurde im Rahmen eines Experimentes vorgenommen. Dazu wurde ein Szenario definiert und die entsprechenden Daten mittels eines *Data Generators* erzeugt. Die aus diesem Szenario resultierenden Daten konnten anschließend im Rahmen der Durchführung des Experimentes genutzt werden, um Modelle zur Anomalieerkennung zu trainieren und evaluieren. Für eine strukturierte und nachvollziehbare Durchführung des Experiments wurden statistische Versuchspläne erstellt und ausgeführt. Die Auswertung der Daten des Experiments ergab zunächst, dass das *Meta-Learning* keinen positiven Einfluss auf die Modellperformance aufweist. Bei einer detaillierteren Analyse der Daten stellte sich jedoch heraus, dass die valide Beurteilung des Einflusses des *Meta-Learnings* nicht vorgenommen werden kann, da die Implementierung des *Meta-Trainings* im Kontext des Szenarios nicht geeignet war. Daher ist eine abschließende Untersuchung der Hypothese im Rahmen der vorliegenden Arbeit nicht möglich.

Die statistischen Versuchspläne wurden im Rahmen der Planung des Experimentes insbesondere dafür genutzt, um die große Anzahl an Versuchen aus einem vollfaktoriellen Versuchspräzisionsplan einzuschränken. Dafür wurden teilstatistische Versuchspläne erstellt, welche 10 % der Kombinationen aus dem dazugehörigen vollfaktoriellen Versuchspräzisionsplan enthalten. Die Reduzierung der Versuchspläne war notwendig, da das Training aller Modelle zu viel Zeit benötigt hätte. Bzgl. der Einschränkung der Versuchspläne sind zwei Punkte kritisch zu betrachten. Die Größe der teilstatistischen Versuchspläne stellt einen Kompromiss aus der benötigten Zeit zur Ausführung eines Planes und der Repräsentativität eines Planes dar. Dafür wurde im Kontext dieser Arbeit eine eigene Definition der Repräsentativität verwendet. Die Minimierung der Versuchspläne auf 10 % stellt aus statistischen Gesichtspunkten jedoch vermutlich keinen vollständig repräsentativen Versuchspräzisionsplan mehr dar. Darüber hinaus wurde bei der Auswahl der Kombinationen ein zufälliges *Sampling* implementiert, welches zusätzliche Bedingungen bei der Auswahl der Versuche berücksichtigt. Das rein zufällige *Sampling* garantiert dabei jedoch keine raumfüllende Auswahl mehr. Besser wäre hierfür die Anwendung verschiedener *Sampling*-Strategien, welche im Kontext der statistischen Versuchspräzisionsplanung bereits kurz diskutiert wurden.

Im Rahmen des Experimentes wurde für alle Modelle  $M_1$  und  $M_2$  jeweils die gleiche Modelltopologie des zugrundeliegenden AE verwendet. Dies impliziert die Annahme, dass die Topologie der AE keinen Einfluss auf die Sensitivität eines Modells bzgl. neuer Aufgaben hat. Die Verwendung der gleichen Modelltopologie ermöglicht zwar den Vergleich der Modelle  $M_1$  und  $M_2$  untereinander, jedoch ist dadurch nicht ausgeschlossen, dass die gewählte Topologie per se nicht für das *Meta-Learning* geeignet ist. Während einer kurzen Recherche konnte keine Literatur zu diesem Thema gefunden werden.

Eine Untersuchung des Einflusses der Topologie eines ANN auf die Fähigkeit des *Meta-Learnings* scheint nach den Erkenntnissen dieser Arbeit jedoch sinnvoll.

Das Ergebnis eines AE ist zunächst ein *Reconstruction Error*, welcher nicht dazu verwendet werden kann ein *Sample* als Anomalie zu klassifizieren. Bei der Verwendung von AE zur Anomalieerkennung werden daher häufig Schwellwerte zur Umwandlung der *Reconstruction Error* in binäre *Label* genutzt. Im Rahmen einer durchgeführten Vorstudie wurde gezeigt, dass die Suche eines Schwellwerts pro AE einerseits sehr zeitintensiv ist und andererseits keine guten Ergebnisse liefert. Bei der Implementierung der AE für die Durchführung des Experimentes wurde daher eine logistische Regression zur Klassifikation verwendet, welche schnell zu trainieren ist und gute Ergebnisse liefert. Die Evaluation der AE nach dem Experiment wurde anschließend anhand der Kennzahlen der *Confusion Matrices* vorgenommen, welche anhand der Vorhersagen der logistischen Regression bestimmt wurden. In Unterabschnitt 6.6.5 wurde dieses Konzept bereits kurz kritisiert. Die Verwendung der logistischen Regression ist an dieser Stelle kritisch zu betrachten, da nicht ausgeschlossen werden kann, dass die Verwendung der logistischen Regression die Ergebnisse des Experimentes verzerren.

Für die Manipulation der Daten durch Erzeugung eines *Concept Drifts* und Anomalien anhand des eigens entwickelten *Data Generators* wird zuvor die Stärke, Art und Stelle einer Anomalie bzw. des *Concept Drift* Events definiert. Dabei wurden sowohl für die Arten der Anomalien, als auch für die Arten der *Concept Drift* Events unterschiedliche Anzahlen an Manipulationen vorgenommen. So gibt es in dem Datensatz  $\mathbf{X}_{drifted,anormal}$  nur 360 Manipulationen des Phasenwinkels, aber 2100 manipulierte *Samples* durch eine Veränderung des Lastprofils. Bei der Beurteilung der Fähigkeit eines Modells, gruppiert nach Art der Anomalie oder Art des *Concept Drift* Events, ist daher jeweils die Anzahl der Manipulationen zu berücksichtigen.

## 7.2. Ausblick

Wie in Unterabschnitt 7.1 bereits dargestellt wurde, konnte die Untersuchung der Hypothese nicht abgeschlossen werden. Dies wurde hauptsächlich darauf zurückgeführt, dass während des *Meta-Trainings* keine unterschiedlichen Aufgaben verwendet werden konnten. Ein offener und wichtiger Untersuchungsgegenstand ist daher die Definition unterschiedlicher Aufgaben bei dem Training von AE zur Anomalieerkennung. Sofern dies gelingt, kann die Untersuchung der Hypothese unter Verwendung der bisherigen Erkenntnisse wieder aufgenommen werden.

Bei der Durchführung des Experimentes wurden eine Vielzahl unterschiedlicher Kennzahlen aufgezeichnet und insgesamt 360 unterschiedliche Modelle trainiert. Bei der Auswertung dieser Daten in Unterabschnitt 6.5 wurde nur ein kleiner Teil dieser Daten verwendet, da der Fokus der Auswertung auf der Untersuchung der Hypothese lag. Die Daten ermöglichen aber generell noch weitere Analysen in verschiedene Richtungen. Die Verwendung der statistischen Versuchsplanung könnte beispielsweise Antworten darauf geben, welcher Faktor den größten oder geringsten Einfluss während des Experimentes aufwies.

Zur Reduzierung der Komplexität wurden zu Beginn der Durchführung des Experimentes die Daten von insgesamt 97 auf 17 *Features* reduziert. Die Anomalieerkennung wurde anschließend in Form eines simplen *feedforward* AE umgesetzt. Im Rahmen weiterer Arbeiten könnte die Reduzierung der *Features* wieder aufgehoben werden. Für die dann komplexere Anomalieerkennung könnten verschiedene Modelltypen rekurrenter ANN, wie LSTM-AE oder LSTM-VAE verwendet und untersucht wer-

den. Auch generative Modelle, wie der *AnoGAN* (von Schlegl et al. in [Sch+17] eingeführt), könnten in diesem Kontext untersucht werden.

Im Rahmen dieser Arbeit wurde nur die Auswirkung des *Meta-Learnings* mittels MAML bzw. FO-MAML untersucht. Neben diesem Ansatz existieren in der Literatur noch weitere Ansätze des *few-shot Learnings*, wie z.B. die Verwendung von *Siamese Networks*. Im Rahmen weiterer Arbeiten könnten diese Ansätze daher im Kontext der Anomalieerkennung weiter evaluiert und verglichen werden. Auch der Vergleich des *Meta-Learnings* mit *Transfer Learning* kann einen weiteren interessanten Forschungsgegenstand darstellen.



## A. Abbildungen

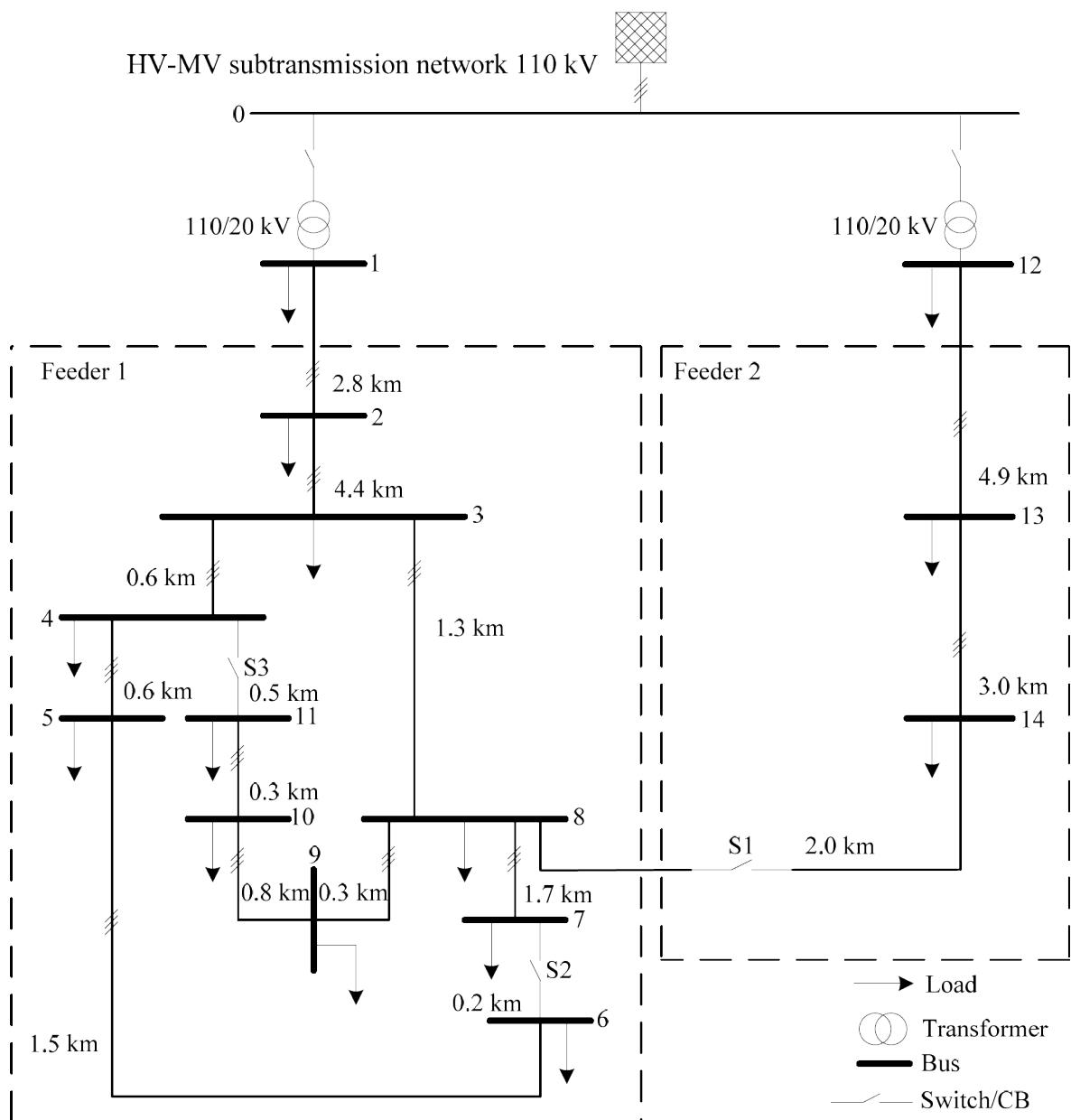


Abbildung A.1.: CIGRE Benchmark Mittelspannungsverteilnetz-Referenzmodell [Str+14]

		<b>p_mw</b>	<b>q_mw</b>
2020-01-01 00:00:00	AGG_BUS_1	0.142169	0.068856
	AGG_BUS_2	0.234400	0.113525
	AGG_BUS_3	-0.031800	-0.015401
	AGG_BUS_4	0.065500	0.031723
	AGG_BUS_5	0.202800	0.098221
	AGG_BUS_6	0.133100	0.064463
	AGG_BUS_7	0.198600	0.096186
	AGG_BUS_8	0.067600	0.032740
	AGG_BUS_9	0.202800	0.098221
	AGG_BUS_10	0.000000	0.000000
	AGG_BUS_11	0.135200	0.065480
	AGG_BUS_12	0.000000	0.000000
	AGG_BUS_13	-0.054000	-0.026153
	AGG_BUS_14	-0.027000	-0.013077

Abbildung A.2.: Beispiel-Matrix zur Aktualisierung der Wirk- und Blindleistungswerte in PandaPower für einen Zeitpunkt (eigene Darstellung)

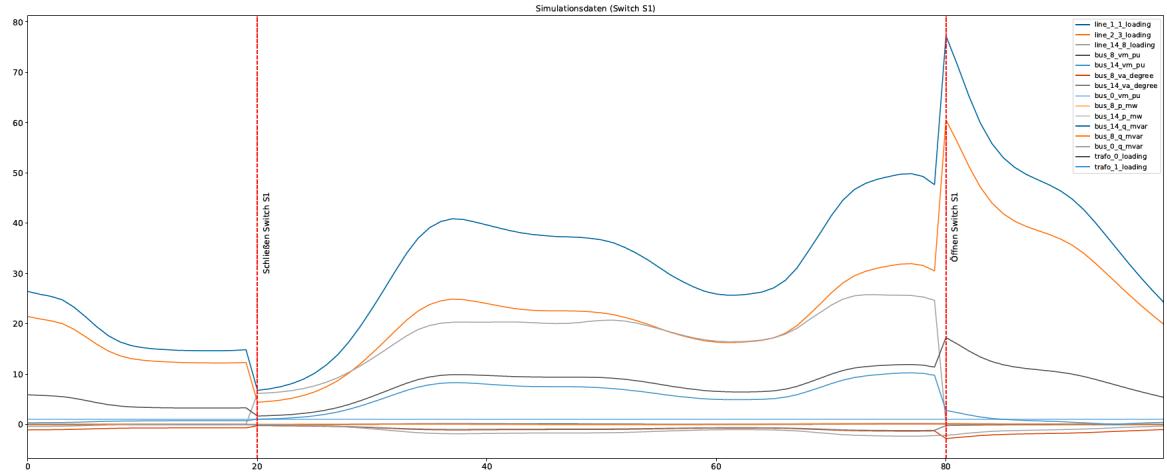


Abbildung A.3.: Auswirkung des Öffnens und Schließens des Schalters S1 im PandaPower CIGRE Netz auf die Messwerte an den entsprechenden Leitungen und Bussen (eigene Darstellung)

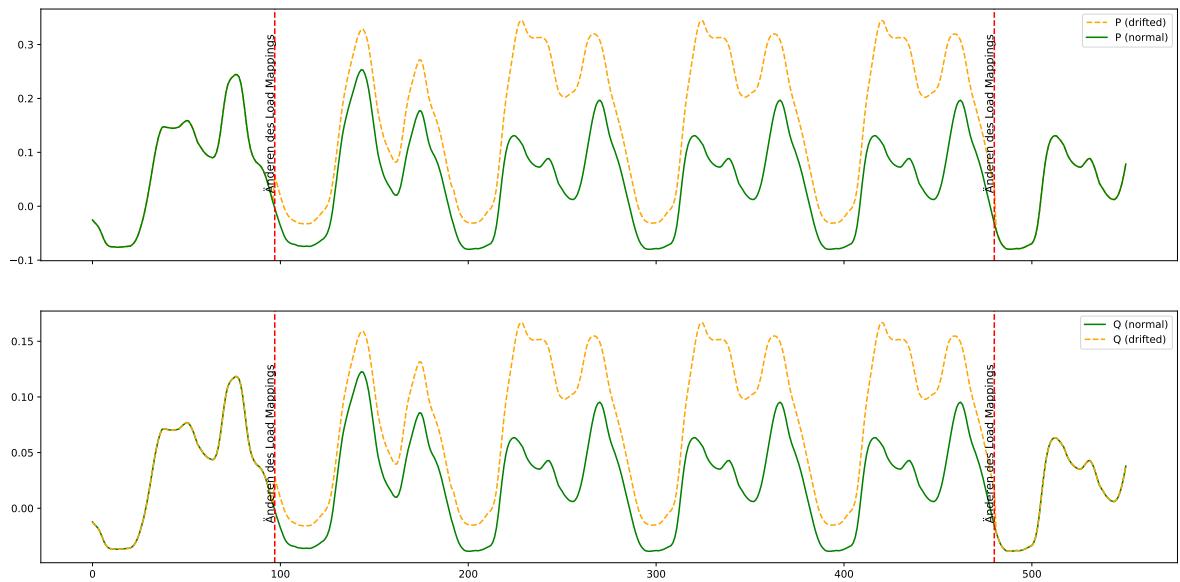


Abbildung A.4.: Auswirkung auf die Wirk- und Blindleistungswerte durch eine Veränderung des Load Mappings über einen Zeitraum von fünf Tagen (eigene Darstellung)

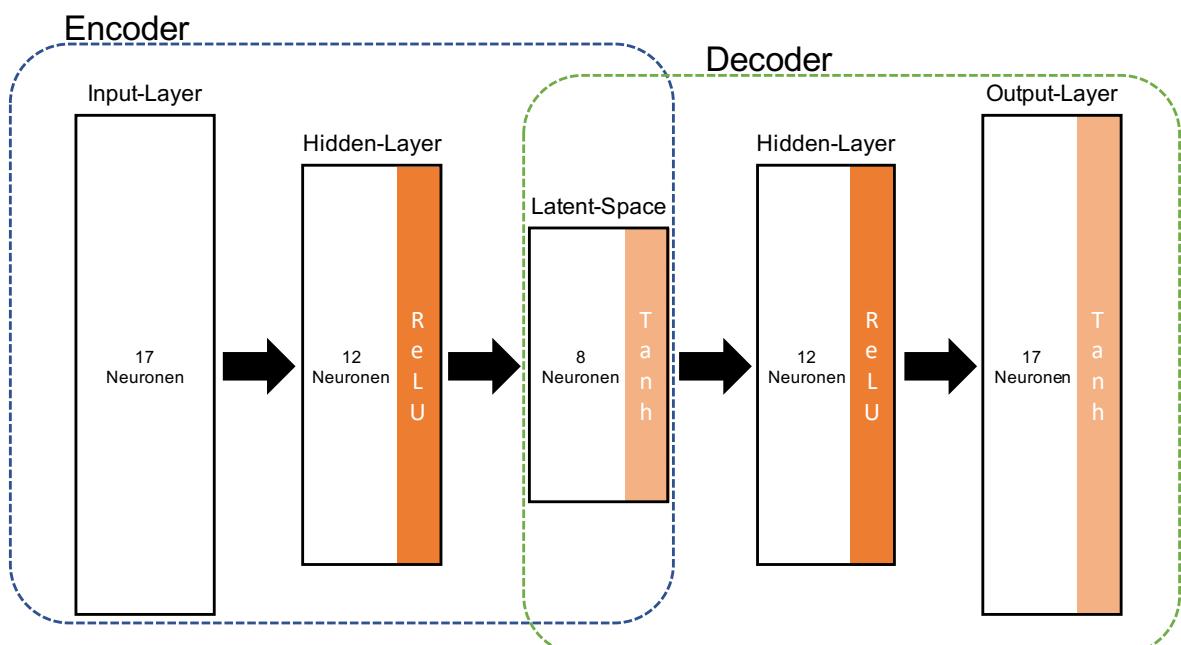


Abbildung A.5.: Schematische Darstellung der Topologie des verwendeten AutoEncoders  
(eigene Darstellung)

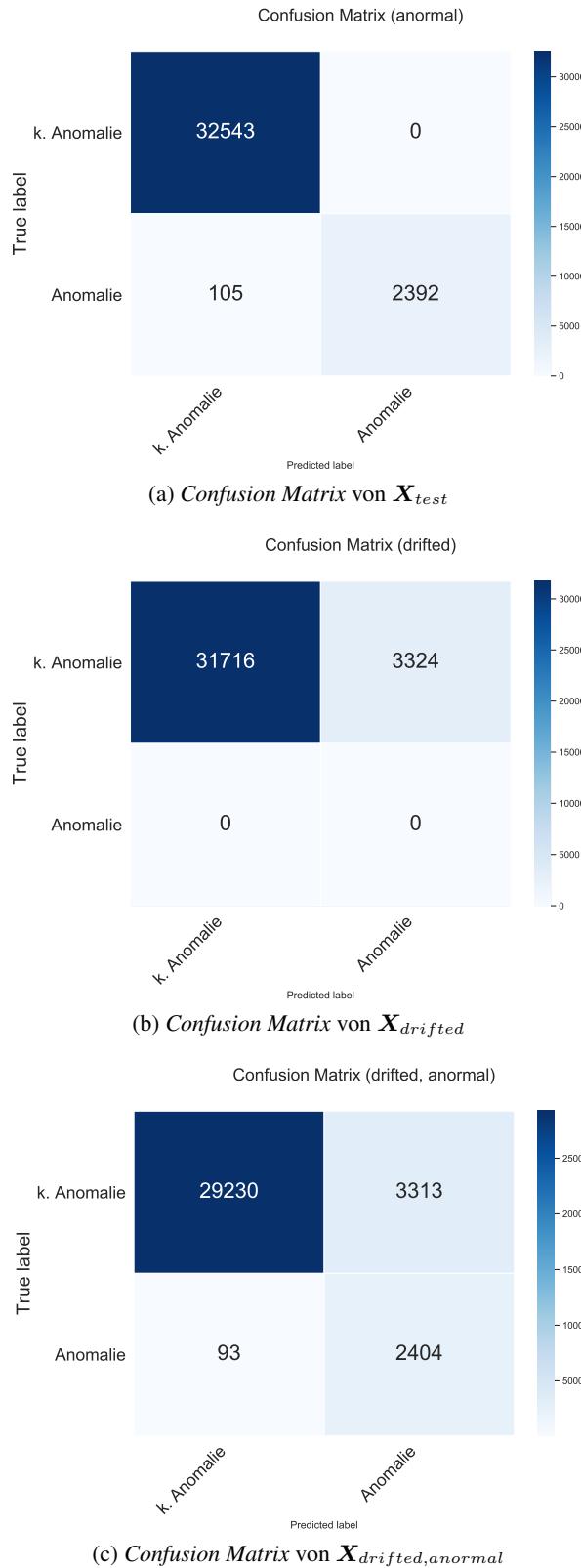


Abbildung A.6.: Drei Confusion Matrices mit der Anzahl der jeweils richtig und falsch Klassifizierten Samples auf den unterschiedlichen Datensätzen  $\mathbf{X}_{test}$ ,  $\mathbf{X}_{drifted}$  sowie  $\mathbf{X}_{drifted,anormal}$  des Modells  $M_2$

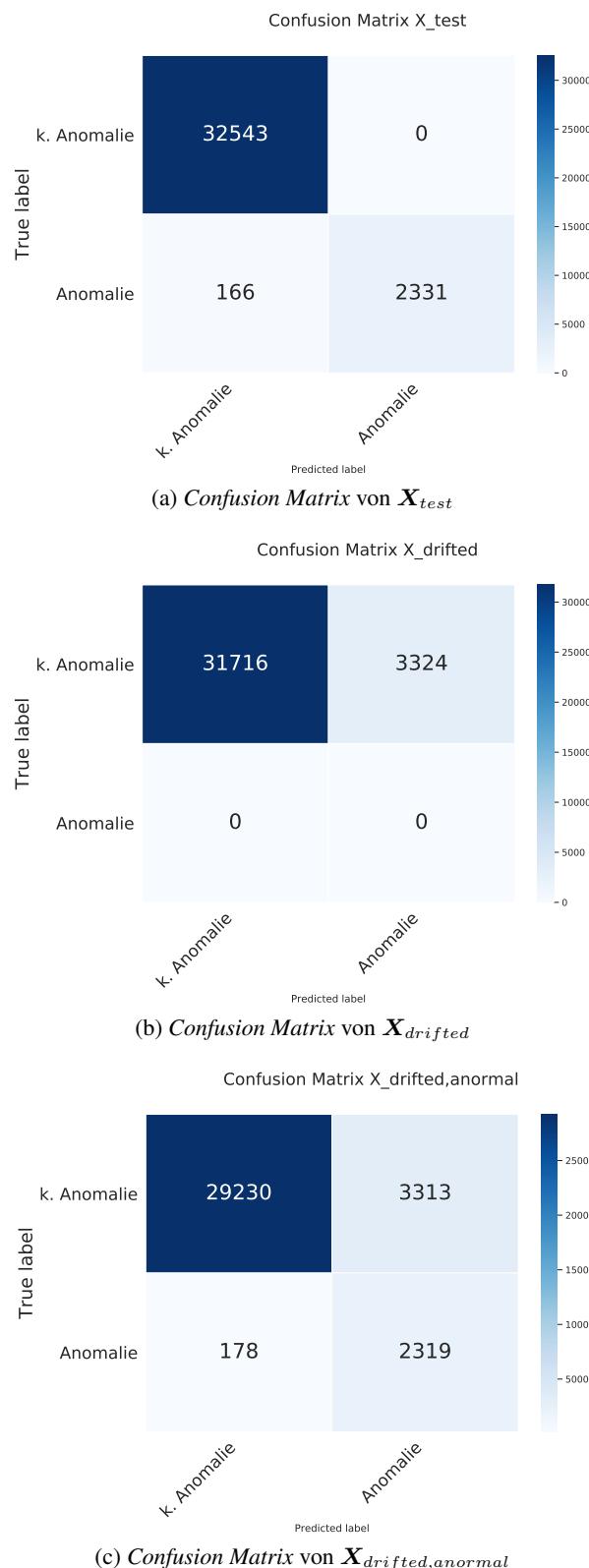


Abbildung A.7.: Drei Confusion Matrices mit der Anzahl der jeweils richtig und falsch Klassifizierten Samples auf den unterschiedlichen Datensätzen  $X_{test}$ ,  $X_{drifted}$  sowie  $X_{drifted,anormal}$  des Modells  $M_1$

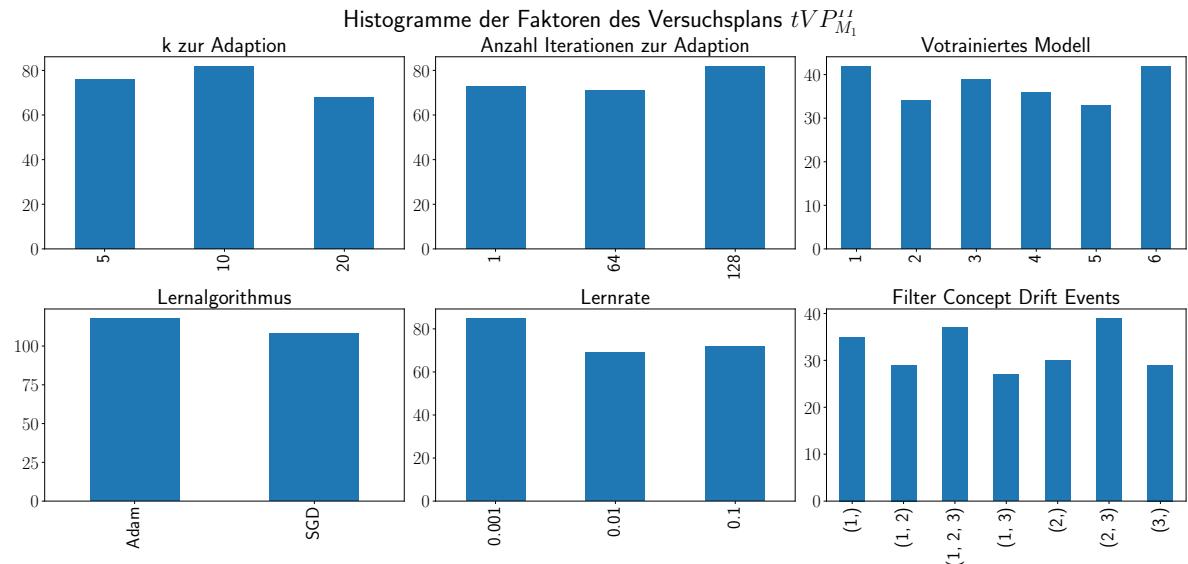


Abbildung A.8.: Histogramme der Faktoren des Versuchsplans  $tVP_{M_1}^{II}$  (eigene Darstellung)

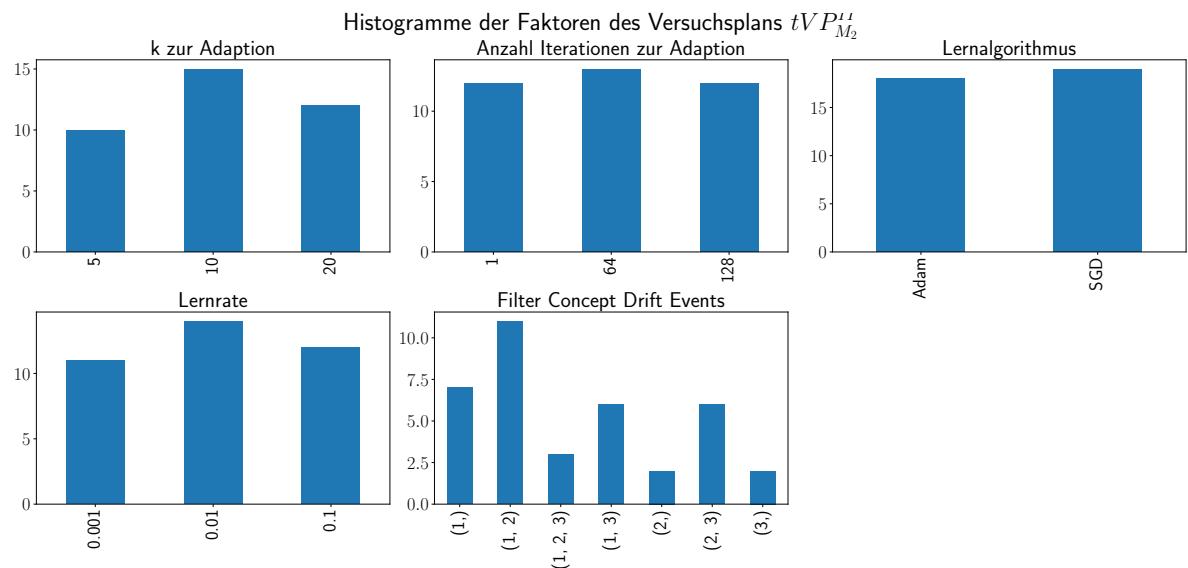


Abbildung A.9.: Histogramme der Faktoren des Versuchsplans  $tVP_{M_2}^{II}$  (eigene Darstellung)



## B. Code-Listings

In dem folgenden Listing B.1 ist die Berechnung der Wirk- und Blindleistungswerte für mehrere Zeitpunkte  $t$  dargestellt. In Zeile 4 wird zunächst ein leerer *Pandas DataFrame* initialisiert. In Zeile 6 bis 22 ist das *Python Dictionary* für die Zuordnung der Profile pro Bus dargestellt. In Zeile 24 und 25 werden die Lasten in Form der Vektor-Vektor-Multiplikation berechnet. Der *DataFrame* *df\_base* enthält dabei die entsprechenden SLP und SEP. Die Funktion zur Berechnung der Blindleistung ist in Zeile 27 bis 29 dargestellt. In Zeile 31 bis 43 werden die für jeden Bus und jeden Zeitpunkt berechneten  $P$  und  $Q$  Werte schließlich in dem in Zeile 4 initialisierten *DataFrame* zusammengefasst.

```

1 import pandas as pd
2 import numpy as np
3
4 df_data = pd.DataFrame()
5
6 load_mapping = {
7     #          0   1   2   3   4   5
8     1: np.array([2, 1, 0, 0, 0, 1]),
9     2: np.array([4, 2, 0, 1, 1, 0]),
10    3: np.array([2, 0, 0, 0, 1, 0]),
11    4: np.array([0, 1, 0, 0, 0, 0]),
12    5: np.array([3, 0, 0, 2, 0, 0]),
13    6: np.array([1, 1, 0, 0, 0, 0]),
14    7: np.array([1, 2, 0, 0, 0, 0]),
15    8: np.array([1, 0, 0, 1, 0, 0]),
16    9: np.array([3, 0, 0, 0, 0, 0]),
17    10: np.array([0, 0, 0, 0, 0, 0]),
18    11: np.array([2, 0, 0, 0, 0, 0]),
19    12: np.array([0, 0, 0, 0, 0, 0]),
20    13: np.array([2, 0, 2, 0, 2, 0]),
21    14: np.array([1, 0, 1, 0, 1, 0]),
22 }
23
24 for bus, load in load_mapping.items():
25     df_data['AGG_BUS_{}'.format(bus)] = np.dot(load, df_base.T)
26
27 def compute_q(p, cos_phi=0.9):
28     abs_q = p * np.tan(np.arccos(cos_phi))
29     return -1 * abs_q
30
31 dfs = {}
32 for col in df_data.cols:
33     tmp = pd.DataFrame(index=df_data.index, columns=['p_mw', 'q_mw'])
34     tmp['p_mw'] = df_data[col]
35     dfs[col] = tmp
36
37 for key, val in dfs.items():
38     tmp_q = []
39     for val in val.iterrows():
40         p = val[1]['p_mw']
41         q = compute_q(p)
42         tmp_q.append(q)
43     dfs[key]['q_mw'] = tmp_q

```

Listing B.1: Berechnung der Wirk- und Blindleistungswerte in Python

In dem Listing B.2 ist die Initialisierung eines leeren CIGRE Mittelspannungsnetzes (Zeile 6 und 7) sowie die Modellierung der Lasten pro Bus (Zeile 9 bis 29) dargestellt.

```

1 import pandas as pd
2 import numpy as np
3 import pandapower as pp
4 from pandapower import networks
5
6 grid = pp.networks.create_cigre_network_mv(with_der=False)
7 grid.load = grid.load[0:0]
8
9 load_mapping = {
10     #      0  1  2  3  4  5
11     1: np.array([2, 1, 0, 0, 0, 1]),
12     2: np.array([4, 2, 0, 1, 1, 0]),
13     3: np.array([2, 0, 0, 0, 1, 0]),
14     4: np.array([0, 1, 0, 0, 0, 0]),
15     5: np.array([3, 0, 0, 2, 0, 0]),
16     6: np.array([1, 1, 0, 0, 0, 0]),
17     7: np.array([1, 2, 0, 0, 0, 0]),
18     8: np.array([1, 0, 0, 1, 0, 0]),
19     9: np.array([3, 0, 0, 0, 0, 0]),
20     10: np.array([0, 0, 0, 0, 0, 0]),
21     11: np.array([2, 0, 0, 0, 0, 0]),
22     12: np.array([0, 0, 0, 0, 0, 0]),
23     13: np.array([2, 0, 2, 0, 2, 0]),
24     14: np.array([1, 0, 1, 0, 1, 0]),
25 }
26
27 for bus_id in load_mapping:
28     pp.create_load(grid, bus=bus_id, p_mw=0., q_mvar=0., name='AGGR_%d' % bus_id,
29                     scaling=1.,
30                     in_service=True, controllable=False)

```

Listing B.2: Berechnung der Wirk- und Blindleistungswerte in Python

In Listing B.3 ist ein Auszug des Codes zur Simulation der Lastflussrechnungen in *PandaPower* dargestellt. In Zeile 5 wird zunächst die Anzahl der Schleifendurchläufe anhand der Dauer bestimmt. In Zeile 6 und 7 werden Parameter des Algorithmus zur Lastflussrechnung in *PandaPower* gesetzt. In Zeile 9 bis 16 ist die Funktion `step(t)` dargestellt, die für einen Zeitschritt  $t$  zunächst die Wirk- und Blindleistungswerte des Netzes aktualisiert (Zeile 13) und anschließend die Lastflussrechnung ausführt (Zeile 16). Die Simulation in Form der Schleife ist von Zeile 18 bis zum Ende des Listings dargestellt. Die Funktion `collect_data(grid)` ist nicht explizit im Listing aufgeführt. Die Aufgabe der Funktion ist das Sichern aller Messwerte des Netzes in einem Zeitpunkt  $t$ .

```

1 import pandas as pd
2 import numpy as np
3 import pandapower as pp
4
5 duration = (366 + 365 + 365 + 365) * 96
6 init = 'auto'
7 algorithm = 'bfsw'
8
9 def step(t):
10     ts = timestamps[t]
11

```

```

12     # update load values
13     grid.load.update(data.loc[ts])
14
15     # run power flow
16     pp.runpp(grid, algorithm=algorithm, init=init)
17
18 for t in range(duration):
19     # run simulation
20     try:
21         step(t)
22         collect_data(grid)
23
24         if t%1000 == 0:
25             print('Current step: {}'.format(t))
26     except Exception as e:
27         print(e)

```

Listing B.3: Durchführung der Simulation in PandaPower

In Listing B.4 ist ein Auszug zur Erstellung der verschiedenen Anomalien dargestellt. In Zeile 1 bis 16 ist zunächst die Definition der Art (*collective*, *point* oder *noise*), der Länge sowie der Stärke der Anomalien in Form eines *Python Dictionaries* dargestellt. In den folgenden Funktionen werden die eigentlichen Anomalien erstellt. Die Haupt-Funktion zur Erstellung der Anomalien ist in Zeile 60 bis 63 dargestellt. Diese Funktion ruft die drei Funktionen `collective_anomaly()`, `point_anomaly()` und `noise_anomaly()` auf. In den jeweiligen Funktionen werden dann die Arten der Anomalien erzeugt.

```

1 outliers = {
2     'collective': [{ 'feature': 'line_1_1_loading',
3                     'timestamps': [(1000, 1222), (4560, 4580), (30001, 30050),
4                     (15000, 15024)],
5                     'factors': [40, 100, 120, 30]
6                 },
7
8     'point': [{ 'feature': 'line_2_3_loading',
9                 'timestamps': [10, 30, 10000, 14420, 17010, 28021, 34340],
10                'factors': [40, 100, 120, 200, 10, 80, 100],
11            },
12
13     'noise': [{ 'feature': 'line_7_8_loading',
14                 'timestamps': [(10, 30), (4000, 4020), (26666, 26670)],
15                 'factors': [10, 3, 5],
16             }]
17 }
18
19 def create_noise_anomaly(column, start_ts, end_ts, factor):
20     selected_vals = anomalous_data[column].iloc[start_ts:end_ts].copy()
21     noise = np.random.normal(1, factor, len(selected_vals))
22     anomalous_vals = selected_vals + noise
23     anomalous_data[column].iloc[start_ts:end_ts] = anomalous_vals
24     anomalous_data['label'].iloc[start_ts:end_ts] = [3] * len(selected_vals)
25
26 def collective_anomaly(collectivs):
27     for feature in collectivs:
28         col = feature['feature']
29         timestamps = feature['timestamps']

```

```

29     factors = feature['factors']
30     assert len(timestamps) == len(factors)
31     for event, fact in zip(timestamps, factors):
32         start_ts = event[0]
33         end_ts = event[1]
34         factor = anomalous_data[col].iloc[start_ts-1] / 100 * fact
35         if end_ts > start_ts:
36             create_collective_anomaly(col, start_ts, end_ts, factor)
37
38 def point_anomaly(points):
39     for feature in points:
40         col = feature['feature']
41         timestamps = feature['timestamps']
42         factors = feature['factors']
43         assert len(timestamps) == len(factors)
44         for event, fact in zip(timestamps, factors):
45             point_ts = event
46             factor = anomalous_data[col].iloc[point_ts-1] / 100 * fact
47             create_point_anomaly(col, point_ts, factor)
48
49 def noise_anomaly(noises):
50     for feature in noises:
51         col = feature['feature']
52         timestamps = feature['timestamps']
53         factors = feature['factors']
54         assert len(timestamps) == len(factors)
55         for event, fact in zip(timestamps, factors):
56             start_ts = event[0]
57             end_ts = event[1]
58             create_noise_anomaly(col, start_ts, end_ts, fact)
59
60 def make_anomalous(outliers):
61     collective_anomaly(outliers['collective'])
62     point_anomaly(outliers['point'])
63     noise_anomaly(outliers['noise'])

```

Listing B.4: Code zur Erzeugung der unterschiedlichen Anomalien

In Listing B.5 ist ein Auszug zur Definition der Maßnahmen zur Erzeugung eines *Concept Drifts* in Form eines *Python Dictionary* dargestellt. Für jedes *Event* wird dabei die betreffende Komponente im *PandaPower* Netz sowie den Start als Index sowie das Ende des Events (sofern notwendig) definiert.

```

1 manipulate_switch = [
2     {
3         'switch_id': 4,
4         'set_closed': True,
5         'at_time_idx': 20
6     },
7     {
8         'switch_id': 4,
9         'set_closed': False,
10        'at_time_idx': 80
11    }
12 ]
13
14 load_mapping = [

```

```

15     {
16         'bus_id': 1,
17         'load_mapping': [3, 0, 2, 0, 1, 0],
18         'at_time_idx': 323,
19         'until_time_idx': 345
20     },
21     {
22         'bus_id': 3, # three means -> 'AGG_BUS_3'
23         'load_mapping': [2, 1, 0, 0, 1, 0],
24         'at_time_idx': 97,
25         'until_time_idx': 5 * 96
26     }
27 ]
28
29 concept_drift = {}
30 concept_drift['manipulate_switch'] = manipulate_switch
31 concept_drift['load_mapping'] = load_mapping

```

Listing B.5: *Code zur Definition mehrerer Events zur Erzeugung eines Concept Drifts*

In dem Listing B.6 ist ein Auszug der Implementierung der Python Klasse Drifter dargestellt, welche die Funktionen und Methoden zur Erzeugung eines *Concept Drifts* anhand der in Listing B.5 dargestellten Events implementiert.

```

1 import pandas as pd
2 import numpy as np
3 import pandapower as pp
4 import GridUtils as gu
5
6 class Drifter:
7
8     def __init__(self, grid, concept_drifts, unscaled_data, timestamp_list,
9      agg_data):
10         self.grid = grid
11         self.unscaled_data = unscaled_data
12         self.data = agg_data
13         self.timestamp_list = timestamp_list
14         self.events_manipulate_switch = concept_drifts['manipulate_switch']
15         self.events = []
16
17     def check_timestamps(self, ts):
18         result_cdl, events_cdl = self._check_manipulate_switch_ts(ts)
19
20         return result_cdl, events_cdl
21
22     def manipulate_switch(self, event):
23         self.events.append(event)
24         switch_no = event['switch_id']
25         switch_status = event['set_closed']
26         ts = event['at_time_idx']
27
28         self.grid.switch.at[switch_no, 'closed'] = switch_status
29         return self.grid
30
31     def manipulate_load_mapping(self):
32         for event in self.events_load_mapping:
33             start_ts = self.timestamp_list[event['at_time_idx']]

```

```

33     end_ts = self.timestamp_list[event['until_time_idx']]
34     start_ts_i = event['at_time_idx']
35     end_ts_i = event['until_time_idx']
36
37     if end_ts_i == -1:
38         ts_list = self.timestamp_list[start_ts_i:]
39     else:
40         end_ts_i += 1
41         ts_list = self.timestamp_list[start_ts_i:end_ts_i]
42
43     bus_id = event['bus_id']
44     bus_tag = 'AGG_BUS_{}'.format(bus_id)
45     new_load_mapping = event['load_mapping']
46     old_load_mapping = self.initial_load_mapping[bus_id]
47
48     tmp_data = self.unscaled_data.loc[start_ts:end_ts]
49     agg_data_p = np.dot(new_load_mapping, tmp_data.T)
50
51     q_list = []
52     for val in agg_data_p:
53         q = gu.compute_q(val)
54         q_list.append(q)
55
56     assert len(ts_list) == len(q_list) == len(agg_data_p)
57
58     for t_step, q_val, p_val in zip(ts_list, q_list, agg_data_p):
59         self.data.loc[(t_step, bus_id-1)] = [bus_tag, p_val, q_val]
60
61     return self.data
62
63 def _check_manipulate_switch_ts(self, ts):
64     events = []
65     result = False
66     for event in self.events_manipulate_switch:
67         if event['at_time_idx'] == ts:
68             events.append(event)
69             result = True
70
71     return result, events

```

Listing B.6: Implementierung der NetworkTopologyChanger Komponente als Python Klasse mit dem Namen Drifter

In Listing B.7 ist die Implementierung des *Meta-Learnings* mittels *l2l* als Learner Klasse in Python dargestellt. Die Klasse Learner erhält bei der Initialisierung alle wichtigen Parameter, wie z.B. Hyper-Parameter des *Meta-Learnings*, das zu trainierende Modell und die entsprechenden Datensätze als TaskDataset Klasse übergeben.

```

1 import learn2learn as l2l
2 import torch
3 from tqdm import tqdm
4
5
6 class Learner:
7
8     def __init__(self, train_task_dataset, eval_task_dataset, model,

```

```

9         adaption_steps=10, use_cuda=False, meta_lr=1e-3, lr=4e-3,
10        optimizer='Adam',
11        first_order=False, num_iterations=1000, tasks_per_step=32,
12        save_weights=False, k_shot=5):
13            self.train_task_data = train_task_dataset
14            self.eval_task_data = eval_task_dataset
15            self.model = model
16            self.meta_model = None
17            self._init_meta_model(meta_lr, first_order)
18            self.optimizer = None
19            self._init_optimizer(lr=lr, optimizer=optimizer)
20            self.num_iterations = num_iterations
21            self.adaption_steps = adaption_steps
22            self.device = torch.device("cuda" if use_cuda else "cpu")
23            self.adaption_loss = []
24            self.evaluation_loss = []
25            self.iteration_error_list = []
26            self.iteration_error = None
27            self.mse_loss = torch.nn.functional.mse_loss
28            self.tasks_per_step = tasks_per_step
29            self.weight_matrices = {'encoder': [], 'decoder': []}
30            self.weight_matrices_meta_model = {'encoder': [], 'decoder': []}
31            self.weight_matrices_diff = {'encoder': [], 'decoder': []}
32            self.save_weights = save_weights
33            self.k_shot = k_shot
34
35    def _update_weight_matrices(self):
36        self.weight_matrices['encoder'].append(self.model.encoder.state_dict())
37        self.weight_matrices['decoder'].append(self.model.decoder.state_dict())
38
39    def _update_weight_matrices_of_meta_model(self):
40        self.weight_matrices_meta_model['encoder'].append(self.meta_model.encoder
41        .state_dict())
42        self.weight_matrices_meta_model['decoder'].append(self.meta_model.decoder
43        .state_dict())
44
45    def _update_weight_matrices_diff(self):
46        for i, _ in enumerate(self.weight_matrices['encoder']):
47            if i >= 1:
48                print('Calculate Diff for weights from iteration: {} - {}'.format
49                (i, i-1))
50                current_encoder_weights_l1 = self.weight_matrices['encoder'][i][
51                '0.weight'].numpy()
52                current_encoder_weights_l2 = self.weight_matrices['encoder'][i][
53                '2.weight'].numpy()
54
55                last_encoder_weights_l1 = self.weight_matrices['encoder'][i-1][
56                '0.weight'].numpy()
57                last_encoder_weights_l2 = self.weight_matrices['encoder'][i-1][
58                '2.weight'].numpy()
59
60                encoder_diff_weights_l1 = current_encoder_weights_l1 -
61                last_encoder_weights_l1
62                encoder_diff_weights_l2 = current_encoder_weights_l2 -
63                last_encoder_weights_l2

```

```

54         current_decoder_weights_11 = self.weight_matrices['decoder'][i]['
55             0.weight'].numpy()
56         current_decoder_weights_12 = self.weight_matrices['decoder'][i]['
57             2.weight'].numpy()
58
59         last_decoder_weights_11 = self.weight_matrices['decoder'][i-1]['
60             0.weight'].numpy()
61         last_decoder_weights_12 = self.weight_matrices['decoder'][i-1]['
62             2.weight'].numpy()
63
64         decoder_diff_weights_11 = current_decoder_weights_11 -
65         last_decoder_weights_11
66         decoder_diff_weights_12 = current_decoder_weights_12 -
67         last_decoder_weights_12
68
69         self.weight_matrices_diff['encoder'].append((
70             encoder_diff_weights_11, encoder_diff_weights_12))
71         self.weight_matrices_diff['decoder'].append((
72             decoder_diff_weights_11, decoder_diff_weights_12))
73
74     def __init_meta_model(self, meta_lr, first_order):
75         self.meta_model = l2l.algorithms.MAML(self.model, lr=meta_lr, first_order
76 =first_order)
77
78     def __init_optimizer(self, lr, optimizer):
79         if optimizer == 'Adam' or optimizer == 'adam':
80             self.optimizer = torch.optim.Adam(self.meta_model.parameters(), lr=lr
81         )
82         elif optimizer == 'SGD' or optimizer == 'sgd':
83             self.optimizer = torch.optim.SGD(self.meta_model.parameters(), lr=lr)
84         else:
85             raise NotImplementedError
86
87     def start_learning_phase(self):
88         """
89         Train initial model with random weights!
90         :return: nothing!
91         """
92
93         tqdm_bar = tqdm(range(self.num_iterations))
94         self.meta_model.module.train()
95
96         for _ in tqdm_bar:
97             iteration_error = 0.0
98
99             # For Loop samples a number of tasks from the train data!
100            for task in self.train_task_data:
101                learner = self.meta_model.clone()
102
103                # Split data for adaption and evaluation
104                adaption_data = task[0]
105                adaption_data = adaption_data.to(self.device)
106
107                eval_data = self.eval_task_data.sample()[0]
108                eval_data = eval_data.to(self.device)
109
110                # Fast Adaption

```

```

101         for step in range(self.adaption_steps):
102             train_error = self.mse_loss(learner(adaption_data),
103                                         adaption_data)
104             self.adaption_loss.append(train_error.item())
105             learner.adapt(train_error)
106
107             # Compute validation loss
108             predictions = learner(eval_data)
109             valid_error = self.mse_loss(predictions, eval_data)
110             valid_error /= len(eval_data)
111             iteration_error += valid_error
112             self.evaluation_loss.append(valid_error.item())
113
114             # Update status bar
115             tqdm_bar.set_description("Adaption Error: {:.6f}, Validation Error: {:.6f}"
116                                     .format(train_error.item(),
117                                             valid_error.item()))
118
119             # Take the meta-learning step
120             self.iteration_error_list.append(iteration_error.item())
121             self.optimizer.zero_grad()
122             iteration_error.backward()
123             self.optimizer.step()
124
125             # Save Weights for Analysis
126             if self.save_weights:
127                 self._update_weight_matrices_of_meta_model()

```

Listing B.7: Implementierung des *Meta-Learnings* mittels *l2l* als Learner Klasse in Python

In Listing B.8 ist die Implementierung des *Fine-Tunings* nach der *Meta-Learning* Phase in Form der Klasse *FineTuner* dargestellt.

```

1 import torch
2 from Experiments.FilteredDataset import FilteredDataset
3 from torch.nn.functional import mse_loss
4
5
6 class FineTuner:
7
8     def __init__(self, fine_tune_data, k, model, optimizer, fine_tune_classes,
9                  num_iterations):
10        self.fine_tune_data = fine_tune_data
11        self.model = model
12        self.optimizer = optimizer
13        self.k = k
14        self.fine_tune_classes = fine_tune_classes
15        self.num_iterations = num_iterations
16        self.all_losses_fine_tune = []
17        self.avg_losses_fine_tune = []
18
19    def _sample_fine_tune_data(self):
20        """
21            Sample k samples from n classes and set in class variable.
22            :return: Nothing.
23        """
24        if not isinstance(self.fine_tune_data, torch.utils.data.TensorDataset):

```

```

24         raise Exception('Fine Tune Data is not a TensorDataset!')
25
26     # select classes
27     if self.fine_tune_classes is not None:
28         dataset = FilteredDataset(self.fine_tune_data, self.fine_tune_classes)
29     else:
30         dataset = self.fine_tune_data
31
32     # select k samples
33     sampler = torch.utils.data.RandomSampler(dataset, replacement=True,
34     num_samples=self.k)
35     fine_tune_DataLoader = torch.utils.data.DataLoader(dataset, batch_size=1,
36     num_workers=1, sampler=sampler)
37
38     # draw distinct data
39     self.X = []
40     self.drift_label = []
41
42     # sample unique
43     for j, (x, y) in enumerate(fine_tune_DataLoader):
44         self.X.append(x)
45         self.drift_label.append(y)
46
47     def fine_tune(self):
48         self._sample_fine_tune_data()
49         self.model.train()
50
51         for _ in range(self.num_iterations):
52             inner_loss = 0.0
53             avg_inner_loss = 0.0
54             for i, (x, y) in enumerate(zip(self.X, self.drift_label)):
55                 x = torch.autograd.Variable(x)
56                 self.optimizer.zero_grad()
57
58                 pred = self.model(x)
59                 loss = mse_loss(pred, x)
60                 self.all_losses_fine_tune.append(loss.item())
61
62                 inner_loss += loss.item()
63                 avg_inner_loss = inner_loss / self.k
64
65                 # Backpropagation
66                 loss.backward()
67                 self.optimizer.step()
68
69             self.avg_losses_fine_tune.append(avg_inner_loss)

```

Listing B.8: *Implementierung des Fine-Tuning als FineTuner Klasse in Python*

## C. Tabellen

<b>Bezeichnung</b>	<b>Beschreibung</b>
line_1_1_loading	Prozentuale Last der Leitung zwischen Bus 1 und 1
line_2_3_loading	Prozentuale Last der Leitung zwischen Bus 2 und 3
line_3_4_loading	Prozentuale Last der Leitung zwischen Bus 3 und 4
line_4_5_loading	Prozentuale Last der Leitung zwischen Bus 4 und 5
line_5_6_loading	Prozentuale Last der Leitung zwischen Bus 5 und 6
line_7_8_loading	Prozentuale Last der Leitung zwischen Bus 7 und 8
line_8_9_loading	Prozentuale Last der Leitung zwischen Bus 8 und 9
line_9_10_loading	Prozentuale Last der Leitung zwischen Bus 9 und 10
line_10_11_loading	Prozentuale Last der Leitung zwischen Bus 10 und 11
line_3_8_loading	Prozentuale Last der Leitung zwischen Bus 3 und 8
line_12_13_loading	Prozentuale Last der Leitung zwischen Bus 12 und 13
line_13_14_loading	Prozentuale Last der Leitung zwischen Bus 13 und 14
line_14_8_loading	Prozentuale Last der Leitung zwischen Bus 14 und 8
trafo_0_loading	Prozentuale Last des Trafos 0
trafo_1_loading	Prozentuale Last des Trafos 1
bus_0_vm_pu	Spannung an Bus 0 in pu
bus_1_vm_pu	Spannung an Bus 1 in pu
bus_2_vm_pu	Spannung an Bus 2 in pu
bus_3_vm_pu	Spannung an Bus 3 in pu
bus_4_vm_pu	Spannung an Bus 4 in pu
bus_5_vm_pu	Spannung an Bus 5 in pu
bus_6_vm_pu	Spannung an Bus 6 in pu
bus_7_vm_pu	Spannung an Bus 7 in pu
bus_8_vm_pu	Spannung an Bus 8 in pu
bus_9_vm_pu	Spannung an Bus 9 in pu
bus_10_vm_pu	Spannung an Bus 10 in pu
bus_11_vm_pu	Spannung an Bus 11 in pu
bus_12_vm_pu	Spannung an Bus 12 in pu
bus_13_vm_pu	Spannung an Bus 13 in pu
bus_14_vm_pu	Spannung an Bus 14 in pu
bus_0_va_degree	Phasenwinkel an Bus 0 in Grad
bus_1_va_degree	Phasenwinkel an Bus 1 in Grad
bus_2_va_degree	Phasenwinkel an Bus 2 in Grad
bus_3_va_degree	Phasenwinkel an Bus 3 in Grad
bus_4_va_degree	Phasenwinkel an Bus 4 in Grad
bus_5_va_degree	Phasenwinkel an Bus 5 in Grad
bus_6_va_degree	Phasenwinkel an Bus 6 in Grad
bus_7_va_degree	Phasenwinkel an Bus 7 in Grad
bus_8_va_degree	Phasenwinkel an Bus 8 in Grad
bus_9_va_degree	Phasenwinkel an Bus 9 in Grad
bus_10_va_degree	Phasenwinkel an Bus 10 in Grad
bus_11_va_degree	Phasenwinkel an Bus 11 in Grad

*Fortsetzung auf nächster Seite*

Tabelle C.1 – Fortsetzung vorheriger Seite

<b>Bezeichnung</b>	<b>Beschreibung</b>
bus_12_va_degree	Phasenwinkel an Bus 12 in Grad
bus_13_va_degree	Phasenwinkel an Bus 13 in Grad
bus_14_va_degree	Phasenwinkel an Bus 14 in Grad
bus_0_p_kw	Wirkleistung an Bus 0 in kW
bus_1_p_kw	Wirkleistung an Bus 1 in kW
bus_2_p_kw	Wirkleistung an Bus 2 in kW
bus_3_p_kw	Wirkleistung an Bus 3 in kW
bus_4_p_kw	Wirkleistung an Bus 4 in kW
bus_5_p_kw	Wirkleistung an Bus 5 in kW
bus_6_p_kw	Wirkleistung an Bus 6 in kW
bus_7_p_kw	Wirkleistung an Bus 7 in kW
bus_8_p_kw	Wirkleistung an Bus 8 in kW
bus_9_p_kw	Wirkleistung an Bus 9 in kW
bus_10_p_kw	Wirkleistung an Bus 10 in kW
bus_11_p_kw	Wirkleistung an Bus 11 in kW
bus_12_p_kw	Wirkleistung an Bus 12 in kW
bus_13_p_kw	Wirkleistung an Bus 13 in kW
bus_14_p_kw	Wirkleistung an Bus 14 in kW
bus_0_q_mvar	Blindleistung an Bus 0 in Var
bus_1_q_mvar	Blindleistung an Bus 1 in Var
bus_2_q_mvar	Blindleistung an Bus 2 in Var
bus_3_q_mvar	Blindleistung an Bus 3 in Var
bus_4_q_mvar	Blindleistung an Bus 4 in Var
bus_5_q_mvar	Blindleistung an Bus 5 in Var
bus_6_q_mvar	Blindleistung an Bus 6 in Var
bus_7_q_mvar	Blindleistung an Bus 7 in Var
bus_8_q_mvar	Blindleistung an Bus 8 in Var
bus_9_q_mvar	Blindleistung an Bus 9 in Var
bus_10_q_mvar	Blindleistung an Bus 10 in Var
bus_11_q_mvar	Blindleistung an Bus 11 in Var
bus_12_q_mvar	Blindleistung an Bus 12 in Var
bus_13_q_mvar	Blindleistung an Bus 13 in Var
bus_14_q_mvar	Blindleistung an Bus 14 in Var
switch_0_status	Status von Switch 0: offen oder geschlossen
switch_1_status	Status von Switch 1: offen oder geschlossen
switch_2_status	Status von Switch 2: offen oder geschlossen
switch_3_status	Status von Switch 3: offen oder geschlossen
switch_4_status	Status von Switch 4: offen oder geschlossen
switch_5_status	Status von Switch 5: offen oder geschlossen
switch_6_status	Status von Switch 6: offen oder geschlossen
switch_7_status	Status von Switch 7: offen oder geschlossen

Tabelle C.1.: *Messwerte der Simulation in PandaPower für einen Zeitpunkt t*

<b>Bezeichnung</b>	<b>Beschreibung</b>
line_1_1_loading	Prozentuale Last der Leitung zwischen Bus 1 und 1
line_2_3_loading	Prozentuale Last der Leitung zwischen Bus 2 und 3
line_3_4_loading	Prozentuale Last der Leitung zwischen Bus 3 und 4
line_4_5_loading	Prozentuale Last der Leitung zwischen Bus 4 und 5
line_5_6_loading	Prozentuale Last der Leitung zwischen Bus 5 und 6
line_7_8_loading	Prozentuale Last der Leitung zwischen Bus 7 und 8
line_8_9_loading	Prozentuale Last der Leitung zwischen Bus 8 und 9
line_9_10_loading	Prozentuale Last der Leitung zwischen Bus 9 und 10
line_10_11_loading	Prozentuale Last der Leitung zwischen Bus 10 und 11
line_3_8_loading	Prozentuale Last der Leitung zwischen Bus 3 und 8
line_12_13_loading	Prozentuale Last der Leitung zwischen Bus 12 und 13
line_13_14_loading	Prozentuale Last der Leitung zwischen Bus 13 und 14
line_14_8_loading	Prozentuale Last der Leitung zwischen Bus 14 und 8
line_6_7_loading	Prozentuale Last der Leitung zwischen Bus 6 und 7
line_11_14_loading	Prozentuale Last der Leitung zwischen Bus 11 und 14
trafo_0_loading	Prozentuale Last des Trafos 0
trafo_1_loading	Prozentuale Last des Trafos 1

Tabelle C.2.: *Reduzierte Anzahl der verwendeten Features für das Training des AutoEncoders*

optimizer	FAS	meta_lr	It.	k	num_train	num_eval	Acc. $\mathbf{X}_{test}$	Acc. $\mathbf{X}_{drifted,ano}$
Adam	15	0.10	100	5	32	32	99.74	90.32
SGD	15	0.01	100	20	128	32	99.73	90.30
SGD	15	0.10	300	20	32	16	99.70	90.27
SGD	10	0.10	100	20	128	32	99.71	90.29
SGD	10	0.01	100	5	32	32	99.74	90.31
Adam	5	0.01	300	10	32	16	99.68	90.25
Adam	15	0.00	300	5	32	32	99.69	90.26
SGD	10	0.00	300	10	128	32	99.70	90.28
SGD	10	0.01	300	20	128	16	99.74	90.31
Adam	15	0.00	100	5	128	16	99.58	90.11
SGD	10	0.01	500	10	128	32	99.69	90.27
Adam	5	0.00	500	20	32	16	99.55	90.07
SGD	15	0.00	500	20	32	16	99.65	90.17
Adam	15	0.10	100	10	64	32	99.73	90.29

Fortsetzung auf nächster Seite

optimizer	FAS	meta_lr	It.	k	num_train	num_eval	Acc. $X_{test}$	Acc. $X_{drifted,ano}$
SGD	5	0.01	100	10	128	16	99.69	90.26
SGD	10	0.10	100	5	32	16	99.72	90.29
Adam	5	0.00	100	20	64	16	99.71	90.28
SGD	15	0.10	100	5	32	16	99.70	90.28
Adam	10	0.01	500	20	128	32	99.71	90.27
SGD	15	0.01	300	5	64	16	99.69	90.26
SGD	10	0.01	500	20	128	32	99.73	90.30
SGD	15	0.10	300	5	32	16	99.61	90.15
Adam	5	0.10	500	5	64	32	99.74	90.31
SGD	5	0.00	300	10	64	32	99.71	90.26
SGD	5	0.10	100	20	32	32	99.73	90.27
SGD	10	0.01	100	10	32	32	99.72	90.27
SGD	5	0.10	500	5	64	32	99.71	90.29
Adam	15	0.10	500	10	32	16	99.69	90.20
Adam	5	0.10	500	20	64	32	99.69	90.21
Adam	10	0.10	300	10	64	32	99.73	90.29
SGD	5	0.00	500	20	128	16	99.73	90.29
Adam	15	0.01	300	5	64	32	99.69	90.21
SGD	10	0.10	100	5	128	32	99.74	90.32
SGD	10	0.10	500	10	128	32	99.60	90.12
SGD	5	0.10	300	10	32	16	99.73	90.30
SGD	15	0.10	500	5	64	16	99.74	90.29
SGD	15	0.00	100	20	32	16	99.69	90.27
Adam	5	0.01	100	10	32	16	99.75	90.32
SGD	5	0.01	300	5	128	32	99.76	90.33
SGD	15	0.01	100	20	32	16	99.73	90.30
SGD	15	0.00	500	10	64	16	99.68	90.25
Adam	10	0.01	100	10	128	16	99.70	90.26
SGD	5	0.00	100	5	128	16	99.70	90.23
Adam	5	0.10	100	20	128	32	99.69	90.27
Adam	10	0.01	100	10	32	16	99.67	90.25
Adam	10	0.10	300	5	32	16	99.69	90.27
SGD	15	0.10	100	20	128	32	99.71	90.29
SGD	5	0.00	100	5	64	16	99.72	90.29
SGD	15	0.01	300	10	128	32	99.71	90.25
Adam	15	0.01	500	10	64	16	99.69	90.26
Adam	10	0.01	500	5	128	16	99.69	90.22
SGD	15	0.10	500	5	128	32	99.73	90.30
Adam	10	0.10	100	10	64	16	99.62	90.14
SGD	10	0.00	500	5	32	16	99.70	90.27
Adam	5	0.00	100	5	32	32	99.74	90.32
Adam	5	0.01	500	20	32	16	99.74	90.28
Adam	5	0.00	100	10	32	32	99.70	90.28

Fortsetzung auf nächster Seite

optimizer	FAS	meta_lr	It.	k	num_train	num_eval	Acc. $\mathbf{X}_{test}$	Acc. $\mathbf{X}_{drifted,ano}$
SGD	10	0.01	100	10	128	32	99.70	90.22
SGD	15	0.00	300	20	32	32	99.69	90.26
SGD	15	0.01	100	5	128	16	99.69	90.25
SGD	5	0.10	100	20	32	16	99.68	90.21
Adam	5	0.01	100	5	128	32	99.72	90.29
Adam	10	0.01	500	10	64	32	99.77	90.34
Adam	15	0.00	300	10	64	16	99.73	90.25
Adam	10	0.10	500	5	64	16	99.36	89.92
Adam	10	0.10	100	5	32	16	99.72	90.29
SGD	15	0.10	500	10	128	16	99.71	90.28
SGD	10	0.00	500	10	128	16	99.71	90.28
SGD	5	0.01	500	20	128	32	99.72	90.29
SGD	15	0.10	100	10	64	16	99.68	90.22
Adam	15	0.10	300	20	32	32	99.67	90.24
Adam	15	0.00	300	5	64	32	99.71	90.23
Adam	15	0.00	300	20	32	16	99.73	90.31
SGD	10	0.10	500	20	128	16	99.70	90.23
Adam	5	0.10	500	5	64	16	99.61	90.13
Adam	10	0.10	300	5	32	32	99.70	90.27
Adam	15	0.10	500	20	32	16	99.67	90.19
SGD	15	0.01	500	10	32	16	99.70	90.27
Adam	5	0.10	100	10	64	32	99.74	90.31
SGD	10	0.01	100	20	32	16	99.71	90.25
SGD	10	0.00	500	10	128	32	99.74	90.31
Adam	15	0.01	100	20	32	16	99.67	90.24
SGD	5	0.10	300	10	32	32	99.72	90.25
SGD	5	0.10	300	10	128	16	99.53	90.12
Adam	5	0.00	500	10	32	32	99.74	90.31
SGD	15	0.00	300	20	32	16	99.69	90.26
SGD	15	0.00	100	20	128	32	99.68	90.21
Adam	15	0.10	100	5	32	16	99.63	90.15
Adam	15	0.00	100	20	32	16	99.71	90.28
SGD	10	0.00	100	10	32	32	99.76	90.33
Adam	15	0.10	100	10	32	32	99.68	90.22
Adam	10	0.00	100	10	128	32	99.62	90.19
SGD	10	0.10	500	5	64	16	99.75	90.33
SGD	15	0.10	500	5	128	16	99.72	90.29
SGD	10	0.01	500	20	64	32	99.70	90.28
SGD	15	0.00	500	5	32	16	99.66	90.23
SGD	15	0.10	100	5	128	16	99.69	90.26

Tabelle C.3.: Vollständiger teilstatistischer Versuchsplan  $tVP_{M_1}^I$

k_train	optimizer	num_iter	lr	filter_cde	model
20	Adam	128	0.001	(1, 3)	6
5	Adam	64	0.100	(1, 3)	5
20	Adam	1	0.100	(1, 2)	3
20	Adam	1	0.100	(1,)	3
10	Adam	64	0.010	(1, 3)	3
10	SGD	128	0.100	(2, 3)	2
20	Adam	1	0.100	(1, 2, 3)	5
5	SGD	128	0.010	(1,)	3
10	Adam	128	0.100	(3,)	6
5	Adam	64	0.100	(2, 3)	6
5	Adam	64	0.010	(2, 3)	5
10	SGD	1	0.001	(1, 2, 3)	4
10	SGD	1	0.100	(3,)	2
20	SGD	1	0.001	(1, 2)	1
20	Adam	128	0.001	(1, 3)	4
5	SGD	1	0.100	(2, 3)	1
10	SGD	128	0.100	(2,)	4
5	SGD	128	0.100	(1, 2, 3)	5
20	SGD	64	0.001	(1,)	4
10	Adam	1	0.100	(1,)	4
5	SGD	1	0.100	(3,)	2
10	SGD	64	0.010	(1, 2, 3)	6
20	SGD	1	0.001	(1,)	2
20	Adam	64	0.001	(1,)	3
10	SGD	128	0.001	(1, 2)	4
5	Adam	1	0.100	(3,)	4
10	SGD	1	0.100	(1, 3)	3
10	SGD	1	0.010	(2, 3)	4
20	SGD	64	0.100	(1, 3)	5
10	Adam	64	0.010	(2, 3)	4
20	Adam	64	0.001	(3,)	5
10	SGD	64	0.010	(1,)	3
10	Adam	1	0.001	(1,)	3
10	Adam	64	0.001	(1,)	3
10	SGD	128	0.001	(2, 3)	3
20	Adam	1	0.010	(2, 3)	6
5	Adam	64	0.010	(1, 2)	4
10	SGD	128	0.100	(1, 2)	1
5	SGD	128	0.010	(2, 3)	1
20	Adam	128	0.001	(2, 3)	4
20	SGD	128	0.010	(2, 3)	6
5	Adam	1	0.010	(1, 2)	2

Fortsetzung auf nächster Seite

k_train	optimizer	num_iter	lr	filter_cde	model
5	Adam	128	0.010	(2,)	2
10	Adam	128	0.010	(1, 2, 3)	6
5	SGD	128	0.010	(2,)	1
20	SGD	64	0.010	(2,)	3
5	SGD	1	0.001	(1, 2)	6
10	SGD	1	0.010	(1,)	4
5	Adam	128	0.010	(1, 2)	1
5	Adam	128	0.100	(1, 3)	1
10	SGD	64	0.001	(2,)	1
10	Adam	128	0.001	(1, 2)	4
10	SGD	128	0.010	(1, 2, 3)	1
5	Adam	1	0.100	(2,)	1
10	SGD	64	0.010	(1, 2, 3)	1
10	Adam	64	0.001	(2,)	5
5	SGD	1	0.001	(3,)	4
20	Adam	128	0.001	(2, 3)	6
20	Adam	128	0.001	(1, 2, 3)	6
5	SGD	128	0.001	(2, 3)	5
20	Adam	128	0.100	(2,)	6
20	SGD	128	0.001	(1, 3)	1
20	SGD	64	0.010	(1, 2, 3)	1
5	SGD	64	0.001	(1, 2, 3)	2
5	SGD	64	0.001	(1, 3)	5
5	Adam	1	0.010	(1,)	3
20	Adam	64	0.100	(1, 2, 3)	6
10	SGD	128	0.100	(1,)	5
20	Adam	128	0.100	(2, 3)	1
10	SGD	1	0.010	(2,)	3
10	Adam	64	0.001	(1, 2)	1
20	SGD	64	0.001	(2, 3)	2
10	Adam	128	0.010	(3,)	2
5	Adam	64	0.100	(1, 2)	2
20	Adam	64	0.001	(2,)	3
10	Adam	128	0.001	(3,)	3
20	Adam	64	0.100	(2, 3)	4
20	Adam	1	0.010	(1, 2, 3)	3
10	Adam	64	0.001	(1, 2, 3)	6
5	Adam	128	0.100	(2, 3)	1
5	SGD	1	0.001	(1, 2, 3)	4
5	Adam	64	0.001	(2,)	4
20	SGD	1	0.001	(2, 3)	6
5	Adam	128	0.001	(1, 2, 3)	5
10	Adam	64	0.100	(1, 2, 3)	3

Fortsetzung auf nächster Seite

k_train	optimizer	num_iter	lr	filter_cde	model
5	SGD	1	0.010	(2,)	6
10	SGD	1	0.001	(1, 2)	6
20	Adam	1	0.100	(1, 2, 3)	1
20	Adam	64	0.010	(2,)	4
20	Adam	1	0.001	(2, 3)	5
5	Adam	128	0.001	(1,)	3
5	SGD	64	0.100	(1, 2, 3)	5
5	SGD	64	0.100	(1, 3)	5
10	SGD	1	0.010	(3,)	1
20	Adam	128	0.001	(2,)	2
10	SGD	128	0.100	(2, 3)	6
5	SGD	128	0.001	(1, 2, 3)	2
20	SGD	1	0.100	(2, 3)	2
20	SGD	1	0.010	(1,)	6
10	Adam	64	0.100	(1,)	1
10	Adam	1	0.001	(3,)	5
20	Adam	1	0.001	(1, 2)	1
5	Adam	1	0.001	(2,)	1
10	SGD	128	0.001	(2,)	1
20	SGD	1	0.001	(1, 2, 3)	1
5	Adam	64	0.001	(1,)	5
5	SGD	1	0.100	(1,)	6
5	SGD	64	0.001	(2, 3)	3
10	SGD	64	0.010	(1, 2)	1
20	Adam	128	0.100	(1, 3)	2
5	Adam	1	0.001	(1,)	6
20	Adam	64	0.010	(1, 3)	3
5	SGD	64	0.100	(2,)	4
10	Adam	64	0.010	(1, 2)	2
5	SGD	64	0.010	(1, 2)	2
20	SGD	128	0.001	(2,)	2
20	SGD	128	0.010	(3,)	5
20	Adam	128	0.100	(2, 3)	5
20	Adam	128	0.001	(3,)	5
5	Adam	128	0.001	(1, 3)	6
10	SGD	64	0.010	(2, 3)	5
10	SGD	128	0.010	(2, 3)	2
5	Adam	1	0.010	(2,)	2
10	Adam	64	0.001	(2, 3)	4
5	SGD	128	0.001	(1, 2)	4
20	Adam	64	0.001	(1, 2)	3
10	SGD	1	0.100	(1,)	3
20	SGD	1	0.010	(1,)	1

Fortsetzung auf nächster Seite

k_train	optimizer	num_iter	lr	filter_cde	model
5	SGD	64	0.100	(1,)	1
10	Adam	128	0.001	(1, 2, 3)	3
5	SGD	1	0.001	(1, 3)	1
10	Adam	1	0.010	(2,)	2
20	Adam	64	0.100	(1, 3)	1
20	SGD	64	0.001	(3,)	6
10	SGD	1	0.001	(2,)	4
20	SGD	1	0.010	(1, 2, 3)	4
20	Adam	1	0.100	(2,)	2
5	Adam	1	0.010	(1, 3)	1
10	Adam	1	0.001	(2,)	4
10	Adam	128	0.010	(1, 2)	3
5	Adam	128	0.001	(1, 2, 3)	1
10	SGD	64	0.100	(1, 2)	1
20	Adam	128	0.010	(2, 3)	6
20	SGD	128	0.100	(1,)	2
10	Adam	1	0.100	(2, 3)	3
5	SGD	128	0.001	(1, 3)	2
10	Adam	64	0.010	(1,)	2
10	Adam	128	0.100	(1, 2, 3)	1
5	Adam	1	0.010	(1, 2, 3)	2
10	SGD	1	0.001	(1,)	6
10	SGD	64	0.100	(1,)	3
5	Adam	128	0.001	(3,)	1
5	SGD	1	0.100	(2, 3)	6
5	SGD	1	0.010	(2, 3)	5
5	Adam	64	0.001	(1,)	1
10	Adam	64	0.100	(2, 3)	6
10	Adam	128	0.100	(1, 2)	3
5	Adam	64	0.100	(1, 2)	6
5	Adam	128	0.010	(3,)	5
5	Adam	128	0.001	(3,)	5
20	SGD	64	0.001	(2,)	4
10	Adam	1	0.001	(1, 3)	5
5	SGD	64	0.100	(1, 3)	3
10	SGD	128	0.001	(1, 2, 3)	5
10	SGD	128	0.100	(3,)	1
20	SGD	1	0.001	(3,)	3
20	Adam	64	0.001	(3,)	2
5	Adam	128	0.001	(1, 2, 3)	2
10	Adam	64	0.001	(2, 3)	3
10	Adam	1	0.001	(2,)	1
5	Adam	1	0.001	(1, 2, 3)	6

Fortsetzung auf nächster Seite

k_train	optimizer	num_iter	lr	filter_cde	model
10	Adam	128	0.001	(1,)	2
5	SGD	64	0.100	(1, 2)	4
20	SGD	128	0.010	(1, 2)	3
5	Adam	128	0.010	(1,)	5
10	SGD	1	0.010	(3,)	6
20	SGD	1	0.001	(2, 3)	4
5	Adam	128	0.010	(1,)	2
10	SGD	128	0.010	(2,)	5
10	SGD	128	0.100	(2, 3)	3
5	Adam	128	0.010	(1, 3)	6
10	Adam	64	0.100	(3,)	4
20	Adam	64	0.010	(1,)	4
10	SGD	128	0.001	(1, 3)	3
20	SGD	128	0.100	(2,)	4
10	Adam	64	0.010	(1, 2, 3)	6
20	SGD	128	0.100	(1, 3)	6
10	Adam	1	0.100	(3,)	3
20	SGD	1	0.100	(1, 2)	5
10	Adam	64	0.001	(2, 3)	2
10	Adam	128	0.100	(1, 2, 3)	6
5	SGD	1	0.010	(2,)	5
5	Adam	1	0.100	(2,)	6
5	Adam	128	0.001	(1, 2)	3
20	SGD	128	0.100	(1, 2, 3)	1
10	SGD	128	0.001	(1, 2, 3)	3
10	SGD	64	0.010	(1,)	6
20	SGD	1	0.100	(1,)	2
5	SGD	128	0.100	(1, 2)	5
10	SGD	128	0.100	(3,)	4
5	Adam	1	0.001	(3,)	5
10	Adam	1	0.010	(1, 2, 3)	4
5	SGD	128	0.010	(3,)	3
20	Adam	64	0.010	(2, 3)	2
10	SGD	64	0.100	(2, 3)	5
10	Adam	64	0.100	(1, 2, 3)	6
20	SGD	128	0.010	(1,)	1
20	SGD	1	0.100	(1, 3)	4
20	Adam	1	0.100	(3,)	2
5	Adam	64	0.001	(3,)	3
20	SGD	128	0.100	(1, 2, 3)	4
5	Adam	1	0.001	(1, 2, 3)	4
20	SGD	128	0.010	(2,)	6
20	Adam	1	0.010	(3,)	6

Fortsetzung auf nächster Seite

k_train	optimizer	num_iter	lr	filter_cde	model
5	SGD	128	0.001	(2, 3)	6
5	Adam	64	0.001	(1, 3)	6
10	SGD	64	0.100	(1, 3)	1
5	SGD	64	0.001	(1, 3)	1
5	Adam	128	0.010	(2, 3)	3
5	SGD	1	0.010	(1,)	6
10	SGD	1	0.001	(3,)	1
10	Adam	1	0.100	(1, 2)	1
10	Adam	128	0.010	(1, 2, 3)	5
5	SGD	64	0.010	(1, 3)	4
20	Adam	128	0.010	(1, 2)	2
10	Adam	64	0.010	(1,)	6

Tabelle C.4.: *Vollständiger teilst faktorieller Versuchsplan tVP<sub>M<sub>1</sub></sub><sup>II</sup>*

k_train	optimizer	num_iter	lr	filter_cde
5	Adam	128	0.010	(1,)
20	Adam	1	0.010	(1, 2, 3)
20	SGD	128	0.010	(2, 3)
5	SGD	1	0.100	(1, 3)
10	Adam	1	0.010	(1,)
5	SGD	64	0.010	(1, 3)
10	Adam	128	0.001	(1, 2, 3)
20	SGD	64	0.010	(1, 3)
10	SGD	128	0.010	(1, 2)
20	Adam	64	0.001	(1,)
20	Adam	64	0.100	(2,)
10	SGD	64	0.001	(1, 2)
20	SGD	128	0.001	(2, 3)
20	Adam	64	0.100	(1, 3)
10	SGD	64	0.010	(2, 3)
10	SGD	1	0.001	(2,)
5	SGD	64	0.001	(1, 3)
10	SGD	128	0.100	(1, 2)
5	SGD	1	0.010	(1, 2)
10	SGD	1	0.010	(1, 2)
20	Adam	1	0.100	(1, 3)
20	Adam	128	0.100	(1, 2)
10	Adam	64	0.010	(1, 2, 3)
20	SGD	128	0.100	(1, 2)
10	Adam	64	0.001	(1, 2)

Fortsetzung auf nächster Seite

k_train	optimizer	num_iter	lr	filter_cde
10	SGD	128	0.010	(1,)
5	SGD	1	0.100	(1,)
5	Adam	1	0.100	(2, 3)
5	SGD	128	0.010	(2, 3)
20	Adam	1	0.001	(1, 2)
10	Adam	64	0.100	(3,)
5	Adam	64	0.001	(3,)
10	Adam	64	0.100	(1,)
10	Adam	128	0.100	(1, 2)
5	SGD	1	0.010	(1,)
20	SGD	1	0.001	(1, 2)
10	Adam	128	0.001	(2, 3)

Tabelle C.5.: *Vollständiger teilst faktorieller Versuchsplan tVP<sub>M<sub>2</sub></sub><sup>II</sup>*

---

## Glossar

Nachfolgend sind die wesentlichen Begriffe dieser Arbeit zusammengefasst und erläutert. Eine ausführliche Erklärung findet sich jeweils in den einführenden Abschnitten sowie der jeweils darin angegebenen Literatur. Das im Folgenden im Rahmen der Erläuterung verwendete Symbol  $\sim$  bezieht sich jeweils auf den im Einzelnen vorgestellten Begriff, das Symbol  $\uparrow$  verweist auf einen ebenfalls innerhalb dieses Glossars erklärten Begriff.

**Accuracy** Die  $\sim$  ist eine Kennzahl, welche anhand der Werte einer  $\uparrow$ Confusion Matrix bestimmt werden kann. Die  $\sim$  gibt dabei den prozentualen Anteil aller Objekte, die richtig klassifiziert, wurden an.

**Anomalie**  $\sim$  ist ein Fremdwort aus dem Griechischen und bedeutet „Unregelmäßigkeit“. Im Kontext des  $\uparrow$ Machine Learnings wird eine  $\sim$  definiert als Beobachtung, die so sehr von den anderen Beobachtungen abweicht, dass sie den Verdacht aufkommen lässt, dass sie durch einen anderen Mechanismus hervorgerufen wurde.

**Anomalieerkennung** Im Kontext des  $\uparrow$ Machine Learnings ist die  $\sim$  ein Teilgebiet mit der Absicht der Identifizierung von  $\uparrow$ Anomalien. Dabei kommen verschiedene Methoden des  $\uparrow$ Machine Learnings, wie z.B.  $\uparrow$ künstliche neuronale Netze, zur Anwendung.

**Autoencoder**  $\sim$  sind eine Art von  $\uparrow$ künstlichen neuronalen Netzen, die darauf trainiert werden ihren *Input* als *Output* zu reproduzieren. Ein  $\sim$  besteht dabei immer aus einem *Encoder* und einem *Decoder*. Der *Encoder* codiert dabei zunächst den *Input*  $x$  zu einem Code  $h$ . Zur Erzeugung des *Outputs* dekodiert der *Decoder* den Code  $h$  anschließend.  $\sim$  werden aufgrund ihrer Eigenschaften häufig für die  $\uparrow$ Anomalieerkennung verwendet.

**Concept Drift** Als  $\sim$  wird die Auswirkung der Veränderung der Verteilungen in den Daten auf ein Modell bezeichnet.

**Confusion Matrix** Im Bereich des  $\uparrow$ Machine Learnings ist die  $\sim$ , auch als Fehlermatrix bezeichnet, ein spezielles Tabellenlayout, welches die Visualisierung der Leistung eines Modells sowie die Berechnung von Kennzahlen ermöglicht.

**Feature** Im Kontext des  $\uparrow$ Machine Learnings ist ein  $\sim$  eine individuell messbare Eigenschaft oder Charakteristik eines beobachteten Phänomens. Bei der Darstellung eines Datensatzes als Tabelle entspricht ein  $\sim$  einer Spalte.

**Few-shot Learning**  $\sim$  ist ein Teilgebiet des  $\uparrow$ Machine Learnings, welches sich mit dem Lernen anhand weniger  $\uparrow$ Samples beschäftigt.

**Künstliche neuronale Netze**  $\sim$  sind informationsverarbeitende Modelle, deren Struktur und Funktionsprinzipien vom Nervensystem und dem Gehirn von Tieren und Menschen inspiriert sind. Sie bestehen aus einer Vielzahl von relativ einfachen Einheiten, den Neuronen, die parallel Arbeiten und kommunizieren, indem sie Informationen in Form von Aktivierungssignalen entlang gerichteter Verbindungen zueinander senden.

**Machine Learning**  $\sim$  ist ein Teilgebiet der künstlichen Intelligenz mit dem Ziel der Generierung von Wissen aus Erfahrung. Algorithmen des  $\sim$  bauen dafür ein mathematisches Modell auf, das auf Beispieldaten, so genannten „Trainingsdaten“, basiert, um Vorhersagen oder Entscheidungen zu treffen, ohne explizit darauf programmiert worden zu sein.

**Meta-Learning** ~ ist ein Teilgebiet des ↑Machine Learnings, in dem verschiedene Ansätze entwickelt wurden, die sich mit dem Problem des ↑few-shot Learnings befassen.

**Precision** Die ~ ist eine Kennzahl, welche anhand der Werte einer ↑Confusion Matrix bestimmt werden kann. Sie gibt den prozentualen Anteil der korrekt als positiv klassifizierten Ergebnisse an der Gesamtheit der als positiv klassifizierten Ergebnisse an. Im Kontext der ↑Anomalieerkennung bestimmt die ~ beispielsweise den Anteil der korrekterweise als ↑Anomalie klassifizierten Ergebnisse an der Gesamtheit aller als ↑Anomalie klassifizierten Objekte.

**Sample** Ein ~ ist die Sammlung aller ↑Features zu einem bestimmten Zeitpunkt. Bei der Betrachtung eines Datensatzes entspricht ein ~ dabei einer Zeile.

**Specificity** Die ~ ist eine Kennzahl, welche anhand der Werte einer ↑Confusion Matrix bestimmt werden kann. Die ~ gibt dabei den prozentualen Anteil der korrekt als negativ klassifizierten Objekte an der Gesamtheit der in Wirklichkeit negativen Objekte an. Für die ↑Anomalieerkennung entspricht die ~ dem Anteil der korrekt als keine ↑Anomalie klassifizierten Objekte an der Gesamtheit aller Objekte ohne ↑Anomalie.

**Sensitivity** Die ~ ist eine Kennzahl, welche anhand der Werte einer ↑Confusion Matrix bestimmt werden kann. Die ~ gibt dabei den prozentualen Anteil der korrekt als positiv klassifizierten Objekte an der Gesamtheit der tatsächlich positiven Objekte an. Im Kontext der ↑Anomalieerkennung entspricht die ~ dem Anteil der als ↑Anomalie klassifizierten Objekte im Verhältnis zu allen positiven Objekten.

**Statistische Versuchsplanung** Bei der ~, auch als *Design of Experiments* bezeichnet, handelt es sich um eine statistische Methode bei der verschiedene Faktoren gleichzeitig geändert werden können, um den Reaktionsraum auf optimale Werte zu untersuchen. Ein Hauptgegenstand der ~ ist dabei die Erstellung von ↑Versuchsplänen.

**Versuchsplan** Ein ~ ist die Grundlage einer wissenschaftlichen Untersuchung und beschreibt, wie eine empirische Fragestellung untersucht werden soll.

## Abkürzungen

AE	Autoencoder
ANN	Artifical Neural Network
API	Application Programming Interface
AUC	Area Under Curve
BDEW	Bundesverband der Energie- und Wasserwirtschaft e.V.
BNetzA	Bundesnetzagentur
CIGRE	Conseil International des Grands Réseaux Électriques
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DAD	Deep Anomaly Detection
DER	Distributed Energy Ressources
DL	Deep Learning
EKG	Elektrokardiogramm
FOMAML	First Order Model Agnostic Meta-Learning
FNR	False Negative Rate
FPR	False Positive Rate
FR	Functional Requirement bzw. funktionale Anforderung
GPU	Graphic Processing Unit
GRU	Gated Recurrent Unit
IDE	Integrierte Entwicklungsumgebung
IDS	Intrusion Detection System
IED	Intelligent Eletronic Device
KI	Künstliche Intelligenz
LSTM	Long Short-Term Memory Network
MAE	Mean Absolute Error
MAML	Model Agnostic Meta-Learning
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NFR	Non-functional Requirement bzw. nicht funktionale Anforderung
PCA	Principal Component Analysis
PMU	Phasor Measurement Units
PRC	Precision-Recall Curve
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic Curve
SCADA	Supervisory Control and Data Acquisition
SEP	Standardeinspeiseprofil
SGD	Stochastic Gradient Descent
SL	Sequence Learning
SLP	Standardlastprofil
SOM	Self-Organizing Map
SVM	Support Vector Machine
TPR	True Positive Rate

VAE	Variational Autoencoder
VDEW	Verband der Elektrizitätswirtschaft

# Symbolverzeichnis

## Machine Learning

$f$	Eine durch ein Modell gelernte Funktion
$\mathbf{x}$	Ein Sample in Form eines Vektors $\mathbf{x} \in \mathbb{R}^n$
$x_i$	Ein einzelnes Feature aus dem Vektor $\mathbf{x}$
$\mathbf{X}$	Designmatrix in Form einer Matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$
$X_{i,j}$	Ein bestimmtes Feature eines bestimmten Samples der Matrix $\mathbf{X}$
$\mathbf{y}$	Ein Vektor von <i>Labels</i> der Form $\mathbf{y} \in \mathbb{R}^m$
$y_i$	Ein einzelnes Label aus dem Vektor $\mathbf{y}$
$\hat{y}$	Ein von einem Modell vorhergesagter Wert
$\varphi$	Aktivierungsfunktion eines Neurons
$\mathbf{w}_j^{(l)}$	Gewichtsvektor für das $j$ -te Neuron in der Schicht $l$
$w_{i,j}^{(l)}$	Das Gewicht des $i$ -ten Neurons in der Schicht $l$ zum $j$ -ten Neuron der Schicht $l + 1$
$z_j^{(l)}$	Gewichteten Eingaben des $j$ -ten Neurons im Layer $l$
$a_j^{(l)}$	Ausgabe der Aktivierungsfunktion des $j$ -ten Neurons in der $l$ -ten Schicht
$\theta$	Gewichte eines <i>Artificial Neural Networks</i>
$\theta_{f_{MAML}}$	Gewichte eines Modells nach dem Training mittels eines <i>Meta-Learning</i> -Algorithmus
$\epsilon$	<i>Reconstruction Error</i> eines <i>Autoencoders</i>
$\mathcal{L}$	<i>Loss</i> Funktion eines ANN
$\mathcal{D}$	Ein Datensatz
$\lambda$	Schwellwert für ein Modell zur Anomalieerkennung
$S(\lambda)$	Von einem Modell $f$ mit einem Schwellwert $\lambda$ als Anomalie gekennzeichnete Datensätze
$\alpha$	Hyperparameter des <i>Inner Loop Learnings</i> von MAML
$\beta$	Hyperparameter des <i>Outer Loop Learnings</i> von MAML
$\eta$	Lernrate von <i>Backpropagation</i>
$\delta_j$	Fehlersignal eines Neurons $j$
$\mathbb{E}[\mathcal{L}(f_\theta)]$	Erwartungswert $\mathbb{E}$ der <i>Loss Function</i> $\mathcal{L}(f_\theta)$
$\mu$	Erwartungswert bzw. Center einer Normalverteilung
$\sigma$	Standardabweichung einer Normalverteilung
$\mathcal{T}$	Ein Lerntask im Rahmen des <i>Meta-Learnings</i>
$k$	Anzahl der <i>Samples</i> die während des <i>Meta-Learnings</i> pro Lerntask $\mathcal{T}_i$ verwendet werden

## Physikalische Größen

$f_{Hz}$	Netzfrequenz eines Stromnetzes, definiert durch $f_{Hz} = \frac{1}{T}$
$\phi(t)$	Phasenwinkel im Zeitpunkt $t$ , Maßeinheit in Grad: $[\phi(t)] = rad$
$U$	Stromspannung, Maßeinheit in Volt: $[U] = V$
$I$	Stromstärke, Maßeinheit in Ampere: $[I] = A$
$Z$	Impedanz, Maßeinheit in Ohm $[Z] = \Omega$
$P$	Wirkleistung, Maßeinheit in Watt $[P] = W$
$Q$	Blindleistung, Maßeinheit in Voltampere Reaktiv: $[Q] = Var$
$S$	Scheinleistung, Maßeinheit in Voltampere: $[S] = VA$



## Abbildungsverzeichnis

1.1. (a) Unsupervised Anomalieerkennung in multivariaten Zeitreihen für Smart Grids. (b) Unterschiedliche <i>System Signature Matrices</i> für normalen und abnormalen Perioden. [Zha+18]	2
2.1. Skizze eines biologischen Neurons mit Beschriftungen der Bestandteile [Kri07]	11
2.2. Darstellung eines künstlichen Neurons mit seinen Elementen (eigene Darstellung nach [Wik19])	12
2.3. Perzeptron für den log. Term $(x_1 \wedge \neg x_2) \vee (x_1 \wedge x_3) \vee (\neg x_2 \wedge x_3)$ [Kru+13]	12
2.4. XNOR-Funktion: Es gibt keine lineare Sparierbarkeit der weißen und schwarzen Punkte [Kru+13]	13
2.5. Lösen der XNOR Funktion durch Kombination mehrerer Neuronen [Gér18]	14
2.6. Multi-Layer Perceptron mit einem Input, Hidden und Output Layer (eigene Abbildung nach [Haz+11])	14
2.7. Verschiedene Aktivierungsfunktionen: (a) Sigmoid, (b) Tangens Hyperbolicus, (c) Schwellwertfunktion, (d) ReLU	17
2.8. Gradientenabstieg zur Minimierung einer Kostenfunktion $\mathcal{L}(\theta)$ (eigene Darstellung nach [Mur12])	18
2.9. Abstrakte Funktionsweise von <i>Backpropagation</i> (eigene Darstellung nach [GBC16; Mur12])	20
2.10. Arten von Änderungen in Datenströmen: (a) plötzlich, (b) schrittweise, (c) inkrementell, (d) wiederkehrend (eigene Abbildung nach [CW17])	21
2.11. Beispiel des <i>Concept Drifts</i> für eine Klassifikation (eigene Abbildung nach [Chi19])	22
2.12. Schema eines Autoencoders (eigene Abbildung nach [GBC16])	23
2.13. Ein einfaches Beispiel für Anomalien in einem zweidimensionalen Datensatz (eigene Darstellung nach [CBK09])	26
2.14. Anomalie in (a) Daten ohne Rauschen und in (b) Daten mit Rauschen (eigene Darstellung nach [AS17])	27
2.15. Grad der Anomalie von Daten nach [AS17]	28
2.16. Beispiel für eine kontextbezogene Anomalie nach [AS17]	29
2.17. Beispiel einer kollektiven Anomalie in Form einer Extrasystole (vorzeitige Kontraktion des Vorhofes) in einem menschlichen Elektrokardiogramm [CBK09]	29
2.18. Problemcharakteristik der Anomalieerkennung nach [CBK09]	32
2.19. Beispiel der Verwendung eines <i>Support Sets</i> $\mathcal{S}$ und eines <i>Query Sets</i> $\mathcal{Q}$ für das <i>Meta-Learning</i> anhand einer Klassifikation von Bildern [RL17]	34
2.20. Diagramm der abstrakten Funktionsweise des MAML Algorithmus [FAL17]	35
2.21. Instabilität während des Trainings mittels MAML gegenüber MAML++ [AES18]	38
2.22. Der <i>Reptile</i> Algorithmus aktualisiert die Modellparameter so, dass sie möglichst nahe an den Oberflächen aller Aufgaben liegen [NAS18]	41
2.23. <i>Reptile</i> vs. FOMAML im Durchlauf der <i>Meta-Optimierung</i> (eigene Darstellung nach [Wen18])	42

2.24. Beispiel einer <i>Precision-Recall Curve</i> [Agg17] . . . . .	44
2.25. Beispiel einer <i>Receiver Operating Characteristic Curve</i> [Agg17] . . . . .	45
3.1. Aktivitätszyklus beim Experimentieren in der Informatik (eigene Darstellung nach [Ste99]) . . . . .	50
3.2. Aktivitätsdiagramm des gesamten Ablaufes inkl. Konzeptionierung, Vorstudie und Experiment (eigene Darstellung) . . . . .	51
3.3. Versuchsaufbau zur Evaluierung des <i>Meta-Learning</i> -Ansatzes gegenüber keinem <i>Meta-Learning</i> (eigene Darstellung) . . . . .	52
3.4. Bestimmung der Gewichte der fünf Kriterien als Ergebnis einer Nutzwertanalyse (eigene Darstellung) . . . . .	54
3.5. Konzept des Datengenerators mit allen Komponenten sowie dem Input und Output (eigene Darstellung) . . . . .	58
3.6. Verwendung von <i>PyTorch</i> und <i>TensorFlow</i> in wissenschaftlichen Publikationen [He19]	61
4.1. Übersicht des Szenarios zur Generierung der Daten (eigene Darstellung nach [Adh+14])	66
4.2. Plot der Messungen der Frequenz an Relais 1 (eigene Darstellung) . . . . .	68
4.3. Schematische Darstellung des nichtlinearen AE (eigene Darstellung) . . . . .	70
4.4. Histogramm der Test Loss für normale und anormale <i>Samples</i> (eigene Darstellung) .	73
4.5. Histogramm des <i>Features R4-PM3:V</i> (eigene Darstellung) . . . . .	74
5.1. Normale und anormale Daten für das <i>Feature „prozentuale Auslastung Trafo 0“</i> (eigene Darstellung) . . . . .	79
5.2. Konzept zur Erstellung der Daten inkl. aller relevanten Phasen und Komponenten (eigene Darstellung) . . . . .	81
5.3. Drei $20 \times 20$ SOMs zur Darstellung der Unterschiedlichkeit der drei Datensätze.	82
5.4. Trainings-Loss des <i>AutoEncoders</i> über 32 Epochen mit Adam (eigene Darstellung)	83
5.5. Schema des Prognoseprozesses vom <i>AutoEncoder</i> über die logistische Regression bis zum binären <i>Label</i> (eigene Darstellung) . . . . .	84
5.6. <i>Evaluation Loss</i> des Meta-Modells während des <i>Meta-Learnings</i> über 400 Iterationen mittels FOMAML (eigene Darstellung) . . . . .	90
5.7. Schema der Wrapper-Klassen <code>MetaTrainExperiment</code> und <code>FineTuneExperiment</code> (eigene Darstellung) . . . . .	95
6.1. Versuchsablauf inkl. der relevanten Phasen, Modelle und Messzeitpunkte (eigene Darstellung) . . . . .	98
6.2. Histogramme der Faktoren des Versuchsplans $tVP_{M_1}^I$ (eigene Darstellung) . . . .	102
6.3. Histogramme der Kennzahlen <i>Accuracy</i> , <i>Precision</i> , <i>Specificity</i> und <i>Sensitivity</i> der Modelle des Versuchsplans $tVP_{M_1}^I$ auf den Datensätzen $X_{test}$ und $X_{drifted,anormal}$ .	106
6.4. Histogramme der Kennzahlen <i>Accuracy</i> , <i>Precision</i> , <i>Specificity</i> und <i>Sensitivity</i> der Modelle des Versuchsplans $tVP_{M_1}^{II}$ auf den Datensätzen $X_{test}$ und $X_{drifted,anormal}$ .	107
6.5. Histogramme der Kennzahlen <i>Accuracy</i> , <i>Precision</i> , <i>Specificity</i> und <i>Sensitivity</i> der Modelle des Versuchsplans $tVP_{M_{21}}^{II}$ auf den Datensätzen $X_{test}$ und $X_{drifted,anormal}$ .	108

---

6.6.	Boxplot der <i>Accuracy</i> der Modelle $M_1$ der Versuchspläne $tVP_{M_1}^I$ und $tVP_{M_1}^{II}$ auf $\mathbf{X}_{drifted,anormal}$ (eigene Darstellung) . . . . .	110
6.7.	<i>Confusion Matrices</i> der Modelle $M_1$ nach Phase I und Phase II auf $\mathbf{X}_{drifted,anormal}$ (eigene Darstellung) . . . . .	111
A.1.	CIGRE Benchmark Mittelspannungsverteilnetz-Referenzmodell [Str+14] . . . . .	121
A.2.	Beispiel-Matrix zur Aktualisierung der Wirk- und Blindleistungswerte in <i>PandaPower</i> für einen Zeitpunkt (eigene Darstellung) . . . . .	122
A.3.	Auswirkung des Öffnens und Schließen des Schalters S1 im <i>PandaPower</i> CIGRE Netz auf die Messwerte an den entsprechenden Leitungen und Bussen (eigene Darstellung) . . . . .	123
A.4.	Auswirkung auf die Wirk- und Blindleistungswerte durch eine Veränderung des <i>Load Mappings</i> über einen Zeitraum von fünf Tagen (eigene Darstellung) . . . . .	123
A.5.	Schematische Darstellung der Topologie des verwendeten <i>AutoEncoders</i> (eigene Darstellung) . . . . .	124
A.6.	Drei <i>Confusion Matrices</i> mit der Anzahl der jeweils richtig und falsch Klassifizierten <i>Samples</i> auf den unterschiedlichen Datensätzen $\mathbf{X}_{test}$ , $\mathbf{X}_{drifted}$ sowie $\mathbf{X}_{drifted,anormal}$ des Modells $M_2$ . . . . .	125
A.7.	Drei <i>Confusion Matrices</i> mit der Anzahl der jeweils richtig und falsch Klassifizierten <i>Samples</i> auf den unterschiedlichen Datensätzen $\mathbf{X}_{test}$ , $\mathbf{X}_{drifted}$ sowie $\mathbf{X}_{drifted,anormal}$ des Modells $M_1$ . . . . .	126
A.8.	Histogramme der Faktoren des Versuchsplans $tVP_{M_1}^{II}$ (eigene Darstellung) . . . . .	127
A.9.	Histogramme der Faktoren des Versuchsplans $tVP_{M_2}^{II}$ (eigene Darstellung) . . . . .	127



## Tabellenverzeichnis

2.1. Ergebnisse von MAML und <i>Reptile</i> auf dem <i>Mini-ImageNet</i> Datensatz nach [NAS18]	42
3.1. Ergebnisse der Auswahl eines Datensatzes im Rahmen der Nutzwertanalyse . . . . .	55
4.1. Kennzahlen beider AutoEncoder . . . . .	72
5.1. Verwendete SEP und SLP zur Modellierung in <i>PandaPower</i> . . . . .	76
5.2. Zuordnung der Lasten im CIGRE Benchmark-Referenzmodell in <i>PandaPower</i> . . . . .	77
5.3. Übersicht über die Anzahl der <i>Samples</i> und Manipulation der Daten pro Jahr . . . . .	81
5.4. Kennzahlen zur Performance des <i>AutoEncoders</i> auf drei unterschiedlichen Datensätzen zur Evaluation. Die jeweils beste Kennzahl ist fett hervorgehoben. . . . .	84
5.5. Kennzahlen des <i>AutoEncoders</i> $M_2$ pro Art der Anomalie (0: keine Anomalie, 1: Punktanomalie, 2: Kollektive Anomalie, 3: Rauschen) und Datensatz $\mathbf{X}_{test}$ und $\mathbf{X}_{drifted,anormal}$	86
5.6. Kennzahlen des <i>AutoEncoders</i> $M_2$ für die drei unterschiedlichen Arten zur Erzeugung eines <i>Concept Drifts</i> auf dem Datensatz $\mathbf{X}_{drifted,anormal}$ . . . . .	87
5.7. Kennzahlen zur Performance des Meta-Modell <i>AutoEncoders</i> $M_1$ auf drei unterschiedlichen Datensätzen zur Evaluation. Die jeweils beste Kennzahl ist fett hervorgehoben. . . . .	93
5.8. Kennzahlen des <i>AutoEncoders</i> $M_1$ pro Art der Anomalie (0: keine Anomalie, 1: Punktanomalie, 2: Kollektive Anomalie, 3: Rauschen) und Datensatz $\mathbf{X}_{test}$ und $\mathbf{X}_{drifted,anormal}$ vor dem <i>Fine-Tuning</i> . . . . .	93
5.9. Kennzahlen des <i>AutoEncoders</i> $M_1$ für die drei unterschiedlichen Arten zur Erzeugung eines <i>Concept Drifts</i> auf dem Datensatz $\mathbf{X}_{drifted,anormal}$ . . . . .	94
6.1. Parameter der Phase I inkl. möglicher Werteausprägungen sowie der Quelle für die Empfehlung der Werteausprägungen . . . . .	99
6.2. Parameter der Phase II inkl. möglicher Werteausprägungen sowie der Quelle für die Empfehlung der Werteausprägungen . . . . .	100
6.3. Größe und geschätzte Zeit der Durchführung der drei unterschiedlichen teilstatistischen Versuchspläne . . . . .	103
6.4. Statistische Kennzahlen der Kennzahlen der 97 Modelle des Versuchsplans $tVP_{M_1}^I$ jeweils auf $\mathbf{X}_{test}$ und $\mathbf{X}_{drifted,anormal}$ . . . . .	103
6.5. Statistische Kennzahlen der Kennzahlen der 97 Modelle des Versuchsplans $tVP_{M_1}^{II}$ jeweils auf $\mathbf{X}_{test}$ und $\mathbf{X}_{drifted,anormal}$ . . . . .	104
6.6. Statistische Kennzahlen der Kennzahlen der 37 Modelle des Versuchsplans $tVP_{M_2}^{II}$ jeweils auf $\mathbf{X}_{test}$ und $\mathbf{X}_{drifted,anormal}$ . . . . .	105
6.7. <i>Accuracy</i> und <i>Precision</i> pro Modell auf $\mathbf{X}_{drifted,anormal}$ pro Versuchsplan $tVP_{M_1}^I$ und $tVP_{M_1}^{II}$ . . . . .	110
6.8. <i>Accuracy</i> der Modelle $M_1$ der Versuchspläne $tVP_{M_1}^I$ und $tVP_{M_1}^{II}$ auf $\mathbf{X}_{drifted,anormal}$ pro Anomalie Art . . . . .	111

6.9. Kennzahlen der Modelle $M_1$ für die Versuchspläne $tVP_{M_1}^I$ und $tVP_{M_1}^{II}$ auf $\mathbf{X}_{drifted,anormal}$ pro <i>Concept Drift Event Art</i> . . . . .	112
6.10. Prozentuale Veränderung der Kennzahlen der Modelle $M_1$ zwischen den Versuchsplänen $tVP_{M_1}^I$ und $tVP_{M_1}^{II}$ auf $\mathbf{X}_{drifted,anormal}$ pro <i>Concept Drift Event Art</i> . . . . .	113
6.11. Kennzahlen der Modelle $M_1$ und $M_2$ auf $\mathbf{X}_{drifted,anormal}$ jeweils nach Phase I und Phase II . . . . .	114
C.1. Messwerte der Simulation in <i>PandaPower</i> für einen Zeitpunkt $t$ . . . . .	140
C.2. Reduzierte Anzahl der verwendeten <i>Features</i> für das Training des <i>AutoEncoders</i> . . . . .	141
C.3. Vollständiger teilst faktorieller Versuchsplan $tVP_{M_1}^I$ . . . . .	143
C.4. Vollständiger teilst faktorieller Versuchsplan $tVP_{M_1}^{II}$ . . . . .	149
C.5. Vollständiger teilst faktorieller Versuchsplan $tVP_{M_2}^{II}$ . . . . .	150

## Literatur

- [AC15] Jinwon An und Sungzoon Cho. "Variational autoencoder based anomaly detection using reconstruction probability". In: *Special Lecture on IE* 2.1 (2015).
- [AC16] Samaneh Aminikhanghahi und Diane J. Cook. "A survey of methods for time series change point detection". In: *Knowledge and Information Systems* 51.2 (Sep. 2016), S. 339–367. DOI: 10.1007/s10115-016-0987-z.
- [Adh+14] Uttam Adhikari et al. *Power System Attack Datasets*. online. 2014.
- [AES18] Antreas Antoniou, Harrison Edwards und Amos Storkey. "How to train your MAML". In: (22. Okt. 2018).
- [Agg17] Charu C. Aggarwal. *Outlier Analysis*. Springer International Publishing, 2017. DOI: 10.1007/978-3-319-47578-3.
- [Amo+18] Matthew Amodio et al. "Out-of-Sample Extrapolation with Neuron Editing". In: (30. Mai 2018).
- [Arn+19] Sebastien M. R. Arnold et al. *learn2learn*. Sep. 2019.
- [AS17] Charu C. Aggarwal und Saket Sathe. *Outlier Ensembles*. Springer International Publishing, 2017. DOI: 10.1007/978-3-319-54765-7.
- [ASK17] Tomasz Andrysiak, Łukasz Saganowski und Piotr Kiedrowski. "Anomaly Detection in Smart Metering Infrastructure with the Use of Time Series Analysis". In: *Journal of Sensors* 2017 (2017), S. 1–15. DOI: 10.1155/2017/8782131.
- [Ass17] Jan Paul Assendorp. "Deep learning for anomaly detection in multivariate time series data". Magisterarb. Hamburg University of Applied Science, 2017.
- [Bal12] Pierre Baldi. "Autoencoders, Unsupervised Learning, and Deep Architectures". In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. Hrsg. von Isabelle Guyon et al. Bd. 27. Proceedings of Machine Learning Research. Bellevue, Washington, USA: PMLR, Juni 2012, S. 37–49.
- [Bis18] Abhijat Biswas. "First-order Meta-Learned Initialization for Faster Adaptation in Deep Reinforcement Learning". In: *32nd Conference on Neural Information Processing Systems (NIPS 2018)* (2018).
- [Bor+18] Andrea Borghesi et al. "Anomaly Detection using Autoencoders in High Performance Computing Systems". In: *CoRR* abs/1811.05269 (2018).
- [Bun10] Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen. *Blindleistungsbereitstellung für den Netzbetrieb*. 2010.
- [Cas+03] Brian Caswell et al. *Snort 2.0 Intrusion Detection*. Syngress, 2003. ISBN: 978-1-931836-74-6.
- [CBK09] Varun Chandola, Arindam Banerjee und Vipin Kumar. "Anomaly detection: A Survey". In: *ACM Computing Surveys* 41.3 (Juni 2009), S. 1–58. DOI: 10.1145/1541880.1541882.
- [CC19] Raghavendra Chalapathy und Sanjay Chawla. "Deep Learning for Anomaly Detection: A Survey". In: *arXiv:1901.03407 [cs, stat]* (10. Jan. 2019).

- [Chi19] Ashok Chilakapati. *Concept Drift and Model Decay in Machine Learning*. 2019. URL: <http://xplordat.com/2019/04/25/concept-drift-and-model-decay-in-machine-learning/> (besucht am 18.08.2019).
- [CW17] Michał Choras und Michał Woźniak. “Concept Drift Analysis for Improving Anomaly Detection Systems in Cybersecurity”. In: *Advances in Cybersecurity 2017*. University of Maribor Press, Nov. 2017, S. 35–42. DOI: 10.18690/978-961-286-114-8.3.
- [Das+19] Shubhomoy Das et al. “Active Anomaly Detection via Ensembles: Insights, Algorithms, and Interpretability”. In: (23. Jan. 2019).
- [Del+19] Tristan Deleu et al. “Torchmeta: A Meta-Learning library for PyTorch”. In: (14. Sep. 2019).
- [Ell18] David Ellison. *Fraud Detection Using Autoencoders in Keras with a TensorFlow Backend*. 8. Aug. 2018. URL: <https://blogs.oracle.com/datascience/> (besucht am 26.01.2020).
- [Ene19] Energienetze Bayern GmbH. *Standardeinspeiseprofile*. 2019.
- [FAL17] Chelsea Finn, Pieter Abbeel und Sergey Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *CoRR* abs/1703.03400 (2017).
- [Feh+16] Jörg Fehr et al. “Best practices for replicability, reproducibility and reusability of computer-based experiments exemplified by model reduction software”. In: *AIMS Mathematics* 1.3 (2016), S. 261–281. DOI: 10.3934/math.2016.3.261.
- [Fen+17] Zheng Fengming et al. “Anomaly detection in smart grid based on encoder-decoder framework with recurrent neural network”. In: *The Journal of China Universities of Posts and Telecommunications* 24.6 (Dez. 2017), S. 67–73. DOI: 10.1016/s1005-8885(17)60243-7.
- [FF02] Christian Fünfgeld und Carsten Fiebig. *Bestimmung von Lastprofilen für unterbrechbare Verbrauchseinrichtungen*. Abschlussbericht. Energieressourcen-Institut e.V., 16. Sep. 2002.
- [Fin17] Chelsea Finn. *Learning to Learn*. Berkeley Artificial Intelligence Research. 18. Juli 2017. URL: <https://bair.berkeley.edu/blog/2017/07/18/learning-to-learn/> (besucht am 23.10.2019).
- [FLV16] Pavel Filonov, Andrey Lavrentyev und Artem Vorontsov. “Multivariate Industrial Time Series with Cyber-Attack Simulation: Fault Detection Using an LSTM-based Predictive Data Model”. In: (20. Dez. 2016).
- [Gal00] A. Galántai. “The theory of Newton’s method”. In: *Journal of Computational and Applied Mathematics* 124.1-2 (Dez. 2000), S. 25–44. DOI: 10.1016/s0377-0427(00)00435-0.
- [Gam+14] João Gama et al. “A survey on concept drift adaptation”. In: *ACM Computing Surveys* 46.4 (März 2014), S. 1–37. DOI: 10.1145/2523813.
- [GBC16] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [Gér18] Aurélien Géron. *Neural networks and deep learning*. O’Reilly Media, Inc., 2018. ISBN: 9781492037354.

- [Goe+14] Nico Goernitz et al. “Toward Supervised Anomaly Detection”. In: *Journal Of Artificial Intelligence Research, Volume 46, pages 235-262, 2013* (23. Jan. 2014). DOI: 10.1613/jair.3623.
- [Goh+17] Jonathan Goh et al. “A Dataset to Support Research in the Design of Secure Water Treatment Systems”. In: *Critical Information Infrastructures Security*. Springer International Publishing, 2017, S. 88–99. DOI: 10.1007/978-3-319-71368-7\_8.
- [Gul+16] Anton Gulenko et al. “Evaluating machine learning algorithms for anomaly detection in clouds”. In: Dez. 2016, S. 2716–2721. DOI: 10.1109/BigData.2016.7840917.
- [Guo+18] Yifan Guo et al. “Multidimensional Time Series Anomaly Detection: A GRU-based Gaussian Mixture Variational Autoencoder Approach”. In: *Proceedings of The 10th Asian Conference on Machine Learning*. Hrsg. von Jun Zhu und Ichiro Takeuchi. Bd. 95. Proceedings of Machine Learning Research. PMLR, Nov. 2018, S. 97–112.
- [Hal19] Jeff Hale. *Which Deep Learning Framework is Growing Fastest?* 2019. URL: <https://towardsdatascience.com/which-deep-learning-framework-is-growing-fastest-3f77f14aa318> (besucht am 23.11.2019).
- [Haw80] D. M. Hawkins. *Identification of Outliers*. Springer Netherlands, 1980. DOI: 10.1007/978-94-015-3994-4.
- [Haz+11] Nurul Hazwani et al. “Performance Comparison between Multilayer Perceptron and Fuzzy ARTMAP Networks for Acute Leukemia Detection”. In: *International Journal of Research and Reviews in Computer Science (IJRRCS)* 2 (Okt. 2011), S. 1160–1166.
- [He+16] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016. DOI: 10.1109/cvpr.2016.90.
- [He19] Horace He. *The State of Machine Learning Frameworks in 2019*. 2019. URL: <https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/> (besucht am 23.11.2019).
- [Hea08] Jeff Heaton. *Introduction to Neural Networks with Java, 2nd Edition*. Heaton Research, Incorporated, 2008. ISBN: 1604390085.
- [Heb02] D.O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Psychology Press, 2002. ISBN: 0-8058-4300-0.
- [Hin+14] Raymond C. Borges Hink et al. “Machine learning for power system disturbance and cyber-attack discrimination”. In: *2014 7th International Symposium on Resilient Control Systems (ISRCS)*. IEEE, 2014. DOI: 10.1109/isrcs.2014.6900095.
- [HKP17] Jiawei Han, Micheline Kamber und Jian Pei. *Data Mining: Concepts and Techniques*. Elsevier LTD, Oxford, 1. Aug. 2017. ISBN: 0123814790.
- [Inn+18] Michael Innes et al. “On Machine Learning and Programming Languages”. In: Feb. 2018.
- [KKH02] Ralph L. Keeney, L. Keeney Ralph und Raiffa Howard. *Decisions with Multiple Objectives*. Cambridge University Press, 2002. 592 S. ISBN: 0521438837.
- [Kon14] Konrad Adenauer Stiftung. *Netzausbau in Deutschland*. Hrsg. von Philipp Lerch und Tobias Montag. 2014.

- [KR17] Matthias Karmasin und Rainer Ribing. *Die Gestaltung wissenschaftlicher Arbeiten: Ein Leitfaden für Facharbeit/VWA, Seminararbeiten, Bachelor-, Master-, Magister- und Diplomarbeiten sowie Dissertationen*. 9. Aufl. Bd. 2774. UTB. Wien: Facultas, 2017. ISBN: 978-3-8252-4822-2.
- [Kri07] David Kriesel. *Ein kleiner Überblick über Neuronale Netze*. 2007.
- [Kru+13] Rudolf Kruse et al. *Computational Intelligence: A Methodological Introduction*. Springer London, 2013. DOI: 10.1007/978-1-4471-5013-8.
- [KTP18] B Ravi Kiran, Dilip Mathew Thomas und Ranjith Parakkal. “An overview of deep learning based methods for unsupervised and semi-supervised anomaly detection in videos”. In: (9. Jan. 2018).
- [Laz+03] Aleksandar Lazarevic et al. “A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection”. In: *Proceedings of the 2003 SIAM International Conference on Data Mining*. Society for Industrial und Applied Mathematics, Mai 2003. DOI: 10.1137/1.9781611972733.3.
- [LBH15] Yann LeCun, Yoshua Bengio und Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (Mai 2015), S. 436–444. DOI: 10.1038/nature14539.
- [Lee17] Doyup Lee. “Anomaly Detection in Multivariate Non-stationary Time Series for Automatic DBMS Diagnosis”. In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, Dez. 2017. DOI: 10.1109/icmla.2017.0-126.
- [LST15] B. M. Lake, R. Salakhutdinov und J. B. Tenenbaum. “Human-level concept learning through probabilistic program induction”. In: *Science* 350.6266 (2015), S. 1332–1338. DOI: 10.1126/science.aab3050.
- [Mal+16] Pankaj Malhotra et al. “LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection”. In: *CoRR* abs/1607.00148 (2016).
- [Mit19] Theophano Mitsa. *How Do You Know You Have Enough Training Data?* 2019. URL: <https://towardsdatascience.com/how-do-you-know-you-have-enough-training-data-ad9b1fd679ee> (besucht am 18.08.2019).
- [Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Education, 1997. ISBN: 0070428077.
- [Mor+13] Leonardo Raul Morelli et al. “Deep Architectures on Drifting Concepts: A Simple Approach”. In: 2013.
- [MP69] Marvin Minsky und Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press Ltd, 1969. ISBN: 9780262630221.
- [Mun+19] Mohsin Munir et al. “DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series”. In: *IEEE Access* 7 (2019), S. 1991–2005. DOI: 10.1109/access.2018.2886457.
- [Mur12] Kevin P Murphy. *Machine learning: a probabilistic perspective*. Cambridge, MA, 2012.
- [NAS18] Alex Nichol, Joshua Achiam und John Schulman. “On First-Order Meta-Learning Algorithms”. In: *CoRR* abs/1803.02999 (2018).
- [NS18] Alex Nichol und John Schulman. “Reptile: a Scalable Metalearning Algorithm”. In: 2018.

- [Ped+11] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830.
- [PMA15a] Shengyi Pan, Thomas Morris und Uttam Adhikari. “A specification-based intrusion detection framework for cyber-physical environment in electric power system”. In: *International Journal of Network Security* 17 (Jan. 2015), S. 174–188.
- [PMA15b] Shengyi Pan, Thomas Morris und Uttam Adhikari. “Classification of Disturbances and Cyber-Attacks in Power Systems Using Heterogeneous Time-Synchronized Data”. In: *IEEE Transactions on Industrial Informatics* 11.3 (Juni 2015), S. 650–662. DOI: 10.1109/tii.2015.2420951.
- [PMA15c] Shengyi Pan, Thomas Morris und Uttam Adhikari. “Developing a Hybrid Intrusion Detection System Using Data Mining for Power Systems”. In: *IEEE Transactions on Smart Grid* 6.6 (Nov. 2015), S. 3104–3113. DOI: 10.1109/tsg.2015.2409775.
- [Pre+95] Lutz Prechelt et al. “Experimental Evaluation in Computer Science: A Quantitative Study”. In: *Journal of Systems and Software* 28.1 (1995), S. 9–18.
- [Qia+19] Jimin Qian et al. “Introducing self-organized maps (SOM) as a visualization tool for materials research and education”. In: *Results in Materials* 4 (Dez. 2019), S. 100020. DOI: 10.1016/j.rinma.2019.100020.
- [Rav18] Sudharsan Ravichandiran. *Hands-On Meta Learning with Python*. Packt Publishing, 28. Dez. 2018. 226 S. ISBN: 1789534208.
- [Ref19] Referat DG II 1. *Handbuch für Organisationsuntersuchungen und Personalbedarfsermittlung*. Techn. Ber. Bundesministerium des Innern, für Bau und Heimat (BMI), 2019.
- [Ric18] Andrew G. Barto Richard S. Sutton. *Reinforcement Learning*. The MIT Press, 13. Nov. 2018. 552 S. ISBN: 0262039249.
- [RL17] Sachin Ravi und Hugo Larochelle. “Optimization as a Model for Few-Shot Learning”. In: *ICLR 2017*. 2017.
- [Ruf+19] Lukas Ruff et al. “Deep Semi-Supervised Anomaly Detection”. In: (2019).
- [Sau+18] Sakti Saurav et al. “Online anomaly detection with concept drift adaptation using recurrent neural networks”. In: *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data - CoDS-COMAD '18*. ACM Press, 2018. DOI: 10.1145/3152494.3152501.
- [SBH10] Karl Siebertz, David van Bebber und Thomas Hochkirchen. *Statistische Versuchsplnung*. Springer Berlin Heidelberg, 2010. DOI: 10.1007/978-3-642-05493-8.
- [Sch+17] Thomas Schlegl et al. “Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery”. In: (17. März 2017).
- [Sch14] Juergen Schmidhuber. “Deep Learning in Neural Networks: An Overview”. In: *Neural Networks, Vol 61, pp 85-117, Jan 2015* (30. Apr. 2014). DOI: 10.1016/j.neunet.2014.09.003.
- [Sha14] Shai Ben-David Shai Shalev-Shwartz. *Understanding Machine Learning*. Cambridge University Pr., 17. Juli 2014. 409 S. ISBN: 1107057132.
- [SM02] Kenneth O. Stanley und Risto Miikkulainen. “Evolving Neural Networks through Augmenting Topologies”. In: *Evolutionary Computation* 10.2 (Juni 2002), S. 99–127. DOI: 10.1162/106365602320169811.

- [SMA19] N. N. R. Ranga Suri, Narasimha Murty M und G. Athithan. *Outlier Detection: Techniques and Applications*. Springer International Publishing, 2019. DOI: 10.1007/978-3-030-05127-3.
- [Son+07] Xiuyao Song et al. “Conditional Anomaly Detection”. In: *IEEE Transactions on Knowledge and Data Engineering* 19.5 (Mai 2007), S. 631–645. DOI: 10.1109/tkde.2007.1009.
- [SSZ17] Jake Snell, Kevin Swersky und Richard S. Zemel. “Prototypical Networks for Few-shot Learning”. In: (15. März 2017).
- [Sta20] Stadtwerke Emmendingen GmbH. *Lastprofile*. 11. Feb. 2020. URL: <https://swe-emmendingen.de/strom-netz/lastprofile/>.
- [Ste99] D. Steinkamp. “Informatik-Experimente im Schullabor”. Magisterarb. Universität Dortmund, Fachbereich Informatik, 1999.
- [Str+14] K. Strunz et al. *Benchmark Systems for Network Integration of Renewable and Distributed Energy Resources*. Technical Report. Conseil International des Grands Réseaux Électriques, 2014.
- [Stu12] Peter Norvig Stuart Russell. *Künstliche Intelligenz*. Pearson Deutschland GmbH, 1. Juni 2012. 1312 S.
- [SU12] Karanjit Singh und Shuchita Upadhyaya. “Outlier Detection: Applications And Techniques”. In: *International Journal of Computer Science Issues* 9 (2012).
- [Tsy04] Alexey Tsymbal. “The Problem of Concept Drift: Definitions and Related Work”. In: (Mai 2004).
- [Van19] Joaquin Vanschoren. “Meta-Learning”. In: *Automated Machine Learning*. Springer International Publishing, 2019, S. 35–61. DOI: 10.1007/978-3-030-05318-5\_2.
- [Wan+19] Yaqing Wang et al. “Generalizing from a Few Examples: A Survey on Few-Shot Learning”. In: (10. Apr. 2019).
- [Wen18] Lilian Weng. “Meta-Learning: Learning to Learn Fast”. In: [lilianweng.github.io/lil-log](https://lilianweng.github.io/lil-log) (2018).
- [Wik19] Wikipedia. *Künstliches Neuron — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 4. Oktober 2019]. 2019.
- [WSK10] Shenghui Wang, Stefan Schlöbach und Michel Klein. “What Is Concept Drift and How to Measure It?” In: *Knowledge Engineering and Management by the Masses*. Hrsg. von Philipp Cimiano und H. Sofia Pinto. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 241–256. ISBN: 978-3-642-16438-5. DOI: 10.1007/978-3-642-16438-5\_17.
- [Zha+18] Chuxu Zhang et al. “A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data”. In: (2018).

# Index

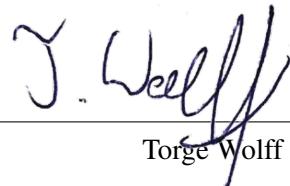
- Anomalie, 1, 26
  - Arten, 28
  - Binary Label, 27
  - Collective Anomaly, 28
  - Contextual Anomaly, 28
  - Outlier Score, 27
  - Point Anomaly, 28
  - Problemcharakteristik, 57
- Artificial Neural Networks, 10
  - Aktivierungsfunktion, 15
  - Backpropagation, 18
  - Loss Function, 19
  - Multi-Layer Perceptron, 13
  - Neuron, 10
  - Perzeptron, 11
  - Training, 17
- Autoencoder, 1, 4, 22, 83
  - Implementierung, 83
  - Overcomplete, 24
  - Reconstruction Loss, 25
  - Training, 85
  - Undercomplete, 24
- Concept Drift, 20, 21
- Data Generator, 57
  - Anforderungen, 57
  - AnomalyGenerator, 78
  - Executor, 75
  - Implementierung, 75
  - Konzept, 58
  - Lastprofile, 76
  - NetworkTopologyChanger, 79
- Deep Learning, 19
- Evaluation, 114
- Experiment, 56, 97
  - Ablauf, 97
  - Durchführung, 102
  - Erkenntnisse, 109
  - Fine-Tuning, 103
  - Meta-Training, 103
  - Parameter, 99
  - Versuchsplanung, 98
- Few-shot Learning, 3
- Forschungsfragen, 47
- Hypothese, 49
- k-shot Learning, 34
- Machine Learning, 1, 7
  - Deep Learning, 19
- Meta-Learning, 4, 33, 87
  - FOMAML, 38
  - Implementierung, 87
  - MAML, 4, 35
  - Reptile, 39
- PandaPower, 76
- Thesen, 48
- Versuchsplan, 100, 101
  - Teilfaktoriell, 100
  - Vollfaktoriell, 100
- Versuchsplanung, 49
- Vorstudie, 65
  - Erkenntnisse, 72
  - Preprocessing, 69
  - Zielstellung, 65
- Wrapper-Klassen, 94
- Zielstellung, 47
  - Analyse, 115
  - Annahmen, 48
  - Forschungsfragen, 47
  - Hypothese, 49
  - Thesen, 48



## Versicherung

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Außerdem versichere ich, dass ich die allgemeinen Prinzipien wissenschaftlicher Arbeit und Veröffentlichung, wie sie in den Leitlinien guter wissenschaftlicher Praxis der Carl von Ossietzky Universität Oldenburg festgelegt sind, befolgt habe.

Oldenburg, den 14. April 2020

A handwritten signature in black ink, appearing to read "T. Wolff".

Torge Wolff