cyber|
agentur

HOCHSCHULE
MITTWEIDA
University of Applied Sciences

MASTERARBEIT

Frau
Olivia Sina Gräupner, B.Sc.

Grundlagen auditierbarer KI-Systeme

Halle (Saale), Dezember 2022

Fakultät **Angewandte Computer- und Biowissenschaften**

MASTERARBEIT

Grundlagen auditierbarer KI-Systeme

Autorin:
Olivia Sina Gräupner

Studiengang:
Cybercrime/Cybersecurity

Seminargruppe:
CY20wC-M

Erstprüfer:
Prof. Dr. rer. pol. Dirk Pawlaszczyk

Zweitprüfer:
Prof. Dr. rer. nat. Christian Hummert

Einreichung:
Halle (Saale), 12.12.2022

Verteidigung/Bewertung:
Halle (Saale), 2023

Faculty of **Applied Computer Sciences and Biosciences**

MASTER THESIS

Basics of auditable AI systems

Author:
Olivia Sina Gräupner

Course of Study:
Cybercrime/Cybersecurity

Seminar Group:
CY20wC-M

First Examiner:
Prof. Dr. rer. pol. Dirk Pawlaszczyk

Second Examiner:
Prof. Dr. rer. nat. Christian Hummert

Submission:
Halle (Saale), 12.12.2022

Defense/Evaluation:
Halle (Saale), 2023

Bibliografische Beschreibung:

Gräupner, Olivia Sina:
Grundlagen auditierbarer KI-Systeme. 2022. 97 S.
Mittweida, Hochschule Mittweida – University of Applied Sciences,
Fakultät Angewandte Computer- und Biowissenschaften, Masterarbeit, 2023.

Referat:

Diese Masterarbeit analysiert die Möglichkeiten der Auditierung von Künstlicher Intelligenz in der Theorie und der Praxis. Im Rahmen eines anwendungsnahen Szenarios wird mithilfe des Frameworks Avalanche ein kontinuierlich lernendes System konstruiert. Deren Gewichtswerte sowie die Änderung der Gewichte werden in einer Logdatei gespeichert. Der Verlauf der Änderungen sowie der entstehende Speicherbedarf bei variierender Hidden Layer-Zahl und Neuronenanzahl gibt Rückschlüsse über die Anwendbarkeit der betrachteten Methode. Diese Vorgehensweise ist alleinstehend nicht ausreichend für eine umfassende Auditierung, wodurch weiterführende Forschung notwendig ist.

Abstract:

This master's thesis analyses the possibilities of auditing artificial intelligence in theory and in practice. Within the framework of an application-oriented scenario, a continuously learning system is constructed with the help of the framework Avalanche. Its weight values and the weight changes are stored in a log file. The development of the changes as well as the resulting memory with varying hidden layer and number of neurons provides conclusions about the applicability of the method. This approach on its own is not sufficient for comprehensive auditing. Therefore further research is necessary.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AI . . . . . Artificial Intelligence

AIA . . . . . Artificial Intelligence Act

AIX360 . . . . . AI Explainability 360

ANN . . . . . Artificial Neural Networks

BSI . . . . . Bundesamt für Sicherheit in der Informationstechnik

CAE . . . . . Claims, Arguments, Evidence

cAI . . . . . connectionist AI

CEN . . . . . Comité Européen de Normalisation

CENELEC . . . . . Comité Européen de Normalisation Electrotechnique

cIT . . . . . classical IT

CL . . . . . Continual Learning

CMD . . . . . Command

CNN . . . . . Convolutional Neural Network

CoC . . . . . Chain of Custody

CRM . . . . . Certification Readiness Matrix

CUDA . . . . . Compute Unified Device Architecture

DeepLIFT . . . . . Deep Learning Important FeaTures

DFR . . . . . Digital Forensic Readiness

DIN . . . . . Deutsche Institut für Normung e.V.

DKE . . . . . Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE

DNN . . . . . Deep Neural Networks

DSGVO . . . . . Datenschutz-Grundverordnung

ETSI . . . . . European Telecommunications Standards Institute

EWC . . . . . Elastic Weight Consolidation

FAT32 . . . . . File Allocation Table 32

FF . . . . . Feed Forward

GeDIS . . . . . Gender/Diversity in Informatics Systems

GPU . . . . . Graphics Processing Unit

Grad-CAM . . . . . Gradient-weighted Class Activation Mapping

GUI . . . . . Graphical User Interface

HHI . . . . . Fraunhofer-Institut für Nachrichtentechnik, Heinrich-Hertz-Institut

HL . . . . . Hidden Layer

IAIS . . . . . Fraunhofer-Institut für Intelligente Analyse- und Informationssysteme

IBM . . . . . International Business Machines Corporation

IEC . . . . . International Electronical Commission

IKT . . . . . Informations- und Kommunikationstechnik

IL . . . . . Incremental Learning

ISIC . . . . . International Skin Imaging Collaboration

ISO . . . . . International Standards Organization

ITU . . . . . International Telecommunication Union

KI . . . . . Künstliche Intelligenz

KNN . . . . . Künstliche Neuronale Netze

LIME . . . . . Local Interpretable Model-agnostic Explanations

LRP . . . . . Layer-Wise Relevance Propagation

macOS . . . . . Macintosh Operating System

MD5 . . . . . Message-Digest Algorithm 5

ML . . . . . Machine Learning

MLOps . . . . . Machine Learning Operations

MLP . . . . . Multi-Layer Perceptron

MNIST . . . . . Modified National Institute of Standards and Technology

NIST . . . . . National Institute of Standards and Technology

NLP . . . . . Natural Language Processing

NN . . . . . Neural Network

NTFS . . . . . New Technology File System

OvA . . . . . One-vs-All

PATE . . . . . Privacy Aggregation of Teacher Ensembles

ReLU . . . . . Rectified Linear Unit

SA . . . . . Sensitivity Analysis

SAI . . . . . Securing Artificial Intelligence

sAI . . . . . symbolic AI

SC . . . . . Spectral Cluster Analysis

SGD . . . . . Stochastic Gradient Descent

SHA-1 . . . . . Secure Hash Algorithm 1

SHA-256 . . . . . Secure Hash Algorithm 256

SHAP . . . . . SHapley Additive exPlanations

SLP . . . . . Single-Layer Perceptron

SpRAy . . . . . Spectral Relevance Analysis

Spyder . . . . . Scientific PYthon Development EnviRonement

SQuaRE . . . . . Systems and software Quality Requirements and Evaluation

SSD . . . . . Solid State Drive

SVM . . . . . Support Vector Machine

t-SNE . . . . . t-distributed Stochastic Neighbor Embedding

Tf . . . . . TensorFlow

VIKING . . . . . Vertrauenswürdige Künstliche Intelligenz für polizeiliche Anwendungen

VOC . . . . . Visual Object Classes

W&B . . . . . Weights & Biases

XAI . . . . . Explainable Artificial Intelligence

XOR . . . . . eXclusive OR

Zennit . . . . . Zennit explains neural networks in torch

ZEW . . . . . Leibniz-Zentrum für Europäische Wirtschaftsforschung

1 Einleitung

1.1 Motivation

Anwendung von KI in der deutschen Wirtschaft Laut der „Zusatzbefragung KI“ von 2019/2020 des Leibniz-Zentrum für Europäische Wirtschaftsforschung (ZEW) verwenden 5,8 % der deutschen Unternehmen Künstliche Intelligenz (KI), das entspricht einer absoluten Zahl von rund 17.500 Firmen. Insbesondere die Branche Informations- und Kommunikationstechnik (IKT) ist mit 17,8 % Vorreiter bei der Anwendung von intelligenten Systemen. Der Einsatz von KI in deutschen Unternehmen nimmt nachweislich zu. Trotz des steigenden Personalbedarfs mangelt es gleichzeitig an Fachpersonal, das die korrekte Arbeitsweise der KI überwacht. Diese Tendenz verdeutlicht die essenzielle Bedeutsamkeit der Auditierung von KI. [1]

Problemstellung Die vorliegende Masterarbeit befasst sich daher mit der Problematik, eine KI, engl. *Artificial Intelligence (AI)*, auditierbar zu machen. Dabei stellt sich die Frage: Was bedeutet „auditierbar“ beziehungsweise „Auditierbarkeit“ und wie kann dies im Hinblick auf intelligente Systeme umgesetzt werden? Laut dem deutschen Rechtschreibwörterbuch Duden wird das Verb „auditieren“ definiert als: „etwas als externer Prüfer auf die Erfüllung bestimmter [Qualitäts]standards hin bewerten und anschließend zertifizieren“ [2]. Synonyme für „auditieren“ sind „bewerten“ und „prüfen“ [2]. Bestimmte Vorgänge zu auditieren, hilft demnach, deren Funktionsweise lückenlos zu erklären und stellt daher ein wichtiges Hilfsmittel bei der rechtskonformen Polizeiarbeit dar.

Forensik Die IT-Forensik im Anwendungskontext intelligenter Systeme beschreibt die Datenanalyse, welche einer streng wissenschaftlich vorgegebenen Methodik unterliegt. Die technische Auswertung bezieht sich hierbei auf jegliche Daten, die von der KI eingegeben, verarbeitet und ausgegeben werden. Das Ziel ist die zeitnahe Erkennung von Straftaten und Angriffen auf KI-Systeme sowie proaktives Reagieren auf Sicherheitsvorfälle durch die Bereitstellung von *Forensic Readiness* Maßnahmen. Die Rechenschaftspflicht der digitalen Forensik wird dabei durch die Einhaltung der Chain of Custody (CoC) gewährleistet. Um gerichtsfeste Aussagen treffen zu können, werden im Folgenden anhand der Anforderungen der KI die Ansätze der Auditierbarkeit untersucht. [3]

Anforderungen KI Im Bereich des Machine Learning (ML), dt. maschinelles Lernen, nimmt der praktische Anwendungsbereich in der jüngsten Vergangenheit einen hohen Stellenwert ein. Das Ziel, eine präzise Genauigkeit bei den Modell-Vorhersagen zu erreichen, ist in der Realität allein nicht mehr umsetzbar. Anhand der Abbildung 1.1 ist erkennbar, dass zusätzliche Anforderungen an ein vertrauenswürdiges KI-System gestellt werden. Im Hinblick auf die Auditierbarkeit werden hierbei die Bereiche Ethik und Recht, Autonomie und Kontrolle, Fairness, Transparenz, Verlässlichkeit, Sicherheit und Datenschutz benannt [4, S. 54]. Die vorliegende Arbeit konzentriert sich dabei besonders auf die Aspekte der Kontrolle, Transparenz, Verlässlichkeit und Sicherheit. Jedoch sind diese untrennbar mit den anderen Anforderungen verbunden. [5]

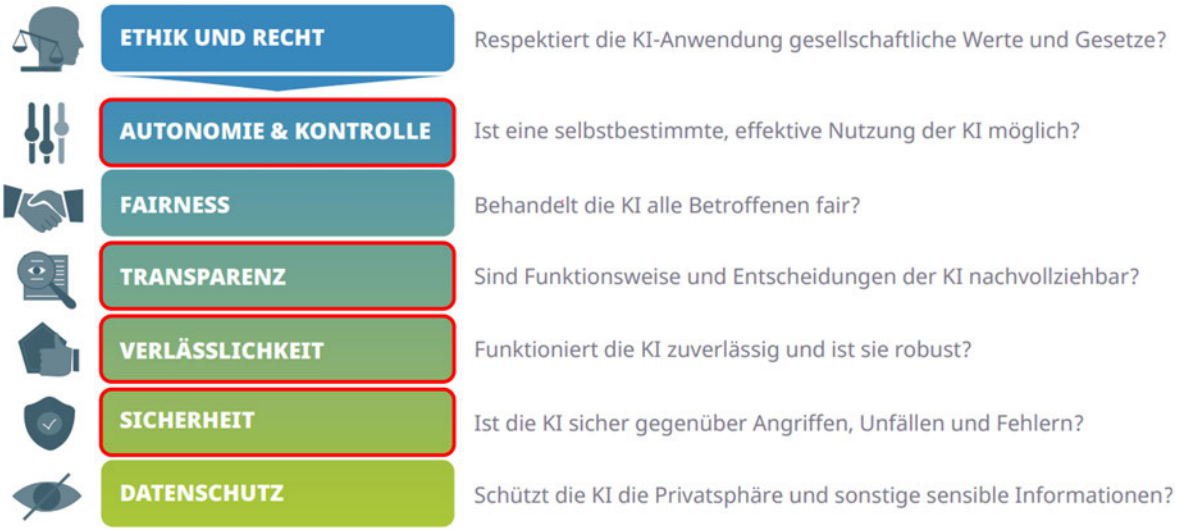


Abbildung 1.1: Die Darstellung zeigt wichtige Fragestellungen, die in Bezug auf eine vertrauenswürdige KI relevant sind. Die rot umrandeten Anforderungen werden hierbei besonders betrachtet. modifiziert nach [4, S. 54]

Anhand dieser Anforderungen ist erkennbar, dass eine stetig wachsende Einigkeit für ein umfassendes Verständnis der ML-Systeme vorherrscht. Im Zuge dessen wurde ein starker Zuwachs an Werkzeugen der Explainable Artificial Intelligence (XAI), als Teilbereich des Machine Learning, verzeichnet. [5]

1.2 Allgemeine Zielstellung

Motiviert wird die Vorgehensweise von dem Standpunkt der digitalen Forensik. Transparenz und Vertrauenswürdigkeit besitzen oberste Priorität bei der Verwendung von KI in allen Anwendungsbereichen. Wird die in Kapitel 1.1 vorgestellte Definition der Auditierbarkeit genauer betrachtet, kann die Zielstellung der folgenden Ausarbeitungen abgeleitet werden. Die Rolle des externen Prüfers wird im Bereich der Forensik klassischerweise durch die IT-Forensiker der Polizei eingenommen. Das Vorhandensein bestimmter Standards bei der Verwendung von KI wird des Weiteren überprüft. Damit sind konkret Auditkriterien gemeint, die notwendig für eine verlässliche Anwendung in der Praxis sind. Hierbei stößt die aktuelle Forschung jedoch bereits an erste Grenzen. Im Bereich der KI existieren große Herausforderungen, die eine einfache Auditierung behindern. Die Arbeitsweise einer KI stellt für die Mehrheit der Anwender*innen eine Art Black-Box dar. Wie sie bestimmte Entscheidungen trifft, ist somit für den Menschen nicht nachvollziehbar. Kommt es durch externe Einflüsse zu Veränderungen, können diese nicht einfach und beweisbar überprüft werden. Herkömmliche forensische Ansätze scheitern bei der Analyse der Entscheidungsfindung der KI. Daher analysiert die vorliegende Arbeit auf welche Art und Weise bestehende Methodiken der Auditierbarkeit auf kontinuierlich lernende KI-Systeme übertragen werden kann. Diese Untersuchung geschieht sowohl aus der theoretischen als auch der praktischen Perspektive.

1.3 Aufbau und Vorgehensweise

Allgemeiner Aufbau Zu Beginn wird eine Übersicht von existierenden Erklärungsverfahren erstellt, die anschießend anhand ausgewählter Merkmale miteinander verglichen werden. Die Gegenüberstellung bestehender Softwarebibliotheken zur Erklärbarkeit von ML-Modellen dient den Anwender*innen als Entscheidungshilfe bei der Verwendung. Es soll geklärt werden, welche Möglichkeiten sie bei einer Auswahl der passenden Bibliothek für den eigenen Anwendungsfall haben. Anschließend soll ein kontinuierlich lernendes ML-System unter der Verwendung eines geeigneten Frameworks aufgebaut werden. Kontinuierliche Systeme zeichnen sich durch ihr realitätsnahes Verhalten aus, fortlaufend neue Eingaben zu erhalten und innerhalb des Modelltrainings zu verwerten. Die internen Hyperparameter unterliegen daher einer ständigen Anpassung, dass die Nachvollziehbarkeit der Entscheidungsfindung erschwert. Diese Überlegungen dienen als Grundlage für die nachfolgenden Ansätze zu Auditierversuchen, die eine eingehende Evaluation erfahren. Konkret sollen die Lernfortschritte innerhalb der KI während des Trainings gespeichert werden. Dieses Protokoll dient als Grundlage für die Manipulationserkennung. Weiterhin soll festgestellt werden, wann das System während des Modelltrainings keinen Lernfortschritt mehr verzeichnet und auf welche Art und Weise sich dieser Zustand äußert. Abschließend soll der Zusammenhang zwischen Anwendbarkeit, Geschwindigkeit und Speicherplatzbedarf ergründet werden. Ob eine solche Vorgehensweise realisierbar ist und welche Ansätze im Bereich der Erklärbarkeit und Auditierbarkeit bestehen, zeigt die folgende Arbeit.

Eigene Leistung Der Eigenanteil der Masterarbeit zeichnet sich durch die Konfiguration eines kontinuierlich lernenden KI-Modells aus, anhand dessen die Lerngewichte während des Modelltrainings mitgeschnitten werden. Auf Grundlage dessen werden Visualisierungen erstellt, die als Unterstützung dienen, die Anwendbarkeit der Methodik zu beurteilen.

1.4 Eingrenzung der Thematik

Die vorliegende Arbeit konzentriert sich auf die technischen Aspekte der Erklärbarkeit sowie der Auditierbarkeit. Daher bleiben ethische, moralische und rechtliche, insbesondere auch datenschutzrechliche Fragestellungen außen vor. Der Aspekt der Zertifizierung bei der Auditierung von KI wird trotz ihrer Relevanz lediglich am Rande der Arbeit behandelt, da bereits Forschungsprojekte in dieser Richtung bestehen. Beispielsweise beschäftigt sich das laufende Projekt von Schmidt et al. mit der Erstellung einer zertifizierten KI [6]. Die vorliegenden Ergebnisse unterliegen dem aktuellen Stand der Technik. Das Forschungsgebiet ist vergleichsweise jung und unterliegt einem kontinuierlichen Wandel und Fortschritt, welcher durch aktuelle Forschungsvorhaben gestützt wird, siehe Kapitel 7. Daher besteht kein Anspruch auf Vollständigkeit. Die Arbeit soll einen umfassenden Überblick geben, bestehende Herausfoderungen aufzeigen und für weiterführende Forschung hinsichtlich der Auditierbarkeit sensibilisieren.

2 Theoretische Grundlagen

Das folgende Kapitel erläutert den aktuellen Stand der Technik sowie die theoretischen Grundlagen der nachfolgenden Ausführungen. Hierbei beschreibt es zu Beginn in 2.1 den Aufbau und die Funktionsweise von Künstlichen Neuronalen Netzen. Danach werden die allgemeinen Herausforderungen sowie die Angriffs- und Abwehrmöglichkeiten auf KI-Systeme diskutiert, Vergleich Kapitel 2.2. Das Kapitel stellt weiterhin die Erklärbare KI in Abschnitt 2.3 vor. Anhand dieser Erkenntnisse folgt der aktuelle Wissenstand im Bereich des Auditierens von KI-Systemen, siehe Abschnitt 2.4. Anschließend wird die bestehende Forschung im Bereich des kontinuierlichen Lernens präsentiert.

Einteilung Im Wesentlichen können IT-Technologien in Abgrenzung zur KI in die drei Bereiche classical IT (cIT), dt. klassische IT, symbolic AI (sAI), dt. symbolische KI, und connectionist AI (cAI), dt. konnektionistische KI, untergliedert werden. Die nachfolgenden Betrachtungen fokussieren sich auf den Bereich der cAI. Dazu gehören beispielsweise Deep Neural Networks (DNN), weil sie in der Mehrheit komplexer Anwendungen eingesetzt werden und Datenstrukturen mit einem großen Parameterraum besitzen. [7, S. 4] [8, S. 4]

Qualitätskriterien Das Fraunhofer-Institut für Nachrichtentechnik, Heinrich-Hertz-Institut (HHI) unterscheidet fünf Qualitätskriterien für KI-Systeme:

- Verallgemeinerbarkeit
- Erklärbarkeit
- Robustheit
- Unsicherheit
- Auditierbarkeit

Das Kriterium Verallgemeinerbarkeit (engl. *Generalizability*) beschreibt die Fähigkeit zum Treffen korrekter Vorhersagen anhand ungesehener Daten. Die Erklärbarkeit (engl. *Explainability*) ermöglicht die Überprüfung von KI-Systemen und das Erlangen neuer Erkenntnisse. Die Widerstandsfähigkeit gegenüber Angriffen und Manipulationsversuchen wird durch die Robustheit (engl. *Robustness*) eines Modells beschrieben. Die Unsicherheit (engl. *Uncertainty*) zeigt das fehlende Wissen eines KI-Systems auf. Mithilfe des Kriteriums Auditierbarkeit (engl. *AI Auditing*) kann die Qualität einer KI bewertet werden. [9]

Digital Forensic Readiness (DFR) DFR bezeichnet die forensische Bereitschaft einer Person oder eines Unternehmens, angemessen auf digitale Sicherheitsrisiken und -vorfälle zu reagieren. Hierbei werden fundierte Entscheidungen getroffen, die im Vorfeld ausgearbeitet wurden und definieren, in welcher Art und Weise digitale Beweise gesichert werden. Dies geschieht im Hinblick darauf, das Risikopotenzial sowie die entstehenden Folgekosten der forensischen Untersuchung proaktiv zu minimieren und die Geschäftsabläufe so wenig wie möglich zu beeinträchtigen. Im Ereignisfall muss das gesamte Unternehmen auf möglichst jede Angriffsart vorbereitet sein und je nach Unternehmensbereich zeitnah sowie angemessen reagieren. Durch die ständige Bereitschaft soll weiterhin die Beweiskraft digitaler Spuren maximiert werden. Um einen Beitrag zur Aufklärung leisten zu können, ist die gerichtsfeste Beweissicherung unter Aufrechterhaltung der CoC notwendig. [10, S. 80 ff.]

2.1 Aufbau und Funktionsweise von Künstlichen Neuronalen Netzen

Künstliche Neuronale Netze (KNN), engl. *Artificial Neural Networks* (ANN), sind ein Bestandteil des *Deep Learning*, das wiederum ein Teilbereich des Machine Learning ist. Die Kernkomponenten von KNN lassen sich anhand eines klassischen Perzeptrons darstellen, dessen Aufbau und Funktion auf natürlichen neuronalen Netzen des menschlichen Gehirns beruht. Im Folgenden werden ausschließlich Beispiele zu vorwärtsbetriebenen Netzwerktopologien, engl. *Feed Forward*, des überwachten ML, engl. *Supervised ML*, betrachtet. Grundsätzlich wird zwischen Single-Layer Perceptron (SLP), Vergleich Abbildung 2.1, und Multi-Layer Perceptron (MLP), Vergleich Abbildung 2.2, unterschieden. Das SLP bildet dabei die Grundlage für mehrschichtige Netzwerkstrukturen. [11]

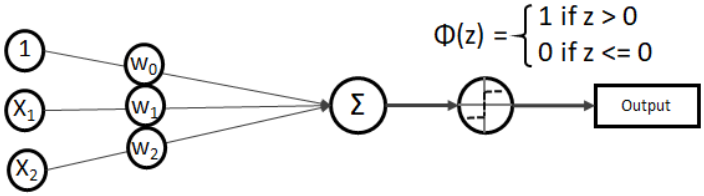


Abbildung 2.1: Erkennbar ist die beispielhafte Netzwerktopologie eines SLP mit dem Bias von 1, festen Eingaben x_1 und x_2, den Gewichten w_0, w_1 und w_2, der gewichteten Summe $\hat{z}$, der Aktivierungsfunktion $\phi(\hat{z})$ und der Ausgabe. [12]

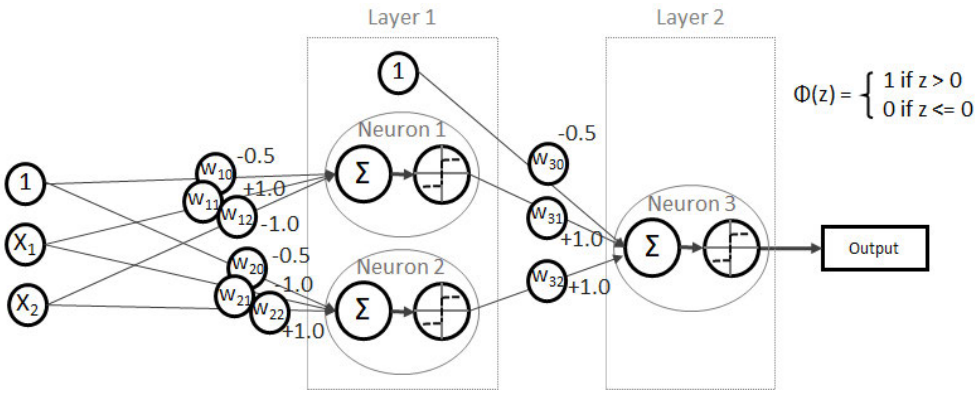


Abbildung 2.2: Dargestellt ist der Ausschnitt der Netzwerktopologie eines XOR-MLP. Sie beinhaltet eine feste Eingabeschicht x_i mit Bias, zwei Hidden Layer (HL) mit drei Neuronen inklusive gewichteter Summe $\hat{z}$ und deren Aktivierungsfunkion $\phi(\hat{z})$, die Verbindungsgewichte w_i und eine Ausgabeschicht. [12]

Initialisierung Der Eingabe-Layer besteht aus einer Reihe von künstlichen Neuronen, auch als Verarbeitungseinheiten (engl. *Processing Elements*) bezeichnet, über die numerische Signale x_i $(i = 1, \ldots, n)$ aufgenommen werden. Die Variable n stellt hierbei die Anzahl der Eingaben dar. Die dynamischen Gewichte w_i dienen der Regulierung der Eingabewerte. Sie sind in einem standardmäßigen KNN als adaptive Parameter charakterisiert, die die Eingabesignale x_i des künstlichen Neurons multiplizieren. Das KNN kann durch eine parametrisierte Funktion $f(x)$, auch Aktivierungsfunktion genannt, mathematisch veranschaulicht werden. [11] [12]

$$f(x_0,\ x_1,\ x_2, \ldots,\ x_n,\ w_0,\ w_1,\ w_2, \ldots,\ w_n)$$

Die Mengen an Eingabewerten x und zugehörigen Gewichten w werden als Vektoren repräsentiert. x_0 stellt hierbei die Bias-Eingabe $x_0 = 1,0$ dar. [11] [12]

$$\vec{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \vec{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

Im Falle eines SLP bestehen ausgehend von x_i direkte Verbindungen in Form von Gewichten zu den Neuronen des Ausgabe-Layers, die einen numerischen Ergebniswert z ausgeben. MLP besitzen im Gegensatz dazu zwischen den festen Eingabe- und Ausgabeneuronen einen oder mehrere HL, in denen der eigentliche Prozess der internen Informationsverarbeitung stattfindet. Sie sind ebenfalls über Verbindungsgewichte w_i zu ihren Vorgänger- und Nachfolger-Layern verknüpft. Der Wert des jeweiligen Ausgabeneurons z errechnet sich folgendermaßen. [11] [12]

$$z = w_0 \cdot x_0 + w_1 \cdot x_1 + ... + w_n \cdot x_n = w^T \cdot x \tag{2.1}$$

Der Wert einer einfachen linearen Kombination der Eingabesignale in einem SLP ist charakteristisch für den Zustand des Neurons und kann mithilfe folgender gewichteter Summe $\hat{z}$ dargestellt werden. Hierbei wird der Schwellwert Θ zu der Netzeingabe (gewichtete Summe) addiert, um eine Verschiebung der gewichteten Eingaben zu erreichen. [11]

$$\hat{z} = \Theta + \sum_{n=0}^{n} w_i \cdot x_i \tag{2.2}$$

Aktivierungszustand Die Aktivierungsfunktion des Neurons wird in Form einer einfachen Sprungfunktion bestimmt, die entweder 0 (Neuron ist nicht aktiviert) oder 1 (Neuron ist aktiviert) ausgibt. Um das Ergebnis der Sprungfunktion $\phi(\hat{z})$ rechnerisch zu ermitteln, wird der Wert des Ausgabeneurons $\hat{z}$ einem Schwellenwert Θ, engl. *Threshold*, in Form einer Berechnungsvorschrift $\phi(\hat{z})$ gegenübergestellt. Im Vergleich zu dem Beispiel aus den Abbildungen 2.1 und 2.2 wird eine binäre Entscheidung getroffen. [11] [12]

$$\phi(\hat{z}) = \begin{cases} 1 & wenn\ \hat{z} > \Theta \\ 0 & wenn\ \hat{z} \leq \Theta \end{cases} \tag{2.3}$$

Im Unterschied zum einschichtigen Perzeptron wird bei der Verwendung mehrschichtiger KNN eine komplexere Aktivierungsfunktion, beispielsweise Sigmoid, Tanh, ReLU oder Softmax, anstatt der elementaren Sprungfunktion für die Bestimmung des Ausgabewertes verwendet. [13]

Training Der eigentliche Lernprozess innerhalb eines KNN erfolgt nach der Initialisierung durch die optimale Anpassung der Gewichte w_i zwischen den Layern. Bei einem Perzeptron ist der Trainingsprozess binär und daher vergleichsweise trivial. Die Ausgabe ist begrenzt auf 0 oder 1. Zunächst werden alle Gewichte w_i mit dem Wert 0 versehen. In jeder Trainingsiteration wird zu Beginn der Ausgabewert $\hat{y} = \phi(\hat{z})$ berechnet. Anschließend wird dieser mit dem tatsächlichen Ergebnis y verglichen und daraufhin alle Gewichtungen mithilfe der Formel $w_i = w_i + \Delta w_i$ aktualisiert. Die Berechnung der dafür notwendigen Gewichtsanpassung Δw_i zur Minimierung des Ausgabefehlers geschieht wie folgt: $\Delta w_i = (\hat{y}_i \check{} y_i) \cdot x_i$ [12]

In der Praxis werden häufig Mehrfachklassifikationen der Art „One-vs-All (OvA)“ angewendet, beispielsweise Bildklassifikation auf der Basis des MNIST-Datensatzes. Hierbei steht die Ausgabe 1 für die korrekte Erkennung der Klasse. Die Bildklassifizierung erfordert im Rahmen der Klassenzuordnung Perzeptron-Netze der Größe $n = 10$, wobei jedes Netz auf das Ermitteln einer konkreten Klasse trainiert wird. [12]

Gradient Descent Die Idee hinter der Deltabildung von Modellgewichten wird vereinfacht, anhand des in Abbildung 2.3 dargestellten Optimierungsalgorithmus *Gradient Descent*, erläutert. Die Gewichte werden während des Modelltrainings mit Beginn des initialen Gewichts w_1 (engl. *initial weight*) in jedem Lernschritt (engl. *learning step*) so optimiert, dass die abgebildete Kostenfunktion $f(x)$ minimal wird. Sobald das globale Minimum y_{min} im Zuge der Kosten erreicht ist, wird das Modelltraining beendet. Das bedeutet gleichermaßen, dass die Hyperparameter ideal auf den jeweiligen Anwendungsfall eingestellt sind. Aus der Grafik ist ebenfalls ersichtlich, dass sich die Delta-Werte Δx_i pro Lernschritt immer weiter verringern, bis irgendwann Delta-Null bei der optimalen Parameterkombination erreicht ist. [14, S. 30]

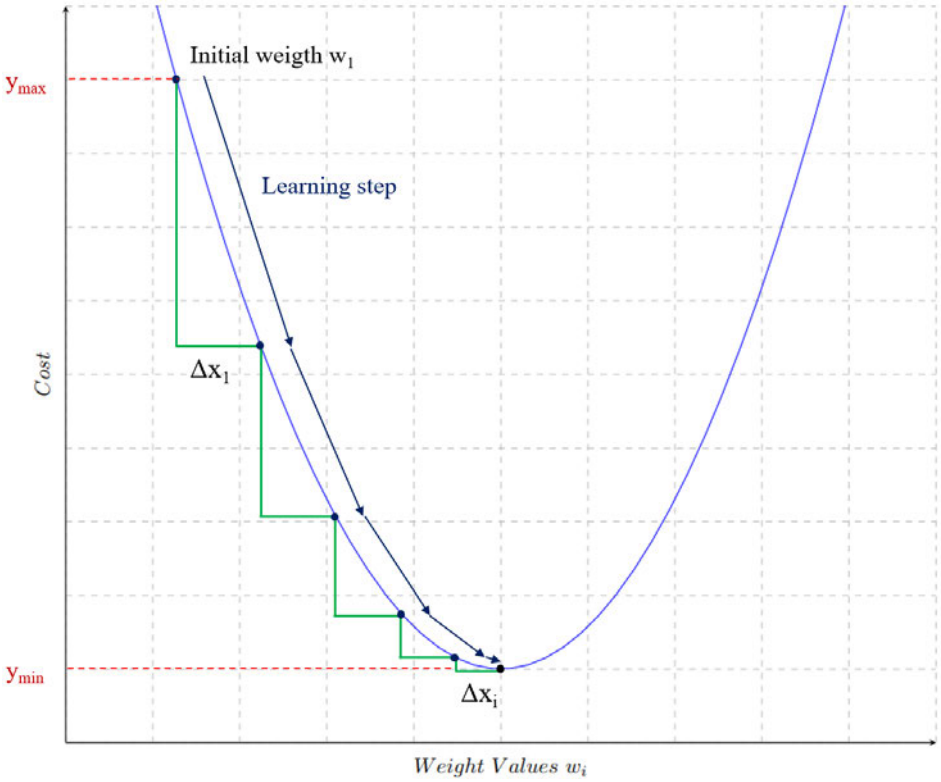


Abbildung 2.3: Der Graph visualisiert die schematische Deltabildung Δx_i anhand des Gradientenabstiegs, angelehnt an [14, S. 30]. Hierbei ist die Kostenfunktion in blau, die Deltas in grün und der Minimal- und Maximalwert innerhalb des Wertebereichs $W = [y_{min}, y_{max}]$ in rot dargestellt.

2.2 Herausforderungen und Angriffe auf Systeme der Künstlichen Intelligenz

Die Herausforderungen im praktischen Umgang mit KI und den daraus abgeleiteten Angriffspotenzialen für KI-Systeme verdeutlichen nachfolgend die Notwendigkeit der Auditierung.

2.2.1 Allgemeine Herausforderungen der Künstlichen Intelligenz

KI dient als Grundlage der Entscheidungsfindung und Steuerung für Systeme zahlloser Anwendungsbereiche. Insbesondere im Bereich der cAI treten neuartige Schwachstellen auf, deren Auditierung anhand von aktuellen Technologien der *classical IT* (cIT) nicht ausreichend möglich ist. Der momentane Trend, verantwortliche und vertrauliche Entscheidungen unter anderem auch in systemkritischen Bereichen einem KI-System zu übertragen, kann bei fehlender Funktionalität schwerwiegende Konsequenzen nach sich ziehen. Um solche Folgen zu vermeiden, besteht die größte Herausforderung darin, eine Interpretation durch den Menschen zu ermöglichen und somit das mangelnde Vertrauen wiederzugewinnen. Hierfür muss die Funktionsweise der KI ordnungsgemäß sichergestellt und tiefgründig untersucht werden, um zu erkennen, wann eine KI gut oder schlecht performt. Damit die Erklärungen nachvollziehbar sein können, muss der KI-Betrieb unter vordefinierten Randbedingungen stattfinden. Diese Garantien sollen vergleichbar zu den cIT Standards sein. Somit leitet sich die Fragestellung ab, auf welche Art und Weise eine Übertragung bestehender Methoden von cIT auf den Bereich der KI stattfinden kann. Angreifer können die aktuell fehlende Audtierbarkeit ausnutzen, da es an Vertrauen durch die Anwender mangelt und die Systemleistung zu sehr von der Datenmenge sowie deren Qualität abhängt. Dieses Risiko muss im Hinblick auf Systemausfälle sowie böswillige Angriffe analysiert werden. [7, S. 4 f.]

Die Lösung einer KI-Problemstellung ist abhängig von dessen Komplexität und der verfügbaren Datenmenge und wird durch verschiedene Paradigmen, wie das *Deep Learning* oder klassische Methoden, umgesetzt. Die Umgebung des Problems ändert sich über die Zeit, sie ist nicht konstant. Eine große Herausforderung besteht darin, trotz Umgebungsänderungen belastbare Ergebnisse zu erzielen. Beim Umgang mit Umweltveränderungen kann das *Transfer Learning* angewendet werden. Bei dem klassischen Ansatz würden jedoch Robustheitsgarantien auf Grundlage der statistischen Lerntheorie verwendet werden, die unter standardisierten Annahmen abgeleitet werden. Alternativ kann die Vorhersagegenauigkeit durch das Ablehnen von Dateneingaben geschehen. Hierbei werden nur die Datenpunkte abgelehnt, deren Entfernung zu bekannten Daten zu groß ist, da sich die Modellsicherheit durch zu große Entfernungen verringert. Die korrekte Identifizierung dieser Datenpunkte gestaltet sich nicht trivial. Der Ansatz kann bei dem *Online Learning*, auch kontinuierliches Lernen genannt, verwendet werden. Dabei wird ein sich konstant aktualisierender Datenstrom verarbeitet, der jeden Datenpunkt berücksichtigt. Die daraus abgeleiteten Herausforderungen werden in Kapitel 2.5 aufgearbeitet. [7, S. 9]

2.2.2 Angriffsarten auf Künstliche Intelligenz

Die Verwendung von Künstlicher Intelligenz als festen Bestandteil des alltäglichen Lebens birgt eine Reihe von Sicherheitsrisiken und Angriffspotenziale für Angreifer*innen. Vor allem in Unternehmen haben die Ergebnisse von KI-Systemen eine starke Auswirkung auf deren Entscheidungsfindung.

Diesen Umstand nutzen gegnerische Parteien aus, um die Verfügbarkeit und die Integrität eines unbeaufsichtigten KI-Systems zu gefährden. Es werden drei Arten der Gefährdung unterschieden: *Evasion Attacks*, *Data Poisoning Attacks* und Modelldiebstahl. [15]

Einteilung Besitzt der Angreifer einen umfangreichen Wissensstand über das Modell, die verwendeten Features und den Datensatz, ist von einer *White-Box* Attacke auszugehen. Ist im Gegensatz dazu das Wissen bezüglich des Ziel-Systems begrenzt, liegt ein *Grey-* oder *Black-Box* Angriff vor. Hat der Angreifer die Kontrolle über die Entscheidungen eines KI-Systems, wird der Angriff *Targeted Attack* genannt. Verändern sich die Entscheidungen in einer willkürlichen Art, wird von einer *Untargeted Attack* gesprochen.

Evasion Attacks Bei Ausweichangriffen versuchen Angreifer die korrekte Erkennung zu umgehen, indem sie fortwährend Klassifikatoren mit neuen, gegnerischen Eingaben (engl. *Adversarial Inputs*) versorgen, die eine möglichst geringe Abweichung von den gewohnten Trainings- und Testdaten aufweisen, gleichzeitig jedoch die Verlustfunktion des KI-Modells maximieren sollen. Oftmals werden iterative und auf einem Gradienten basierende Verfahrensweisen angewendet. Beispielsweise kann, wie in Abbildung 2.4 dargestellt, das Ergänzen einer gegnerischen Eingabe den Klassifikator dazu bringen, das Abbild eines Pandas fälschlicherweise als „Gibbon“ einzuordnen [16]. [15] [4, S. 110 ff.]

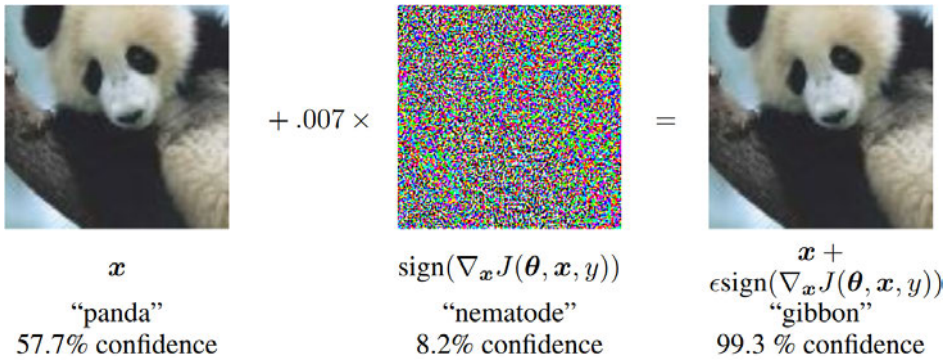


Abbildung 2.4: Die Darstellung verdeutlicht das Einfügen eines Vektors multipliziert mit dem Wert 0.07 zu dem Eingabebild x „panda“. Nach der ersten Iteration wird dem Bild mit dem Label „panda“ eine Wahrscheinlichkeit von 57,7 % zugeordnet. Das Addieren der unmerklich kleinen, gegnerischen Eingabe verursacht die Klassifikation des Labels „gibbon“ mit einer Wahrscheinlichkeit von 99,3 %. [16]

Eine Sonderform der Evasion Attacks sind **kontradiktorische Beispiele**. Sie stellen Eingaben für maschinelle Lernmodelle dar, die die Absicht verfolgen, gezielt Fehler zu verursachen. Autonomes Fahren kann beispielsweise kompromittiert werden, indem gezielt Aufkleber oder eine bestimmte Farbgebung zur Erzeugung eines gegnerischen Stoppschilds appliziert werden. Das Straßenschild wird dann durch das Fahrzeug fehlinterpretiert und das resultierende Verhalten des Fahrzeugs ist nicht verkehrsgerecht. [17]

Data Poisoning Für eine hohe Modell-Performance werden große Datenmengen aus vielen verschiedenen Ressourcen benötigt. Diese besitzen oftmals einen geringen Qualitätsstandard, wodurch Probleme während des Trainingsprozesses entstehen. Unbeabsichtigtes Verhalten wird dabei verursacht, weil KI-Modelle aktuell meist Korrelationen aus fehlerhaften Daten extrahieren. [7, S. 12]
Beim Data Poisoning (dt. Datenvergiftung) werden konkret verunreinigte Daten in das System eingeschleust, mit dem Ziel, falsche Klassifikationen zu erzeugen. Dadurch verwischt die Abgrenzung

zwischen guten und schlechten Klassifizierungen und es resultiert eine Bedrohung der Modellintegrität. Es treten zwei Arten des Data Poisoning auf. Zum einen können Falschinformationen in das System eingeschleust werden und zum anderen kann eine Hintertür engl. *Backdoor* generiert werden. Ein prominentes Beispiel für Falschinformationen sind sogenannte *Deepfakes* oder *Fake News*. Modellverzerrungen stellen hierbei die häufigste Art der Data Poisoning Angriffe dar. Schlechte Eingaben werden dabei durch den Klassifikator als gut eingestuft. [15] [18]

Backdoor Attacks *Backdoor Attacks* gehören zu den Data Poisoning Angriffen. Hierbei werden Trainingsdaten gezielt verfälscht, um die Generalisierungsfähigkeit des Modells zu beeinträchtigen, indem die Entscheidungsgrenze durch die Eingabe falscher Daten verschoben wird. *Backdoor Poisoning* beeinträchtigt die Integrität der Daten und ist im Nachhinein schwer erkennbar. Ohne den Nachweis ihrer Identität erhalten Angreifer hierbei Zugriff auf die Modelleingaben. Das bedeutet sie gehen im übertragenen Sinne durch eine versteckte Hintertür in das System. Der Angreifer erschafft diese Hintertür durch das Einbringen bestimmter Neuronen in das Neural Network (NN). Werden diese manipulierten Eingaben dann vom System beurteilt, können die evaluierten Informationen vom Angreifer eingesehen werden. [7, S. 12] [19]

Modelldiebstahl Modelldiebstahl (engl. *Model Stealing* oder *Model Extraction Attacks*) zählt zu den besorgniserregendsten Szenarien, ist aber im Vergleich zu den vorangegangenen Angriffen unwahrscheinlich, da eine Schnittstelle zum Modell benötigt wird. Diese wird verwendet, um Eingabe- und Ausgabedaten zu aggregieren. Auf Grundlage der gestohlenen Trainingsdaten soll das ursprüngliche Modell oder Informationen diesbezüglich rekonstruiert und verbessert werden. Hierbei wird besonders auf sensible Daten abgezielt. Dazu zählen beispielsweise geheime Transaktionen aus dem Finanzbereich, Krankenakten oder generell Benutzungstransaktionen. Diese Informationen können weiterhin verwendet werden, um Evasion Attacks mittels eines Substitution-Modells zu erzeugen. [15] [4, S. 110 ff.]

Inference Attacks Inferenzangriffe, auch *Privacy Attacks* genannt, ermöglichen den Zugriff auf Informationen über die Trainingsdaten des ML-Modells. Sie können in die vier Arten Modellinversionsangriffe, *Membership Inference Attacks*, *Attribut Inference Attacks* und *Database Reconstuction Attacks* unterschieden werden. [20]
Bei Modellinversionsangriffen (engl. *Model Inversion Attacks*) wird der Modellzugang missbraucht, um Informationen über die zugrundeliegenden Trainingsdaten des KI-Systems abzuleiten. Die Trainingsdaten beinhalten oftmals datenschutzrelevante Informationen, die es zu schützen gilt. Obwohl diese Angriffsart bisher nur bei einfachen KI-Modellen (zum Beispiel Regression) erfolgreich war, ist das zukünftige Gefahrenpotenzial, beispielsweise durch den generativen Modellinversionsangriff (engl. *generative model-inversion attack*) auf DNNs von [21], nicht zu vernachlässigen. [21]
Membership Inference Attacks zeichnen sich dadurch aus, dass Angreifer, die Zugriff auf das trainierte Modell besitzen, testen, ob einzelne Datenproben Teil der Trainingsdaten sind. Durch die Rekonstruktion der verwendeten Datenstichproben kann die Mitgliedschaft abgeleitet werden und hochsensible Informationen, die die Privatsphäre von Personen beeinträchtigen, werden ggf. offengelegt. [22]
Bei Attribut Inference Attacks werden gezielt sensible Attribute von Personen durch einen Angreifer adressiert. Dies wird ermöglicht durch das Vorhandensein von Teilwissen über den Trainingsdatensatz. Um konkrete Attributwerte, die bisher nicht bekannt sind, mit hoher Wahrscheinlichkeit vorhersagen zu können, werden bekannte Attribute (zum Beispiel personenbezogene Daten) in den Trainingsprozess des KI-Modells integriert. Das Modelltraining wird so lange optimiert, bis die Leistungsfähigkeit

des KI-Modells bestmöglich ist. [23]
Das Ziel einer Database Reconstruction Attack ist die Wiederherstellung der dem Modelltraining zugrundeliegenden Trainingsdatenbank. [24]

2.2.3 Abwehrmöglichkeiten bekannter Angriffe auf Künstliche Intelligenz

Die in Kapitel 2.2.2 dargestellten, allgemeingültigen Angriffe auf KI-Systeme werden im Folgenden hinsichtlich deren Abwehrmöglichkeiten betrachtet. Die Angreifer weisen trotz vielschichtiger Verteidigungsstrategien oftmals ein adaptives Verhalten auf. Hierzu existieren neue Ideen in Form von zertifizierten Erkennungen, die Robustheit gegen adaptive Angriffe versichern. Da die Verteidigungsmethoden die Leistung beeinträchtigen, ist eine Verwendung von Evaluationsmetriken notwendig. Diese wägen den Trade-Off zwischen Modellleistung und Verteidigungsmaßnahmen bei harmlosen oder gegnerischen Eingaben ab, wobei die Bedingungen der Umgebung sowie der Robustheit des Systems einbezogen werden. [7, S. 13 f.]

Evasion Attacks Die korrekte Erkennung sowie das Einleiten zuverlässiger Schutzmaßnahmen gegen Evasion Attacks, auch *Adversarial Attacks* genannt, ist überaus wichtig. Das Erkennen gestaltet sich in der Praxis jedoch sehr schwierig, da gegnerische Eingaben zu subtil sind, um durch die menschliche Beobachtung wahrgenommen zu werden. [7, S. 13 f.] [19]
Ein in der Literatur vielversprechender Ansatz für eine erhöhte Robustheit gegen gegnerische Eingaben stellt das *Adversarial Training* dar, das kontradiktorische Beispiele in die Trainingsphase einbindet. Nachteilig ist hierbei jedoch die erhöhte Laufzeit, die fehlende Adaptivität zu unbekannten gegnerischen Eingaben und dass die Robustheit dadurch nicht formal garantiert werden kann. Um effizienteres kontradiktorisches Training mit starken Eingaben durchzuführen, können die Ansätze des *Shared Adversarial Training* [25] und des *Meta Adversarial Training* [26] angewendet werden. [7, S. 13 f.] [19]
Die Problematik, dass immer Angriffe existieren, die bestehende Vereidigungen durchbrechen können, wird mit dem Bedrohungsmodell *Adversarial Patches* adressiert [27]. Hierbei werden zertifizierte Abwehrmechanismen angeboten [28][29]. Das *Gradient Masking* verhindert den externen Zugriff auf Informationen über den Gradienten des KI-Modells. Auf der Grundlage eines Modells wird bei der *Defensive Distillation* ein neues KI-Modell antrainiert, mit dem Ziel, wahrscheinlichkeitsbasiert glattere Entscheidungsgrenzen zu erzeugen und die Genauigkeit zu erhöhen. Bei *Ensemble Methods* werden Ausgaben von verschiedenen Modellensembles miteinander kombiniert. Weiterhin kann bei einem Angriff der Gewinn an Informationen durch die Integration von Zufall in den Datensatz oder das Modell verringert werden. Alternativ können Eingaben bei dem *Feature Squeezing* mithilfe von Kompressions- oder Glättungsoperatoren transformiert werden, um Manipulationen wirkungslos zu machen. Ein sogenannter *Reformer* wird vor das KI-Modell geschaltet, wodurch die Eingabeinformationen zu nahegelegenen Datenpunkten des Trainingsdatensatzes verschoben werden. Außerdem können KI-Systeme auf der Basis vollständig verschlüsselter Datensätze arbeiten. Dieser Ansatz wird *Fully Homomorphic Encryption* genannt. Bei der *Input Reconstruction* (dt. Eingabe-Rekonstruktion) werden die Eingabedaten transformiert, ohne dass die Modellfunktion beeinträchtigt wird. Dies geschieht unter Anwendung von Rauschen, Rauschunterdrückung, Datenvorverarbeitung, Maskierung des Gradienten sowie Autoencodern, die die Eingabedaten verändern. Die Methode der *DNN Verification* überprüft die Eigenschaften eines DNN-Modells mithilfe sogenannter *Solver*. Die Datenerweiterung, engl. *Data Augmentation*, erweitert den Datensatz mit gegnerischen Beispielen, um eventuelle Verluste auszugleichen und effektives Modelltraining sicherzustellen. [7, S. 13 f.] [19] [4, S. 110 ff.] [17]

Alternative Verteidigungen, wie beispielsweise *Gradient Obfuscation* [30], entsprechen nicht den Robustheit-Anforderungen und können durch ein irrtümliches Sicherheitsempfinden ein zusätzliches Angriffspotenzial darstellen. [7, S. 13 f.]

Data Poisoning Um *Data Poisoning* Angriffe abzuschwächen, ist es notwendig, kontinuierlich und in Echtzeit die Rechte einer jeden Nutzer*in sowie die Modell-Eingaben zu überwachen und die Trainingsdaten zu filtern. Mithilfe geeigneter Statistik-Methoden können manipulierte Daten identifiziert und anschließend eliminiert werden. Eine kontinuierliche Beobachtung und Kontrolle des Modellverhaltens können ebenfalls zum Feststellen von Unterschieden bezüglich legitimer und manipulierter Daten führen. Die Regressionsanalyse (engl. *Regression Analysis*) dient der Erkennung von Rauschen sowie Ausreißern in Trainingsdatensätzen. Beim *Ensemble Learning* kommt es zur Erstellung und Kombination mehrerer ML-Klassifikatoren, wodurch die Robustheit gegenüber *Data Poisoning* gestärkt wird. Bei der Methode des *Iterative Retraining* werden neuronale Netze iterativ mit Gegenbeispielen neu trainiert. [15] [18] [4, S. 110 ff.] [19]

Backdoor Attacks Bei Backdoor-Angriffen gestaltet sich die fehlende Vorkenntnis der KI in Bezug auf die Zieldomäne problematisch. Um diese Wissenslücke zu umgehen, können Daten bösartiger Natur durch sorgfältiges Anschauen des Inneren eines neuronalen Netzes identifiziert werden. Sobald Datensätze mit einem andersartigen Verhalten im Vergleich zu anderen Datenstichproben der gleichen Kategorie auftreten, können sie mittels *Input Preprocessing* innerhalb des Netzwerkes identifiziert werden. Sie geben demnach ein Indiz für die Verwendung anderer Merkmale, um Vorhersagen zu treffen. Die Strategie des Ansehens des Netzwerkinnenlebens funktioniert in der Praxis nur bedingt. Um die Problematik der fehlenden Modell-Priorisierung zu lösen, kann Expertenwissen des Menschen als eine Art Hybridlösung in den Prozess interaktiv integriert werden. [7, S. 14]
Aufgrund des Mangels an menschlicher Interpretation können Angriffe mithilfe einer Ausreißererkennung aufgedeckt werden. Hierzu werden interne Modellrepräsentationen in Kombination mit XAI-Methoden, siehe Kapitel 2.3, verwendet. Die existierenden Minderungstechniken performen nicht perfekt, da es keine automatische Lösung darstellt und müssen in Zukunft so konzipiert werden, dass Störkorrelationen adressiert und Backdoor-Angriffe verhindert werden. Hierbei ist menschliches Vorwissen erforderlich, um beschädigte und gutartige Daten voneinander unterscheiden zu können. Um eine Problemlösung zu finden, müssen falsche Korrelationen mithilfe von XAI-Methoden aus den Trainingsdaten entfernt werden. Außerdem können Zufallstechniken in Bezug auf Pipeline-Artefakte während des Trainings von Nutzen sein. Zum Schutz vor Backdoor-Angriffen müssen zu Beginn des Lernprozesses Sicherheitsüberprüfungen stattfinden und der Zugriff auf die Datenspeicherung und Entwicklungsmaschinen limitiert werden. Außerdem können Neuronen, die die Backdoor darstellen, mithilfe des *Model Pruning* entfernt werden. [7, S. 12] [19]

Modeldiebstahl Gegen Modelldiebstahl können drei Verteidigungsstrategien angewendet werden: *Privacy Aggregation of Teacher Ensembles* (PATE), *Differential Privacy* und *Model Watermarking*. Bei PATE werden die Ausgangsdaten in mehrere Datensätze aufgeteilt, die zum Trainieren für verschiedene unabhängige KI-Lehrer-Modelle dienen. Anhand dieser Ausgaben werden Studenten-Modelle trainiert, ohne dass die Vertraulichkeit und Privatsphäre der Trainingsdaten gefährdet wird, da die Studenten-Modelle keine Informationen zu den ursprünglichen Daten besitzen. Bei der Methode *Differential Privacy*, dt. differenzielle Vertraulichkeit, wird ein Rauschen zu den Trainingsdaten oder der Phase des Modelltrainings addiert, um die Privatsphäre der Daten zu schützen. Alternativ kann bei der

Methode des *Model Watermarking* eine spezielle Kennzeichnung in das originale Systeme eingebettet werden. Dadurch können ähnliche und ggf. manipulierte Modelle von dem Original unterschieden werden. [19]

Inference Attack Zum Schutz gegen Inferenzangriffe bietet sich die Methode des *Data Sanitization* an, wobei sensible sowie personenbezogene Informationen dauerhaft aus Datensätzen gelöscht werden. Dies geschieht ohne die Möglichkeit der Wiederherstellung von Informationen. Weiterhin kann ebenfalls die zuvor beschriebene Verteidigungsstrategie *Differencial Privacy* angewendet werden. [31]

Zusammenfassung Die Tabelle 2.1 fasst die erläuterten Abwehrmechanismen zu Angriffen auf KI-Systeme übersichtlich zusammen. Spezifische Lösungsansätze für die Angriffe auf KI-Systeme sind in der Fachliteratur bekannt. Jedoch konnten diese Ansätze bisher nicht verallgemeinert werden und erfordern weitergehender Forschung. [15] [19]

Tabelle 2.1: Zusammenfassung der Abwehrmechanismen von Angriffen auf KI-Systeme, erweitert nach [19]

Angriffsart	Datensammlung	Trainingsphase	Anwendungsphase
Evasion Attack	Adversarial Example Detection, Ensemble Methods, Zufall, Feature Squeezing, Einsatz eines Reformers, Fully Homomorphic Encryption, Data Augmentation	Adversarial Training, Network Distillation, Zufall	Erkennung von Adversarial Examples, Input Reconstruction, DNN Verification, Ensemble Methods
Poisoning Attack	Filtern der Trainingsdaten, Regressionsanalyse, Iterative Retraining	Integrationsanalyse	Ensemble Learning
Backdoor Attack	N/A	Model Pruning	Input Preprocessing
Modell Stealing Attack	Differential Privacy	PATE, Model Watermarking	N/A
Inference Attack	Data Sanitization, Differencial Privacy	N/A	N/A

2.3 Erklärbare Künstliche Intelligenz

Anhand der Erläuterung von Black-Box-Modellen wird im Folgenden der Ausdruck „Erklärbare KI“ (engl. *Explainable Artificial Intelligence*) vorgestellt und welchen Einfluss dieser auf die Auditierbarkeit von KI-Systemen birgt.

2.3.1 Grundlagen von Erklärbarer Künstlicher Intelligenz

Das Innenleben komplexer KI-Modelle wird charakterisiert durch Funktionen, die anhand großer Trainingsdatensätze lernen und anschließend mathematisch kodiert werden. Zu den Gründen einer Interpretation der Entscheidungen eines KI-Modells gehören das Finden von Fehlern, Schwachstellen und Beschränkungen. Die Zielstellung ist eine verbesserte Leistung und Robustheit gegenüber Angriffen zu generieren, um im Idealfall die in der EU-Datenschutz-Grundverordnung (DSGVO) vorgeschriebene Transparenz zu gewährleisten. Das Forschungsfeld zur Erklärung komplexer KI-Systeme, wie beispielsweise neuronale Netzwerke, nennt sich *Explainable AI* (XAI) und ist stark mit dem Forschungsgebiet *Auditable AI* verbunden, wobei Auditable AI eine Teilmenge von XAI ist. Die verschiedenen Methoden für Erklärungsversuche streben unter anderem eine globale Interpretierbarkeit an, die wertvolle Informationen über das Modell bereitstellt. Die Interpretation wird beispielsweise durch die Analyse von Extremalpunkten der codierten Funktion oder durch Untersuchungsansätze bezüglich des Einflusses einzelner Neuronen in einem DNN realisiert. Bei den globalen Ansätzen ist es jedoch nicht möglich, individuelle Vorhersagen zu treffen, die die positiven oder negativen Auswirkungen auf die Eingabemerkmale identifizieren. Um diese Problematik zu lösen, werden lokale XAI-Methoden angewendet, die jedem Merkmal der Eingabe eine jeweilige Bewertung der Relevanz zuordnet. Hierbei wird in die folgenden drei Ansätze unterschieden: [7, S. 17 f.]

- **Störungsbasierte Methoden** (engl. *perturbation-based methods*) stellen die Evaluation der Modellausgabe bereit, nachdem unendlich viele oder relativ grobe Störungen auf die eingegebenen Daten appliziert wurden. Aus den resultierenden Änderungen werden Erklärungen für das jeweilige Verhalten abgeleitet. Die vergleichsweise einfache Anwendung der Methode erfordert eine hohe Rechenleistung und zuverlässige Auswertung der abgeleiteten Verhaltensweisen. [7, S. 17 f.]
- **Ersatzbasierte Methoden** (engl. *surrogate-based methods*) zeichnen sich durch eine große Abfrage von vielgestaltigen Eingaben aus, die anschließend durch ein einfacheres Modell angenähert werden. Dieses Modell ermöglicht eine Interpretation des Entscheidungsverhaltens auf der Grundlage des Ursprungsmodells. Hierbei ist ebenfalls ein hoher Rechenaufwand erforderlich. Außerdem besteht eine deutliche Abhängigkeit der Erklärungen von den Abfragen der Eingabedaten und der daraus folgenden Anpassung des Modells. Zu den ersatzbasierten Methoden gehört beispielsweise Local Interpretable Model-agnostic Explanations (LIME) [32]. [7, S. 17 f.]
- **Strukturbasierte Methoden** (engl. *structure-based methods*) verwenden die interne Netzwerkstruktur zur Verbreitung relevanzbasierter Informationen zwischen den Ein- und Ausgabedaten von Netzwerkschichten. Obwohl deutlich weniger Rechenleistung erforderlich ist, ist jedoch ein Zugriff auf die interne Struktur notwendig. Im Vergleich zu den beiden anderen Methoden liefert dieser Ansatz die umfassendsten Erklärungen für das Modellverhalten. Die Methode Layer-Wise Relevance Propagation (LRP) [33] ist ein typischer Vertreter der strukturbasierten Methoden und wird näher in Kapitel 2.3.5 betrachtet. [7, S. 17 f.]

Durch das Zusammenfassen mehrerer lokaler XAI-Erklärungsmethoden kann ein umfassenderes Gesamtbild der Vorhersagestrategien eines KI-Modells generiert werden. Der LRP-Ansatz kann beispielsweise verwendet werden, um Datenverzerrungen zu detektieren und die grundlegende Fähigkeit zur Verallgemeinerung zu verbessern. Zu den Herausforderungen gehört eine Art Interpretationslücke, die entsteht, wenn die eingegebenen Merkmale durch Menschen ebenfalls nicht interpretierbar sind. Weiterhin wird der Einfluss durch mögliche Wechselwirkungen zwischen mehreren Eingaben nicht in die Interpretation einbezogen. [7, S. 17 f.]

2.3.2 Black-Box Modelle

Erklärbare KI KI-Systeme werden oftmals in der Literatur erst dann als erklärbar bezeichnet, wenn die externen Eigenschaften einer KI-Lösung ebenfalls berücksichtigt wurden. Diese externen Merkmale hängen hierbei sowohl von den Eigenschaften der KI als auch der Nutzer*innen selbst ab. Zu den externen Merkmalen gehören unter anderem Vertrauen, Ethik, Sicherheit und Fairness. Diese Aspekte unterliegen den gesellschaftlichen Normen und der Evaluation der Nutzer*innen. [34]

Vertrauenswürdigkeit Die Entscheidungen einer KI können vielfältige Auswirkungen auf wichtige Lebensbereiche des Menschen haben. Beispielsweise kann sie die Entscheidungsfindung im Finanzbereich, im sicherheitstechnischen Bereich oder bei persönlichen Angelegenheiten steuern. Basieren diese Entscheidungen möglicherweise auf unethischen Gründen, stellt sich die Frage, wer das KI-System zur Rechenschaft zwingt. Daher ist es vor allem bei hochkritischen Situationen und ohne tiefgreifende Fachkenntnis schwierig, deren Arbeitsweise blind zu vertrauen. Wünschenswert wäre demnach die vollständige Vertrauenswürdigkeit eines KI-Systems zu etablieren. Hierbei sollte die KI idealerweise automatisch unter ethischen und moralischen Standards bewertet werden. Um dieses Ziel erreichen zu können, ist eine detaillierte Erklärung bezüglich der Art und Weise der Entscheidungsfindung des KI-Systems zwingend notwendig. [34]

Interpretierbarkeit Viele Anwender müssen trotz dieser Überlegungen immer noch auf die Richtigkeit der KI-gestützten Ergebnisse vertrauen, da die fachliche Kompetenz nicht ausreichend vorhanden ist. Es muss demnach eine für den Menschen verständliche Erklärung der Funktionsweise und der Ausgabe bereitgestellt werden, die anhand rationaler Argumente bewertet wird. Hierbei muss die Konfliktfreiheit bezüglich ethischer und gesetzlicher Normen gewährleistet werden. Eine solche Erklärung erfordert jedoch immer eine von Menschen gesteuerte Nachbereitung im Zuge der Interpretation. Eine weitere Schwierigkeit besteht in dem Aspekt, dass Interpretierbarkeit innerhalb der Literatur nicht einheitlich definiert wird. [34]

Kriterien für Interpretierbarkeit Damit ein ML-Modell als erklärbar und somit als interpretierbar gelten kann, muss das System grundsätzliche Eigenschaften erfüllen. Zunächst muss das Kriterium der Simulierbarkeit erfüllt sein. Die Nutzer*innen von Erklärungsverfahren sollen unter Zuhilfenahme identischer Eingabedaten und Modellparameter in einem angemessenen Zeitraum die gleiche Ausgabe, das heißt die gleichen Erklärungen, erhalten. Hierbei muss jeder einzelne Berechnungsschritt für jede Vorhersage nachvollziehbar gemacht werden. Das zweite Kriterium der Zerlegbarkeit beschreibt die intuitive Erklärbarkeit jedes Bestandteils des ML-Modells, inklusive Datenvorverarbeitungsschritten. Bei der dritten Eigenschaft, der algorithmischen Transparenz, wird die Nachvollziehbarkeit des angewendeten ML-Algorithmus vorgeschrieben. Sind alle drei Kriterien hinreichend erfüllt, kann von einem interpretierbaren Modell, auch White-Box-Modell genannt, gesprochen werden. [35]

Ziele von XAI KI-Systeme weisen in der Regel eine verschachtelte und nicht lineare Struktur auf. Da keine Informationen über die Erstellung der Vorhersagen vorliegen, werden sie in der Fachwelt als eine Art Black-Box angesehen. Der größte Nachteil besteht in der mangelnden Transparenz. Schwierigkeiten treten vor allem in kritischen Umgebungen, wie der medizinischen Diagnostik oder beim autonomen Fahren, auf. Dem Anwender fällt es demnach schwer, der KI standardmäßig zu vertrauen, wodurch eine Validierung durch den Menschen notwendig wird. Die Entscheidungsfähigkeit ist demnach die Schnittstelle zwischen dem KI-System und der Gesellschaft. Die Herausforderung des Öffnens einer Black-Box erfährt demnach in der kürzlichen Vergangenheit eine große Aufmerksamkeit. Nach Samek et al. werden folgende Ziele für die Erklärbarkeit von KI-Systemen genannt:

- Verifizierung des Systems,
- Verbesserung des Systems,
- Lernen vom System und
- Einhaltung von Rechtsvorschriften. [33]

2.3.3 Einteilung von Erklärbarer Künstlicher Intelligenz

Auf Grundlage einschlägiger Literatur wurde laut Doran et al. in folgende drei Begriffe unterschieden: Bei undurchsichtigen Systemen (engl. *opaque systems*) ist für die Benutzer*innen nicht einsehbar, wie der Algorithmus innerhalb der KI die Eingaben zu den Ausgaben abbildet. Gewissermaßen beruhen diese Ansätze auf einer Art „Black-Box“, die keine Einsicht in die Implementation und Argumentation der KI geben. [34]
Die interpretierbaren Systeme (engl. *interpretable systems*) werden dadurch charakterisiert, dass die Nutzer*innen den mathematischen Algorithmus hinter der Ausgabenerzeugung eines KI-Systems untersuchen und verstehen können. Dieser Ansatz impliziert demnach eine umfassende Modell-Transparenz. Es ist jedoch notwendig, dass die Benutzer*innen bei der Untersuchung des algorithmischen Mechanismus eine fachspezifische Wissensbasis mitbringen. Oftmals ist die Funktionsweise von *Deep Learning* Netzen für den Großteil der Nutzer*innen aufgrund fehlender Expertise nicht interpretierbar. Beispielhaft für interpretierbare Systeme sind Regressionsmodelle, Entscheidungsbäume und SVM. [34]
Verständliche Systeme (engl. *comprehensible systems*) geben gemeinsam mit dem Output (verständlichere, nicht-mathematische) Symbole in Form von beispielsweise Wörtern oder Visualisierungen aus, die eine Verbindung zu den Eigenschaften der Eingabe herstellen. Die Nutzer*innen müssen wiederum aktiv werden und die Symbole und deren Beziehungen untereinander interpretieren. Die Schlussfolgerungen über das KI-System sind daher auf die Fachkenntnis der jeweiligen Person beschränkt und können somit nicht für Alle vergleichbar gemacht werden. Dieses abgestufte Verständnis wird weiterhin durch die individuelle und subjektive Betrachtungsweise und Interpretation der Nutzer*innen gestützt. Zu den verständlichen Systemen gehören zumeist mehrdimensionale Datenvisualisierungen. [34]
Diese Unterteilung suggeriert, dass die Kategorien der interpretierbaren und verständlichen Systeme eine Verbesserung gegenüber den undurchsichtigen Systemen sind. Jedoch kann auch ein verständliches System trotz ausgegebener Symbole ebenfalls undurchsichtig sein. Daher ist eine weitergehende Forschung in diesem Bereich sinnvoll. [34]

Unterscheidung lokal vs. global Die im Folgenden vorgestellten XAI-Methoden zielen darauf ab, transparente Vorhersagen für maschinelle Lernmodelle zu treffen. Hierbei wird anhand des Geltungsbereichs in lokale und globale XAI unterschieden. *Local XAI* bezieht sich auf die Interpretation einzelner Modellvorhersagen, wobei die Bedeutung der Eingabefunktionen anhand spezifischer Beispiele bewertet wird. *Global XAI* konzentriert sich im Gegensatz dazu auf ein allgemeines Verständnis der erlernten Modellmerkmale. Die Voreingenommenheit eines menschlichen Prüfers während der Analyse beeinträchtigt die Leistungsfähigkeit beider Enden des XAI-Spektrums. Für sich genommen sind die lokale und globale XAI für das Erkunden von Verzerrungen, falschen Korrelationen und falsch erlerntem Modellverhalten begrenzt anwendbar. Global XAI dient der Messung von Auswirkungen auf vordefiniertes oder erwartetes Merkmalsverhalten, wodurch das Erforschen von unbekanntem Modellverhalten beschränkt möglich ist. Im Vergleich dazu bietet Local XAI das Potenzial detaillierte Informationen pro Merkmal zu liefern. Anhand dieser könnten in der Theorie Informationen bezüglich des Modellverhaltens zusammengestellt werden. Dieser Prozess gestaltet sich jedoch sehr zeit- und rechenintensiv, da Informationen von zum Teil über Millionen Erklärungen aggregiert werden müssen. Weiterhin ist ein tiefergehendes Verständnis über die Domäne des Datensatzes notwendig. [33]

Unterscheidung modellspezifisch vs. modellagnostisch Methoden, die ein KI-System erklärbar machen, können entweder modellspezifisch oder modellagnostisch sein. Modellspezifische Verfahren konzentrieren sich hierbei auf die Herstellung der Erklärbarkeit für eine bestimmte Art von KI-Systemen. Im Vergleich können modellagnostische Methoden für unterschiedliche Modellarten angewendet werden. [36, S. 17]

Unterscheidung ante-hoc vs. post-hoc XAI Der Erklärungsansatz ante-hoc beschreibt die Entwicklung eines KI-Modells, bei der eine Interpretation inbegriffen ist. Die getroffenen Entscheidungen sollen direkt für Anwender*innen nachvollziehbar sein. Im Gegensatz dazu wird bei post-hoc Verfahren nach Beendigung der Anwendung der Entscheidungsprozess von Black-Box-Modellen erklärt. [36, S. 15]

2.3.4 Ergebnisse von Erklärbarer Künstlicher Intelligenz

Eigenschaften von Erklärungen Zu den Erklärungseigenschaften gehören die Genauigkeit, die Wiedergabetreue, die Konsistenz sowie die Verständlichkeit und Praxistauglichkeit. Vorhersagen müssen auch bei unbekannten Datensätzen genau sein. Das Modellverhalten muss nach post-hoc Erklärungen korrekt wiedergegeben werden, um anhand der Analyse der Erklärungen nützliche Schlussfolgerungen ziehen zu können. Die Eigenschaft der Konsistenz verdeutlicht die Fähigkeit, bei der gleichbleibenden Verwendung eines KI-Modells von ähnlichen Eingabedaten und -parametern auf ähnliche Ausgaben zu schließen. Weiterhin ist es von großer Bedeutung, dass die Erklärungen für Anwender*innen verständlich und praktisch anwendbar sind. [36, S. 20]

Ergebnisse von XAI-Verfahren Bei der Anwendung von XAI-Verfahren können verschiedene Ergebnisse zur Erklärung des Entscheidungsverhaltens resultieren. In Abbildung 2.5 sind die möglichen Ergebnisarten dargestellt. Die Merkmalsrelevanz beschreibt den Grad der Relevanz eines Merkmals für eine bestimmte Entscheidung oder eine Gruppe von Entscheidungen. Hierfür erfolgt die Zuweisung eines nummerischen Wertes zu jedem Merkmal. Datenpunkte als Erklärungen stellen Beispieldaten dar, die aus den Trainingsdaten resultieren. Die Texterklärungen erfolgen in natürlicher Sprache oder

textuellen Wenn-Dann-Regelungen. Modellinterna sind Parameter oder Bestandteile des zugrundeliegenden KI-Modells. Surrogatmodelle, auch Stellvertretermodelle genannt, werden anhand des zu erklärenden Modells neu generiert und dienen als Erklärungsgrundlage. [36, S. 19]

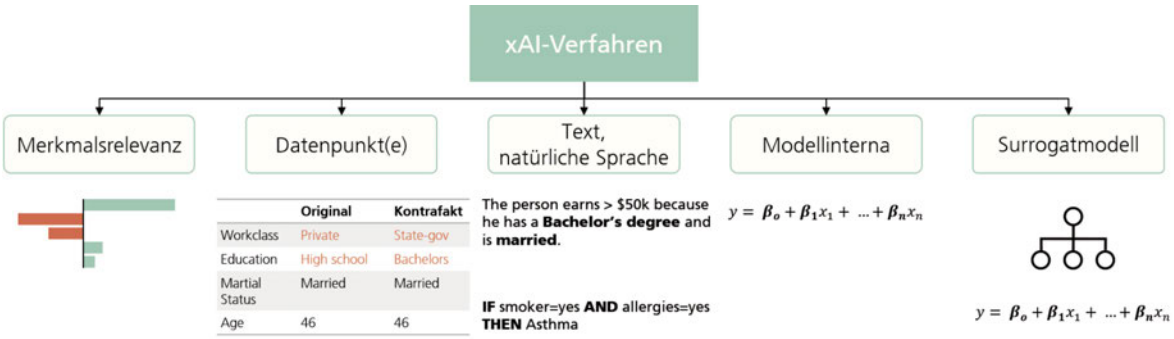


Abbildung 2.5: Dargestellt sind die unterschiedlichen Ergebnisse bei der Anwendung von XAI-Verfahren. Speziell können aus einem Verfahren Merkmalsrelevanzen, Datenpunkte, Text in natürlicher Sprache, Modellinterna oder ein Surrogatmodell resultieren. [36, S. 19]

Vergleich Heatmaps Eine *Heatmap* visualisiert den Relevanzwert jedes Pixels in einem Eingabebild. Hierbei wird auf der Grundlage einer Bewertungsfunktion jedem Pixel ein Wert zugewiesen [36, S. 29]. Die Heatmap kann als Überprüfung der Arbeitsweise eines KI-Systems dienen. Es kann nachvollzogen werden, wenn sich die relevanten Merkmale der ursprünglichen Entscheidungsfindung ändern. In diesem Fall würde eine Veränderung in der Heatmap des neuen Bildes im Vergleich zur Heatmap des Originalbildes sichtbar werden. [33]
In Abbildung 2.6 ist erkennbar, dass sich die Heatmaps verschiedener XAI-Methoden anhand ihres Detaillierungsgrades unterscheiden. Zum einen bieten die Methoden LRP [33], *Gradient x Input* [37] und *Integrated Gradients* genaue Visualisierungen jedes einzelnen Pixels, zum anderen generiert die Methode Gradient-weighted Class Activation Mapping (Grad-CAM) [38] eine grobe Lokalisierung der Relevanzwerte. Außerdem findet bei LIME [32] und KernelSHapley Additive exPlanations (SHAP) [39] eine Unterteilung in Superpixel statt, welche anschließend einem Relevanzwert zugeordnet wird. [36, S. 29]

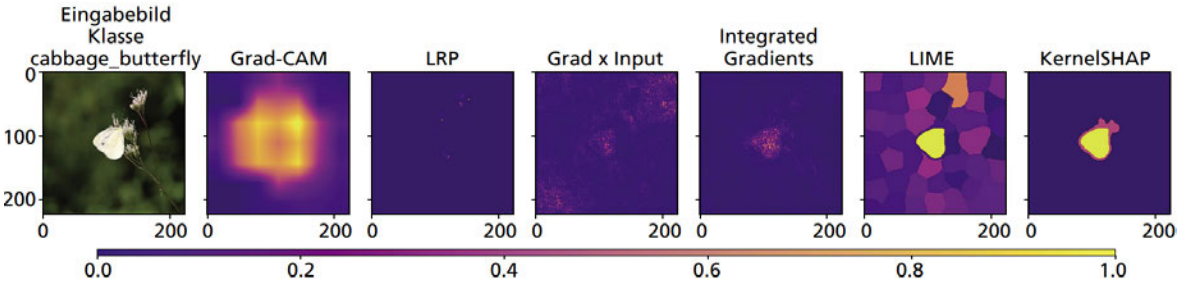


Abbildung 2.6: Die Darstellung vergleicht die Heatmaps des Eingabebildes „cabbage_butterfly“, welche durch die XAI-Methoden Grad-CAM, LRP, Gradient x Input, Integrated Gradients, LIME und KernelSHAP erstellt wurden. Die Relevanz jedes einzelnen Pixels wird durch eine Wertzuweisung von 0.0 bis 1.0 definiert, wobei 0.0 auf keine Merkmalsrelevanz hinweist und ein Wert von 1.0 entscheidungsgebend ist. Diese Relevanzzuweisung jedes Pixels ist in der Abbildung mithilfe des Farbverlaufs von violett zu gelb erkennbar. [36, S.29]

Evaluation Heatmap Damit die Heatmaps verschiedener Modell-Iterationen vergleichbar gemacht werden können, wird ein objektiver Maßstab für die Qualität der Modellerklärungen benötigt. Eingabestörungen verursachen bei Variablen mit wichtigen Vorhersagen einen größeren Rückgang der Ergebnisse als Vorhersagen mit geringerer Bedeutung. Aufgrund der Größe dieses Rückgangs kann auf die Wichtigkeit der Vorhersage einer Eingangsvariable geschlossen werden. Der durchschnittliche Rückgang kann somit als Objektivitätsmaß für die Qualität der Erklärungen dienen. Beispielsweise können die Vorhersageergebnisse nach jeder Störung nachverfolgt werden, wenn zuvor eine iterative Störung der Eingangsvariablen stattgefunden hat. Erfolgreiche Erklärungsmethoden weisen sich dadurch aus, dass bei relevanten Variablen ein starker Rückgang der Vorhersageergebnisse auftritt. [33] Eine zweite und einfachere Möglichkeit der Evaluationsbetrachtung liegt in der Sortierung der Eingabevariablen anhand ihres Relevanzwertes R_i. [33]

2.3.5 Anwendung von Erklärbarer Künstlicher Intelligenz

XAI-Techniken Grundlegend wird in vier verschiedene Arten von XAI-Techniken unterschieden, die jeweils post-hoc Anwendung finden: interpretierbare lokale Surrogate (engl. *Interpretable Local Surrogates*) [32], Okklusionsanalyse (engl. *Occlusion Analysis*) [40], gradientenbasierte Techniken (engl. *Integrated Gradients / SmoothGrad*) [41] [42] und die schichtweise Relevanzausbreitung (LRP), die auf der Sensitivity Analysis (SA) basiert. [5]

Vorbetrachtung Als Ausgangspunkt wird festgehalten, dass die Trainingsphase des ML-Prozesses abgeschlossen ist. Mithilfe der Funktion 2.4 findet eine Abstrahierung der Input-Output-Beziehung des ML-Modells statt. Reellwertige Merkmale $x = (x_1, \ldots, x_d)$ sind die Eingabe der Funktion, während als Ausgabe ein reellwertiger Score erzeugt wird, der die Entscheidungsgrundlage der XAI-Methode darstellt. Das heißt, sie beinhaltet Merkmale für oder gegen eine spezifische Entscheidung. [5]

$$f\colon \mathbb{R}^d \to \mathbb{R} \tag{2.4}$$

Interpretable Local Surrogates Die Entscheidungsfunktion wird bei der Methode der interpretierbaren lokalen Surrogate durch ein lokales Ersatzmodell ausgetauscht, das als selbsterklärend charakterisiert ist und somit interpretierbarere Erklärungen liefert. Der Hauptvertreter dieser XAI-Technik ist die Methode LIME [32]. [5]

Occlusion Analysis Als Art der Störungsanalyse werden bei der Okklusionsanalyse mehrfach der Einfluss verschiedener Einzelmerkmale des Eingabebildes sowie okkludierender Patches auf die Ausgabe des DNN geprüft. [5]

Integrated Gradients/SmoothGrad Die dritte Technik kombiniert die Methoden Integrated Gradients [41] und SmoothGrad [42]. Hierbei werden die aus den integrierten Gradienten resultierenden Relevanzwerte über mehrere Integrationspfade einer Zufallsverteilung gemittelt. Das Problem des Gradientenrauschens wird somit weiter gemindert. [5]

SA Die Sensitivitätsanalyse, engl. *Sensitivity Analysis*, berechnet die Sensitivität einer Vorhersage bezüglich der Eingabeänderungen. Konkret werden lokal Gradienten (partielle Ableitung) des Modells gebildet. Die Auswertung dieser Gradienten dient als Grundlage für die Erklärung der Vorhersage.

Merkmale, die bei der Eingabe für die Erklärung am meisten relevant sind, reagieren besonders empfindlich bei der Ausgabe. Es wird weiterhin die Variation des Funktionswertes $f(x)$ erklärt. Mathematisch wird anhand der Gleichung 2.5 die Bedeutung jeder Eingangsvariable i quantifiziert. Die Variable i kann beispielsweise ein einzelnes Bildpixel sein. [33]

$$R_i = ||\frac{\partial}{\partial x_i} f(x)|| \tag{2.5}$$

LRP Bei der schichtweisen Relevanzausbreitung, Vergleich Abbildung 2.7, erfolgt eine sinnvolle Zerlegung der Entscheidung anhand der Eingabevariablen. Der Klassifikator entscheidet demnach durch eine Dekomposition. Im Gegensatz zu SA wird bei LRP der Funktionswert $f(x)$ direkt erklärt. Lokale Umverteilungsregeln R_j definieren die Rückwärts-Umverteilung der Vorhersage $f(x)$. Dies geschieht so lange, bis jeder Eingabevariable ein Relevanzwert R_i zugeordnet wurde. Die Erhaltung der Relevanz ist demnach ein Schlüsselelement und wird in der Gleichung 2.6 verdeutlicht. Konkret bedeutet es, dass bei jedem Umverteilungsschritt weder Relevanz hinzugefügt noch entfernt wird. Somit bleibt die Gesamtmenge der Relevanz konstant. [43] [33]

$$\sum_i R_i = f(x) \tag{2.6}$$

Der Relevanzwert R_i jeder einzelnen Eingabevariable stellt einen entsprechenden Beitrag bezüglich der Vorhersage dar. Somit begründet sich ebenfalls die tatsächliche Zerlegung von $f(x)$ vergleichend zu SA. [43] [33]

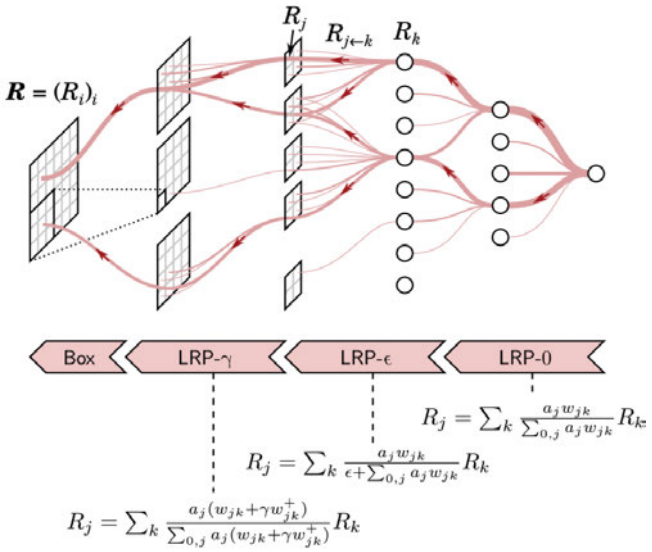


Abbildung 2.7: Sichtbar ist eine rückwärtsgerichtete Propagation der Vorhersagen ausgehend vom Ende des NN bis zu den Eingabemerkmalen. Die Art und Weise der *Back Propagation* wird mithilfe der Formeln R_j bestimmt. Der Ausbreitungsstrom ist rot hinterlegt, wobei die ebenfalls roten Pfeile die Laufrichtung anzeigen. [5]

Software Die auf Python-basierende Toolbox der LRP-Methode ermöglicht eine direkte Anwendung auf eigene KI-Modelle. Zusammenfassend bietet das Werkzeug leistungsstarke Funktionen für das Aufdecken von Fehlern innerhalb des Modells, Verbesserungspotenziale der Leistung des Modells und Erkenntnisgewinn bezogen auf die aktuelle Problemstellung. [33]

Spectral Relevance Analysis (SpRAy) Im Folgenden wird die XAI-Methode der spektralen Relevanzanalyse (SpRAy) vorgestellt. Um die Lücke zwischen Global XAI und Local XAI schließen zu können, automatisiert SpRAy die Analyse der großen Menge an lokalen LRP-Erklärungen. Beispielweise wurde Fehlverhalten eines hochmodernen ML-Modells zur Bildklassifizierung auf Grundlage des PASCAL VOC Benchmark-Datensatzes[44] mithilfe SpRAy entdeckt. Hierbei stützte sich die Klassifikationsentscheidung des Labels „Pferd“ auf ein Copyright-Wasserzeichen anstatt auf das tatsächlich abgebildete Pferd. Diese Art des unerwünschten Modellverhaltens wurde ebenfalls in weiteren bekannten Datensätzen, wie ImageNet[45] oder ISIC[46], detektiert. Die SpRAy Methode kann demnach unerwünschtes Modellverhalten erkennen, das anschließend ordnungsgemäß und präzise korrigiert werden kann. Die Anfälligkeit für böswillige Angriffe wird weiterhin minimiert. [47]
Konkret wird anhand eines Datensatzes, der die LRP-Erklärungen beinhaltet, ein spektrales Clustering durchgeführt. Hierbei soll typisches und atypisches Verhalten bei der Entscheidungsfindung von Modellen identifiziert und interpretierbar präsentiert werden. Merkmale des Entscheidungsverhaltens werden anhand von Heatmaps dargestellt, die eine vereinfachte Interpretation wiederkehrender Muster ermöglichen. Identifizierte Merkmale sind gemeinsam auftretende Besonderheiten, die einerseits vom Modell gelernt wurden, aber andererseits kein Teil des finalen Entscheidungsprozesses sind. SpRAy kann effizient auf große Datensätze angewendet werden, um einen umfassenden Überblick über das Klassifikationsverhalten des Modells zu erhalten. Außerdem können unerwartete oder sogenannte „Clever Hans“-artige Muster der Entscheidungsfindung ermittelt werden. [48]

Die SpRAy-Methodik wird in vier Hauptschritte unterteilt:

1. Berechnung der *Relevance Maps* mit LRP für die interessanten Datenproben,
2. Verkleinerung und Vereinheitlichung der Relevance Maps in Form und Größe,
3. Anwendung einer *Spectral Cluster Analysis* (SC) auf den Relevance Maps,
4. Identifizierung interessanter Cluster durch *Eigengap*-Analyse [48]

In Schritt 1 zeigen die Relevance Maps in Form von Heatmaps Informationen über relevante Bildbereiche, die der Klassifikator zur Entscheidungsfindung verwendet. Indem im Schritt 2 die Dimensionen reduziert werden, kann die Analyse der Relevance Maps beschleunigt werden. Im dritten Schritt werden Strukturen in der Verteilung der Relevance Maps gefunden, die zu endlich vielen Clustern gruppiert werden. Das durch Spectral Cluster Analysis (SC) entstandene Eigenwertspektrum codiert in Schritt 4 Informationen bezüglich der Struktur der Cluster aus den Relevance Maps. Eine gute Trennung der Cluster wird durch einen starken Anstieg der Differenz aus zwei aufeinanderfolgenden Eigenwerten deutlich. Die detektierten Cluster müssen abschließend eingehend inspiziert werden. Optional kann in einem fünften Schritt eine Visualisierung der Ergebnisse mithilfe der *t-distributed Stochastic Neighbor Embedding* (t-SNE) Methode stattfinden. [48]
Anders et al. stellen außerdem eine erweiterte Version der SpRAy-Methode vor: *Spectral Relevance Analysis Extended*. [49]

2.3.6 Vergleich und Verwendung von Softwarebibliotheken der Erklärbaren Künstlichen Intelligenz

Zunächst werden ausgewählte open-source Softwarebibliotheken vorgestellt, die Implementierungen von Verfahren für XAI bereitstellen. Oftmals werden in den Bibliotheken mehrere XAI-Methoden kombiniert. [36, S. 36 f.]

- **AI Explainability 360** fügt modell-agnostische und modell-spezifische Methoden für sowohl lokale als auch globale Erklärungen für KI-Modelle zusammen. Das Framework AI Explainability 360 (AIX360) wurde 2018 von IBM vorgestellt und weiterentwickelt. [50]
- **Alibi** wurde von der Firma Seldon Technologies Ltd. erschaffen und bietet vergleichbar mit AIX360 Verfahren der Modellagnostik und Modellspezifik an. [51]
- **Captum** wird von dem Unternehmen Facebook betrieben und dient dem Erklären von NN, wobei es auf die PyTorch Umgebung spezialisiert ist. [52]
- **iNNvestigate** wurde von Wissenschaftlern entwickelt und bündelt verschiedene Erklärungsverfahren zu DNNs. [53]
- **InterpretML** wurde von Microsoft entwickelt und beinhaltet eine Sammlung von White-Box-Modellen und modell-agnostischen Methoden zur Erklärbarkeit von Black-Box-Modellen, die auf tabellarischen Daten basieren. [54]
- **LIME** wird als Erklärungsverfahren ebenfalls von den Autoren implementiert und zur Verfügung gestellt. [32]
- **SHAP** ist ein von Wissenschaftlern implementiertes Verfahren zur Erklärung von NN. [39]
- **Skater** wurde von Oracle 2017 entwickelt und stellt vorwiegend modell-agnostische Methoden der KI-Erklärbarkeit bereit. [55]
- **TensorFlow (Tf)-explain** wurde als XAI-Framework für DNNs von der französischen Firma Sicara entwickelt. [56]
- **Quantus** ist eine XAI-Toolbox, die verschiedene XAI-Methoden anwendet, um schnell KI-Verfahren (Black-Box) mithilfe von lokalen und globalen Erklärungen auswerten zu können. [57]
- **Zennit explains neural networks in torch (Zennit)** dient als PyTorch-basierte Toolbox für die Erklärung von NN mithilfe der XAI-Methode LRP. Es wird laufend weiterentwickelt und wirbt mit einer hohen Anpassungs- und Integrationsfähigkeit individueller Projekte. [58]

Der in Abbildung 2.8 dargestellte Entscheidungsbaum des Bedarfs an Erklärbarkeit ist eine Hilfestellung bei der Verwendung ausgewählter XAI-Methoden. Der Einsatz der passenden Softwarebibliothek kann somit eingegrenzt werden. Hierbei wird jedoch nur auf den Teilbereich der Anwendungsfälle von KI eingegangen, der sich auf Bilddaten und tabellarische Daten stützt. [36]

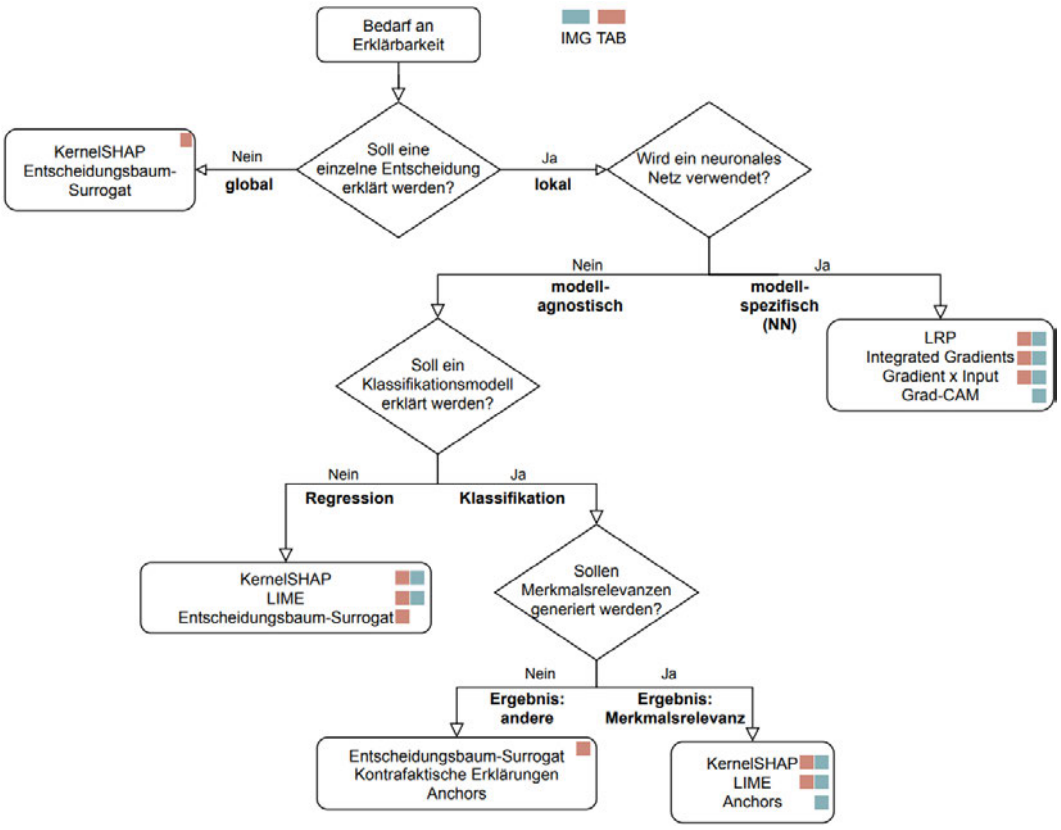


Abbildung 2.8: Dargestellt ist ein Entscheidungsbaum zur Verwendung der passenden XAI-Softwarebibliothek. Die der KI zugrundeliegenden Bilddaten werden durch ein hellblaues Quadrat und die tabellarischen Daten durch ein orangefarbenes Quadrat verdeutlicht. [36]

2.4 Aktueller Stand der Technik zur Auditierbarkeit von Künstlicher Intelligenz

Das folgende Kapitel bietet einen Überblick über die aktuellen Standardisierungsversuche in Bezug auf die Prüfbarkeit von KI-Systemen. Hierbei wird zunächst der Lebenszyklus eines KI-Systems vorgestellt und bekannte Abwehrmöglichkeiten von Angriffen auf KI-Systeme aufgezeigt. Weiterhin wird der Begriff der Verifikation erläutert und Audits im sicherheitskritischen Bereich betrachtet. Anschließend folgen die aktuellen Standardisierungsversuche mit vielversprechenden Ansätzen und der priorisierten Vorgehensweise bei der Auditierung von KI.

2.4.1 Gesamter Lebenszyklus eines Systems der Künstlichen Intelligenz

Um allgemeingültige Standards zur Auditierbarkeit von KI definieren zu können und eine nachhaltige IT-Sicherheit zu gewährleisten, muss zunächst ein tiefergehendes Verständnis für den gesamten Lebenszyklus eines cAI-Systems erlangt werden. Wie in Abbildung 2.9 erkennbar, besteht der Lebenszyklus aus einer Vielzahl von einzelnen Verarbeitungselementen, die untereinander vielfältige Verbindungen besitzen und deren Organisation in Schichten ausgeprägt ist. Der komplexe Lebenszyklus weist demnach mehrere Phasen auf. Zu Beginn wird das Konzept anhand anwendungsspezifischer Anforderungen und den gegebenen Umgebungsbedingungen geplant. Es folgt die Phase der

Datensammlung, wobei sowohl auf Qualität als auch auf Quantität geachtet wird. Innerhalb der Trainingsphase werden die Parameter trainiert. Die Leistungsfähigkeit dieser Phase wird anschließend anhand eines Testdatensatzes bewertet, um die korrekte Verallgemeinerungsfähigkeit des Modells sicherzustellen. Bei zufriedenstellenden Ergebnissen kann das System in Betrieb genommen werden. Hierbei ist es immer in organisatorische Prozesse eingebettet und interagiert mit der physischen Welt. [7, S. 6 ff.] [4, S. 112] [7, S. 4]

Der enorme Eingabe- und Parameterraum bei DNNs von über 100 Millionen Parametern erfordert eine interne Abstimmung, da die manuelle Einstellung nicht realisierbar ist. Demnach muss das Modell anhand der Trainingsdaten, einer Fehlerfunktion sowie einer Lernregel automatisch justiert werden. Hierbei wird in interne und externe Parameter unterschieden. Interne Parameter werden innerhalb der Trainingsphase erlangt, während externe Parameter, auch Hyperparameter genannt, einen Einfluss auf den Prozess des Lernens sowie der Modellarchitektur haben. Letztere werden mithilfe von Validierungsdatensätzen abgestimmt. [7, S. 6 ff.]

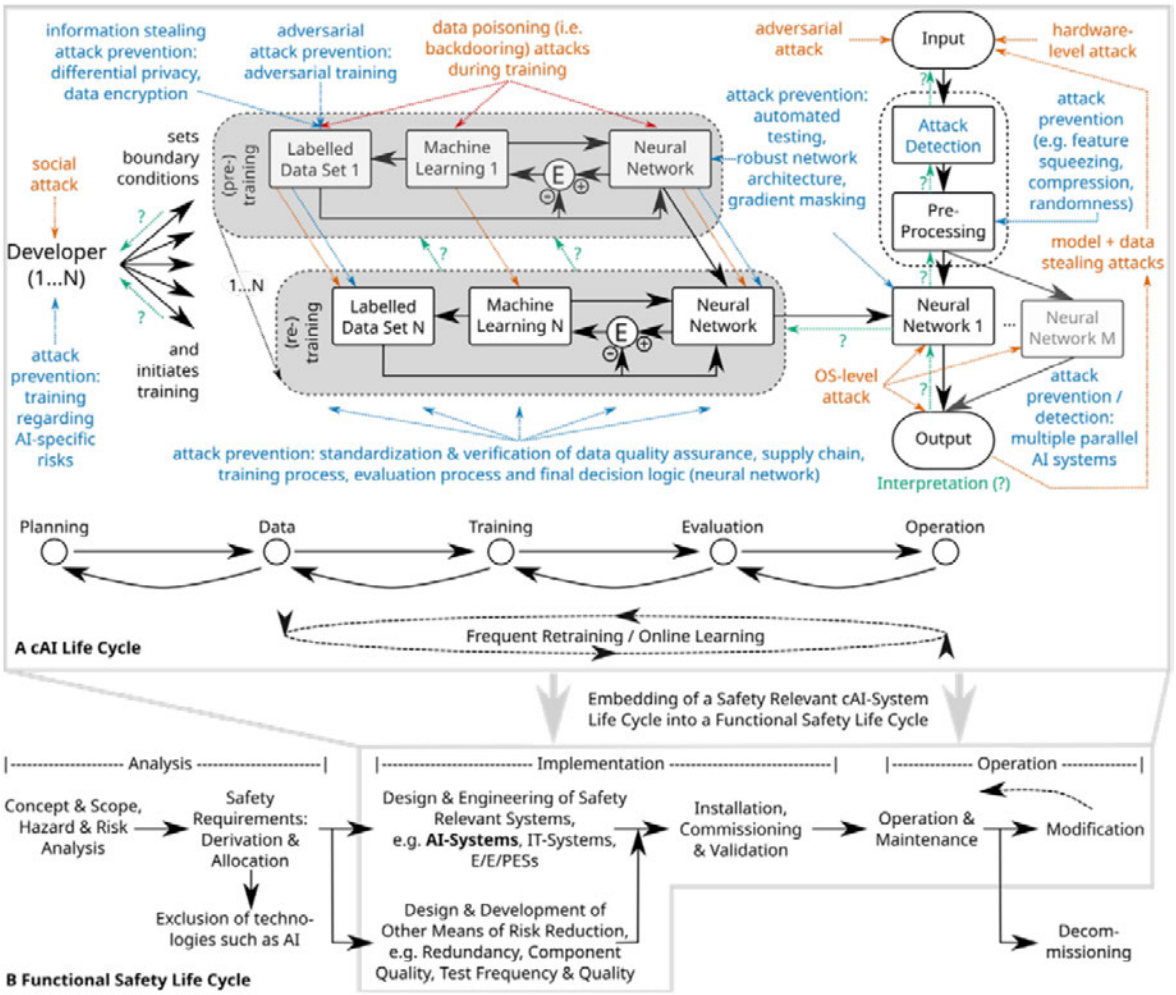


Abbildung 2.9: Die Verallgemeinerung des Lebenszyklus eines cAI-Systems ist in Teilbereich (A) dargestellt und wird aus der Perspektive der IT-Sicherheit betrachtet. Hierbei sind die Schwachstellen in Rot, die Abwehrmaßnahmen in Blau und die Interpretation in Grün visualisiert. Teilbereich (B) zeigt die Einbettung des cAI-Lebenszyklus in einen funktionalen Sicherheitslebenszyklus, der in die Phasen Analyse, Entwicklung und Betrieb untergliedert ist. Beide Lebenszyklen unterliegen einer hochgradigen Iteration. [7, S. 7]

Die schiere Größe der Parameteranzahl macht eine Interpretation der Funktionsweise nahezu unmöglich. Für ausgewählte Testeingaben wird aktuell die Eingabe-Ausgabe-Beziehung betrachtet, jedoch werden dafür große Mengen an Ressourcen und eine logische Systematik benötigt. Weiterhin treten bei cAI-Systemen qualitative Defizite auf, deren Angriffspotenzial und Abwehrmöglichkeiten in Kapitel 2.2.2 dargestellt werden. [7, S. 6 ff.]

2.4.2 Verifikation Künstlicher Intelligenz

Unbeabsichtigtes Verhalten von KI-Ausgaben wird durch Störungen der Eingabedaten verursacht, die durch natürliches Schwankungsverhalten oder einen beabsichtigten Angriff entstehen. Bei der Verifikation eines KI-Systems muss die vollständige Abwesenheit eines solchen Fehlverhaltens nachgewiesen sein. Die Verifikation urteilt demnach über die Sicherheit einer KI. Da die zu berücksichtigenden Störungen auf den großen Eingaberaum theoretisch in einer zahllosen Vielfalt auftreten können, werden Brute-Force-Ansätze zum Verifizieren ausgeschlossen. Weiterhin skalieren standardisierte Lösungen schlecht, da sich DNNs nicht linear verhalten und somit eine logische Überprüfung vorhandener Nebenbedingungen erschwert wird. [7, S. 14 f.]
Ein anerkannter Lösungsansatz stellt die abstrakte Interpretation dar. Hierbei wird ein möglicherweise unendlich großer Zustandsraum durch eine endliche und somit begrenzte Darstellung repräsentiert. Durch diesen Ansatz können die Zustände gespeichert werden und es können auf Symbolen basierende Berechnungen durchgeführt werden. Bezogen auf DNNs kann jede Art von Störungen der Eingabeparameter mittels Codierung symbolisch beschränkt werden. Diese Beschränkung dient als Grundlage für die Berechnung von abstrakten Einflüssen auf jede einzelne Schicht des Neuronalen Netzes. Daraus resultieren alle durchführbaren, codierten Ausgabemöglichkeiten, die mit der Eingabe übereinstimmen. Sie werden verwendet, um die zu verifizierenden Garantien zu überprüfen. In der Praxis äußern sich Symbolbeschränkungen durch eine Annäherung an die Vielfältigkeit des Datensatzes. Es muss daher ein Kompromiss zwischen der Genauigkeit der Annäherung und der Rechenkomplexität eingegangen werden. [7, S. 14 f.]
Nachfolgend sind Anmerkungen für eine verbesserte Verifikationsfähigkeit aufgezählt.

- Erweiterung des Anwendungsbereichs von geometrischen zu semantischen Störungen
- Einsatz verbesserter und benutzerdefinierter Lockerungen zur Verbesserung des Kompromisses zwischen Präzision und Komplexität
- Verallgemeinerung des Anwendungsbereichs bezüglich einer größeren Anzahl an Modelltypen und andersartige Aufgabenbereiche
- Verbesserung der Skalierbarkeit der Ansätze [7, S. 14 f.]

Ihr vollständiges Potenzial können die bestehenden Ansätze erst entfalten, wenn sie einen sogenannten zertifizierten Trainingsprozess durchlaufen. Weiterhin kann dieses zertifizierte Training mit dem Konzept der Verifizierung kombiniert werden, um zertifizierbare Verteidigungen zu erhalten. Neben der verbesserten Robustheit eines System kann das kontradiktorische Training helfen, die Verifizierungen zu erleichtern. In neuen Forschungsansätzen wird demnach das kontradiktorische und das zertifizierbare Training miteinander kombiniert. [7, S. 14 f.]

Die Verifikation der Quelltexte von KI-Systemen unterliegt aktuellen Beschränkungen. Konventionelle Testverfahren wie „die statistische Code-Analyse (Grammatech's CODESURFER), Runtime Verification (Java Pathfinder) oder Model Checking (SPIN model checker)“ [4, S. 93] ermöglichen

eine eingeschränkte Verifizierung. Die unterschiedlichen Methoden basieren auf dem Erfüllen der Formeln der Booleschen Aussagenlogik oder der Prädikatenlogik. Weiterhin können sie auf lineare Probleme reduziert werden sowie über die Robustheit eines MLP abgeleitet werden. Jedoch weisen die bestehenden Verifikationsverfahren aufgrund des enormen Parameterraums eine hohe Rechenleistung auf. [4, S. 93]

2.4.3 Prüfung sicherheitskritischer Systeme der Künstlichen Intelligenz

Insbesondere im Bereich der sicherheitskritischen Systeme ist die Sicherstellung normenbasierte Anforderungen notwendig, um die Robustheit und die Zuverlässigkeit eines KI-Modells nachzuweisen. Jedoch können bestehende und validierte Software-Standards, wie zum Beispiel die IEC 61508, nicht vollständig auf die technologischen Neuerungen angewendet werden. Um dem gegenwärtigen Fehlen dieser Richtlinien entgegenzuwirken, findet ein argumentbasierter Ansatz Anwendung, bei dem bestimmte Behauptungen durch formal strukturiertes Argumentieren gerechtfertigt werden. Die Flexibilität bei der Art und Weise der Rechtfertigung sicherheitsrelevanter Aussagen ist besonders vorteilhaft. Dementsprechend können bisher unerkannte Herausforderungen auf unbekannten Gebieten identifiziert werden. [7, S. 15 f.]
Speziell wird die Anwendung des CAE-Frameworks (Claims, Arguments, Evidence) vorgeschlagen. CAE basiert auf der Annahme, dass neue Erkenntnisse mithilfe von logischer Deduktion erlangt werden. *Claims* sind hierbei Behauptungen, die allgemeingültig akzeptiert sind. Argumente (engl. *Arguments*) verknüpfen die Behauptungen mit den Beweisen (engl. *Evidence*), wobei die Beweise eine Behauptung rechtfertigen. Mithilfe von Gegenansprüchen und der Bestätigungstheorie kann CAE erweitert werden, um die Wahrscheinlichkeit des Auftretens von Bestätigungsverzerrungen zu minimieren. Die formale Verifikation wird weiterhin durch eine fehlende Definition von Schlüsseleigenschaften, wie der Systemrobustheit, erschwert. CAE liefert hierbei Ansätze, wie eine solche Erklärung stattfinden kann. Zunächst muss der vergleichsweise einfache Nachweis der punktweisen Robustheit eines KI-Systems erfolgen. Im Idealfall mündet dieser Nachweis in einer vollständigen Definition der Systemrobustheit. [7, S. 15 f.]
Dieses bestmögliche Szenario ist jedoch zurzeit bei dem Großteil der Praxisanwendungen nicht realisierbar, da jede mögliche Zukunftseingabe einbezogen werden muss. Da derzeit keine vollständige formale Verifikation von KI stattfinden kann, werden statische Analysewerkzeuge verwendet, um das Ausbreiten von Fehlern zu verringern oder gänzlich zu verhindern und eine Grundlage für die Systemsicherheit bereitzustellen. [7, S. 15 f.]

2.4.4 Standardisierungsversuche

Weltweite KI-Standards Im Hinblick auf das Auditieren von KI-Systemen ist die Entwicklung einheitlicher technischer Qualitätsstandards erforderlich, um eine unabhängige Beschreibung von KI zu gewährleisten [7]. Da sowohl auf nationaler (DIN / DKE), europäischer (CEN / CENELEC / ETSI und internationaler Ebene (ISO / IEC / ITU) Normungsorganisationen agieren, ist eine Berücksichtigung aller Normungsebenen wichtig für einheitliche Auditierungsstandards. [4, S. 30]

Artificial Intelligence Act Die Europäische Kommission veröffentlichte am 21. April 2021 einen Entwurf für Vorschriften über KI mit dem Ziel der Änderung bestimmter EU-weiter Gesetzgebungen. Der Rechtsrahmen namens Artificial Intelligence Act (AIA) zur Regulierung von KI-Systemen verfolgt einen risikobasierten Ansatz, bei dem abgegrenzte Anforderungen an KI-Systeme erfüllt sein

müssen und KIs mit hohem Gefährdungspotenzial vor allem in sicherheitskritischen Funktionen verboten werden. Hierbei wird in die Risikoklassen minimal, gering, hoch und inakzeptabel unterschieden. Die Einordnung erfolgt anhand des Schadenspotenzials sowie der Abhängigkeit des Menschen von der Anwendung. [59] [8]

Deutsche KI-Standards Die deutsche Standardisierungs-Roadmap für Künstliche Intelligenz [4] stellt unter Einbezug bestehender Normungsvorschriften eine umfangreiche Ist-Zustand-Analyse bereit und zeigt den Bedarf an einheitlichen Qualitätsstandards auf. Hierbei wird besonders auf den in Abbildung 2.10 dargestellten Konformitätsnachweis und die Konformitätsbewertung eingegangen. Die Standards beziehen sich derzeit, Vergleich Abbildung 2.10(a), auf die Gütekriterien Sicherheit, Verlässlichkeit, Datenschutz, Transparenz und Fairness und müssen im Zuge einer einheitlichen Regulation für den Konformitätsnachweis von KI-Prozessen erweitert werden. [7] [4, S. 80 ff.]

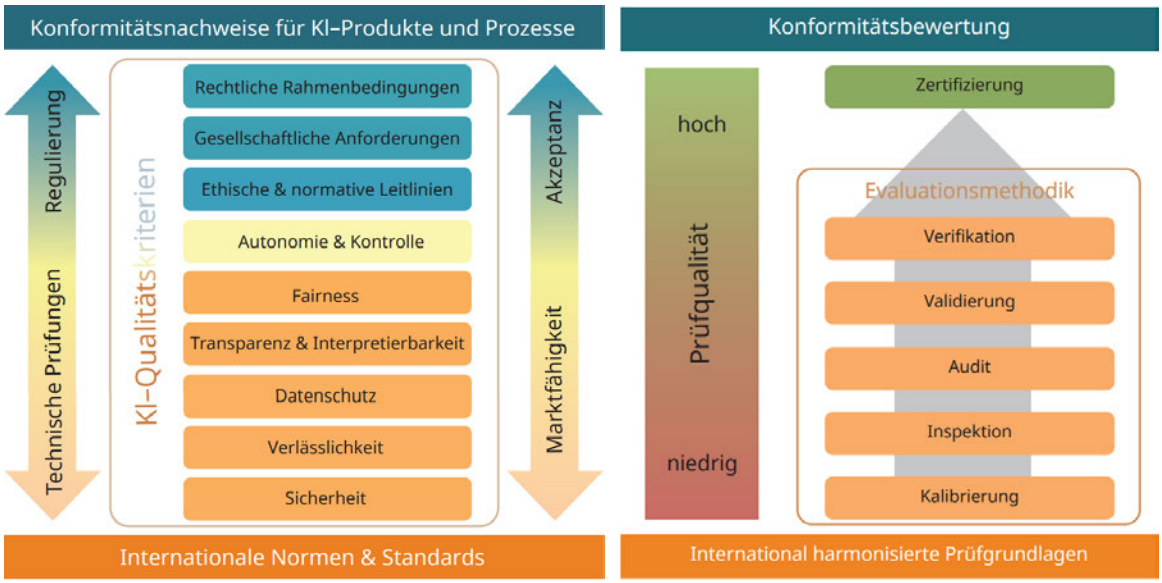
Konformitätsnachweis Die Prüfung von KI-Systemen in Form eines Konformitätsnachweises setzt sich anhand von Abbildung 2.10(a) aus den zwei Ebenen der technischen Prüfung und deren Bewertung zusammen. Zum einen wird das Vorhandensein spezifischer Eigenschaften des KI-Systems und zum anderen die Eignung für das geplante Anwendungsfeld untersucht. Die Prüfung geschieht anhand der KI-Qualitätskriterien. [4, S. 80 ff.]

Konformitätsbewertung Das Bewertungsmaß stellt fest, ob alle vorgeschriebenen Anforderungen an KI-Systeme erfüllt sind. Zu den in Abbildung 2.10(b) dargestellten Methoden der Konformitätsbewertung und Evaluation gehört das Audit. [4, S. 80 ff.]

Audit Allgemeingültig wird bei einem Audit überprüft, ob festgelegte Prozesse und Verfahrensweisen die Forderungen eines anerkannten Standards sowie deren Prüfungskriterien erfüllen. Hierbei ist die Einsicht in die Dokumentation des Systems gleichermaßen wie eine eingehende Untersuchungen vor Ort unverzichtbar. Audits können durch die Organisation selbst, einen Partner der Organisation oder durch eine unabhängige Partei erfolgen. Das Ziel einer Auditierung durch eine dritte Partei ist der Erhalt einer Zertifizierung. Der Standard **ISO 19011** definiert einen Leitfaden zur Planung, Durchführung und Nachbereitung für Audits von Managementsystemen. [4, S. 80 ff.]

Folgende Standards regulieren die Arbeit mit Künstlicher Intelligenz:

- **ISO/IEC 22989**, Artificial intelligence – Concepts and terminology,
- **ISO/IEC 23053**, Framework for Artificial Intelligence Systems Using Machine Learning,
- **ISO/IEC 23894**, Information Technology – Artificial Intelligence – Risk Management,
- **ISO/IEC 38507**, Information technology – Governance of IT – Governance implications of the use of artificial intelligence by organizations,
- **ISO/IEC 20546**, Information technology – Big data – Overview and vocabulary,
- **ISO/IEC 5059**, Software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Quality Model for AI-based systems,
- **ISO/IEC TR 24027**, Information technology – Artificial Intelligence – Bias in AI systems and AI aided decision making,
- **ISO/IEC TR 24368**, Information technology – Artificial intelligence – Overview of ethical and societal concerns. [4, S. 38]



(a) Konformitätsnachweis (b) Konformitätsbewertung

Abbildung 2.10: Nachweis und Bewertung der Konformität von KI-Systemen [4, S. 80 ff.]

Wie in Abbildung 2.11 einsehbar, ist ab Stufe 2 der Kritikalitätspyramide „Anwendungen mit einem gewissen Schädigungspotenzial“ eine Auditierung des KI-Systems sowie die Pflicht zum Offenlegen deren Funktionsweise gegenüber Aufsichtsinstitutionen vorgeschrieben. Hierbei muss jegliche Art von Fehlverhalten erkannt und eingehend analysiert werden. Von Stufe 2 bis 5 muss eine Risikofolgenabschätzung durchgeführt werden, die Transparenz des Algorithmus voraussetzt. [4, S. 52 f.]

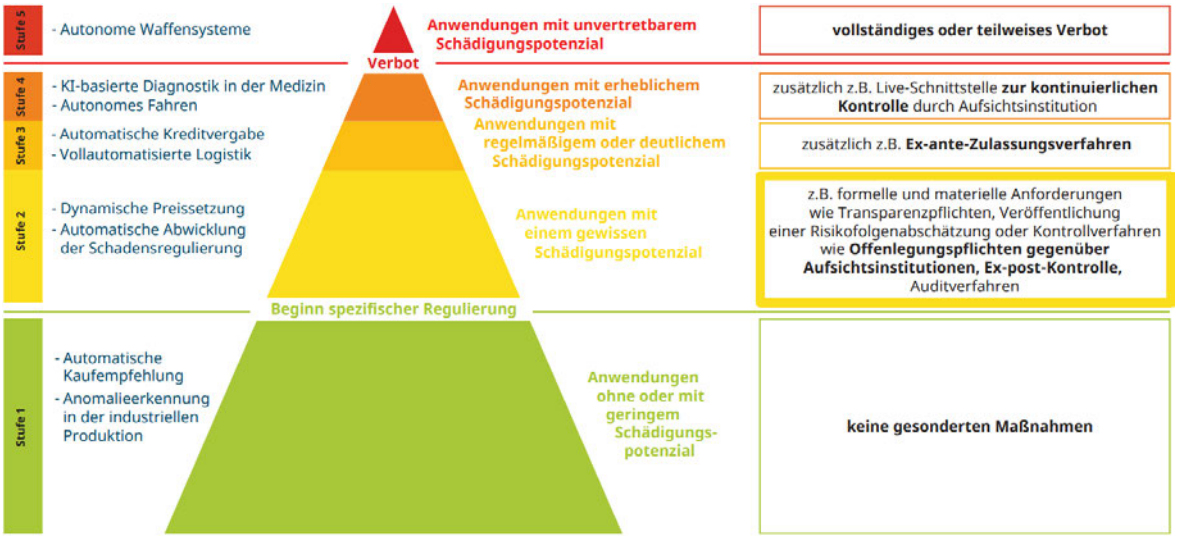


Abbildung 2.11: Dargestellt ist eine Kritikalitätspyramide, die die Risiko-Adaption für den Einsatz von algorithmusbasierten Systemen reguliert. [4, S. 53]

Zusammenfassend bedürfen technische Testverfahren einer erheblichen Weiterentwicklung bezogen auf die Durchführung dynamischer Sicherheitstests sowie deren sicherheitskritische Belastbarkeit. Insbesondere muss die weiterführende Erforschung von standardisierten Validierungs- und Verifikationsmethoden forciert werden. [7]

2.4.5 Certification Readiness Matrix

Die Certification Readiness Matrix (CRM) ist eine zweidimensionale Matrix zur Überwachung von Fortschritten im Bereich der Auditierbarkeit von KI in unterschiedlichen Anwendungsszenarien. Anhand Abbildung 2.12 sind zwei Dimensionen erkennbar. Die erste Dimension zeigt die Phasen des cAI-Lebenszyklus sowie deren Integration in die Organisation. In Abhängigkeit dazu werden relevante IT-sicherheitstechnische Aspekte, die objektiv bewertet werden, in der zweiten Dimension verdeutlicht. Eine solche Bewertung erfolgt anhand der Vergabe von Noten für jeden Matrixeintrag. Dieser Notenwert von 0 bis 10 entspricht der Zertifizierungsreife, welche von keine (rot) bis vollständige Reife (grün) geht. Die zeitliche Überprüfbarkeit zwischen Audit-Anwendungen soll hiermit verglichen werden können. Die CRM ist einfach erweiterbar. Aus der Abbildung 2.12 geht hervor, dass die Aspekte Rückverfolgbarkeit, Risikomanagement und teilweise die Modellleistung im Vergleich zu den anderen Aspekten den höchsten Grad an Auditierbarkeit aufweisen. Im Gegensatz dazu besteht bei den Kriterien Sicherheit, Robustheit und Interpretierbarkeit ein deutlicher Handlungsbedarf, der durch die technologischen Herausforderungen begründet ist. Zusammenfassend sind Auditiermethoden für bestimmte Bewertungsaspekte gut ausgeprägt, während bei anderen Aspekten weitergehende Forschung und Entwicklung notwendig ist. [8, S. 5 ff.]

	Lifecycle Phase / Aspect	Security	Safety	Performance	Robustness	Interpret-/ Explainability	Tracability	Risk Management
Embedding	organization	3	2	5	3	4	6	6
	use case specific requirements & risks	5	5	5	5	4	4	6
	Embodiment & situatedness of AI module	5	5	5	5	6	2	5
AI module life cycle	planning phase	4	4	5	4	4	6	6
	data acquisition and QA phase	4	5	6	6	4	6	6
	training phase	5	5	5	5	6	6	6
	evaluation phase	5	5	5	5	6	6	6
	deployment and scaling phase	4	2	5	3	4	6	6
	operational (& maintenance) phase	5	2	5	3	4	6	6

Auditability Scoring: 0 none | 2.5 | 5 average | 7.5 | 10 full

Abbildung 2.12: Die CRM verdeutlicht die Phasen des cAI-Lebenszyklus in Abhängigkeit zu den Bewertungsaspekten der IT-Sicherheit. modifiziert nach [8]

2.4.6 Vielversprechende Ansätze für konnektionistische Künstliche Intelligenz

Neben den vielen Herausforderungen und offenen Fragen der vorangegangenen Kapitel existieren darüber hinaus vielversprechende Ansätze zur Entwicklung von Standards für sicherheitsrelevante KI-Anwendungen sowie zur Lösung und Verringerung bestehender Problemstellungen. [7]

Neue Methoden zur Prüfung von strukturierten KI-Systemen KI-Systeme arbeiten grundlegend auf einer Basis von strukturierten Daten (Tabellen) oder unstrukturierten Daten, beispielsweise Bild-, Video- und Audiodateien. Im Bereich der strukturierten Daten lassen sich vielversprechende Methoden, wie beispielsweise LIME [32], Shapley [60], DeepLIFT [37] und QII [61] finden. Im Hinblick auf kontinuierlich lernende Systeme wird im Folgenden vorrangig auf vielversprechende Ansätze für Prüfmethoden zu unstrukturierten Daten eingegangen. Sie befinden sich derzeit noch in der Erforschungs- und Entwicklungsphase. [4, S. 93]

Herausforderungen Ein cAI-System ist zumeist in einen komplexen Systemlebenszyklus mehrerer cIT- und sAI-Systeme eingebettet, wodurch offene Fragestellungen in den Bereichen der Auditierbarkeit und der IT-Security resultieren. Für jeden Anwendungsfall ist ein tiefergehendes Verständnis für Eigenschaften, wie beispielsweise die interne Struktur sowie die externe Umgebung notwendig, um eine Risikobewertung im Hinblick auf die funktionale Sicherheit durchzuführen. Neben der Umgebung muss ein Verwendungszweck in Form einer Aufgabenstellung sowie deren Merkmalsausprägung identifiziert und verstanden werden. Anschließend folgt eine Evaluation des KI-Modells. Idealerweise sollte der Wissensstand von KI-Entwickler*innen sowie Anwender*innen auf einer einheitlichen Standardisierung basieren, damit ein verantwortungsvoller Umgang bei Implementation und Verwendung gewährleistet werden kann. Dieser Gedanke schließt das Treffen von fundierten Entscheidungen mit ein. [7]
Oftmals muss ein Kompromiss zwischen gewünschten Systemeigenschaften (Robustheit, Sicherheit, Überprüfbarkeit) und den tatsächlichen Eigenschaften eines Systems (ML-Algorithmus, Daten, Randbedingungen) getroffen werden. Die folgenden Beispiel-Trade-Offs stellen eine Einschränkung hinsichtlich der Skalierbarkeit und Verallgemeinerungsfähigkeit von KI-Systemen dar. [7]

- Modellkomplexität vs. Interpretierbarkeit, Verteidigung
- Größe des Aufgabenbereichs und des Datensatzes vs. Verifizierung
- Verstärkte Abwehrmaßnahmen vs. Leistung des KI-Systems
- Verbesserte Überprüfbarkeit durch White-Box Zugriff vs. Datenschutz, Privatsphäre
- Verwendung externer Datensätze und vortrainierter KI-Modelle zur Kostenreduzierung vs. erhöhtes Risikopotenzial für Schwachstellen und Backdoor Attacks

Zusammenfassung erfolgversprechender Methoden Die Tabelle 2.2 adressiert offene Problemstellungen sowie deren Verbesserungspotenziale und fasst Lösungsansätze zusammen, die in der aktuellen Forschung als sehr erfolgsversprechend gelten. Die vielversprechenden Ansätze können für einen verbesserten Optimierungsversuch kombiniert werden. Die Komplexität zukünftiger Problemstellungen wird zunehmen, wodurch noch bessere Ansätze benötigt werden. [7]

Tabelle 2.2: Erfolgversprechende Ansätze für standardisiertes Auditieren von KI-Systemen [7]

	Erfolgversprechende Ansätze	**Offene Problemstellungen**
1.	Verwendung von Transfer-Lernen und *few-shot learning* Methoden Verwendung nicht-parametrischer Methoden und ensemble Methoden	Effizienteres Re-Training einer KI Verbessertes Tuning von Metaparametern, Ausgewogenheit zwischen Plastizität und Stabilität bei gering gewichtigen KI-Modellen
2.	Optimierung einer geeigneten Performance-Metrik zur Evaluation von Verteidigungsstrategien unter Berücksichtigung natürlicher und gegnerischer Eingaben	Vermeidung des üblichen Leistungsabfalls bei der Verwendung starker Verteidigungsmaßnahmen
3.	Shared und Meta Adversarial Training	Kostenreduzierung im Umgang mit universellen Störungen
4.	Systematische Verwendung synthetischer und erweiterbarer Datensätze und Simulationen	Identifizierung von Fehlerquellen, Verbesserung der Systemrobustheit trotz großer Aufgabenräume
5.	zum Teil abstrakte Interpretation und zertifiziertes Training	Verifizierung von KI-Systemen mit größeren Aufgabenräumen
6.	Anwendung von argumentbasierten Ansätzen (zum Beispiel CAE) und einer widerlegbaren Argumentationsstrategie	Prüfung von KI-Systemen bei Nicht-Anwendbarkeit von bestehenden Methoden
7.	Einsatz eines menschlichen Priors Verwendung hybrider Modelle	Verbesserung der Interpretierbarkeit Verbesserung der Systemrobustheit
8.	Datenbereinigung und Ausreißer-Erkennung in Datensätzen durch Interpretationsmethoden (Zurückweisen negativer Einflüsse und verwandter Ansätze während der Trainingsphase)	Abwehr von Backdoor-Angriffen
9.	Einsatz von Ersatzmodellen und Ersatzdatensätzen	potenzielle Verbesserung der Prüfungsqualität (kein White-Box Zugriff vorhanden)
10.	Verwendung kryptographischer Verfahren und Vertrauensketten	Sicherstellung der Daten- und Modellintegrität innerhalb der Lieferkette

2.4.7 Priorisierte Vorgehensweise bei der Auditierung

Aktuell existiert kein allgemeingültiger Kriterienkatalog, der einen Nachweis über ein vollständig abgesichertes KI-System mit hinreichend geringer Fehler- und Ausfallwahrscheinlichkeit erbringen kann. Im Folgenden werden zwei Strategien unterschieden, die das Vorgehen zum Erhalten einer sicheren und gleichzeitig auditierbaren KI beschreiben. [7, S. 22 f.]
Im ersten Anwendungsfall wird der Fokus auf das Erzeugen günstiger Randbedingungen in Abhängigkeit zu der vorliegenden Aufgabe gelegt. Hierbei sollen sowohl Entwickler*innen als auch Anwender*innen entsprechend einer klaren Aufgabendefinition sowie angemessener Randbedingungen geschult werden. Es folgt eine Analyse möglicher Risikofaktoren bei der Integration in ein übergeordnetes IT-System. Auf Grundlage dieser Überlegungen können fundierte Entscheidungen seitens der Anwender*innen und seitens der KI getroffen werden. In Abhängigkeit von der jeweiligen Anwendung kann flexibel eine Einschränkung des Aufgabenraums und der zuvor definierten Randbedingungen erfolgen. Die dadurch reduzierte Komplexität ermöglicht eine vereinfachte Überprüfung der Arbeitsweise und Sicherheit des KI-Systems. Während des gesamten Lebenszyklus können Maßnahmen technischer und organisatorischer Art verknüpft werden, um einen Beitrag für die Sicherheit und Überprüfbarkeit des Systems zu leisten. [7, S. 22 f.]
Der zweite Ansatz konzentriert sich auf die zukünftige Forschungstätigkeit und wie diese Entwicklung innovativ und interdisziplinär anhand bestehender Techniken und Forschungsinvestitionen intensiviert werden kann. Idealerweise soll die Sicherheit von KI-Systemen auch bei komplexen Randbedingungen unter Berücksichtigung der Verbesserung der Skalierbarkeit und Generalisierbarkeit gewährleistet werden. Hierfür existieren vielversprechende Ansätze: [7, S. 22 f.]

- Erforschung geeigneter Sicherheits-Metriken zur Reduzierung des Kompromisses zwischen Performance und Abwehrmaßnahmen,
- Verknüpfung robuster KI-Modelle mit Malware-Erkennungsalgorithmen,
- zusätzlicher Einsatz menschlicher Überwachung,
- Erstellung effizienter sowie qualitativ und quantitativ hochwertiger Angriffspotenziale zur Verbesserung bestehender Verteidigungsstrategien,
- Generieren von realitätsnahen Trainingsdatensätzen für robuste KI-Modelle,
- Durchführen von realitätsnahen Simulationen bei gleichzeitiger Bewertung dieser Ergebnisse,
- Anwendung und Kombination verschiedener cAI-, cIT- und sAI-Systeme [7, S. 22 f.]

Zukünftige Arbeiten sollen demnach die qualitativ hochwertige Durchführung beider Strategien in Bezug auf sicherheitskritische Anwendungsfälle priorisieren. [7, S. 22 f.]

2.5 Continual Learning

Aktueller Forschungsbedarf Ein schnell wachsendes Interesse an Continual Learning (CL) wird in einschlägiger Literatur deutlich. In diesem Sinne ist es notwendig, weiterführende Forschung zu betreiben und in zukunftsweisende Innovationen zu investieren. Im Folgenden werden daher Herausforderungen, Methoden und Szenarien des kontinuierlichen Lernens aufgegriffen. Im zweiten Teil des Kapitels steht das Framework Avalanche im Mittelpunkt, das eine programmtechnische Umsetzung kontinuierlich lernender KI-Systeme ermöglicht. [62]

Definition Kontinuierliches Lernen, engl. *Continual Learning* (CL), unterscheidet sich von herkömmlichen maschinellen Lernalgorithmen durch einen kontinuierlichen Datenstrom als Dateneingabe. Dabei geschieht eine Abkehr von den aufgabenspezifischen Architekturen hin zu einer generalisierten Architektur, die nicht ausschließlich auf eine Aufgabe (engl. *Task*) beschränkt ist. Die über den Lernprozess erworbenen Informationen dienen als Grundlage für das Lösen neuer Aufgabenstellungen. Bei kontinuierlichen Lernalgorithmen treten zwei Herausforderungen auf: die Verteilungsverschiebung und das katastrophale Vergessen. [63]

Anwendungsbereiche Zu den Anwendungsfeldern von kontinuierlichen Lernmodellen gehören unter anderem die Bildklassifizierung, das *Reinforcement Learning*, die *Continual Semantic Segmentation* und das *Natural Language Processing* (NLP). Der Großteil der CL-Modelle wird durch Bildklassifikationen vertreten, da die Mehrheit der Daten bereits annotiert ist und auf vollständiger Überwachung basiert. Diese Aspekte vereinfachen die Trainingsphase sowie das Definieren eines Tasks. Außerdem kann ohne Störungen anderer ML-Probleme eine kontinuierliche Bewertung vereinfacht untersucht werden. [64]

Herausforderungen Das Problem der Verteilungsverschiebung, auch Konzeptdrift genannt, (engl. *Distributional Shift / Concept Drift*) entsteht, wenn sich die Verteilung der Daten verändert, beispielsweise bei der Veränderung der Bedeutung von Worten im Laufe der Zeit. Dies wird ermöglicht, da CL-Algorithmen auf einem sich ständig ändernden Eingabedatenstrom basieren. Durch die sich verändernden Eingabedaten ist das CL-Modell dazu gezwungen, die Daten adäquat zu interpretieren und anpassungsfähig auf Änderungen zu reagieren. Die Herausforderung des Konzeptdrifts wird grundsätzlich in virtuellen und realen Konzeptdrift unterteilt. Bei dem virtuellen Konzeptdrift existiert eine unausgewogene Verteilung von Daten zwischen den Klassen. Der reale Konzeptdrift äußert sich hingegen durch das Hinzufügen einer neuen Klasse (oder eines neuen Datenpunkts) zu dem bestehenden Datensatz. [63]
Das katastrophale Vergessen (engl. *Catastrophic Forgetting*) wird durch das Verlernen bereits bekannter Konzepte beschrieben. Dies geschieht, da das CL-Modell auf neue Konzepte antrainiert wird. Alte Konzepte werden von der KI vergessen und sind somit nicht mehr auf das aktuelle Problem anwendbar. Dieses Phänomen ist in Abbildung 2.13 erkennbar. Hier werden die Kurvenverläufe von rechts nach links kürzer, je mehr Aufgaben zum Training hinzugefügt werden. [63] [65]

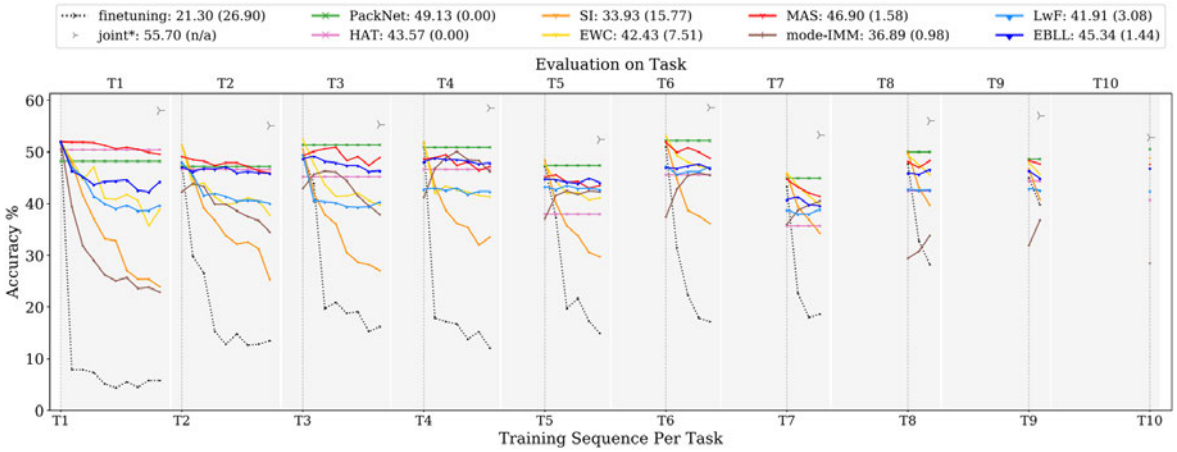


Abbildung 2.13: Das Diagramm zeigt die Testgenauigkeit verschiedener CL-Modelle für eine bestimmte Aufgabe in Abhängigkeit der Trainingssequenz pro Aufgabe. Es wird in die Trainingssequenzen T1 bis T10 unterschieden, die in der Trainingsphase iterativ nacheinander ausgeführt werden. Die Legende zeigt die durchschnittliche Genauigkeit und das Vergessen pro angewendetem CL-Modell. Die Entwicklung der Testgenauigkeit wird pro Methode farblich voneinander abgegrenzt. [65]

Bei dem Datendrift stellt sich die Frage, welche Veränderungen für korrekte Vorhersagen zu einem definierten Zeitpunkt relevant sind und welche Informationen verworfen werden sollten. Diese Überlegung wird in dem Plastizität-Stabilität-Dilemma aufgegriffen. Plastizität stellt hierbei die Fähigkeit zur Integration neuer Informationen dar, während bei der Stabilität vergangene, aber noch aktuelle Informationen beibehalten werden. Ein Gleichgewicht beider Eigenschaften ist für eine gute Systemleistung bei einfachen KI-Modellen trotz Datendrift notwendig. Hierbei ist zu beachten, dass Meta-Patameter zu Modell-Parametern werden und sich somit möglicherweise die Modellkomplexität ändern kann. Um diese Herausforderung zu meistern, eignet sich die Verwendung von nicht-parametrischen Modellen oder Ensemble-Modellen. [7, S. 9 f.]

2.5.1 Continual Learning Methoden

Vergleichbar zu Abbildung 2.14 existieren drei Hauptansätze für kontinuierliches Lernen: die Replay-Methoden (engl. *Replay Methods*), die regularisierungsbasierten Methoden (engl. *Regularization-based Methods*) sowie die architekturbasierten Methoden. Die Replay-Methoden zeichnen sich durch Wiederholungen aus. Das erworbene Wissen wird regelmäßig neu geprüft und angelernt, wodurch vergangene Erfahrungen gespeichert werden können. Regularisierungsbasierte Methoden verwenden Callback-Funktionen, wie *Dropout* und *Early Stopping*, die Overfitting und katastrophales Vergessen vermeiden sollen. Die Callbacks helfen hierbei im Umgang mit wechselnden Gewichten. Bei den architekturbasierten Methoden wird die Architektur direkt modifiziert, um das katastrophale Vergessen zu verhindern. Ein Beispiel hierfür ist das Erstellen und Verknüpfen einer Vielzahl von Subnetzwerken zu einem globalen Netzwerk. Dabei ist jedes Subnetzwerk für eine eigene Aufgabe zuständig. Alternativ kann bei dem Vorkommen neuer Aufgaben ein neues Modell erstellt werden, dass Verbindungen zu früheren Modellen aufbaut. Hierbei unterliegt das Modell einer ständigen Beobachtung, um wichtige Gewichte zu detektieren, zu maskieren und anschließend einzufrieren, um eine Änderung des Gewichtes zu verhindern. Weiterhin wurde die Verwendung eines dualen Ansatzes betrachtet, bei dem ein Modell fortwährend neue Konzepte lernt, während ein anderes Modell zuvor erworbenes Wissen speichert. [63]

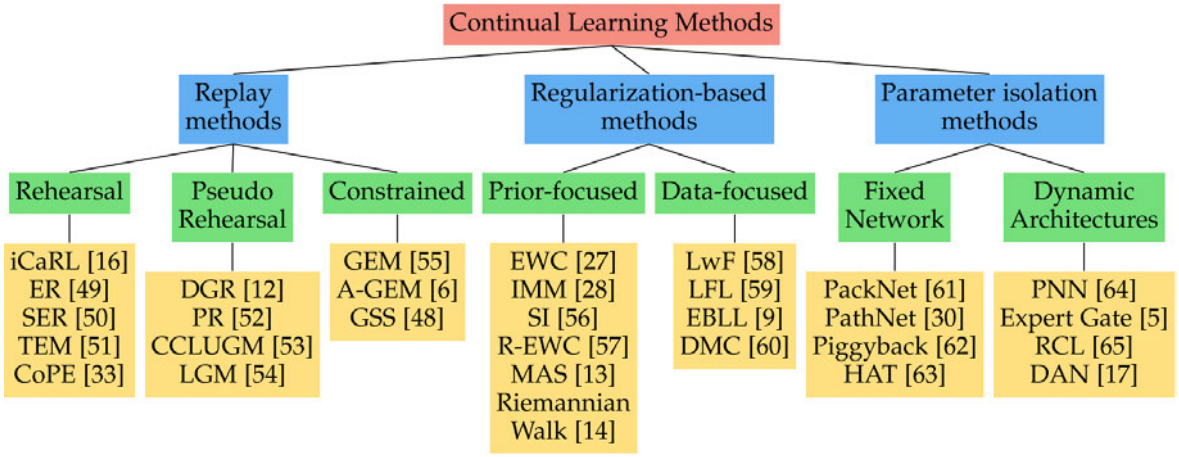


Abbildung 2.14: Im Bereich des CL wird anhand des Diagramms in Replay Methods, Regularization-based Methods und *Parameter Isolation Methods* unterschieden. Für jede Methode sind Unterkategorien sowie Beispielmethodiken dargestellt. [65]

2.5.2 Continual Learning Szenarien

Die Szenarien für kontinuierliches Lernen sind vielgestaltig. Wird der Aspekt der Datenänderungen betrachtet, kann in inkrementelles und lebenslanges Lernen unterschieden werden.

Incremental Learning (IL) Bei dem inkrementellen Lernansatz werden auf Grundlage von neuen Aufgaben sequenziell neue Konzepte erstellt. In einem einzigen Modell werden diese gelabelten Objekte, auch Konzepte genannt, beibehalten. Bei Klassifikationsaufgaben werden beispielsweise anhand neuer Aufgaben neue Konzepte entwickelt, wobei im Bereich des *Reinforcement Learnings* auf der Grundlage von Multitask-Einstellungen Aufgaben einzeln gelernt werden.

Lifelong Learning Im Unterschied zum inkrementellen Lernen entstehen beim lebenslangen Lernen aus neuen Aufgaben neue Instanzen. Die Konzepte bleiben jedoch gleich. Das Phänomen des lebenslangen Lernens zeichnet sich über die durchgehende Verbesserung einer bestimmten Aufgabe mit neuen Daten aus. Bei Klassifikationsaufgaben liegen Aufgabensequenzen mit denselben Klassen, aber unterschiedlichen Datenpunkten vor. Indem die Kapazität eines Modells bewertet wird, können bekannte Konzepte mit neuen Daten verbessert werden. Dies geschieht im Hinblick auf die Entwicklung einer fortgeschrittenen Generalisierung. [64]
Die CL-Szenarien unterliegen einer großen Abhängigkeit bezüglich der Überwachung. Die Informationen der Daten werden durch Überwachungsspezifikationen charakterisiert. Erfolgt eine Änderung innerhalb der Datenverteilung, wird das System über dieses Ereignis informiert. [64]

Alternative Einteilung der Szenarien Laut van de Ven und Tolias können kontinuierliche Lernsysteme in drei Szenarien untergliedert werden, um die Leistungsfähigkeit kürzlich vorgeschlagener CL-Methoden pro Szenario zu bewerten. Zum Zweck der vereinfachten Vergleichbarkeit der verschiedenen Ansätze wird jeweils ein neuronales Netz betrachtet, das nacheinander mehrere Aufgaben lösen muss. Während der laufenden Trainingsphase hat das Modell ausschließlich Zugriff auf die Daten der aktuellen Aufgabe. Die Tasks sind eindeutig voneinander abgegrenzt. Weiterhin sind Informationen über die Verfügbarkeit der Task-IDs zur Testzeit notwendig. Die Tabelle 2.3 zeigt einen Überblick über die drei Szenarien des kontinuierlichen Lernens mit zunehmendem Schwierigkeitsgrad. [66]

Tabelle 2.3: Die Tabelle vergleicht die drei Incremental Learning Szenarien Task-IL, Domain-IL und Class-IL anhand der notwendigen Einflussgrößen während der Testzeit. Es wird betrachtet, ob zur Testzeit die Tasks gelöst wurden und ob die Task-ID angegeben ist. angelehnt an [66]

Szenario	Anforderungen zur Testzeit
Task-IL	Task gelöst, Task-ID bereitgestellt
Domain-IL	Task gelöst, Task-ID nicht bereitgestellt
Class-IL	Task gelöst und Task-ID abgeleitet

Das erste und einfachste kontinuierliche Lernszenario heißt aufgaben-inkrementelles Lernen oder Task-IL. Es basiert auf der Kenntnis, dass das Modell über die zu lösenden Tasks informiert ist und die Task-ID zur Testzeit bekannt ist. Im Unterschied zur Task-IL ist im zweiten Lernszenario, des domänen-inkrementellen Lernens oder Domain-IL, keine Task-ID zur Testzeit verfügbar. Anstehende Tasks werden durch die Modelle gelöst und müssen vorher nicht abgeleitet werden. Modelle des klassen-inkrementellen Lernens (Class-IL) müssen beim dritten Szenario in der Vergangenheit gesehene Tasks lösen können und die Task-ID, das heißt die aktuell gestellte Aufgabe, anhand dieser Informationen ableiten. Zusammengefasst beinhaltet das dritte Szenario demnach die bekannte Problemstellung des inkrementellen Lernens neuer Klassen von Objekten in der Realität. [66]

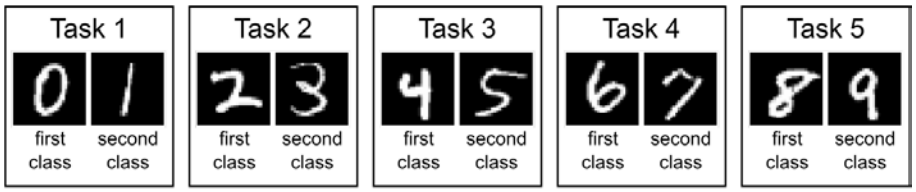


Abbildung 2.15: Das Bild verdeutlicht das schematische Task-Protokoll des sogenannten Split Modified National Institute of Standards and Technology (MNIST) Datensatzes. [66]

Tabelle 2.4: Die Fragestellung erklärt das Verhalten der jeweiligen CL-Szenarien anhand des Task-Protokolls in Abbildung 2.15. in Anlehnung an [66]

Szenario	Verständnisfrage
Task-IL	Bei bekanntem Task, liegt die 1. oder 2. Klasse vor? (z. B. 0 oder 1)
Domain-IL	Bei unbekanntem Task, liegt die 1. oder 2. Klasse vor? (z. B. in [0,2,4,6,8] oder [1,3,5,7,9])
Class-IL	Bei unbekanntem Task, welche Ziffer ist vorhanden? (d. h. Wahl von 0 bis 9)

Anhand der Abbildung 2.15 sowie der Tabelle 2.4 wird der Unterschied der drei CL-Szenarien verdeutlicht. Die Grundlage der Beispiel-Demonstration bildet der SplitMNIST Datensatz, welcher verwendet wird, um sequenziell MNIST-Ziffern zu klassifizieren. Hierbei sind die Ziffern 0 bis 9 in 5 Tasks unterteilt, die jeweils eine binäre Klassifikation beschreiben. Bei Task-IL stellt sich die Frage, ob bei bekannter Aufgabe (1-5) eine Klassifikation in die Klassen 1 oder 2 stattfindet. Das Szenario Domain-IL besitzt keine Kenntnis des Task und fragt sich daher, ob bei einem beliebigen Task in die erste oder

zweite Klasse eingeordnet wird. Im dritten Szenario, dem Class-IL, ist der Task ebenfalls unbekannt und es muss die gesuchte Ziffer abgeleitet werden. Die Information, dass ein Task in zwei Klassen unterscheidet, liegt demnach nicht vor. Es stellt sich die Frage: Welche Ziffer von 0 bis 9 liegt vor? [66]

2.5.3 Continual Learning Framework Avalanche

Allgemeines Aktuell ist ein schnell wachsendes Interesse für kontinuierliches Lernen in der KI-Community zu verzeichnen. Eine große Herausforderung stellt jedoch die Verfügbarkeit von einheitlichen Tools und Bibliotheken dar, die die Implementation, Bewertung und Replikation von CL-Systemen vereinfachen. Avalanche setzt an diesem Punkt an und stellt eine auf Python basierende, open-source, end-to-end Bibliothek für kontinuierliches Lernen bereit. Die aktuelle Version 0.2.1 wurde am 29.07.2022 auf GitHub[1] veröffentlicht. Es existiert eine offizielle Dokumentation[2], die kontinuierlich erweitert wird. Der Fokus der CL-Bibliothek liegt auf fünf kohärenten Hauptbestandteilen: den Benchmarks, dem Training, den Modellen, der Evaluation und dem Logging. Wie in Abbildung 2.16 zu sehen, wird durch die Benchmarks ein Strom an Lernerfahrungen erzeugt, die sequenziell dem internen Modell zugeführt und während des Trainings verarbeitet werden. Anschließend erfolgt eine Evaluation des zugrundeliegenden CL-Algorithmus des Modells. Hierbei werden verschiedene Performance-Metriken berechnet, deren Ergebnisse im Logging-Modul gespeichert werden. Das Modell sowie dessen interner Zustand wird im letzten Schritt aktualisiert und der Lernprozess startet von Neuem. [67]

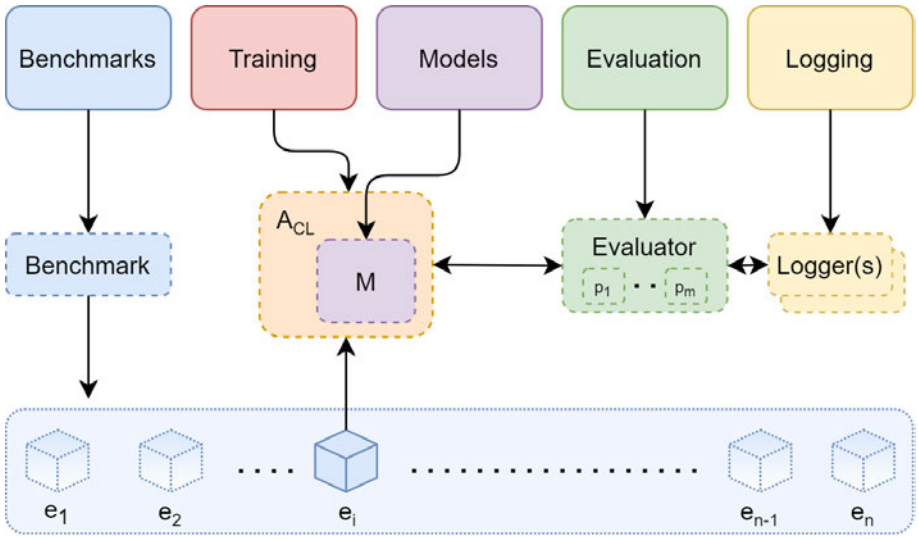


Abbildung 2.16: Es sind die fünf maßgebenden Module des CL-Frameworks Avalanche sowie deren Beziehungen zueinander und zum Datenstrom $e_1 ... e_n$ (hellblau) dargestellt. Hierbei sind die `Benchmarks` in blau, das `Training`-Modul in rot, das `Models`-Modul M in violett, das `Evaluation`-Modul mit $p_1 .. p_n$ in grün und das `Logging`-Modul in gelb visualisiert. [67]

Avalanche ist so konzipiert, dass durch weniger Code in kürzester Zeit ein Prototyp erstellt werden kann. Das Augenmerk liegt hierbei auf der Fehlerreduktion und der verbesserten Reproduzierbarkeit. Die Modularität und die Wiederverwendbarkeit soll gefördert werden, wobei gleichzeitig die Effizienz des Codes, die Skalierbarkeit und die Portabilität erhöht wird. Weiterhin werden die Wirkung und die Nutzbarkeit des CL-Systems unterstützt. [67]

[1] `https://github.com/ContinualAI/avalanche`
[2] `https://avalanche-api.continualai.org/en/latest/index.html`

Benchmarks Die Grundlage der Benchmarks stellt ein nicht-stationärer Strom an Erfahrungen (engl. *Experiences*) dar. Benchmarks sind definiert als Spezifikationen, die die Art und Weise zur Erstellung eines Datenstroms beschreiben. Dabei werden die Ursprungsdatensätze, deren Inhalt, die Anzahl der Beispiele, die Label-Bezeichnung und die Abgrenzung zu anderen Daten aufgezeigt. Die Definition zeichnet sich durch einen Freiheitsgrad aus, der zusätzliche Instanzen von Benchmarks ermöglicht. Daher sind nicht alle Benchmarks durch einen einzigen Datenstrom begrenzt. Es wird aktuell in Trainings- und Testdatenströme unterschieden. Die Erfahrungen entstehen aufgrund von Datenströmen, die iterativ den CL-Algorithmus durchlaufen. Konkret beinhalten die Experiences Informationen über den PyTorch Datensatz, die Task Labels und weitere Benchmark-Spezifikationen. Die Lokalisation von Erfahrungen innerhalb eines Stroms erfolgt über eine interne Indexierung. Diese Art der Lokalisierung ermöglicht die Herstellung von Verbindungen zwischen zwei verwandten Erfahrungen aus verschiedenen Strömen. [67]

Training In der Trainingsphase werden bekannte CL-Strategien sowie ein Satz an Abstraktionen für eine erleichterte Benutzung implementiert. Jede Strategie zeichnet sich durch die Implementation einer eigenen Methode für Training und Evaluation aus. Diese arbeiten auf Grundlage einzelner Erfahrungen oder Teilen des Datenstroms. Die Verwendung der Subklasse `BaseStrategy` stellt generische Schleifen für die Schritte des Trainings und der Evaluation zur Verfügung. Weiterhin können einfache Callback-Mechanismen in Form von Plugins eingefügt werden, die den Lernvorgang während der Trainingsschleife bei vordefinierten Ereignissen abbrechen. Mithilfe der Integration von Plugins können mehrere CL-Strategien miteinander kombiniert werden. [67]

Evaluation Die Leistungsfähigkeit eines Systems erfolgt bei dem Schritt der Evaluation durch eine Überwachung verschiedener Aspekte. Hierbei liegt ein breites Set an Evaluationsexperimenten zugrunde. Bei der Leistungsüberwachung wird einerseits der konkrete Betrachtungsgegenstand definiert und andererseits die Art und Weise der Überwachung (engl. *Logging*) festgelegt. Es existieren Standalone Metriken und Plugin Metriken, die zu bestimmten Zeitpunkten Metrikwerte während der Trainings- und Evaluationsphase ausgeben. Weiterhin kann ein Evaluation-Plugin verwendet werden. Genauer gesagt kann innerhalb der Metriken mithilfe von Argumenten spezifiziert werden, zu welchen Zeitpunkten die Berechnung der Metriken stattfindet. Diese Metrik-Berechnung kann nach jedem Minibatch, nach jeder Epoche (engl. *Epoch*), nach jeder Erfahrung sowie nach jedem Strom von Daten (engl. *Stream*) durchgeführt werden. Eine Epoche bedeutet hierbei eine Iteration über den Trainingsdatensatz. Im Vergleich zu einem Batch, bei dem der gesamte Datensatz für das Training verwendet wird, ist ein Minibatch eine Teilmenge des Batches, die während einer Iteration genutzt wird. Durch die vielseitigen Evaluationsmöglichkeiten kann eine umfangreiche Überwachung verschiedener Facetten der Leistungsfähigkeit erfolgen. Das Ziel hierbei ist die Ermöglichung einer breitgefächerten, experimentellen Bewertung. [67] [68]

Logging Die Logging-Funktionalität zeichnet sich durch eine Überwachung des laufenden Experiments in Echtzeit aus. Während der verschiedenen Phasen des Experiments werden Ergebnisse jeder Plugin-Metrik protokolliert. Avalanche stellt vier verschiedene Arten von Loggern bereit: den `TextLogger`, den `InteractiveLogger`, den `TensorboardLogger` und den `WandBLogger`. Diese unterscheiden sich in der Ausgabeform der folgenden Berichte: Textdatei, Standardausgabe, TensorBoard und Weights & Biases (W&B). Der `TextLogger` speichert die resultierenden Metrikwerte nach jeder Trainingsepoche, nach der Evaluation der Experience sowie am Ende des Evaluationsstroms in eine Textdatei, die den Metriknamen sowie den zugehörigen Wert anzeigt. Der `InteractiveLogger`

stellt die Protokollierungsfunktion für die standardmäßige Ausgabe der Konsole bereit. Er erzeugt die gleiche Ausgabe wie der `TextLogger` mit dem Unterschied, dass zusätzlich das `tqdm`-Paket verwendet wird. Dieses ermöglicht die Ausgabe einer Fortschrittsanzeige für die Nutzenden während der Trainings- und Evaluationsphase. Der `TensorboardLogger` unterstützt die Integration der Tensorboard-Protokollierung, wodurch die Metriken automatisiert durch TensorBoard überwacht werden können. Die Ergebnisse können seitens der Nutzer*innen in Echtzeit geprüft werden. Die Logging-Funktionalität des `TensorboardLogger` zeichnet sich durch die Anzeige von Bildern und komplexeren Ausgaben in Form von Tabellen und Diagrammen aus. Die eigenständige Bibliothek W&B kann barrierefrei in das CL-System integriert werden. Der `WandBLogger` stellt die Protokollierungsfunktion in Form eines dedizierten Dashbords der Anwendung W&B bereit. Dieses Dashboard ist für benutzerdefinierte Projekte ausgelegt und kann mit TensorBoard synchonisiert werden. Sind die Logger fertig konfiguriert, erfolgt die Weitergabe an das `EvaluationPlugin`, das für die Umleitung der Metrikausgaben zu dem jeweiligen Logger zuständig ist. [67] [68]

Models Das Framework Avalanche stellt ein verwendungsfertiges Paket an Architekturen von ML bereit. Es beinhaltet verschiedene Versionen von *Convolutional Neural Network (CNN)*, wie MobileNet, wobei der Fokus auf den Hauptfeatures von Avalanche liegt. Zukünftig ist eine Erweiterung durch fortgeschrittene Architekturen geplant, die speziell auf CL-Anwendungen angepasst sind. [67]

2.6 Softwarebibliotheken, Entwicklungsumgebungen und Datensätze

Im Weiteren wird die Funktionalität relevanter Software und Datensätze untersucht. Aufgrund des Bekanntheitsgrades und der häufigen Verwendung im wissenschaftlichen Bereich geschieht dies in prägnanter Weise.

2.6.1 TensorFlow

TensorFlow TensorFlow stellt eine end-to-end Unterstützung für das Entwickeln und das Trainieren von Maschine Learning Modellen bereit. Die Plattform ist open-source und beinhaltet flexible Tools, Bibliotheken und eine umfangreiche Unterstützung durch die Community. Das Ziel von TensorFlow umfasst eine einfache Implementation von ML-Lösungen, eine umfangreiche und robuste Produktion von Modellen und das Konzipieren leistungsstarker Experimente, die zukünftige Entwicklungen vorantreiben sollen. [69]

TensorBoard TensorBoard ist eine TensorFlow interne Anwendung, die Webanwendungen zum Überprüfen und Verstehen der Prozesse innerhalb von TensorFlow zur Verfügung stellt. Sie läuft vollständig offline und ist kompatibel mit den Browsern Google Chrome und Firefox. Mithilfe von zusammenfassenden Operationen können Daten eines TensorFlow Durchlaufs gesammelt werden. Anschließend kommt es zur Aufnahme und der Erzeugung von Tensoren aus einem TensorFlow Diagramm. Zusammenfassende Daten werden aus dem TensorFlow Programm entnommen und in ein Verzeichnis gespeichert. Sobald ein Absturz erfolgt, werden Ereignisdaten jeglicher Art gesammelt und in einer konsistenten Historie von Ereignissen zusammengefügt. Jeder Lerndurchlauf des Modell kann visuell miteinander verglichen werden, um beispielsweise den Einfluss bei der Variation von Hyperparametern zu verdeutlichen. Es wird eine Liste von Protokollverzeichnissen übergeben, die einer strengen Überwachung unterliegen. Es existiert eine Vielzahl an Möglichkeiten zur Visualisierung der

verschiedenen Lern-Durchläufe. Dazu gehört ein Dashboard, das Skalare, Histogramme, Verteilungen sowie Bild-, Audio-, Video- und Text-Daten verdeutlicht. Weiterhin können Zeitreihen, Diagramme und ein Einbettungsprojektor dargestellt werden. [70]

2.6.2 PyTorch

PyTorch ist eine Python-basierte Bibliothek für *Deep Learning*, dessen Fokus auf die Benutzungsfreundlichkeit sowie die Geschwindigkeit ausgelegt ist. Hierbei stellt sie eine einfache Debugging-Funktionalität bereit. Weiterhin ist PyTorch mit anderen Bibliotheken kompatibel und kann demnach erweitert werden. Neben der Eigenschaft der Interoperabilität, arbeitet sie effizient und unterstützt die Verwendung von Hardware, beispielsweise einer Graphics Processing Unit (GPU). Die gesamte Implementierung ist daher auf eine optimierte Leistung ausgerichtet. Genauer gesagt, wird zum Beispiel ein effizienter C++ Kern verwendet und *Multiprocessing* unterstützt. [71]
Die aktuelle Version 1.12 läuft stabil und ist mit einer umfassenden Dokumentation[3] versehen.

2.6.3 Anaconda Navigator, Jupyter Notebook und Spyder

Anaconda Navigator Das Graphical User Interface (GUI) Anaconda Navigator enthält Anaconda-Distributionen und vereinfacht die Versionsverwaltung sowie das Paket- und Umgebungsmanagement ohne die Benutzung von Befehlszeilenkommandos. Beispielsweise kann der Navigator unter anderem auf die Anwendungen Jupyter Notebook und Scientific PYthon Development EnviRonement (Spyder) zugreifen. Es wird von Windows, macOS und Linux unterstützt und eignet sich für die Verwendung von Data Scientists. [72]

Jupyter Notebook Jupyter Notebook ist eine Entwicklungsumgebung von Python, die als webbasiert und interaktiv charakterisiert wird. Sie eignet sich insbesondere für das Bearbeiten und Ausführen menschenlesbarer Dokumente und für die Anwendung im Bereich der Datenanalyse. Der Austausch von Dokumenten und somit Wissensinhalten wird durch die übersichtliche Verwendung von Jupyter Notebook vereinfacht. Es wird in der aktuellen Version[4] 6.4.11 verwendet. [73]

Spyder Im Vergleich zu Jupyter Notebook ist Spyder eine ebenfalls fortgeschrittene Python-Entwicklungsumgebung, die jedoch nicht webbasiert arbeitet, sondern eine lokale open-source Plattform mit umfassenden Entwicklungswerkzeugen ist. Dazu zählen wissenschaftliche Tools für das Bearbeiten, die Analyse und das Debuggen von Datensätzen, die gleichermaßen integrierte Möglichkeiten der Visualisierungen bieten. Spyder wird in der derzeit aktuellen Version 5.5.3 verwendet, die wiederum auf der stabilen Python Version 3.8 basiert. [74]

2.6.4 MNIST-Datensatz

MNIST Die MNIST-Datenbank ist eine öffentlich zugängliche Datenbank der US-amerikanischen Bundesbehörde National Institute of Standards and Technology (NIST). Sie beinhaltet eine Sammlung von Bildern mit handgeschriebenen Ziffern von 0 bis 9. Diese Ziffern wurden größennormalisiert und innerhalb des Graustufen-Bildes mit einer festen Größe von 28 x 28 Pixeln zentriert. Der Bildbestand

[3] `https://pytorch.org/docs/1.12/`
[4] `https://jupyter-notebook.readthedocs.io/en/stable/changelog.html`

ist in einen Trainingsdatensatz mit 60.000 Beispielen und einen Testdatensatz mit 10.000 Beispielen untergliedert. Die MNIST-Datenbank ist Teil einer größeren NIST-Datenbank, die ebenfalls eine Sammlung monochromer Bilder mit handgeschriebenen Ziffern beinhaltet. [75] [76]

SplitMNIST SplitMNIST ist ein auf dem MNIST-Datensatz basierender klassischer Benchmark des CL-Frameworks Avalanche. Es kann in das CL-Szenario Class-IL eingeordnet werden, bei der standardmäßig die gleiche Klassenanzahl jeder einzelnen Erfahrung zugeordnet wird. [77]

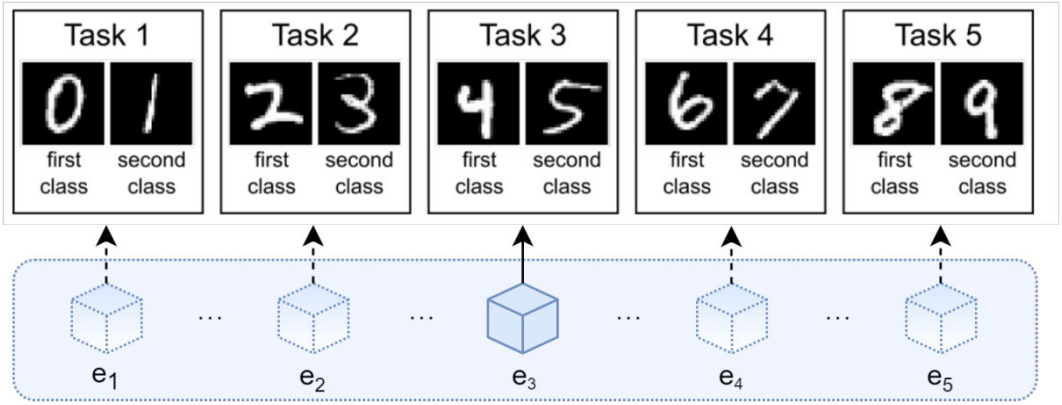


Abbildung 2.17: Die Abbildung zeigt den Benchmark des Datensatzes SplitMNIST anhand eines Datenstroms in der Umgebung des Avalanche CL-Frameworks. [67]

2.7 Weights & Biases

Die eigenständige Bibliothek W&B ist die erste MLOps-Plattform für Entwickler*innen, die eine optimierte Modellerstellung, das Tracking des Experiments, die Datensatzversionierung sowie das Modell-Management und Optimierung von Hyperparametern anbietet. Insbesondere ist die Möglichkeit der interaktiven Datenvisualisierung von Interesse für die Interpretierbarkeit der Erklärungen. Bei der Anwendung von W&B wird die aktuelle Version 0.13.5 verwendet. Die Verwendung ist in der offiziellen Dokumentation[5] beschrieben. [78]

2.8 Kryptographisches Hashing

Zum Schutz sensibler Datensätze werden kryptographische Hash-Algorithmen eingesetzt. Dies geschieht indem ein Hash-Wert berechnet wird. Dieser ist eine kurze Folge von hexadezimalen Werten, die die zugrundeliegenden komplexen Daten eindeutig als solche identifiziert. Er ist somit vergleichbar zu einem Fingerabdruck. Jegliche Änderungen in den Ausgangsdaten führen zu einer Änderung des Hashwertes. Sie werden daher im Bereich der digitalen Forensik als Indikator für Manipulationen eingesetzt. Grundsätzlich sind kryptographische Hash-Funktionen Einwegcodierungen. Sie müssen kollisionsfrei, nicht vorhersehbar und vergleichsweise einfach zu berechnen sein. Ein wichtiger Vertreter der Hash-Algorithmen ist Secure Hash Algorithm 256 (SHA-256)[6]. Ein bekanntes Anwendungsbeispiel für Hash-Funktionen befindet sich im Bereich des Passwort-Hashings. Außerdem kann ein byteweiser Vergleich großer Dateien stattfinden, um Unterschiede zu lokalisieren. [79]

[5] `https://docs.wandb.ai/`
[6] Visualisierung der schrittweisen Funktionalität von SHA-256: `https://sha256algorithm.com/`

3 Methodik

Anhand der soeben vorgestellten Theorie wird im Folgenden die praktische Vorgehensweise bei der Betrachtung der Auditierbarkeit von KI-Systemen verdeutlicht. Zu Beginn wird der Vergleich ausgewählter XAI-Softwarebibliotheken beschrieben. Es folgt ein detaillierter Aufbau des als Grundlage dienenden CL-Modells. Die nachfolgende Programmierung basiert auf der Programmiersprache Python unter Verwendung der interaktiven Entwicklungsumgebungen Jupyter Notebook [73] und Spyder [74]. Zur Verdeutlichung der Modellkonfiguration, der Trainingsgeschwindigkeit sowie des Speicherplatzbedarfs dienen Visualisierungen, die deren Abhängigkeit zueinander beschreiben.

3.1 Vergleich ausgewählter Softwarebibliotheken der Erklärbaren Künstlichen Intelligenz

Für den Vergleich der XAI-Softwarebibliotheken werden geeignete Bewertungskriterien herangezogen, die im Folgenden vorgestellt werden. Die Ergebnisse werden für eine verbesserte Übersicht in Tabellenform dargestellt, wobei die Softwarebibliotheken den Bewertungskriterien gegenübergestellt werden.

Zunächst werden die bei der Implementierung unterstützten ML-Modellformate betrachtet. Speziell wird auf die Unterstützung von Scikit-Learn, TensorFlow (Keras) und PyTorch unter Voraussetzung der Verwendung der aktuellen Version geprüft. Weiterhin wird auf das Kriterium Trainingsdaten überprüft. Hierbei steht die Fragestellung im Vordergrund, ob diese Daten für das Generieren einer Erklärung notwendig sind. Die Bedingung ist erfüllt, wenn Trainingsdaten für das Generieren von Erklärungen benötigt werden. Anschließend wird die Visualisierungsfähigkeit einer Softwarebibliothek nach Vorhandensein oder Nicht-Vorhandensein beurteilt. Ob eine detailreiche Dokumentation der Bibliothek inklusive angebotener Funktionen sowie Methoden existiert, wird ebenfalls betrachtet. Ist diese Bedingung erfüllt, erfolgt der Verweis auf die offizielle Dokumentation mithilfe einer Fußnote. Das Bewertungskriterium Beispiele gibt an, für welchen Datentyp (Tabelle, Bild, Text) Beispiele für nachvollziehbare Erklärungen der Ergebnisse verfügbar sind. Das Kriterium Metriken zeigt das Vorhandensein von Metriken zur Evaluation der resultierenden Erklärungen. Bei der Softwareaktualität wird in letztes Release und letzter Commit bei der Entwicklungsplattform GitHub[7] unterschieden. Letztes Release zeigt das Veröffentlichungsdatum der neusten Version sowie die aktuelle Versionsnummer an. Das letzte Commit ist das Durchführungsdatum der letzten Codeänderung oder Fehlerbehebung. [36, S. 37 f.]

[7] `https://github.com/`

3.2 Aufbau und Konfiguration des Continual Learning Modells unter Avalanche

Mithilfe des in Abschnitt 2.5.3 vorgestellten CL-Frameworks Avalanche wird zu Beginn eine KI konstruiert, die die Grundlage für die nachfolgenden Betrachtungen darstellt. Die Art und Weise des Aufbaus wird im Folgenden schrittweise anhand ausgewählter Codebeispiele erklärt. Hierbei werden nur Ausschnitte und nicht der vollständige Programmcode gezeigt. Die Zeilenangaben beziehen sich jeweils auf den nächstfolgenden Programmcode-Abschnitt, welcher durch eine hellblaue Umgebung gekennzeichnet ist. Es existieren fünf Jupyter-Notebook-Dateien, die den nachfolgend beschriebenen Programmcode beinhalten. Sie unterscheiden sich lediglich in der Konfiguration des Parameters `hidden_layers` innerhalb der Definition für das SimpleMLP Modell sowie den Bezeichnungen für die Dateipfade und den Logdateien. Ansonsten ist das Vorgehen für alle fünf Modellgrößen gleich.

Konfigurieren der Jupyter Notebook Umgebung Vor der Ausführung des Codes wird der Anaconda Navigator gestartet und ein neues Environment angelegt, in dem die notwendigen Packages der aktuellsten, aber gleichzeitig lauffähigen Version für das Projekt installiert werden. Nach der Umstellung der Arbeitsumgebung wird über die Windows-Eingabeaufforderung CMD.exe Prompt (v0.1.1) in das zuvor angelegte Arbeitsverzeichnis gewechselt. Anschließend kann Jupyter Notebook anhand Zeile 1 über das Terminal gestartet werden. Das Limit der Konfigurationsvariable IOPub-Datenrate `iopub_data_rate_limit` wird auf $1.0e10$ Bytes erweitert. Weiterhin wird in Zeile 2 das RAM-Limit `max_buffer_size` auf $8.0e9$ Bytes vergrößert.

```
1 jupyter notebook --NotebookApp.iopub_data_rate_limit = 1.0e10 \
2   --NotebookApp.max_buffer_size = 8.0e9
```

Importe und Vorbetrachtungen Zu Beginn muss eine Reihe von Importen durchgeführt werden, beispielsweise Avalanche und Torch Importe, die für die Funktionalität des CL-Modells notwendig sind. Die Verwendung des CL-Frameworks Avalanche setzt die Installation der Avalanche Bibliothek der aktuellen Version 0.2.1 voraus [68]. Weiterhin muss die Softwarebibliothek W&B der aktuellen Version v0.13.5 installiert werden. Damit der vollständige Tensor mithilfe des `print`-Befehls ausgegeben werden kann, müssen die Ausgabeoptionen angepasst werden. Außerdem wird bei der Vorbetrachtung getestet, ob eine GPU verfügbar ist und verwendet werden kann. Im Folgenden wird die CPU des lokalen Rechners bei der Programmdurchführung verwendet.

SimpleMLP Im Rahmen der Konfiguration der zu betrachteten MLP-Modelle wird die Größe der Beispieldaten des Eingabe-Layers (`input_size`) sowie die HL Größe (`hidden_size`) festgelegt. Hierbei kann die Anzahl n der HL (`hidden_layers`) und die Abbruchrate (`drop_rate`) flexibel bestimmt werden. Konkret wurden in Zeile 1 der objektorientierten Initialisierungsfunktion 10 Eingabeklassen (eine pro Ziffer 0 bis 9) definiert, deren Eingaben der Größe $28 * 28$ (insgesamt 784) den 512 Neuronen des HL hinzugefügt werden. Die Layer des SimpleMLP-Modells werden anschließend zur Optimierung der Parameter pro Instanz definiert. Dies geschieht mithilfe eines sequenziellen Containers (`nn.Sequential`, siehe Zeile 2) der Python Bibliothek `torch.nn`, der die Module `nn.Linear` in Zeile 3, `nn.ReLU` in Zeile 4 und `nn.Dropout` in Zeile 5 unter Beachtung dieser Reihenfolge beinhaltet. Die eingehenden Daten der Größe $25 * 25$ werden mithilfe linearer Transformation innerhalb des Neurons in einen Tensor der Größe 512 umgewandelt. Die Aktivierungsfunktion *Rectified Linear Unit* (ReLU) wendet elementweise eine lineare Transformation auf die eingehenden Daten an. Bei

der Abbruchfunktion werden während der Trainingsphase zufällige Elemente des Eingabetensors mit der Wahrscheinlichkeit von 0 % ($p = 0$) auf null gesetzt. Dies geschieht unter Verwendung von Stichproben aus einer Bernoulli-Verteilung. Zusammenfassend erfolgt eine sequenzielle Verkettung der Ausgänge des einen Moduls mit dem Eingang des nachfolgenden Moduls. Dies geschieht so lange, bis der Ausgang des letzten Moduls erreicht ist. Der entsprechende Wert wird ausgegeben. [80]

```
def __init__(self, num_classes = 10, input_size = 28 * 28, hidden_size = 512,
    hidden_layers = n, drop_rate = 0):
  layers = nn.Sequential(*(
    nn.Linear(input_size, hidden_size),
    nn.ReLU(inplace = True),
    nn.Dropout(p = drop_rate)))
```

Szenario Die Datenbasis des Benchmark-Moduls stellt der beliebte CL-Benchmark SplitMNIST (Vergleich Zeile 1) des Submoduls `classic` dar. Dieser besitzt eine Anzahl von fünf Experiences sowie einen Startwert von 42. Zusätzlich werden der Trainingsdatenstrom `train_stream` in Zeile 2 sowie der Testdatenstrom `test_stream` in Zeile 3 aus dem SplitMNIST-Benchmark erzeugt. [68]

```
benchmark = SplitMNIST(n_experiences = 5, return_task_id = True, seed = 42)
train_stream = benchmark.train_stream
test_stream = benchmark.test_stream
```

Continual Learning Modell Im Folgenden wird die CL-Strategie mit drei vorgeschriebenen Argumenten instanziiert. Für das erste Argument ML-Modell wird `SimpleMLP` verwendet. In Zeile 1 wird der Variable `model` das ML-Modell `SimpleMLP` mit dem Parameter `num_classes` zugewiesen, welcher die Anzahl der ausgegebenen Klassen als `benchmark.n_classes` definiert. Weiterhin wird in Zeile 2 anhand des zuvor ausgesuchten Modells als zweites Argument der PyTorch-Optimizer Stochastic Gradient Descent (SGD) gewählt, der einen stochastischen Gradientenabstieg implementiert [81]. Das dritte Argument in Zeile 3 ist die Verlustfunktion (`Criterion`) namens `CrossEntropyLoss`, die den (Kreuz-)Entropieverlust zwischen den Eingabevariablen und den Zielvariablen errechnet [82]. [68] [83]

```
model = SimpleMLP(num_classes=benchmark.n_classes)
optimizer = SGD(model.parameters(), lr = 0.001, momentum = 0.9)
criterion = CrossEntropyLoss()
```

Logging Zu Beginn wird in Zeile 1 eine zunächst leere Liste namens `Logger` erstellt, die die nachfolgenden Protokollierungstypen sowie deren Hyperparameter speichert. Das Logging-Modul stellt hierfür den `TextLogger` in Zeile 2, den `InteractiveLogger` in Zeile 3, den `TensorboardLogger` in Zeile 4 und den `WandBLogger` in Zeile 5 bereit. Für die Verwendung des `WandBLogger` ist ein W&B Account[8] notwendig, da ein personalisierter *Key* bei dem Import abgefragt wird. Um externen Nutzer*innen ohne W&B-Konto zu ermöglichen die Diagramme in der privaten Nutzungsoberfläche einzusehen, wird der Parameter `anonymous` auf `allow` gesetzt. [68]

[8] `https://wandb.ai/site`

```
loggers = []
loggers.append(TextLogger(open("weights_log_nhl.txt", "a")))
loggers.append(InteractiveLogger())
loggers.append(TensorboardLogger())
loggers.append(WandBLogger(project_name="Avalanche-CL", run_name="SimpleMLP-nHL"))
run = wandb.init(project="Avalanche-CL", name="SimpleMLP-nHL", anonymous="allow")
```

Evaluation In den Zeilen 1 bis 3 werden die drei Plugins `EarlyStopping`, `Replay` und `Elastic Weight Consolidation (EWC)` definiert. Das Plugin `EarlyStopping` wird in Zeile 1 mit den Parametern `patience` und `val_stream_name` versehen. `patience` ist die Anzahl der Epochen, die abgewartet wird, bis das Training beendet wird. In diesem Fall sind es 10 Epochen. `val_stream_name` gibt den Namen des Validation-Stream (`train`) an, der die Grundlage für die Leistungsentwicklung bei den verwendeten Metriken bildet. Das `Replay`-Plugin in Zeile 2 wird mit dem Parameter `mem_size` versehen, der die gesamte Anzahl der zu speichernden Muster (100) im externen Speicher definiert. Als drittes Plugin wird `EWC` in Zeile 3 mit dem Hyperparameter und dem Wert `ewc_lamda=0.001` initialisiert, der die Strafe innerhalb des Gesamtverlustes gewichtet. Das in Zeile 4 definierte `eval_plugin` stellt die zuvor definierten Logger sowie die verwendete Benchmark-Instanz als Eingabe der Klasse `EvaluationPlugin` bereit. [68] [83]

```
early_stopping = EarlyStoppingPlugin(patience = 10, val_stream_name = "train")
replay = ReplayPlugin(mem_size = 100)
ewc = EWCPlugin(ewc_lambda = 0.001)
eval_plugin = EvaluationPlugin(loggers = loggers, collect_all = True, benchmark =
    benchmark, strict_checks = False)
```

Strategie Als einfachste CL-Strategie wird in Zeile 1 `Naive` verwendet, die in Zeile 2 mit den Parametern des Modell-Trainings `model`, `optimizer` und `criterion` gefüllt wird. Zusätzlich zu dem Modell-Training werden in Zeile 3 die Plugins `EarlyStopping`, `Replay` und `EWC` verwendet, um einen überwachten Trainingsprozess innerhalb der CL-Strategie zu erzeugen. Die in Zeile 4 auftretenden Attribute `train_mb_size` und `eval_mb_size` definieren die Größe des Trainings- und Evaluations-Minibatch mit dem Wert 100. `train_epochs` ist die Anzahl der Trainingsepochen, die mit dem Wert 5 initialisiert wird. Der Hyperparameter `eval_every=0` ruft die Evaluationsauswertung nach dem Modelltraining auf. Als Basis für die Evaluation wird das zuvor definierte `eval_plugin` in Zeile 5 übergeben und die gesammelten Ergebnisse über den `evaluator` an die Strategie zurückgegeben. [68] [83]

```
cl_strategy = Naive(
    model, optimizer, criterion, device=device,
    plugins = [early_stopping, replay, ewc],
    train_mb_size = 100, train_epochs = 5, eval_mb_size = 100, eval_every = 0,
    evaluator = eval_plugin)
```

Training Vor dem Trainingsbeginn werden sowohl das TensorBoard Dashboard (Zeile 1 und 2) als auch das W&B Dashboard (Zeile 3) geladen. Beide können während der Trainingsphase laufend aktualisiert werden und zeigen aktuelle Informationen und Visualisierungen des Modelltrainings.

```
1 %load_ext tensorboard
2 %tensorboard --logdir tb_data --port 6006
3 run.display(height = 720);
```

Unmittelbar vor Beginn der Trainingsschleife werden in den Zeilen 1 bis 5 leere Listen für Hashwerte, Dateigrößen, Zeitpunkte und die Metrikergebnisse initialisiert. Die Listen in Zeile 1, 3 und 4 dienen als Grundlage für den `TextLogger`. Anschließend wird in Zeile 7 die aktuelle Zeit `start` gespeichert und in Zeile 8 festgelegt, dass die Modell-Parameter mit `wandb.watch()` in W&B gespeichert werden. Das kontinuierliche Training des Modells geschieht ab Zeile 10 durch das Iterieren des vom Szenario bereitgestellten Trainingsdatenstroms `train_stream`. `train` und `eval` akzeptieren beide als Eingabe eine Liste von $n = 5$ Experiences auf Grundlage des SplitMNIST-Datensatzes. Das eigentliche Modelltraining geschieht dann in Zeile 11. Die darauffolgenden Codezeilen definieren wie die Dateigröße der Weights-Logdatei in Bytes unter Angabe des Dateipfades nach jeder Trainingserfahrung ermittelt und in der Liste `log_size` (Vergleich Zeile 2) gespeichert wird. Es folgt die Berechnung der Metrikwerte auf den gesamten Testdatensatz des zuvor definierten Benchmarks. Diese Metrikwerte werden in der Liste `results` (Vergleich Zeile 5) gespeichert. Nach Beendigung der Trainingsschleife wird in Zeile 16 erneut die aktuelle Zeit gemessen (`end`) und in Zeile 18 die gesamte Laufzeit des Modelltrainings in Minuten (`runtime`) ausgegeben. [68] [83]

```
1  hash_values = []
2  log_size = []
3  log_size_extended = []
4  time_checkpoints = []
5  results = []
6
7  start = time.time()
8  wandb.watch(model, criterion, log = "all", log_freq = 10, log_graph = True)
9
10 for experience in benchmark.train_stream:
11    cl_strategy.train(experience)
12    file_size = os.path.getsize(r"E:Jupyter_Notebook\weights_log_nhl.txt")
13    log_size.append(file_size)
14    results.append(cl_strategy.eval(benchmark.test_stream))
15
16 end = time.time()
17 runtime = (end-start) / 60
18 print(f"Laufzeit in Minuten: {round(runtime, 2)}")
```

3.3 Interne Logging Funktionalität des Continual Learning Modells

Die Auswertung der Logging Funktionalität von den in Avalanche integrierten Loggern `TextLogger`, `InteractiveLogger`, `TensorboardLogger` und `WandBLogger` erfolgt anhand von Abbildungen, die bei der Ausgabe sowie nach dem Trainingsprozess erzeugt werden. Diese werden inhaltlich in Textform anhand geeigneter Kriterien miteinander verglichen. Abschließend wird die Eignung für das geplante Vorgehen bei der Auditierung von KI-Systemen bewertet.

3.3.1 Konfiguration der Logging-Module von Avalanche

Der `TextLogger` dient als Grundlage zum Speichern der Modellgewichte. Der ursprünglich von Avalanche bereitgestellte Programmcode für das Protokollieren in eine Textdatei wird dahingehend verändert und befindet sich in Anhang A.2. Wie sich diese Anpassungen genau gestalten, wird im nächsten Kapitel 3.3.3 verdeutlicht.

Die ursprüngliche Konfiguration des `InteractiveLogger` seitens Avalanche wurde nicht verändert, siehe Anhang A.3. Bei dem `TensorboardLogger` (Vergleich Anhang A.4) und dem `WandBLogger` (Vergleich Anhang A.5) wurden lediglich geringfügige Code-Anpassungen bezüglich der Pfadangaben sowie der initialen Definition vorgenommen. Speziell wurde bei dem `WandbLogger` der Projektname (`project_name`) und der Name der aktuellen Ausführung (`run_name`) konkretisiert sowie die Option zum Protokollieren der Modellgewichte als W&B-Artefakte (`log_artifacts`) erlaubt. Für jeden Durchlauf des Programms (engl. *run*) werden die Daten in W&B in einem separaten Ordner `wandb` mit dem Erstellungszeitpunkt sowie einer individuellen Kennung gespeichert (zum Beispiel `run-20221127_202003-2qvnuwtt`).

3.3.2 Verlauf der Gewichtsänderung

Dieser Abschnitt demonstriert exemplarisch den Verlauf der Gewichtsänderungen als Grundlage für das Loggen der Modellgewichte und die Deltabildung. Konkret wird dieser am Beispiel des CL-Modells SimpleMLP mit einem HL beschrieben und analysiert. Hierbei wird zunächst die Änderung des ersten Gewichtswertes der Layer `features[0]` und `classifier` zum Zeitpunkt der Änderung in Tabellenform erfasst. Stellvertretend dient jeweils der erste Gewichtswert von insgesamt $512 * 784 = 401.408$ Werten des Layers `features[0]` und $10 * 512 = 5120$ Werten des Layers `classifier` als Repräsentant für den Verlauf der Gewichtsänderung während des Modelltrainings. Neben den eigentlichen Gewichtswerten wird zusätzlich die schrittweise Änderung innerhalb der Werte innerhalb des Layers dokumentiert. Der Hauptaugenmerk liegt jedoch auf der Berechnung des Delta-Wertes (Δ) zwischen den beiden Layern nach der letzten Änderung. Dieser Wert nach dem Training wird separat hervorgehoben. Das Vorgehen erfolgt anhand der in Kapitel 3.3.5 beschriebenen Berechnung der Delta-Werte, bei der benachbarte Layer-Werte zwischen zwei Layern voneinander subtrahiert werden. Der Verlauf bis zur Deltawertberechnung nach der letzten Änderung wird ebenfalls erfasst und formal beschrieben. Der berechnete Deltawert ist zum Vergleich als erster Eintrag in der Datei `delta_weights_1hl` gespeichert. Die verbleibenden Einträge des Delta-Log entstehen analog.

3.3.3 Erstellung der Weights-Logdatei während des Modelltrainings

Als Voraussetzung für die Funktionalität des `TextLogger` werden sowohl die initialen Layernamen `weights_names` (siehe Zeile 4) als auch die Gewichte jedes einzelnen Layers `weights_list` (siehe Zeile 12) vor dem Trainingsbeginn in der richtigen Reihenfolge in einer Liste gespeichert. Die Namen der Modell-Layer können über den Befehl zur Ausgabe allgemeiner Parameterinformationen eingesehen werden. Nacheinander werden sie unter anderem mithilfe der `while`-Schleife in Zeile 6 der Liste `weights_names` angehängt. Beispielsweise verbirgt sich hinter dem Ausdruck `model.features[0]` der Name des Input-Layers. Da sich lediglich der Input-Layer sowie der Output-Layer in ihrer Namensgebung unterscheiden, werden sie außerhalb der Schleife der Liste angefügt (Vergleich Zeilen 5 und 10). Dies geschieht analog zu der Speicherung der initialen Gewichtstensoren. Sie werden beispielsweise mithilfe des Befehls `model.features[0].weight.data` in Zeile 13 einzeln extrahiert und der Liste `weights_list` angehängt.

```
1  i=3
2  j=1
3
4  weights_names = []
5  weights_names.append("model.features[0]")
6  while i in range(3, len(model.features)) and j < len(model.features):
7      weights_names.append(f"model.features[{i}][0]")
8      i+=1
9      j+=1
10 weights_names.append("model.classifier")
11
12 weights_list = []
13 weights_list.append(model.features[0].weight.data)
14 while i in range(3, len(model.features)) and j < len(model.features):
15     weights_list.append(model.features[i][0].weight.data)
16     i+=1
17     j+=1
18 weights_list.append(model.classifier.weight.data)
```

Da die Längen der beiden Listen `weights_names` und `weights_list` übereinstimmen, können die Inhalte anschließend in einem Python *Dictionary* namens `log_dict` (siehe Zeile 2) mithilfe der `zip()`-Funktion in Zeile 1 verbunden werden.

```
1  a = zip(weights_names, weights_list)
2  log_dict = dict(a)
```

Dem Avalanche-internen `TextLogger` werden anschließend bezüglich der Erstellung der Weights-Logging Datei während des Modelltrainings Erweiterungen hinzugefügt. Die gespeicherten Informationen vor und nach dem Modelltraining sowie vor und nach der Modellevaluation unterscheiden sich hinsichtlich der Gewichte. Sie werden nur vor und nach dem Training geloggt, nicht jedoch bei der Evaluation. Anhand des nachfolgenden Programmcodes wird die Logging-Funktionalität beschrieben.

```
1  def before_training(self, strategy: "SupervisedTemplate", metric_values:
       List["MetricValue"], **kwargs):
2      super().before_training(strategy, metric_values, **kwargs)
3      with open(r"weights_log_nhl.txt", "a+") as f1,
           open(r"info_weights_log_nhl.txt", "a+") as f2:
4          print("-- >> Beginn der Trainingsphase << --", file=self.file,
               flush=True)
5          for key, value in log_dict.items():
6              i=0
7              while i < len(log_dict):
8                  a = f"Gewicht {i+1} von {len(weights_names)} des Layers {key}
                       ({weights_list[i].shape}) vor der Trainingsphase: \n {value}"
9                  f1.write("%s\n" % a)
10                 file_size = os.path.getsize(r"C:\SimpleMLP\weights_log_nhl.txt")
11                 b = f"Dateigröße der Logdatei vor der Trainingsphase (Gewicht
                       {i+1}): {file_size} Bytes"
12                 f2.write("%s\n" % b)
13                 i+=1
```

Im Rahmen der Protokollierungsfunktionalität wird zunächst in Zeile 2 festgestellt, welche der vier Phasen zum jetzigen Zeitpunkt aktuell vorliegt. Daraufhin werden in Zeile 3 die `weights_log.txt` Datei des jeweiligen CL-Modells sowie die `info_weights_log.txt` Datei geöffnet. Bei dem Öffnen beider Dateien wird die Option `a+` (engl. *append and read*) übergeben, die definiert, dass die Datei gelesen und ihr Inhalt angehängt werden darf. Da beide Logdateien zu Beginn nicht existieren, werden sie aufgrund der vorangegangenen Konfigurationen erstellt. Das zuvor initialisierte `log_dict` bildet ab Zeile 5 die Grundlage für das Speichern der Modellgewichte. Konkret werden vergleichbar zu Zeile 8 und 9 die Nummer, der Gewichtsname, der Layername, die Tensorform und der eigentliche Gewichtswert als Zeichenkette (engl. *string*) in der ersten Datei gespeichert. In Zeile 10 wird die Dateigröße `file_size` ermittelt. Die zweite Datei speichert die Größe der `weights_log_nhl.txt` anschließend in regelmäßigen Abständen ebenfalls als String (siehe Zeile 11 und 12).

3.3.4 Information Logging

Parallel zur Erstellung der Weights-Logdatei während des Modelltrainings wird im Vergleich zum vorangegangenen Kapitel für jedes der fünf SimpleMLP-Modelle die Datei `info_weights_log_nhl.txt` erstellt. Zur Trainingszeit werden die ergänzenden Informationen zur aktuellen Experience und zum Start und Ende der Trainings- und Evaluationsphase des jeweiligen Layers übergeben. Diese Daten sind auch in der Datei `weights_log_nhl.txt` wiederzufinden. Im Gegensatz zu dieser enthält die Info-Logdatei keine Gewichtstensoren. Der Fokus liegt hierbei auf der Dokumentation der aktuellen Dateigröße sowie der darauffolgenden regelmäßigen Speicherung eines Hashwertes. Dieser repräsentiert den aktuellen Zustand der Weights-Logdatei. Abschließend werden Evaluationsmetriken am Ende jeder Experience übergeben.

3.3.5 Erstellung der Delta-Logdatei nach dem Modelltraining

Die Bildung der Gewicht-Delta-Werte nach der Trainingsphase bildet die Grundlage für deren Protokollierung. Die Vorgehensweise bei der Erstellung findet schrittweise statt. Zunächst wird anhand einer `for`-Schleife getestet, ob die Formen (engl. *shapes*) benachbarter Gewichte zwischen zwei benachbarten Layern übereinstimmen. Hierbei ist die Ausgabe der Parameter-Information hilfreich. Die Bias Parameter bleiben bei der Betrachtung der Modellparameter außen vor.

```
for param_tensor in model.state_dict():
    x = param_tensor.endswith(".weight")
    if x is True:
        print(param_tensor, "\t", model.state_dict()[param_tensor].size())
```

Wenn die Formen nicht übereinstimmen, wird der kleinere Tensor, das heißt die Gewichts-Matrix mit der kleineren Form, durch eine bedingte Anweisung mit Nullen aufgefüllt. Dieses sogenannte *Padding* orientiert sich an der Größe des Tensors mit den höheren Shape-Werten. War das Padding des Tensors erfolgreich, erfolgt eine erneute Überprüfung der Formen beider Tensoren.

```
for i in range(len(weights_dict)-1):
    x = weights_dict[f"weights_{i}"].shape == weights_dict[f"weights_{i+1}"].shape
    if x is False:
        a = weights_dict[f"weights_{i+1}"].shape == torch.Size([512, 512])
        if a is True:
            weights_dict.update({f"weights_{i+1}": F.pad(input =
                weights_dict[f"weights_{i+1}"], pad = (0,272,0,0), mode =
                "constant", value = 0)})
        b = weights_dict[f"weights_{i+1}"].shape == torch.Size([10, 512])
        if b is True:
            weights_dict.update({f"weights_{i+1}": F.pad(input =
                weights_dict[f"weights_{i+1}"], pad = (0,272,0,502), mode =
                "constant", value = 0)})
    else:
        pass
    i+=1
```

Stimmen die Tensor-Shapes nach der zuvor durchgeführten Überprüfung überein, kann die elementweise Subtraktion in Zeile 7 angewendet werden.

```
tensor_deltas = []
for i in range(len(weights_dict)-1):
   x = weights_dict[f"weights_{i}"].shape == weights_dict[f"weights_{i+1}"].shape
   if x is False:
      pass
   else:
      tensor_deltas.append(abs(weights_dict[f"weights_{i}"] -
         weights_dict[f"weights_{i+1}"]))
```

Hinweis: Bei wiederholtem Ausführen des Programmcodes zur Erstellung der Delta-Weights, beispielsweise aufgrund von Debugging-Schritten, muss das Weights Dictionary `log_dict` erneut initialisiert werden, damit die Shapes auf ihren Originalzustand zurückgesetzt werden. Weiterhin müssen die neu initialisierten Listen geleert und die erstellte Datei `delta_weights.txt` gelöscht werden. Ansonsten werden die neuen Ergebnisse an die Alten innerhalb der Datei angehängt. Bestehende Daten werden mit der vorliegenden Konfiguration nicht überschrieben.

Für die fehlerfreie Berechnung der Delta-Werte zwischen aufeinanderfolgenden Gewichtswerten, müssen die ursprünglichen Formen der Tensoren zum Teil wiederhergestellt werden. Hierfür wird die ursprüngliche Form *b* in Zeile 2 mit der aktuellen Form in Zeile 3 verglichen. Dies geschieht in Python durch sogenanntes *Slicing*. Hierbei werden anhand Zeile 5 die überschüssigen Nullen abgetrennt.

```
1 for i in range(1, len(tensor_deltas)):
2     b = torch.Size([512, 512])
3     y = b == tensor_deltas[i].shape
4     if y is False:
5         tensor_deltas[i] = np.array(tensor_deltas[i])[:, :512]
6     else:
7         pass
```

Im Rahmen der Speicherung werden die Tensoren bei Modellen mit einem HL in `tensor_deltas` in das Dataframe `df_deltas` überführt. Die Delta-Werte des Dataframes werden auf vier Nachkommastellen gerundet und in der zuvor geöffneten Textdatei `delta_weights_nhl.txt` als String gespeichert. Gleichzeitig wird das Vorkommen der Nullen (0.0000) innerhalb des Dataframes untersucht. Das Dataframe wird daraufhin in ein Numpy Array umgewandelt. Die genaue Anzahl wird anschließend in der Variable *W* gespeichert und die Datei wird geschlossen.

```
1 for i in range(len(weights_dict)-1):
2     df_deltas = pd.DataFrame(tensor_deltas[i])
3     with open("delta_weights_nhl.txt", "a+") as file:
4         np.savetxt(file, df_deltas.values.round(4), fmt="%s")
5         arr = np.array(df_deltas.round(4))
6         W = len(arr[arr == 0.0000])
7         file.close()
```

Besteht das CL-Modell aus mehr als einem HL werden anhand Zeile 1 Dataframes für jeden Eintrag *i* in `tensor_deltas[i]` erstellt. Diese werden in einer Liste gespeichert (siehe Zeile 2), um sie anschließend in Zeile 3 mit der Funktion `pd.concat()` entlang der Achse `axis=1` miteinander zu verketten. Erst dann kann die zuvor beschriebene Vorgehensweise zur Speicherung der Tensorwerte angewendet werden.

```
1 df_deltas_i = pd.DataFrame(tensor_deltas[i])
2 frames = [df_deltas_0, ..., df_deltas_i]
3 df_deltas = pd.concat(frames, axis=1)
```

Eine Übersicht der Inhalte des Dataframes wird in Zeile 1 ausgegeben. Detaillierte Informationen (`count`, `mean`, `std`, `min`, 25 %, 50 %, 75 %, `max`) können mithilfe des `df_deltas.describe()` Befehls in Zeile 2 angezeigt werden. Zusätzlich werden diese Gewichte anhand der Zeilen 3 und 4 in einer Python-Datei mit der Dateiendung `.pth` gespeichert.

```
1 df_deltas.round(4)
2 df_deltas.describe()
3 filename_weights = "delta_weights_nhl.pth"
4 torch.save(tensor_deltas, filename_weights)
```

3.4 Visualisierung der Abhängigkeiten zwischen Modellkonfiguration, Trainingsgeschwindigkeit und Speicherplatzbedarf

Wahl der Entwicklungsumgebung Die folgenden Visualisierungen werden bis auf eine Ausnahme innerhalb der Entwicklungsumgebung Jupyter Notebook durchgeführt.

Allgemeines Die folgenden Visualisierungen sind durch drei verschiedene Betrachtungsweisen charakterisiert. Für jeden der drei Gesichtspunkte werden fünf verschiedene Darstellungen erstellt, die auf der variierenden Größe des zugrundeliegenden CL-Modells basieren. Daraus resultieren 15 Grafiken, die grundlegend für die Diskussion der Ergebnisse sind. Für die Erstellung aller nachfolgenden Visualisierungen wird die Python-Bibliothek `Matplotlib` sowie deren Funktionen verwendet, die in den Zeilen 1 bis 4 importiert werden. Sie können mithilfe des Befehls in Zeile 5 innerhalb des Jupyter Notebooks dargestellt werden. [84]

```
1 import matplotlib as mpl
2 import matplotlib.pyplot as plt
3 import matplotlib.font_manager as font_manager
4 from matplotlib.pyplot import Figure
5 %matplotlib inline
```

Grundlegender Programmcode Die Visualisierung der drei verschiedenen Betrachtungsweisen gleicht sich in grundlegenden Konfigurationsschritten. Zunächst wird in Zeile 3 ein `figure`-Objekt mit einer resultierenden Breite und Höhe von 10 cm x 4 cm erstellt. Anschließend wird in den Zeilen 5 bis 8 ein Gitternetz im Hintergrund des Funktionsgraphen definiert, das durch seine hellgraue Färbung sowie die gestrichelte Linienführung und eine Breite von 1 (entspricht einem Punkt) charakterisiert ist. Hierbei wird mithilfe einer `for`-Schleife von jedem x- und y-Wert auf der entsprechenden Achse eine Linie zu dem Datenpunkt (x, y) erstellt. Das eigentliche Diagramm wird mithilfe des Befehls in Zeile 10 definiert. Hierbei werden die in Zeile 1 und 2 initialisierten Variablen x und y als Abszisse und Ordinate übergeben. Weiterhin wird durch das dritte Argument `"o-"` definiert, dass jeder Datenpunkt der Größe 2 (siehe Argument fünf) einzeln durch einen ausgefüllten Kreis gekennzeichnet wird. Diese Kreise sind durch die Linie des Funktionsgraphen miteinander verbunden. Der Graph wird durch eine dunkelblaue Farbgebung charakterisiert. Die Linie des Funktionsgraphen erhält außerdem ein benutzerdefiniertes Label in Argument sechs. Zusätzlich kann der Wertebereich in Zeile 11 sowie der Definitionsbereich in Zeile 12 definiert werden. Neben dem Titel des Diagramms (`ax.set_title()`, siehe Zeile 14) kann ebenfalls die benutzerdefinierte Bezeichnung beider Achsen (`ax.set_xlabel()` in Zeile 15 und `ax.set_ylabel()`) in Zeile 16 erfolgen. Die Schriftart der x- und y-Label ist `serif` und sie werden in der Größe 10 dargestellt. Das Titel-Label erhält bei gleicher Schriftart die Größe 11. Die Anpassung der graphischen Darstellung äußert sich außerdem durch ausgewählte y-Werte, die in Textform entlang des Funktionsgraphen angezeigt werden (Vergleich Zeilen 18 und 19). Die Zeilen 20 und 21 verdeutlichen die Definition der Legende unter Angabe der Positionierung sowie deren Schriftart. Abschließend wird das `figure`-Objekt, das heißt die eigentliche Grafik, anhand der lokalen Pfadangabe in Zeile 23 gespeichert. Um diese ebenfalls in der Standardausgabe von Jupyter Notebook zu sehen, wird zusätzlich der Befehl in Zeile 24 ausgeführt. [84]

```
1  x = x-values
2  y = y-values
3  fig, ax = plt.subplots(figsize=(10, 4))
4
5  for i in list(range(len(y))):
6      plt.axvline(x = i, linestyle = "dashed", color = "lightgray", linewidth = 1)
7  for i in y:
8      plt.axhline(y = i, linestyle = "dashed", color = "lightgray", linewidth = 1)
9
10 ax.plot(x, y, "o-", color = "midnightblue", markersize = 4, label = "plot label")
11 ax.set_xlim(xmin, xmax)
12 ax.set_xlim(ymin, ymax)
13
14 ax.set_title("title", fontfamily = "serif", fontsize = 11)
15 ax.set_xlabel("x label", fontfamily = "serif", fontsize = 10)
16 ax.set_ylabel("y label", fontfamily = "serif", fontsize = 10)
17
18 for index in range(len(x)):
19     ax.text(x[index], y[index], y[index], rotation = "horizontal", fontfamily =
           "serif", fontsize = 10)
20 font = font_manager.FontProperties(family = "Times New Roman", style = "normal",
       size = 10)
21 ax.legend(loc = "best", prop = font)
22
23 fig.savefig("Bilder/visualisation-name.png")
24 plt.show()
```

3.4.1 Speichergröße der Weights-Logdatei vs. Trainingszyklus

Zu Beginn wird die Abhängigkeit der Speicherplatzgröße der Weights-Logdatei zu der Lerniteration während der Trainingsphase betrachtet. Die Speichergröße wird in MB oder GB auf zwei Nachkommastellen gerundet angegeben. Hierbei soll die Art und Weise des ansteigenden Speicherplatzbedarfs mit zunehmender Trainingszeit analysiert werden. Daher ist ebenfalls der Verlauf der Laufzeit während des Modelltrainings abgebildet, die in Minuten angegeben wird. (Die Ausnahme bildet die Laufzeit bei 100 HL, welche aufgrund der Größenordnung in Stunden umgerechnet wurde.) Konkret wird für den anschließenden Vergleich das CL-Modell mit 1, 5, 10, 50 und 100 HL konfiguriert, die jeweils 512 Neuronen der Eingabegröße 784 besitzen. Es entstehen folglich fünf Darstellungen, die mithilfe des nachfolgenden Codes jeweils ein Liniendiagramm ergeben. Die Abszisse ist in Zeile 1 und die Ordinaten in Zeile 2 und 3 definiert. Die Ordinate $y1$ ist hierbei als Liste mit Einträgen der Dateigröße in regelmäßigen Abständen x gekennzeichnet. Analog gilt dieses Verhalten anhand Zeile 6 der Ordinate $y2$, da sich beide die gleiche Abszisse teilen. Die verbleibenden Einstellungen bezüglich der Visualisierungen entsprechen dem eingangs definierten Programmcode. [84]

```
1  x = list(range(len(log_size)))
2  y1 = log_size_mb
3  y2 = runtime_min
4
5  fig, ax1 = plt.subplots(figsize = (10, 4))
6  ax2 = ax1.twinx()
7
8  ax1.plot(x, y, label = "Speichergröße")
9
10 ax1.set_title("Speichergröße der Logdatei vs. Trainingszyklus bei n Hidden Layern")
11 ax1.set_xlabel("Experience während des Trainingszyklus")
12 ax1.set_ylabel("Speichergröße der Logdatei [Bytes]")
13 ax1.set_ylabel("Laufzeit [min]")
14
15 fig.savefig("Bilder/SimpleMLP-nHL-log-size-experience.png")
```

3.4.2 Speichergröße der Weights-Logdatei vs. Anzahl der Neuronen

Entwicklungsumgebung Spyder Im Gegensatz zu den anderen beiden Visualisierungen in 3.4.1 und 3.4.3 wird Spyder als Entwicklungsumgebung verwendet. Der in Kapitel 3.3.5 beschriebene Programmcode zur Erstellung der Delta-Logdatei bleibt hierbei unberücksichtigt. Vor der eigentlichen Visualisierung wird die `buffer_size` der interaktiven Python Konsole unter `Preferences` → `IPython console` auf 100.000 erhöht. Die Konsole muss anschließend neu gestartet werden, um die Änderung der Konfiguration vollständig zu übernehmen. IPython lauft unter der Version 8.3.0 der 64-bit Variante von Python.

Visualisierung Bei einer konstanten Anzahl von Trainingszyklen wird die Speicherplatzgröße der Weights-Logdatei zu der variierenden Anzahl an Neuronen dargestellt. Die Speichergröße wird in MB oder GB auf zwei Nachkommastellen gerundet angegeben. Hierbei wird ebenfalls der Verlauf der Laufzeit des Modelltrainings in Minuten betrachtet. Konkret wird das Training eines SimpleMLP-Modells mit 1 und 5 HL jeweils für 512, 1024, 1536, 2048, 2560 und 3072 Neuronen eingestellt. Das Training der Modelle mit 10, 50 und 100 HL wird für 128, 256, 384, 512, 640 und 768 Neuronen durchgeführt. Dabei ist es wichtig, dass der Abstand zwischen der größer werdenden Neuronenzahl gleichbleibt. Das bedeutet bei einem und fünf HL eine Vergrößerung von jeweils 512 Neuronen sowie 128 Neuronen bei den verbleibenden Modellen. Anhand der fünf verschiedenen HL-Einstellungen resultieren fünf Liniendiagramme, jeweils eins pro Konfiguration. Bei der Erstellung der Visualisierung wird in Zeile 1 die `num_neurons` als Abszisse gewählt. Die Werte der zwei verschiedenen Ordinaten werden in Zeile 2 und 3 durch die Inhalte der Liste `neuron_log_size` und `runtime` dargestellt. Die Zeilen 5 und 6 verdeutlichen das Definieren von zwei Ordinaten mithilfe der Funktion `twinx()`, welche sich die gleiche Abszisse teilen. Die übrigen Programmzeilen entsprechen den zu Beginn charakterisierten Konfigurationen. [84]

```
1  x = num_neurons
2  y1 = neuron_log_size
3  y2 = runtime
4
5  fig, ax1 = plt.subplots(figsize=(10, 4))
6  ax2 = ax1.twinx()
7
8  ax1.plot(x, y1, "o-", color = "midnightblue", markersize = 4, label =
       "Speichergröße")
9  ax2.plot(x, y2, "o", color = "darkmagenta", markersize = 2, linestyle = "dotted",
       label = "Laufzeit t")
10
11 ax1.set_title("Speichergröße der Logdatei und Laufzeit vs. Neuronenzahl bei n
       Hidden Layern", fontfamily = "serif", fontsize = 11)
12 ax1.set_xlabel("Absolute Anzahl der Neuronen", fontfamily = "serif", fontsize = 10)
13 ax1.set_ylabel("Speichergröße der Logdatei [MB]", fontfamily = "serif", fontsize =
       10)
14 ax2.set_ylabel("Laufzeit [min]", fontfamily = "serif", fontsize = 10)
15
16 ax1.legend(loc = "upper left", prop = font)
17 ax2.legend(loc = "lower right", prop = font)
18
19 fig.savefig("Bilder/SimpleMLP-nHL-log-size-neurons.png")
```

3.4.3 Anteil an Nullen in der Delta-Logdatei vs. Trainingszyklus

Zuletzt wird die Abhängigkeit der Prozentzahl an in der Delta-Logdatei gespeicherten Nullen mit steigender Trainingsiteration betrachtet. Hierfür wird zunächst in Zeile 1 der Grundwert G bestimmt. Dies geschieht über die Multiplikation der Anzahl aller Elemente in `tensor_deltas` mit deren Länge, welche der Anzahl an vorkommenden HL entspricht. Anschließend werden die Prozentwerte des Anteils an Nullen in der Liste `percentage_deltas` (siehe Zeile 2) gespeichert. Hierbei müssen sowohl die Vorkommastelle als auch alle vier Nachkommastellen eine Null besitzen (Vergleich Zeile 6). Diese Werte dienen als Grundlage für die Erstellung der Liste W in Zeile 8, welche die kumulierte Anzahl an Nullen in `tensor_deltas` beinhaltet. Mithilfe der Formel 3.1 werden vergleichbar zu Zeile 9 die kumulierten Prozentsätze an Nullen berechnet und in der Liste p gespeichert.

$$p = \frac{W}{G} \tag{3.1}$$

```
1 G = tensor_deltas[0].size * len(tensor_deltas)
2 percentage_zeros = []
3 i = 0
4 while i in range(0, 512):
5     arr = np.array(df_deltas[i].round(4))
6     percentage_zeros.append(len(arr[arr == 0.0000]))
7     i+=1
8 W = np.cumsum(percentage_zeros)
9 p = W / G
```

Die zuvor mit Prozentwerten gefüllte Liste p wird als Ordinate in Zeile 2 übergeben. In Zeile 5 wird zusätzlich zu dem Funktionsgraphen eine Linie der Funktion $f(x) = 1$ im Wertebereich von $W_f = [0,512]$ definiert, welche den maximal möglichen Prozentsatz als Vergleichswert darstellt. [84]

```
1  x = list(range(len(p)))
2  y = p
3  y2 = p
4
5  ax2 = ax1.twinx()
6  ax1.plot(x, y1, label = "Kumulierter Prozentsatz")
7  ax2.plot(x, y2, color = "midnightblue", linestyle = "dotted", label = "Kumulierter
       Prozentsatz")
8  ax.hlines(y = 1.0, xmin = 0.0, xmax = 512.0, color = "g", linestyle = "dashed",
       label = "Grenze max. Prozentsatz: 1.0")
9  ax1.set_ylim(bottom = -0.05, top = 1.1)
10 ax2.set_ylim(bottom = -0.0001, top = 0.0021)
11
12 ax1.set_title("Prozentsatz an Nullen in der Delta-Logdatei bei n Hidden Layern")
13 ax1.set_xlabel("Trainingszeit t")
14 ax1.set_ylabel("Prozentsatz p")
15 plt.legend(loc = "center left")
16
17 fig.savefig("Bilder/SimpleMLP-nHL-percentage-zeros.png")
```

4 Ergebnisse

Die Ergebnisse der vorangegangenen Methoden werden in diesem Kapitel dargestellt. Hierbei werden bestehende XAI-Methoden sowie der Aufbau des CL-Modells präsentiert. Anhand dessen werden die zuvor beschriebenen Visualisierungen dargestellt und miteinander verglichen.

4.1 Vergleich ausgewählter Softwarebibliotheken der Erklärbaren Künstlichen Intelligenz

Um einen Überblick über bestehende XAI-Frameworks sowie deren Softwarebibliotheken zu erhalten, wurden vordefinierte Bewertungskriterien anhand der in Kapitel 3.1 beschriebenen Methoden in Tabellenform einander gegenübergestellt. Dieser Vergleich dient der Entscheidungshilfe bei der Auswahl eines passenden XAI-Verfahrens sowie deren Bibliothek bei verschiedenen Anwendungsszenarien. Es folgen die in Tabelle 4.1 dargestellten Ergebnisse des Vergleichs. [36, S. 36 f.]

Tabelle 4.1: Vergleich ausgewählter Softwarebibliotheken für XAI-Methoden (Stand 11.08.2022), aktualisiert und erweitert nach [36, S. 47]

Bibliothek	**ML-Modell**			**Trainingsdaten**	**Visualisierung**
	Scikit-Learn	TensorFlow	PyTorch		
AIX360	○	●	○	○	○
Alibi	○	●	○	○	○
Captum	–	–	●	●	●
iNNvestigate	–	●	–	○	●
InterpretML	○	○	○	●	●
LIME	●	●	●	○	○
SHAP	○	●	●	○	●
Skater	○	●	○	○	●
Tf-explain	–	●	–	●	●
Quantus	–	●	●	●	●
Zennit	–	–	●	●	●

Bibliothek	Dokumentation	Beispiele	Metriken	Softwareaktualität (GitHub)	
				Ltz. Release	Ltz. Commit
AIX360	● [9]	Tabelle, Bild, Text	●	28.10.2020 v0.2.1	26.07.2022
Alibi	● [10]	Tabelle, Bild, Text	●	18.05.2022 v.0.7.0	08.08.2022
Captum	● [11]	Tabelle, Bild, Text	●	04.03.2022 v0.5.0	10.08.2022
iNNvestigate	● [12]	Bild, Text	●	01.08.2022 v2.0.0	01.08.2022
InterpretML	● [13]	Tabelle	○	23.09.2021 v0.2.7	09.08.2022
LIME	● [14]	Tabelle, Bild, Text	–	04.04.2020 v0.2.0.0	30.07.2021
SHAP	● [15]	Tabelle, Bild, Text	○	16.06.2022 v0.41.0	16.06.2022
Skater	● [16]	Tabelle, Bild, Text	–	21.09.2018 v1.1.2	11.02.2022
Tf-explain	○ [17]	Bild	–	18.11.2021 v0.3.1	30.06.2022
Quantus	–	Bild	●	10.08.2022 v0.1.5	10.08.2022
Zennit	● [18]	Bild	○	20.07.2022 v0.5.0	20.07.2022

Legende: ● Bedingung erfüllt
○ Bedingung teilweise / nicht für alle enthaltenen Methoden erfüllt
– Bedingung nicht erfüllt

[9] `https://aix360.readthedocs.io/en/latest/`
[10] `https://docs.seldon.io/projects/alibi/en/latest/`
[11] `https://captum.ai/`
[12] `https://innvestigate.readthedocs.io/en/latest/`
[13] `https://interpret.ml/docs/intro.html`

ML-Modelle Bei der Betrachtung der Modelle fällt auf, dass der Großteil der XAI-Bibliotheken hauptsächlich eine ML-Bibliothek unterstützt. Die Ausnahme stellt InterpretML dar, die für alle drei ML-Bibliotheken eine Unterstützung, jedoch mit Einschränkungen, anbietet. LIME ist die einzige Bibliothek, die mit allen drei Arten der ML-Bibliotheken vollständig kompatibel ist. SHAP und Quantus folgen mit zwei kompatiblen ML-Modellen (TensorFlow und PyTorch). Bis auf die Bibliotheken Captum und Zennit unterstützen alle anderen Frameworks das ML-Modell TensorFlow, wobei InterpretML eine eingeschränkte Unterstützung anbietet und iNNvestigate vorrangig die mit TensorFlow kompatible Deep-Learning-Bibliothek Keras verwendet. Scikit-Learn wird einzig von LIME vollständig unterstützt und ist nur teilweise kompatibel mit InterpretML, AIX360, Skater, Alibi und SHAP, wodurch eine geringere Unterstützung im Vergleich zu TensorFlow und PyTorch stattfindet. PyTorch-Modelle finden gänzlich Anwendung in Captum, SHAP, LIME, Quantus und Zennit. Sie werden jedoch von iNNvestigate und Tf-explain nicht gestützt.

Trainingsdaten Die Trainingsdaten werden bei allen XAI-Bibliotheken teilweise oder vollständig für die Erklärung von ML-Modellen herangezogen. Eine vollständige Verwendung ist bei InterpretML, Tf-explain, Captum, Quantus und Zennit zu verzeichnen.

Visualisierung Alle XAI-Bibliotheken bieten ebenso wie bei den Trainingsdaten eine teilweise bis vollständige Visualisierung der Ergebnisse an. Eine eingeschränkte Bereitstellung ist bei den Bibliotheken AIX360, Alibi und LIME vorhanden.

Dokumentation Das Framework Quantus weist keine hinreichende Dokumentation seiner Funktionalitäten auf. Weiterhin ist die Funktionsweise von Tf-explain nur teilweise dokumentiert. Die übrigen XAI-Bibliotheken zeichnen sich durch eine vollständige Dokumentation auf.

Beispiele Bei der Betrachtung der Beispiele fällt auf, dass bis auf InterpretML alle XAI-Bibliotheken Bild-gestützte Modelle verwenden können. ML-Modelle auf Grundlage von Tabellen- oder Textdaten, werden von jeweils sieben Bibliotheken unterstützt. Die Bibliotheken AIX360, Skater, Alibi, Captum, SHAP, und LIME bieten eine umfängliche Unterstützung aller drei Anwendungen an.

Metriken Die Mehrheit der Bibliotheken bietet Metriken zur Evaluation der Erklärungsfunktionalität an. Ausgenommen davon sind Skater, Tf-explain und LIME. Im Gegensatz dazu ist bei den Bibliotheken AIX360, Alibi, iNNvestigate, Captum und Quantus eine vollständige Unterstützung vorzufinden.

Softwareaktualität Bezogen auf die Softwareaktualität ist das letzte Release-Update bei der Hälfte der XAI-Bibliotheken weniger als ein Jahr her. Am ältesten ist der letzte Release bei Skater (09/2018), gefolgt von LIME (04/2020) und AIX360 (10/2020). Auffällig ist weiterhin, dass der letzte Commit der meisten XAI-Bibliotheken im Jahr 2022 war. Lediglich bei LIME wurde zuletzt im Juli 2021 der letzte Commit veröffentlicht.

[14] `https://lime-ml.readthedocs.io/en/latest/`
[15] `https://shap.readthedocs.io/en/latest/`
[16] `https://oracle.github.io/Skater/index.html`
[17] `https://tf-explain.readthedocs.io/en/latest/`
[18] `https://zennit.readthedocs.io/en/latest/`

4.2 Aufbau und Konfiguration des Continual Learning Modells unter Avalanche

SplitMNIST Abbildung 4.1 verdeutlicht beispielhaft die Datengrundlage für das Modelltraining. Sichtbar sind sechs Bilder des MNIST-Datensatzes, die eine handschriftlich verfasste weiße Ziffer auf schwarzfarbenem Hintergrund beinhalten. Am linken und unteren Bildrand ist weiterhin die Pixelgröße jedes einzelnen Bildes von $28 * 28$ Pixeln zu sehen. Daraus ergibt sich eine Gesamtgröße von insgesamt 784 Pixeln pro Eingabebild.

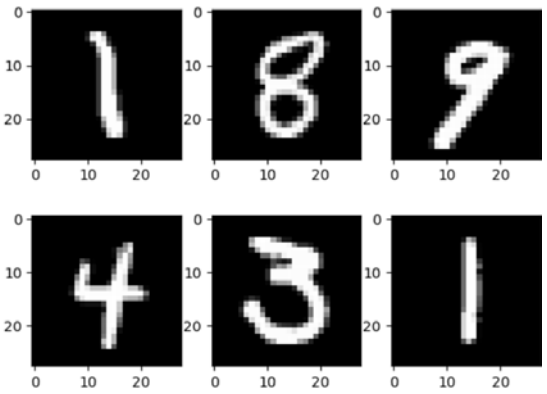


Abbildung 4.1: Die sechs Bilder des SplitMNIST Datensatzes dienen als Anschauung der Eingabedaten.

Da ein kontinuierlich lernendes System betrachtet wird, kann nicht die gesamte Datengrundlage klassifiziert werden, sondern nur ausgewählte Klassen. Im Vergleich zu Abbildung 4.2 ist erkennbar, dass die fünf Klassen $[\{2,6\},\{8,1\},\{4,5\},\{0,9\},\{3,7\}]$ in jeder Experience hinsichtlich der fünf Label $[[0],[1],[2],[3],[4]]$ angelernt werden. Dabei wird deutlich, dass nur auf die Hälfte aller möglichen Label getestet wird und nur fünf Klassen von insgesamt $V_{wk}^{n} = n^k = 10^2 = 100$ Kombinationen betrachtet werden.

```
Original Datensätze:
<avalanche.benchmarks.utils.avalanche_dataset.AvalancheSubset object at 0x000001E00CC2BAC0>
<avalanche.benchmarks.utils.avalanche_dataset.AvalancheSubset object at 0x000001E00CC2BE50>
================================================================================================================
Task Labels:
[[0], [1], [2], [3], [4]]
================================================================================================================
Trainings- und Testdatenstrom:
<avalanche.benchmarks.scenarios.classification_scenario.ClassificationStream object at 0x000001E00CC0FC70>
Länge des Trainingsdatenstroms: 5
<avalanche.benchmarks.scenarios.classification_scenario.ClassificationStream object at 0x000001E0176B9040>
Länge des Testdatenstroms: 5
================================================================================================================
Klassen in jeder Experience:
[{2, 6}, {8, 1}, {4, 5}, {0, 9}, {3, 7}]
```

Abbildung 4.2: Die Konsolenausgabe zeigt Informationen zum Trainings- und Testdatensatz, zu den Task Labeln und zu den verwendeten Klassen je Trainings-Experience.

SimpleMLP Die Ausgabe des Befehls `print(model)` in Abbildung 4.3(a) verdeutlicht die Unterteilung des CL-Modells `SimpleMLP` in `features` und `classifier`. Der Layer `features` ist hierbei aufgebaut aus den sequenziellen Modulen `nn.Linear`, `nn.ReLU` und `nn.Dropout`. Die Torch-Module `nn.ReLU` und `nn.Dropout` des Layers `nn.Sequential` werden innerhalb des Trainingsprozesses nicht durch das Hinzufügen von Lerngewichten beeinflusst. Im Gegensatz dazu lassen sich die Gewichte der Module `nn.Linear` des sequenziellen Containers sowie des `Classifier` Layers anzeigen und speichern. Neben den Gewichten gehören zu den PyTorch Parametern auch die Bias der Layer. Diese werden ebenfalls gespeichert und beeinflussen die Optimierung innerhalb des Trainingsprozesses. Wird die rechte Abbildung 4.3(b) betrachtet, sind die Weight und Bias Parameter mit der

zugehörigen Tensorgröße zu sehen. Hierbei ist erkennbar, dass sich der Parametername aus dem Layernamen, entweder `features` oder `classifier`, der Nummer des Layers sowie der Kennzeichnung des Modellgewichts (`weights`) oder des Bias (`bias`) zusammensetzt.

```
SimpleMLP(
  (features): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
  )
  (classifier): Linear(in_features=512, out_features=10, bias=True)
)
```

(a) Aufbau des CL-Modells SimpleMPL

```
name:  features.0.weight
<class 'torch.nn.parameter.Parameter'>
param.shape:  torch.Size([512, 784])
param.requires_grad:  True
==========
name:  features.0.bias
<class 'torch.nn.parameter.Parameter'>
param.shape:  torch.Size([512])
param.requires_grad:  True
==========
name:  classifier.weight
<class 'torch.nn.parameter.Parameter'>
param.shape:  torch.Size([10, 512])
param.requires_grad:  True
==========
name:  classifier.bias
<class 'torch.nn.parameter.Parameter'>
param.shape:  torch.Size([10])
param.requires_grad:  True
```

(b) Parameter von SimpleMLP

Abbildung 4.3: Die linke Darstellung 4.3(a) kennzeichnet die Konsolenausgabe des `print`-Befehls. Hierbei ist die interne Struktur des Modells SimpleMLP mit einem HL erkennbar. Auf der rechten Seite 4.3(b) sind die Parameter von SimpleMLP aufgeschlüsselt. Die grün umrandeten Parameter kennzeichnen die wichtigen Modellgewichte.

Der folgende Tensor des Layers `Linear(Sequential)` in Abbildung 4.4 verdeutlicht dessen interne Struktur. Erkennbar ist, dass Tensoren durch eckige Klammern voneinander abgegrenzt werden. Da mehrere dieser Klammern vorhanden sind, besteht der Parameter aus mehreren ineinander verschachtelten Tensoren. Dies wird auch durch den Rang 2 deutlich. Er besagt, dass die Ausprägung zweidimensional verläuft. Aufgrund dessen wird er auch „Weight Matrix“, dt. Gewichtsmatrix, genannt. Innerhalb der Klammern befinden sich die gerundeten Tensorwerte, welche durch Kommas voneinander separiert werden. [85]

```
tensor([[ 0.0041,  0.0165, -0.0057,  ...,  0.0223, -0.0335, -0.0205],
        [-0.0067, -0.0249,  0.0341,  ..., -0.0005, -0.0060, -0.0181],
        [-0.0046, -0.0302,  0.0083,  ...,  0.0015, -0.0077,  0.0324],
        ...,
        [-0.0130, -0.0029, -0.0233,  ..., -0.0167, -0.0342,  0.0044],
        [ 0.0046,  0.0303, -0.0073,  ...,  0.0324,  0.0012,  0.0016],
        [ 0.0094, -0.0243,  0.0124,  ...,  0.0284,  0.0360,  0.0163]])
Dimension:  2
Shape:  torch.Size([512, 784])
```

Abbildung 4.4: Die zweidimensionale Gewichtsmatrix des Layers `Linear(Sequential)` besitzt eine Größe (`Shape`) von $[512, 784]$. Die Darstellung des Tensors ist charakterisiert durch die verkürzte Ausgabe des `print`-Befehls und ist somit nicht vollständig.

Training Der Ausschnitt der Konsolenausgabe in Abbildung 4.5 zu Beginn des Experiments verdeutlicht den aktuellen Stand des Modelltrainings mithilfe von Fortschrittsbalken des `InteractiveLogger` nach jeder der fünf Epochen. Analog zu der Ausgabe der Experience 0 verhält sich die Ausgabe der folgenden Experiences. Nach jeder Experience wird die aktuelle Größe der Logdatei ausgegeben und die Genauigkeit (engl. *Accuracy*) für den betrachteten Testdatensatz berechnet.

```
Experiment beginnen...
Beginn der Experience:  0
Aktuelle Klasse der Experience:  [0, 1]
-- >> Beginn der Trainingsphase << --
-- >> Beginn der Evaluationsphase << --
-- >> Starte Evaluation der Experience 0 (Task 0) des train Datenstroms << --
100%|████████████████████████████████████████████████████████████| 119/119 [00:09<00:00, 12.73it/s]
-- >> Evaluation der Experience 0 (Task 0) des train Datenstroms endet. << --
        Top1_Acc_Exp/eval_phase/train_stream/Task000/Exp000 = 0.0093
-- >> Ende der Evaluationsphase << --
100%|████████████████████████████████████████████████████████████| 119/119 [00:09<00:00, 11.96it/s]
Epoche 0 endet.
100%|████████████████████████████████████████████████████████████| 119/119 [00:09<00:00, 12.19it/s]
Epoche 1 endet.
100%|████████████████████████████████████████████████████████████| 119/119 [00:09<00:00, 12.27it/s]
Epoche 2 endet.
100%|████████████████████████████████████████████████████████████| 119/119 [00:09<00:00, 12.33it/s]
Epoche 3 endet.
100%|████████████████████████████████████████████████████████████| 119/119 [00:09<00:00, 12.68it/s]
Epoche 4 endet.
-- >> Beginn der Evaluationsphase << --
-- >> Starte Evaluation der Experience 0 (Task 0) des train Datenstroms << --
100%|████████████████████████████████████████████████████████████| 119/119 [00:09<00:00, 13.08it/s]
-- >> Evaluation der Experience 0 (Task 0) des train Datenstroms endet. << --
        Top1_Acc_Exp/eval_phase/train_stream/Task000/Exp000 = 0.9859
-- >> Ende der Evaluationsphase << --
-- >> Ende der Trainingsphase << --
Training abgeschlossen
Die Dateigröße nach der Experience 0 beträgt: 24399719 Bytes.
Berechnung der Genauigkeit für die gesamte Testmenge
-- >> Beginn der Evaluationsphase << --
-- >> Starte Evaluation der Experience 0 (Task 0) des test Datenstroms << --
100%|████████████████████████████████████████████████████████████| 20/20 [00:01<00:00, 14.36it/s]
-- >> Evaluation der Experience 0 (Task 0) des test Datenstroms endet. << --
        Top1_Acc_Exp/eval_phase/test_stream/Task000/Exp000 = 0.9854
-- >> Starte Evaluation der Experience 1 (Task 1) des test Datenstroms << --
100%|████████████████████████████████████████████████████████████| 22/22 [00:01<00:00, 12.65it/s]
-- >> Evaluation der Experience 1 (Task 1) des test Datenstroms endet. << --
        Top1_Acc_Exp/eval_phase/test_stream/Task001/Exp001 = 0.6064
-- >> Starte Evaluation der Experience 2 (Task 2) des test Datenstroms << --
100%|████████████████████████████████████████████████████████████| 19/19 [00:01<00:00, 10.79it/s]
-- >> Evaluation der Experience 2 (Task 2) des test Datenstroms endet. << --
        Top1_Acc_Exp/eval_phase/test_stream/Task002/Exp002 = 0.3783
-- >> Starte Evaluation der Experience 3 (Task 3) des test Datenstroms << --
100%|████████████████████████████████████████████████████████████| 20/20 [00:01<00:00, 12.89it/s]
-- >> Evaluation der Experience 3 (Task 3) des test Datenstroms endet. << --
        Top1_Acc_Exp/eval_phase/test_stream/Task003/Exp003 = 0.4319
-- >> Starte Evaluation der Experience 4 (Task 4) des test Datenstroms << --
100%|████████████████████████████████████████████████████████████| 21/21 [00:01<00:00, 13.27it/s]
-- >> Evaluation der Experience 4 (Task 4) des test Datenstroms endet. << --
        Top1_Acc_Exp/eval_phase/test_stream/Task004/Exp004 = 0.5402
-- >> Ende der Evaluationsphase << --
```

Abbildung 4.5: Der Ausschnitt zeigt den Beginn der Konsolenausgabe während des Modelltrainings. Konkret ist der Trainingszyklus von Epoche 0 bis 4 für die Experience 0 dargestellt. Neben der aktuellen Experience und Klasse wird die Dateigröße ausgegeben.

4.3 Interne Logging Funktionalität des Continual Learning Modells

Die Ergebnisse der Logging Funktionalität innerhalb des CL-Modells werden im Folgenden durch die Logging-Module von Avalanche, der Weights-Logdatei, der Information-Logdatei und der Delta-Logdatei verdeutlicht. Zusätzlich wird ein beispielhafter Verlauf der Gewichtsänderung gezeigt.

4.3.1 Logging-Module von Avalanche

TextLogger Die Ergebnisse des `TextLogger` sind im nachfolgenden Abschnitt 4.3.3 einsehbar.

InteractiveLogger Die Funktionalität des `InteractiveLogger` ist ab Kapitel 4.2 als Konsolenausgaben zu sehen. Insbesondere wurde die Ausgabe während des Modelltrainings hierbei von den Konfigurationen des Loggers beeinflusst.

TensorboardLogger In Anhang B wird die graphische Darstellung des CL-Modells von TensorBoard dargestellt. Konkret ist die Ansicht des `Graph Exporer` zu sehen, der die interne Modellstruktur vom `input` zum `output` verdeutlicht und gleichzeitig die Gewichte innerhalb des Modells SimpleMLP lokalisiert. Hierbei ist ebenfalls die Anordnung der Layer `Sequential[features]` und `Linear[classifier]` sowie deren Komponenten ersichtlich. Vergleichbar zu W&B wird der Verlauf der Metrikwerte in Echtzeit während des Modelltrainings graphisch dokumentiert.

WandbLogger Die Benutzungsoberfläche von W&B ist in Abbildung 4.6 dargestellt. Konkret ist der Run `SimpleMLP-5HL` des Projekts `avalanche-cl` erkennbar. Hierbei sind beispielhaft vier Histogramme dargestellt, die die Graphen des Gewichtsverlaufs im oberen Bildbereich sowie die Gradientenverläufe im unteren Bildbereich nach dem Modelltraining zeigen. Die Histogramme sind sowohl bei dem W&B-Dashboard als auch bei der Webseite von W&B einsehbar. Zusätzlich werden ebenfalls die Metrikwerte während des Trainings- und Testdatenstroms kontinuierlich gespeichert und visualisiert, sodass Änderungen direkt innerhalb des Jupyter Notebooks oder alternativ auf der Webseite in Echtzeit eingesehen werden können.

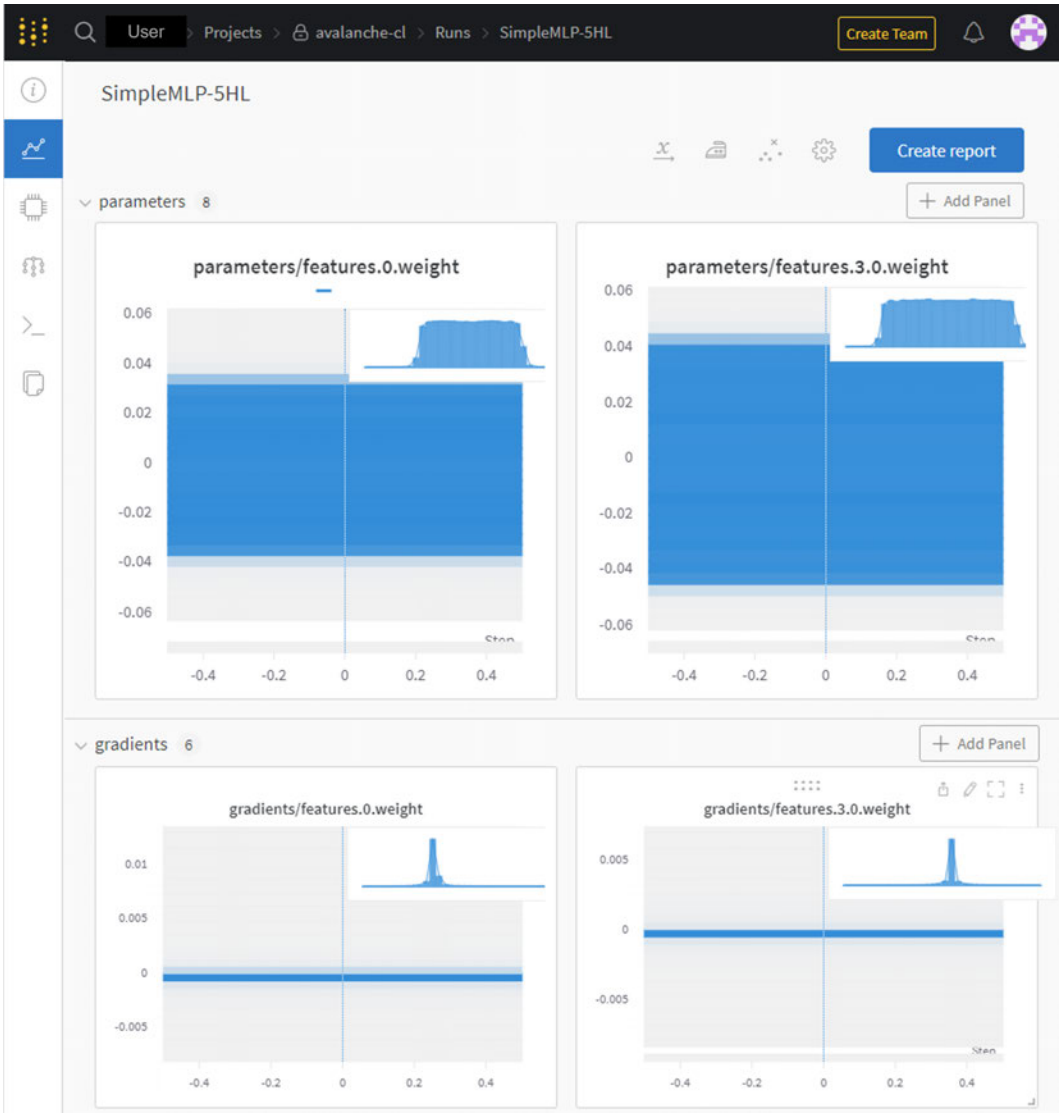


Abbildung 4.6: Dargestellt sind die Funktionsverläufe der Gewicht-Parameters und der Gradienten der Gewichte `features.0.weight` und `features.3.0.weight` nach dem Ende der Trainingsphase des aktuellen Runs von SimpleMLP-5HL in der Benutzungsoberfläche `Charts` von W&B.

Im Rahmen der Beschreibung der Kurvenverläufe in Abbildung 4.6 fällt auf, dass sich die Modellgewichte insgesamt in einem Bereich von -0.04 bis 0.04 befinden. Aufgrund der hohen Dichte an Gewichtswerten erscheint das Diagramm als blauer Balken. Es kann jedoch der grobe Verlauf in der oberen Mitte angezeigt werden, indem der Mauszeiger über die Werte des Balkens bewegt wird. Hierbei ist ein annähernd symmetrischer Verlauf erkennbar, der zunächst keine Steigung aufweist, anschließend stark ansteigt, konstant auf einem Level verbleibt und letztendlich wieder stark auf das Ausgangsniveau abfällt. Auf diesem Level verbleibt der Graph bis zum letzten Step.
Der Funktionsgraph der Gradienten weist im mittleren Bereich einen kurzen und starken Anstieg auf, welcher den Peak des Graphen darstellt. Anschließend sinkt der Graph auf das Ausgangsniveau zurück, wo er bis zum letzten Step verbleibt. Die Gradienten schwanken im Bereich von -0.0005 bis 0.0005. Zusammenfassend kann dokumentiert werden, dass sich sowohl die beiden Verläufe der Gewichtswerte als auch die Verläufe der Gradienten von `features.0.weight` und `features.3.0.weight` untereinander gleichen.
Neben den Panel Typen `parameters` und `gradients` werden ebenfalls Liniendiagramme zum Systemverhalten (`System`) und Histogramme zu dem HL-Verlauf (`Hidden Panels`) visualisiert. Die Benutzungsoberfläche `models` in Abbildung B.2 zeigt außerdem interne Modellbestandteile.

Nach der Trainingsphase wird die W&B Sitzung beendet. Es resultiert die Ausgabe der gespeicherten *Run history* und *Run summary* Informationen in Abbildung 4.7, welche mit den Daten der Webseite von W&B synchronisiert wurden. Bei einer Gesamtzahl von vier `TrainingExperience` beträgt die Genauigkeit des Modelltrainings nach dem letzten Evaluationsschritt des Testdatensatzes 0.96467.

Run history:

Top1_Acc_Exp/eval_phase/test_stream/Task000/Exp000	
Top1_Acc_Exp/eval_phase/test_stream/Task001/Exp001	
Top1_Acc_Exp/eval_phase/test_stream/Task002/Exp002	
Top1_Acc_Exp/eval_phase/test_stream/Task003/Exp003	
Top1_Acc_Exp/eval_phase/test_stream/Task004/Exp004	
Top1_Acc_Exp/eval_phase/train_stream/Task000/Exp000	
Top1_Acc_Exp/eval_phase/train_stream/Task001/Exp001	
Top1_Acc_Exp/eval_phase/train_stream/Task002/Exp002	
Top1_Acc_Exp/eval_phase/train_stream/Task003/Exp003	
Top1_Acc_Exp/eval_phase/train_stream/Task004/Exp004	
TrainingExperience	

Run summary:

Top1_Acc_Exp/eval_phase/test_stream/Task000/Exp000	0.90452
Top1_Acc_Exp/eval_phase/test_stream/Task001/Exp001	0.82314
Top1_Acc_Exp/eval_phase/test_stream/Task002/Exp002	0.56884
Top1_Acc_Exp/eval_phase/test_stream/Task003/Exp003	0.93012
Top1_Acc_Exp/eval_phase/test_stream/Task004/Exp004	0.96467
Top1_Acc_Exp/eval_phase/train_stream/Task000/Exp000	0.98594
Top1_Acc_Exp/eval_phase/train_stream/Task001/Exp001	0.97499
Top1_Acc_Exp/eval_phase/train_stream/Task002/Exp002	0.98109
Top1_Acc_Exp/eval_phase/train_stream/Task003/Exp003	0.98172
Top1_Acc_Exp/eval_phase/train_stream/Task004/Exp004	0.97644
TrainingExperience	4

Abbildung 4.7: Die gespeicherten W&B-Informationen zu dem Trainingsverlauf (*Run history*) werden der Zusammenfassung (*Run summary*) des aktuellen Durchlaufs von SimpleMLP-1HL nach dem Modelltraining gegenübergestellt.

4.3.2 Verlauf der Gewichtsänderung

Die Tabelle 4.2 zeigt den Verlauf der Gewichte und deren Änderung ab dem Beginn der Trainingsphase (Experience 0) bis Ende der letzten Training Experience (4).
Die Änderung der Werte des Layers `features[0]` ist in der Spalte zwei der Tabelle 4.2 ersichtlicht. Hierbei ist zu Beginn des Modelltrainings der Gewichtswert von 0.0048 zu verzeichnen. Dieser ändert sich während des Trainings zu 0.0032. Insgesamt ergibt sich daraus eine Änderung der Delta-Werte innerhalb des Layers von $0.0048 - 0.0032 = 0.0016$. Wird im Vergleich dazu der Verlauf des Layer-Wertes von `classifier` in Spalte vier betrachtet, beginnt dieser mit 0.441 und endet nach der letzten Training Experience bei 0.0518. Daraus errechnet sich eine layerinterne Änderung von $|0.441 - 0.0518| = 0.0107$. Wird der Deltaverlauf innerhalb der Layer betrachtet (Vergleich Spalte 3 und 5), fällt auf, dass die Änderungen im Vergleich zum Start der Trainingsschleife mit jeder Experience tendenziell kleiner werden.
Das Ziel der Betrachtung des Gewichtsverlaufs ist die Berechnung der Delta-Werte zwischen benachbarten Layern, Vergleich Spalte sechs. Für den dargestellten Layerwert ergibt sich am Ende des Modelltrainings somit ein Deltawert von $|0.0032 - 0.0518| = 0.0486$, welcher an erster Stelle in der Datei `delta_weights_1hl.txt` gespeichert ist, siehe Abbildung 4.14. Bei der Betrachtung der Änderung der Delta-Werte in Spalte 7 fällt ebenfalls eine Verringerung der Werte von der ersten ($|0.0393 - 0.0584| = 0.0191$) zur letzten Änderung (0.0057) auf.

Tabelle 4.2: Die Modellgewichte und deren Änderung während des Trainings werden beispielhaft für das Modell SimpleMLP mit einem HL betrachtet. Hierbei sind die eigentlichen Werte pro Layer nach jedem Trainingsschritt schwarz und die Änderung zwischen zwei Trainingsschritten und zwischen zwei Delta-Werten grau dargestellt. Der resultierende Deltawert nach dem letzten Trainingsschritt ist aufgrund seiner Relevanz grün umrandet.

Änderungszeitpunkt	**Layerwert** `features[0]`		**Layerwert** `classifier`		$\lvert\Delta\rvert$	
Beginn des Trainings	0.0048		0.0441		0.0393	
Training Experience 0	0.0041	0.0007	0.0625	0.0184	0.0584	0.0191
Training Experience 1	0.0036	0.0005	0.0616	0.0009	0.0580	0.0004
Training Experience 2	0.0035	0.0001	0.0619	0.0003	0.0584	0.0004
Training Experience 3	0.0033	0.0002	0.0576	0.0043	0.0543	0.0041
Training Experience 4	0.0032	0.0001	0.0518	0.0058	0.0486	0.0057

4.3.3 Weights-Logdatei während des Modelltrainings

Anhand der in Kapitel 3.3.3 beschriebenen Vorgehensweise zur Umsetzung der Logging-Funktionalität werden die folgenden Ergebnisse der Protokollierung von Modellgewichten beschrieben. Konkret sind sie in dem Ausschnitt der Datei `weights_log_5hl.txt` während der Trainingsphase in Abbildung 4.14 einsehbar. Bei dem Verlauf der geloggten Daten ist im vorliegenden Beispiel der Beginn der Trainingsphase mit den ersten beiden Gewichten des HL `model.features` der Größen $[512, 784]$ und $[512, 512]$ vor der Trainingsphase erkennbar. Die Tensoren der Modellgewichte sind hierbei in der verkürzten Schreibweise dargestellt, um die Inhalte des Protokolls besser zu verdeutlichen.

weights_log_5hl.txt

```
1  TRAININGSSCHLEIFE BEGINNT
2  -- >> Beginn der Trainingsphase << --
3  Gewichtswerte des Layers model.features[0] (torch.Size([512, 784])) vor der Trainingsphase:
4  tensor([[ 0.0048,  0.0172, -0.0050,  ...,  0.0230, -0.0328, -0.0198],
5          [-0.0060, -0.0241,  0.0349,  ...,  0.0003, -0.0052, -0.0173],
6          [-0.0047, -0.0304,  0.0081,  ...,  0.0014, -0.0078,  0.0322],
7          ...,
8          [-0.0127, -0.0027, -0.0231,  ..., -0.0165, -0.0339,  0.0046],
9          [ 0.0042,  0.0300, -0.0076,  ...,  0.0320,  0.0009,  0.0013],
10         [ 0.0091, -0.0247,  0.0121,  ...,  0.0280,  0.0356,  0.0159]])
11 Gewichtswerte des Layers model.features[3][0] (torch.Size([512, 512])) vor der Trainingsphase:
12 tensor([[ 0.0441, -0.0045,  0.0086,  ..., -0.0017, -0.0240,  0.0139],
13         [-0.0231, -0.0351, -0.0090,  ...,  0.0321, -0.0412, -0.0183],
14         [-0.0202, -0.0398,  0.0434,  ...,  0.0002,  0.0334,  0.0195],
15         ...,
16         [ 0.0325, -0.0059,  0.0397,  ...,  0.0375, -0.0110, -0.0297],
17         [ 0.0234, -0.0391,  0.0301,  ...,  0.0266,  0.0089, -0.0123],
18         [-0.0171,  0.0229, -0.0188,  ...,  0.0316,  0.0161, -0.0282]])
```

Abbildung 4.8: Der Auszug der Weights-Logdatei `weights_log_5hl.txt` verdeutlicht den Protokollverlauf der Modellgewichte des ersten HLs während des Modelltrainings mit verkürzten Tensoren.

4.3.4 Information Logging

Zusätzlich wurde die Datei `info_weights_log_nhl` während des Modelltrainings erstellt, die Informationen bezüglich der Weights-Logdatei beinhaltet. Ein Ausschnitt des Inhalts dieser Information-Logdatei ist in Anhang C.1 in Abbildung C.1 dargestellt. Hierbei liegt der Fokus auf dem Verlauf des Modelltrainings während der ersten Experience (0), siehe Zeile 2. Insbesondere werden der Beginn (Zeile 4) und das Ende der Trainingsphase für die Layer `model.features[0]` und `model.classifier` dokumentiert, Vergleich Zeilen 4 und 5. Gleichermaßen gilt dieses Dokumentationsverhalten für die Evaluationsphase (Zeile 12). Weiterhin ist die Größe und der Hashwert der Logdatei in regelmäßigen Abständen abgebildet, zum Beispiel in den Zeilen 54 und 55. Konkret ist hier eine Dateigröße von 24400889 Bytes und ein Hashwert des Typs `SHA-256` gespeichert. Vor dem Ende der ersten Experience sind ebenfalls die Metrik-Werte der Accuracy in Zeile 56 ersichtlich.

In der Abbildung C.2 werden beispielhaft die ersten 16 Hashwerte der insgesamt 80 Einträge der Liste `hash_values` gezeigt. Pro Experience wurden jeweils 16 Werte gespeichert ($5 * 16 = 80$). Auffällig hierbei ist, dass für jede Trainings- oder Evaluationsphase zwei Hashwerte gespeichert werden. Bei dem Beginn und dem Ende der Trainingsphase ändert sich dieser Hashwert im Vergleich zu dem vorherigen. Wird jedoch der Beginn sowie das Ende der Evaluationsphase betrachtet, findet innerhalb der jeweiligen Phase keine Änderung zum vorangegangenen Hashwert statt. Zwischen dem Beginn und dem Ende der Evaluation ist aber eine Änderung des SHA-256-Hashs erkennbar.

4.3.5 Delta-Logdatei nach dem Modelltraining

Bei der Erstellung der Delta-Logdatei `delta_weights.txt` wurden diverse Konsolenausgaben erstellt, die sowohl aktuelle Zustände als auch den internen Fortschritt des Programmcodes dokumentieren. Die Screenshots dieser Ausgaben werden im Folgenden beschrieben. Weiterhin wird abschließend ein Ausschnitt des Endergebnisses präsentiert.

Zum Überprüfen der Formen werden zunächst die Shapes der Gewichtsparameter betrachtet. Die Ausgabe des Programmcodes ist in Abbildung 4.9 dargestellt. Hierbei ist erkennbar, dass in zwei Fällen die Shapes benachbarter Tensoren nicht übereinstimmen. Konkret besteht ein Unterschied zwischen den Gewichten `features.0.weight` der Größe $[512, 784]$ und `features.3.0.weight` der Größe $[512, 512]$ von $[0, 272]$ sowie zwischen den Gewichten `features.6.0.weight` der Größe $[512, 512]$ und `classifier.weight` der Größe $[10, 512]$ von $[502, 0]$.

```
Größe der Parameter Tensoren (weights) in 'state_dict':
features.0.weight        torch.Size([512, 784])
features.3.0.weight      torch.Size([512, 512])
features.4.0.weight      torch.Size([512, 512])
features.5.0.weight      torch.Size([512, 512])
features.6.0.weight      torch.Size([512, 512])
classifier.weight        torch.Size([10, 512])
```

Abbildung 4.9: Sowohl die Gewichtsnamen als auch die zugehörigen Shapes der Tensorgewichte des `state_dict` Dictionary werden in der Konsolenausgabe angezeigt. Dargestellt sind die Gewichte des SimpleMLP-Modells mit fünf HL.

In Abbildung 4.10 werden die Konsolenausgaben während der Ausführung der `for`-Schleife aus Kapitel 3.3.5 dargestellt. Erkennbar ist, dass zu Beginn des Aufrufs der Delta-Schleife keine Übereinstimmung der Formen zwischen zwei aufeinanderfolgenden Gewichten vorliegt. Beispielsweise besteht in Zeile 1 zwischen den Gewichten `weights_0` und `weights_1` eine Diskrepanz bezüglich ihrer Shapes ($[512, 784]$ und $[512, 512]$). Anhand der `print`-Ausgabe wird der kleinere Gewichtstensor mit Nullen aufgefüllt. Es schließt sich ein erneuter Vergleich der Shapes an, der die Übereinstimmung benachbarter Gewichte nach dem Padding bestätigt. Diese Ausgaben werden analog für alle Nachbarschaften der Modellgewichte angezeigt.

```
keine Übereinstimmung der Formen von 'weights_0' und 'weights_1':  torch.Size([512, 784]) und torch.Size([512, 512])
Shape des Gewichts-Tensors ist torch.Size([512, 512]):  True -> Auffüllen mit Nullen
Übereinstimmung der Formen von 'weights_0' und 'weights_1':  torch.Size([512, 784]) und torch.Size([512, 784])
============================
keine Übereinstimmung der Formen von 'weights_1' und 'weights_2':  torch.Size([512, 784]) und torch.Size([512, 512])
Shape des Gewichts-Tensors ist torch.Size([512, 512]):  True -> Auffüllen mit Nullen
Übereinstimmung der Formen von 'weights_1' und 'weights_2':  torch.Size([512, 784]) und torch.Size([512, 784])
============================
keine Übereinstimmung der Formen von 'weights_2' und 'weights_3':  torch.Size([512, 784]) und torch.Size([512, 512])
Shape des Gewichts-Tensors ist torch.Size([512, 512]):  True -> Auffüllen mit Nullen
Übereinstimmung der Formen von 'weights_2' und 'weights_3':  torch.Size([512, 784]) und torch.Size([512, 784])
============================
keine Übereinstimmung der Formen von 'weights_3' und 'weights_4':  torch.Size([512, 784]) und torch.Size([512, 512])
Shape des Gewichts-Tensors ist torch.Size([512, 512]):  True -> Auffüllen mit Nullen
Übereinstimmung der Formen von 'weights_3' und 'weights_4':  torch.Size([512, 784]) und torch.Size([512, 784])
============================
keine Übereinstimmung der Formen von 'weights_4' und 'weights_5':  torch.Size([512, 784]) und torch.Size([10, 512])
Shape des Gewichts-Tensors ist torch.Size([10, 512]):  True -> Auffüllen mit Nullen
Übereinstimmung der Formen von 'weights_4' und 'weights_5':  torch.Size([512, 784]) und torch.Size([512, 784])
```

Abbildung 4.10: In dem Screenshot der Konsolenausgabe wird getestet, ob die Übereinstimmung der Parameter Shapes von den Tensorgewichten vor und nach dem Padding vorliegt.

Die Konsolenausgabe während der Ausführung der Delta-Schleife aus Kapitel 3.3.5 wird in Abbildung 4.11 verdeutlicht. Zunächst wird die Übereinstimmung der Formen benachbarter Modellgewichte bestätigt sowie deren Größe von $[512, 784]$ ausgegeben. Es folgt die Berechnung der elementweisen und absoluten Differenz zwischen benachbarten Gewichten. Hierbei ist konkret definiert, zwischen welchen beiden Gewichten die Operation stattfindet, beispielsweise in Zeile 2 zwischen `weights_0` und `weights_1`. Abschließend wird die Anzahl der Einträge, das heißt die Anzahl an Tensoren, die die Delta-Werte beinhalten, ausgegeben. Dabei fällt auf, dass nach jeder Delta-Berechnung die Anzahl der Einträge in der Liste `tensor_deltas` jeweils um eins inkrementiert wird. Ein Eintrag entspricht daher einem Tensor, der die Deltas beider benachbarten Gewichte beinhaltet.

```
Übereinstimmung der Formen von 'weights_0' und 'weights_1':  torch.Size([512, 784]) und torch.Size([512, 784])
Berechnung der elementweisen und absoluten Differenz zwischen 'weights_0' und 'weights_1'
Abzahl der Einträge (Deltas) in tensor_deltas: 1
============================
Übereinstimmung der Formen von 'weights_1' und 'weights_2':  torch.Size([512, 784]) und torch.Size([512, 784])
Berechnung der elementweisen und absoluten Differenz zwischen 'weights_1' und 'weights_2'
Abzahl der Einträge (Deltas) in tensor_deltas: 2
============================
Übereinstimmung der Formen von 'weights_2' und 'weights_3':  torch.Size([512, 784]) und torch.Size([512, 784])
Berechnung der elementweisen und absoluten Differenz zwischen 'weights_2' und 'weights_3'
Abzahl der Einträge (Deltas) in tensor_deltas: 3
============================
Übereinstimmung der Formen von 'weights_3' und 'weights_4':  torch.Size([512, 784]) und torch.Size([512, 784])
Berechnung der elementweisen und absoluten Differenz zwischen 'weights_3' und 'weights_4'
Abzahl der Einträge (Deltas) in tensor_deltas: 4
============================
Übereinstimmung der Formen von 'weights_4' und 'weights_5':  torch.Size([512, 784]) und torch.Size([512, 784])
Berechnung der elementweisen und absoluten Differenz zwischen 'weights_4' und 'weights_5'
Abzahl der Einträge (Deltas) in tensor_deltas: 5
```

Abbildung 4.11: Die Konsolenausgabe verdeutlicht die elementweise Subtraktion der Tensorgewichte während des Modelltrainings bei fünf HL sowie die Anzahl der Einträge in der Tensor-Delta-Liste.

Im Rahmen des Slicings wird in Abbildung 4.12 zunächst die aktuelle Form der ersten drei Einträge i in `tensor_deltas[i]` ausgegeben und diese auf Übereinstimmung mit der Zielform überprüft. Auffällig ist, dass bei dem Eintrag `tensor_deltas[0]` eine sofortige Übereinstimmung der Shapes von $[512, 784]$ herrscht. Das bedeutet, es wird kein Slicing überschüssiger Tensorwerte durchgeführt. Die folgendenen Einträge $[1]$ bis $[4]$ weisen einen Unterschied der aktuellen Form ($[512, 784]$) und der Zielform ($[512, 512]$) auf, worauf das Abtrennen der überschüssigen Nullen in Form des Slicings erfolgt. Abschließend wird die neue Form der Delta-Tensoren je Eintrag ausgegeben. Warum sich die Zielform bei `tensor_deltas[0]` und den verbleibenden Einträgen unterscheidet, wird in Kapitel 5.3.6 verdeutlicht.

```
Aktuelle Form von tensor_deltas[0]: torch.Size([512, 784])
Übereinstimmung der Zielform  torch.Size([512, 784]) und der Originalform  torch.Size([512, 784])
====================
Aktuelle Form von tensor_deltas[1]: torch.Size([512, 784])
keine Übereinstimmung der Zielform  torch.Size([512, 512]) und der Originalform  torch.Size([512, 784])
Abtrennen der überschüssigen Nullen durch Slicing
Neue Form von tensor_deltas[1]:      (512, 512) als Numpy Array
====================
Aktuelle Form von tensor_deltas[2]: torch.Size([512, 784])
keine Übereinstimmung der Zielform  torch.Size([512, 512]) und der Originalform  torch.Size([512, 784])
Abtrennen der überschüssigen Nullen durch Slicing
Neue Form von tensor_deltas[2]:      (512, 512) als Numpy Array
```

Abbildung 4.12: Die gekürzte Konsolenausgabe verdeutlicht die interne Vorgehensweise während des Slicings der Tensoren vor dem Speichern der Delta-Werte. Hierbei sind die Tensor-Delta-Ausgaben des CL-Modells mit fünf HL zu sehen.

Die Abbildung 4.13 beschreibt die deskriptive Statistik des Python-Dataframes `df_deltas`, welches die Delta-Werte von `tensor_deltas` beinhaltet. Die Statistik dient als Zusammenfassung der Tensorwerte innerhalb des Dataframes. Hierbei wird das Vorhandensein erster Nullwerte innerhalb der `min`-Zeile deutlich. In diesem Fall sind in den ersten zehn Spalten (von insgesamt 784) mindestens vier Nullwerte vorhanden.

	0	1	2	3	4	5	6	7	8	9	...
count	512.0000	512.0000	512.0000	512.0000	512.0000	512.0000	512.0000	512.0000	512.0000	512.0000	...
mean	0.0172	0.0175	0.0184	0.0182	0.0180	0.0181	0.0178	0.0185	0.0194	0.0186	...
std	0.0111	0.0102	0.0109	0.0111	0.0106	0.0104	0.0109	0.0106	0.0108	0.0108	...
min	0.0000	0.0001	0.0000	0.0000	0.0000	0.0001	0.0000	0.0003	0.0002	0.0001	...
25%	0.0079	0.0087	0.0091	0.0083	0.0096	0.0095	0.0085	0.0097	0.0104	0.0098	...
50%	0.0165	0.0174	0.0182	0.0178	0.0168	0.0178	0.0179	0.0183	0.0203	0.0181	...
75%	0.0256	0.0258	0.0271	0.0274	0.0269	0.0266	0.0264	0.0277	0.0277	0.0274	...
max	0.0836	0.0551	0.0692	0.0646	0.0621	0.0639	0.0741	0.0618	0.1072	0.0777	...

8 rows × 784 columns

Abbildung 4.13: Die beschreibenden Statistiken `count`, `mean`, `std`, `min`, 25 %, 50 %, 75 % und `max` des vorliegenden Dataframe werden durch die Konsolenausgabe dargestellt. Die grün umrandeten Werte entsprechen den gesuchten Nullwerten.

Entsprechend der Abbildung 4.14 ist ein Auszug des Endergebnisses in Form von gespeicherten Delta-Werten dargestellt. In dem vorliegenden Beispiel der Datei `delta_weights_1hl.txt` ist konkret der Beginn der ersten zehn Zeilen erkennbar. Die Delta-Werte wurden dabei auf vier Nachkommastellen gerundet und sind eindeutig durch ein Leerzeichen voneinander getrennt. Der erste Eintrag der Delta-Logdatei in Abbildung 4.14 ist grün umrandet und beträgt 0.0486. Diese Delta-Ausgabe stützt die in Abschnitt 4.3.2 durchgeführte Berechnung des Änderungsverlaufs der ersten Gewichtswerte.

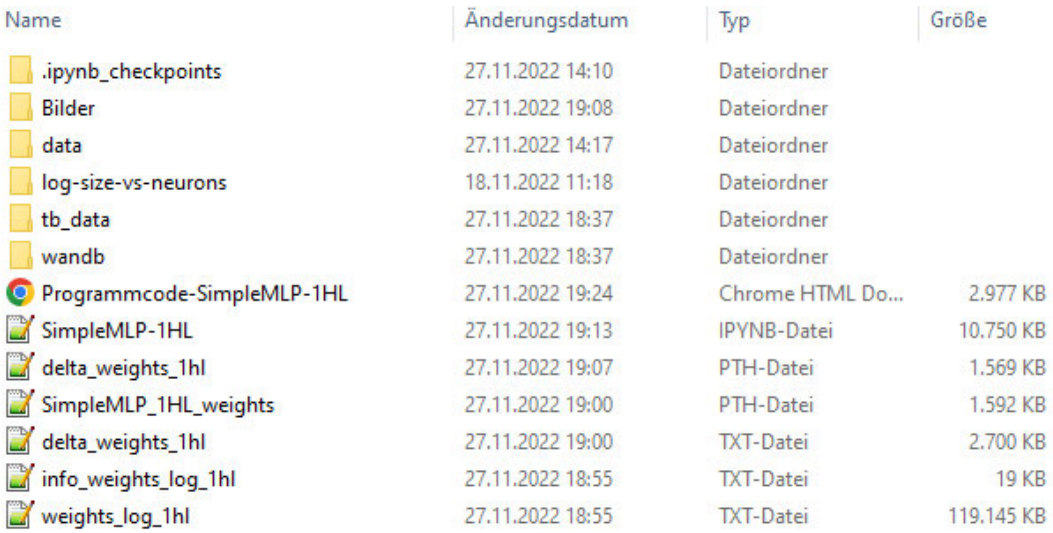
delta_weights_1hl.txt

```
1  0.0486 0.013 0.0632 0.0363 0.0388 0.0413 0.0721 0.0213 0.0129 0.0516
2  0.0536 0.0016 0.04 0.0456 0.0552 0.0451 0.0741 0.0098 0.1072 0.0161 0
3  0.0268 0.0163 0.0275 0.0101 0.0035 0.0278 0.0322 0.0271 0.0409 0.0486
4  0.0169 0.0551 0.0692 0.025 0.0162 0.0639 0.0358 0.0076 0.0221 0.0468
5  0.0068 0.0332 0.0026 0.0606 0.0621 0.0261 0.0044 0.0097 0.0098 0.0777
6  0.0045 0.0267 0.0048 0.0312 0.0103 0.0121 0.0155 0.0618 0.0142 0.063
7  0.0244 0.0341 0.0437 0.0636 0.0384 0.0014 0.0524 0.0174 0.0002 0.0319
8  0.0836 0.0171 0.0232 0.0207 0.0093 0.0493 0.0061 0.0373 0.0212 0.0268
9  0.065 0.0212 0.0204 0.0646 0.0345 0.0275 0.014 0.0163 0.0119 0.008 0.
10 0.0157 0.021 0.0062 0.0282 0.0056 0.0118 0.0364 0.0284 0.0091 0.0125
```

Abbildung 4.14: Der Ausschnitt der Logdatei `delta_weights_1hl.txt` zeigt die gerundeten Delta-Werte des CL-Modells SimpleMLP mit einem HL.

4.3.6 Zusammenfassung

Die Abbildung 4.15 zeigt die erstellten Dateien nach einem vollständigen Durchlauf der Trainingsschleife sowie dem anschließenden Speichern der Delta-Werte. Für jede der fünf verschiedenen CL-Modelle werden die gleichen fünf Dateien erstellt, jedoch mit verschiedenen Inhalten (zum Beispiel Modellgewichten). Zunächst wird vor dem Beginn des Modelltrainings die Datei `weights_log.txt` erstellt. Es folgt zu Beginn des laufenden Trainings die Datei `info_weights_log.txt`. Anschließend geschieht das Erstellen der Dateien `model_weights.pth` nach dem Speichern der Modellgewichte und `delta_weights.txt` nach der abgeschlossenen Trainingsphase. Letztendlich wird nach dem Speichern der Delta-Werte die Datei `delta_weights.pth` ausgegeben.
Neben der Jupyter Notebook Datei, die den Programmcode beinhaltet, sowie den fünf resultierenden Ausgabedateien sind Datei-Ordner für die TensorBoard-Dateien, die W&B-Dateien und die erstellten Visualisierungen vorhanden. Es werden die MNIST-Daten in `data` gespeichert. Während der Programmausführung werden außerdem regelmäßige Checkpoints von Jupyter Notebook erstellt. Die Dateien für die Visualisierung der Speichergröße vs. der Neuronenanzahl werden ebenfalls separat gesichert.



Name	Änderungsdatum	Typ	Größe
.ipynb_checkpoints	27.11.2022 14:10	Dateiordner	
Bilder	27.11.2022 19:08	Dateiordner	
data	27.11.2022 14:17	Dateiordner	
log-size-vs-neurons	18.11.2022 11:18	Dateiordner	
tb_data	27.11.2022 18:37	Dateiordner	
wandb	27.11.2022 18:37	Dateiordner	
Programmcode-SimpleMLP-1HL	27.11.2022 19:24	Chrome HTML Do...	2.977 KB
SimpleMLP-1HL	27.11.2022 19:13	IPYNB-Datei	10.750 KB
delta_weights_1hl	27.11.2022 19:07	PTH-Datei	1.569 KB
SimpleMLP_1HL_weights	27.11.2022 19:00	PTH-Datei	1.592 KB
delta_weights_1hl	27.11.2022 19:00	TXT-Datei	2.700 KB
info_weights_log_1hl	27.11.2022 18:55	TXT-Datei	19 KB
weights_log_1hl	27.11.2022 18:55	TXT-Datei	119.145 KB

Abbildung 4.15: Der Screenshot des Datei-Explorers zeigt die erstellten Dateien und Ordner nach erfolgreicher Beendigung des Programms. Konkret ist die Ausgabe des CL-Modells SimpleMLP-5 erkennbar, inklusive Ordner- und Dateiname, Änderungsdatum, Dateityp und Speichergröße.

Die Tabelle 4.3 fasst die relevanten Werte des Ergebnisteils der internen Logging Funktionalität zusammen und gibt eine Übersicht zu den verwendeten CL-Modellen verschiedener Größe. Bei der Betrachtung der Modell-Laufzeit in Minuten fällt auf, dass die zeitliche Dauer des Trainingsprozesses von einem hin zu 100 HL ansteigt. Gleichermaßen erhöht sich die Größe der beiden Dateien pro Konfiguration. Die Dateien Weights-Log und Delta-Log unterscheiden sich jedoch in ihrer Größenordnung. Erstere besitzt einen Speicherbedarf im unteren GB-Bereich, wobei zweitere in MB angegeben wird. Auffallend hierbei ist, dass beispielsweise bei zehn HL nach ca. 20 Minuten 0.8 GB an Daten anfallen. Im Vergleich dazu werden bei 100 HL nach ca. 4.2 Stunden Laufzeit 7.7 GB Daten gespeichert. Bei den Werten des Prozentsatzes an Nullen fällt lediglich eine gleichbleibende Tendenz auf, die bei einem HL minimal von den anderen Werten abweicht.

Tabelle 4.3: Die Zusammenfassung der internen Logging Funktionalität umfasst die Laufzeit des Modelltrainings, die Größe des Weights-Log, die Größe des Delta-Log sowie der Prozentsatz an Nullen in der Delta-Logdatei je SimpleMLP-Modell-Konfiguration.

Anzahl der Hidden Layer in SimpleMLP	**1**	**5**	**10**	**50**	**100**
Laufzeit Modelltraining [min]	15.35	12.79	21.11	98.42	251.33
Größe Weights-Log [GB]	0.12	0.43	0.81	3.89	7.73
Größe Delta-Log [MB]	2.70	9.75	18.57	89.09	177.24
Prozentsatz an Nullen in der Delta-Logdatei	0.0014	0.0011	0.0011	0.0011	0.0011

4.4 Visualisierung der Abhängigkeiten zwischen Modellkonfiguration, Trainingsgeschwindigkeit und Speicherplatzbedarf

Auf Basis der Ergebnisse des zuvor konfigurierten CL-Modells werden die Relationen zwischen der Modellkonfiguration, der Geschwindigkeit des Modelltrainings und dem Speicherplatzbedarf der resultierenden Logdatei anhand von Visualisierungen dargestellt. Hierbei werden konkret die Speichergröße der Weights-Logdatei vs. dem Trainingszyklus, die Speichergröße der Weights-Logdatei vs. der Anzahl der Neuronen und der Anteil an Nullen in der Delta-Logdatei vs. dem Trainingszyklus betrachtet. Insgesamt resultierten 15 Visualisierungen in Form der Abbildungen 4.16 bis 4.30.

4.4.1 Speichergröße der Weights-Logdatei vs. Trainingszyklus

Die nachfolgenden Abbildungen 4.16 bis 4.20 verdeutlichen die Speichergröße der Logdatei in Abhängigkeit zu der aktuellen Experience während des Trainingszyklus in Form eines Liniendiagramms. Insgesamt werden innerhalb des Modelltrainings fünf Experiences betrachtet. Die Diagramme unterscheiden sich hinsichtlich der Anzahl der HL des CL-Modells SimpleMLP. Neben dem Speicherbedarf wird ebenso die Laufzeit t pro Experience visualisiert.

Das Diagramm 4.16 verdeutlicht zum Zeitpunkt der ersten Experience (0) eine Speichergröße von 24.40 MB. Diese steigt linear an, bis bei der Experience 4 ca. 122.00 MB an Speicherplatz verwendet wurde. Die Steigung der Geraden beträgt gleichbleibend 24.4. Die Laufzeit umfasst zu Beginn der ersten Experience 1.97 min. Bei Experience 4 ist eine Laufzeit t von 15.07 Minuten zu verzeichnen. Der zeitliche Verlauf gestaltet sich ebenfalls monoton steigend und linear.

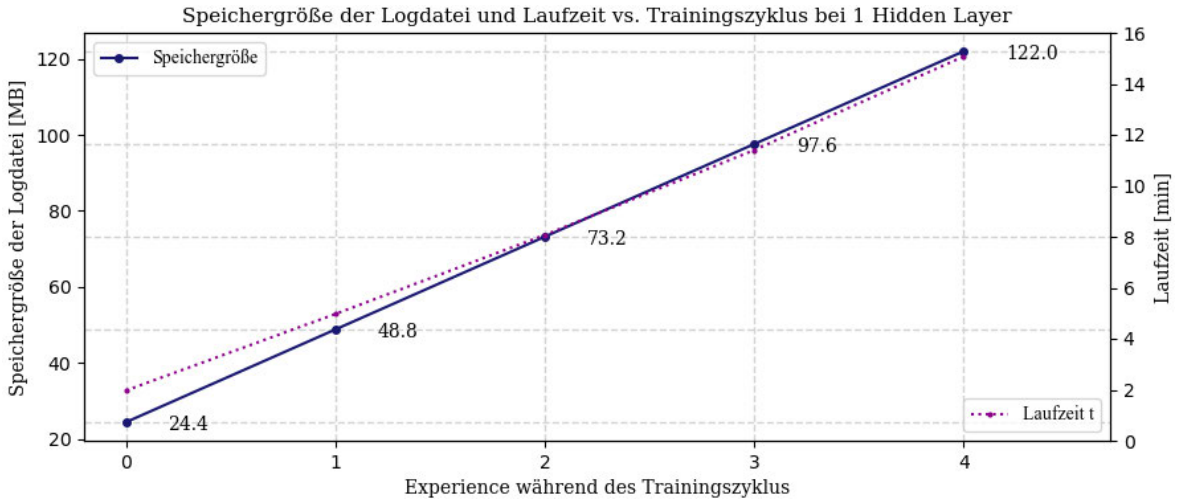


Abbildung 4.16: Visualisierung der Speichergröße der Weights-Logdatei und der Laufzeit vs. Trainingszyklus bei einem HL

Analog zu 4.16 verdeutlicht das Diagramm 4.17 zum Zeitpunkt der Experience 0 eine Speichergröße von 87.37 MB. Diese steigt linear an, bis bei der Experience 4 ca. 436.87 MB an Speicherplatz verwendet wurde. Der Anstieg der Geraden umfasst konstant 87.38. Die Laufzeit t steigt von Experience 0 bei 1.61 min bis Experience 4 auf 12.53 min monoton an. Der zeitliche Verlauf kann als annähernd linear steigend charakterisiert werden.

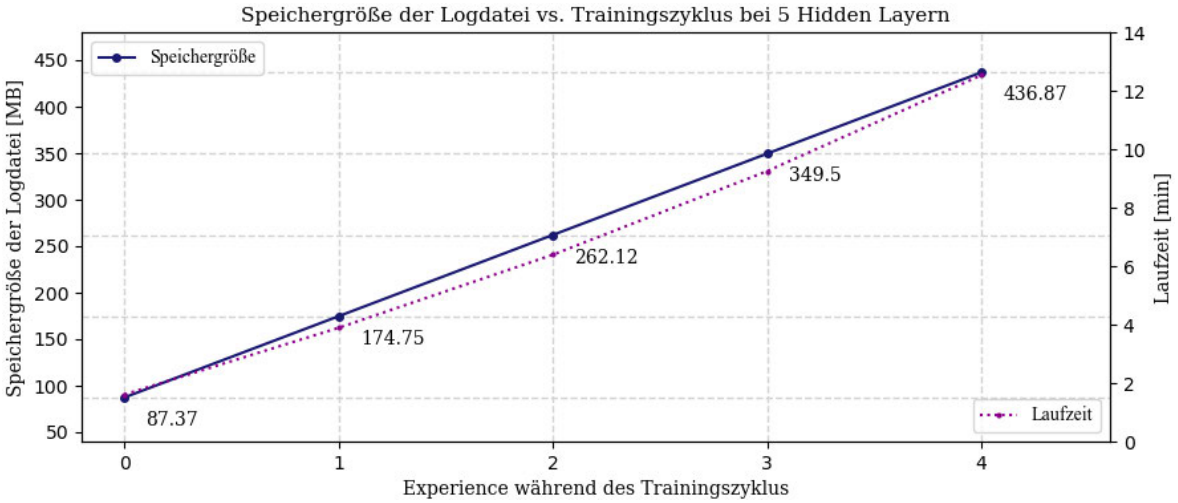


Abbildung 4.17: Visualisierung der Speichergröße der Weights-Logdatei und der Laufzeit vs. Trainingszyklus bei fünf HL

Bei der Abbildung 4.18 ist von Experience 0 bis 4 ein monoton steigender Verlauf des Graphen von 166.09 MB auf 830.45 MB an Speichergröße der Logdatei erkennbar. Der Anstieg der Geraden umfasst konstant 166.09. Der Zeitverlauf steigt von 2.64 min bei der ersten Experience auf 20.68 min bei Experience 4. Dabei kann der monotone Anstieg als annähernd linear charakterisiert werden.

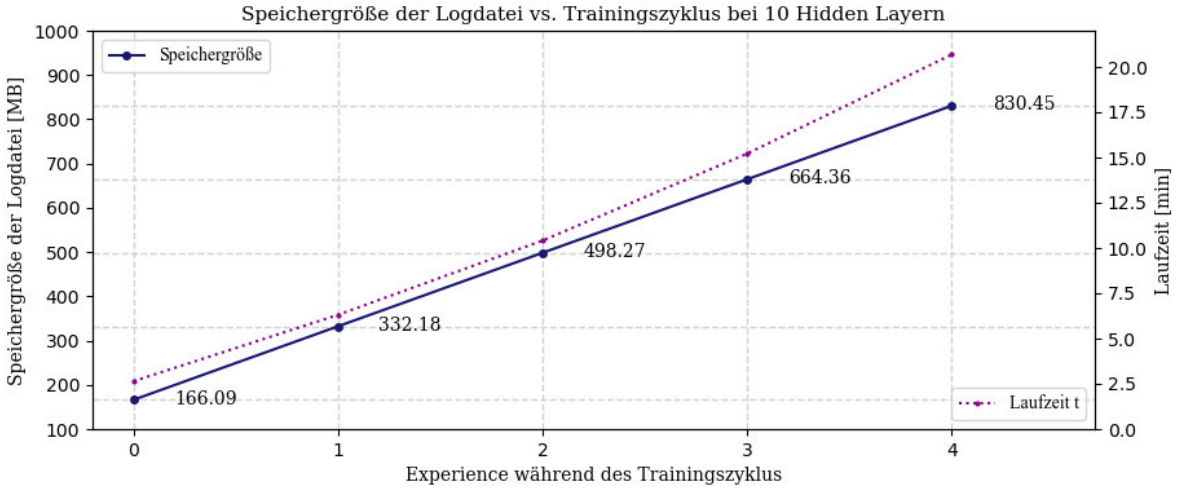


Abbildung 4.18: Visualisierung der Speichergröße der Weights-Logdatei und der Laufzeit vs. Trainingszyklus bei zehn HL

Die Abbildung 4.19 zeigt, dass zum Zeitpunkt der Experience 0 eine Speicheraufwand von 0.8 GB entsteht. Es folgt ein linearer und stetiger Anstieg auf 3.98 GB bei der Experience 4. Die Steigung der Geraden beträgt konstant 0.79. Die Laufzeit t steigt ebenfalls monoton von 22.67 min bis 95.21 min annähern linear an. Hierbei übersteigt die Modelllaufzeit zum ersten Mal die zeitliche Marke von einer Stunde.

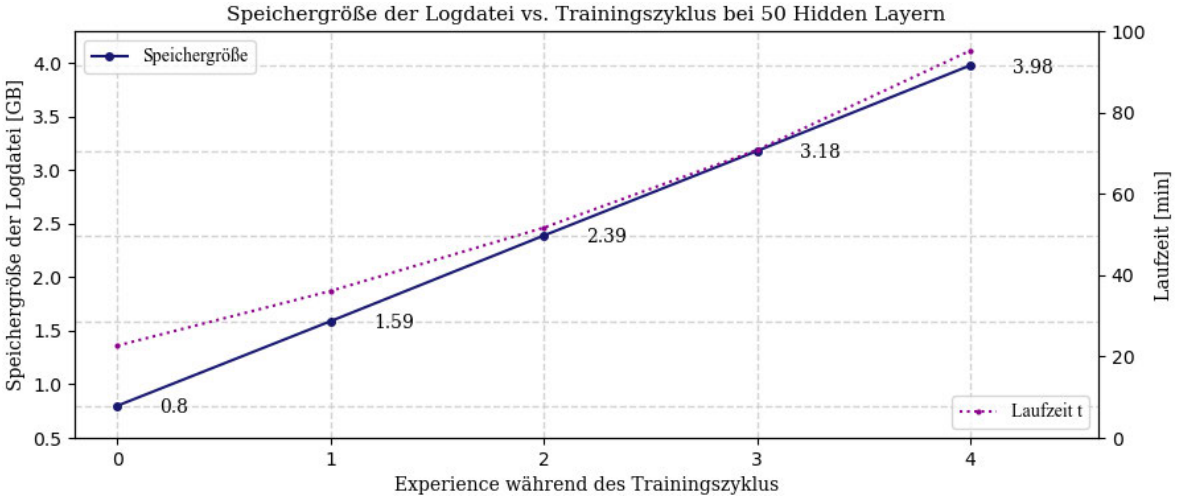


Abbildung 4.19: Visualisierung der Speichergröße der Weights-Logdatei und der Laufzeit vs. Trainingszyklus bei 50 HL

Gleichermaßen verhält sich der Anstieg des Speicherbedarfs bei Abbildung 4.20, welcher kontinuierlich von 1.58 GB auf 7.91 GB ansteigt. Der Anstieg der Geraden umfasst konstant 1.58. Die Laufzeit verhält sich ebenso monoton steigend von 24.29 min auf 3.93 h. Dabei ist ein annähernd linearer Verlauf charakteristisch für den zeitlichen Anstieg t.

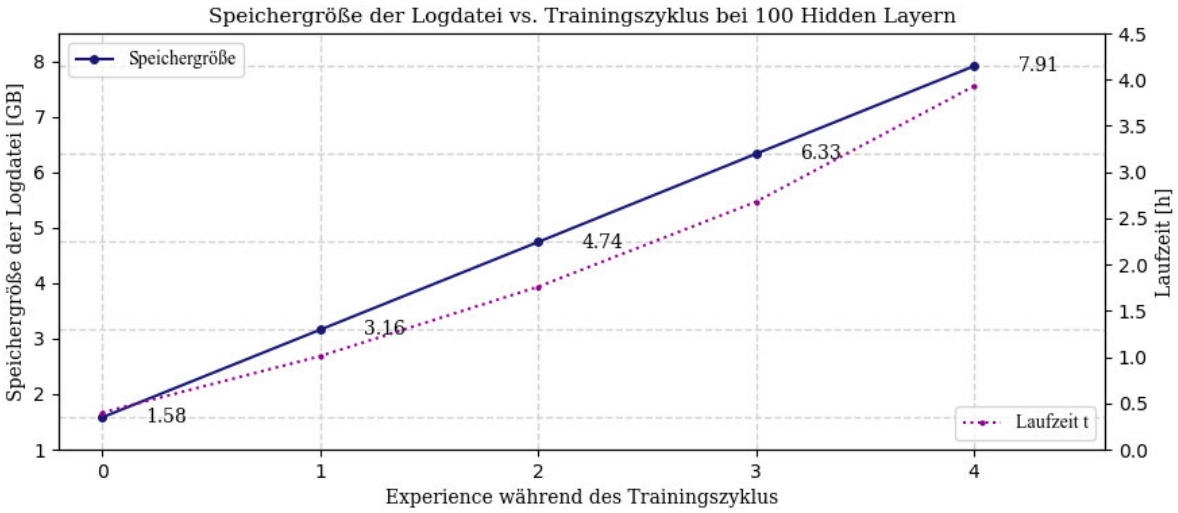


Abbildung 4.20: Visualisierung der Speichergröße der Weights-Logdatei und der Laufzeit vs. Trainingszyklus bei 100 HL

Erweiterte Laufzeitbetrachtungen Für alle fünf Modelle wurden zusätzlich `time_checkpoints` nach der Protokollierung jedes Trainingsgewichts gesetzt, um die Dateigröße in regelmäßigen Abständen zu speichern. In den Diagrammen D.1 bis D.5 in Anhang D ist anhand dieser Checkpoints der Anstieg der Speichergröße mit einer größeren Anzahl an Zwischenwerten dargestellt. Hierbei ist eine leichte Stufenform erkennbar, die sich mit steigender Laufzeit gleichmäßig fortsetzt. Der Funktionsgraph weist jeweils fünf Ebenen auf, die weniger stark ansteigen im Vergleich zu den Bereichen vor und nach der Stufe. Mit größer werdender Anzahl an HL und der einhergehenden Vergrößerung der gespeicherten Zwischenwerte, weist die Stufenform eine weniger starke Ausprägung auf.

Zusammenfassung Bei der Betrachtung der Abbildungen 4.16 bis 4.20 fällt eine große Ähnlichkeit der Kurvenverläufe auf. Alle fünf Diagramme weisen einen streng monoton steigenden Verlauf im positiven Bereich auf, der durch sein lineares Verhalten charakterisiert ist. Sie unterscheiden sich jedoch bei der Höhe der Speichergröße der jeweiligen Logdatei. Je größer die Anzahl der HL des ML-Modells ist, desto größer ist die Speichergröße der resultierenden Logdatei und der somit entstehende Speicherbedarf. Relativ zu den x-Werten der Funktion verhält sich die Änderung der y-Werte gleichbleibend. Der Anstieg der Geraden ist konstant.
Die Beschreibung des Funktionsgraphens der Speichergröße kann ebenso auf den Verlauf der Laufzeit t angewendet werden. Im Zuge des Laufzeitverhaltens fällt bei allen Visualisierungen auf, dass diese zum Zeitpunkt der Experience 0 nicht bei Minute null beginnt und dass die Laufzeit bei der letzten Experience (4) nicht mit der Gesamtzeit des Modelltrainings in Tabelle 4.3 übereinstimmt. Eine Erklärung dieser Gegebenheit erfolgt in Kapitel 5.4.1.

4.4.2 Speichergröße der Weights-Logdatei vs. Anzahl der Neuronen

Nachfolgend werden die Ergebnisse des Vergleichs der Speichergröße nach dem Modelltraining in Abhängigkeit zu der Anzahl der Neuronen in den Abbildungen 4.21 bis 4.25 dargestellt. Hierbei werden fünf Liniendiagramme zu den SimpleMLP-Modellen mit 1, 5, 10, 50 und 100 HL erstellt. Die Anzahl der Neuronen bei einem und fünf HL beträgt: 512, 1024, 1536, 2048, 2560 und 3072. Im Vergleich dazu werden bei den Modellen mit zehn, 50 und 100 HL der Speicherbedarf bei 128, 256, 384, 512, 640 und 768 Neuronen betrachtet. Dabei wird das Änderungsverhalten der Funktion aufgezeigt. Zusätzlich zum Speicherbedarf wird die benötigte Laufzeit t während des Modelltrainings in die Visualisierung eingebunden.

In Abbildung 4.21 ist ein positiv lineares Wachstum der Speichergröße bei einem HL von 119.15 MB bei 512 Neuronen auf 714.76 MB bei 3072 Neuronen erkennbar. Die Steigung der Geraden beträgt gleichbleibend rund 0.23. Die Laufzeit t des Modells weist bei 1536 Neuronen einen augenscheinlichen Einbruch von 9.68 min auf. Ansonsten ist jedoch insgesamt ein leichter Anstieg von 14.35 min bei 512 Neuronen auf 16.52 min bei 3072 Neuronen zu verzeichnen.

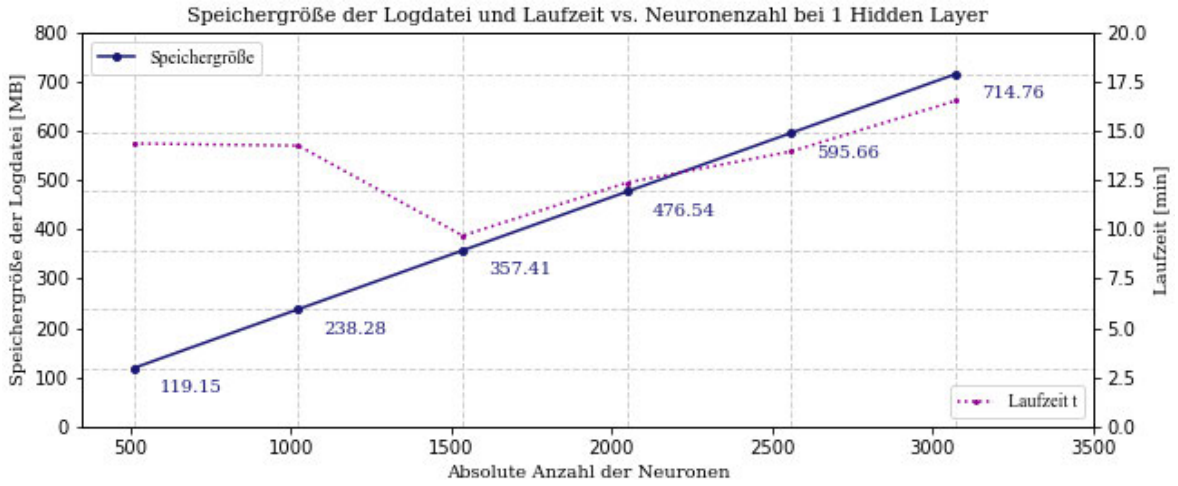


Abbildung 4.21: Visualisierung der Speichergröße der Weights-Logdatei und der Laufzeit vs. Anzahl der Neuronen bei 1 HL

Die Abbildung 4.22 zeigt den Verlauf des Speicherbedarfs von 0.43 GB auf 11.78 GB bei fünf HL. Das Wachstum verläuft im positiven Bereich monoton steigend. Das Änderungsverhalten der Funktion wird durch die steigenden Deltawerte 0.002, 0.003, 0.004, 0.006 und 0.007 charakterisiert. Das Wachstum ähnelt im Gegensatz zu Abbildung 4.21 annähernd einem quadratischen Anstieg. Die Laufzeit t ist ebenfalls durch ein positives Wachstum charakterisiert. Hierbei steigt es von 13.05 min auf 187.65 min an. Auffällig ist jedoch ein leichter Abfall von t zwischen 2560 und 3072 Neuronen.

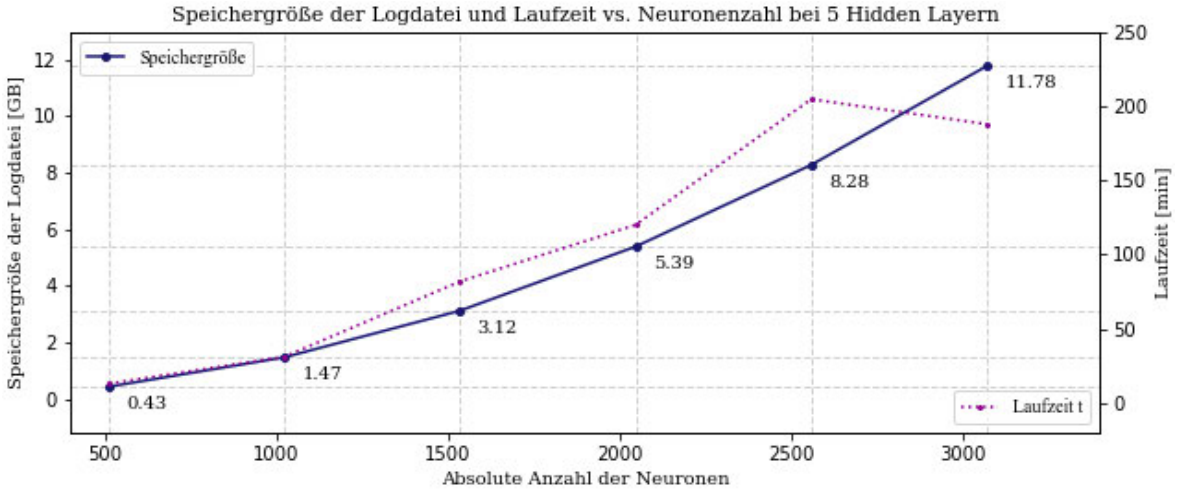


Abbildung 4.22: Visualisierung der Speichergröße der Weights-Logdatei und der Laufzeit vs. Anzahl der Neuronen bei 5 HL

Die Speichergröße des Modells mit zehn HL wächst in Abbildung 4.23 von 73.13 MB bei 512 Neuronen auf 1734.60 MB bei 768 Neuronen. Das Änderungsverhalten der Funktion wird durch die steigenden Deltawerte 1.25, 1,92, 2,60, 3,27 und 3.95 charakterisiert. Der Funktionsgraph steigt hierbei monoton im positiven Definitions- und Wertebereich und weist einen annähernd quadratischen Charakter auf. Das Änderungsverhalten der Funktion wird durch die steigenden Deltawerte 1.25, 1.92, 2.60, 3.27 und 3.95 charakterisiert. Bei der Betrachtung der Laufzeit t fallen eindeutige Parallelen zu dem Verlauf des Speicherbedarfs auf. Aufgrund dessen ist ebenfalls ein nicht linear steigender Funktionsverlauf im positiven Bereich zu erkennen, der von 5.48 min bei 128 Neuronen bis 34.82 bei 768 Neuronen reicht.

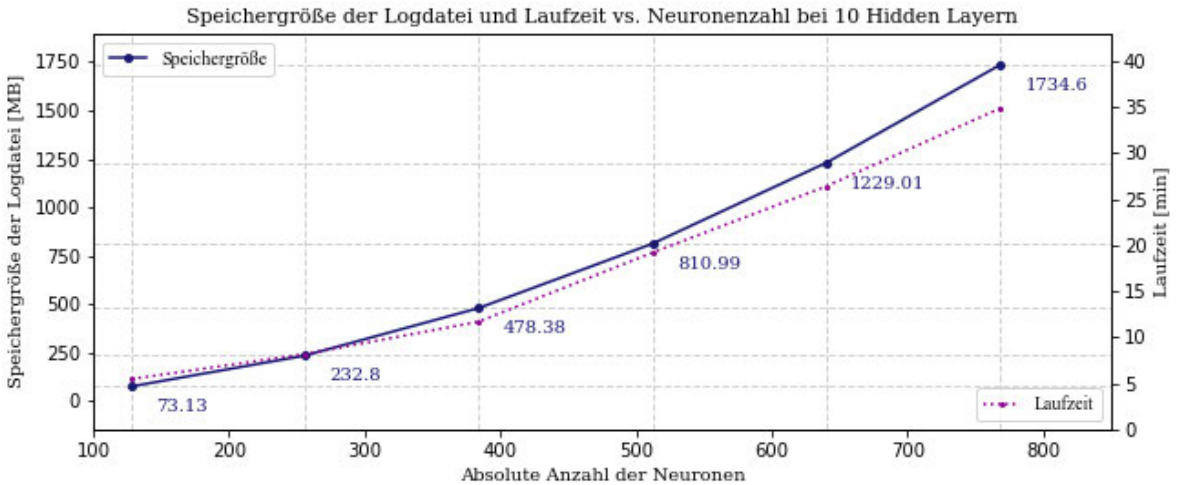


Abbildung 4.23: Visualisierung der Speichergröße der Weights-Logdatei und der Laufzeit vs. Anzahl der Neuronen bei 10 HL

Anhand Abbildung 4.24, welche den Verlauf des Speicherbedarfs für 50 HL beinhaltet, ist eine Annäherung des Funktionsgraphen an ein quadratisches Wachstum einsehbar. Konkret findet ein monotoner Anstieg im positiven Bereich von 0.26 GB auf 8.65 GB statt. Das Änderungsverhalten der Funktion wird durch die steigenden Deltawerte 0.006, 0.009, 0.013, 0.016 und 0.019 charakterisiert. Mit großer Ähnlichkeit verläuft die Kurve der Laufzeit t, welche von 9.61 min auf 217.94 min ansteigt.

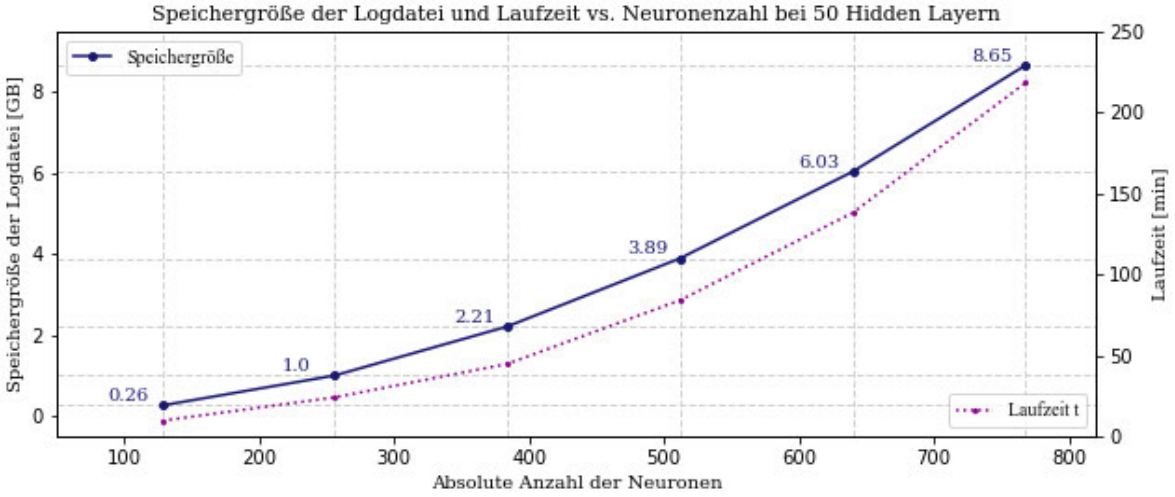


Abbildung 4.24: Visualisierung der Speichergröße der Weights-Logdatei und der Laufzeit vs. Anzahl der Neuronen bei 50 HL

Das Modell SimpleMLP-100HL weist bei größer werdender Neuronenzahl einen annähernd quadratisch größer werdenden Speicherbedarf auf. Konkret wird dieser monotone Anstieg verdeutlicht durch die Werte: 0.51 GB bei 128 Neuronen und 17.29 GB bei 768 Neuronen. Das Änderungsverhalten der Funktion wird durch die steigenden Deltawerte 0.011, 0.019, 0.026, 0.033 und 0.041 charakterisiert. Im Zuge der Vervielfachung der Neuronen steigt ebenfalls die Laufzeit vergleichbar zu dem Speicherplatzbedarf. Hierbei ist zu Beginn der Funktion der Punkt $(128, 18.66)$ zu verzeichnen. Das Ende des Verlauf stellt der Punkt $(768, 323.43)$ dar, woraus sich eine Gesamtlaufzeit von 323.43 Minuten bei 768 Neuronen ergibt.

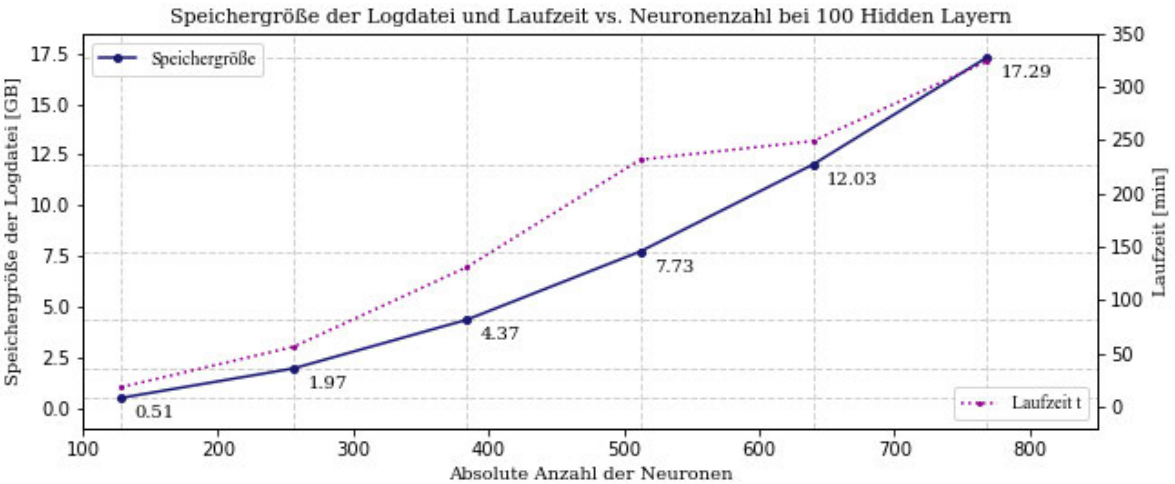


Abbildung 4.25: Visualisierung der Speichergröße der Weights-Logdatei und der Laufzeit vs. Anzahl der Neuronen bei 100 HL

Zusammenfassung Bis auf das lineare Verhalten des ersten Funktionsgraphen besitzen die verbleibenden Funktionen einen annähernd quadratischen Charakter, da sie dem Verlauf einer Parabel ähneln und jeweils eine nahezu gleichbleibende Wachstumsänderung aufweisen. Das bedeutet, die Steigung der Speichergröße ändert sich linear bei gleichförmiger Vergrößerung der Neuronenzahl. Die Änderungsrate nimmt demnach proportional zu. Der Verlauf ist im positiven Definitions- und Wertebereich lokalisiert. Diese Beschreibung der Funktionsgraphen kann bis auf den Ausnahmefall bei einem HL auf die Laufzeit t angewendet werden.

4.4.3 Anteil an Nullen in der Delta-Logdatei vs. Trainingszyklus

In den fünf Abbildungen 4.26 bis 4.30 wird der prozentuale Anteil an Nullen in der Delta-Logdatei in Abhängigkeit zu dem aktuellen Trainingszyklus aufgezeigt. Dies geschieht in Form von je einem Liniendiagramm zu den SimpleMLP-Modellen mit 1, 5, 10, 50 und 100 HL.

In Abbildung 4.26 ist der gesamte Prozentsatz an Nullen von $p = 0.0014$ am Ende des Modelltrainings bei einem HL ersichtlich. Obwohl der Graph innerhalb des Wertebereichs von $W_f = [0.0, 1.0]$ keine sichtbare Steigung aufweist, ist in der zweiten Betrachtungsweise ($W_f = [0.000, 0.002]$) eine annähernd lineare Steigung des kumulierten Prozentsatzes zu erkennen. Daraus schlussfolgernd, nähert sich der Funktionsgraph der Vergleichslinie ($f(x) = 1.0$) bei einem Prozentwert von $p = 1.0$ nicht an.

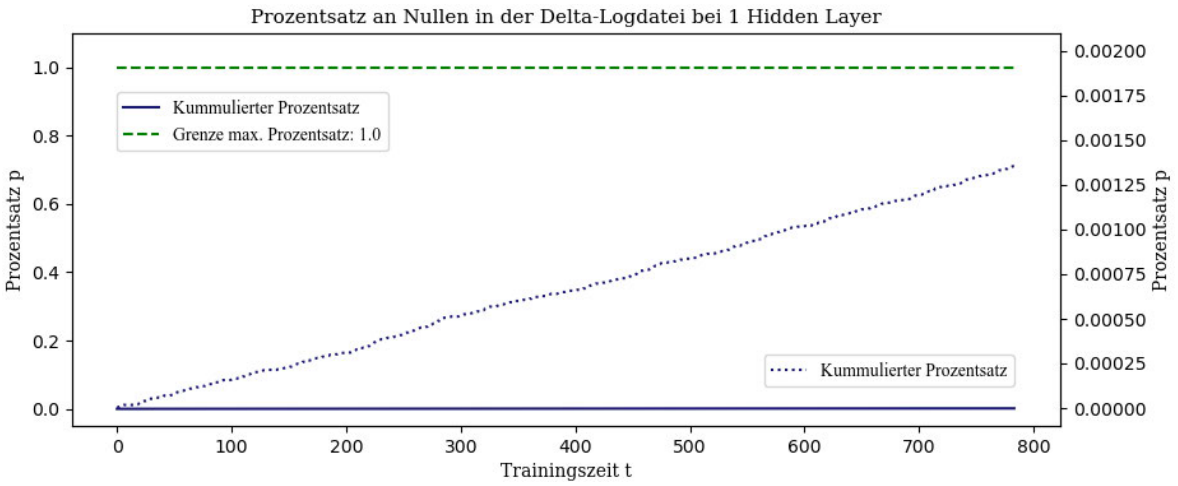


Abbildung 4.26: Visualisierung des Anteils an Nullen in der Delta-Logdatei vs. Trainingszyklus bei einem HL im Vergleich zum maximalen Prozentsatz

Die Abbildungen 4.27 bis 4.30 der SimpleMLP-Modelle mit 5, 10, 50 und 100 HL weisen sehr ähnliche Funktionsgraphen auf, weshalb sie im Folgenden gleichermaßen beschrieben werden. Vergleichbar zu Abbildung 4.26 ist kein Anstieg der kumulierten Prozentwerte erkennbar. Wird der Graph jedoch innerhalb von $W_f = [0.000, 0.002]$ betrachtet, ist eine Steigung auf 0.0011 ersichtlich. Diese weist bis zu einer Trainingszeit von 512 einen steileren Anstieg auf, welcher ab diesem Wert geringfügiger ausgeprägt ist. Beide Anstiege verlaufen annähernd linear und weisen eine monotone Steigung im positiven Bereich auf.

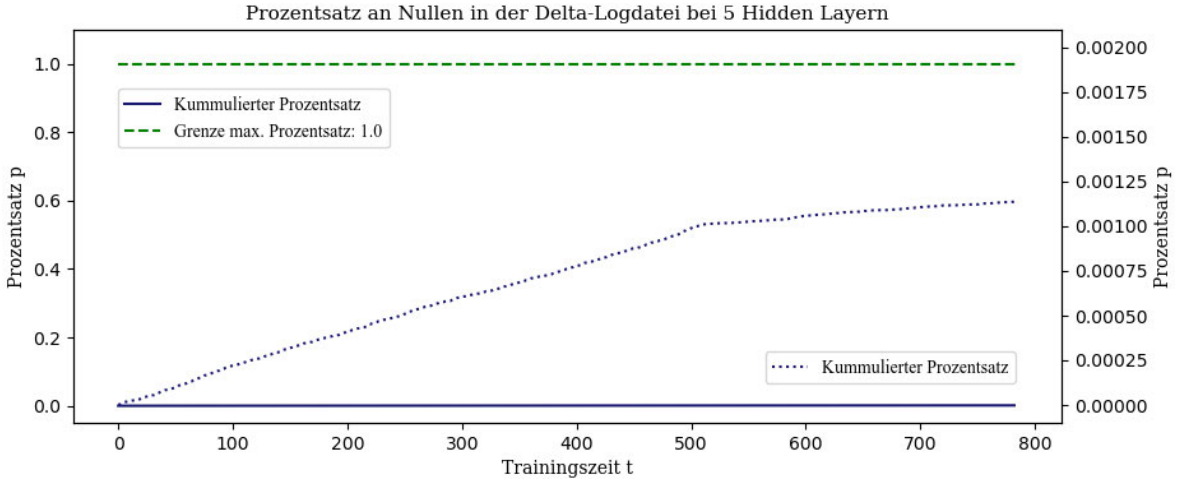


Abbildung 4.27: Visualisierung des Anteils an Nullen in der Delta-Logdatei vs. Trainingszyklus bei fünf HL im Vergleich zum maximalen Prozentsatz

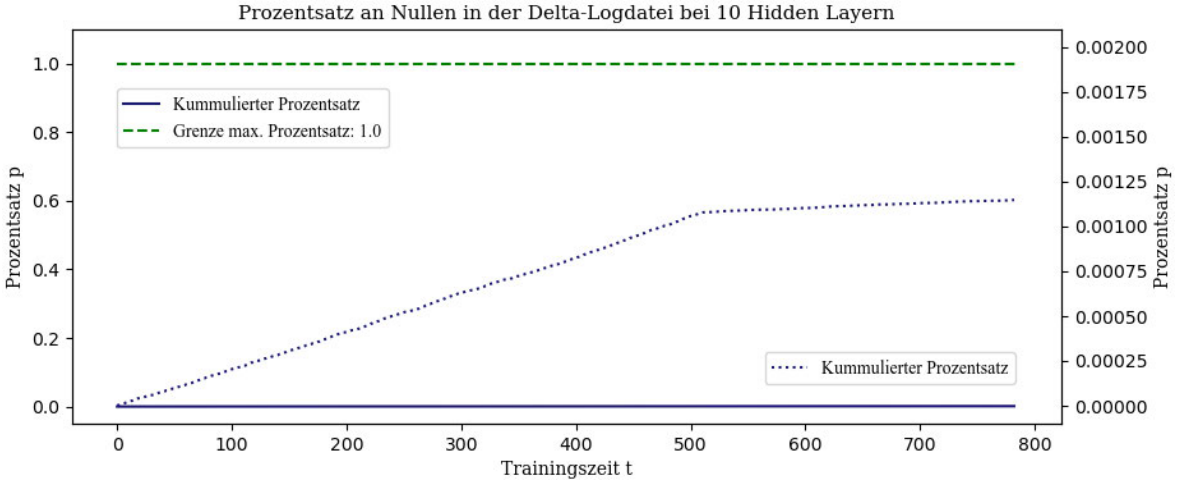


Abbildung 4.28: Visualisierung des Anteils an Nullen in der Delta-Logdatei vs. Trainingszyklus bei zehn HL im Vergleich zum maximalen Prozentsatz

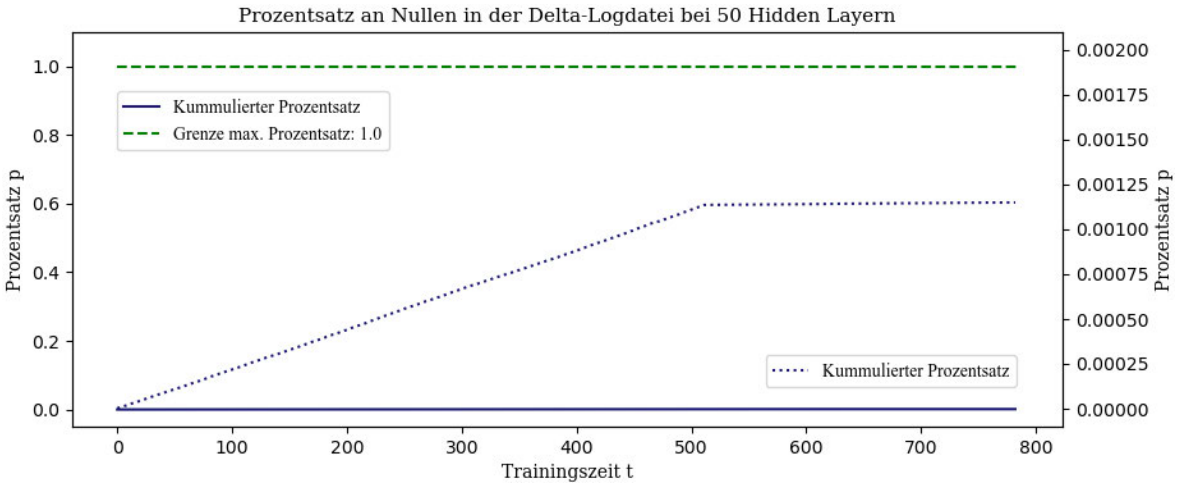


Abbildung 4.29: Visualisierung des Anteils an Nullen in der Delta-Logdatei vs. Trainingszyklus bei 50 HL im Vergleich zum maximalen Prozentsatz

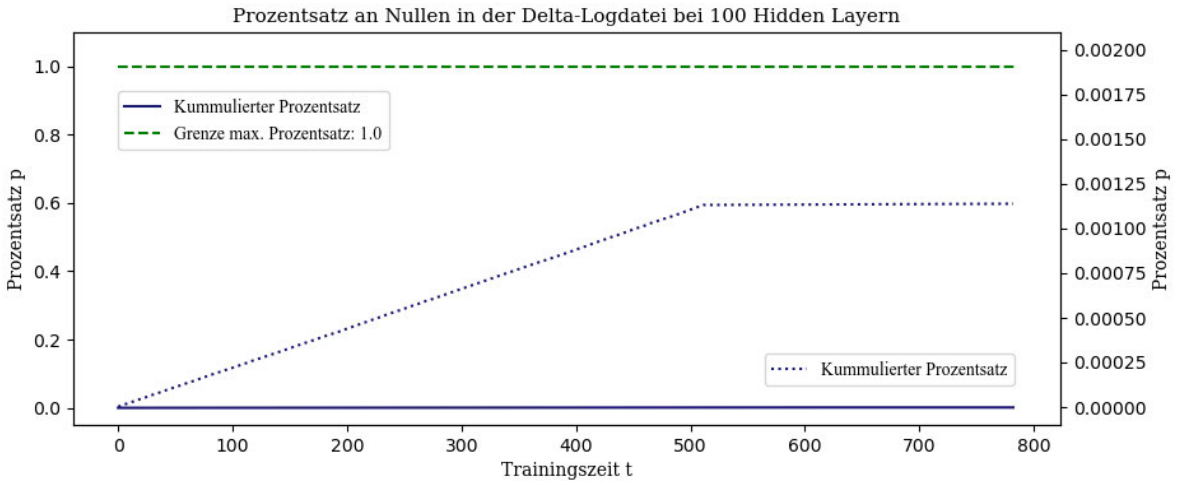


Abbildung 4.30: Visualisierung des Anteils an Nullen in der Delta-Logdatei vs. Trainingszyklus bei 100 HL im Vergleich zum maximalen Prozentsatz

Zusammenfassung Die Abbildungen 4.26 bis 4.30 gleichen sich in der Hinsicht, dass optisch kein Anstieg des Funktionsgraphen hinsichtlich des Prozentsatzes an Nullen erfolgt. Jedoch ist bei einer kleineren Einteilung der Ordinate ein geringer Anstieg von p erkennbar. Der gerundete Prozentsatz aller fünf CL-Modelle beträgt 0.001. Dieser Wert entspricht dem Vorkommen an Nullwerten in der Delta-Logdatei von 0.01%.

5 Diskussion

Das folgende Kapitel evaluiert die Ergebnisse aus Kapitel 4. Der Schwerpunkt liegt dabei auf der Beziehung der Erklärbarkeit zur Forensik. Hierbei wird analysiert, wie sich die Abhängigkeit zwischen der Modellkonfiguration, der Trainingsgeschwindigkeit und dem Speicherplatzbedarf auf die Umsetzung der Anforderungen von Auditierbarkeit anwenden lässt. Im Zuge dessen werden Herausforderungen und Möglichkeiten für Digital Forensic Readiness Maßnahmen bei kontinuierlichen KI-Systemen diskutiert, um im Falle von Manipulationsversuchen adäquat agieren zu können.

5.1 Vergleich ausgewählter Softwarebibliotheken der Erklärbaren Künstlichen Intelligenz

Die Grundvoraussetzung für eine auditierbare KI ist eine erklärbare KI. Nur wenn das Verhalten einer KI nachvollziehbar verstanden wurde, ist es möglich, die Ergebnisse einer Überwachung sinnvoll interpretieren zu können.
Die in Kapitel 4.1 dargestellten Ergebnisse des Vergleichs der elf XAI-Softwarebibliotheken (AIX360, Alibi, Captum, iNNvestigate, InterpretML, LIME, Tf-explain, SHAP, Skater, Quantus und Zennit) werden im Folgenden näher betrachtet. Es wird geprüft, inwiefern sie sich hinsichtlich der digitalen Forensik und der Auditierbarkeit eignen. Durch ihr mehrfaches Vorkommen in einschlägiger Literatur wurden die Bibliotheken für den Vergleich ausgewählt. Diese Auswahl erhebt keinen Anspruch auf Vollständigkeit. Die Diskussion orientiert sich an dem Stand der Ergebnisse vom 09.08.2022. Nachträgliche Änderungen der Softwarebibliotheken bleiben unberücksichtigt.

ML-Modelle In TensorFlow erstellte ML-Modelle werden von dem Großteil der Bibliotheken (acht von elf) vollständig unterstützt. Diese weitreichende Unterstützung erleichtert die Kompatibilität zu anderen TensorFlow Projekten, wodurch eine Verwendung empfehlenswert ist. Bei alleiniger Betrachtung der kompatiblen ML-Modelle, wird die Anwendung der Bibliothek LIME empfohlen, da diese als einzige sowohl TensorFlow als auch PyTorch und Scikit-Learn unterstützt.

Trainingsdaten Idealerweise sollte die XAI-Methode nicht auf die Trainingsdaten des ML-Modells zugreifen müssen, da dem Modell häufig sensible Daten zugrunde liegen. Oftmals liegt ausschließlich das fertige Modell vor und die benötigten Trainingsdaten sind schwer zu beschaffen. Daher ist es ratsam die Bibliotheken Captum, InterpretML, Tf-explain, Quantus sowie Zennit anzuwenden.

Visualisierung Sind Visualisierungen der ausgegebenen Erklärungen vorhanden, können sie das Verständnis und die Auswertung seitens der Anwender*innen und seitens von Dritten, beispielsweise dem Gericht, erleichtern. Da jede XAI-Methodik mindestens eine teilweise Unterstützung anbietet, sollte bei der Verwendung auf den konkreten Anwendungsfall sowie die Gruppe der Nutzer*innen geachtet werden. Verschiedene Arten der Visualisierung eignen sich für Gruppen mit unterschiedlichem Fachwissen zur Domäne.

Dokumentation Eine vollständige Dokumentation ist von wesentlicher Bedeutung für eine reibungslose Anwendung des XAI-Frameworks. Die implementierten Methoden und Parameter können ohne Beispielcode und dazugehörige Erklärungen nicht effektiv auf die eigene Problemstellung angepasst werden. Bis auf das Framework Quantus sind alle Bibliotheken hinreichend dokumentiert. Auf GitHub[19] sind lediglich ausgewählte Codebeispiele der Bibliothek Quantus vorhanden, die einen ersten Einblick in die Funktionsweise geben. Ist keine vollständige Dokumentation vorhanden, schließt sich eine Anwendung der Bibliothek im Sinne der Gerichtsverwertbarkeit aus. Die CoC kann ohne eine umfassend beschriebene Dokumentation nicht hinreichend belegt werden, wodurch die Bibliothek nicht als Grundlage des forensischen Gutachtens verwendet werden sollte. Solange keine ausführliche Dokumentation vorliegt, ist demnach eine Verwendung nicht zu empfehlen. Da Quantus im Vergleich zu den anderen Bibliotheken erst im Februar 2022 vorgestellt wurde, ist die zeitnahe Veröffentlichung einer Dokumentation wahrscheinlich. Da neben Quantus auch Tf-explain eine unvollständige Dokumentation aufweist, empfiehlt sich bis zur Erweiterung der Dokumentation vorgegebene Anwendungsbeispiele zu verwenden oder auf eine der anderen neun XAI-Bibliotheken zurückzugreifen.

Beispiele Die Anwendungsbeispiele der XAI-Bibliotheken sind vielfältig. Je mehr Beispiele unterstützt werden, desto flexibler ist die Verwendung der Bibliothek bei eigenen Projekten. Andererseits kann die Konzentration auf eine einzige Art der Datengrundlage tiefgründigere Funktionalitäten und eine umfassendere Community-Unterstützung ergeben. Grundsätzlich sind aber modellagnostische Bibliotheken den Modellspezifischen vorzuziehen.

Metriken Metriken dienen der Evaluationsfähigkeit der XAI-Bibliotheken. Sie geben Auskunft über die Güte der getroffenen Erklärung und sind daher wesentlich bei der Beurteilung des resultierenden Ergebnisses. Im Vergleich zu den anderen Bibliotheken ist Quantus mit der Bereitstellung von über 30 verschiedenen Metriken der Spitzenreiter in der Kategorie Metrik. Im Rahmen forensischer Untersuchungen nehmen sie einen nebensächlichen Stellenwert ein.

Softwareaktualität Der Großteil der Software der XAI-Bibliotheken wird kontinuierlich weiterentwickelt und aktuell gehalten. Diese Schlussfolgerung wurde anhand der Daten der letzten Commits getroffen, da diese weniger als ein Jahr alt sind. Regelmäßig auftretende Commits, trotz älterer Release-Updates bei Skater, LIME und AIX360, weisen auf eine aktive Community-Unterstützung hin. Zukünftige Release-Updates sind nicht auszuschließen. Die kontinuierliche Unterstützung ist bei allen XAI-Bibliotheken erkennbar.

Zusammenfassung Grundsätzlich kann festgehalten werden, dass für die betrachteten Domänen Methoden der Erklärbarkeit existieren, die sich, je nach Modus Operandi der Modellfunktionalität, unterschiedlich eignen. Aufgrund der zuvor diskutierten Aspekte, sind insbesondere die XAI-Bibliotheken Captum, LIME und Quantus empfehlenswert. Obwohl Quantus bisher keine umfangreiche Dokumentation besitzt, werden Funktionen von Captum und Zennit unterstützt. Zusammenfassend sind daher Captum und LIME für den forensischen Gebrauch zu empfehlen.

[19] `https://github.com/understandable-machine-intelligence-lab/quantus/`

Abgeleitete Erkenntnisse bezüglich Auditierbarkeit Die bestehenden XAI-Methoden sind zumeist nicht universell für jeden konkreten Anwendungsfall geeignet. Deswegen wird ein einheitliches Standardverfahren für die Prüfung der Funktionsweise von ML-Modellen benötigt. Dieser Standard muss im Hinblick auf die Datengrundlage folgende Aspekte erfüllen:

- umfangreiche Unterstützung von ML-Modellen
- Schutz von sensiblen Daten
- Bereitstellung von Visualisierungen, die für den Menschen interpretierbar sind
- Vorhandensein einer lückenlosen Dokumentation (CoC)
- modellagnostische Anwendbarkeit
- umfassende Community-Unterstützung
- regelmäßige Aktualisierung der Software
- Integration von Digital Forensic Readiness und Abwehrmaßnahmen für Angriffe auf KI-Systeme

Anwendung auf Continual Learning Die Anwendung von XAI-Methoden auf das im vorangegangenen evaluierte Kapitel 5.1, ist zurzeit nicht umsetzbar. Mehrfache Versuche der Integration von XAI-Softwarebibliotheken in das vorgestellte CL-Modell schlugen fehl. Der Grund dafür war die aktuelle Inkompatibilität mit dem von Avalanche bereitgestellten Eingabedatenstrom. Gängige XAI-Bibliotheken unterstützten lediglich abgeschlossene Datensätze, die klassisch in feste Trainings- und Testdaten eingeteilt werden können. Laufend erweiterbare Datenströme sind demnach nicht kompatibel für die Erstellung von Erklärungen.

5.2 Aufbau und Konfiguration des Continual Learning Modells unter Avalanche

Jupyter Notebook Konfiguration Die Standardkonfiguration von Jupyter Notebook wird hinsichtlich der zu verarbeitenden IOPub-Datenrate von 1 MB auf 10 GB erweitert, um die Visualisierung größerer Datensätze mithilfe der Python-Bibliothek Matplotlib zu ermöglichen. Weiterhin erlaubt diese Erweiterung den fehlerfreien Austausch großer Datenmengen, ohne dass die IOPub-Datenrate beim Ausführen des Programmcodes überschritten wird. Die maximale Speichergröße des RAMs von rund $0,5$ GB (536870912 Bytes) wird außerdem auf 8 GB angehoben, um die Verarbeitungsgeschwindigkeit der auszuführenden Aufgaben zu steigern sowie den vorhandenen Speicherplatz des Arbeitsspeichers effektiv auszunutzen. [86]

Konfiguration der Tensorausgabe Eine verkürzte Ansicht des Gewichtstensors innerhalb der Standardausgabe ist für das Speichern der Gewichtswerte in die Weights-Logdatei nicht praktikabel, da lediglich eine unvollständige Anzahl an Gewichtswerten protokolliert wird. Dieses Verhalten hat einen direkten Einfluss auf die Speichergröße der Logdatei. Sie besitzt eine deutlich reduzierte Speichergröße, die nicht repräsentativ für das Wachstum des Speicherplatzbedarfs ist. Stattdessen wird die vollständige Ausgabe der Tensoren bevorzugt und vor Beginn des ML-Prozesses festgelegt.

Konfiguration der Hidden Layer Insbesondere bei mehrschichtigen Netzwerken ist es möglich, dass sich die Gradienten, also die partiellen Ableitungen der Sigmoid-Verlustfunktion, in kurzer Zeit einem Wert nahe der Null nähern. Daher wird das vergleichsweise längerfristige Training des NN

und somit dessen Leistungsfähigkeit erschwert. Um dieses sogenannte *Vanishing Gradient Problem* zu umgehen, wurde bei der Konfiguration des CL-Modells als Aktivierungsfunktion Rectified Linear Unit (ReLU) verwendet, da diese im Gegensatz zu einer Sigmoidfunktion keine kleinen Ableitungen besitzt. Das Problem kann somit umgangen werden, wodurch ein fehlerfreies Modelltraining begünstigt wird. [87]
Die Verwendung von ReLU kann bei gradientenbasierten Lernmethoden auch das *Exploding Gradient Problem* auslösen. Hierbei treten während des Modelltrainings sehr große Gewichtswerte auf, die ebenfalls große Gradienten verursachen, wodurch ein instabiles Netzwerk entsteht. Der Lerneffekt bei dem Trainingsprozess wird hierbei verhindert. [87]
Beide in der Literatur bekannten Probleme konnten durch die in Kapitel 3.2 beschriebene Konfiguration des SimpleMLP Modells sowie deren HL vermieden werden. Im Falle des explodierenden Gradientenproblems kann sogenanntes `Gradient Clipping` Abhilfe schaffen. [87]

Dropout Die Abbruchfunktion des Moduls `nn.Dropout` des sequentiellen Containers wurde mit $p = 0$ initialisiert, da modellseitig sonst ein Teil der Gewichte auf Null gesetzt wird. Um unabhängig von der Abbruchfunktion die Modellgewichte zu protokollieren, wurde sie deaktiviert.

Evaluation Seitens des CL-Frameworks Avalanche wurde in Kapitel 2.5.3 unter anderem die eigenständige Evaluationsmetrik `WeightCheckpoint` vorgestellt. Obwohl die Metrik in der Theorie sehr gut für das Speichern der Modellgewichte geeignet ist, stellte sich eine Anwendung als ungeeignet heraus. Die in der Dokumentation beschriebene Einbindung in den Programmcode führte trotz intensiver Fehlersuche nicht zu der Ausgabe der Gewichtswerte. Deswegen wurde sich gegen die Verwendung der Metrik entschieden. Da das Framework vergleichsweise jung ist, unterliegt es einem stetigen Verbesserungsprozess. In der Zukunft wäre eine fehlerfreie Verwendung der Metrik daher durchaus denkbar.

5.3 Interne Logging Funktionalität des Continual Learning Modells

Die Funktionsweise des internen Loggings wird zu Beginn anhand der Log-Module von Avalanche beschrieben. Es folgt die Analyse der Gewichtsänderung und der Modell-Laufzeit. Abschließend werden die Ausgaben der Logdateien der Modellgewichte, der Gewichtsinformationen und der Deltas interpretiert.

5.3.1 Konfiguration der Logging-Module von Avalanche

Vorbetrachtungen Grundsätzlich wird die Verwendung einer GPU der Standardeinstellung (CPU) vorgezogen, da die Berechnungsschritte beschleunigt werden. Da jedoch die fehlerfreie Einbindung der Programmierschnittstelle CUDA[20] aufgrund Versionskomplikationen fehlgeschlagen ist, wurde die interne CPU Intel(R) Core(TM) i7-9750H für die nachfolgenden Berechnungen verwendet. Die Verwendung der CPU beeinflusst lediglich die Laufzeit, jedoch nicht den Speicherbedarf beim Logging. Die anschließende Freigabe von 8 GB RAM bewirkte eine Beschleunigung der Laufzeit des Programms.

[20] `https://developer.nvidia.com/cuda-toolkit`

Datensatz Da die MNIST-Sammlung auf realen Daten basiert und bereits vollständig vorverarbeitet und formatiert ist, gestaltet sich die Handhabung bezüglich der Kompatibilität zu eigenen Modellen vergleichsweise einfach. Somit ist der Datensatz für das Erzeugen reproduzierbarer Ergebnisse geeignet und dient im Folgenden als Lerngrundlage für die Erstellung des CL-Modells.

Continual Learning Model Die Ausgabe allgemeiner Informationen zu Modellparametern dient als Grundlage für die Extraktion und Protokollierung der Gewichte.

TextLogger und InteractiveLogger Zu den positiven Aspekten der `TextLogger`-Klasse gehört, dass er metrische Ergebnisse zu verschiedenen Zeitpunkten während des Lernprozesses speichert. Jedoch werden diese Ergebnisse sowohl beim `TextLogger` als auch beim `InteractiveLogger` nicht nach jeder Iteration gedruckt, um die Anzahl an gedruckten Zeilen zu verringern. Beispielsweise werden nicht alle Metrik-Ergebnisse nach jedem Minibatch gespeichert. Lediglich der letzte protokollierte Metrik-Wert am Epochenende wird innerhalb der Zieldatei `log.txt` aufgezeichnet. Weiterhin bietet das Format der Standardausgabe beider Protokollierungsfunktionen keine Unterstützung für Bilder oder komplexe Visualisierungen, die durch manche Metriken erzeugt werden. Für die korrekte Darstellung der Metrikausgabe wird die Verwendung eines weiteren Loggers empfohlen. [68]

TensorboardLogger und WandBLogger Die Verwendung von TensorBoard und W&B ermöglicht interaktive Visualisierungsoptionen nach der Aufnahme von Metrikwerten. Insbesondere werden konfigurierbare Tabellen, Graphen und Bilder in der Anzeige unterstützt. Aufgrund der Möglichkeit der individuellen Anpassung ist die Verwendung der beiden Logging-Optionen dem `TextLogger` und dem `InteractiveLogger` vorzuziehen. Ein weiterer Vorteil des `TensorboardLogger` und des `WandBLogger` ist, dass keine Speicherplatzlimitation, wie bei den anderen Protokollierungsoptionen, besteht. Dadurch können auch große Mengen an Metrikdaten geloggt werden. Weiterhin kann TensorBoard die Gewichtungen in Form von Histogrammen abbilden und unterstützt die Visualisierung der Modellgraphen, speziell die Operationen und die Layer.

Zusammenfassung Eine Zusammenführung der Logger in die Trainings- und Evaluationsschleifen erweist sich als unkompliziert. Es besteht die Möglichkeit, einen eigenen Logger anhand individueller Vorstellungen einfach zu implementieren. In diesem Sinne konnte der `TextLogger` problemlos an bestehende Gegebenheiten angepasst werden. Der Hauptvorteil der Logging-Funktion von Avalanche besteht weiterhin in der Protokollierung der Ergebnisse in Echtzeit, das heißt während der Laufzeit des Experiments [68]. Die interne Logging Funktionalität ist somit ideal für das Protokollieren interner Abläufe geeignet. Für eine umfassende Fehlerdetektion müssen jedoch Erweiterungen vorgenommen werden. Daher bietet sich im nächsten Schritt die Integration von XAI-Softwarebibliotheken an.

5.3.2 Verlauf der Gewichtsänderung

Anhand der in Tabelle 4.2 dargestellten Änderungszeitpunkte der Layer-Werte ist eindeutig ersichtlich, dass sich die Werte nur nach dem Modelltraining und nicht nach der Evaluationsphase ändern. Dieses Verhalten entspricht dem erwarteten Normalverhalten bei der Gewichtsänderung. Das Kleinerwerden der internen Layer-Anpassungen lässt auf eine funktionsfähige Optimierungsfunktion schließen, da eine Annäherung an das Optimum geschieht und die layerinternen Delta-Werte somit tendenziell schrittweise verkleinert werden. Einzelne Ausreißer können hierbei vernachlässigt werden, da innerhalb des Trainingsprozess keine lineare Optimierung der Hyperparameter stattfindet. Daher erfolgt

die Änderung der Modellgewichte ebenfalls nicht linear.
Die Änderung der Delta-Werte zwischen benachbarten Layern wird ebenfalls tendenziell geringer. Dieses Verhalten begründet das Erfassen und Visualisieren der Prozentsätze an Nullen für jeden Eintrag in dem berechneten Delta-Tensor. In Kapitel 5.4.3 folgt eine weiterführende Analyse dieser Untersuchung. Hierbei wird ebenso dargestellt, warum das Identifizieren und Speichern der Nullwerte interessant ist.

5.3.3 Allgemeine Laufzeitbetrachtungen

Da in der Praxis eine regelmäßige Evaluation der Trainingsfortschritte stattfindet, wurde diese, in Form der Berechnung der Accuracy, in den Trainingsprozess integriert. Zunächst wurden die Modellgewichte sowohl vor und nach der Trainingsphase als auch vor und nach der Evaluationsphase gespeichert. Anhand der zuvor diskutierten Ergebnisse der Tabelle 4.2 wurde sich gegen die Speicherung der Gewichtswerte in der Evaluationsphase entschieden. Diese Maßnahme vermeidet Redundanz innerhalb der Logdatei. Die Gewichte wurden daher nur vor und nach dem Modelltraining gespeichert. Das Durchführen einer zusätzlichen Evaluation führt zu höheren Trainings-Laufzeiten. Die Größe der Logdatei wird jedoch nicht beeinflusst. Das komplette Weglassen der Evaluationsphase aufgrund der erhöhten Trainingsdauer würde nicht einer praxisnahen Anwendung entsprechen, weshalb sie beibehalten wurde. Realitätsnahe Anwendungen beinhalten komplexere Evaluationsmetriken als nur die Accuracy. Die Berechnung dieser Werte erhöht die Laufzeit des Programms zusätzlich.

5.3.4 Weights-Logging während des Modelltrainings

Konfiguration Die eindeutige Trennung der Layer Namen zu den Layer Gewichten stellt die Grundlage der Protokollierung dar. Die Namen der Layer dienen der eindeutigen Zuordnung zu den entsprechenden Layer-Gewichten bei dem Loggen der Modellgewichte in die Textdatei. Um die Relevanz der eindeutigen Zuordnung zu verdeutlichen, wird die Anzahl der Gewichte eines Layers (beispielsweise `model.features[0]`) mit dem Befehl `len(model.features[0].weight.data)` ausgegeben. Der resultierende Wert von 512 besagt, dass sich innerhalb des Input-Layers 512 Gewichte befinden, die jeweils eine Anzahl von 784 Einträgen haben (`len(model.features[0].weight.data[0])`). Jeder dieser Gewichtswerte wird innerhalb des Trainingsprozesses optimiert. Die große Anzahl an Gewichtswerten erschwert die Übersicht innerhalb der gespeicherten Logdatei, weshalb eindeutige Erklärungen zu den eigentlichen Werten ergänzt wurden.

Herausforderung bei dem Speicherplatzbedarf Bei der Erstellung der Logdatei für das Modellgewicht, wurde der Speichervorgang zu Beginn häufig vor dem Ende des Modelltrainings abgebrochen, da der lokale Speicherplatz nicht ausreichend war. Erste Analysen haben gezeigt, dass bei der Ausgabe des vollständigen Tensors mehrere Gigabyte an Daten gespeichert werden. Aufgrund dessen wurden 25 TB externer Speicherplatz bereitgestellt, der für die vorliegenden Betrachtungen ausreichend war. Zusätzlich muss beachtet werden, dass bei dem Dateisystem File Allocation Table 32 (FAT32) eine Begrenzung der Dateigröße von maximal 4 GB vorliegt. Da dieser Grenzwert bei der resultierenden Logdatei häufig überschritten wird, empfiehlt sich die Verwendung des neueren Dateisystems New Technology File System (NTFS) von Windows, das eine maximale Speichergröße von 16 TB aufweist. Dieser Größenwert wurde nicht annähernd von der Ausgabedatei erreicht. [88]
Der Speicherplatzbedarf stellt demnach einen starken Begrenzungsfaktor dar.

Daten-Sparsity bei KI Im Bereich der KI wird mithilfe der *Data Sparsity* versucht die Laufzeit eines DNN zu beschleunigen, indem nicht wesentliche Parameterwerte entfernt werden. Dies geschieht mit dem Ziel einer platzsparenden Speicherung von Dateien. Hierbei konnten bei bisherigen Forschungsansätzen bis zu 95% der Gewichtswerte identifiziert werden, die nur einen verschwindend geringen Einfluss auf die Entscheidungsfindung haben und somit vernachlässigbar sind. Jedoch erzielten die bisherigen Versuche der Datensparsamkeit nur im modellspezifischen Kontext Erfolge. Ideal wäre eine modellagnostische Anwendung, um die Sparsity in die Betrachtungen der Auditierbarkeit mit einzubeziehen. [89]
Das Dateisystem NTFS unterstützt *Sparse Files* im Sinne der Datenkompression. Oftmals kann es bei klassischen Festplattenlaufwerken zu einer Erhöhung der Laufzeit kommen. Um dieses Problem zu umgehen, empfiehlt sich eine Verwendung der neueren SSDs. Sparsity kann weiterhin in Kombination mit komprimierten Daten angewendet werden. [90, S. 43 f.]

5.3.5 Information-Logging

Das Loggen der Informationen im Rahmen des Modelltrainings und der Erstellung der Weights-Logdatei dient der schrittweisen Protokollierung der KI-internen Vorgänge in chronologischer Reihenfolge. Um den Schutz vor Angriffen auf KI-Systeme zu gewährleisten, wird eine geprüfte Datengrundlage benötigt. Die Unveränderlichkeit der Daten muss demnach zweifelsfrei nachweisbar sein. Hierzu kann die Datei, welche die sensiblen Daten beinhaltet, gehasht werden, indem die Berechnung eines Hashwertes durchgeführt wird. Anschließend kann dieser Logging-Hash regelmäßig, zu festgelegten Zeitpunkten, mit dem Originalhash verglichen werden. Sollte während dieser Überprüfung eine Abweichung der Hashwerte von dem Originalhash stattfinden, ist mit hinreichender Wahrscheinlichkeit von einer externen Manipulation auszugehen, wodurch die Ergebnisse des KI-Systems nicht mehr als Entscheidungsgrundlage verwendet werden können.
Für die Erstellung der Datei-Hashwerte wurde die Hash-Funktion SHA-256 ausgewählt, da sie im Gegensatz zu MD5 und SHA-1 ungebrochen ist und somit für sicherheitsrelevante Anwendungen verwendet werden kann. [79]
Im Rahmen der vorliegenden Arbeit wurde die Überprüfung auf mögliche Manipulationsversuche nicht vorgenommen und ist daher nicht Bestandteil der nachfolgenden Analysen.

5.3.6 Delta-Logging nach dem Modelltraining

Im Zuge der Erstellung der Delta-Logdatei wurden anhand der Vorgehensweise in Kapitel 3.3.5 zunächst Modifikationen vor dem Speichern der Datei vorgenommen. Diese Modifikationen werden im Folgenden schrittweise erläutert und analysiert.

Modifikationen vor dem Speichern Zu Beginn wurden die durch das Padding der Shapes entstandenen überschüssigen Nullen abgetrennt, um eine Verfälschung der Delta-Werte zu verhindern. Das Ziel der Berechnung der Delta-Werte ist die Identifizierung des Anteils an Nullen in der `delta_weights` Datei. Wenn kein Slicing stattfinden würde, entstehen ab der Stelle 512 in jedem Tensor Nullen. Konkret ergeben sie sich aus der Subtraktion von zwei zuvor hinzugefügten Nullen. Da diese nicht natürlich, also nicht im Laufe des Trainingsprozesses entstanden sind, müssen sie vor dem Abspeichern der Delta-Werte entfernt werden. Dadurch kann Speicherplatz gespart werden und die Verfälschung der Visualisierung wird verhindert. Die ursprünglichen Shapes der Tensoren können nur teilweise wiederhergestellt werden, da von manchen Originalwerten Nullen subtrahiert wurden. Im

Hinblick auf das Modelltraining hat demnach keine Änderung stattgefunden. Diese Werte werden beibehalten und nicht abgeschnitten, obwohl sie mit einer künstlich hinzugefügten Null interagieren. Diese theoretischen Ausführungen werden nachfolgend veranschaulicht.

Veranschaulichung der Modifikationen Um die soeben beschriebenen Überlegungen zu verdeutlichen, werden sie anhand eines vereinfachten Beispiels mit kleineren Shapes visualisiert. Sie dienen als Referenz für die Original-Shapes der Gewicht-Tensoren: $[512,784]$, $[512,512]$, $[512,512]$ und $[10,512]$. Die Shapes der vier Referenzmatrizen $[3,5]$, $[3,3]$, $[3,3]$ und $[1,3]$ weisen unterschiedliche Größen auf, weshalb sie im ersten Schritt mit den blau umrahmten Nullen erweitert werden müssen. Da alle Matrizen nun die gleiche Größe besitzen, können anschließend die Delta-Werte berechnet werden: $\Delta 1 = x_1 - x_2$, $\Delta 2 = x_2 - x_3$, $\Delta 3 = x_3 - x_4$. Die grün umrandeten Delta-Werte weisen keine Veränderung auf, da sie mit Null subtrahiert werden. Diese Werte werden nicht gelöscht. Dahingegen werden die rot umrandeten Deltas mithilfe des Slicings entfernt, da sie aus der Subtraktion zweier Nullen resultieren. Das Beispiel gliedert sich folgendermaßen:

0. Original-Matrizen (Shapes):

$$\underset{[3,5]}{\begin{bmatrix} x_1 & x_1 & x_1 & x_1 & x_1 \\ x_1 & x_1 & x_1 & x_1 & x_1 \\ x_1 & x_1 & x_1 & x_1 & x_1 \end{bmatrix}} \quad \underset{[3,3]}{\begin{bmatrix} x_2 & x_2 & x_2 \\ x_2 & x_2 & x_2 \\ x_2 & x_2 & x_2 \end{bmatrix}} \quad \underset{[3,3]}{\begin{bmatrix} x_3 & x_3 & x_3 \\ x_3 & x_3 & x_3 \\ x_3 & x_3 & x_3 \end{bmatrix}} \quad \underset{[1,3]}{\begin{bmatrix} x_4 & x_4 & x_4 \end{bmatrix}}$$

---

1. Erweitern der Matrix mit Nullen auf Ziel-Shape (Padding):

$$\underset{[3,5]}{\begin{bmatrix} x_1 & x_1 & x_1 & x_1 & x_1 \\ x_1 & x_1 & x_1 & x_1 & x_1 \\ x_1 & x_1 & x_1 & x_1 & x_1 \end{bmatrix}} \quad \underset{[3,5]}{\begin{bmatrix} x_2 & x_2 & x_2 & 0 & 0 \\ x_2 & x_2 & x_2 & 0 & 0 \\ x_2 & x_2 & x_2 & 0 & 0 \end{bmatrix}} \quad \underset{[3,5]}{\begin{bmatrix} x_3 & x_3 & x_3 & 0 & 0 \\ x_3 & x_3 & x_3 & 0 & 0 \\ x_3 & x_3 & x_3 & 0 & 0 \end{bmatrix}} \quad \underset{[3,5]}{\begin{bmatrix} x_4 & x_4 & x_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}$$

Padding Padding Padding

---

2. Deltabildung durch elementweise Subtraktion (Shapes):

$$\underset{[3,5]}{\begin{bmatrix} \Delta 1 & \Delta 1 & \Delta 1 & x_1-0 & x_1-0 \\ \Delta 1 & \Delta 1 & \Delta 1 & x_1-0 & x_1-0 \\ \Delta 1 & \Delta 1 & \Delta 1 & x_1-0 & x_1-0 \end{bmatrix}} \quad \underset{[3,5]}{\begin{bmatrix} \Delta 2 & \Delta 2 & \Delta 2 & 0 & 0 \\ \Delta 2 & \Delta 2 & \Delta 2 & 0 & 0 \\ \Delta 2 & \Delta 2 & \Delta 2 & 0 & 0 \end{bmatrix}} \quad \underset{[3,5]}{\begin{bmatrix} \Delta 3 & \Delta 3 & \Delta 3 & 0 & 0 \\ x_3-0 & x_3-0 & x_3-0 & 0 & 0 \\ x_3-0 & x_3-0 & x_3-0 & 0 & 0 \end{bmatrix}}$$

keine Veränderung Δ Nullen keine Veränderung Δ Nullen

---

3. Abtrennen der überschüssigen Delta-Nullen (Slicing) vor dem Speichern der Delta-Werte (Shapes):

$$\underset{[3,2]}{\begin{bmatrix} \Delta 1 & \Delta 1 & \Delta 1 & x_1 & x_1 \\ \Delta 1 & \Delta 1 & \Delta 1 & x_1 & x_1 \\ \Delta 1 & \Delta 1 & \Delta 1 & x_1 & x_1 \end{bmatrix}} \quad \underset{[3,3]}{\begin{bmatrix} \Delta 2 & \Delta 2 & \Delta 2 \\ \Delta 2 & \Delta 2 & \Delta 2 \\ \Delta 2 & \Delta 2 & \Delta 2 \end{bmatrix}} \quad \underset{[3,3]}{\begin{bmatrix} \Delta 3 & \Delta 3 & \Delta 3 \\ x_3 & x_3 & x_3 \\ x_3 & x_3 & x_3 \end{bmatrix}}$$

5.4 Visualisierung der Abhängigkeiten zwischen Modellkonfiguration, Trainingsgeschwindigkeit und Speicherplatzbedarf

Im Rahmen der nachfolgenden Analyse der Visualisierungen werden Herausforderungen und sich daraus resultierende Anpassungen der Umgebungseinstellungen dargelegt. Weiterhin wird der vorliegende Funktionsverlauf mit den zuvor getroffenen Erwartungen verglichen und es wird diskutiert, wie sich diese Abhängigkeiten auf die Anwendbarkeit von Methoden der Auditierbarkeit auswirken. Dabei wird auch die Laufzeit des Modelltrainings berücksichtigt.

5.4.1 Speichergröße der Weights-Logdatei vs. Trainingszyklus

Auswertung Visualisierung Die Kurvenverläufe der Ergebnisvisualisierungen in Kapitel 4.4.1 entsprechen dem erwarteten linearen Verlauf der Funktion. Es liegt somit eine annähernd konstante Änderungsrate vor. Mit zunehmender Trainingszeit wächst die Speichergröße in gleichen Abständen an. Der Funktionsgraph weist demnach die Form einer Geraden auf. Dieses monotone Wachstum ist in allen betrachteten Modellen gleich. Sie unterscheiden sich lediglich in der Größe der Ausgabedatei sowie der Laufzeit t. Je größer die Anzahl der HL und Trainingszyklen pro Modell ist, desto größer gestaltet sich die resultierende Weights-Logdatei. Im Hinblick auf die Speichergröße stellt die Anzahl der Trainingszyklen somit einen limitierenden Faktor des Speicherplatzbedarfs dar.

Laufzeit Das grundlegende Ansteigen der Laufzeiten mit größer werdender HL-Anzahl entspricht dem erwarteten Verhalten. Auffällig ist, dass die Laufzeiten der fünf Modelltrainings bei Experience null nicht ebenfalls bei Sekunde null beginnen. Der Grund dafür ist, dass vor der ersten Experience die Konfigurationen des Modelltrainings abgerufen werden müssen. Gleichzeitig werden die Modellgewichte vor dem Training dokumentiert. Das Speichern der Gewichte vor Beginn der ersten Experience dauert länger, je größer die Anzahl der HL in dem Modell SimpleMLP ist. Entgegen der Erwartung stimmt das Ende der letzten Experience weiterhin nicht mit dem eigentlichen Ende des Modelltrainings überein. Beispielsweise ist bei dem Modell mit einem HL bei Experience 5 eine Laufzeit von 15.07 Minuten zu verzeichnen. Anhand der Ergebnisse in Tabelle 4.3 dauerte das Modelltraining insgesamt jedoch 15.35 Minuten. Während dieser Differenz von 0.28 min fand die Evaluation nach der Training-Experience, inklusive Berechnung der Accuracy-Metrik, statt. Diese Differenz erhöht sich, je größer das betrachtete Modell ist, da sich der Evaluationsaufwand mit steigender HL-Zahl vergrößert.

Kompressionsmöglichkeit Bei dem Auftreten großer Datenmengen besteht die Möglichkeit Ausgabedateien im Rahmen einer effektiven Speicherung zu komprimieren. Das funktioniert beispielsweise mithilfe der Python Bibliothek `pickle`. Jedoch kann der Verlauf der Erstellung der Gewichtswerte nicht in komprimierter Form erfolgen, da die Informationen durch Kompressionsschritte für den Menschen nicht mehr lesbar sind. Demnach wäre es notwendig, im Zuge der Nachverfolgung der Trainingsschritte, die Ausgabedateien zu dekomprimieren und deren Ausgangszustand wiederherzustellen. Hierbei besteht die Gefahr einer fehlerbehafteten Dekompression. Daran anschließend muss die Unveränderlichkeit der Inhalte zweifelsfrei gewährleistet werden. Hierfür bietet sich die Verwendung des Hashwertvergleichs vor und nach Kompressions- und Dekompressionschritten an.

5.4.2 Speichergröße der Weights-Logdatei vs. Anzahl der Neuronen

Anpassung der Spyder-Konfiguration Im Rahmen der Erstellung der Visualisierungen wurde zu Beginn die standardmäßige Länge der `buffer_size` von 500 auf 100.000 `lines` erhöht, da sie die Funktionsfähigkeit bei komplexeren Operationen erheblich einschränkt.

Herausforderungen Bei der Durchführung des Modelltrainings für verschiedene Neuronenanzahlen ist bei einer vergleichsweise großen Zahl ein Memory Error bei der Evaluation der letzten Training Experience aufgetreten. Dieses Verhalten wurde bei fünf HL mit jeweils 3072 Neuronen beobachtet. Um den vorzeitigen Abbruch des Trainings zu verhindern, wurde auf das regelmäßige Speichern der Hashwerte verzichtet, welches bei der Codierung eine große Auslastung des RAMs verursacht hat. Dadurch wurde mehr interner RAM-Speicher für die Evaluation geschaffen. Die Änderung der Konfiguration zeigt sich ebenfalls in der Ausprägung der Laufzeit, welche deutlich geringer ist, als bei 2560 Neuronen. Diese Laufzeitanomalie zeigt sich ebenfalls bei 640 und 768 Neuronen des Modells mit 100 HL. Da der Fokus jedoch auf dem Verlauf des Speicherplatzbedarfs liegt, können diese Abweichungen vernachlässigt werden.

Vorliegender Funktionsverlauf Bei näherer Betrachtung der Visualisierungen in Kapitel 4.4.2 ist bei einem HL ein lineares Wachstum des Speicherplatzbedarfs erkennbar. Da der Definitionsbereich der Funktion jedoch beschränkt ist, können ohne weitere Betrachtungsweisen keine zuverlässigen Aussagen über den möglichen Verlauf, bei größer werdender Anzahl von Neuronen, getroffen werden. Eine mögliche Erklärung des Verhaltens ist, dass bei einem exponentiellen Wachstum zu Beginn nur kleine Veränderungen in der Höhe des Speicherplatzbedarfs auftreten, welche sich bei weiterführender Betrachtung erheblich erhöhen würden. Andererseits liegt ein weiterführendes lineares Wachstum aufgrund der geringen Anzahl an Hidden Layern ebenfalls im Bereich des Möglichen. Die verbleibenden Visualisierungen weisen eine leichte, parabelähnliche Krümmung des Kurvenverlaufs auf, wodurch weder ein eindeutig linearer noch ein exponentieller Funktionsverlauf vorliegt. Um mit hinreichender Wahrscheinlichkeit Aussagen über den zukünftigen Modellverlauf treffen zu können, sind mehr Messwerte notwendig. Eine Erstellung von mehr als sechs Messwerten, war im Rahmen der Arbeit nicht realisierbar.

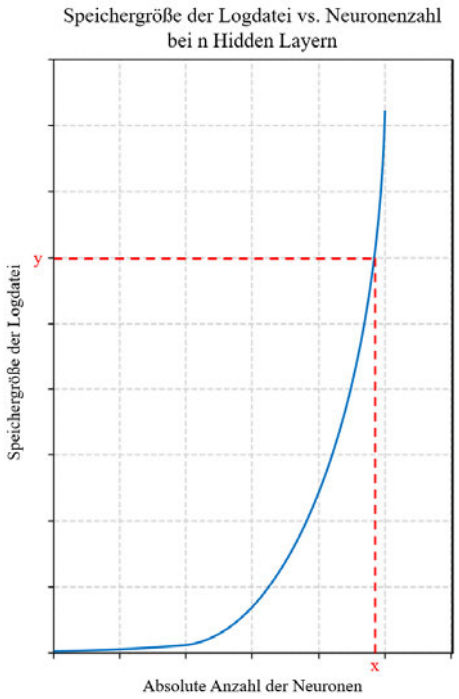


Abbildung 5.1: Die erwartete Größe der Weights-Logdatei ist in Abhängigkeit zur Anzahl der Neuronen bei n HL dargestellt.

Erwarteter Funktionsverlauf Der Verlauf des Graphen in Abbildung 5.1 zeigt den exponentiellen Charakter des Funktionsgraphen, welcher zu Beginn der Betrachtungen in ähnlicher Ausprägung vorhergesagt worden war. Der nach dem Modelltraining entstandene, parabelförmige Verlauf der Abbildungen 4.22 bis 4.25 deckt sich nur teilweise mit dem in Abbildung 5.1. Der typische Anstieg des Speicherbedarfs einer exponentiellen Funktion liegt jedoch nicht vor. Daher stimmt die Erwartungshaltung nicht mit dem resultierenden Funktionsgraphen nach dem Trainingsprozess überein.

Erklärung und Hypothese Bei einem erwarteten exponentiellen Kurvenverlauf des Funktionsgraphen würde mit größer werdender Neuronenzahl der Speicherplatzverbrauch stark ansteigen. Bei der Betrachtung des steigenden Speicherplatzbedarfs gilt es herauszufinden, bis zu welcher Anzahl an Neuronen die Speichergröße der Logdatei noch praktikabel ist. In dieser Hinsicht wirkt sich ein exponentielles Wachstum negativ auf die praktische Anwendung aus, da oftmals große NN mit einer hohen Anzahl von Neuronen benötigt werden. Obwohl mittlerweile große Datenmengen gespeichert werden können, unterliegen Datenbanken, beispielsweise in Form von *Data Warehouses*, ebenfalls Limitationen bezüglich des vorhandenen Speicherplatzes. Es müsste eine Art Schwellwert (x, y), vergleichbar zu Abbildung 5.1, identifiziert werden, um das Aufkommen an zu großen Speichermengen zu verhindern.

Laufzeit Das unerwartete zeitliche Verhalten bei einem HL kann durch eine mögliche Auslastung interner Ressourcen bezüglich der Rechenleistung erklärt werden. Da sich die Laufzeit zum Großteil ähnlich zu dem Speicherverhalten verhält, ist das Speichern der Modellgewichte bei großen Neuronenanzahlen in Kombination mit mehr als fünf HL nicht zu empfehlen. Ideal wäre ein logarithmischer Verlauf, der den Wachstum an Speicherplatzbedarf begrenzt. Gegenüber dem annähernd quadratischen Wachstum wäre alternativ auch ein lineares Wachstum der Laufzeit im Hinblick auf größere Neuronen und HL-Anzahlen vorteilhafter.

5.4.3 Anteil an Nullen in der Delta-Logdatei vs. Trainingszyklus

Intension des Delta-Logging In Abbildung 5.2 ist der Verlauf der Gewichtswerte nach der abgeschlossenen Trainingsphase verdeutlicht. Die Delta-Werte der Modellgewichte werden im Hinblick auf den Graphen während des Modelltrainings tendenziell kleiner. Hierbei ist weiterhin erkennbar, dass mit steigender Trainingsiteration die Werte Δw_i gegen Null konvergieren. Die rot gekennzeichnete Linie verdeutlicht sinnbildlich die Stelle x, ab dem die Delta-Werte der Modellgewichte die Grenze von 0.0000 erreichen. Ab diesem Punkt finden keine signifikanten Änderungen im Zuge der Optimierung statt. Das heißt die Gewichtswerte, die ab hier gespeichert werden, sind für die forensische Erfassung des Delta-Verlaufs nicht relevant. Sie müssen nicht weiter in der Weights-Logdatei protokolliert werden. Daher ist es wichtig diesen Zeitpunkt innerhalb des Modelltrainings zu identifizieren, um unnötigen Analyseaufwand sowie Speicherverbrauch zu vermeiden.

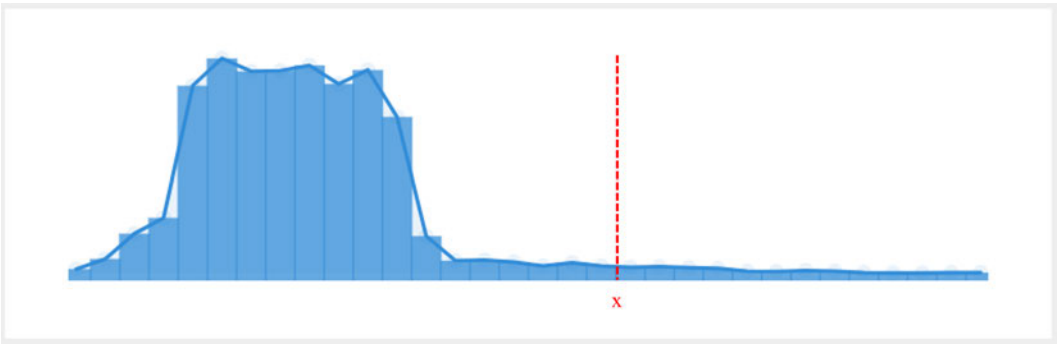


Abbildung 5.2: Schematischer Gewichtsverlauf nach dem Modelltraining

Auf Grundlage dieser Erkenntnisse wird anschließend der Prozentsatz an Nullen analysiert, um den Anteil der Nullwerte in der Delta-Logdatei zu ermitteln.

Vorliegender Funktionsverlauf Werden lediglich die Ergebnisse der fünf Visualisierungen in Kapitel 4.4.1 betrachtet, ist erkennbar, dass alle Graphen auf der Ebene von $x = 0.0$ verlaufen. Das impliziert kein Vorkommen an Delta-Null Werten in der Ausgabedatei. Werden jedoch die Funktionsverläufe mit einer kleineren Einteilung der Ordinate betrachtet, ist ein minimaler, konstanter Anstieg des Prozentsatzes an vorkommenden Delta-Nullen erkennbar. Dieser ist jedoch sehr geringfügig ausgeprägt. Das vereinzelte Auftreten der Nullen verläuft nach keinem konkreten Muster, es verhält sich zufällig. Demnach kann deren Auftreten vernachlässigt werden. Der maximale Prozentsatz von 1.0 wird von keinem der betrachteten Modelle erreicht.

Erwarteter Funktionsverlauf Die Abbildung 5.3 zeigt den erwarteten Verlauf des Funktionsgraphen bei der Gegenüberstellung von Prozentsatz und dem aktuellen Trainingszyklus. Zu Beginn des Modelltrainings verläuft der Funktionsgraph bis zum Zeitpunkt a bei einem Prozentsatz von 0.0 beziehungsweise nahe der Null. Dieses Verhalten ist vergleichbar mit den Ergebnisverläufen aus den Abbildungen 4.26 bis 4.30. Im Gegensatz zu diesen erfolgt bei dem erwarteten Verlauf ein monotoner Anstieg, der sich dem maximalen Prozentwert von 1.0 annähert, bis er zum Zeitpunkt b während des Modelltrainings die 1.0 erreicht und dieses Niveau bis zum Ende hält.

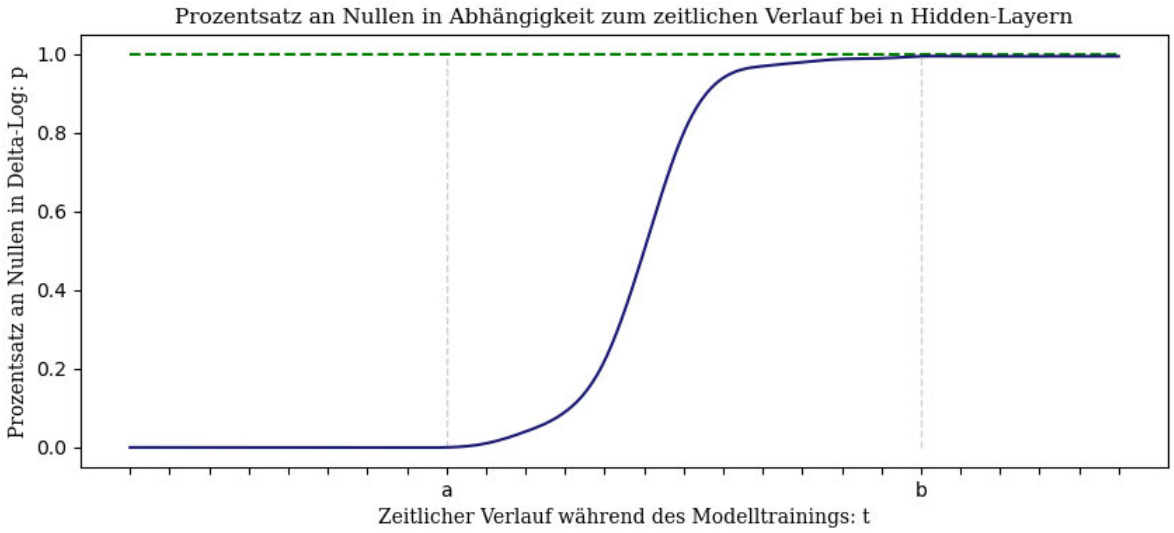


Abbildung 5.3: Es wird der Prozentsatz p an Nullen in Abhängigkeit zum zeitlichen Verlauf t während des Modelltrainings betrachtet. a und b kennzeichnen wichtige Stellen auf der Abszisse.

Zu Beginn des Modelltrainings gleicht der Kurvenverlauf im Wertebereich $W_f = [0,a]$ der Nullfunktion $f(x) = 0$. Ab dem Zeitpunkt a ($W_f = [a,y]$) ist der Kurvenverlauf vergleichbar zu dem logistischen Wachstum in der Funktion 5.1. b kennzeichnet die Stelle, an der $y = 1.0$ erreicht ist. Ab hier verläuft der Graph konstant bei $f(x) = 1$. Der Definitionsbereich ist wie folgt festgelegt: $\mathbb{D}_f = [0,1]$.

$$f(x) = \frac{1}{1+e^{-(x-5)}} \tag{5.1}$$

Erklärung Eine Null nach der Deltabildung bedeutet, dass keine Optimierung des Modellgewichtes stattgefunden hat. Daher sind beim Start des Modelltrainings erwartungsweise noch keine oder nur sehr wenige Delta-Nullen vorhanden. Wird das kumulierte Auftreten solcher Nullen gespeichert, ist anzunehmen, dass ein monotoner Anstieg erfolgt, bis irgendwann 100 % der neuen Delta-Werte

0.0000 sind. Zu diesem Zeitpunkt *b* findet keine Änderung der Gewichte mehr statt. Das heißt, dass das Modelltraining abgeschlossen ist, da die Hyperparameter optimal für die vorliegende Trainingsaufgabe eingestellt sind. Da während des Trainingsprozesses jede Hyperparameter-Kombination getestet wird, ist es sehr wahrscheinlich, dass die optimale Kombination nicht erst am Ende gefunden wird. Somit wird das Testen der nachfolgenden Kombinationen, wie eingangs beschrieben, redundant.

Hypothese Sobald der Zeitpunkt *b* in der Theorie erreicht wird, werden erwartungsgemäß in dem Ausgabe-Log nur noch Nullen gespeichert. Um den verbrauchten Speicherplatz beim Weights-Logging zu reduzieren, empfiehlt sich alle Werte (hier Null-Werte) nach *x* aus der Datei zu entfernen. Sie werden im Weiteren nicht benötigt, um den Verlauf der Gewichtsänderung zu rekonstruieren.

Ausblick Es besteht die Möglichkeit, dass bei der Vergrößerung der Anzahl an Hidden Layern in Kombination mit den Neuronen das erwartete Verhalten auftritt. Um diese Erwartung zu beweisen, müsste der Versuchsaufbau mit mehr Messwerten, das heißt höheren HL- und Neuronenzahlen, durchgeführt werden. Im Hinblick auf den bisherigen Verlauf des Speicherbedarfs entstehen mit hinreichender Wahrscheinlichkeit immens große Ausgabedateien im zweistelligen Gigabyte-Bereich. Deswegen war eine weiterführende Analyse im Rahmen der Arbeit nicht möglich.

5.5 Schlussfolgerungen zur Anwendbarkeit der Logging-Funktionalität und anderer Forensic Readiness Maßnahmen im Anwendungskontext autonomes Fahren

Autonomes Fahren Der Anwendungsbereich des autonomen Fahrens ist einer der bekanntesten Vertreter im Bereich des Continual Learning und der autonomen Systeme. Daher beziehen sich die folgenden Ausführungen speziell auf dieses Anwendungsgebiet. Analog gelten die Erkenntnisse jedoch auch für andere Bereiche des CL, wie beispielsweise bei Empfehlungssystemen oder im Bereich der medizinischen Diagnose. In der Praxis ist es von enormer Relevanz rechtzeitig jegliche Art der Manipulation am Auto nachzuweisen, um hochkritischen Konsequenzen, wie beispielweise Autounfällen, vorzubeugen und die Ausfallsicherheit des Systems zu gewährleisten. Um im Falle von Fehlverhalten seitens der KI auf die lückenlose Beweissicherung der Forensik vorbereitet zu sein, müssen bereits im Vorfeld Maßnahmen der DFR getroffen werden. Zu diesen Methodiken gehört beispielsweise das in den vorangegangenen Kapiteln beschriebene Protokollieren der Gewichte des NN.

Praktische Anwendung von DFR Maßnahmen Nachfolgend wird ein fiktiver Anwendungsfall betrachtet. Angenommen ein sehr einfaches autonomes System basiert auf einem CL-Modell mit 50 HL und 512 Neuronen pro Layer. Dann ergibt sich eine resultierende Trainingszeit von 98.42 Minuten (Vergleich Tabelle 4.3). In dieser Zeit entsteht ein Speicheraufwand der Modellgewichte von 3.89 GB. Da autonome Systeme jedoch einem immerwährenden Lernprozess unterliegen, würden in der Theorie am Tag $56,92$ GB an Daten anfallen. Wird nur dieses Szenario betrachtet, wäre der berechnete Speicherplatzverbrauch tragbar. Da das System jedoch nicht fertig trainiert ist, sondern fortlaufend aus neuen Eingabedaten lernt, entsteht gleichermaßen ein weiterführender Speicherplatzverbrauch über die 24 h hinaus. Weiterhin basiert das betrachtete Szenario auf einer sehr einfach konstruierten Konfiguration. In der Praxis treten deutlich komplexere Architekturen von DNN auf, die aufgrund höherer

HL- und Neuronenzahlen größere Ausgabe-Logdateien erzeugen würden. Dieses Szenario betrachtet lediglich den Speicherbedarf eines einzigen Systems. Da derzeit eine breite Anwendung autonomer Systeme umgesetzt wird und dieser Aufwand bezogen auf autonome Fahrsysteme erwartungsgemäß in der Zukunft steigt, ist der errechnete Mindestaufwand in der Praxis nicht realisierbar. Außerdem ist zu beachten, dass bei vollständig autonomen Systemen eine Vielzahl an Umwelteinflüssen von dem System parallel erkannt und verarbeitet werden müssen.

Herausforderungen von DFR Die gewonnenen Logging-Daten müssen für nachträgliche forensische Untersuchungen zur Verfügung stehen. Daher werden sie auf begrenzte Zeit gespeichert. Hierbei entstehen zwei Anwendungsszenarien. Einerseits können die Daten pro autonom fahrendem System so lange gespeichert werden, bis kein freier Platz mehr zur Verfügung steht und die ältesten Daten überschrieben werden. Diese Sichtweise würde einen immensen Speicheraufwand bedeuten, welcher im Hinblick auf IT-forensische Untersuchungen jedoch von Vorteil ist. Auf der anderen Seite besteht die Möglichkeit, dass alle Daten nach einem festgelegten Zeitraum, bei Aufzeichnungen von Sicherheitkameras beispielsweise nach drei Tagen, vom Server gelöscht werden. Hierbei besteht ein deutlich begrenzter Sicherungszeitraum für die IT-Forensiker*innen. Daher muss eine Abwägung der Machbarkeit im Sinne des Speicherplatzverbrauchs und der Begünstigung von DFR-Maßnahmen getroffen werden. Zusammenfassend kann festgestellt werden, dass die Anwendung von DFR-Maßnahmen umso schwieriger ist, je komplexer das autonome System ist.
Neben der eingangs ausgeführten Problematik des katastrophalen Vergessens wird in der Literatur aufgezeigt, dass eine korrekte Unterscheidung zwischen einer normalen Fehlfunktion des Systems oder eines absichtlichen Kompromittierungsversuch nicht trivial ist. Werden demnach Angriffe als Systemfehler missinterpretiert, werden keine DFR-Maßnahmen eingeleitet und die Erkennung des Angriffs verzögert sich. Die Ausfallsicherheit des Systems kann daher nicht umfassend gewährleistet werden. Dadurch verringert sich die Akzeptanz im Rahmen der Anwendung von beispielsweise autonomen Fahrsystemen. Da jedoch gleichzeitig ein hohes Potenzial für den Einsatz dieser CL-Systeme existiert, müssen die bestehenden Herausforderungen zukünftig mit neuen Technologien begegnet werden.

Zusammenfassung der Nachweismöglichkeiten von Manipulationen an KI-Systemen Zu den DFR-Maßnahmen gehört die frühzeitige Manipulationserkennung bei KI-Systemen im Angriffsfall. Hierbei ist keine allgemeingültige Vorgehensweise möglich, da die Erkennung domänenabhängig (Bild, NLP, autonome Systeme) ist.
Im Hinblick auf das Logging der Gewichts- und Delta-Werte können bei der Durchführung mehrerer Runs mit gleicher Konfiguration Abweichungen auftreten. Diese können durch die Veränderung der gespeicherten Hashwerte erkannt werden. Der direkte Vergleich der Hashwerte zweier Runs ermöglicht weiterhin die Lokalisierung der Veränderung innerhalb des Lernprozesses. Hierbei stellt sich die Frage, ob lediglich eine abweichende Anpassung der Hyperparameter im Zuge der Backpropagation stattgefunden hat, oder ob eine Änderung durch unberechtigte Dritte aufgetreten ist.
Diese Überlegung kann mithilfe der Überprüfung auf die Zugangsberechtigung beantwortet werden. Weiterhin deutet ein gleichbleibender Hashwert des Trainings- und Testdatensatzes auf eine unveränderte Datengrundlage hin. Angriffsspezifische Maßnahmen der Erkennung und Verhinderung von externem Zugriff wurden in Kapitel 2.2.3 eingehend präsentiert. Außerdem können angewendete Methoden der Explainable Artificial Intelligence das Verständnis der Entscheidungsfindung begünstigen. Eine Kombination dieser Methodiken ist in der Praxis empfehlenswert, um die IT-forensische Untersuchung zu unterstützen.

6 Fazit

Erklärbare Künstliche Intelligenz Die eingangs betrachteten Methoden der Erklärbaren KI stellen eine Grundlage für die zukünftige Auditierung von KI-Systemen bereit. Hierbei muss der Fokus vermehrt von der lokalen zur globalen Erklärbarkeit verschoben werden. Derzeit liefern XAI-Methoden entweder markante Merkmale innerhalb des Datenraums (lokal) oder sie identifizieren für das ML-Modell empfindliche Datenmerkmale (global). Am verständlichsten sind solche Erklärungen bei bildgestützten Anwendungen, da diese Domäne kein tiefergehendes Fachwissen zur Interpretation der Ergebnisse erfordert. Andere Domänen, wie zum Beispiel Textanalyse oder Autonome Systeme, setzen spezielles Wissen voraus, sodass die Erklärungen für den Menschen nicht interpretierbar und somit nicht informativ sind. [91]
Aktuell gibt es keine universellen Lösungsansätze für XAI-Verfahren. Es existieren eine Reihe von XAI-Softwarebibliotheken, die zumeist modellspezifisch konzipiert sind. Obwohl ebenfalls modellagnostische Ansätze bestehen, muss eine standardisierte Lösung geschaffen werden, um dem Anspruch der Auditierbarkeit und der DFR gerecht zu werden. Verschiedene Typen von Nutzer*innen benötigen unterschiedliche Arten von Erklärungen, um diese in ihrem Bereich optimal anwenden zu können.

Continual Learning Das CL-Framework Avalanche ist für einfache Simulationen bei Bildklassifikationsaufgaben geeignet, bedarf jedoch einer weiterführenden Optimierung der Funktionalität hinsichtlich einer modellagnostischen Anwendung. Grundlegend ist eine Auditierbarkeit bei abgeschlossenen Systemen leichter realisierbar als bei kontinuierlich lernenden Modellen.

Auditierbarkeit Im Hinblick auf die steigende Anwendung von KI-Systemen, sowohl in der deutschen Wirtschaft als auch weltweit, ist es von Bedeutung, die Art und Weise der Entscheidungsfindung innerhalb der Black-Box KI zu überprüfen. Das zukünftige Etablieren standardisierter Audits schafft eine Basis des Vertrauens. Das Ziel der vorliegenden Masterarbeit war daher die Untersuchung einer praktischen Auditiermethode im Anwendungskontext von kontinuierlich lernenden Systemen. Hierfür ist es nicht ausreichend allein die Modellgewichte in einer Logdatei zu speichern, um deren Verlauf überprüfbar zu machen. Die Vorhersage des Speicherplatzbedarfs sowie der Trainingslaufzeit gestaltet sich schwierig, da sie abhängig von der zugrundeliegenden Architektur sowie von der Art und Weise der Initialisierung ist.

Problemstellung Obwohl eine umfassende Auditierbarkeit im Hinblick auf den praktischen Teil der Arbeit nicht hinreichend umgesetzt werden konnte, ist eine eindeutige Problemstellung sichtbar geworden: Die Maßnahmen der Digital Forensic Readiness können nicht einfach auf Künstliche Intelligenz angewendet werden, da sie derzeit nicht modellagnostisch anwendbar sind. Das alleinige Speichern der Modellgewichte ist aufgrund des immensen Speicheraufwandes und der nicht ausschließlich gewährleisteten Manipulationssicherheit nicht ausreichend für die DFR. Es entsteht somit ein Zielkonflikt zwischen der angestrebten Auditierbarkeit auf der einen Seite und der fehlenden Umsetzbarkeit im Anwendungskontext KI auf der anderen Seite.

7 Ausblick

Neben den vorausgegangenen Betrachtungen existieren Ansatzpunkte für Erklärbarkeit und Auditierbarkeit, die in dieser Arbeit nur am Rande betrachtet wurden. Derzeit gibt es zukunftsweisende Projekte und Strategien, bei denen nach Ende ihrer Laufzeit eine eingehende Analyse empfehlenswert ist. Nachfolgend werden vielversprechende Ansätze kurz vorgestellt.

Erklärbarkeit Damit auch die Erklärungen von XAI-Methoden für Anwender*innen interpretierbar sind, werden zukünftig Lösungsansätze für XAI auf der Ebene des Datensatzes sowie der Modellebene kombiniert. Dies ermöglicht es den Nutzer*innen, die Aufgabe interner Modellkomponenten während der Entscheidungsanalyse zu ermitteln. Weiterhin sollen die Methoden der Erklärbarkeit (lokal und global) zukünftig weniger Expertenwissen benötigen, um die Domäne des Datensatzes sowie die Art und Weise der Modellfunktionalität zu erfassen. Mithilfe von semantische Ergänzungen können sie für den Menschen verständlich gemacht werden. In Zukunft muss dahingehend ein Wendepunkt bei der Interaktion zwischen Mensch und KI geschehen, um den Weg zur Auditierbarkeit zu ebnen. [91]
Ausgehend von der Literaturrecherche ist außerdem zu empfehlen, dass Methoden der post-hoc Erklärbarkeit und der direkt implementierten Erklärbarkeit (ante-hoc) verbunden werden, um eine umfassendere Erklärung der KI zu ermöglichen. [36]
Die neue XAI-Methode SpRAy verspricht die Verbindung von lokalen und globalen Erklärungsansätzen für KI-Systeme. Die Kombination ist vielversprechend für eine weiterführende Anwendung im Bereich der Auditierbarkeit, deren Ziel unter anderem in einer umfassenden globalen Erklärbarkeit liegt. [48]

Zukünftige Arbeit am Framework Avalanche Die Recherche zu ML-Techniken weist in den vergangenen zehn Jahren signifikante Bemühungen auf, um Transparenz, Reproduzierbarkeit und einen offenen Zugang bereitzustellen. Oftmals entsteht die Herausforderung, bereits vorhandene Softwarelösung in die eigene Arbeitsumgebung einzupassen und auszuführen. Diese Gesichtspunkte versucht Avalanche zu meistern. Mit der aktuellen Version Alpha liegt das Hauptaugenmerk auf der Bildverarbeitung, dessen Lernfortschritt kontinuierlich überwacht werden soll. In Zukunft sollen zusätzliche Lernparadigmen, Aufgabentypen und Anwendungskontexte integriert werden. Dies soll mithilfe einer unterstützenden Community geschehen. Damit eine effektive Zusammenarbeit gewährleistet werden kann, muss eine Vergrößerung der Community angestrebt werden. [67]

Auditierbarkeit Zukünftige Forschung sollte sich der Problemstellung widmen, wie ein Trade-off zwischen den Zielen der DFR bei KI und deren praktische Umsetzung realisiert werden kann.

Zukunftsprojekte Aktuell existieren laufende Projekte, die Methoden zur Auditierung von KI-Systemen versprechen. Beispielsweise läuft das „Flagship-Projekt Zertifizierte KI“ der Projektpartner Fraunhofer IAIS, BSI, DIN e.V., et al. seit 03/2021 bis 02/2026. Die Zielstellung dieses Projekts ist die Entwicklung und Standardisierung von „Prüfkriterien, -methoden und -werkzeugen für KI-Systeme“, um das Anwendungsvertrauen zu verstärken. [6]
Weiterhin wurde ein neues Projekt „AI Forensics: Accountability through Interpretability in Visual AI Systems“ der Forschungsgruppe Gender/Diversity in Informatics Systems (GeDIS) vorgestellt, welches ab 2022 für drei Jahre durch die Volkswagenstiftung gefördert wird. Das Ziel ist die Erstellung von open-source Werkzeugen der IT-Forensik zur Untersuchung der sozialen Auswirkungen bei der

Verwendung von realen KI-Systemen. Hierbei stehen u. a. die Interpretierbarkeit sowie die Rechenschaftspflicht von KI-Systemen im Vordergrund der Untersuchung. [92]
Das Verbundprojekt Vertrauenswürdige Künstliche Intelligenz für polizeiliche Anwendungen (VIKING) basiert auf der Bekanntmachung des Bundesministeriums für Bildung und Forschung im Mai 2021 zur Thematik „Künstliche Intelligenz in der zivilen Sicherheitsforschung“. Das von 2018 bis 2023 laufende Forschungsprojekt agiert bundesweit mit dem Ziel die Arbeit der polizeilichen Ermittler effektiv zu unterstützen, um den „Schutz vor Kriminalität und Terrorismus“ gewährleisten zu können. Ein interessantes Teilvorhaben stellt das von 2022 bis 2024 laufende Projekt „Erklärbarkeit vertrauenswürdiger KI-Sprachmodelle für den transparenten Gebrauch bei Sicherheitsbehörden zur Textklassifikation“ dar. Im Fokus liegt hierbei der Aspekt der Vertrauenswürdigkeit bei der Anwendung von KI-Systemen zur Textklassifikation im polizeilichen Umfeld. [93] [94]
Vielversprechend ist außerdem die Evaluation der Ergebnisse des laufenden Projekts „Security Testing of AI“ Securing Artificial Intelligence (SAI). Das Institut European Telecommunications Standards Institute (ETSI) ermittelt hierbei Methodiken zur Sicherheitsprüfung von KI-Systemen. [95]

Artificial Intelligence Act Das Gesetz AIA der Europäischen Union soll sich zukünftig zum globalen Standard im Bereich der Sicherheit und Auditierbarkeit von KI-Anwendungen etablieren. Die EU hat somit als erste große Regulierungsbehörde einen Schritt in Richtung der Auditierbarkeit von intelligenten Systemen getan. Hierfür muss gewährleistet werden, dass das Gesetz flexibel auf zukünftige Gefahrenpotenziale, besonders bei *„high risk“* und *„unacceptable“* Einstufungen, reagieren kann. [96]

Wenn diese Projekte in Zukunft hinreichend analysiert werden, können die Ansätze dieser Arbeit als Grundlage für weiterführende Forschung dienen, mit dem Ziel den Trade-off zwischen Auditierbarkeit und Machbarkeit zufriedenstellend zu bewältigen.

Anhang A: Programmcode Continual Learning und Logging

A.1 Continual Learning Strategy: Naive

Der Programmcode wurde von der Avalanche Dokumentation[21] übernommen.

```
class Naive(SupervisedTemplate):
   def __init__(
       self,
       model: Module,
       optimizer: Optimizer,
       criterion=CrossEntropyLoss(),
       train_mb_size: int = 1,
       train_epochs: int = 1,
       eval_mb_size: Optional[int] = None,
       device=None,
       plugins: Optional[List[SupervisedPlugin]] = None,
       evaluator: EvaluationPlugin = default_evaluator,
       eval_every=-1,
       **base_kwargs
    ):
    super().__init__(
           model,
           optimizer,
           criterion,
           train_mb_size=train_mb_size,
           train_epochs=train_epochs,
           eval_mb_size=eval_mb_size,
           device=device,
           plugins=plugins,
           evaluator=evaluator,
           eval_every=eval_every,
           **base_kwargs
       )
```

[21] https://avalanche-api.continualai.org/en/v0.2.1/_modules/avalanche/training/supervised/strategy_wrappers.html#Naive

A.2 TextLogger

Der Programmcode wurde von der Avalanche Dokumentation[22] übernommen und modifiziert.

```
if TYPE_CHECKING:
    from avalanche.training.templates import SupervisedTemplate

UNSUPPORTED_TYPES: Tuple[Type] = (TensorImage)

class TextLogger(BaseLogger, SupervisedPlugin):

    def __init__(self, file=sys.stdout):
        super().__init__()
        self.file = file
        self.metric_vals = {}

    def log_single_metric(self, name, value, x_plot) -> None:
        self.metric_vals[name] = (name, x_plot, value)

    def _val_to_str(self, m_val):
        if isinstance(m_val, torch.Tensor):
            return "\n" + str(m_val)
        elif isinstance(m_val, float):
            return f"{m_val:.4f}"
        else:
            return str(m_val)

    def print_current_metrics(self):
        sorted_vals = sorted(self.metric_vals.values(), key=lambda x: x[0])
        for name, x, val in sorted_vals:
            if isinstance(val, UNSUPPORTED_TYPES):
                continue
            val = self._val_to_str(val)
            print(f"\t{name} = {val}", file=self.file, flush=True)

    def before_training_exp(self, strategy: "SupervisedTemplate", metric_values:
        List["MetricValue"], **kwargs):
        super().before_training_exp(strategy, metric_values, **kwargs)
        self._on_exp_start(strategy)

    def before_eval_exp(self, strategy: "SupervisedTemplate", metric_values:
        List["MetricValue"], **kwargs):
        super().before_eval_exp(strategy, metric_values, **kwargs)
        self._on_exp_start(strategy)

    def after_training_epoch(self, strategy: "SupervisedTemplate", metric_values:
        List["MetricValue"], **kwargs):
        super().after_training_epoch(strategy, metric_values, **kwargs)
        print(f"Epoche {strategy.clock.train_exp_epochs} endet.", file=self.file,
            flush=True)
```

[22] https://avalanche-api.continualai.org/en/v0.2.1/_modules/avalanche/logging/text_logging.html#TextLogger

```
        self.print_current_metrics()
        self.metric_vals = {}

    def after_eval_exp(self, strategy: "SupervisedTemplate", metric_values:
        List["MetricValue"], **kwargs):
        super().after_eval_exp(strategy, metric_values, **kwargs)
        exp_id = strategy.experience.current_experience
        task_id = phase_and_task(strategy)[1]
        if task_id is None:
            print(f"-- >> Evaluation der Experience {exp_id} des
                {stream_type(strategy.experience)} Datenstroms endet. << --",
                file=self.file, flush=True)
        else:
            print(f"-- >> Evaluation der Experience {exp_id} (Task {task_id}) des
                {stream_type(strategy.experience)} Datenstroms endet. << --",
                file=self.file, flush=True)
        self.print_current_metrics()
        self.metric_vals = {}

    def before_training(self, strategy: "SupervisedTemplate", metric_values:
        List["MetricValue"], **kwargs):
        super().before_training(strategy, metric_values, **kwargs)
        with open(r"weights_log_nhl.txt", "a+") as f1,
            open(r"info_weights_log_nhl.txt", "a+") as f2:
            print("-- >> Beginn der Trainingsphase << --", file=self.file,
                flush=True)
            for key, value in log_dict.items():
                begin_train = f"-- >> Beginn der Trainingsphase für die
                    Gewichtswerte des Layers {key} << --"
                f2.write("%s\n" % begin_train)
                a = f"Gewichtswerte des Layers {key} ({value.shape}) vor der
                    Trainingsphase:\n{value}"
                f1.write("%s\n" % a)
        f1.close()
        f2.close()
        with open(r"weights_log_nhl.txt", "r+") as f1,
            open(r"info_weights_log_nhl.txt", "a+") as f2:
            file_size = os.path.getsize(r"E:\Logs\SimpleMLP-nHL\weights_log_nhl.txt")
            time_checkpoints.append(time.time() - start)
            log_size_extended.append(file_size)
            b = f"Größe der Logdatei: {file_size} Bytes"
            f2.write("%s\n" % b)
            content = f1.read().encode("utf-8")
            sha256_hash = hashlib.sha256()
            sha256_hash.update(content)
            hash_info1 = f"{sha256_hash.name}: {sha256_hash.hexdigest()}"
            f2.write("%s\n" % hash_info1)
            hash_values.append(sha256_hash.hexdigest())
        f1.close()
        f2.close()

    def before_eval(self, strategy: "SupervisedTemplate", metric_values:
        List["MetricValue"], **kwargs):
```

```
84          super().before_eval(strategy, metric_values, **kwargs)
85          with open(r"weights_log_nhl.txt", "a+") as f1,
                open(r"info_weights_log_nhl.txt", "a+") as f2:
86              print("-- >> Beginn der Evaluationsphase << --", file=self.file,
                    flush=True)
87              begin_eval = "-- >> Beginn der Evaluationsphase << --"
88              f2.write("%s\n" % begin_eval)
89          f1.close()
90          f2.close()
91          with open(r"weights_log_nhl.txt", "r+") as f1,
                open(r"info_weights_log_nhl.txt", "a+") as f2:
92              file_size = os.path.getsize(r"E:\Logs\SimpleMLP-nHL\weights_log_nhl.txt")
93              time_checkpoints.append(time.time() - start)
94              log_size_extended.append(file_size)
95              d = f"Größe der Logdatei: {file_size} Bytes"
96              f2.write("%s\n" % d)
97              content = f1.read().encode("utf-8")
98              sha256_hash = hashlib.sha256()
99              sha256_hash.update(content)
100             hash_info2 = f"{sha256_hash.name}: {sha256_hash.hexdigest()}"
101             f2.write("%s\n" % hash_info2)
102             hash_values.append(sha256_hash.hexdigest())
103         f1.close()
104         f2.close()
105
106     def after_training(self, strategy: "SupervisedTemplate", metric_values:
            List["MetricValue"], **kwargs):
107         super().after_training(strategy, metric_values, **kwargs)
108         with open(r"weights_log_nhl.txt", "a+") as f1,
                open(r"info_weights_log_nhl.txt", "a+") as f2:
109             print("-- >> Ende der Trainingsphase << --", file=self.file, flush=True)
110             for key, value in log_dict.items():
111                 end_train = f"-- >> Ende der Trainingsphase für die Gewichtswerte
                        des Layers {key} << --"
112                 f2.write("%s\n" % end_train)
113                 e = f"Gewichtswerte des Layers {key} ({value.shape}) nach der
                        Trainingsphase:\n{value}"
114                 f1.write("%s\n" % e)
115         f1.close()
116         f2.close()
117         with open(r"weights_log_nhl.txt", "r+") as f1,
                open(r"info_weights_log_nhl.txt", "a+") as f2:
118             file_size = os.path.getsize(r"E:\Logs\SimpleMLP-nHL\weights_log_nhl.txt")
119             time_checkpoints.append(time.time() - start)
120             log_size_extended.append(file_size)
121             f = f"Größe der Logdatei: {file_size} Bytes"
122             f2.write("%s\n" % f)
123             content = f1.read().encode("utf-8")
124             sha256_hash = hashlib.sha256()
125             sha256_hash.update(content)
126             hash_info3 = f"{sha256_hash.name}: {sha256_hash.hexdigest()}"
127             f2.write("%s\n" % hash_info3)
128             hash_values.append(sha256_hash.hexdigest())
```

```
129         f1.close()
130         f2.close()
131
132     def after_eval(self, strategy: "SupervisedTemplate", metric_values:
            List["MetricValue"], **kwargs):
133         super().after_eval(strategy, metric_values, **kwargs)
134         with open(r"weights_log_nhl.txt", "a+") as f1,
                open(r"info_weights_log_nhl.txt", "a+") as f2:
135             print("-- >> Ende der Evaluationsphase << --", file=self.file,
                    flush=True)
136             end_eval = "-- >> Ende der Evaluationsphase << --"
137             f2.write("%s\n" % end_eval)
138             self.print_current_metrics()
139             self.metric_vals = {}
140         f1.close()
141         f2.close()
142         with open(r"weights_log_nhl.txt", "r+") as f1,
                open(r"info_weights_log_nhl.txt", "a+") as f2:
143             file_size = os.path.getsize(r"E:\Logs\SimpleMLP-nHL\weights_log_nhl.txt")
144             time_checkpoints.append(time.time() - start)
145             log_size_extended.append(file_size)
146             h = f"Größe der Logdatei: {file_size} Bytes"
147             f2.write("%s\n" % h)
148             content = f1.read().encode("utf-8")
149             sha256_hash = hashlib.sha256()
150             sha256_hash.update(content)
151             hash_info4 = f"{sha256_hash.name}: {sha256_hash.hexdigest()}"
152             f2.write("%s\n" % hash_info4)
153             hash_values.append(sha256_hash.hexdigest())
154         f1.close()
155         f2.close()
156
157     def _on_exp_start(self, strategy: "SupervisedTemplate"):
158         action_name = "Training" if strategy.is_training else "Evaluation"
159         exp_id = strategy.experience.current_experience
160         task_id = phase_and_task(strategy)[1]
161         stream = stream_type(strategy.experience)
162         if task_id is None:
163             print("-- >> Starte {} der Experience {} des {} Datenstroms <<
                    --".format(action_name, exp_id, stream), file=self.file, flush=True)
164         else:
165             print("-- >> Starte {} der Experience {} (Task {}) des {} Datenstroms <<
                    --".format(action_name, exp_id, task_id, stream), file=self.file,
                    flush=True)
```

A.3 InteractiveLogger

Der Programmcode wurde von der Avalanche Dokumentation[23] übernommen.

```
if TYPE_CHECKING:
    from avalanche.training.templates.supervised import SupervisedTemplate

class InteractiveLogger(TextLogger, SupervisedPlugin):

    def __init__(self):
        super().__init__(file=sys.stdout)
        self._pbar = None
        self.last_length = 0

    def before_training_epoch(self, strategy: "SupervisedTemplate", metric_values:
        List["MetricValue"], **kwargs):
        super().before_training_epoch(strategy, metric_values, **kwargs)
        self._progress.total = len(strategy.dataloader)

    def after_training_epoch(self, strategy: "SupervisedTemplate", metric_values:
        List["MetricValue"], **kwargs):
        self._end_progress()
        super().after_training_epoch(strategy, metric_values, **kwargs)

    def before_training_exp(self, strategy: "SupervisedTemplate", metric_values:
        List["MetricValue"], **kwargs):
        if isinstance(strategy.experience, OnlineCLExperience):
            experience = strategy.experience.logging()
            if experience.is_first_subexp:
                super().before_training_exp(strategy, metric_values, **kwargs)
                self._progress.total = \
                    experience.sub_stream_length * \
                    strategy.train_passes * \
                    (experience.subexp_size // strategy.train_mb_size)
                self.last_length = self._progress.total

    def after_training_exp(self, strategy: "SupervisedTemplate", metric_values:
        List["MetricValue"], **kwargs):
        if isinstance(strategy.experience, OnlineCLExperience):
            experience = strategy.experience.logging()
            if experience.is_last_subexp:
                self._end_progress()
                super().after_training_exp(strategy, metric_values, **kwargs)

    def before_eval_exp(self, strategy: "SupervisedTemplate", metric_values:
        List["MetricValue"], **kwargs):
        super().before_eval_exp(strategy, metric_values, **kwargs)
        self._progress.total = len(strategy.dataloader)

```

[23] https://avalanche-api.continualai.org/en/v0.2.1/_modules/avalanche/logging/interactive_logging.html

```
41      def after_eval_exp(self, strategy: "SupervisedTemplate", metric_values:
            List["MetricValue"], **kwargs):
42          self._end_progress()
43          super().after_eval_exp(strategy, metric_values, **kwargs)
44
45      def after_training_iteration(self, strategy: "SupervisedTemplate",
            metric_values: List["MetricValue"], **kwargs):
46          self._progress.update()
47          self._progress.refresh()
48          super().after_training_iteration(strategy, metric_values, **kwargs)
49
50      def after_eval_iteration(self, strategy: "SupervisedTemplate", metric_values:
            List["MetricValue"], **kwargs):
51          self._progress.update()
52          self._progress.refresh()
53          super().after_eval_iteration(strategy, metric_values, **kwargs)
54
55      @property
56      def _progress(self):
57          if self._pbar is None:
58              self._pbar = tqdm(leave=True, position=0, file=sys.stdout)
59          return self._pbar
60
61      def _end_progress(self):
62          if self._pbar is not None:
63              self._pbar.close()
64              self._pbar = None
```

A.4 TensorboardLogger

Der Programmcode wurde von der Avalanche Dokumentation[24] übernommen und modifiziert.

```
class TensorboardLogger(BaseLogger):

    def __init__(self, tb_log_dir: Union[str, Path] =
        f"./tb_data/{datetime.now():%Y-%m-%d}", filename_suffix: str = "",):
        super().__init__()
        tb_log_dir = _make_path_if_local(tb_log_dir)
        self.writer = SummaryWriter(tb_log_dir, filename_suffix=filename_suffix)
        weakref.finalize(self, SummaryWriter.close, self.writer)

    def log_single_metric(self, name, value, x_plot):
        if isinstance(value, AlternativeValues):
            value = value.best_supported_value(Image, Tensor, TensorImage, Figure,
                float, int)

        if isinstance(value, Figure):
            self.writer.add_figure(name, value, global_step=x_plot)

        elif isinstance(value, Image):
            self.writer.add_image(name, to_tensor(value), global_step=x_plot)

        elif isinstance(value, Tensor):
            self.writer.add_histogram(name, value, global_step=x_plot)

        elif isinstance(value, (float, int)):
            self.writer.add_scalar(name, value, global_step=x_plot)

        elif isinstance(value, TensorImage):
            self.writer.add_image(name, value.image, global_step=x_plot)

def _make_path_if_local(tb_log_dir: Union[str, Path]) -> Union[str, Path]:
    if isinstance(tb_log_dir, str) and _is_aws_or_gcloud_path(tb_log_dir):
        return tb_log_dir

    tb_log_dir = Path(tb_log_dir)
    tb_log_dir.mkdir(parents=True, exist_ok=True)
    return tb_log_dir

def _is_aws_or_gcloud_path(tb_log_dir: str) -> bool:
    return tb_log_dir.startswith("gs://") or tb_log_dir.startswith("s3://")

__all__ = ["TensorboardLogger"]
```

[24] https://avalanche-api.continualai.org/en/v0.2.1/_modules/avalanche/logging/tensorboard_logger.html

A.5 WandBLogger

Der Programmcode wurde von der Avalanche Dokumentation[25] übernommen und modifiziert.

```
if TYPE_CHECKING:
    from avalanche.evaluation.metric_results import MetricValue
    from avalanche.training.templates.supervised import SupervisedTemplate

class WandBLogger(BaseLogger, SupervisedPlugin):

    def __init__(
        self,
        project_name: str = "Avalanche-CL",
        run_name: str = "SimpleMLP-5HL",
        log_artifacts: bool = True,
        path: Union[str, Path] = "Checkpoints",
        uri: str = None,
        sync_tfboard: bool = False,
        save_code: bool = True,
        config: object = None,
        dir: Union[str, Path] = None,
        params: dict = None):

        super().__init__()
        self.import_wandb()
        self.project_name = project_name
        self.run_name = run_name
        self.log_artifacts = log_artifacts
        self.path = path
        self.uri = uri
        self.sync_tfboard = sync_tfboard
        self.save_code = save_code
        self.config = config
        self.dir = dir
        self.params = params
        self.args_parse()
        self.before_run()
        self.step = 0
        self.exp_count = 0

    def import_wandb(self):
        try:
            import wandb
        except ImportError:
            raise ImportError('Bitte führen Sie "pip install wandb" aus, um wandb zu
                installieren.')
        self.wandb = wandb

    def args_parse(self):
        self.init_kwargs = {
```

[25] https://avalanche-api.continualai.org/en/v0.2.1/_modules/avalanche/logging/wandb_logger.html

```
            "project": self.project_name,
            "name": self.run_name,
            "sync_tensorboard": self.sync_tfboard,
            "dir": self.dir,
            "save_code": self.save_code,
            "config": self.config,
        }
        if self.params:
            self.init_kwargs.update(self.params)

    def before_run(self):
        if self.wandb is None:
            self.import_wandb()
        if self.init_kwargs:
            self.wandb.init(**self.init_kwargs)
        else:
            self.wandb.init()
        self.wandb.run._label(repo="Avalanche")

    def after_training_exp(self, strategy: "SupervisedTemplate", metric_values:
        List["MetricValue"], **kwargs):
        for val in metric_values:
            self.log_metrics([val])

        self.wandb.log({"TrainingExperience": self.exp_count}, step=self.step)
        self.exp_count += 1

    def log_single_metric(self, name, value, x_plot):
        self.step = x_plot

        if isinstance(value, AlternativeValues):
            value = value.best_supported_value(Image, Tensor, TensorImage, Figure,
                float, int, self.wandb.viz.CustomChart)

        if not isinstance(value, (Image, Tensor, Figure, float, int,
            self.wandb.viz.CustomChart)):
            return

        if isinstance(value, Image):
            self.wandb.log({name: self.wandb.Image(value)}, step=self.step)

        elif isinstance(value, Tensor):
            value = np.histogram(value.view(-1).numpy())
            self.wandb.log({name: self.wandb.Histogram(np_histogram=value)},
                step=self.step)

        elif isinstance(value, (float, int, Figure, self.wandb.viz.CustomChart)):
            self.wandb.log({name: value}, step=self.step)

        elif isinstance(value, TensorImage):
            self.wandb.log({name: self.wandb.Image(array(value))}, step=self.step)

        elif name.startswith("WeightCheckpoint"):
```

```
            if self.log_artifacts:
                cwd = os.getcwd()
                ckpt = os.path.join(cwd, self.path)
                try:
                    os.makedirs(ckpt)
                except OSError as e:
                    if e.errno != errno.EEXIST:
                        raise
                suffix = ".pth"
                dir_name = os.path.join(ckpt, name + suffix)
                artifact_name = os.path.join("Models", name + suffix)
                if isinstance(value, Tensor):
                    torch.save(value, dir_name)
                    name = os.path.splittext(self.checkpoint)
                    artifact = self.wandb.Artifact(name, type="model")
                    artifact.add_file(dir_name, name=artifact_name)
                    self.wandb.run.log_artifact(artifact)
                    if self.uri is not None:
                        artifact.add_reference(self.uri, name=artifact_name)

__all__ = ["WandBLogger"]
```

Anhang B: Arten der Darstellung des Continual Learning Modells

B.1 Graphische Modelldarstellung in TensorBoard

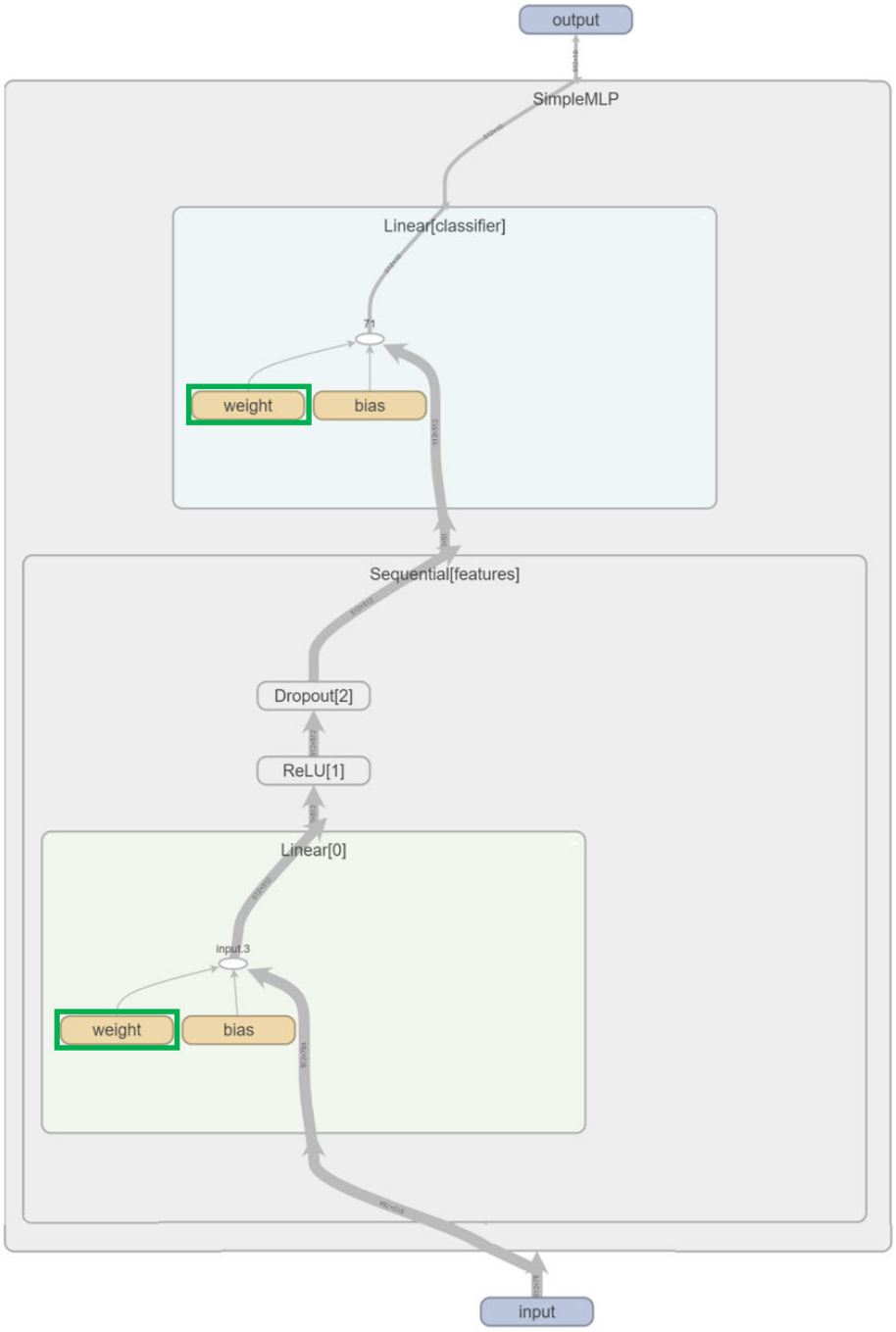
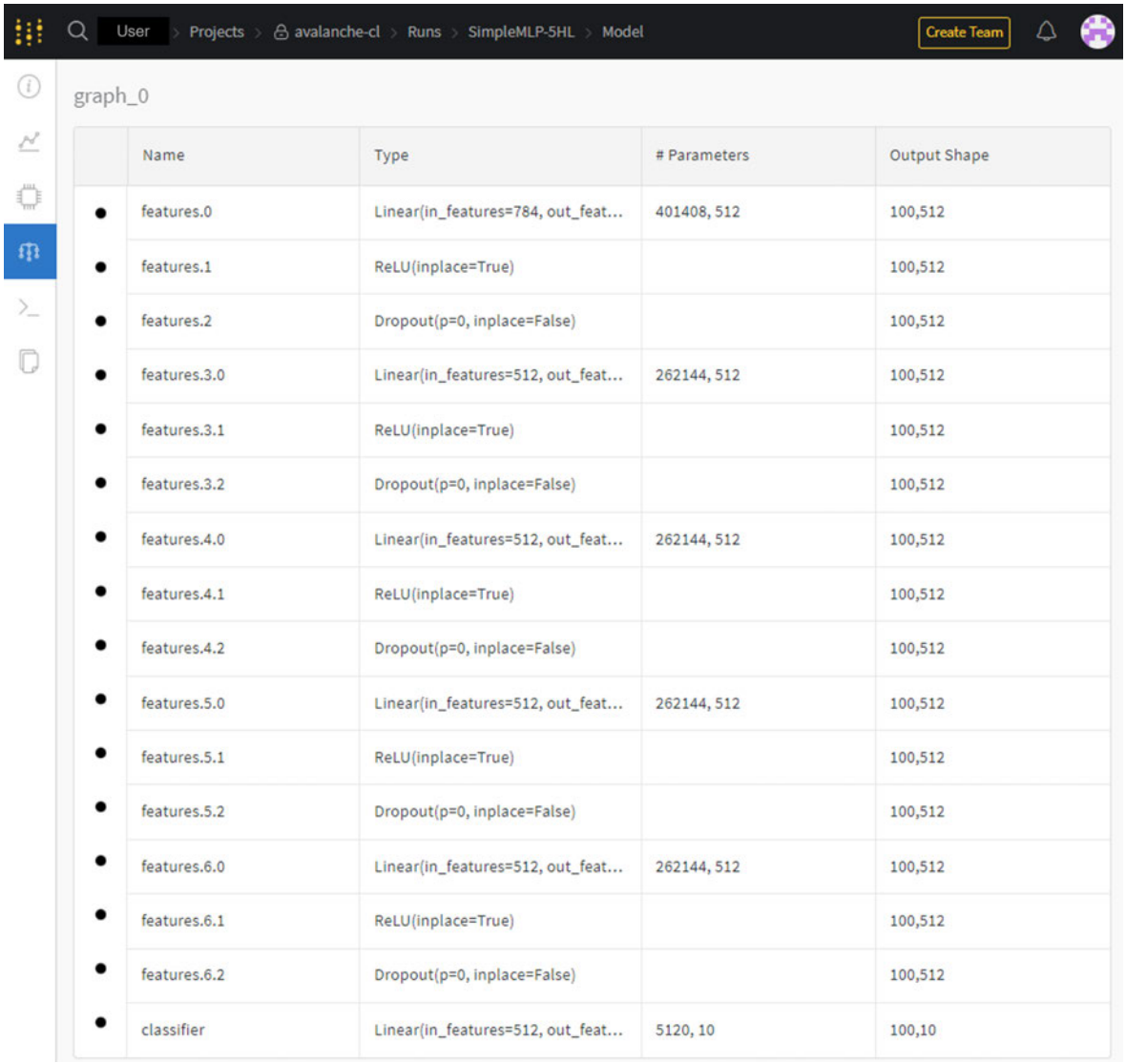


Abbildung B.1: Sichtbar ist die interne Struktur des konstruierten CL-Modells in TensorBoard mithilfe des Modell-Graphen aus dem `GraphExplorer` von TensorBoard. Zur graphischen Veranschaulichung wurden die Modellgewichte der Layer `Linear[0]` und `Linear[classifier]` grün umrandet.

B.2 Tabellarische Modelldarstellung in Weights & Biases



	Name	Type	# Parameters	Output Shape
●	features.0	Linear(in_features=784, out_feat...	401408, 512	100,512
●	features.1	ReLU(inplace=True)		100,512
●	features.2	Dropout(p=0, inplace=False)		100,512
●	features.3.0	Linear(in_features=512, out_feat...	262144, 512	100,512
●	features.3.1	ReLU(inplace=True)		100,512
●	features.3.2	Dropout(p=0, inplace=False)		100,512
●	features.4.0	Linear(in_features=512, out_feat...	262144, 512	100,512
●	features.4.1	ReLU(inplace=True)		100,512
●	features.4.2	Dropout(p=0, inplace=False)		100,512
●	features.5.0	Linear(in_features=512, out_feat...	262144, 512	100,512
●	features.5.1	ReLU(inplace=True)		100,512
●	features.5.2	Dropout(p=0, inplace=False)		100,512
●	features.6.0	Linear(in_features=512, out_feat...	262144, 512	100,512
●	features.6.1	ReLU(inplace=True)		100,512
●	features.6.2	Dropout(p=0, inplace=False)		100,512
●	classifier	Linear(in_features=512, out_feat...	5120, 10	100,10

Abbildung B.2: Die Darstellung des CL-Modells in W&B ist tabellarisch organisiert und zeigt die internen Komponenten eines SimpleMLP Modells mit fünf HL. Die Spalten sind in Name, Typ, Anzahl der Parameter und dem Output Shape unterteilt, wobei die Zeilen anhand dieser Unterscheidung alle Modellbestandteile verdeutlichen.

Anhang C: Information Logging während des Modelltrainings

C.1 Information-Logdatei

info_weights_log_1hl.txt

```
TRAININGSSCHLEIFE BEGINNT
Beginn der Experience: 0
Aktuelle Klasse der Experience: [0, 1]
-- >> Beginn der Trainingsphase für die Gewichtswerte des Layers model.features[0] << --
-- >> Beginn der Trainingsphase für die Gewichtswerte des Layers model.classifier << --
Größe der Logdatei: 6099778 Bytes
sha256: f415eac061e4dd9b974dfb30bd9afe32ca162f03deaaf1cf6dab8726a92796a7
-- >> Beginn der Trainingsphase für die Gewichtswerte des Layers model.features[0] << --
-- >> Beginn der Trainingsphase für die Gewichtswerte des Layers model.classifier << --
Größe der Logdatei: 12199490 Bytes
sha256: 246e519d8bd2e0a77d87f94053dac6b468975965fb2df21146863e6b85b4a36a
-- >> Beginn der Evaluationsphase << --
Größe der Logdatei: 12199531 Bytes
sha256: 9c98c632036a9bd64e015cefb303733b4ca78ee717d57b8bf48046390149f7bf
-- >> Beginn der Evaluationsphase << --
Größe der Logdatei: 12199531 Bytes
sha256: 9c98c632036a9bd64e015cefb303733b4ca78ee717d57b8bf48046390149f7bf
-- >> Ende der Evaluationsphase << --
Größe der Logdatei: 12199791 Bytes
sha256: 7da102441551e0237d78168cfea5fc7a94fbd08ccef030a4c99145c4347e513c
-- >> Ende der Evaluationsphase << --
Größe der Logdatei: 12199791 Bytes
sha256: 7da102441551e0237d78168cfea5fc7a94fbd08ccef030a4c99145c4347e513c
-- >> Beginn der Evaluationsphase << --
Größe der Logdatei: 12199994 Bytes
sha256: 805bfd6ce7002395dc4b8cf847b5f80b8d931ae2ca142654759ff3a8d825e94f
-- >> Beginn der Evaluationsphase << --
Größe der Logdatei: 12199994 Bytes
sha256: 805bfd6ce7002395dc4b8cf847b5f80b8d931ae2ca142654759ff3a8d825e94f
-- >> Ende der Evaluationsphase << --
Größe der Logdatei: 12200254 Bytes
sha256: b5bad1d8289d97d028e39722d264f3ce4bb0f2233d9f09d3db4920ae8c14a348
-- >> Ende der Evaluationsphase << --
Größe der Logdatei: 12200254 Bytes
sha256: b5bad1d8289d97d028e39722d264f3ce4bb0f2233d9f09d3db4920ae8c14a348
-- >> Ende der Trainingsphase für die Gewichtswerte des Layers model.features[0] << --
-- >> Ende der Trainingsphase für die Gewichtswerte des Layers model.classifier << --
Größe der Logdatei: 18300005 Bytes
sha256: 23931959566561760880f3f472841f8d68388f2f44907a2335026670d6b9db85
-- >> Ende der Trainingsphase für die Gewichtswerte des Layers model.features[0] << --
-- >> Ende der Trainingsphase für die Gewichtswerte des Layers model.classifier << --
Größe der Logdatei: 24399719 Bytes
sha256: a58b4033dec37acfa91909b8649b75c4d4e1550053f7a995fa150a7b52a7cee3
-- >> Beginn der Evaluationsphase << --
Größe der Logdatei: 24399760 Bytes
sha256: 3f7c34b7f1965b46519ffa371d9032ff76f46523e68c101bc2d20abedd2c1723
-- >> Beginn der Evaluationsphase << --
Größe der Logdatei: 24399760 Bytes
sha256: 3f7c34b7f1965b46519ffa371d9032ff76f46523e68c101bc2d20abedd2c1723
-- >> Ende der Evaluationsphase << --
Größe der Logdatei: 24400889 Bytes
sha256: b284b83209c2985053352868ee6ae88b810a70917fa13385a441acc0adc8c4f6
-- >> Ende der Evaluationsphase << --
Größe der Logdatei: 24400889 Bytes
sha256: b284b83209c2985053352868ee6ae88b810a70917fa13385a441acc0adc8c4f6
Die Metrik Accuracy Werte nach der Experience 0 betragen:
{'Top1_Acc_Exp/eval_phase/train_stream/Task000/Exp000': 0.9859380262714719, 'Top1_Acc_
```

Abbildung C.1: Die Logdatei beinhaltet Informationen, welche die Weights-Logdatei ergänzen. Zu den zusätzlichen Informationen gehört die Größe der Logdatei nach jeder Trainings- und Evaluationsphase. Gleichermaßen wird ein Hashwert der Weights-Logdatei gespeichert und die Accuracy ausgegeben. Die relevanten Informationen sind grün umrandet. Abgebildet sind lediglich die geloggten Informationen von Beginn der Trainingsschleife bis Ende der ersten Experience des CL-Modells SimpleMLP mit einem HL.

C.2 Dokumentation der Hashwerte

```
Anzahl der Einträge in hash_values: 80
Hashwerte nach dem Start des Modelltrainings:

['48ce201df6ee5989f05b0acd6d78a181288ccd4841a6c0753cf68e07e207e1bd',   Beginn Trainingsphase
 '564fae6036175245220482b4b00e437f0eab38eeea4e40b2729cf43aee43dcf6',
 '5ba92be49b7066a1da3ccc906f893216e9c19456903eea6af9692ddca7b01dfb',   Beginn Evaluationsphase
 '5ba92be49b7066a1da3ccc906f893216e9c19456903eea6af9692ddca7b01dfb',
 'a4d7aa64002d8bc4974556e1c1477e175c4ae16a31315c8770b0c1f6e6d18a41',   Ende Evaluationsphase
 'a4d7aa64002d8bc4974556e1c1477e175c4ae16a31315c8770b0c1f6e6d18a41',
 '1bb95f6fc1903b9c5085f7d653e8002e1835a9175afeae6c6a2c5eaf4cca2c5e',   Beginn Evaluationsphase
 '1bb95f6fc1903b9c5085f7d653e8002e1835a9175afeae6c6a2c5eaf4cca2c5e',
 '203f124bbd3fb4ba5fee8cf99941e579c5a8c2931d5c770a50820b8756f2212a',   Ende Evaluationsphase
 '203f124bbd3fb4ba5fee8cf99941e579c5a8c2931d5c770a50820b8756f2212a',
 'dead627908cb0b32fedb66e15307bbeb81b726c9f61c3b66dd2459f7e2a2a627',   Ende Trainingsphase
 '619b25119266e6f9fcf9a8e898d70140bc7eaf78141c9e93c4f06e6bd8ea238e',
 '595add40b176087a353862afa6959cddb97685735fed856724c537390a415188',   Beginn Evaluationsphase
 '595add40b176087a353862afa6959cddb97685735fed856724c537390a415188',
 '4d151fd01c9edbfb50c3af54c89340ff48263640c59b3d62a11c02bf9abff1e8',   Ende Evaluationsphase
 '4d151fd01c9edbfb50c3af54c89340ff48263640c59b3d62a11c02bf9abff1e8',
```

Abbildung C.2: Der Ausschnitt der Konsolenausgabe zeigt die ersten 16 gespeicherten Hashwerte der Experience 0 der Information-Logdatei des Modells SimpleMLP-5HL.

Anhang D: Visualisierung der Speichergröße bei erweiterter Trainingszeit

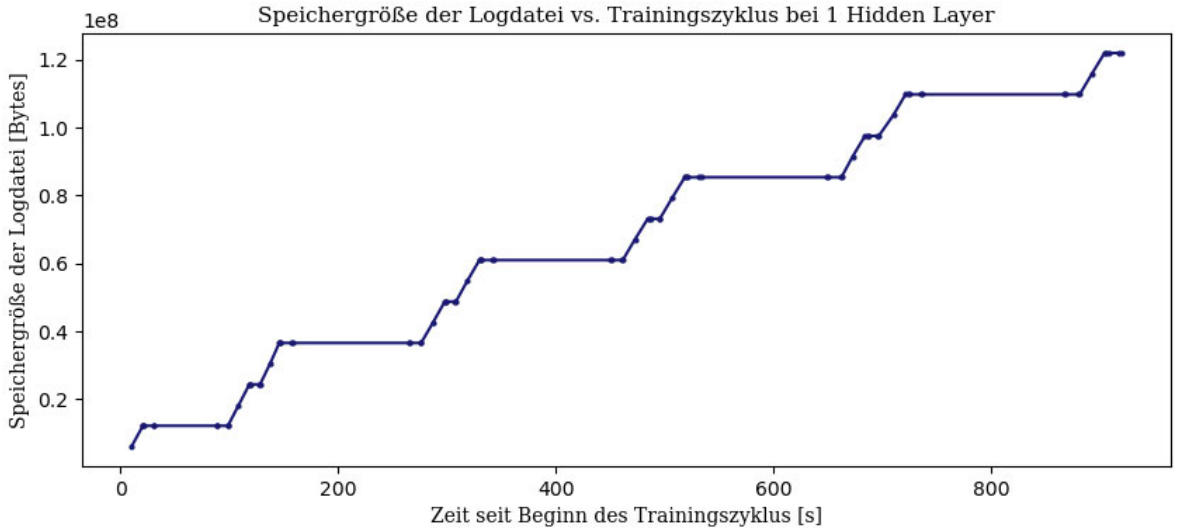


Abbildung D.1: Visualisierung der Speichergröße der Weights-Logdatei vs. Trainingszeit bei einem HL (höhere Anzahl an zwischengespeicherten Werten)

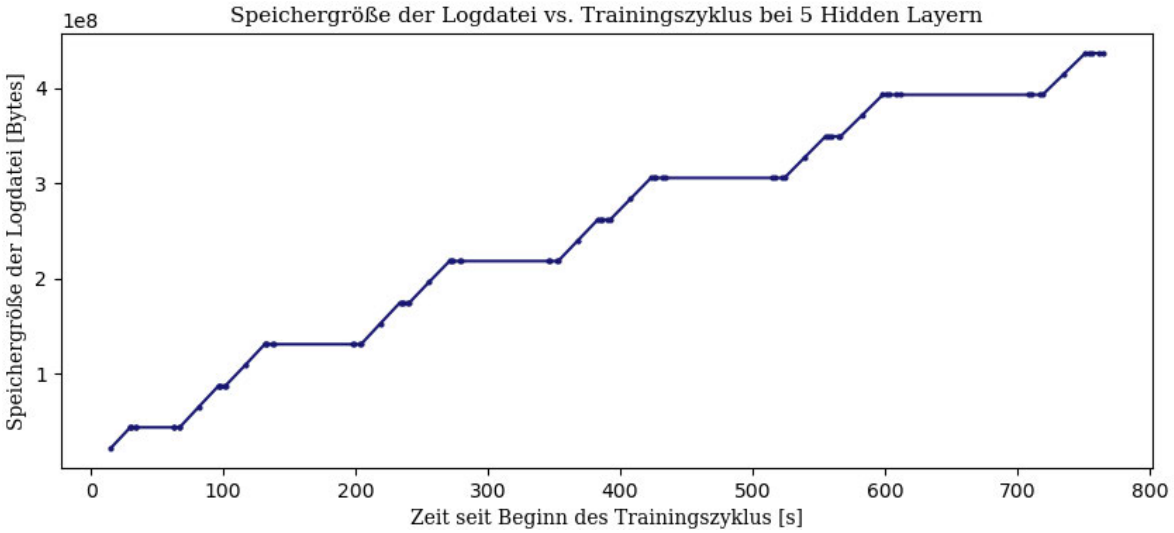


Abbildung D.2: Visualisierung der Speichergröße der Weights-Logdatei vs. Trainingszeit bei fünf HL (höhere Anzahl an zwischengespeicherten Werten)

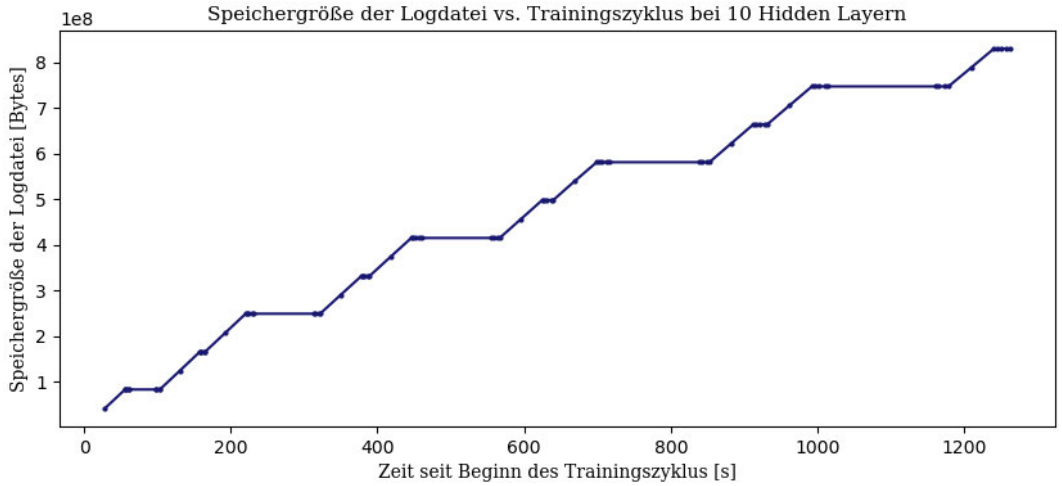


Abbildung D.3: Visualisierung der Speichergröße der Weights-Logdatei vs. Trainingszeit bei zehn HL (höhere Anzahl an zwischengespeicherten Werten)

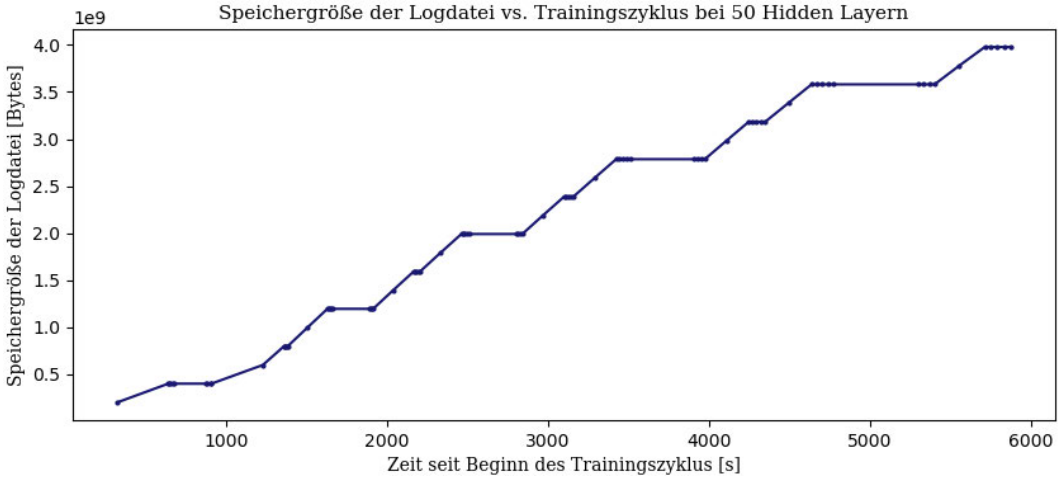


Abbildung D.4: Visualisierung der Speichergröße der Weights-Logdatei vs. Trainingszeit bei 50 HL (höhere Anzahl an zwischengespeicherten Werten)

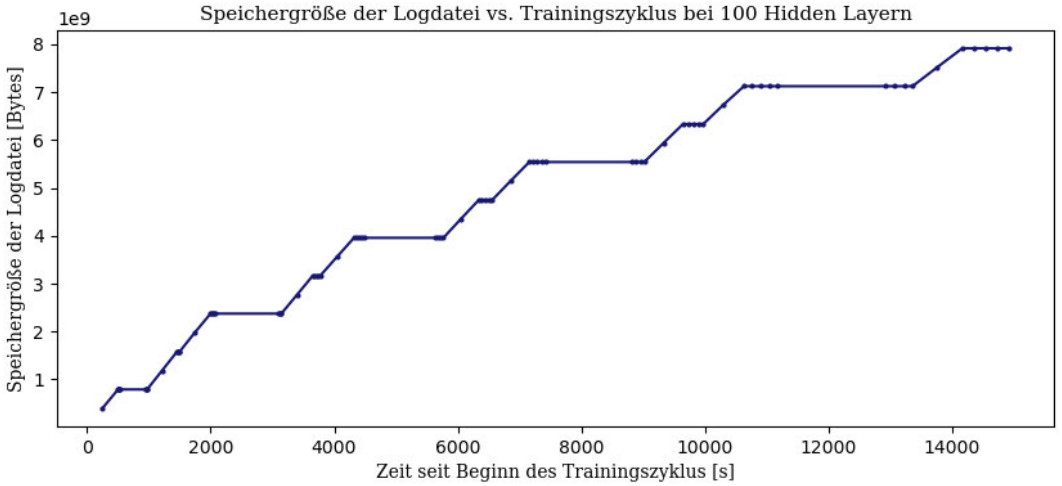


Abbildung D.5: Visualisierung der Speichergröße der Weights-Logdatei vs. Trainingszeit bei 100 HL (höhere Anzahl an zwischengespeicherten Werten)

Literaturverzeichnis

[1] Bundesministerium für Wirtschaft und Klimaschutz, Hrsg. „Künstliche Intelligenz, Stand der KI-Nutzung in der deutschen Wirtschaft im Jahr 2019“. dt. (2019), Adresse: https://www.de.digital/DIGITAL/Navigation/DE/Lagebild/Kuenstliche-Intelligenz/kuenstliche-intelligenz.html (besucht am 10. 11. 2022) (siehe S. 1).

[2] Duden.de, Hrsg. „au-di-tie-ren, Bedeutung | Synonyme zu auditieren“. dt. (2021), Adresse: https://www.duden.de/rechtschreibung/auditieren (besucht am 17. 06. 2022) (siehe S. 1).

[3] Bundesamt für Sicherheit in der Informationstechnik, Hrsg. „Leitfaden IT-Forensik“. dt. Version Version 1.0.1. (2011), Adresse: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/Themen/Leitfaden_IT-Forensik.pdf?__blob=publicationFile&v=1 (besucht am 11. 11. 2022) (siehe S. 1).

[4] W. Wahlster und C. Winterhalter, Hrsg. „Deutsche Normungsroadmap Künstliche Intelligenz“. dt. DIN DKE. (2020), Adresse: https://www.dke.de/resource/blob/2019482/0c29125fa99ac4c897e2809c8ab343ff/nr-ki-deutsch---download-data.pdf (besucht am 05. 07. 2022) (siehe S. 1, 2, 9–12, 24–28, 30).

[5] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders und K.-R. Muller, „Explaining Deep Neural Networks and Beyond: A Review of Methods and Applications“, engl. *Proceedings of the IEEE*, Jg. 109, Nr. 3, S. 247–278, 2021. DOI: 10.1109/JPROC.2021.3060483. Adresse: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9369420 (besucht am 30. 08. 2022) (siehe S. 1, 2, 19, 20).

[6] C. Schmidt, T. Lehmann, A. Zimmermann und C. Fuß. „Flagship-Projekt ZERTIFIZIERTE KI“. dt. DIN e.V., Hrsg. (2021), Adresse: https://din.one/display/ZKI/Flagship-Projekt+ZERTIFIZIERTE+KI (besucht am 13. 07. 2022) (siehe S. 3, 96).

[7] C. Berghoff u. a., *Towards Auditable AI Systems, Current status and future directions*. 2021. Adresse: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/KI/Towards_Auditable_AI_Systems.pdf?__blob=publicationFile&v=4 (besucht am 23. 06. 2022) (siehe S. 4, 8–12, 14, 15, 24–28, 30–32, 34).

[8] C. Berghoff u. a., *Towards Auditable AI Systems, From Principles to Practice*. 2022. Adresse: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/KI/Towards_Auditable_AI_Systems_2022.pdf?__blob=publicationFile&v=4 (besucht am 23. 06. 2022) (siehe S. 4, 27, 29).

[9] Fraunhofer Heinrich-Hertz-Institut, Hrsg. „Field of Research Artificial Intelligence, Quality Criteria for AI“. engl. (2022), Adresse: https://www.hhi.fraunhofer.de/en/fraunhofer-hhi/fields-of-research/artificial-intelligence.html (besucht am 13. 07. 2022) (siehe S. 4).

[10] J. Sachowski, *Implementing digital forensic readiness, From reactive to proactive process*, engl. Second edition. Boca Raton and London and New York: CRC Press, 2019, ISBN: 9780429441363. DOI: 10.1201/9780429441363. Adresse: https://www.taylorfrancis.com/books/mono/10.1201/9780429441363/implementing-digital-forensic-readiness-jason-sachowski (besucht am 30. 07. 2022) (siehe S. 4).

[11] S. Strecker, *Künstliche Neuronale Netze: Aufbau und Funktionsweise*, dt. 2004. Adresse: https://core.ac.uk/download/pdf/56347505.pdf (besucht am 05. 09. 2022) (siehe S. 5, 6).

[12] B. Aunkofer. „Funktionsweise künstlicher neuronaler Netze“. dt. Data Science Blog, Hrsg. (2018), Adresse: https://data-science-blog.com/blog/2018/08/31/funktionsweise-kunstlicher-neuronaler-netze/ (besucht am 06. 09. 2022) (siehe S. 5–7).

[13] A. Oppermann. „Aktivierungsfunktionen in Neuronalen Netzen: Sigmoid, tanh, ReLU“. dt. (2021), Adresse: https://artemoppermann.com/de/aktivierungsfunktionen/ (besucht am 07. 09. 2022) (siehe S. 6).

[14] J. Patterson und A. Gibson, *Deep learning, A practitioner's approach*, engl. First edition. Beijing u. a.: O'Reilly, 2017, 507 S., ISBN: 9781491914250. Adresse: https://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=1564784 (besucht am 22. 11. 2022) (siehe S. 7).

[15] R. Gupta. „Machine Learning Security Risks“. engl. Medium, Hrsg. (2019), Adresse: https://medium.com/@rohit_32839/machine-learning-security-risks-87bc9039a587 (besucht am 18. 06. 2022) (siehe S. 9, 10, 12, 13).

[16] I. J. Goodfellow, J. Shlens und C. Szegedy, *Explaining and Harnessing Adversarial Examples*, engl. arXiv, 2015. DOI: 10.48550/arXiv.1412.6572. Adresse: https://arxiv.org/pdf/1412.6572.pdf (besucht am 20. 06. 2022) (siehe S. 9).

[17] I. Goodfellow, N. Papernot, S. Huang, R. Duan, P. Abbeel und J. Clark. „Attacking Machine Learning with Adversarial Examples“. engl. OpenAI Blog, Hrsg. (2017), Adresse: https://openai.com/blog/adversarial-example-research/ (besucht am 18. 06. 2022) (siehe S. 9, 11).

[18] M. Asif. „Data Poisoning: When Artificial Intelligence and Machine Learning Turn Rouge“. engl. Medium, Hrsg. (2021), Adresse: https://medium.com/analytics-vidhya/data-poisoning-when-artificial-intelligence-and-machine-learning-turn-rouge-d8038f423922 (besucht am 18. 06. 2022) (siehe S. 10, 12).

[19] Z. Kong, J. Xue, Y. Wang, L. Huang, Z. Niu und F. Li, „A Survey on Adversarial Attack in the Age of Artificial Intelligence“, engl. *Wireless Communications and Mobile Computing*, Jg. 2021, S. 1–22, 2021, ISSN: 1530-8669. DOI: 10.1155/2021/4907754. Adresse: https://www.hindawi.com/journals/wcmc/2021/4907754/ (besucht am 06. 07. 2022) (siehe S. 10–13).

[20] Bundesamt für Sicherheit in der Informationstechnik. „Sicherer, robuster und nachvollziehbarer Einsatz von KI, Probleme, Maßnahmen und Handlungsbedarfe“. dt. Bundesamt für Sicherheit in der Informationstechnik. (2021), Adresse: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/KI/Herausforderungen_und_Massnahmen_KI.pdf?__blob=publicationFile&v=6 (besucht am 13. 07. 2022) (siehe S. 10).

[21] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li und D. Song, „The Secret Revealer: Generative Model-Inversion Attacks Against Deep Neural Networks“, engl. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 250–258, Juni 2020. DOI: 10.1109/cvpr42600.2020.00033. Adresse: https://openaccess.thecvf.com/content_CVPR_2020/html/Zhang_The_Secret_Revealer_Generative_Model-Inversion_Attacks_Against_Deep_Neural_Networks_CVPR_2020_paper.html (besucht am 30. 07. 2022) (siehe S. 10).

[22] T. F. Blauth, O. J. Gstrein und A. Zwitter, „Artificial Intelligence Crime: An Overview of Malicious Use and Abuse of AI“, engl. *IEEE Access*, Jg. 10, S. 77 110–77 122, 2022, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3191790. Adresse: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9831441 (besucht am 21. 09. 2022) (siehe S. 10).

[23] L. Yugeng u. a., „ML-Doctor: Holistic Risk Assessment of Inference Attacks Against Machine Learning Models“, engl. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA: USENIX Association, Aug. 2022, S. 4525–4542, ISBN: 978-1-939133-31-1. Adresse: https://www.usenix.org/conference/usenixsecurity22/presentation/liu-yugeng (besucht am 21. 09. 2022) (siehe S. 11).

[24] C. Dwork und A. Roth, „The Algorithmic Foundations of Differential Privacy“, engl. *Foundations and Trends in Theoretical Computer Science*, Jg. 9, Nr. 3-4, S. 211–407, 2014. DOI: 10.1561/0400000042. Adresse: https://www.cis.upenn.edu/~aaroth/Papers/privacybook.pdf (besucht am 22. 09. 2022) (siehe S. 11).

[25] C. K. Mummadi, T. Brox und J. H. Metzen, *Proceedings of the IEEE/CVF International Conference on Computer Vision, Defending Against Universal Perturbations With Shared Adversarial Training*, engl. 2019, 4928–4937. DOI: 10.1109/iccv.2019.00503. Adresse: https://openaccess.thecvf.com/content_ICCV_2019/papers/Mummadi_Defending_Against_Universal_Perturbations_With_Shared_Adversarial_Training_ICCV_2019_paper.pdf (besucht am 06. 07. 2022) (siehe S. 11).

[26] J. H. Metzen, N. Finnie und R. Hutmacher. „Meta Adversarial Training against Universal Patches“. engl. ICML, Hrsg. (2021), Adresse: https://arxiv.org/pdf/2101.11453.pdf (besucht am 06. 07. 2022) (siehe S. 11).

[27] T. B. Brown, D. Mané, A. Roy, M. Abadi und J. Gilmer. „Adversarial Patch“. engl. (2018), Adresse: https://arxiv.org/pdf/1712.09665.pdf?source=post_page (besucht am 06. 07. 2022) (siehe S. 11).

[28] A. Levine und S. Feizi. „(De)Randomized Smoothing for Certifiable Defense against Patch Attacks“. engl. Advances in Neural Information Processing Systems, Hrsg. (2020), Adresse: https://proceedings.neurips.cc/paper/2020/file/47ce0875420b2dbacfc5535f94e68433-Paper.pdf (besucht am 06. 07. 2022) (siehe S. 11).

[29] J. H. Metzen und M. Yatsura, „Efficient Certified Defenses Against Patch Attacks on Image Classifiers“, engl. *arXiv*, 2021. DOI: 10.48550/arXiv.2102.04154. Adresse: https://arxiv.org/pdf/2102.04154.pdf (besucht am 06. 07. 2022) (siehe S. 11).

[30] A. Athalye, N. Carlini und D. Wagner, „Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples“, engl. *International Conference on Machine Learning*, S. 274–283, 2018, ISSN: 2640-3498. Adresse: http://proceedings.mlr.press/v80/athalye18a.html (besucht am 06. 07. 2022) (siehe S. 12).

[31] I. Bär. „Machine Learning Security – Teil 2, Eine neue Herausforderung“. innoQ Deutschland GmbH, Hrsg. (Aug. 2021), Adresse: https://www.innoq.com/de/articles/2021/08/machine-learning-security-teil-2/ (besucht am 21. 09. 2022) (siehe S. 13).

[32] M. T. Ribeiro, S. Singh und C. Guestrin, „"Why Should I Trust You?": Explaining the Predictions of Any Classifier“, engl. *arXiv*, S. 1135–1144, Aug. 2016. Adresse: https://arxiv.org/pdf/1602.04938.pdf?source=post_page (besucht am 05. 07. 2022) (siehe S. 14, 18, 19, 22, 30).

[33] W. Samek, T. Wiegand und K.-R. Müller, „Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models“, engl., 2017. Adresse: https://arxiv.org/pdf/1708.08296.pdf (besucht am 20. 06. 2022) (siehe S. 14, 16–20).

[34] D. Doran, S. Schulz und T. R. Besold, „What Does Explainable AI Really Mean? A New Conceptualization of Perspectives“, engl. *arXiv*, 2017. Adresse: http://arxiv.org/pdf/1710.00794v1 (besucht am 20. 06. 2022) (siehe S. 15, 16).

[35] Z. C. Lipton, „The Mythos of Model Interpretability“, engl. *Queue*, Jg. 16, Nr. 3, S. 31–57, 2016. DOI: {10.48550/ARXIV.1606.03490}. Adresse: https://arxiv.org/pdf/1606.03490v3.pdf (besucht am 19. 08. 2022) (siehe S. 15).

[36] N. Schaaf, S. J. Wiedenroth und P. Wagner, *Erklärbare KI in der Praxis, Anwendungsorientierte Evaluation von xAI-Verfahren*, dt. T. Bauernhansl, M. Huber und K. Werner, Hrsg. Fraunhofer-Institut für Produktionstechnik und Automatisierung IPA, März 2021. Adresse: http://publica.fraunhofer.de/dokumente/N-630667.html (siehe S. 17, 18, 22, 23, 42, 57, 96).

[37] A. Shrikumar, P. Greenside und A. Kundaje, „International conference on machine learning, Learning Important Features Through Propagating Activation Differences“, engl. *PMLR*, S. 3145–3153, 2017. DOI: 10.48550/arXiv.1704.02685. Adresse: http://proceedings.mlr.press/v70/shrikumar17a/shrikumar17a.pdf (besucht am 08. 07. 2022) (siehe S. 18, 30).

[38] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh und D. Batra, „Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization“, engl. *CoRR*, Jg. 128, Nr. 2, S. 336–359, 2016. DOI: 10.48550/arXiv.1610.02391. Adresse: http://arxiv.org/abs/1610.02391 (besucht am 12. 08. 2022) (siehe S. 18).

[39] S. M. Lundberg und S.-I. Lee, „A unified approach to interpreting model predictions“, engl. *Advances in neural information processing systems*, Jg. 30, Guyon, I. and Von Luxburg, U. and Bengio, S. and Wallach, H. and Fergus, R. and Vishwanathan, S. and Garnett, R., Hrsg., S. 4765–4774, 2017. Adresse: https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf (besucht am 09. 08. 2022) (siehe S. 18, 22).

[40] R. Zeiler Matthew D.and Fergus, „Visualizing and Understanding Convolutional Networks“, engl. In *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele und T. Tuytelaars, Hrsg., Cham: Springer International Publishing, 2014, S. 818–833 (siehe S. 19).

[41] M. Sundararajan, A. Taly und Q. Yan, „Axiomatic Attribution for Deep Networks“, engl. *International Conference on Machine Learning*, S. 3319–3328, 2017, ISSN: 2640-3498. Adresse: http://proceedings.mlr.press/v70/sundararajan17a/sundararajan17a.pdf (besucht am 20. 06. 2022) (siehe S. 19).

[42] D. Smilkov, N. Thorat, B. Kim, F. Viégas und M. Wattenberg. „SmoothGrad: Removing noise by adding noise“. engl. (2017), Adresse: http://arxiv.org/abs/1706.03825 (besucht am 14. 09. 2022) (siehe S. 19).

[43] S. Lapuschkin, A. Binder, G. Montavon, K.-R. Müller und W. Samek, „The LRP Toolbox for Artificial Neural Networks“, engl. *Journal of Machine Learning Research*, Jg. 17, Nr. 114, S. 1–5, 2016. Adresse: http://jmlr.org/papers/v17/15-618.html (siehe S. 20).

[44] Z. Dong, K. Xu, Y. Yang, H. Bao, W. Xu und R. W. H. Lau, „Location-aware Single Image Reflection Removal“, engl., Aug. 2021. DOI: 10.48550/arXiv.2012.07131. Adresse: https://arxiv.org/pdf/2012.07131v2.pdf (besucht am 18. 07. 2022) (siehe S. 21).

[45] L. Fei-Fei, J. Deng, O. Russakovsky, A. Berg und K. Li. „ImageNet“. engl. Stanford Vision Lab, Hrsg., Stanford University, Princeton University. (2020), Adresse: https://www.image-net.org/ (siehe S. 21).

[46] A. Cummings. „Welcome to ISIC, The International Skin Imaging Collaboration“. engl. University Dermatology Center. (2022), Adresse: https://www.isic-archive.com (besucht am 19. 07. 2022) (siehe S. 21).

[47] Fraunhofer Heinrich-Hertz-Institut, Hrsg. „Spectral Relevance Analysis“. (2022), Adresse: https://www.hhi.fraunhofer.de/en/departments/ai/technologies-and-solutions/spectral-relevance-analysis.html (besucht am 18. 07. 2022) (siehe S. 21).

[48] S. Lapuschkin, S. Wäldchen, A. Binder, G. Montavon, W. Samek und K.-R. Müller, „Unmasking Clever Hans predictors and assessing what machines really learn“, engl. *Nature communications*, Jg. 10, Nr. 1, S. 1096, 2019. DOI: 10.1038/s41467-019-08987-4. eprint: 30858366. Adresse: https://www.nature.com/articles/s41467-019-08987-4.pdf?origin=ppub (besucht am 18. 07. 2022) (siehe S. 21, 96).

[49] C. J. Anders, L. Weber, D. Neumann, W. Samek, K.-R. Müller und S. Lapuschkin, „Finding and removing Clever Hans: Using explanation methods to debug and improve deep models“, engl. *Information Fusion*, Jg. 77, S. 261–295, 2022, ISSN: 15662535. DOI: \url{10.1016/j.inffus.2021.07.015}. Adresse: https://reader.elsevier.com/reader/sd/pii/S1566253521001573?token=3C9B01FE9D6AEF4C93F6F0F18C063D7F9D8AD27CE283461137AF90ED165972748C318F5ED0A3640FFC369C390593BAEB&originRegion=eu-west-1&originCreation=20220718200626 (besucht am 18. 07. 2022) (siehe S. 21).

[50] V. Arya u. a., „One explanation does not fit all: A toolkit and taxonomy of ai explainability techniques“, engl. *arXiv preprint arXiv:1909.03012*, Sep. 2019. Adresse: https://arxiv.org/pdf/1909.03012.pdf (besucht am 09. 08. 2022) (siehe S. 22).

[51] Seldon Technologies Ltd., Hrsg., *Alibi Explain*, 2019. Adresse: %7Bhttps://github.com/SeldonIO/alibi%7D (besucht am 09. 08. 2022) (siehe S. 22).

[52] Meta Platforms, Inc., Hrsg., *Captum, Model Interpretability for PyTorch*, 2019. Adresse: https://captum.ai/ (besucht am 09. 08. 2022) (siehe S. 22).

[53] M. Alber u. a., „iNNvestigate Neural Networks!“, *Journal of Machine Learning Research*, Jg. 20, Nr. 93, S. 1–8, 2019. Adresse: http://jmlr.org/papers/v20/18-540.html (besucht am 09. 08. 2022) (siehe S. 22).

[54] Nori, Harsha and Jenkins, Samuel and Koch, Paul and Caruana, Rich, „Interpretml: A unified framework for machine learning interpretability“, engl. *arXiv preprint arXiv:1909.09223*, Sep. 2019. Adresse: https://arxiv.org/pdf/1909.09223.pdf (besucht am 09. 08. 2022) (siehe S. 22).

[55] Oracle, Hrsg., *Skater*, 2017. Adresse: https://github.com/oracle/Skater (besucht am 09. 08. 2022) (siehe S. 22).

[56] Sicara, Hrsg., *tf-explain*, 2019. Adresse: https://github.com/sicara/tf-explain (besucht am 09. 08. 2022) (siehe S. 22).

[57] A. Hedström u. a., „Quantus: An Explainable AI Toolkit for Responsible Evaluation of Neural Network Explanations“, engl., 2022. DOI: 10.48550/arXiv.2202.06861. eprint: 2202.06861. Adresse: https://arxiv.org/abs/2202.06861 (besucht am 04. 08. 2022) (siehe S. 22).

[58] C. J. Anders, D. Neumann, W. Samek, K.-R. Müller und S. Lapuschkin, „Software for Dataset-wide XAI: From Local Explanations to Global Insights with Zennit, CoRelAy, and ViRelAy“, engl. *CoRR*, Jg. abs/2106.13200, 2021. DOI: 10.48550/arxiv.2106.13200. Adresse: https://arxiv.org/abs/2106.13200 (besucht am 10. 08. 2022) (siehe S. 22).

[59] Europäische Kommission, Hrsg. „Proposal for a regulation of the European Parliament and of the Council, Laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts“. engl. (2021), Adresse: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A52021PC0206 (besucht am 10. 07. 2022) (siehe S. 27).

[60] E. Štrumbelj und I. Kononenko, „Explaining prediction models and individual predictions with feature contributions“, engl. *Knowledge and Information Systems*, Jg. 41, Nr. 3, S. 647–665, 2014, ISSN: 0219-3116. DOI: 10.1007/s10115-013-0679-x. Adresse: https://idp.springer.com/authorize/casa?redirect_uri=https://link.springer.com/article/10.1007/s10115-013-0679-x&casa_token=d14ztgsz6qkaaaaa:orh68vz2wshtyyswmuumud0a8xfnklf7z8hxyerh5g-0mcz95nf9-ncz9je-jrayxoghudib4g2fo07y5w (besucht am 08. 07. 2022) (siehe S. 30).

[61] A. Datta, S. Sen und Y. Zick, „Algorithmic Transparency via Quantitative Input Influence: Theory and Experiments with Learning Systems“, engl. *IEEE Symposium on Security and Privacy*, S. 598–617, 2016. DOI: 10.1109/SP.2016.42. Adresse: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7546525 (besucht am 08. 07. 2022) (siehe S. 30).

[62] M. Mundt, S. Lang, Q. Delfosse und K. Kersting, *CLEVA-Compass: A Continual Learning EValuation Assessment Compass to Promote Research Transparency and Comparability*, engl. arXiv, 2022. DOI: 10.48550/arXiv.2110.03331. Adresse: https://arxiv.org/pdf/2110.03331.pdf (besucht am 20. 06. 2022) (siehe S. 33).

[63] S. Mishra. „A Brief Introduction to Continual Learning“. engl. Weights & Biases, Hrsg. (2021), Adresse: https://wandb.ai/shambhavicodes/ewc/reports/A-Brief-Introduction-to-Continual-Learning--VmlldzoxMTE4MTQ5?galleryTag=ml-news (besucht am 11. 06. 2022) (siehe S. 33, 34).

[64] A. Douillard und T. Lesort, *Continuum: Simple Management of Complex Continual Learning Scenarios*, engl. arXiv, 2021. DOI: 10.48550/arXiv.2102.06253. Adresse: https://arxiv.org/pdf/2102.06253.pdf (besucht am 20. 06. 2022) (siehe S. 33, 35).

[65] M. de Lange u. a., „A Continual Learning Survey: Defying Forgetting in Classification Tasks“, engl. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Jg. 44, Nr. 7, S. 3366–3385, 2022, ISSN: 0162-8828. DOI: 10.1109/tpami.2021.3057446. eprint: 33544669. Adresse: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9349197 (besucht am 12. 07. 2022) (siehe S. 33–35).

[66] G. M. van de Ven und A. S. Tolias, *Three scenarios for continual learning*, engl. arXiv, 2019, 18 S. DOI: 10.48550/arXiv.1904.07734. Adresse: https://arxiv.org/pdf/1904.07734.pdf (besucht am 20. 06. 2022) (siehe S. 35–37).

[67] V. Lomonaco u. a., „Avalanche: an End-to-End Library for Continual Learning“, engl. *arXiv*, S. 3600–3610, 2021. DOI: 10.48550/arXiv.2104.00405. Adresse: https://openaccess.thecvf.com/content/CVPR2021W/CLVision/papers/Lomonaco_Avalanche_An_End-to-End_Library_for_Continual_Learning_CVPRW_2021_paper.pdf (besucht am 20. 06. 2022) (siehe S. 37–39, 41, 96).

[68] Continual, Inc., Hrsg. „Welcome to Avalanche's API documentation!“ engl. (2022), Adresse: https://avalanche-api.continualai.org/en/latest/index.html (besucht am 15. 08. 2022) (siehe S. 38, 39, 43–46, 85).

[69] TensorFlow, Hrsg. „TensorFlow“. (2021), Adresse: https://www.tensorflow.org/ (besucht am 18. 06. 2022) (siehe S. 39).

[70] yatbear. „TensorBoard“. engl. GitHub, Inc., Hrsg. (2022), Adresse: https://github.com/tensorflow/tensorboard (besucht am 18. 06. 2022) (siehe S. 40).

[71] A. Paszke u. a., *Proceedings of the 33rd International Conference on Neural Information Processing Systems, PyTorch: An Imperative Style, High-Performance Deep Learning Library*, engl. Vancouver, Canada, 2019, 8026-8037. Adresse: https://www.researchgate.net/profile/benoit-steiner/publication/337756689_pytorch_an_imperative_style_high-performance_deep_learning_library (besucht am 18. 06. 2022) (siehe S. 40).

[72] Anaconda, Inc., Hrsg. „Anaconda Navigator, Desktop-Portal für Data Science“. (2022), Adresse: https://docs.anaconda.com/anaconda/navigator/ (besucht am 24. 08. 2022) (siehe S. 40).

[73] Jupyter, Hrsg. „Jupyter, Jupyter Notebook: Die klassische Notebook-Oberfläche“. (2022), Adresse: https://jupyter.org/ (besucht am 17. 08. 2022) (siehe S. 40, 42).

[74] Anaconda, Inc., Hrsg. „Spyder 5.3.3, The Scientific Python Development Environment“. engl. (2022), Adresse: https://anaconda.org/anaconda/spyder (besucht am 30. 11. 2022) (siehe S. 40, 42).

[75] Y. LeCun, C. Cortes und C. J. Burges, „The MNIST database of handwritten digits“, engl., 1998. Adresse: http://yann.lecun.com/exdb/mnist/ (besucht am 19. 06. 2022) (siehe S. 41).

[76] „National Institute of Standards and Technology | NIST“. engl. National Institute of Standards and Technology. (2022), Adresse: https://www.nist.gov (besucht am 19. 06. 2022) (siehe S. 41).

[77] ContinualAI, Hrsg. „avalanche.benchmarks.classic.SplitMNIST“. engl. (2022), Adresse: https://avalanche-api.continualai.org/en/v0.2.0/generated/avalanche.benchmarks.classic.SplitMNIST.html (besucht am 19. 06. 2022) (siehe S. 41).

[78] Weights & Biases, Hrsg. „The developer-first MLOps platform“. (2022), Adresse: https://wandb.ai/site (besucht am 19. 08. 2022) (siehe S. 41).

[79] J. Ernesti und P. Kaiser, *Python 3, Das umfassende Handbuch* (Rheinwerk Computing), dt. 6., aktualisierte Auflage. Bonn: Rheinwerk Verlag, 2020, 1079 S., ISBN: 978-3-8362-7926-0. Adresse: https://openbook.rheinwerk-verlag.de/python/index.html (besucht am 08. 09. 2022) (siehe S. 41, 87).

[80] PyTorch Contributors. „TORCH.NN“. (2022), Adresse: https://pytorch.org/docs/stable/nn.html (besucht am 26. 09. 2022) (siehe S. 44).

[81] PyTorch Contributors, Hrsg. „SGD“. (2022), Adresse: https://pytorch.org/docs/stable/generated/torch.optim.SGD.html (besucht am 16. 08. 2022) (siehe S. 44).

[82] PyTorch Contributors, Hrsg. „CrossEntropyLoss“. (2022), Adresse: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html (besucht am 16. 08. 2022) (siehe S. 44).

[83] Continual, Inc., Hrsg. „From Zero to Hero Tutorial“. engl. (2022), Adresse: https://avalanche.continualai.org/from-zero-to-hero-tutorial/01_introduction (besucht am 16. 08. 2022) (siehe S. 44–46).

[84] G.-L. Ingold. „Python für Naturwissenschaftler, Erstellung von Grafiken“. dt. GitHub, Inc., Hrsg. (2017), Adresse: https://gertingold.github.io/pythonnawi/graphics.html (besucht am 04. 11. 2022) (siehe S. 52–56).

[85] Deeplizard, Hrsg. „PyTorch - Python Deep Learning Neural Network API, CNN Weights - Learnable Parameters In PyTorch Neural Networks“. engl. (2018), Adresse: https://deeplizard.com/learn/video/stWU37L91Yc (besucht am 02. 10. 2022) (siehe S. 61).

[86] C. Wyawahare. „Leveraging the Power of Jupyter Notebooks“. engl. Towards Data Science, Hrsg. (Feb. 2020), Adresse: https://towardsdatascience.com/leveraging-the-power-of-jupyter-notebooks-26b4b8d7c622 (besucht am 11. 11. 2022) (siehe S. 83).

[87] M. Roodschild, J. Gotay Sardiñas und A. Will, „A new approach for the vanishing gradient problem on sigmoid activation“, engl. *Progress in Artificial Intelligence*, Jg. 9, Nr. 4, S. 351–360, 2020, ISSN: 2192-6360. DOI: 10.1007/s13748-020-00218-y. Adresse: https://link.springer.com/article/10.1007/s13748-020-00218-y (besucht am 01. 12. 2022) (siehe S. 84).

[88] LSoft Technologies Inc., Hrsg. „NTFS vs FAT vs exFAT“. (2022), Adresse: https://www.ntfs.com/ntfs_vs_fat.htm (besucht am 19. 11. 2022) (siehe S. 86).

[89] D. Salvator. „How Sparsity Adds Umph to AI Inference, What Is Sparsity in AI?“ engl. Nvidia Corporation, Hrsg. (2020), Adresse: https://blogs.nvidia.com/blog/2020/05/14/sparsity-ai-inference/ (besucht am 02. 12. 2022) (siehe S. 87).

[90] D. Giampaolo, *Practical file system design with the BE file system*, engl. San Francisco, California: Morgan Kaufmann Publishers, Inc., 1999, 237 S., ISBN: 1-55860-497-9. Adresse: http://www.nobius.org/dbg/practical-file-system-design.pdf (besucht am 02. 12. 2022) (siehe S. 87).

[91] Fraunhofer Heinrich-Hertz-Institut, Hrsg. „Human Understandable Explanations“. (2022), Adresse: https://www.hhi.fraunhofer.de/en/departments/ai/research-groups/explainable-artificial-intelligence/research-topics/human-understandable-explanations.html (besucht am 27. 08. 2022) (siehe S. 95, 96).

[92] Universität Kassel, Hrsg. „Neues Projekt ÄI Forensics: Accountability through Interpretability in Visual AI Systems", gefördert durch die VolkswagenStiftung“. dt. (2021), Adresse: https://www.uni-kassel.de/eecs/gedis/neuigkeiten/2021/12/8/neues-projekt-ai-forensics-accountability-through-interpretability-in-visual-ai-systems-gefoerdert-durch-die-volkswagenstiftung?cHash=d98cf5ef7dedbf1b57a06603b3dae727 (besucht am 18. 07. 2022) (siehe S. 97).

[93] Universität der Bundeswehr München, Hrsg. „Vertrauenswürdige KI für die polizeiliche Textauswertung“. dt. (2022), Adresse: https://www.unibw.de/code/news/viking-vertrauenswuerdige-ki-textauswertung (besucht am 18. 07. 2022) (siehe S. 97).

[94] Universität der Bundeswehr München, Hrsg. „Vertrauenswürdige Künstliche Intelligenz für polizeiliche Anwendungen (2022 – 2024)“. dt. (2022), Adresse: https://www.unibw.de/datensicherheit/professuren/data-science/forschung/viking (besucht am 18. 07. 2022) (siehe S. 97).

[95] „All Work Items For SAI“. engl. ETSI. (2022), Adresse: https://portal.etsi.org/webapp/WorkProgram/Frame_WorkItemList.asp?optDisplay=12&qTB_ID=877%3BSAI (besucht am 22. 08. 2022) (siehe S. 97).

[96] The EU AI Act Newsletter, Hrsg. „The Artificial Intelligence Act“. engl. (2021), Adresse: https://artificialintelligenceact.eu/ (besucht am 10. 11. 2022) (siehe S. 97).

Eidesstattliche Erklärung

Hiermit versichere ich – Olivia Sina Gräupner – an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt oder anderweitig veröffentlicht.

Halle (Saale), 12.12.2022 | Olivia Sina Gräupner, B.Sc.

Ort, Datum | Olivia Sina Gräupner, B.Sc.