Fakultät Elektrotechnik, Medien und Informatik

Analyse und Bewertung von Methoden zur Sicherung der Vertraulichkeit in Neuronalen Netzen

Masterarbeit im Studiengang Künstliche Intelligenz

vorgelegt von

Sabau Patrick

Matrikelnummer 12913724

Erstgutachter:	Prof. Dr.-Ing. Christoph P. Neumann
Zweitgutachter:	Prof. Dr. rer. nat. Daniel Loebenberger

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Selbstständigkeitserklärung

Name und Vorname
der Studentin/des Studenten: **Sabau Patrick**

Studiengang: **Künstliche Intelligenz**

Ich bestätige, dass ich die Masterarbeit mit dem Titel:

Analyse und Bewertung von Methoden zur Sicherung der Vertraulichkeit in Neuronalen Netzen

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum: 24. September 2023

Unterschrift:

Kurzdarstellung

Die steigende Popularität von neuronalen Netzen sorgt dafür, dass die Anzahl und Variation der Angriffe auf diese zu nimmt. Dabei können Eigenschaften der genutzten Trainingsdaten ermittelt werden oder sogar einzelne Datensätze rekonstruiert werden. Um dies zu verhindern, gibt es kryptografische und statistische Methoden, welche die Vertraulichkeit der Daten in neuronalen Netzen schützen. Kryptografische Methoden verschleiern dabei die Berechnungen, sodass der Anbieter des Modells keinen Zugriff auf die Daten des Anwenders erhält. Statistische Methoden beeinflussen die Daten, um einzelne Datensätze zu schützen. Hierbei ist die Technik Differential Privacy im Fokus. Diese macht den maximalen Einfluss eines Datensatzes auf eine Datenmenge quantifizierbar und kann diesen sogar einschränken. Differential Privacy kann während des Modelltrainings genutzt werden, um neuronale Netze und die Vorhersagen dieser zu schützen. Diese Arbeit stellt ein Handlungsmodell vor, welches den Einsatz von Differential Privacy in Kombination mit neuronalen Netzen nur empfiehlt, wenn die Effektivität der Angriffe einen festgelegten Schwellwert überschreitet.

Abstract

The popularity of neural networks leads to an increase in the number and variety of attacks against them. In the process, characteristics of the utilized training data can be determined, or even individual data records can be reconstructed. To prevent this, there are cryptographic and statistical methods that protect the confidentiality of data in neural networks. Cryptographic methods obscure the computations, so that the model provider can't gain access to the user's data. Statistical methods influence the data to protect individual data records. In this context, the technique of Differential Privacy is in focus. This makes the maximum impact of a data record on a dataset quantifiable and can even restrict it. Differential Privacy can be used during model training to protect neural networks and their predictions. This work presents an action model that recommends the use of Differential Privacy in combination with neural networks only when the effectivness of attacks exceeds a specified threshold.

Inhaltsverzeichnis

1 Einleitung **1**

2 Angriffe gegen Machine Learning Anwendungen **5**
2.1 Abhängigkeit von der Plattform 5
2.2 De-Anonymisierung und Re-Identifikation 6
2.3 Model Inversion Attacke 7
2.4 Property Inference Attacke 9
2.5 Membership Inference Angriff 11
2.6 Data Extraction Attacke 12
2.7 Poisoning Attacke 14
2.8 Angriffe gegen verteiltes Lernen 15
2.9 Zusammenfassung der Angriffe 18

3 Methoden zur Sicherung der Vertraulichkeit **21**
3.1 Übersicht der Pipeline 21
3.2 Aufbereitung des Datenbestands 23
3.2.1 Anonymisierung 23
3.2.2 Differential Privacy 28
3.2.3 Synthetische Daten 36
3.3 Training des Modells 40
3.3.1 Training mit Differential Privacy 40
3.3.2 Private Aggregation of Teacher Ensembles 42
3.3.3 Homomorphe Verschlüsselung 44
3.3.4 Funktionale Verschlüsselung 48
3.3.5 Verteiltes Lernen 50
3.4 Anpassung und Betrieb des Modells 53
3.4.1 Kryptografische Inferenz 54
3.4.2 Kompression des Modells 57
3.5 Zusammenfassung der Methoden 59

4 Bewertung der Methoden **63**
4.1 Kategorisierung anhand technischer Grundlage 63
4.2 Bewertung kryptografischer Methoden 64
4.3 Bewertung von Differential Privacy 67

4.4 Zusammenfassung der Bewertungen . . . 71

5 Experimente . . . 73
5.1 Datensatz und Use Cases . . . 73
5.2 Wahl des Frameworks . . . 76
5.3 Technische Implementierung von DPSGD . . . 77
5.4 Hyperparameter von DPSGD . . . 84
5.4.1 Hyperparameter CIFAR-10 Modell . . . 84
5.4.2 Hyperparameter ResNet-18 Modell . . . 90
5.4.3 Hyperparameter Vision Transformer Modell . . . 91
5.4.4 Handlungsempfehlung für die Wahl der Hyperparameter . . . 93
5.5 Angriffe . . . 94
5.5.1 Membership Inference Attacke . . . 94
5.5.2 Model Inversion Attacke . . . 97
5.6 Zusammenfassung der Experimente . . . 100

6 Abgeleitetes Handlungsmodell . . . 103

7 Diskussion und Ausblick . . . 107
7.1 Effektivität der Angriffe . . . 107
7.2 Weiterentwicklung von Angriffen durch moderne Modelle . . . 110

8 Zusammenfassung . . . 113

Abbildungsverzeichnis . . . 115

Tabellenverzeichnis . . . 117

Quellcodeverzeichnis . . . 119

Literaturverzeichnis . . . 121

A Übersicht von Hardware und Software . . . 131

B Ergebnisse im Detail . . . 133
B.1 Hyperparametertuning DPSGD CIFAR-10 Modell . . . 133
B.2 Hyperparametertuning DPSGD CelebA ResNet-18 Modell . . . 137
B.3 Hyperparametertuning DPSGD CelebA Vision Transformer Modell . . . 140
B.4 Membership Inference Attacke CIFAR-10 Modell . . . 143

Kapitel 1

Einleitung

Machine Learning ist spätestens seit der Veröffentlichung von ChatGPT[1] im Mainstream angelangt. Dabei handelt es sich um einen Chatbot des Unternehmens OpenAI, welcher mittels natürlicher Sprache mit Nutzern kommuniziert und eine Vielzahl an Aufgaben bewältigen kann. Bereits nach zwei Monaten nutzen über 100 Millionen verschiedene Personen den Chatbot und machen diesen damit zu der am schnellsten wachsenden Plattform überhaupt [1]. Im Hintergrund von ChatGPT läuft ein sogenanntes Large Language Modell, kurz LLM, welches anhand von menschlichem Feedback optimiert wurde [2].

Neuronale Netze und Machine Learning bilden die Grundlage von ChatGPT und sind bereits seit vielen Jahren in einer Vielzahl an Produkten integriert. So sortieren beispielsweise soziale Netzwerke Beiträge mittels Machine Learning nach der Beliebtheit [3], Sprachassistenten nutzen neuronale Netze, um schneller auf Nutzereingaben zu reagieren [4], autonomes Fahren wird auf Basis von Machine Learning erforscht [5] und für Übersetzungen wird ebenfalls Machine Learning genutzt [6]. Sogar im Bereich der Medizin und Biologie wird Machine Learning genutzt, um die Forschung voranzutreiben. So gibt es AlphaFold [7], ein neuronales Netz, welches die Faltung von Proteinen vorhersagt, um unter anderem die Entwicklung von Medikamenten zu beschleunigen. Jeder große Cloud-Provider bietet Services an, die Machine Learning für Kunden ermöglichen sollen, darunter Google Cloud Platform[2], Amazon Web Services[3] und Microsoft Azure[4]. Die Venture-Capital Gesellschaft FirstMark gibt ein Überblick über Unternehmen, die Produkte im Machine Learning Umfeld entwickeln und anbieten [8]. Dabei werden hunderte Unternehmen und Produkte dargestellt, welche den kompletten Lebenszyklus von Machine Learning Modellen abdecken, von der Datensammlung bis hin zur Integration in einen nutzbaren Service.

Eine Voraussetzung für diese Anwendungen sind Daten, viele Daten. Nicht nur öffentliche Daten, sondern auch private, sensible Daten. Diese werden von Unternehmen gesammelt und dazu genutzt, Produkte zu verbessern und neue Services zu entwickeln und anzubieten. Beispielsweise nutzen soziale Netzwerke die Interessen und Leidenschaften der Nutzer, um

[1] https://openai.com/blog/chatgpt
[2] https://cloud.google.com/products/ai
[3] https://aws.amazon.com/de/machine-learning/
[4] https://azure.microsoft.com/de-de/solutions/ai

die Reihenfolge von Beiträgen zu sortieren [3], Sprachassistenten hören ganze Gespräche mit, um jederzeit verfügbar zu sein und Systeme des autonomen Fahrens filmen die Umgebung und lesen den Standort des Fahrers aus, um Fahrsituationen meistern zu können. Die Kontrolle über die eigenen Daten an Unternehmen weiterzugeben, bring eine Vielzahl an Risiken mit sich. So wurden beispielsweise über 50 Millionen Profile des sozialen Netzwerks Facebook ausgelesen, ausgewertet und anschließend genutzt, um die Personen hinter den Profilen, in Bezug auf die US-Wahl 2016 zu manipulieren [9]. Solche Datenlecks oder auch Sammelklagen gegen Machine Learning Modelle, wie das Beispiel GitHub Copilot [10], zeigen, dass die Vertraulichkeit von Daten noch besser geschützt werden kann.

Das Bundesamt für Sicherheit in der Informationstechnik, kurz BSI, definiert das Schutzziel der Vertraulichkeit wie folgt [11]: *"Vertraulichkeit ist der Schutz vor unbefugter Preisgabe von Informationen. Vertrauliche Daten und Informationen dürfen ausschließlich Befugten in der zulässigen Weise zugänglich sein"*. Da Machine Learning Anwendungen, darunter auch neuronale Netze, vertrauliche Daten zum Training als auch für eine Vorhersage nutzen, gilt es auch hier, das Schutzziel der Vertraulichkeit zu gewähren. Dabei können unterschiedlichste Szenarien auftreten. Das wohl häufigste Szenario ist es, dass ein Unternehmen bereits Daten gesammelt hat, beispielsweise durch die gewöhnliche Nutzung einer Anwendung. Damit sollen nun Modelle trainiert werden, welche bestehende Anwendungen verbessern oder als neues Produkt angeboten werden. In diesem Fall soll sichergestellt werden, dass Nutzer der neuen Modelle, keine vertraulichen Informationen, die für das Training des Modells genutzt wurden, unbefugt erlangen können. Jedoch gibt es hier eine Reihe an Angriffen, welche einem Angreifer ermöglichen, diese vertraulichen Informationen zu erhalten. Ein weiteres Szenario ist das Training eines Modells auf einem fremden Server. Dies kann der Fall sein, wenn ein Unternehmen einen Cloud-Service nutzt, oder in einer verteilten Lernumgebung. Die Daten sollen nicht geteilt werden und dementsprechend nicht für andere Parteien einsehbar sein.

Die folgende Arbeit geht auf die Frage ein, wie der Schutz der Vertraulichkeit mit der Nutzung von Daten in neuronalen Netzen in Einklang gebracht werden kann. Dazu werden zuerst Angriffe beleuchtet, welche die Vertraulichkeit von neuronalen Netzen gefährden. Diese haben den Fokus, Daten aus bereits trainierten Modellen zu extrahieren, die eigentlich nicht extrahierbar sein sollten. Anschließend wird eine Reihe von Maßnahmen aufgeführt, die das Ziel haben, die vorher genannten Angriffe unwirksam zu machen. Diese Maßnahmen werden anschließend in kryptografische und statistische Methoden eingegliedert und bewertet. Teile der Bewertung entstammt dabei Experimenten, welche im Anschluss detailliert beschrieben werden. Außerdem wird anhand einiger Codebeispiele gezeigt, wie verschiedene Methoden technisch eingesetzt werden können. Die Bewertungen und die Ergebnisse der Experimente fließen in eine Handlungsempfehlung ein. Diese beschreibt, nach welchen Kriterien ein neuronales Netz in Bezug auf den Schutz der Vertraulichkeit bewertet werden soll und wann welche Maßnahmen sinnvoll eingesetzt werden. Aktuelle Probleme, wie die Effektivität von

Angriffen, werden zum Abschluss der Arbeit diskutiert. Der Ausblick zeigt, wie sich Angriffe weiterentwickeln und welche Risiken dadurch entstehen können.

Kapitel 2

Angriffe gegen Machine Learning Anwendungen

Neben den allgemeingültigen Risiken gegen die Informationssicherheit gibt es eine Reihe spezifischer Risiken von Machine Learning Anwendungen. Im Folgenden werden typische Angriffe gegen Machine Learning Modelle betrachtet und analysiert. Der Fokus liegt dabei auf Angriffen, welche besonders das Schutzziel der Vertraulichkeit bei neuronalen Netzen bedrohen. Angriffe, welche nur die Verfügbarkeit oder die Güte eines Modells angreifen, werden hier außer Betracht gelassen.

2.1 Abhängigkeit von der Plattform

Machine Learning Anwendungen werden oftmals in einer Public Cloud trainiert. Dafür gibt es eine Vielzahl an Gründen [12]:

- **Rechenleistung**: Die Komplexität eines neuronalen Netzes steigt mit der Komplexität der zu bewältigenden Aufgabe und der Menge an verfügbaren Daten. Mehr Komplexität bedeutet auch, dass mehr Rechenleistung benötigt wird. Eine Public Cloud hat die Möglichkeit, genügend Ressourcen bereitzustellen, um diese Rechenleistung zu bewältigen.
- **Flexibilität**: Das Training eines neuronalen Netzes ist rechenintensiver als die Inferenz von diesem. Durch dir Nutzung einer Public Cloud lässt sich die Rechenleistung so anpassen, dass während dem Betrieb des Modells, weniger Ressourcen genutzt werden, als während des Trainings. Diese Flexibilität sorgt für eine Kosteneffizienz. Zusätzlich bieten spezielle Hardware, spezielle Infrastruktur und spezielle Software die Möglichkeit, die Plattform für den Use Case anzupassen.
- **Hardware**: Neuronale Netze werden auf spezieller Hardware trainiert. Dazu zählen Grafikkarten, auch Graphics Processing Units oder GPUs genannt, und Tensor Processing Units, auch TPUs genannt. Bei der Nutzung der Public Cloud fällt die Anschaffung dieser Hardware weg. Stattdessen wird die Nutzungszeit der Hardware berechnet.

- **Integration**: Neben Services für die Entwicklung von Machine Learning Anwendungen gibt es eine Reihe weiterer Services in einer Public Cloud. Diese reichen von Datenbanken bis hin zu Visualisierungen. Dabei sind die Services so ausgelegt, dass die Integration verschiedener Services ohne großen Aufwand möglich ist.

Durch die Nutzung einer Public Cloud entstehen jedoch zusätzliche Sicherheitsrisiken. Neben gängigen Angriffen gegen jede Art von Webanwendungen, wie Ransomware-Angriffe oder Denial-of-Service-Attacken, gibt es eine Reihe weiterer Sicherheitsrisiken, welche die Vertraulichkeit der Daten angreifen [12].

Ein Datenleck ist ein nicht autorisierter Zugriff auf Daten und stellt damit eine große Gefahr bei der Nutzung von sensiblen Daten dar. Dabei können Daten von einem Angreifer gestohlen werden oder durch andere Fehler an die Öffentlichkeit gelangen. In einer Public Cloud können diese Datenlecks durch falsche Konfigurationen oder mangelnde Sicherheitsmaßnahmen entstehen [12]. Zusätzlich können Sicherheitslücken in der Software für ein Datenleck sorgen. Die Sicherheitslücken können dabei in dem eigenen Programmcode integriert sein, oder durch die Nutzung von Software von Drittanbietern stammen [12].

Die Public Cloud ist dabei nur ein Beispiel für die Abhängigkeit von Machine Learning Anwendungen an die darunterliegende Plattform. Ebenfalls kann der Betrieb eigener Server die gleichen Sicherheitsrisiken aufweisen.

2.2 De-Anonymisierung und Re-Identifikation

Für Machine Learning Anwendungen werden, je nach Komplexität der Aufgabe, eine Vielzahl an Daten benötigt. Durch Datenlecks können diese, oftmals privaten, Daten an die Öffentlichkeit oder in die Hände eines Angreifers gelangen. Ein Beispiel hierfür ist das Datenleck von Facebook, bei welchem 50 Millionen Nutzerprofile von dem Datenanalyse-Unternehmen Cambridge Analytica ausgelesen wurden. Die Daten der Profile wurden genutzt, um zielgerichtete politische Werbung auszuspielen, wodurch die US-Wahl 2016 beeinflusst wurde [9].

Allerdings kommt es auch vor, dass Unternehmen freiwillig Daten veröffentlichen. Netflix veröffentlichte 2006 einen Datenbestand, welcher Filmbewertungen von knapp 500.000 Nutzern enthält [13]. Um nicht absichtlich private Daten zu veröffentlichen, war der Datenbestand anonymisiert. Neben einem Wettbewerb steht dieser Datenbestand zu Forschungszwecken öffentlich zur Verfügung. Narayanan und Shmatikov [14] zeigen jedoch, dass die Anonymisierung des Datenbestands nicht ausreichend ist, um private Informationen zu schützen. Die Bewertungen der anonymisierten Benutzer, können mit Bewertungen der öffentlichen Filmdatenbank IMDb abgeglichen werden, sodass Nutzerprofile verbunden werden können. Dabei genügt es, wenn Präferenzen in Korrelation gesetzt werden können, die genauen Wert

sind nicht notwendig. Narayanan und Shmatikov [14] beschreiben, dass sogar politische oder religiöse Informationen abgeleitet werden können. Hierzu werden beispielsweise positive Bewertungen von religiösen Dokumentationsfilmen, die privat auf Netflix abgegeben werden, den entsprechenden öffentlichen Profilen auf IMDb zugeordnet.

2.3 Model Inversion Attacke

Bei der Model Inversion Attacke, versucht ein Angreifer, durch bestimmte Eingaben in ein Modell, Rückschlüsse auf die Trainingsdaten von diesem zu ziehen. Dies kann so weit führen, dass einzelne Datensätze nachgebildet werden können [15].

Fredrikson et. al. [15] zeigen anhand eines Gesichtserkennungsmodells, dass es möglich ist, das Bild einer Person zu rekonstruieren. Dabei handelt es sich um einen sogenannten White-Box Angriff. White-Box bedeutet, dass das Modell vollumfänglich in den Händen des Angreifers ist. Dies ist beispielsweise der Fall, wenn ein Modell öffentlich geteilt wird. Eine weitere Voraussetzung ist, dass die zu rekonstruierende Person von dem anzugreifenden Modell klassifiziert werden kann, folglich auch Bilder dieser Person in den Trainingsdaten vorhanden sind. Abbildung 2.1 zeigt, wie sehr das rekonstruierte Bild (links) dem Originalbild (rechts) ähnelt.

Abbildung 2.1: Rekonstruktion eines Bildes [15]

Der Angriff, auch Reconstruction Attacke genannt, wird durch einen iterativen Algorithmus durchgeführt. Zu Beginn wird ein initiales Bild, als Startbild gesetzt. Hat der Angreifer Hintergrundwissen zu den Daten, so kann er das initiale Bild ähnlich zu dem Zielbild setzen, ansonsten kann es mit festgelegten konstanten Werten oder zufälligen Werten initialisiert werden. In jedem Schritt des Algorithmus wird dieses Bild durch das Modell inferiert und anschließend der Wert einer Verlustfunktion bestimmt. Der Wert dieser ist, sofern das Modell nicht mehr Details zur Vorhersage angibt, lediglich der Wert 1 minus die Wahrscheinlichkeit, auch Confidence Score genannt, ob es sich bei dem eingegebenen Bild um die gesuchte Klasse, auch Label genannt, handelt. Konkret bedeutet dies, wenn das Bild bereits die zu rekonstruierende Person zeigt, ist der Confidence Score des Labels der Person nahe 1 und

damit der Wert der Verlustfunktion nahe 0. Der Wert der Verlustfunktion kann nun durch das Modell abgeleitet werden und ergibt somit Gradienten für das Bild. Dies gleicht dem Backpropagation Schritt des Trainings eines neuronalen Netzes, mit dem Unterschied, dass hier das Bild wie ein Teil des Modells behandelt wird und für dieses ebenfalls Gradienten berechnet werden. Diese Gradienten werden genutzt, um das Bild entgegengesetzt dieser Gradienten anzupassen, sodass der Wert der Verlustfunktion sinkt. Zusätzlich nutzen die Autoren einen Autoencoder, welcher das angepasste Bild in jedem Schritt harmonisiert und hochskaliert. Bei diesem Autoencoder handelt es sich um ein neuronales Netz, welches einen Input (hier ein Bild) in einen Vektor mit niedrigerem Rang transformiert und anschließend wieder in einen Vektor mit dem gleichen Rang wie der des Inputs transformiert. Input und Output eines Autoencoders haben somit den gleichen Rang, hier wird also ein Bild zu einem anderen Bild der gleichen Größe transformiert. Der Teil des Autoencoders, welcher das Eingabebild in einen Vektor mit niedrigerem Rang transformiert, wird Encoder genannt. Der Teil des Autoencoders, welcher das Hochskalieren übernimmt, heißt Decoder. Zum Training des Autoencoders werden Bilder genutzt, wobei der Input auch gleich dem gewünschten Output entspricht. Somit lernt ein Autoencoder, aus einem Bild das gleiche Bild zu erzeugen, jedoch mit der Einschränkung, dass der Vektor innerhalb des Modells einen niedrigen Rang hat. Folglich ist das erzeugte Bild nicht exakt identisch zu dem eingegebenen Bild. Der Autoencoder von Fredrikson et. al. [15] weist jedoch die Besonderheit auf, dass die Eingabebilder verrauscht werden und die gewünschten Ausgabebilder gleich bleiben. Ziel dadurch ist es, dass der trainierte Autoencoder genutzt werden kann, um unscharfe Bilder zu möglichst scharfen Bildern zu transformieren. Mit diesem Autoencoder soll also das rekonstruierte Bild nach jeder Iteration schärfer werden und damit näher an dem ursprünglichen Bild liegen. Die Reconstruction Attacke läuft so lange, bis die maximal konfigurierte Anzahl an Schritten erreicht wird, der Wert der Verlustfunktion einen bestimmten Wert unterschreitet oder sich eine definierte Anzahl an Schritten der Wert der Verlustfunktion nicht reduziert.

Zhang et. al. [16] erweitern diesen Angriff, indem die Qualität des Autoencoders verbessert wird. Der Autoencoder wird nicht nur auf verschwommenen Bildern trainiert, sondern zusätzlich noch auf Bilder, bei welchen jeweils nur ein Bildausschnitt maskiert wird. Eine weitere Verbesserung besteht darin, zusätzlich ein Modell zu nutzen, welches reale Bilder von synthetischen Bildern unterscheiden soll. Dieses Konstrukt entspricht dem Diskriminator eines Generative Adversarial Networks [17]. Als Input nutzt dieser Diskriminator Bilder, welche von dem Autoencoder generiert wurden. Der Wert einer Verlustfunktion des Outputs des Diskriminators kann dabei nicht nur durch den Diskriminator selbst backpropagiert werden, sondern zusätzlich auch noch durch den Autoencoder. Dies sorgt dafür, dass der Autoencoder ergänzend versucht, möglichst realistische Bilder zu erzeugen. Mithilfe der zusätzlichen Trainingsdaten und des Diskriminators, wird der Autoencoder verbessert, wodurch die Qualität der einzelnen Bilder erhöht wird und folglich auch die Qualität des rekonstruierten Bildes steigt. Der restliche Angriff erfolgt simultan zu der bereits beschrie-

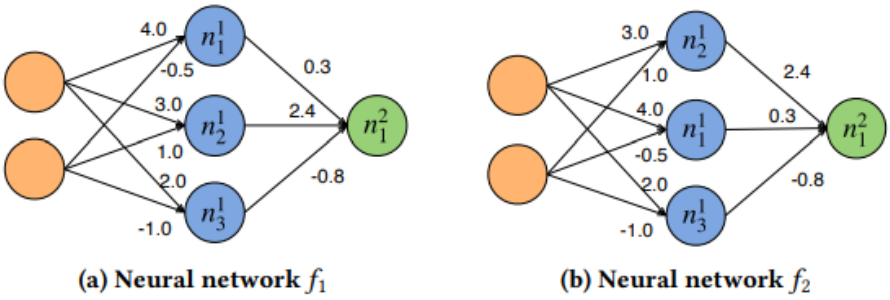
benen Vorgehensweise. Zhang et. al. [16] zeigen einen zusätzlichen Angriffsvektor, bei dem ein Angreifer bereits Teile eines Bildes hat. Dabei kann es sich um eine verschwommene Version des originalen Bildes handeln, oder um ein Bild, bei dem ein Teil beispielsweise mit einem schwarzen Balken zensiert wird. Da der Autoencoder zusätzlich mit diesen Fällen trainiert wurde, ist es möglich, die fehlenden Informationen eines Bildes zu ergänzen.

2.4 Property Inference Attacke

Bei der sogenannten Property Inference Attacke versucht ein Angreifer, bestimmte Eigenschaften über die Daten eines Modells herauszufinden, welche nur indirekt von einem Modell gelernt wurden und auch nicht bewusst veröffentlicht wurden [18].

Ateniese et. al. [18] zeigen erstmals, wie so ein Angriff bei einer Support Vektor Maschine, einer Art Machine Learning Modell, funktionieren kann. Dabei handelt es sich um einen White-Box Angriff. Es wird gezeigt, dass es möglich ist, herauszufinden, ob das angegriffene Modell Daten nutzte, welche eine bestimmte Eigenschaft besitzen. Bei dieser Eigenschaft könnte es sich z. B. um ein bestimmtes Geschlecht oder eine bestimmte Ethnie handeln. Um dies zu erreichen, konstruiert der Angreifer mehrere Datenmengen, in denen die zu untersuchende Eigenschaft vorhanden ist oder nicht. Anschließend werden diese Datenmengen genutzt, um verschiedene Modelle zu trainieren, welche die gleiche Architektur wie das anzugreifende Modell nutzen. Diese werden auch Shadow Modelle genannt. Der Schlüssel dieser Methode ist es, nachfolgend einen Meta-Klassifikator mit diesen Modellen zu trainieren. Bei Meta-Klassifikatoren handelt es sich um Modelle, welche aus anderen Modellen lernen. Konkret werden hierbei die Parameter der Shadow Modelle als Input genutzt, um vorherzusagen, ob die Trainingsdaten des jeweiligen Modells die zu untersuchende Eigenschaft besitzen. Dies ist möglich, da alle Modelle, das angegriffene Modell und die vom Angreifer trainierten Modelle, eine ähnliche oder sogar identische Architektur haben und dadurch auch zum gleichen Format transformiert werden können. Außerdem weiß der Angreifer, welches Shadow Modell eine Datenmenge mit oder ohne dem untersuchenden Attributwert nutzt. Ateniese et. al. [18] konnten mit diesem Angriff zeigen, dass es möglich ist, herauszufinden, ob ein Sprachmodell mit Daten der Eigenschaft "Sprecher mit indischem Dialekt "trainiert wurde.

Die Methode von Ateniese et. al. [18] ist auf diverse Machine Learning Modelle wie Hidden Markov Modelle oder Support Vektor Maschinen ausgelegt, weshalb diese bei neuronalen Netzen nicht sonderlich gut funktioniert. Ganju et. al. [19] adaptieren den Angriff für neuronale Netze, indem zwei geeignete Repräsentation für neuronale Netze genutzt werden, um den Meta-Klassifikator zu trainieren. Die erste Repräsentation eines neuronalen Netzes basiert auf der Eigenschaft von neuronalen Netzen, dass die Reihenfolge der Gewichte von Neuronen innerhalb einer Schicht, beliebig vertauscht werden können. Abbildung 2.2 zeigt



(a) Neural network f_1 **(b) Neural network f_2**

Abbildung 2.2: Permutation eines neuronalen Netzes [19]

zwei neuronale Netze, bei denen die Reihenfolge der Knoten in der ersten Hidden Layer vertauscht ist, jedoch das Modell exakt die gleiche Funktion berechnet. Diese Eigenschaft kann nun genutzt werden, um die Gewichte jeder Schicht nach der Größe zu sortieren und dadurch eine einheitliche Matrix für verschiedene Permutationen des gleichen Modells zu erhalten. Die zweite Repräsentation beruht darauf, die Schichten eines neuronalen Netzes nicht als Vektor, sondern als ein Set darzustellen. Im Gegensatz zu einem Vektor hat ein Set keine feste Reihenfolge oder Ordnung, sondern ist lediglich eine Menge von Objekten, in diesem Fall eine Menge der Gewichte der Knoten. Dies sorgt ebenfalls dafür, dass Permutationen des gleichen Modells, in das gleiche Format übertragen werden können. Ganju et. al. [19] zeigen anhand des MNIST Datenbestands [20], dass beide Repräsentationsformen die Genauigkeit des Meta-Klassifikators erhöhen können.

Gopinath et. al. [21] zeigen ein Vorgehen, welches ebenfalls Eigenschaften der Trainingsdaten aus einem Modell extrahiert. Dabei handelt es sich eigentlich nicht um einen bösartigen Angriff. Ziel der Methode ist es, nachvollziehen zu können, warum ein Modell gewisse Entscheidungen trifft, also die Erklärbarkeit eines Modells. Jedoch ist es nicht auszuschließen, dass die folgende Methode bösartig eingesetzt wird. Die Methode wird deshalb als White-Box Angriff behandelt, bei welchem die Trainingsdaten ebenfalls bekannt sind. Gopinath et. al. [21] zeigen, dass viele Informationen eines neuronalen Netzes in der Aktivierung der Neuronen steckt, also ob der Output eines Neurons in einem neuronalen Netz positiv oder negativ ist. Um die Aktivierung zu messen, wird eine Datenmenge aus den Trainings- und Testdaten gewählt und durch das Modell inferiert. Anschließend werden die Werte der Neuronen gemessen. Dabei reicht es, einen Wert in **on** und **off** zu unterteilen, wobei **on** einem Wert größer 0 entspricht und **off** einem Wert von 0 oder kleiner. Es ist möglich, ein vergleichbares Vorgehen, bei den Shadow Modellen durchzuführen und die daraus resultierenden Matrizen der Neuronenaktivierung zu nutzen, um einen Meta-Klassifikator zu trainieren.

2.5 Membership Inference Angriff

Bei der Membership Inference Attacke versucht ein Angreifer herauszufinden, ob ein Datensatz Bestandteil der Trainingsdatenmenge eines Modells ist. Dies bedroht die Vertraulichkeit, da beispielsweise herausgefunden werden kann, ob eine bestimmte Person Teil einer Datenmenge für die Diagnose einer Krankheit ist und folglich auch mit der entsprechenden Krankheit diagnostiziert ist [22].

Shokri et. al. [22] zeigen eine Membership Inference Attacke, welche nur die Vorhersage des anzugreifenden Modells nutzt. Dies wird auch Black-Box Angriff genannt, da die internen Gewichte des Modells nicht ersichtlich sind und damit das Modell wie eine Art Black-Box fungiert. Die Attacke wird durchgeführt, indem eine Reihe von Modellen trainiert wird, die ähnlich dem Angriffsziel-Modell sind. Diese Modelle sind Shadow Modelle. Ähnlich bedeutet hier, dass sowohl die Architektur der Modelle, als auch der Trainingsdatenbestand, mit dem angegriffenen Modell vergleichbar sind. Da es sich hier um einen Black-Box Angriff handelt, kann der Angreifer die Architektur nur anhand vergleichbarer Use Cases herleiten oder anhand öffentlicher Architekturen nachahmen. Als Trainingsdaten kann ein Angreifer öffentliche Datenbestände nutzen. Die Shadow Modelle werden dabei jeweils nur auf Teildatenmengen trainiert. Dadurch weiß der Angreifer, welche Datensätze in welchem Shadow Modell zum Training genutzt wurden und welche nicht. Anschließend wird mit den Shadow Modellen eine Datenmenge erzeugt, welcher die Wahrscheinlichkeiten der Klassifikationen der einzelnen Modelle enthält, so wie ein Label, ob das entsprechende Shadow Modell den besagten Datensatz in der Trainingsdatenmenge enthält oder nicht. Diese Datenmenge dient als Trainingsdaten für einen binären Meta-Klassifikator, welcher anhand der Vorhersagewahrscheinlichkeiten eines Modells bestimmen kann, ob ein bestimmter Datensatz in der Trainingsdatenmenge enthalten ist. Wird in diesem Meta-Klassifikator die Vorhersage des angegriffenen Modells als Input genutzt, so erhält man die Antwort, ob der zu untersuchende Datensatz für das Modelltraining genutzt wurde oder nicht. Laut Shokri et. al. [22] funktioniert dieser Angriff, da ähnliche Modelle, die mit einem ähnlichen Datenbestand trainiert werden, sich auch ähnlich verhalten. Somit kann bei den selbst trainierten Modellen, welche den zu untersuchenden Datensatz enthalten, ein Muster gelernt werden, welches auch auf andere Modelle anwendbar ist. Overfitting eines Modells und eine hohe Anzahl an Klassen, welche vorhergesagt werden, erhöhen die Wahrscheinlichkeit eines erfolgreichen Angriffs auf ein Modell. Overfitting bedeutet, dass ein Modell sehr stark an die Trainingsdaten angepasst ist und deshalb nicht mehr gut generalisiert.

Carlini et. al. [23] zeigen eine Alternative des Angriffs, bei welcher kein Meta-Klassifikator genutzt wird. Die Shadow Modelle werden analog zu der bereits beschriebenen Methode trainiert. Ebenfalls werden die Shadow Modelle genutzt, um Vorhersagewahrscheinlichkeiten zu ermitteln, von Datensätzen, die im Training genutzt wurden oder nicht. Anstatt damit einen Meta-Klassifikator zu trainieren, werden die Vorhersagewahrscheinlichkeiten genutzt,

um jeweils eine Gaußverteilung zu modellieren. Gaußverteilungen von Datensätzen, welche im Training genutzt werden, haben dabei andere statistische Merkmale, z. B. Mittelwerte und Standardabweichungen, wie die Gaußverteilungen von Datensätzen, welche nicht im Training genutzt werden. Die Vorhersagewahrscheinlichkeiten eines anzugreifenden Modells, mit einem speziellen Datensatz, können anschließend über einen Likelihood-Quotienten-Test den ähnlicheren Verteilungen zugeordnet werden. Dies entspricht dabei der Aussage, ob der spezielle Datensatz im Trainingsdatenbestand enthalten ist oder nicht. Ein Vorteil dieser Methode liegt darin, dass weniger Shadow Modelle trainiert werden müssen, um eine vergleichbare Qualität des Angriffs zu erhalten.

Eine Voraussetzung bei Shokri et. al. [22] ist es, dass der Confidence Score mit ausgegeben wird. Choquette-Choo et. al. [24] wandeln den Angriff ab, sodass dieser Score nicht mehr benötigt wird. Der Angriff funktioniert simultan zu [22], jedoch wird nicht nur der Datensatz selber in die Modelle als Input gegeben, sondern auch Abwandlung davon. Diese Abwandlungen könnte das Hinzufügen von zufälligem Rauschen sein, oder bei Bilddateien beispielsweise Rotation oder Translation. Die Hypothese der Autoren ist, dass das Modell bei Datensätzen, die im Training genutzt wurden, robuster gegenüber diesen Abwandlungen ist und dennoch den Datenpunkt korrekt klassifiziert. Zusätzlich könnten Abwandlungen der Daten über einen Data Augmentation Schritt auch direkt vom Modell gelernt worden sein. Werden diese Abwandlungen also falsch klassifiziert, ist dies ein Indiz dafür, dass der Datensatz nicht im Trainingsdatenbestand des anzugreifenden Modells ist.

2.6 Data Extraction Attacke

Bei der Data Extraction Attacke versucht ein Angreifer, Informationen eines Modells zu extrahieren, die gelernt wurden, obwohl dies generell nicht der Fall sein sollte [25]. Der Angriff unterscheidet sich von der Model Inversion Attacke (Kapitel 2.3) und von der Property Inference Attacke (Kapitel 2.4), indem Daten direkt aus dem Modell extrahiert werden und nicht anhand der Vorhersage des Modells nachgebildet werden.

Carlini et. al. [25] beschreiben, dass konkrete Zeichenketten oder Werte, wie eine Kreditkartennummer oder eine Sozialversicherungsnummer, von einem Sprachmodell ungewollt gelernt werden können. Um dies zu evaluieren, wird ein Sprachmodell mit der Penn Treebank Datenmenge trainiert, welche ca. 5 MB groß ist. Zusätzlich wurde ein Satz eingefügt, welcher mit den Worten "*My social security number is*" beginnt und anschließend eine Zahlenfolge beinhaltet. Die Funktionalität des Modells liegt darin, das nächste Wort oder Zeichen vorherzusagen, wenn eine Zeichenkette eingegeben wird. Anzumerken ist hierbei noch, dass dieses Modell signifikant kleiner als 5 MB ist, was folglich bedeutet, dass nicht alle Trainingsdaten in dem Modell gespeichert sein können. Die Experimente von Carlini et. al. [25] zeigen, dass

dieses Modell die Zahlenfolge ungewollt gelernt hatte und als mögliche Vorhersage ausgibt, wenn der oben genannte Satz als Input genutzt wird.

In einem anderen Forschungsprojekt, ebenfalls unter der Leitung von Carlini [26], zeigen die Autoren eine Data Extraction Attacke am Beispiel des Sprachmodells GPT-2. Dabei handelt es sich um ein Sprachmodell des Unternehmens OpenAI, welches der Vorgänger von ChatGPT (siehe Kapitel 1) ist und Open-Source zur Verfügung steht. Obwohl voller Zugriff auf das Modell besteht, wird lediglich die Ausgabe des Modells betrachtet. Folglich bedeutet dies, dass der Angriff auf jedes Sprachmodell mit der gleichen Funktionsweise anwendbar wäre. Ziel des Angriffs ist es, Tokens vom Modell vorgeschlagen zu bekommen, welche sensible Informationen enthalten. Bei Token handelt es sich um Wörter, Teile von Wörtern, Zahlen oder Zeichen, mit welchen ein Sprachmodell trainiert wird. Zur Durchführung des Angriffs wird lediglich ein Starttoken in das Modell eingegeben und anschließend vielfach das vorgeschlagene Folgetoken gesammelt. Wird dies lang genug wiederholt, erhält man eine lange Tokenabfolge, welche Sätzen entspricht, die vom Modell gelernt wurden. Dabei kann es sich um öffentliche Texte handeln, wie beispielsweise der Text der MIT Open-Source-Lizenz, aber auch private Daten wie E-Mail-Adressen kommen in der gesammelten Tokenabfolge vor. Diese Variation des Angriffs kann in gewissem Maße funktionieren, liefert jedoch oftmals gleiche Wortabfolgen und hat zusätzlich eine hohe Falsch-positiv-Rate. Carlini et. al. [26] variieren deshalb die Methodik, mit der die Tokenabfolge gesammelt wird. Bevor GPT-2 das wahrscheinlichste Folgewort vorschlägt, werden die Wahrscheinlichkeiten in den Wertebereich (0,1) transformiert und so skaliert, dass diese Werte addiert 1 ergeben. Dies entspricht der Softmax-Funktion. Wird der Softmax-Funktion ein Hyperparameter namens Temperatur mit einem Wert über 1 mitgegeben, wird das Modell unsicherer und erhöht dadurch die Diversität der Vorhersagen des Modells. Folglich werden öfters unterschiedliche Tokens vorgeschlagen, die bisher noch nicht gesammelt wurden. Neben dem Setzen der Temperatur einer Softmax-Funktion, wird eine zweite Verbesserung vorgeschlagen. Anstatt nur einen Starttoken zu nutzen, werden die ersten Wörter von verschiedenen, öffentlichen Datenquellen genutzt. Mit diesen zwei Verbesserungen können mehr unterschiedliche Arten von Texten, die das Modell gelernt hat, extrahiert werden. Neben Newsartikeln oder Forumsbeiträgen befinden sich auch Kontaktdaten einiger Privatpersonen in diesen Tokenabfolgen.

Large Language Modelle, kurz LLMs, haben durch die Veröffentlichung von ChatGPT enorm an Beliebtheit gewonnen [1]. Jedoch sorgt die Beliebtheit ebenfalls dafür, dass Angreifer diese Modelle als Ziel betrachten. Die Open Web Application Security Project Foundation, kurz OWASP Foundation, hat deshalb ein Ranking der 10 größten Sicherheitsrisiken von LLMs und Anwendungen, die auf diesen aufbauen, veröffentlicht [27]. In diesem Ranking findet sich auch der Punkt "*Sensitive Information Disclosure*", welcher die Preisgabe von sensiblen Informationen beschreibt. Die eben beschriebenen Data Extraction Angriffe zielen auf dieses Sicherheitsrisiko.

2.7 Poisoning Attacke

Bei der sogenannten Poisoning Attacke werden manipulierte Datensätze in die Trainingsdatenmenge eines Modells injiziert, wodurch das Modell schlechtere oder sogar falsche Vorhersagen trifft. Ursprünglich ist diese Art des Angriffs recht populär bei Support Vektor Maschinen. Biggio et. al. [28] zeigen, dass einige modifizierte Datenpunkte in der Nähe der Entscheidungsgrenze einer Support Vektor Maschine genügen, um die gelernte Funktion deutlich negativ zu beeinflussen.

Yang et. al. [29] zeigen ein Verfahren, bei dem eine Poisoning Attacke auf neuronalen Netzen angewendet wird. Ziel hierbei ist es, die gefälschte Daten mit einem absichtlich falschen Label so zu wählen, dass der Wert der Verlustfunktion möglichst groß ist. Um dies zu erreichen, wird der Wert der Verlustfunktion des anzugreifenden Modells durch das Modell backpropagiert, wobei die gefälschten Daten als Teil des Modells betrachtet werden. Die Daten werden nun in Richtung der Gradienten angepasst, wodurch der Wert der Verlustfunktion steigt. Werden die gefälschten Daten anschließend genutzt, um das Modell weiter zu trainieren, wird die Verlustfunktionen einen erhöhten Wert aufweisen, was zu einer stärkeren Anpassung des Modells führt, was aufgrund der gefälschten Label zu einer Verschlechterung der Güte des Modells führt. Um nicht als gefälschte Daten aufzufallen, nutzen Yang et. al. [29] einen Autoencoder, der Daten so transformiert, dass diese vom Modell als echte Daten erkannt werden. Damit der Autoencoder lernt, realistische Daten nachzubilden, wird ein zusätzliches Modell genutzt, welches einem Diskriminator der Generative Adversarial Network Architektur [17] entspricht.

Guo und Liu [30] nutzen einen Ansatz, bei welchem der Angreifer keinen Zugriff auf die Gradienten des angegriffenen Modells braucht. Stattdessen wird ein vortrainiertes Modell genutzt, welches ähnlich zu dem angegriffenen Modell ist. Da diverse Modellarchitekturen Open-Source sind, finden sich auch einige vortrainierte Varianten von diesen im Internet. Bei Bildklassifikation lässt sich beispielsweise ein vortrainiertes YOLO-Modell nutzen. Dieses kann dann genutzt werden, um ein generatives Modell zu trainieren, welches wie bei Yang et. al. [29] die Gradienten des anzugreifenden Modells nutzt, um Daten für das anzugreifende Modell zu verschlimmern. Guo und Liu [30] gehen davon aus, dass das angegriffene Modell noch optimiert wurde und deshalb eine bessere Feature-Erkennung hat, als die öffentlich vortrainierten Modelle. Dies macht den Angriff effektiver, sofern die Modelle nicht zu unterschiedlich sind.

Poisoning Attacken verschlechtern in der Regel lediglich die Performance eines Modells und sorgen für falsche Vorhersagen. Tramèr et. al. [31] zeigen jedoch, dass manipulierte Daten dafür sorgen können, dass andere Angriffe, welche die Vertraulichkeit angreifen, effektiver werden können. Durch Ändern des Labels eines Datensatzes kann dieser gegebenenfalls zu einem Ausreißer transformiert werden. Dadurch passt sich das Modell stärker diesem

an, als wenn sich der Datenpunkt in die Messreihe einordnet. Die falsche Klassifizierung des veränderten Datensatzes würde eine Memberhsip Inferenze Attacke auf diesen verbessern.

2.8 Angriffe gegen verteiltes Lernen

Die Größe und Komplexität eines Machine Learning Modells korreliert mit dem Funktionsumfang der zu bewältigen Aufgabe. Dies führt dazu, dass eine große Datenmenge und viel Rechenleistung benötigt werden. Ist eines dieser beiden Ressourcen knapp, können diese von mehreren Partnern geteilt werden. Das verteilte Lernen birgt jedoch einige besondere Risiken, welche im Folgenden detaillierter betrachtet werden.

Verteiltes Lernen, auch Federated Learning oder Collaborative Learning genannt, kann unterschiedlich durchgeführt werden. Diverse Parameter, beispielsweise wie oft Systeme ihre Änderungen übermittelt, können konfiguriert werden. Jedoch ist eine wichtige Unterscheidung die Topologie des Systems, welche in Abbildung 2.3 gezeigt werden. Links sieht man ein System, welches einen zentralisierten Server benutzt. Die Clients, dargestellt durch Smartphones, berechnen mit ihren privaten Daten Gradienten, welche dann an einen zentralen Server übermittelt werden. Dort findet die Anpassung des Modells statt. Rechts sieht man einen dezentralen Ansatz, bei dem Clients miteinander vernetzt sind und ihre Updates (z. B. Gradienten) untereinander austauschen [32].

Abbildung 2.3: Verteiltes Lernen Topologien [32]

Melis et. al. [33] zeigen, dass verteiltes Lernen anfällig für Membership Inference Attacken (siehe Kapitel 2.5) und Property Inference Attacken (siehe Kapitel 2.4) ist. Ein Membership Inference Angriff kann durchgeführt werden, indem die Gradienten der Eingabeschicht eines neuronalen Netzes beobachtet werden. Bei Textmodellen beispielsweise, ist es oftmals der Fall, dass die Worte bzw. Tokens in einen Vektor übertragen werden. Dieser Vektor enthält dabei die Information, welche Knoten der ersten Schicht aktiviert sind. An besagten Stellen der ersten Schicht, sind die Gradienten deshalb ungleich 0. Durch diese Beobachtung kann ein Angreifer also feststellen, welche Wörter in einem Datensatz oder einem Batch aus Datensätzen enthalten sind. Für einen Property Inference Angriff, kann der Angreifer verschiedene Snapshots des gemeinsam trainierten Modells nutzen. Jeder dieser Snapshots

wird zweimal separat weitertrainiert, einmal mit einer Datenmenge, welcher den zu untersuchende Attributwert enthält und einmal mit einer Datenmenge, welche diesen nicht enthält. Die Unterschiede von den ursprünglichen Snapshot-Modellen und den weitertrainierten Modellen sind folglich die Inputs und die Labels sind die Information, ob Daten mit der Eigenschaft oder ohne die Eigenschaft zum Lernen genutzt wurden. Damit kann nun ein Meta-Klassifikator trainiert werden, der vorhersagt, ob ein Modell weitertrainiert wurde, mit Daten, welche den zu untersuchenden Attributwert enthalten. Bei jedem Update des gemeinsam gelernten Modells kann dieser Meta-Klassifikator genutzt werden.

Zhu et. al. [34] beschrieben einen Angriff auf verteilte Systeme, welcher Deep Leakage genannt wird. Der Angriff funktioniert sowohl bei einer zentralisierten als auch einer dezentralisierten Topologie. Beim zentralisierten System muss der Angreifer jedoch Zugriff auf den zentralen Server haben, beim dezentralisierten System reicht es, ein Teilnehmer zu sein. Dies liegt daran, dass die Gradienten, bzw. eine Gradientenmatrix, mit dem Angreifer geteilt werden müssen, damit der Angriff funktioniert. Im Laufe des Trainingsprozesses werden mehrfach Gradienten dem Angreifer übergeben, und für jeden Austausch, können die eingegebenen Trainingsdaten ermittelt werden. Um den Angriff für eine Gradientenmatrix durchzuführen, wird initial ein Eingabevektor generiert. Dieser wird durch das gemeinsam zu trainierende Modell inferiert, der Wert der Verlustfunktion bestimmt und anschließend durch das Modell backpropagiert. Anstatt jedoch die Gewichte der Knoten des Modells anzupassen, werden nur die Werte des Eingabevektors iterativ, multipliziert mit einer festgelegten Lernrate, angepasst, sodass sich die Gradienten des Eingabevektors der geteilten Gradientenmatrix annähern. Durch diese Angleichung, wird der ursprünglich initial gesetzte Eingabevektor immer ähnlicher zu den originalen Trainingsdaten, die ein anderer Nutzer des verteilten Systems zum Training beigetragen hat. Die Autoren zeigen, dass mit dieser Attacke rekonstruierte Bilder, nahezu identisch mit den Originalbildern sind. Die Attacke ähnelt dabei einer Model Inversion Attacke, jedoch in einem verteilten System, bei welchem der Angreifer Zugriff auf das Modell durch das verteilte Training erhält. Abbildung 2.4 zeigt eine Gegenüberstellung der Bilder. Die Bilder können ebenfalls rekonstruiert werden, wenn mehrere Bilder in Batches zum Training genutzt werden.

Zhao et. al. [35] optimierten den Deep Leakage Angriff, indem ein zusätzlicher Schritt in den Algorithmus eingefügt wird. Dieser beinhaltet, dass zuerst das Label des Datensatzes herausgefunden wird, von dem die Gradienten stammen. Dies kann dadurch ermittelt werden, dass die Kostenfunktion eines Outputs bei der richtigen Klasse den Wertebereich (-1,0) annimmt und bei den falschen Klassen den Wertebereich (0,1), sofern keine Quadrierung stattfindet. Der initiale Eingabevektor hat nun von Beginn an das richtige Label. Dadurch werden weniger Iterationen benötigt, um Daten zu rekonstruieren.

Hitaj et. al. [36] zeigen einen alternativen Angriff. Bei diesem Szenario einigen sich die Trainingsteilnehmer darauf, von welchen Klassen sie jeweils Daten haben und entsprechend auch,

Abbildung 2.4: Rekonstruktion durch Deep Leakage [34]

welche Klassen von welchem Teilnehmer im gemeinsamen Modell trainiert werden. Klassen können sich unter den Teilnehmern überlappen, sodass mehrere Teilnehmer diese Klasse trainieren können. Jedoch benötigt der Angreifer eine Klasse, die außer ihm niemand anderes trainiert. Diese muss nicht wirklich in den Daten vorkommen, sondern es geht darum, dass kein anderer Teilnehmer die Klassifikation dieser Klasse beeinflusst. Zur Durchführung des Angriffs, wird das Modell wie geplant trainiert, bis eine gewisse Güte erreicht ist. Der Angreifer versucht anschließend, Daten einer Klasse zu rekonstruieren, die nur von anderen Teilnehmern bespielt wurde. Dazu wird ein Generative Adversarial Network [17] genutzt. Dabei handelt es sich um eine Architektur, welche sich aus zwei Modellen, einem Generator und einem Diskriminator, zusammensetzt. Der Diskriminator entscheidet dabei, ob die Daten, die vom Generator erzeugt werden, echt oder unecht sind. In diesem Angriff wird als Diskriminator das gemeinsam gelernte Modell genutzt, welches vorhersagt, ob die Daten Teil der angegriffenen Klasse sind oder nicht. Somit sind die Label der angegriffenen Klasse echten Daten, jede andere Klasse entspricht unechten Daten. Der Generator wird vom Angreifer trainiert, indem er das gemeinsame Modell durch den Generator backpropagiert. Diese erzeugten Daten, kann der Angreifer der Klasse zuordnen, die außer ihm niemand trainiert. Somit werde echte Daten dieser Klasse öfters falsch klassifiziert, nämlich in die Klasse des Angreifers. Dadurch werden die anderen Teilnehmer gedrängt, mehr oder öfters Daten für die angegriffene Klasse preis zu geben, da das gemeinsame Modell nicht mehr zwischen der angegriffenen Klasse und der zugeordneten Klasse des Angreifers unterscheiden kann. Anschließend kann der Angreifer erneut den Generator verbessern. Anzumerken ist hierbei noch, dass der Generator nicht die Trainingsdaten im Detail rekonstruiert, son-

dern nur Daten erzeugt, welche der gleichen Klasse zugeordnet werden können. Die Autoren zeigen jedoch, dass bei Bildern eine erkennbare Ähnlichkeit vorhanden ist.

2.9 Zusammenfassung der Angriffe

Machine Learning Anwendungen haben, wie jede Softwareanwendung, eine gewisse Abhängigkeit an die Plattform, auf welcher die Anwendung läuft. Dabei hat jede Plattform eigene Risiken. Bei einer Public Cloud können beispielsweise falsche Konfigurationen oder fehlende Sicherheitsmaßnahmen für ein Datenleck sorgen (Kapitel 2.1).

Ein Datenleck ist eines der größten Probleme von datengetriebenen Anwendungen. Dabei werden Daten ungewollt veröffentlicht oder von Angreifern entwendet. Selbst wenn diese Daten anonymisiert sind, können diese oftmals zu den Originaldaten zurückgeführt werden. Dies kann über statistische Methoden erfolgen, indem die anonymisierten Daten mit weiteren, öffentlichen Datenbeständen verbunden werden (Kapitel 2.2).

Jedoch können Daten auch rekonstruiert werden, wenn nur ein Modell bereitgestellt wird. Dies wird Model Inversion Attacke genannt. Dabei versucht ein Angreifer, die Vorhersage eines Modells zu nutzen, um einen Datensatz so anzupassen, dass dieser einem echten Datensatz ähnelt. Dafür wird ein iterativer Prozess genutzt, welcher in jedem Schritt den initialen, zufällig erzeugten Datensatz mittels der Vorhersage des Modells anpasst und immer mehr einem echten Datensatz angleicht (Kapitel 2.3).

Wenn das Ziel nicht einzelne Datensätze sind, sondern Eigenschaften, die für eine ganze Datenmenge gelten, können diese mit der Property Inference Attacke analysiert werden. Dabei werden Shadow Modelle trainiert, welche versuchen, das originale Modell zu imitieren. Das Besondere an diesen Shadow Modellen ist es, dass manche mit einer Datenmenge trainiert werden, welche die zu untersuchende Eigenschaft besitzt und andere wiederum nicht. Die Shadow Modelle werden genutzt, um einen Meta-Klassifikator zu trainieren, welcher bestimmt, ob die Eigenschaft in der genutzten Datenmenge vorkommt oder nicht. Wird dies auf das originale Modell angewendet, kann ein Angreifer Aussagen zu dieser Eigenschaft treffen (Kapitel 2.4).

Die Membership Inference Attacke versucht herauszufinden, ob ein spezifischer Datensatz für das Training eines Modells genutzt wurde. Dafür werden ebenfalls Shadow Modelle genutzt, anhand dessen Vorhersagewahrscheinlichkeiten ein Meta-Klassifikator trainiert werden kann. Dieser bestimmt, ob der spezifische Datensatz in der Trainingsdatenmenge eines Modells ist. Alleine dieser Fakt kann sensibel sein (Kapitel 2.5).

Sprachmodelle, welche das nächste Wort vorhersagen, sind besonders von der Data Extraction Attacke betroffen. Durch lange, fortlaufende Nutzung eines Sprachmodells wird

eine Menge aus Tokenabfolgen gesammelt, welche jeweils vom Modell vorhergesagt werden. In diesen Tokenabfolgen können sich sensible Informationen befinden, welche vom Modell gelernt wurden. Durch effizientere Sammelmethoden, wie die Anpassung der Eingabetexte oder die Veränderung der Vorhersagewahrscheinlichkeiten, kann die Anzahl an gesammelten, sensiblen Information sogar erhöht werden (Kapitel 2.6).

Bei der Poisoning Attacke werden manipulierte Daten zu einer Datenmenge hinzugefügt, welche dafür sorgen, dass das Training des Modells sabotiert wird. Das resultierende Modell wird verschlechtert oder sogar unbrauchbar gemacht. Werden die manipulierten Daten jedoch bewusst gewählt, kann es sein, dass andere Angriffe wie die Membership Inference Attacke wesentlich effektiver werden (Kapitel 2.7).

Verteiltes Lernen bringt einige besondere Herausforderungen mit sich. Durch das Teilen von beispielsweise Gradienten, können Teilnehmer Rückschlüsse auf Eingabedaten ziehen, was Membership Inference und Property Inference Attacken erleichtert. Zusätzlich lassen sich mit den geteilten Gradienten auch ganze Datensätze rekonstruieren (Kapitel 2.8).

Die in diesem Kapitel besprochenen Angriffe zeigen, dass Machine Learning Modelle wie neuronale Netze ein potenzielles Sicherheitsrisiko für die Vertraulichkeit von Daten birgt. Dieses Risiko ist besonders hoch, wenn der zum Trainieren genutzte Datenbestand private und sensible Informationen enthält. Um dieses Risiko zu minimieren oder gar zu neutralisieren, gibt es eine Reihe an Methoden, welche die Vertraulichkeit in neuronalen Netzen sichern. Im Folgenden werden diese Methoden genauer betrachtet.

Kapitel 3

Methoden zur Sicherung der Vertraulichkeit

Neben den beschriebenen Angriffen aus Kapitel 2 gibt es eine Vielzahl an Methoden, diese Angriffe abzuschwächen oder sogar ganz zu unterbinden. Dieses Kapitel stellt zuerst eine typische Pipeline zum Training eines neuronalen Netzes vor und teilt diese in drei Phasen ein. Phase eins ist dabei die Aufbereitung des Datenbestands, also jegliche Verarbeitung der Daten, die vor dem eigentlichen Training stattfinden. Anschließend folgt Phase zwei, welche das Training des Modells umfasst. Das Ende der Pipeline, Phase drei, beinhaltet das Deployment so wie den Betrieb des Modells. Die Gegenmaßnahmen werden jeweils der entsprechenden Phase untergeordnet.

3.1 Übersicht der Pipeline

Abbildung 3.1 zeigt den Aufbau der typischen Pipeline, um ein neuronales Netz zu trainieren. Diese wird im Folgenden genauer beschrieben. Schritt 1 umfasst die Vorverarbeitung der Daten. Es wird angenommen, dass die Daten bereits gesammelt sind und in einer Datenbank abgelegt sind. Die Vorverarbeitung der Daten ist recht individuell und kann je nach Anwendungsfall variieren. Typische Handlungen sind:

- Vereinheitlichung des Datenformats
- Entfernen von starken Ausreißer und Fehlern
- Normalisierung der Daten
- Erstellung neuer Daten durch Transformationen
- Kodierung von kategorialen Variablen und Labels
- Aufteilung in Trainings-, Validierungs- und Testdatenmengen

Abbildung 3.1 stellt die Vorverarbeitung in einem Schritt dar, bei welchem mehrere Datenquellen verbunden werden und nur ein Dokument übrig bleibt. Dieses eine Dokument soll zeigen, dass nur die wichtigen Informationen der Daten erhalten bleiben, jedoch kann es in der Praxis sein, dass die Datenmenge beim Aufbereiten der Daten größer wird. Kapitel 3.3 stellt verschiedene Methoden vor, die bei der Vorverarbeitung der Daten angewendet werden können, um die Vertraulichkeit zu sichern.

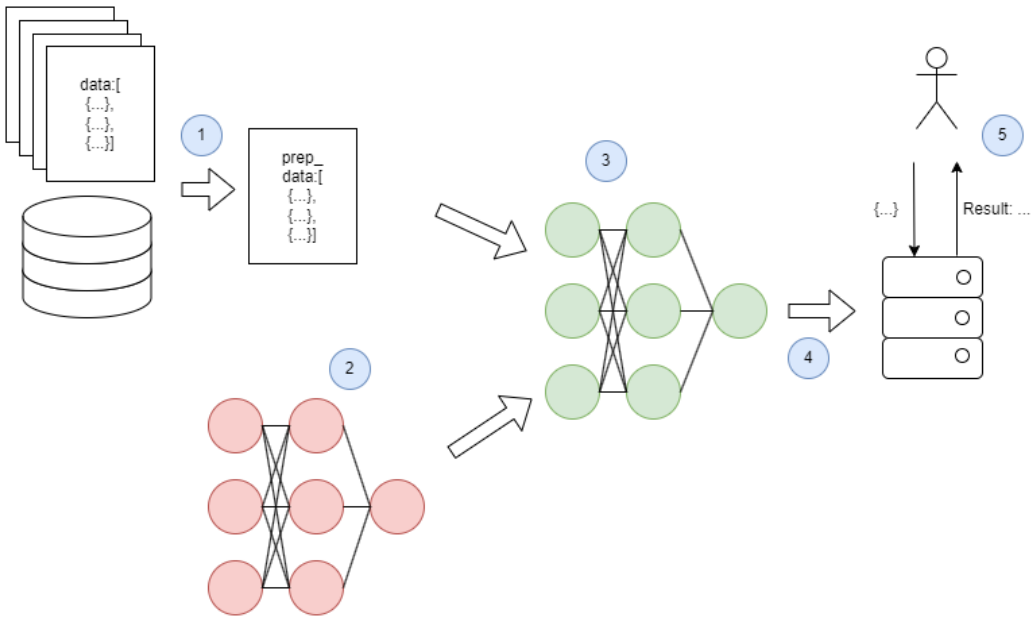


Abbildung 3.1: Training eines neuronalen Netzes

Schritt 2 umfasst die Architektur eines Modells. Je nach Anwendungsfall und Komplexität der Aufgabe werden unterschiedliche Konfigurationen der einzelnen Schichten eines neuronalen Netzes vorgenommen. Dabei werden die Gewichte zufällig oder anhand einer definierten Verteilung initialisiert. In der Abbildung wird dies durch ein rotes Modell angezeigt, da die Gewichte des Modells noch nicht trainiert sind und demnach keine sinnvollen Vorhersagen getroffen werden können. Schritt 3 ist der tatsächliche Trainingsvorgang, bei dem das untrainierte Modell lernt. Grob lässt sich das Training in folgende Schritte einteilen, die mehrfach mit jedem Datenpunkt durchgeführt werden können:

1. **Forward-Pass:** Ein Datensatz oder mehrere gebündelte Datensätze, auch Batch genannt, werden durch das Modell gegeben, wodurch eine Vorhersage berechnet wird. Dies wird auch Inferenz genannt.
2. **Backpropagation:** Die Abweichung von der tatsächlichen Vorhersage zu dem Label des Datensatzes wird mittels einer Verlustfunktion quantifiziert. Mittels dieses Wertes der Verlustfunktion lassen sich Gradienten für die Gewichte des neuronalen Netzes berechnen. Diese bestimmen die Richtung der Anpassung der aktuellen Gewichte, wohingegen die Lernrate die Intensität der Anpassung bestimmt.

Ein Durchgang der obigen Schritte mit jedem Datensatz der Trainingsdatenmenge wird dabei Epoche genannt. Um den Trainingsfortschritt zu beobachten, kann eine Epoche zusätzlich auch eine Evaluierung mittels einer Validierungsdatenmenge enthalten. In der Praxis können zusätzliche Schritte in das Training einer Epoche integriert werden, wie ein erweitertes Logging oder eine zusätzliche Überwachung. Die optimale Anzahl an Epochen ist dabei für jedes Modell unterschiedlich. Grafisch wird das Training in Abbildung 3.1 durch

den Verbund der Datenmenge und des untrainierten, roten Modells dargestellt. Es entsteht ein grünes Modell, welches in der Lage ist, sinnvolle Vorhersagen zu erzeugen. Kapitel 3.3 widmet sich Methoden, die während des Trainings, welche aus Schritt 2 und 3 besteht, angewendet werden.

Schritt 4 und 5 beschreiben das Deployment und den Betrieb des Modells. Das Ziel dieser Schritte ist es, ein trainiertes neuronales Netz für die vorgesehenen Anwender zur Verfügung zu stellen. Je nach Anwendungsfall kann die Systemarchitektur unterschiedlich aussehen. Einige Modelle werden mit Trainingscode auf öffentlichen Plattformen, wie HuggingFace[1], geteilt, wohingegen andere Modelle, wie ChatGPT[2], nur über eine spezielle Oberfläche erreichbar sind. Abbildung 3.1 zeigt deshalb lediglich einen Anwender, der direkt einen Request an das Modell schickt. In Kapitel 3.4 werden Maßnahmen besprochen, die in dieser Phase die Vertraulichkeit sichern. Dabei kann es sich um Transformationen des Modells vor dem Deployment handeln, oder um Mechanismen, die Anfragen und Antworten des Modells abwandeln.

3.2 Aufbereitung des Datenbestands

Bereits beim ersten Schritt einer Machine Learning Pipeline, dem Vorverarbeiten der Daten, können diverse Methoden genutzt werden, um die Vertraulichkeit zu wahren. In diesem Kapitel werden drei Kategorien dieser Methoden vorgestellt, und wie diese genutzt werden können, um bereits vor dem eigentlichen Training Modelle sicherer zu machen.

3.2.1 Anonymisierung

Bei der Anonymisierung von Daten geht es darum, identifizierende Eigenschaften zu entfernen oder unkenntlich zu machen. Dabei soll dennoch ein gewisser Nutzen erhalten bleiben.

Sweeney [37] stellt eine Methode der Anonymisierung namens k-Anonymität, im Englischen k-Anonymity, vor. Die Methode wird im folgenden mittels eines Auszugs der Titanic Datenmenge [38] erläutert. Tabelle 3.1 zeigt einen Auszug aus dieser Datenmenge. Die hier ausgewählten Spalten enthalten beschreibende Merkmale einer Person, sowie Informationen über die Reise auf der Titanic. Zur Verdeutlichung wird angenommen, dass es sich bei dem Einstiegsort um eine private Information handelt, die den Wohnort einer Person verraten könnte.

[1] https://huggingface.co/models
[2] https://chat.openai.com/

Name	Geschlecht	Alter	Buchungsklasse	Einstiegsort
Owen Harris Braund	m	22	3	S
Florence Briggs Thayer	w	38	1	C
Laina Heikkinen	w	26	3	S
Lily May Peel	w	35	1	S
William Henry Allen	m	35	3	S
Anna McGowan	w	15	3	Q
James Moran	m	30	3	Q
Adele Achem Nasser	w	14	2	C
Don Manuel Uruchurtu	m	40	1	C
Elizabeth Anne Wilkinson	w	29	2	S
Henry Birkhardt Harris	m	45	1	S

Tabelle 3.1: Nicht-Anonymisierter Titanic Datensatz [38]

Bei k-Anonymität werden die Attribute in 3 separate Klassen eingeteilt: Identifikatoren, Quasi-Identifikatoren und sensible Attribute. Bei diesem Beispiel wäre der Name der einzige Identifikator, da über diesen eine Person eindeutig zugeordnet werden kann. Quasi-Identifikatoren sind alle Attribute, welche Information über einen Datenpunkt preisgeben, aber nicht direkt auf diesen schließen lassen. Um mit Quasi-Identifikatoren einen Datensatz zu identifizieren, braucht es in der Regel mehr Informationen, beispielsweise einen zusätzlichen Datenbestand. In diesem Beispiel könnte also jede Spalte ein Quasi-Identifikator sein. Da der Name bereits ein direkter Identifikator ist, wird dieser dort zugeordnet. Die dritte Klasse sind die sensiblen Attribute, die es zu sichern gilt. Hier gehen wir davon aus, dass der Einstiegsort eine schützenswerte Information ist. Um k-Anonymität zu erreichen, muss jede Kombination aus Quasi-Identifikatoren mindestens k mal vorkommen, wobei k festgelegt werden kann. Ein größeres k sorgt für mehr Privatsphäre. Um dies zu erreichen, können die Quasi-Identifikatoren gruppiert werden, so kann beispielsweise anstatt des Alters, ein Zahlenbereich als Altersgruppierung dienen. Tabelle 3.2 zeigt, wie eine mögliche Gruppierung aussieht.

In Tabelle 3.2 ist zu sehen, dass die Identifikatoren, hier nur der Name, entfernt wurden. Geschlecht und Buchungsklasse sind unverändert geblieben. Die Gruppierung erfolgte anhand der Quasi-Identifikatoren, wobei das Alter durch einen Zahlenbereich ersetzt wurde. Hier ist anzumerken, dass die Spanne der Altersgruppen unterschiedlich groß ist. Je nachdem, wie diese Spannen aufgebaut sind, würden sich die Gruppierungen verändern. k-Anonymität mit $k = 2$ ist hier erfüllt, da jede Kombination von Quasi-Identifikatoren mindestens 2 Mal vorkommt. Dabei ist es auch möglich, dass einzelne Einträge redundant vorkommen. So sind in Tabelle 3.2 die ersten beiden Einträge identisch, obwohl diese von 2 unterschiedlichen Personen stammen.

Machanavajjhala et. al. [39] zeigen anhand von zwei Angriffen, dass k-Anonymität nicht ausreichend ist. Bei der Homogenitätsattacke kann die Eigenschaft, dass sensible Attribute

Identifikator	Quasi-Identifikatoren			sensibles Attribut
Name	Geschlecht	Alter	Buchungsklasse	Einstiegsort
-	m	20 - 35	3	S
-	m	20 - 35	3	S
-	m	20 - 35	3	Q
-	w	15 - 30	3	S
-	w	15 - 30	3	Q
-	w	10 - 30	2	C
-	w	10 - 30	2	S
-	w	30 - 40	1	C
-	w	30 - 40	1	S
-	m	40 - 45	1	C
-	m	40 - 45	1	S

Tabelle 3.2: k-Anonymity Titanic Datensatz [37][38]

nicht einzigartig sind, ausgenutzt werden. Tabelle 3.3 zeigt abgewandelt die ersten Zeilen der Tabelle 3.2. Das sensible Attribut, der Einstiegsort, ist jedoch in dieser Quasi-Identifikatoren Kombination identisch. Sollte also eine Person männlich, zwischen 20 und 35 Jahren alt sein und in der Buchungsklasse 3 mitgefahren sein, so kennt man ebenfalls den Einstiegsort.

Identifikator	Quasi-Identifikatoren			sensibles Attribut
Name	Geschlecht	Alter	Buchungsklasse	Einstiegsort
-	m	20 - 35	3	S
-	m	20 - 35	3	S
-	m	20 - 35	3	S

Tabelle 3.3: Angreifbare Abwandlung von Tabelle 3.2

Ein weiteres Problem ist ein Angriff mit Hintergrundwissen, bei dem gewisse sensible Attribute ausgeschlossen werden können oder zumindest unwahrscheinlicher ausfallen. In diesem Beispiel könnte es also sein, dass ein Angreifer weiß, wann ein Passagier zugestiegen ist und an welchem Hafen das Schiff zu dieser Zeit war. Damit können Rückschlüsse auf den Einstiegsort gemacht werden.

Aufgrund dieser beiden Angriffe, schlagen Machanavajjhala et. al. [39] eine Erweiterung mit dem Namen l-Diversität, im Englischen l-Diversity, vor. Dabei wird k-Anonymität auf die Daten angewendet und zusätzlich eine Bedingung eingeführt. Diese kann sowohl für einen Block, eine einheitliche Kombination der Werte der Quasi-Identifikatoren, als auch für die ganze Datenmenge gelten. Ein Block ist dabei l-divers, wenn die Werte des sensiblen Attributes *"gut repräsentiert"* sind, wobei l einer Zahl zugeordnet werden kann. Ist jeder Block

des Datensatzes l-divers, dann ist auch der ganze Datensatz l-divers. Dabei gibt es 3 Grundvarianten laut Machanavajjhala et. al. , wie *"gut repräsentiert"* definiert werden kann [39]:

- **Unterscheidbare l-Diversität:** Bei dieser Variante hat ein Block l unterschiedliche Werte eines sensiblen Attributes. Ein Block ist daher immer mindestens 1-divers, da dies bedeutet, dass der Wert eines sensiblen Attributs immer den gleichen Wert annimmt.
- **Entropie l-Diversität:** Hier wird die Entropie der sensiblen Attribute eines Blocks berechnet. Dabei ist ein Block l-divers, wenn die Entropie $\geq \log(l)$ ist. Folglich ist 1-Diversität dabei immer gegeben.
- **Rekursive (c,l)-Diversität:** Diese Definition besagt, dass der häufigste sensible Attributwert eines Blocks, seltener vorkommt, als die Anzahl der restlichen Attributwerte, multipliziert mit einem konstanten Wert c. Folglich darf kein sensibles Attribut zu oft vorkommen. Ein Block ist dabei (c, l)-divers, wenn $l-1$ verschiedene einzigartige Attribute entfernt werden können und die Bedingung immer noch erfüllt ist.

Je nach Datenmenge, kann es Sinn ergeben, einige Ausnahmen zu erlauben. So könnte es sein, dass ein Datensatz von einem Attributwert dominiert wird, die jedoch keine Verletzung der Privatsphäre darstellt. Ein Beispiel der Autoren ist, wenn eine Kardiologie preisgibt, dass die meisten Patienten eine Herzkrankheit haben.

Sollte ein Datensatz mehrere sensible Attribute besitzen, so muss l-Diversität für jede dieser Attribute gelten. Für diese Überprüfung, werden jeweils alle anderen Spalten, auch die sensiblen Attribute, als Quasi-Identifikator angesehen.

Li et. al. [40] zeigen, dass l-Diversität zwei Angriffsflächen bietet. Die erste Angriffsfläche ergibt sich, wenn die Verteilung des sensiblen Attributs sehr stark links- oder rechtsschief ist. Die Autoren zeigen ein Beispiel, bei der das sensible Attribut eine Infektion mit einem bestimmten Virus abbildet. Dabei sind 99 % der Personen gesund und lediglich 1 % der Personen infiziert. Die Verteilung des Attributs ist stark schief. Hat jetzt ein Block, welcher durch k-Anonymität entsteht, eine 50 % Aufteilung beider Werte, so wäre dieser Block l-divers mit $l = 2$. Kann man jedoch eine Person diesem Block zuordnen, so wäre dies ein Informationsgewinn, da besagte Person ein überdurchschnittliches Risiko der Infektion besitzt. Die zweite Angriffsfläche entsteht dadurch, dass l-Diversität nicht berücksichtigt, ob die Werte des Attributes eine ähnliche Bedeutung haben. Bei einem Krankheitsbeispiel könnten die Werte alle unterschiedliche Krankheiten annehmen, die jedoch den gleichen Körperteil betreffen. Diese Angriffsfläche ähnelt der Homogenitätsattacke gegen k-Anonymität, bloß dass hier zusätzlich Werte semantisch verbunden werden können.

Aufgrund dieser beiden Angriffsflächen stellen Li et. al. [40] ein neues Maß an Sicherheit vor: t-Nähe, im Englischen t-Closseness. Ziel dieses Maßes ist es, zu zeigen, dass die Verteilung eines sensiblen Attributes in einem einzelnen Block ähnlich zu der Verteilung des

gleichen Attributes im gesamten Datensatz ist. Der Unterschied zwischen den beiden Verteilungen soll kleiner als ein Grenzwert t sein. Die Autoren prüfen verschiedene Verfahren der Distanzmessung der Verteilungen und favorisieren die sogenannte Earth-Mover-Distanz. Dabei handelt es sich um eine Metrik zweier Verteilungen, welche die minimale Arbeit berechnet, die nötig ist, um eine Verteilung zu der anderen Verteilung zu transformieren, indem Werte innerhalb der Verteilung verschoben werden. Die Metrik liegt immer im Wertebereich (0,1) wodurch diese auch vergleichbar ist. Ein Wert nahe 0 ist dabei besser. Mathematisch gesehen, handelt es sich um ein Optimierungsproblem, jedoch gehen die Autoren auf zwei unterschiedliche Arten von Attributen ein, numerische und kategoriale, um zu zeigen, wie die Earth-Mover-Distanz berechnet wird. Um die Distanz für numerische Werte berechnet zu werden, werden diese zuerst sortiert. Sofern es sich um eine ungleiche Anzahl an Werten handelt, können Werte mehrfach genutzt werden. Anschließend wird die durchschnittliche, normalisierte Differenz zwischen den Werten an gleicher Stelle beider sortierten Verteilungen berechnet. Im Folgenden wird eine Beispielrechnung exerziert, welches ein sensibles Attribut, Stundenlohn in Euro, darstellt. Verteilung 1 ist dabei das sortierte Gehalt eines Blockes nach k-Anonymität und Verteilung 2 das Gehalt des gesamten Datensatzes:

$$\text{Verteilung 1} = \{20, 30, 40\}$$
$$\text{Verteilung 2} = \{20, 25, 25, 30, 35, 35, 35, 40, 40\}$$

Da Verteilung 2 dreimal so viele Elemente enthält wie Verteilung 1, wird jedes Element dreimal genutzt. Dadurch entsteht:

$$\text{Verteilung 1'} = \{20, 20, 20, 30, 30, 30, 40, 40, 40\}$$

Die größte Differenz ist $40 - 20 = 20$, somit wird der Betrag jeder Differenz durch 20 dividiert, dass diese jeweils im Wertebereich (0,1) liegen. Werden jetzt die einzelnen Wertepaare verglichen, so ergibt sich folgend Distanz:

$$(20-20)+(20-25)+(20-25)+(30-30)+(30-35)+(30-35)+(40-35)+$$
$$(40-35)+(40-40)=-10$$

Der durchschnittliche, normalisierte Wert dieser Distanz, ist die gesuchte Earth-Mover-Distanz:

$$1/9 \times |-10| \div 20 = 0,056$$

Damit hat dieser Block eine 0,056-Nähe. Dies bedeutet, dass wenn eine Person diesem Block zugeordnet wird, dennoch kaum Informationsgewinnung möglich ist.

Bei kategorialen Werten ist es schwieriger, eine Differenz zu bilden. Es gibt die Möglichkeit den Wert 1 zuzuweisen, wenn die beiden Kategorien unterschiedlich sind und den Wert 0, sofern beide gleich sind. Dies würde jedoch bedeuten, dass semantische Ähnlichkeiten

der Werte nicht berücksichtigt werden. Eine Alternative wäre es, alle möglichen Werte semantisch in einer Art Baumstruktur zu gliedern. Bei Krankheiten wäre beispielsweise die Wurzel "*Krankheit*", die Nachfolger wären dann gewisse Systeme des Körpers wie beispielsweise "*Herz-Kreislaufsystem*" und "*Verdauungssystem*". Die Distanz ist nun die Anzahl der Schritte, die benötigt wird, um die Werte zu verbinden. Zwei unterschiedliche Herzkrankheiten sind über einen Schritt mittels "*Herz-Kreislaufsystem*" verbunden, wohingegen eine Herzkrankheit und eine Darmkrankheit über 2 Schritte mittels der Wurzel "*Krankheit*" verbunden wären.

3.2.2 Differential Privacy

Differential Privacy ist eine Technik, welche 2006 von Cynthia Dwork [41] vorgestellt wurde. Ziel dabei ist es, Zugriff auf eine Datenmenge zu ermöglichen, welche sowohl nützliche Erkenntnisse zulässt, als auch die Privatsphäre eines einzelnen Datensatzes schützt.

Differential Privacy kann dabei an 3 unterschiedlichen Stellen der Machine Learning Pipeline genutzt werden:

- **Verfremdung der Trainingsdaten:** Diese Methodik wird folgend in diesem Kapitel erläutert.
- **Trainingsalgorithmus:** Kapitel 3.3.1 beschreibt, welche Anpassungen am Trainingsalgorithmus vorgenommen werden können, um Differential Privacy zu gewährleisten.
- **Vorhersage des Modells:** Bevor die Vorhersage des Modells weitergeleitet wird, könnte diese verrauscht werden. Kapitel 3.4 stellt dieses Vorgehen vor.

Differential Privacy [41] sorgt dafür, dass ein Mechanismus, auch Abfrage oder Funktion, welche eine Menge an Datensätzen als Eingabe akzeptiert, keinen konstanten Wert mehr zurückgibt. Stattdessen wird dem Mechanismus zufälliges Rauschen mit einer festgelegten Intensität hinzugefügt. Der Mechanismus gibt also eine Stichprobe einer Verteilung zurück, bei der das tatsächliche Ergebnis dem Erwartungswert entspricht. Demnach können eindeutige Feststellungen über Eigenschaften nicht getroffen werden. Ziel des Verrauschens ist es, dass wenn der Mechanismus auf zwei Datenmengen, die sich in einem Datensatz unterscheiden, ausgeführt wird, die Ergebnisse sich maximal um einen Faktor e^{ϵ} unterscheiden. Demnach ist der Einfluss eines Datensatzes auf eine Berechnung mit der gesamten Datenmenge quantifizierbar und kann begrenzt werden. Dabei kann der Wert ϵ, welcher auch Privacy Budget genannt wird, festgelegt werden.

Formal lautet die Definition von ϵ-Differential Privacy wie folgt [41]:
Ein randomisierter Mechanismus M, welche eine Menge an Datensätzen D auf einen Wer-

tebereich R abbildet, weist ϵ-Differential Privacy auf, wenn für alle Mengen an Datensätze D_1 und D_2 die sich in höchstens einem Datensatz unterscheiden $||D_1 - D_2||_1 \leq 1$, gilt:

$$P[M(D_1) \in R] \leq e^{\epsilon} \times P[M(D_2) \in R] \tag{3.1}$$

P beschreibt dabei die Wahrscheinlichkeit, dass der Erwartungswert gezogen wird.

Es gibt unterschiedliche Algorithmen, um das Ergebnis eines Mechanismus zu verrauschen. Diese werden im späteren Verlauf des Kapitels genauer beleuchtet.

Dwork und Roth [42] fügten der Definition noch einen Parameter δ hinzu, welcher erlaubt, dass die Bedingungen von ϵ-Differential Privacy zu einem definierten Grad verletzt werden können. Der Wert von δ sollte dabei niedriger sein, als die Inverse der Anzahl an Datensätzen im Datenbestand. Die damit angepasste Definition von (ϵ,δ)-Differential Privacy lautet [42]:
Ein randomisierter Mechanismus M, welche eine Menge an Datensätzen D auf einen Wertebereich R abbildet, erfüllt (ϵ,δ)-Differential Privacy, wenn für alle Mengen an Datensätze D_1 und D_2 die sich in höchstens einem Datensatz unterscheiden $||D_1 - D_2||_1 \leq 1$, gilt:

$$P[M(D_1) \in R] \leq e^{\epsilon} \times P[M(D_2) \in R] + \delta \tag{3.2}$$

Konkret sagen die Definitionen aus, dass eine randomisierte Funktion M, auf zwei Datenbeständen D_1 und D_2 mit maximal einem unterschiedlichen Datensatz $||D_1 - D_2||_1 \leq 1$, jeweils Stichproben einer Verteilung ausgibt, wobei sich die Verteilungen nur um den Faktor ϵ und den Summand δ unterscheiden dürfen. Dabei bestimmt das Privacy Budget, ϵ und δ, wie stark sich die Ergebnisse unterscheiden dürfen. Wie diese beiden Werte konfiguriert werden können, hängt dabei vom Algorithmus des Rauschens ab.

Konfiguration des Privacy Budgets

Der δ-Wert wird oftmals als konstanter Wert festgelegt, wobei dieser kleiner als die Inverse der Anzahl an Datensätzen im Datenbestand sein sollte [42]. Die Wahl von ϵ ist daher entscheidend. Abbildung 3.2 zeigt den Einfluss des ϵ-Werts auf die Nützlichkeit und die Vertraulichkeit von Mechanismen. Kleine Werte für ϵ, also ein kleines Privacy Budget, bedeutet dabei, dass die Differenz eines Mechanismus durch einen zusätzlichen Datensatz, sich weniger stark verändern kann, was für einen besseren Schutz der Privatsphäre sorgt. Jedoch wirkt sich dies negativ auf die Nützlichkeit der Abfragen aus, da kleine Privacy Budgets für ein großes Rauschen sorgen, was öfters falsche Ergebnisse liefert. Dies bedeutet, dass es keinen optimalen Wert für ϵ gibt, sondern dieser für jeden Use Case mittels einer Abwägung zwischen Sicherheit und Nützlichkeit, neu bestimmt werden muss. Nutzt ein Modell beispielsweise nur öffentliche, unsensible Daten, ergibt es keinen Sinn einen niedrigen ϵ-Wert

festzulegen, da die Vertraulichkeit der Daten nicht entscheidend ist. Werden jedoch hochsensible Daten genutzt, kann es sein, dass die Sicherheit der Daten wichtiger eingestuft wird, als die Nützlichkeit des Modells. Hier empfiehlt sich ein niedriger ϵ-Wert. Ein ϵ-Wert von Unendlich bedeutet, dass sich die Ergebnisse von Mechanismen um beliebige Werte unterscheiden dürfen, weshalb kein Rauschen notwendig wäre. Dies ist bei Mechanismen ohne Differential Privacy bereits der Fall.

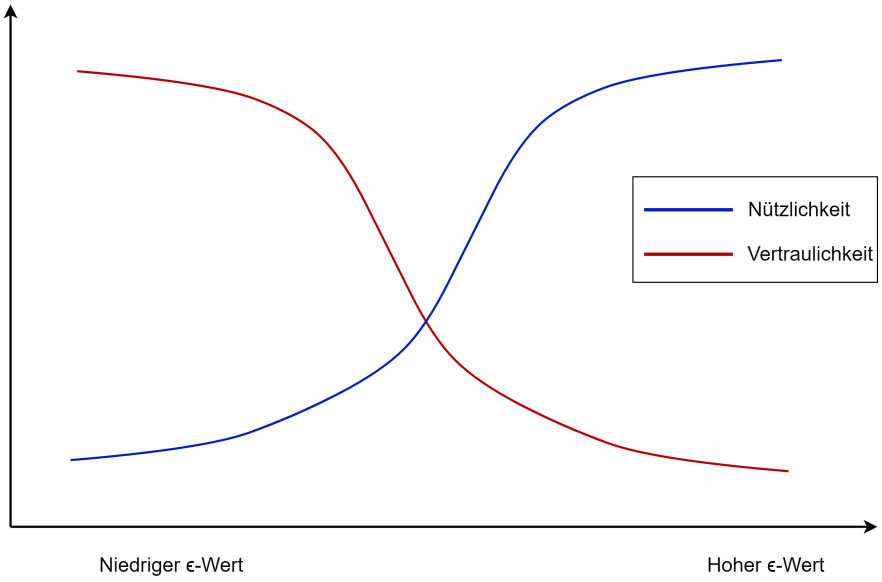


Abbildung 3.2: Einfluss von ϵ

Beispiel für Differential Privacy

Abbildung 3.3 zeigt, wie sich Abfragen mit und ohne Differential Privacy unterscheiden. Dabei führt ein Angreifer zunächst eine Abfrage über das Durchschnittsgehalt eines Unternehmens auf einem Datenbestand ohne Differential Privacy aus. Er erhält hier einen konkreten Wert als Ergebnis. Anschließend wird der Datenbestand um einen Eintrag erweitert, weil das Unternehmen einen neuen Mitarbeiter hat. Führt der Angreifer die Abfrage um das Durchschnittsgehalt erneut aus, erhält dieser einen angepassten, neuen Wert. Dadurch kann er anhand der Anzahl der Mitarbeiter des Unternehmens und der Differenz der beiden Abfragen das exakte Gehalt des neuen Mitarbeiters bestimmen. Anders sieht dies mit Differential Privacy aus. Dabei werden beide Abfragen mit zufälligem Rauschen angereichert. Es wird also nur eine Stichprobe einer Verteilung zurückgegeben. Bei einer gleichen Abfrage auf den gleichen Datenbestand, können also ebenfalls unterschiedliche Werte zurückgegeben werden. Führt der Angreifer zwei Abfragen aus, einmal auf dem alten Datenbestand und auf dem Datenbestand mit dem neuen Mitarbeiter, kann er keinen exakten Wert des Gehalts des neuen Mitarbeiters ermitteln. Differential Privacy bietet einige Algorithmen, wie dieses

Rauschen hinzugefügt werden kann, wobei Parameter ϵ und δ die Stärke des Rauschens bestimmen und damit auch, wie nah die beiden Ergebnisse aneinander sind.

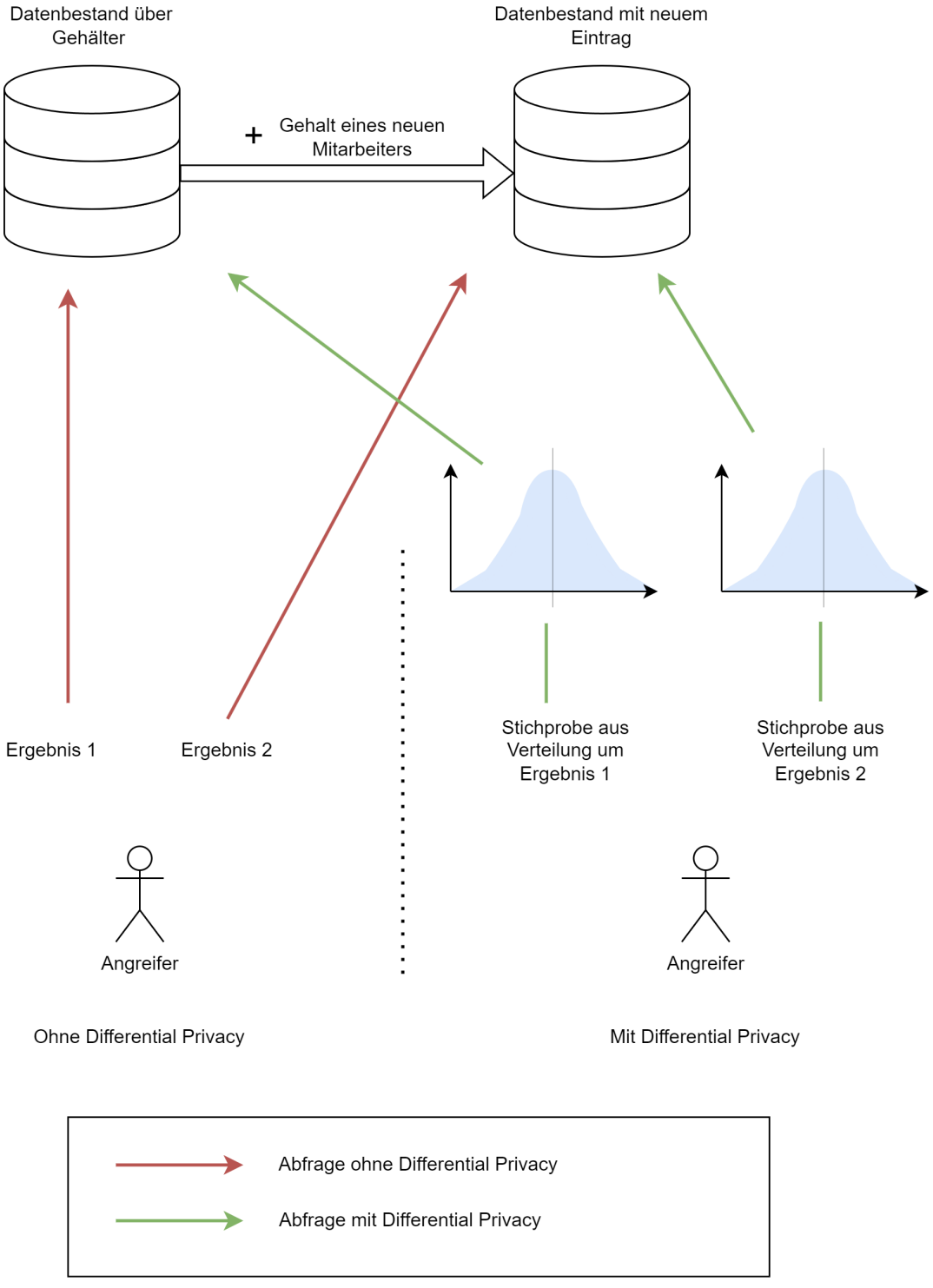


Abbildung 3.3: Beispiel Differential Privacy

Algorithmen für Differential Privacy

Die Art des Rauschens, welche über die Ausgabe einer Abfrage gelegt wird, kann unterschiedlichster Herkunft sein. Dwork und Roth [42] stellen eine ganze Reihe dieser Mechanismen vor, die im Folgenden beschrieben werden.

Bei der Technik der randomisierten Antwort, im Englischen Random Response, wird mit einer festgelegten Wahrscheinlichkeit, ein falsches Ergebnis ausgegeben. Dies entspricht dem Rauschen dieser Methode. Wird beispielsweise eine Person gefragt, ob sie eine gewisse Aktivität durchgeführt hat, wird mit 25-prozentiger Wahrscheinlichkeit die falsche Antwort selektiert. Ist die Wahrheit *"Ja"*, dann gilt $P[Antwort = "Ja" \mid Wahrheit = "Ja"] = 3/4$ und $P[Antwort = "Nein" \mid Wahrheit = "Ja"]$. Gleiches gilt, wenn die Wahrheit *"Nein"* wäre. Die jeweils ausgegebene Antwort kann dadurch glaubhaft abgestritten werden.

$$\frac{P[Antwort = "Ja" \mid Wahrheit = "Ja"]}{P[Antwort = "Ja" \mid Wahrheit = "Nein"]} = \frac{3/4}{1/4} = \frac{P[Antwort = "Nein" \mid Wahrheit = "Nein"]}{P[Antwort = "Nein" \mid Wahrheit = "Ja"]} = 3 \tag{3.3}$$

Formel 3.3 zeigt, dass eine Antwort, egal ob diese *"Ja"* oder *"Nein"* lautet, mit einem Verhältnis von 3 abgestritten werden kann. Der Faktor, um den sich die Wahrscheinlichkeit einer Antwort unterscheidet, wenn sich die Wahrheit ändert, liegt demnach auch bei 3. Daraus resultiert, dass die Technik der randomisierten Antwort, mit einer 25-prozentigen Wahrscheinlichkeit einer falschen Antwort, eine ($ln3$, 0)-Differential Privacy unterstützt. Der natürliche Logarithmus ln muss hier genutzt werden, da die Definition von Differential Privacy die Exponentialfunktion als Faktor betrachtet. Demnach erhält man durch $\epsilon = ln3$ einen Unterschied um den Faktor $e^{ln3} = 3$.

Eine weitere Technik für Differential Privacy ist der Laplace-Mechanismus. Dabei wird das Rauschen, welches über das Ergebnis der Abfrage gelegt wird, aus einer Laplace-Verteilung ermittelt. Eine Bedingung ist dabei, dass es sich bei dem Ergebnis der Abfrage um einen Zahlenwert handelt. Die Dichtefunktion der Laplace-Verteilung, zentriert um den Erwartungswert $\mu = 0$, mit dem Skalenparameter σ, lautet [42]:

$$Lap(x|\sigma) = \frac{1}{2\sigma} \times e^{(-\frac{|x|}{\sigma})} \tag{3.4}$$

wobei σ die Steigung der Funktion beeinflusst. Um nun ein geeignetes σ zu wählen, muss zuerst ein Wert ermittelt werden, um den sich eine Funktion bei Änderung eines Datenpunktes maximal unterscheiden kann. Diese sogenannte Sensitivität wird als Δf notiert. Geht es beispielsweise um eine Anzahl an Datensätzen, so würde ein neuer Datensatz die Anzahl um den Wert 1 erhöhen. Folglich ist $\Delta f = 1$, denn die Anzahl wird sich durch einen neuen Datensatz um den Wert 1 verändern. Es gibt jedoch auch Fälle, in denen der Wert der

Sensitivität nicht eindeutig ist, beispielsweise bei dem obigen Gehaltsbeispiel. Das Gehalt einer neuen Person könnte für eine beliebige Veränderung des Durchschnittsgehalts sorgen, da theoretisch jedes Gehalt möglich wäre. Realistisch betrachtet gibt es jedoch einen Wertebereich, in dem sich alle Gehälter befinden. Dieser könnte beispielsweise von 0 Euro bis 10 Milliarden Euro reichen. Jedoch sollten die Grenzen möglichst nahe an den echten Grenzen liegen. 10 Milliarden Euro ist theoretisch möglich, jedoch ist eine Grenze von 200.000 Euro realistischer. Dieser Wertebereich muss also fachlich festgelegt werden. In diesem Beispiel wäre die Sensitivität $200.000/n$, wobei n die Anzahl der Mitarbeiter ist.

Um (ϵ,0)-Differential Privacy zu erreichen, muss die Ausgabe einer Abfrage M mit zufälligen Werte der Laplace-Verteilung $Lap(x|\Delta f/\epsilon)$ verrauscht werden [42]:

$$M(D) = f(D) + (Y_1, ...Y_k), \text{ mit } Y_i \sim Lap(x|\Delta f/\epsilon) \tag{3.5}$$

Eine Abwandlung des Laplace-Mechanismus ist es, anstatt der Laplace-Verteilung, eine Gaußverteilung zu nutzen. Die Dichtefunktion dieser zentriert um den Erwartungswert $\mu = 0$ lautet [42]:

$$Gauß(x|\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \times e^{-\frac{1}{2}(\frac{x}{\sigma})^2} \tag{3.6}$$

Der Gauß-Mechanismus liefert (ϵ,δ)-Differential Privacy, wenn $\sigma = \Delta f \frac{\ln(1/\delta)}{\epsilon}$ ist [42].

Neben den bereits beschriebenen Mechanismen gibt es noch den Exponential-Mechanismus. Bei diesem wird ein Element aus einer Menge ausgegeben, welches anhand einer Bewertungsfunktion ausgewählt wird. Die Abfrage an einen Datenbestand liefert also keine Zahl, sondern einen Datensatz oder Attributwert aus diesem. Der Exponential-Mechanismus benötigt eine Bewertungsfunktion $u : DxR \to \mathbb{R}$, welche für jede potenzielle Ausgabeoption R aus einer Menge an Datensätzen D, einen Nutzwert im Wertebereich $\mathbb{R}$ berechnet. Als Sensitivität Δf entspricht der Sensitivität der Bewertungsfunktion $\Delta f = \Delta u$ Die Anfrage erfüllt dabei (ϵ,0)-Differential Privacy, wenn der Mechanismus $M(D, u, R)$ ein Element $r \in R$ auswählt und ausgibt, mit einer Wahrscheinlichkeit proportional abhängig von

$$e^{\frac{\epsilon \times u(D,r)}{2\Delta u}} \tag{3.7}$$

Als Beispiel könnte ein Mechanismus genutzt werden, welcher ausgeben soll, ob Krankheit "A" oder "B" häufiger vorkommt. Somit wären die möglichen Optionen R={"A","B"}, die Bewertungsfunktion $u(D, r)$ = Count(r in D). Die Sensitivität ist $\Delta u = 1$, da ein zusätzlicher Datensatz die Zählung maximal um die Anzahl 1 verändern kann. Für jede der Optionen in R, würde die Ausgabewahrscheinlichkeit mit Formel 3.7 berechnet werden. Anschließend gibt der Mechanismus entweder "A" oder "B" zurück, abhängig der zuvor berechneten Aus-

gabewahrscheinlichkeiten. Das Rauschen des Exponential-Mechanismus ist demnach, dass nicht immer die Option $r \in R$ mit dem größten Nutzwert ausgegeben wird.

Eine alternative Möglichkeit, neben dem Exponential-Mechanismus, einen Wert aus einem Datensatz mittels Ausgabewahrscheinlichkeiten zu bestimmen, ist die sogenannte Report Noisy Max Methode. Dabei werden die echten Wahrscheinlichkeitswerte mittels Laplace-Mechanismus verrauscht und anschließend wird der Wert oder Datenpunkt mit der höchsten Wahrscheinlichkeit selektiert.

Eigenschaften von Differential Privacy

Das Ziel von Differential Privacy, die Vertraulichkeit eines Datensatzes zu schützen und dennoch die Nützlichkeit des Datenbestands zu bewahren, wurde bereits zu Beginn des Kapitels geschildert. Jedoch bringt Differential Privacy eine Reihe weiterer nützlicher Eigenschaften mit sich [42]:

- **Gruppen Privacy:** Differential Privacy betrachtet zwei Datenbestände, die sich in einem Datensatz unterscheiden. Jedoch gelten die gleichen Regeln für Datenbestände, die sich in mehreren Datensätzen unterscheiden, wobei ϵ dabei mit der Anzahl der unterschiedlichen Datensätze multipliziert wird.
- **Resistenz gegen Umkehrung bei der Weiterverarbeitung:** Das Ergebnis eines Mechanismus, welcher Differential Privacy nutzt, ist geschützt und es gibt keinen Mechanismus, welcher dies umkehren könnte. Somit sind geschützte Weiterverarbeitungen der Ergebnisse möglich.
- **Quantifizierung der Vertraulichkeit:** Mit den Parametern ϵ und δ kann angegeben werden, wie stark die Vertraulichkeit der Daten geschützt wird.
- **Bewertung zusammengesetzter Mechanismen:** Durch die Quantifizierung der Vertraulichkeit, können auch zusammengesetzte und parallele Berechnungen bewertet werden.

Die Bewertung von mehreren zusammengesetzten Mechanismen ist dabei komplex und kann für bestimmte Berechnungen verfeinert werden [42]. Der dabei berechnete ϵ-Wert ist eine Obergrenze, welcher durch bestimmte Restriktionen und Berechnungen noch genauer definiert werden kann. Grundlegend jedoch, besitzt ein Mechanismus, welcher aus nacheinander ausgeführten Teilmechanismen besteht, einen ϵ-Wert und δ-Wert, welcher der Summe aus den einzelnen Teilmechanismen entspricht. Erfüllt $M_1(D)$ eine (ϵ_1, δ_1)-Differential Privacy und $M_2(D)$ eine (ϵ_2, δ_2)-Differential Privacy, dann erfüllt der gesamte Mechanismus $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$-Differential Privacy. Wird ein und derselbe Teilmechanismus mit (ϵ, δ)-Differential Privacy t mal ausgeführt, so entspricht der Gesamtmechanismus $(t\epsilon, t\delta)$-Differential Privacy. Zusammengefasst lässt sich dies durch folgende Definition beschreiben [42]:

Wenn ein Mechanismus $M_{|t|}$, aus t Teilmechanismen M_i mit (ϵ_i, δ_i)-Differential Privacy besteht, dann erfüllt dieser $(\Sigma_{i=1}^{t}\epsilon_i, \Sigma_{i=1}^{t}\delta_i)$-Differential Privacy

Alternativ kann ein Mechanismus auch aus Teilmechanismen bestehen, welche jeweils nur auf einer disjunkten Menge an Datensätzen des gesamten Datenbestands ausgeführt wird. Der gesamte Mechanismus hat als ϵ und δ-Wert dabei die maximalen Werte der Teilmechanismen.
Wenn ein Mechanismus $M_{|t|}$, aus t Teilmechanismen M_i mit (ϵ_i, δ_i)-Differential Privacy besteht, welche jeweils nur auf einer disjunkten Menge an Datensätzes des gesamten Datenbestands D_i ausgeführt werden, dann erfüllt dieser Mechanismus $(max_i(\epsilon_i), max_i(\delta_i))$-Differential Privacy

Ein Beispiel für einen Algorithmus, welcher eine angepasste Bewertung von zusammengesetzten Mechanismen besitzt, ist der Trainingsalgorithmus DPSGD (Kapitel 3.3.1) [43]. Diese Anpassung sorgt dafür, dass der ϵ-Wert über den ganzen Trainingsprozess möglichst niedrig bleibt. Ziel davon ist es, eine Obergrenze für ϵ zu beschreiben, welche näher an den tatsächlichen Berechnungen liegt, als das Aufsummieren der einzelnen ϵ Werte.

Vorverarbeitung der Trainingsdaten

Um die Trainingsdaten eines Modells mit Differential Privacy zu schützen, ist es möglich, die Daten vor der Eingabe in das Modell zu verrauschen. Der Forward-Pass eines Modells wird demnach nicht auf den echten Daten ausgeführt, sondern auf den jeweils verrauschten Versionen der einzelnen Datensätze.

Um einen einzelnen Datensatz zu verrauschen, muss jedes Attribut des Datensatzes individuell betrachtet werden. Dabei ist die Sensitivität individuell zu setzen, beeinflusst von statistischen Merkmalen des Attributes. Ebenfalls kann sich der Differential Privacy Mechanismus unterscheiden, abhängig davon, ob es sich um kategoriale oder numerische Werten handelt. Bei numerischen Werten kann der Laplace-Mechanismus oder Gauß-Mechanismus gewählt werden, bei kategorialen Werten der Exponential-Mechanismus. Der ϵ-Wert und δ-Wert eines einzelnen Datenpunktes entspricht anschließend der Summe der Werte des einzelnen Verrauschens jeder Variable.

Da jeder Datensatz alleine eine disjunkter Menge des Datenbestands darstellt, gilt, dass der ϵ-Wert und δ-Wert dem Maximum des Rauschens auf die einzelnen Datensätze entspricht. Wird also jeder Datensatz mit dem gleichen, festgelegten Mechanismus mit (ϵ,δ)-Differential Privacy verrauscht, dann entspricht dies auch den Werten des gesamten Vorverarbeitungsprozess.

Alternativ kann ein ganzer Datenbestand mit Differential Privacy geschützt werden, indem dieser nur dazu genutzt wird, einen synthetischen Datenbestand zu erzeugen. Dieser kann

anschließend wie der originale Datenbestand genutzt werden. Das folgende Kapitel stellt einige Methoden dazu vor.

3.2.3 Synthetische Daten

Bei einem synthetischen Datenbestand handelt es sich um einen künstlich erzeugten Datenbestand, welcher die gleichen statistischen Merkmale wie der ursprüngliche Datenbestand enthält. Statistische Abfragen oder auch komplexere Modelle, wie neuronale Netze, sollen demnach bei beiden Datenbeständen vergleichbare Ergebnisse liefern.

Bisher wurde Differential Privacy für einzelne Abfragen genutzt. Es gibt jedoch auch Verfahren, welche es ermöglichen, eine synthetische Menge an Datensätzen zu veröffentlichen, welcher mit Differential Privacy geschützt werden. Hardt et. al. [44] stellen solch ein Verfahren vor, welches MWEM genannt wird. Dieses basiert auf Marginalverteilungen, auch Randverteilungen genannt, von verschiedenen Attributen zueinander. Dabei handelt es sich um Anzahlen verschiedener Attributwerte eines Attributs in Kombination mit den Anzahlen verschiedener Attributwerte eines anderen Attributes. Einzelne Werte der Marginalverteilungen werden auch Marginalhäufigkeiten genannt. Tabelle 3.4 zeigt, wie ein Beispiel von Marginalverteilungen zwischen den Attributen "A" und "B", welche jeweils zwei unterschiedliche Attributwerte annehmen können. Stetige Werte müssten vorher in verschiedene Klassen gruppiert werden.

	Attributwert "A_1"	Attributwert "A_2"	Marginalverteilung
Attributwert "B_1"	10	5	15
Attributwert "B_2"	13	4	17
Marginalverteilung	23	9	32

Tabelle 3.4: Beispiel Marginalverteilungen

Um eine Datenmenge umzuwandeln, wird eine Datenmenge D' mit der gleichen Anzahl an Elementen wie die ursprüngliche Datenmenge D initialisiert, wobei die Initialwerte der Attribute mittels einer Gleichverteilung über alle potenziellen Attributwerte des jeweiligen Attributes ermittelt werden. Das Verfahren besteht aus 3 Schritten, welche iterativ wiederholt werden können, um die künstliche Datenmenge D' an die echte Datenmenge D anzugleichen:

1. **Wahl einer Anfrage:** Aus allen Marginalhäufigkeiten der ursprünglichen Datenmenge, wird mittels Exponential-Mechanismus die Marginalhäufigkeit gewählt, die den größten Unterschied zwischen den Eingaben D' und D besitzt. Die Bewertungsfunktion u des Exponential-Mechanismus bewertet folglich die Differenzen der Marginalverteilungen der beiden Datenmengen (originale Datenmenge und synthetische Datenmenge).

2. **Verrauschen:** Mittels Laplace-Mechanismus wird nun das Ergebnis der gewählten Marginalverteilung aus Schritt 1 verrauscht.
3. **Anpassen von D':** Abhängig von dem verrauschten Ergebnis der gewählten Marginalverteilung, werden die Attributwerte von D' so angepasst, dass die Marginalverteilungen sich angleichen.

Nach Beendigung des Algorithmus, ist D' eine synthetische Datenmenge, welche die originale Datenmenge D mit Differential Privacy abbildet.

Die Erzeugung synthetischer Daten wurde im Jahre 2016 maßgeblich durch die Generative Adversarial Network Architektur, auch GAN genannt, beeinflusst [17]. Bei dieser Architektur gibt es zwei gegensätzliche neuronale Netzwerke, der Generator und der Diskriminator. Der Diskriminator lernt dabei, zwischen echten Daten und nicht echten Daten zu unterscheiden, welche abwechselnd von dem echten Datenbestand und von dem Generator kommen. Im Gegensatz dazu, versucht der Generator, Datensätze zu erzeugen, die den Diskriminator täuschen. Der Diskriminator kann dabei zwei Verlustfunktionen haben, einmal für sich selber und einmal für den Generator. Die Verlustfunktion des Generators kann dabei durch den Generator backpropagiert werden, wodurch dieser immer realistischere Datensätze generiert. Im besten Falle, erzeugt der Generator nach dem Training Datensätze, die so echt wirken, dass der Diskriminator diese nicht mehr unterscheiden kann. Folglich erzeugt der Generator synthetische Daten, welche die statistischen Strukturen der originalen Daten enthalten. Abbildung 3.4 zeigt die bereits beschriebene Architektur.

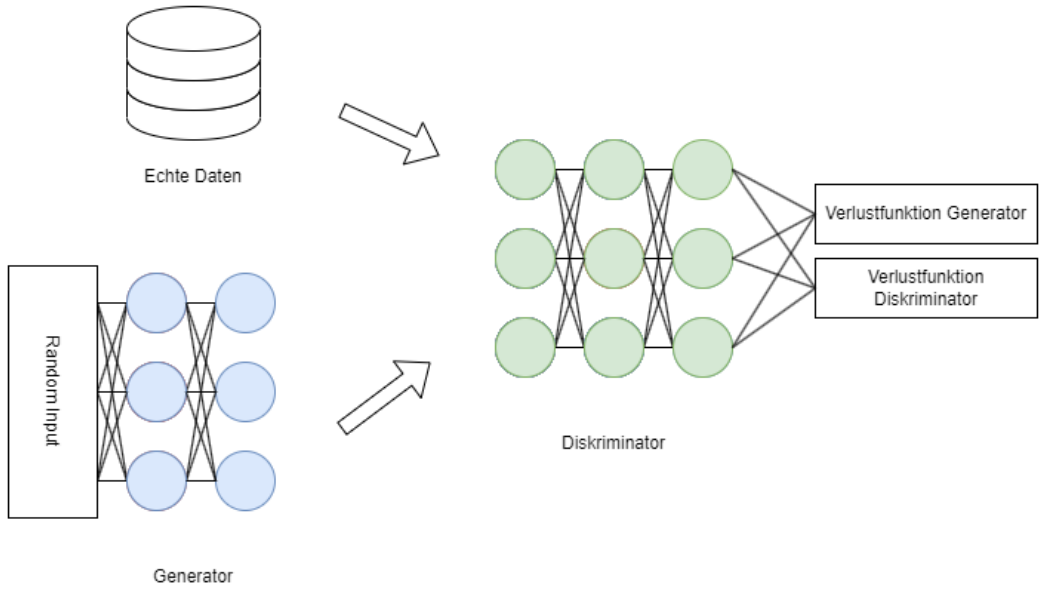


Abbildung 3.4: Generative Adversarial Network nach [17]

Die Erzeugung von Daten mittels GANs kann nicht nur für Angriffe genutzt werden (siehe Kapitel 2), sondern auch für das Training von neuronalen Netzen. Eine Erweiterung der

GANs ist das sogenannte Wasserstein GAN oder auch kurz WGAN [45]. Standardmäßig kann bei GANs der Fall auftreten, dass die erzeugten Daten nicht jeden Teil einer Verteilung abbilden, sondern nur einen Teil, z. B. den am häufigsten vorkommenden Datensatz. Um dieses Problem zu mindern, wird beim WGAN die Verlustfunktion des Diskriminators verändert. Anstatt einer binären Klassifikation (echt oder unecht), wird die Wasserstein-Distanz genutzt, welche angibt, wie viel Arbeit benötigt wird, um eine Verteilung in eine andere Verteilung zu transformieren. Die Wasserstein-Distanz gleicht der Earth-Mover-Distanz aus Kapitel 3.2.1. Diese Metrik als Verlustfunktion ist jedoch nur nützlich, wenn nicht mit einzelnen Datensätzen, sondern jeweils mit Batches trainiert wird. Der Austausch der Verlustfunktion sorgt dafür, dass der Generator Daten aus der ganzen Verteilung der Daten nachstellen muss und nicht nur aus einem Teil dieser.

Xie et. al. [46] stellen eine besondere Form des GANs vor, das sogenannte Differentially Private Generative Adversarial Network oder kurz DPGAN. Dieses DPGAN nutzt als Basis das WGAN, fügt jedoch bei Berechnung der Verlustfunktion (Wasserstein-Distanz) Rauschen mittels des Gauß-Mechanismus aus Kapitel 3.2.2 hinzu. Durch die Eigenschaft, dass Differential Privacy resistent gegenüber Nachbearbeitung ist, kann auch garantiert werden, dass die Gradienten, die im Generator ankommen, mittels Differential Privacy geschützt sind. Die Autoren können zeigen, dass es möglich ist, Bilder der MNIST Datenmenge [20] zu erzeugen. Abbildung 3.5 zeigt diese Erzeugung mit unterschiedlichen ϵ Werten. Es ist zu sehen, dass bei kleiner werdendem ϵ, die Qualität der Bilder schlechter wird und auch mehr Daten der falschen Klasse erzeugt werden. Bei $\epsilon = 9,6$ ist die Anzahl an Daten mit falschem Label größer, als die Anzahl der Daten mit richtigem Label. Folglich ist die Wahl von ϵ entscheidend, wie gut die synthetischen Daten die Originaldaten wiedergeben. Die Wahl muss von ϵ muss dabei für jeden Use Case neu evaluiert werden.

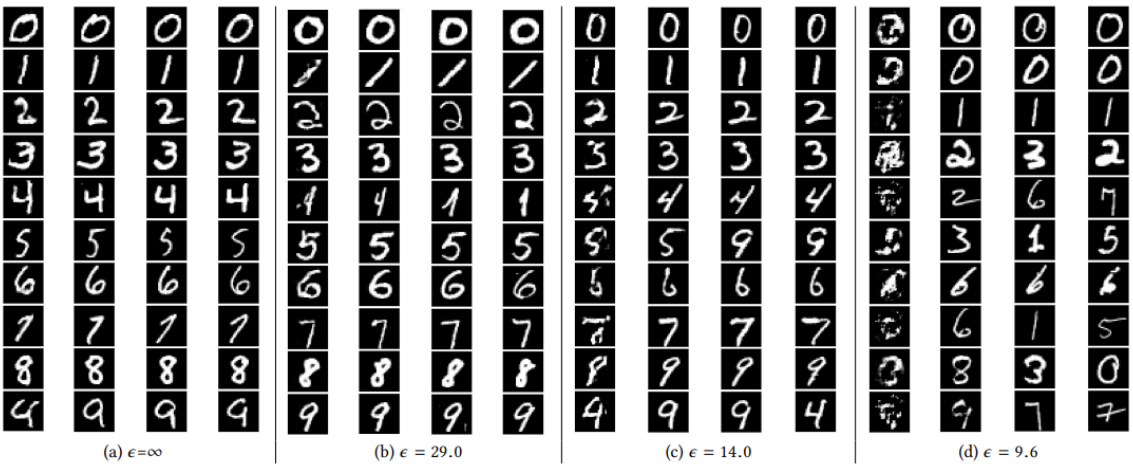


Abbildung 3.5: Synthetische MNIST Datenmenge mittels DPGAN [46]

Jordon et. al. [47] stellten eine alternative Form des GANs vor, welches ebenfalls synthetische Daten erzeugt. Dieses nutzt das Private Aggregation of Teacher Ensembles Framework,

kurz PATE, welches in Kapitel 3.3.2 im Detail beleuchtet wird. Bei der PATE-Architektur wird der Datenbestand in verschiedene Teildatenmengen unterteilt. Verschiedene Modelle, die sogenannten Lehrer oder Teacher Modelle, lernen die Klassifikation jeweils an einer der unterschiedlichen Teildatenmengen. Ein weiteres Modell, welches Schüler oder Student Modell genannt wird, kann nun mittels des Exponential-Mechanismus aus 3.2.2 aus den aggregierten Vorhersagen der Lehrer Modelle, eine Klasse vorhersagen. Das PATE-GAN nutzt die PATE-Architektur für den Diskriminator, welcher ein binärer Klassifikator ist. Wie das DPGAN, nutzt auch das PATE-GAN die Resistenz gegenüber Nachbearbeitungen von Differential Privacy aus, um Differential Privacy auch für die synthetischen Daten zu garantieren.

Ein weiterer Algorithmus, welcher kein GAN zur Erzeugung künstlicher Daten nutzt, ist NIST-MST von McKenna et. al. [48]. Mit NIST-MST gewannen McKenna et. al. die *Differential Privacy Synthetic Data Competition*, welche vom National Institute of Standards and Technology der USA ausgetragen wurde. Neben NIST-MST, welcher auf den obigen Wettbewerb angepasst wurde, gibt es noch MST für generelle Anwendungsfälle. Die MST Methode basiert, wie MWEM [44], auf Marginalverteilungen. MST besteht dabei aus 3 Schritten [48]:

1. **Wahl von Marginalverteilungen:** Aus dem Datenbestand, über den synthetische Daten erzeugt werden, können mehrere Marginalverteilungen gewählt werden. Dabei sollten die wichtigsten Zusammenhänge und Abhängigkeiten des Datenbestands in diesen vorkommen. Es wird deshalb empfohlen, dass ein Fachexperte die Marginalverteilungen aussucht.
2. **Zählen der Marginalverteilungen:** Marginalverteilungen enthalten die Anzahl von Attributwerten eines Attributes in Abhängigkeit von den Attributwerten eines anderen Attributes. Um die Vertraulichkeit der Daten mittels Differential Privacy zu schützen, werden die unterschiedlichen Anzahlen der Variablen mittels Gauß-Mechanismus verrauscht.
3. **Erzeugung der Daten:** MST nutzt ein Tool namens Private-PGM [49], welches von den gleichen Autoren stammt, um einen künstlichen Datenbestand zu erzeugen. Dabei sollen die Marginalverteilungen des künstlichen Datenbestands möglichst nahe an den gemessenen Marginalverteilungen liegen.

Die MST Methode ähnelt der MWEM Methode, jedoch gibt es einige Unterschiede. MWEM ist ein iterativer Algorithmus, welcher in jeder Iteration die Marginalhäufigkeit mit der größten Differenz zwischen originalem Datenbestand D und synthetischem Datenbestand D' wählt und die Attributwerte des synthetischen Datenbestands so anpasst, dass diese sich der Marginalhäufigkeit des originalen Datensatzes angleicht. MST hingegen stellt mit dem Tool Private-PGM [49] ein Optimierungsproblem auf, welches versucht, einen Datenbestand

zu finden, dessen Marginalverteilungen möglichst Nahe an den gemessenen Marginalverteilungen des originalen Datenbestands liegt.

3.3 Training des Modells

Das Training eines neuronalen Netzes bietet eine Vielzahl an Anknüpfungspunkten, um die Vertraulichkeit der Daten zu sichern. Der für das Training leichteste Fall ist es, wenn bereits bei der Vorverarbeitung der Daten ein Mechanismus angewendet wurde, der die Vertraulichkeit schützt. Dies sorgt dafür, dass das Training ohne Anpassung stattfinden kann. Alternative Methoden, welche im Folgenden detailliert erläutert werden, beeinflussen die Algorithmen der Trainingsmethodik, verändern die Architektur des Modells oder verschlüsseln das gesamte Modell.

3.3.1 Training mit Differential Privacy

Nachdem Kapitel 3.2.2 bereits zeigt, wie Differential Privacy definiert wird und bei der Vorverarbeitung von Daten genutzt werden kann, behandelt das folgende Kapitel, wie Differential Privacy während des Trainings genutzt werden kann.

Abadi et. al. [43] zeigen, wie der Trainingsalgorithmus angepasst wird, um Differential Privacy zu unterstützen. Die Methode trägt den Namen Differentially private SGD oder auch DPSGD. Ein Trainingsschritt mittels DPSGD sieht dabei wie folgt aus:

1. Ein Batch von zufälligen Daten wird als Input des Modells für den Forward-Pass genutzt. Dabei kommt jeder Datensatz mit der gleichen Wahrscheinlichkeit in einem Batch vor oder nicht. Dabei werden die Wahrscheinlichkeiten so gewählt, dass im Durchschnitt jeder Batch die gewünschte Größe hat, jedoch kann es einzelne Batches geben, die von dieser Größe abweichen. Zusätzlich könnten einzelne Datensätze auch mehrmals oder gar nicht innerhalb einer Epoche zum Trainieren genutzt werden, wohingegen bei einem normalen Training jeder Datensatz genau einmal pro Epoche vorkommt. Diese Methode zum Zusammenstellen eines Batches wird Poisson-Sampling genannt.
2. Die Gradienten werden für jeden Datensatz aus dem Batch mittels der Verlustfunktion berechnet. Das normale Training ohne Differential Privacy würde aus Gründen der Performance lediglich die Gradienten über den gesamten Batch berechnen, jedoch nicht für jeden Datensatz einzeln.
3. Die maximale Größe der einzelnen Gradienten wird durch das sogenannte Clipping beschränkt. Dabei wird die Länge der Gradienten, welche als Vektoren dargestellt werden, beim Überschreiten eines Schwellwerts C auf den Wert von diesem begrenzt. Der Schwellwert wird auch Clipping-Norm genannt. Dies liegt daran, dass Gradienten

potenziell beliebig groß werden könnten, was dafür sorgen kann, dass einige Datensätze unverhältnismäßig großen Einfluss auf einzelne Gewichte ausüben. Außerdem kann dadurch die Sensitivität Δf bestimmt werden, welche bei der Überwachung des Privacy Budgets während des Trainings relevant ist.

4. Anschließend werden die einzelnen Gradienten durch den Gauß-Mechanismus verrauscht.
5. Die Anpassung der Gewichte erfolgt in die umgekehrte Richtung der Gradienten, skaliert mit einer Lernrate. Hierbei können die Gradienten eines Gewichts über einen Batch wieder aggregiert werden. Dies gleicht dem normalen Trainingsvorgehen.

Ein wichtiger Teil der Methode ist jedoch die Berechnung des ϵ-Werts und des δ-Werts über den Trainingsprozess hinweg. Das Rauschen in Schritt 4 kann dabei so gewählt werden, dass jeder Datensatz innerhalb eines Batches, (ϵ,δ)-Differential Privacy in Bezug auf den Batch erfüllt. Damit dies erfüllt ist, wird der σ-Parameter des Gauß-Mechanismus auf folgenden Wert gesetzt, wobei C der Schwellwert des Clippings ist [43]:

$$C \times \frac{\sqrt{2 \log \frac{1.25}{\delta}}}{\epsilon} \tag{3.8}$$

Anschließend wird für jeden Batch berechnet, welchen Einfluss dieser über die Gradienten auf die Gewichte des Modells hat. Dadurch, dass ein Batch aus zufälligen Datenpunkten des Datensatzes besteht, kann das sogenannte Privacy Amplification Theorem genutzt werden [50]. Dieses besagt, dass jede Anpassung in Bezug auf den ganzen Datenbestand ($\mathcal{O}(q\epsilon)$,$q\delta$)-Differential Privacy erfüllt, wobei q dem Strichprobenverhältnis von Batch-Größe zu der Größe der Datenmenge entspricht und $\mathcal{O}$ dabei der Big-$\mathcal{O}$-Notation. Die Big-$\mathcal{O}$-Notation zeigt hier, dass das Privacy Budget höchstens so schnell wächst, wie das Stichprobenverhältnis q multipliziert mit dem ϵ-Wert eines Batches. Um nun mehrere Trainingsschritte zu bewerten, könnte das Privacy Budgets eines Schritts mit der Anzahl der Schritte T multipliziert werden. Dadurch erfüllt der Trainingsprozess ($\mathcal{O}(q\epsilon T)$,$q\delta T$)-Differential Privacy.

Bei der beschriebene Berechnung des Privacy Budgets, handelt es sich um eine Obergrenze, welche mathematisch bewiesen werden kann. Es ist jedoch vorteilhaft, eine beweisbare Obergrenze zu finden, welche möglichst nahe an dem tatsächlichen Wert liegt. Dies sorgt dafür, dass mehr Rauschen eingefügt werden kann, jedoch die Quantifizierung des Privacy Budgets den gleichen Wert annimmt, was wiederum die tatsächliche Privatsphäre der Daten mehr schützt. Eine Möglichkeit die Distanz der berechneten Obergrenze bei DPSGD zu minimieren, ist das Strong Composition Theorem [42]. Dabei handelt es sich um ein Theorem, welches dafür sorgt, dass das Privacy Budget über mehrere Schritte geringer ansteigt, primär dadurch, dass nicht mehr mit der Anzahl der Schritte T multipliziert werden muss, sondern nur mit der Wurzel davon. Formell erfüllt der Trainingsprozess ($\mathcal{O}(q\epsilon\sqrt{Tlog(1/\delta)})$,$q\delta T$)-Differential Privacy.

Abadi et. al. [43] zeigen jedoch, dass die Obergrenze sogar noch geringer gesetzt werden kann, als mit dem Strong Composition Theorem. Diese Methode wird Moment-Berechnung genannt und sorgt dafür, dass der Trainingsprozess mit DPSGD ($\mathcal{O}(q\epsilon\sqrt{T})$,$\delta$)-Differential Privacy erfüllt. Da δ normalerweise kleiner gesetzt wird, als die Inverse der Anzahl an Datensätzen im Datenbestand, sorgt das Wegfallen des Terms $\sqrt{log(1/\delta)}$ im ϵ-Teil, für eine signifikante Verkleinerung des Privacy Budgets über den Trainingsprozess hinweg. Zusätzlich entfällt im δ-Teil der Faktor qT, wodurch der gesetzte δ-Wert über den Trainingsprozess konstant bleibt.

3.3.2 Private Aggregation of Teacher Ensembles

Die Private Aggregation of Teacher Ensembles Architektur (kurz PATE-Architektur) wurde 2017 von Papernot et. al. [51] erstmals vorgestellt. Dabei handelt es sich um eine sogenannte Wissenstransfer-Architektur (Knowledge Transfer), bei welcher mindestens ein Modell genutzt wird, um ein weiteres Modell zu trainieren.

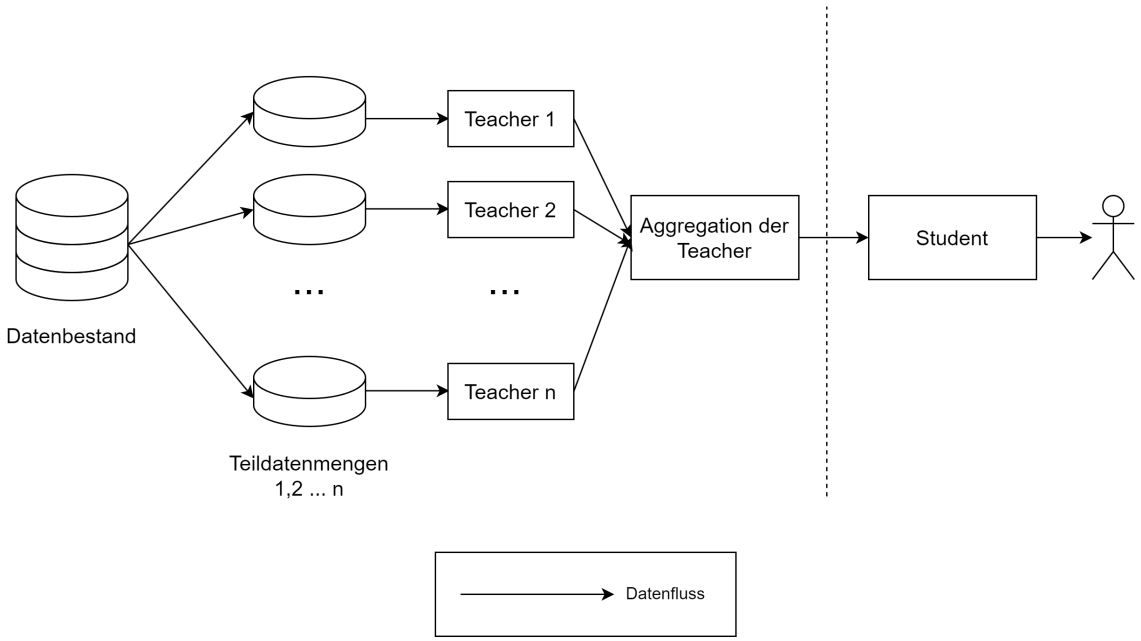


Abbildung 3.6: PATE-Architektur nach [51]

Abbildung 3.6 zeigt eine Übersicht der PATE-Architektur. Der Datenbestand wird dabei zuerst in verschiedene Teildatenmengen aufgeteilt. Für jede dieser Teildatenmengen wird anschließend ein sogenanntes Teacher oder Lehrer-Modell trainiert. Wenn das gesamte Modell eine Klassifikation aus 10 unterschiedlichen Klassen vornimmt, dann muss jedes Teacher-Modell ebenfalls genau diese 10 Klassen abbilden. Die Teacher-Modelle müssen also die gleiche Aufgabe bewältigen, wie das gesamte Modell. Hierbei ist es ebenfalls wichtig, dass

die Teildatenmengen nicht zu klein sind, da ansonsten die Teacher-Modelle nicht gut funktionieren würden.

Die Teacher-Modelle werden anschließend genutzt, um einen Trainingsdatenbestand für das Student-Modell zu labeln. Dieser Trainingsdatenbestand besteht aus originalen Trainingsdaten sowie zusätzliche öffentliche Daten, wobei diese keine Labels benötigen, da die Teacher-Modelle die Labels bestimmen. Um ein Label für einen Trainingsdatensatz zu erhalten, werden die Vorhersagen aller Teacher-Modelle aggregiert, indem die Vorhersagen der einzelnen Teacher-Modelle addiert werden. An dieser Stelle werden die Anzahlen jeder Klasse mittels des Laplace-Mechanismus verrauscht. Dieser Trainingsdatenbestand mit den erzeugten Labels, kann nun an das sogenannte Student oder Schüler-Modell weitergegeben werden, um dieses zu trainieren. Das Student-Modell ist auch das einzige, welches für den Nutzer erreichbar ist. In Abbildung 3.6 ist dies der rechte Teil der gestrichelten Linie. Der linke Teil, die Teacher-Modelle sowie die Aggregation dieser, sind nicht für einen externen Nutzer erreichbar.

Die Moment-Berechnung von DPSGD [43] zur Bestimmung des Privacy Budgets über den Trainingsprozess, kann bei der PATE-Architektur ebenfalls genutzt werden. Das Rauschen wird dabei beim Labeln der Daten mit dem Laplace-Mechanismus hinzugefügt. Dabei kann Rauschen für einen einzigen Datensatz hinzugefügt werden, aber auch für einen ganzen Batch. Wird das Rauschen des Laplace-Mechanismus durch $Lap(1/0.5\gamma)$ definiert, hat ein Batch (γ,0)-Differential-Privacy im Verhältnis zum gesamten Datenbestand. Durch die Moment-Berechnung ergibt sich demnach ein ϵ-Wert von $\gamma\sqrt{T}$ für T Trainingsschritte die jeweils einen gelabelten Batch nutzen.

Die Autoren untersuchen zusätzlich noch mehrere Möglichkeiten, das Student-Modell zu trainieren [51]. Die erfolgreichste Methode war ein teilweise beaufsichtigtes (semi-supervised) Lernen, welches auf GANs basiert. Diese wird folglich PATE-G genannt. Hier werden auch öffentliche, unsensible und ungelabelte Daten genutzt. Teilweise werden die ungelabelten Daten mit der Aggregation der Teacher-Modelle gelabelt, dies ist jedoch nicht für alle Datenpunkte notwendig. PATE-G nutzt den Student als Diskriminator eines GANs, der eine Klasse für "*unechte Daten*" besitzt. Anstelle einer Klasse für "*echte Daten*", gibt es jedoch Neuronen für jede Klasse des Datensatzes. Somit ist das Student-Modell ein Diskrimator, welcher jede echte Klasse klassifizieren kann, sowie zusätzlich eine Klasse für "*unechte Daten*" besitzt. Der Generator ist ein zusätzliches neuronales Netz, welches Datensätze erzeugt, die versuchen, echten Datensätze nachzubilden. Das Student-Modell, der Diskriminator, soll während des Trainings falsche Verteilungen in die Klasse für unechte Daten einordnen. Echte, von den Teacher-Modellen gelabelte Daten, sollen der richtigen Klasse zugeordnet werden, wohingegen echte, ungelabelte Daten irgendeiner beliebigen, echten Klasse zugeordnet werden können. Ein Vorteil von PATE-G ist, dass die ungelabelten Daten nicht verrauscht werden und sich dadurch das Privacy Budget nicht erhöht. Obwohl die Klassen in dem Trai-

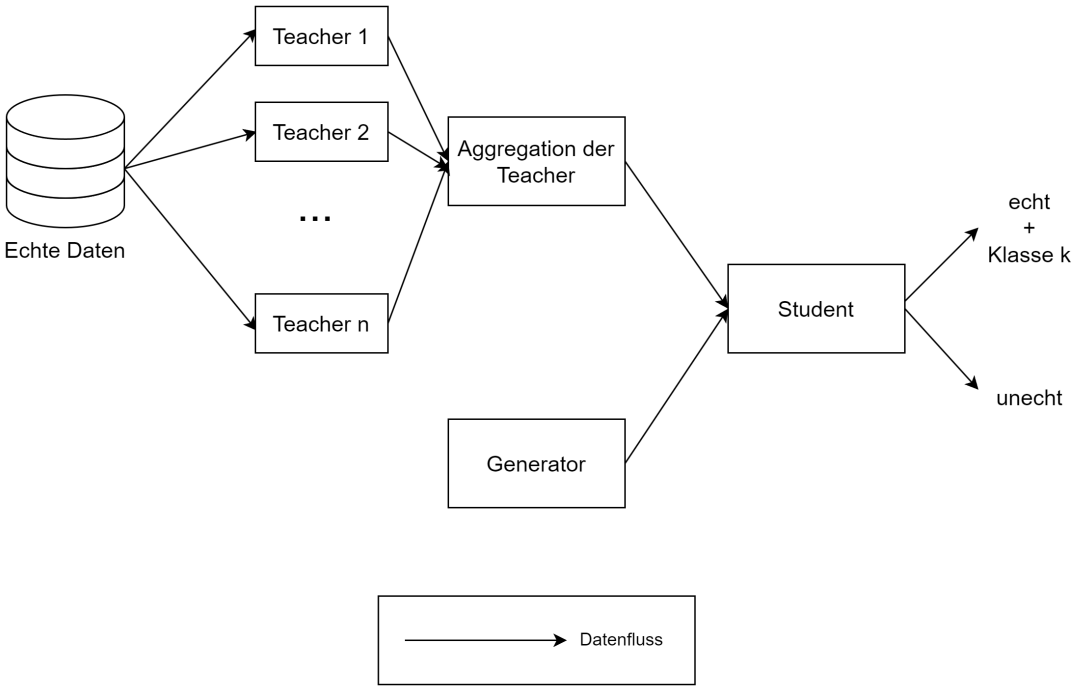


Abbildung 3.7: PATE-G

ning des Student-Modells nicht zwingend richtig klassifiziert werden müssen, steigert dies dennoch die Güte des Modells. Abbildung 3.7 zeigt die bereits beschriebene Trainingsarchitektur von PATE-G.

3.3.3 Homomorphe Verschlüsselung

Homomorphe Verschlüsselung ist eine Methodik, um Berechnung auf verschlüsselten Daten durchführen zu können. Dadurch können Daten beispielsweise in der Cloud verarbeitet werden, ohne dass es dem Anbieter des Services möglich ist, die Vertraulichkeit der Daten zu gefährden.

Ein Homomorphismus in der Algebra bezeichnet eine strukturerhaltende Abbildung einer Mengen G in eine andere Menge H. Dabei hat jedes Element $g \in G$ mindestens ein Bild $h \in H$ und die Relationen der Elemente $g \in G$ zueinander, finden sich auch in H wieder [52].

Ein typisches Beispiel ist der Homomorphismus zwischen zwei Gruppen $(G, \circ)$ und $(H, *)$, wobei $\circ$ und $*$ jeweils eine Verknüpfung innerhalb der Gruppe symbolisieren. Die Beziehung der beiden Gruppen wird Gruppenhomomorphismus genannt, wenn es eine Funktion f :

$G \to H$ gibt, die Elemente der Gruppe G auf die Gruppe H abbildet und dabei für alle Elemente $g_1, g_2 \in G$ gilt [53]:

$$f(g_1 \circ g_2) = f(g_1) * f(g_2)$$

Abbildung 3.8 zeigt eine grafische Darstellung dieses Gruppenhomomorphismus.

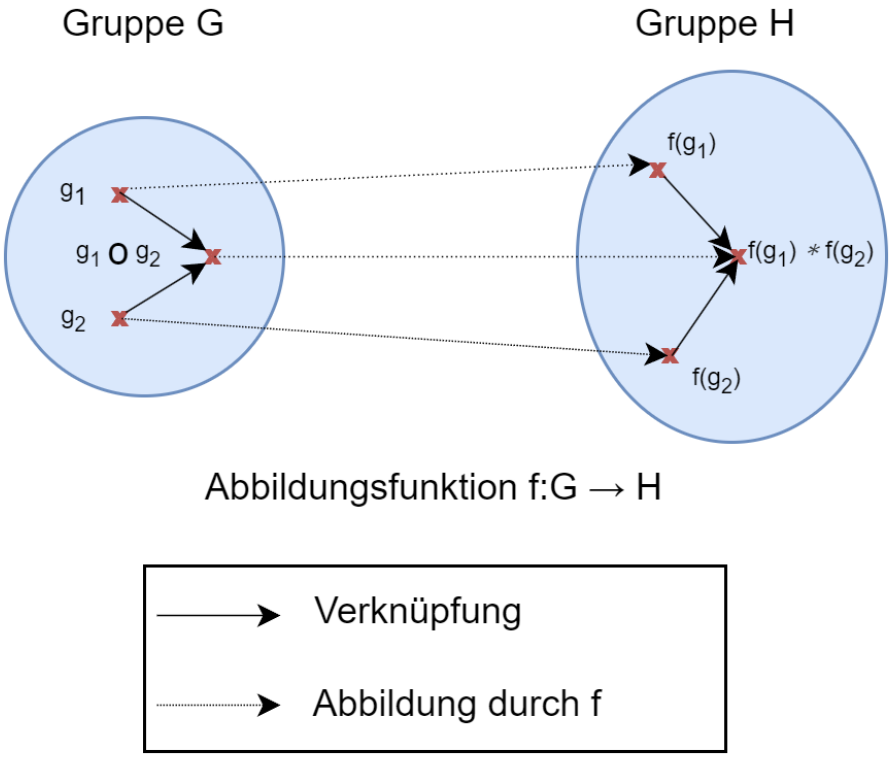


Abbildung 3.8: Gruppenhomomorphismus nach [53]

Bei der homomorphen Verschlüsselung, handelt es sich um einen Gruppenhomomorphismus zwischen der Gruppe der Klartexte $(P, \circ)$ und der Gruppe der Geheimtexte $(C, *)$. Die Abbildfunktionen sind dabei der Verschlüsselungsalgorithmus $Enc_k : P \to C$ und der Entschlüsselungsalgorithmus $Dec_k : C \to P$ mit einem Schlüssel $k \in K$ [53]. Daraus lässt sich ableiten, dass folgende Bedingungen erfüllt sind:

$$Enc_k(p_1 \circ p_2) = Enc_k(p_1) * Enc_k(p_2)$$

$$Dec_k(c_1 * c_2) = Dec_k(c_1) \circ Dec_k(c_2)$$

Abbildung 3.9 zeigt, wie besagter Homomorphismus aussieht.

Konkret bedeutet dies, dass es für eine Verknüpfung innerhalb der Gruppe der Klartexte P, von zwei Elemente $p_1, p_2 \in P$ auf ein drittes Element $p_3 \in P$, eine andere Verknüpfung

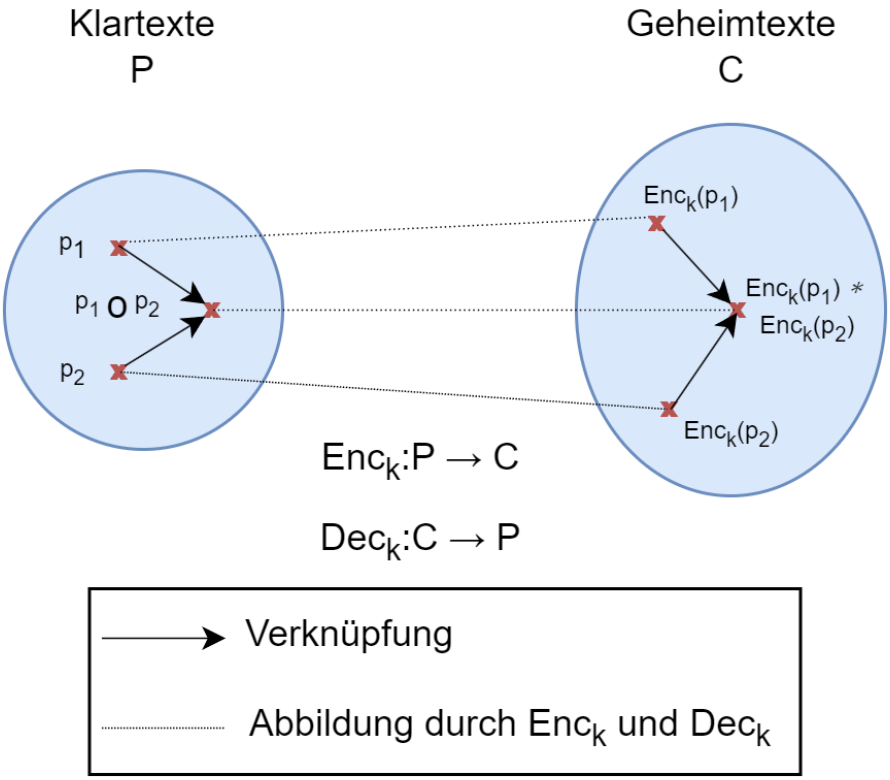


Abbildung 3.9: Homomorphe Verschlüsselung

innerhalb der Gruppe der Geheimtexte C gibt, welche die gleiche Verknüpfung mit den verschlüsselten Daten berechnet. Die Verknüpfung innerhalb der Gruppe der Geheimtexte bildet also zwei Elemente $Enc(p_1), Enc(p_2) \in C$ auf ein drittes Element $Enc(p_3) \in C$ ab, welches dem verschlüsselten Wert von p_3 entspricht. Ist die Verknüpfung eine Addition, so gibt es demnach die Möglichkeit, diese Addition nur mit den verschlüsselten Daten zu berechnen.

Homomorphe Verschlüsselungen lassen sich dabei in 3 Kategorien einteilen, je nachdem welche Verknüpfungen innerhalb der Gruppen möglich sind [54]:

- **Teilweise homomorphe Verschlüsselung (partially):** Entweder Multiplikation oder Addition als Verknüpfung der Klartexte möglich, jedoch nicht beides.
- **Eingeschränkte homomorphe Verschlüsselung (somewhat oder leveled):** Sowohl Multiplikation als auch Addition möglich, jedoch beschränkt durch die Anzahl an durchführbaren Berechnungen.
- **Vollständige homomorphe Verschlüsselung (fully):** Multiplikation und Addition für eine unbegrenzte Anzahl an Berechnungen möglich

Da sich die gängigsten Bestandteile eines neuronalen Netzes über Addition und Multiplikation berechnen lassen, bedeutet dies, dass das Training und auch die Inferenz eines Modells mittels homomorpher Verschlüsselung möglich ist.

Gentry [55] stellte 2009 das erste vollständig homomorphe Verschlüsselungssystem vor. Dabei nutzte er eine eingeschränkt homomorphe Verschlüsselung, welche auf mathematischen Gittern basiert. Das System war eingeschränkt homomorph, da die Verschlüsselung auf einem Rauschen basierte, welches mit jeder Operation größer wurde und letztendlich nicht mehr für eine korrekte Entschlüsselung sorgte. Er erweiterte das System mit einer Technik namens Bootstrapping. Dabei wird der Geheimtext ein zweites Mal verschlüsselt, sodass dieser doppelt verschlüsselt ist. Anschließend kann mittels des verschlüsselten Schlüssels die ursprüngliche Verschlüsselung homomorph herausgerechnet werden. Dadurch wird das Rauschen des Verschlüsselungssystems zurückgesetzt und eine weitere Berechnung ist möglich. Kann das ursprünglich eingeschränkte homomorphe Verschlüsselungssystem die homomorphe Entschlüsselung und eine weitere Operation durchführen, dann kann es mittels Bootstrapping zu einem vollständig homomorphen Verschlüsselungssystem umgewandelt werden. So nutzen beispielsweise Van Dijk et. al. [56] diesen Fakt aus, und ersetzten die auf Gittern basierte Verschlüsselung durch eine auf Ganzzahlen basierte, eingeschränkt homomorphe Verschlüsselung aus.

Brakerski und Vaikuntanathan [57] verbesserten die Effizienz des bereits geschilderten Ansatzes. Sie nutzen eine eingeschränkte homomorphe Verschlüsselung auf Basis des Lernen mit Fehlern Problems (Learning with errors) zusätzlich zu einem Relinearisierungsschritt. Dieser zusätzliche Relinearisierungsschritt reduziert die Größe des Geheimtextes, wodurch die homomorphe Entschlüsselung beim Bootstrapping vereinfacht wird.

Gentry et. al. [58] stellten eine weitere vollständig homomorphe Verschlüsselung auf Basis des Lernen mit Fehlern Problems vor. Die darin eingesetzte, eingeschränkt homomorphe Verschlüsselung stellt den Geheimtext als eine Matrix dar. Die Dimension der Matrix bleibt bei jeder homomorphen Operation gleich und wächst dadurch nicht. Dies ermöglicht, den Relinearisierungsschritt zu entfernen. Bei dem Bootstrapping bisheriger Algorithmen, musste der verschlüsselte Schlüssel oder der Public Key des Nutzers mitgeschickt werden. Bei diesem Ansatz ist es jedoch möglich, alleine mit dem Geheimtext Operationen durchzuführen, die anschließend nur der Nutzer entschlüsseln kann.

Theoretisch wäre das Training eines neuronalen Netzes mittels vollständiger homomorpher Verschlüsselung möglich, jedoch ist es nicht praktikabel. Das Training besteht aus vielen Berechnungsschritten (Inferenz, Berechnen der Verlustfunktion, Gradientenberechnung, Anpassen der Gewichte), welche mit der Größe des neuronalen Netzes ansteigt. So wird bereits bei einem neuronalen Netz mit 7 Schichten, Faltungsschichten und vollständig verbundene Schichten, die Trainingszeit auf einer gewöhnlichen CPU, von ungefähr einer Stunde auf ein

ganzes Jahr erhöht [59]. Es ist möglich, dass weitere, verbesserte Methoden der homomorphen Verschlüsselung entdeckt werden, wodurch sich der Mehraufwand weiter reduzieren lässt und die vollständige homomorphe Verschlüsselung praktikabler wird.

Eine Alternative ist es, keine vollständige, sondern nur eine eingeschränkte homomorphe Verschlüsselung zu nutzen. Takabi et. al. [60] zeigen, wie dies möglich ist. Um das Problem der begrenzten Anzahl an Berechnungen von eingeschränkter homomorphen Verschlüsselung zu umgehen, wird ein zusätzlicher Schritt eingeführt. Wird das Rauschen der Verschlüsselung zu groß und überschreitet einen festgelegten Schwellenwert, muss der aktuelle Zustand entschlüsselt und neu verschlüsselt werden. Hierdurch wird das Rauschen zurückgesetzt, ohne dass Bootstrapping nötig ist. Dadurch, dass kein vollständig homomorphes Verschlüsselungssystem genutzt werden muss, können performantere, teilweise homomorphe Verschlüsselungssysteme genutzt werden.

Neben dem Training eines Modells, gibt es eine Vielzahl an Techniken, welche homomorphe Verschlüsselung nur bei der Inferenz von neuronalen Netzen nutzt. Diese werden in Kapitel 3.4.1 genauer beschrieben. Alternativ kann die homomorphe Verschlüsselung beim verteilten Lernen eingesetzt werden, was in Kapitel 3.3.5 beleuchtet wird.

3.3.4 Funktionale Verschlüsselung

Eine weitere Methodik, Berechnungen auf verschlüsselten Daten durchzuführen, ist die sogenannte funktionale Verschlüsselung. Diese wurde 2011 von Boneh et. al. [61] vorgestellt. Funktionale Verschlüsselung erlaubt es, eine im Voraus definierte, mathematische Funktion f eines Klartextes zu berechnen, wobei nicht der Klartext als Input der Funktion genutzt wird, sondern der Geheimtext. Anders als bei der homomorphen Verschlüsselung, liegt das Ergebnis der Berechnung nicht verschlüsselt vor, sondern als Klartext. Funktionale Verschlüsselung basiert auf folgenden vier Schritten [61]:

1. **Setup:** In einem Vorbereitungsschritt wird ein Public Key pk und ein Master Secret Key msk erzeugt.
$$(pk, msk) \leftarrow Setup$$
2. **Schlüsselgenerierung:** Der Master Secret Key kann nun genutzt werden, um einen spezifischen Secret Key sk für eine definierte Funktion f zu erzeugen.
$$sk \leftarrow Keygen(msk, f)$$
3. **Verschlüsselung:** Der Public Key pk wird genutzt, um den Klartext x, auf welchem die Berechnung durchgeführt werden soll, zu verschlüsseln.
$$c \leftarrow Enc(pk, x)$$

4. **Entschlüsselung:** Die Entschlüsselung des Geheimtextes c mittels des spezifischen Secret Key sk entspricht hierbei der Berechnung der Funktion f mit dem Parameter x.

$$f(x) \leftarrow Dec(sk, c)$$

Die Berechnung der Funktion erfolgt demnach während der Entschlüsselung. Das System, welches die Entschlüsselung vornimmt, sieht dementsprechend nur die verschlüsselten Daten sowie das Ergebnis der Berechnung, nicht jedoch den unverschlüsselten Eingabewert.

Die ersten Ansätze, funktionale Verschlüsselung und neuronalen Netze zu verbinden, fokussierten sich auf die Inferenz der Modelle, welche in Kapitel 3.4.1 genauer betrachtet werden. Ein Framework für das Training des Modells mit dem Namen CryptoNN wurde jedoch von Xu et. al. [62] vorgestellt. Dieses ermöglicht, ein Modell auf einem fremden Server, z. B. in einer Public Cloud, zu trainieren, ohne dass der Provider des Servers Einblick in die Daten erhält. Dabei wird eine funktionale Verschlüsselung genutzt, welche die Berechnung des Skalarprodukts und zusätzlich auch die Grundrechenarten (Addition, Subtraktion, Multiplikation und Division) von zwei Vektoren elementweise ermöglicht. Kombiniert ermöglicht dies, alle notwendigen Berechnungen beim Training des Modells durchzuführen. CryptoNN führt dabei 3 Rollen ein: Autorität, Client und Server. Dabei ist es auch möglich, dass es mehrere Clients gibt, die zusammen ein Modell trainieren. Gibt es jedoch nur einen Client, so übernimmt dieser auch die Rolle der Autorität. Die Autorität ist dafür zuständig, einen Master Secret Key msk und einen Public Key pk zu erzeugen, den Public Key pk an den Client zu verteilen und bei Bedarf spezifische Secret Keys sk_n zu erzeugen und an den Server weiterzugeben. Der Client ist Besitzer der Daten und verschlüsselt diese vor der Übergabe an den Server mit dem Public Key pk. Der Server führt das tatsächliche Training des Modells durch. Beim Forward-Pass wird funktionale Verschlüsselung zur Berechnung der Neuronen der ersten Hidden Layer genutzt. Dafür fordert der Server einen spezifischen Secret Key sk_n an, der die Berechnung der ersten Schicht ermöglicht. Der restliche Forward-Pass erfolgt unverschlüsselt. Wenn der Client auch das Label verschlüsselt, kann die Berechnung der Verlustfunktion für jedes Output Neuron ebenfalls verschlüsselt erfolgen.

Ein Problem des CryptoNN Frameworks ist jedoch, dass davon ausgegangen wird, dass der Server kein Angreifer ist, der effektiv versucht Daten zu extrahieren. Ansonsten könnte dies dazu führen, dass der Server diverse White-Box Angriffe (Kapitel 2) durchführen könnte, da das Modell unverschlüsselt vorliegt. Die Methodik ist demnach darauf ausgelegt, dafür zu sorgen, dass Clients Daten verschlüsselt übertragen können und diese nicht beispielsweise von dem Cloud-Provider mitgelesen werden können.

3.3.5 Verteiltes Lernen

Verteiltes Lernen bietet einige besondere Herausforderungen, die bereits in Kapitel 2.8 betrachtet wurden. Einige bereits beschriebene Methoden lassen sich problemlos auf das verteilte Lernen anwenden. Sollen beispielsweise die Daten der einzelnen Teilnehmer geteilt werden, so ist es möglich, diese mit den Methoden aus Kapitel 3.2 vorzuverarbeiten. So sind beispielsweise die einzelnen Datenbestände jedes Teilnehmers disjunkte Teile des gesamten Datenbestandes aller Teilnehmer, wodurch sich Differential Privacy in der Vorverarbeitung von jedem Teilnehmer anwenden lässt und dennoch eine Quantifizierung der Privatsphäre möglich ist. Jedoch gibt es auch spezielle Methoden, die für das verteilte Lernen ausgerichtet sind. Im Folgenden werden einige davon genauer beschrieben.

Distributed Selective SGD

Shokri und Shmatikov [63] stellen eine Methode vor, bei welcher mehrere Teilnehmer gleichzeitig ein Modell trainieren, ohne dabei die Daten untereinander zu teilen. Diese wird Distributed Selective Stochastic Gradient Descent oder auch Distributed Selective SGD genannt. Das Modell liegt dabei auf einem zentralen Server. Bei der ersten Iteration laden die Teilnehmer das gesamte Modell herunter, bei weiteren Iterationen nur eine festgelegte Anzahl der am meisten geupdateten Parametern (Gewichte). Dadurch soll vermieden werden, dass Overfitting auf den Daten eines einzelnen Teilnehmers auftritt. Die heruntergeladenen Parameter ersetzen die alten Parameter an der entsprechenden Stelle im lokalen Modell. Anschließend wird dieses lokale Modell mit den eigenen Daten trainiert und im Nachhinein eine festgelegte Menge an Gradienten übertragen. Diese können dabei entweder randomisiert ausgewählt werden, wobei eine Sortierung nach Größe empfohlen wird. Alle Teilnehmer wählen die zu teilenden Gradienten ihres lokalen Modells mit der gleichen Strategie aus und behalten diese Strategie über den Trainingsprozess bei. Zusätzlich ist es möglich, die Gradienten vor dem Teilen noch in der Größe zu begrenzen oder Rauschen mittels Differential Privacy hinzuzufügen.

Durch das eingeschränkte Teilen der Gradienten werden so wenig Information wie nötig geteilt, jedoch ist die Güte des Modells kaum schlechter als bei normalem Training. Die Autoren begründen dies damit, dass das lokale Modell lokale Minima durch das Ersetzen von Parametern aus dem geteilten Modell verlassen kann und so weiter in Richtung globales Minimum konvergiert. Zwei Parameter steuern dabei die Performance des Modells beim Training mit Distributed Selective SGD: das Privacy Budget ϵ und die Anzahl der zu teilenden Gradienten. Jeder Teilnehmer, kann mit der Moment-Berechnung von DPSGD [43] die Privatsphäre seines eigenen Trainingsdatensatzes überwachen. Da jedoch nicht alle Gradienten geteilt werden, fällt das tatsächliche Privacy Budget geringer aus, als mit der Moment-Berechnung ermittelt wird, da die Moment-Berechnung sich auf alle Gradienten

bezieht. Werden also weniger Gradienten nach jedem Schritt von jedem Teilnehmer geteilt, so kann das Privacy Budget ϵ größer konfiguriert werden. Werden hingegen viele oder sogar alle Gradienten geteilt, sollte das Privacy Budget kleiner angesetzt werden.

Anspruchsvolle kryptografische Methoden

Takabi et. al. [60] nutzen homomorphe Verschlüsselung, um ein Modell zu trainieren, welches Daten von mehreren Teilnehmern nutzen kann. Die Funktionsweise der Methode mit einem Teilnehmer wurde bereits in Kapitel 3.3.3 beschrieben. Diese lässt sich problemlos auf mehrere Teilnehmer erweitern, indem abwechselnd Daten jedes Teilnehmers verschlüsselt an den Server übertragen wird und diese für das Training des Modells mittels homomorpher Verschlüsselung genutzt wird. Da Daten jeweils verschlüsselt sind, ist es nicht möglich, Daten anderer Teilnehmer zu extrahieren.

Auch das auf funktionaler Verschlüsselung basierende Framework CryptoNN [62], welches in Kapitel 3.3.4 vorgestellt wurde, kann für verteiltes Lernen genutzt werden. Die Rolle des Clients können dabei mehrere Teilnehmer übernehmen, wohingegen die Autorität jedoch von einem separaten System oder Teilnehmer übernommen werden muss. Anschließend können auch bei dieser Methode abwechselnd Daten von verschiedenen Teilnehmern zum Trainieren genutzt werden.

Ein weiterer Ansatz für das verteiltes Lernen, welches auf funktionaler Verschlüsselung basiert, wurde von Xu et. al. [64] mit dem Namen HybridAlpha vorgestellt. Ähnlich zu dem bereits beschriebenen CryptoNN Framework, gibt es auch eine Autorität, welche die benötigten kryptografischen Schlüssel an den Server und die Teilnehmer (Clients) verteilt.

Jedoch übertragen die Teilnehmer keine Daten an den Server. Stattdessen trainiert jeder Teilnehmer eine Kopie des globalen Modells bei sich lokal unverschlüsselt mit dem eigenen Datenbestand. Nach jeder Epoche können die aktualisierten Modellparameter mit dem Laplace-Mechanismus oder Gauß-Mechanismus verrauscht werden. Die Autoren geben jedoch nicht an, wie das Privacy Budget über den Trainingsprozess mehrere Teilnehmer getrackt werden kann. Anschließend verschlüsselt jeder Teilnehmer die Modellparameter mit dem Public Key pk_i, welchen sie von der Autorität bekommen. Diese verschlüsselten Werte, werden an den Server übertragen. Hat der Server alle verschlüsselten Modellparameter jedes Teilnehmers gesammelt, wird mittels funktionaler Verschlüsselung die Summe der Gewichte jedes Neurons gebildet. Daraus kann der Server anhand der Anzahl an Teilnehmern, den Durchschnittswert für jedes Gewicht jedes Neurons bilden und aktualisiert damit das globale Modell. Den dafür benötigten spezifischen Secret Key sk_n, erhält der Server von der Autorität. Die Autoren zeigen anhand der MNIST Datenmenge [20], dass die Güte eines Modells, welches mit HyrbidAlpha ohne Differential Privacy trainiert wurde, sehr nahe der Güte eines

Modells ist, welches in einem verteilten Lernen Szenario ohne HyrbidAlpha gelernt wurde. Wird jedoch zusätzlich Differential Privacy genutzt, sinkt die Güte des Modells.

Secure Multi-Party Computation

Bei der Secure Multi-Party Computation handelt es sich um einen Forschungsbereich mit dem Ziel, dass Teilnehmer gemeinsam eine Funktion berechnen können, ohne dass die einzelnen Eingabewerte aufgedeckt werden. Homomorphe Verschlüsselung und funktionale Verschlüsselung können, je nach Definition, ebenfalls Teil dieses Forschungsgebiets sein. Zusätzlich gibt es weitere Methoden dieses kryptografischen Forschungsgebiets, welche für neuronale Netze genutzt werden können.

Rouhani et. al. [65] stellten ein Framework namens DeepSecure vor, welches Oblivious Transfer, zu Deutsch vergessliche Übertragung, und Garbled Circuits, zu Deutsch verdrehte Schaltkreise, nutzt. Oblivious Transfer ist ein kryptografisches Protokoll zwischen einem Sender und einem Empfänger, bei dem der Empfänger einen Index zwischen 1 und n auswählt und der Sender die Nachricht mit dem entsprechenden Index übermittelt. Der Sender weiß dabei jedoch nicht, welcher Index ausgewählt wurde. Diese Methodik wird auch 1-aus-n Oblivious Transfer genannt. Garbled Circuits, auch Yao's Garbled Circuits genannt, ist ebenfalls ein Protokoll, bei der eine Funktion als Boolescher Schaltkreis mit zwei Eingabegattern dargestellt wird. Dabei erstellt einer der beiden Teilnehmer, hier Alice genannt, Wahrheitstabellen zu jedem Logikgatter des Schaltkreises. Die Inputs sind dabei nicht 0 und 1, sondern jeweils eine Folge von k randomisierten Bits, welche 0 und 1 kodieren. Die Ergebnisspalte dieser Wahrheitstabellen verschlüsselt Alice anschließend mit den beiden Inputs, sodass dies nur mit den beiden Inputs wieder entschlüsselt werden kann. Zusätzlich wird die Reihenfolge der Zeilen randomisiert, damit aufgrund der Reihenfolge keine Rückschlüsse gewonnen werden können. Dieser Schritt wird Garbling genannt und die entstandenen Tabellen sind sogenannte Garbled Tabellen. Anschließend überträgt Alice die Garbled Tabellen an den zweiten Teilnehmer, hier Bob. Mittels 1-aus-2 Oblivious Transfer wählt Bob eine von zwei Nachrichten aus, wobei der Index seinem Input entspricht und die zwei Nachrichten die kodierten Labels von Alice sind. Die erhaltene Nachricht und das eigene Label können nun genutzt werden, um die Ergebnisspalte einer Garbled Tabelle zu entschlüsseln. Bob führt dies für jedes Gatter des Schaltkreises aus. Am Ende erhält Bob den Output des letzten Gatters, welchen jedoch einer der randomisierten Bitfolgen ist. Er übermittelt diesen an Alice und erhält dadurch den entsprechenden 0 oder 1 Wert. DeepSecure wendet Garbled Circuits auf neuronale Netze an. Alice würde in diesem Fall die Daten besitzen und Bob das Modell, welches trainiert wird. Der Forward-Pass würde dabei durch einen Booleschen Schaltkreis aus XOR und XNOR Gattern implementiert werden, wodurch die Berechnung der Vorhersage erfolgt. Dadurch kann Bob den Wert der Verlustfunktion und anschließend die Gradienten der Gewichte bestimmen, ohne die Daten von Alice zu kennen. Alice würde

jedoch auch nicht die genauen Gewichte des Modells kennen. Allerdings ist die Anzahl an benötigten Gattern, um ein neuronales Netz darzustellen, enorm. Einige Operationen, wie die Anwendung einer Aktivierungsfunktion, benötigt mehrere tausende Gatter. Jedes dieser Gatter sorgt ebenfalls dafür, dass eine Menge an Daten übertragen werden muss. Ein neuronales Netz, welches 28×28 Pixel Bilder als Input nimmt, zwei Hidden Layers mit 300 und 100 Knoten (Sigmoid Aktivierungsfunktion) besitzt und eine Softmax Output Layer mit 10 Knoten hat, würde circa 171.300.000 Gatter ausmachen und in einem Forward-Pass ungefähr 2 Gigabyte an Daten übertragen.

Aggregation

Eine alternative Methode wird von Bonawitz et. al. [66] vorgestellt. Diese basiert auf sicherer Aggregation, welche mehrere Daten von unterschiedlichen Teilnehmern verbindet, ohne dass die Daten eines einzelnen Teilnehmers erkenntlich werden. Teilnehmer trainieren ein lokales Modell mit den eigenen privaten Daten. Bevor die angepassten Parameter aber an das globale Modell übertragen werden, werden die Parameter mit den Parametern anderer Teilnehmer kryptografisch aggregiert. Dadurch erhält das globale Modell Gradienten aller Trainingsdaten, ohne die einzelnen Daten zu kennen.

3.4 Anpassung und Betrieb des Modells

Nachdem ein neuronales Netz trainiert wurde, muss dieses für Nutzer zugänglich gemacht werden. Häufig wird das Modell dabei auf einem Server bereitgestellt und über eine API direkt angeboten oder in ein bestehendes Produkt integriert. Dabei ist es möglich, zusätzlich die Vertraulichkeit zu sichern. Grundsätzlich ist ein Modell, welches über eine API bereitgestellt wird, wie ein Stück Software zu behandeln. Best Practices der Softwareentwicklung (wie z. B. Authentifizierung) können und sollten genutzt werden, um grundlegende Sicherheit zu gewährleisten. Dabei können die Sicherheitsmechanismen, abhängig von der Plattform, variieren. Jedoch gibt es einige Methoden, die speziell für Machine Learning Modelle und damit auch neuronale Netze genutzt werden können.

Das Ergebnis eines Modells kann beispielsweise mittels Differential Privacy (Kapitel 3.2.2) verrauscht werden, bevor es über die API zurückgegeben wird. Bei einer Regression, wo die Vorhersage ein Zahlenwert ist, könnte entweder der Gauß-Mechanismus oder der Laplace-Mechanimus genutzt werden. Bei einer Klassifikation kann der Exponential-Mechanismus genutzt werden. Die Wahl des Privacy Budget ϵ ist dabei abhängig von der Sensibilität der Daten.

Im Folgenden werden weitere Methoden betrachtet, welche die Berechnung der Vorhersage verändern. Dies geschieht beispielsweise durch Nutzung von zusätzlichen kryptografischen

Methoden wie homomorpher Verschlüsselung, aber auch durch eine Transformation des Modells.

3.4.1 Kryptografische Inferenz

Wird ein Modell auf einen fremden Server (beispielsweise in einer Cloud) deployt, wäre es möglich, dass der Provider des Servers Informationen über die Daten, die zur Vorhersage genutzt werden, herausfindet. Dies wird sowohl durch homomorphe Verschlüsselung als auch durch funktionale Verschlüsselung verhindert. Beide Methoden sorgen dafür, dass Daten verschlüsselt in das Modell gegeben werden, sodass sich diese Daten zu keinem Zeitpunkt unverschlüsselt auf dem fremden Server befinden. Auch andere Methoden, wie Garbled Circuits, erlauben die Inferenz des Modells, ohne dass die Daten zu einem Zeitpunkt unverschlüsselt veröffentlicht werden.

Inferenz mittels Homomorpher Verschlüsselung

Das bereits in Kapitel 3.3.3 und 3.3.5 beschrieben Framework von Takabi et. al. [60] zeigt, wie homomorphe Verschlüsselung genutzt werden kann, um die Inferenz des Modells durchzuführen. Dabei wird ein eingeschränkt homomorphes Verschlüsselungssystem genutzt, um die Daten eines Nutzers verschlüsselt durch das Modell zu inferieren. Wird das Rauschen des Verschlüsselungssystems zu groß (festgelegter Schwellenwert), erhält der Nutzer den aktuellen Zustand, muss diesen entschlüsseln und neu verschlüsseln, bevor die Berechnung des Modells fortgesetzt werden kann.

Eine weitere Möglichkeit homomorphe Verschlüsselung für die Inferenz eines Modells zu nutzen, wird von Gilad-Bachrach et. al. mit dem Framework CryptoNets [67] vorgestellt. Die Architektur beziehungsweise die Wahl der Schichten des Modells ist jedoch eingeschränkt. Als Aktivierungsfunktion wird eine Quadratfunktion angewendet (anstelle von beispielsweise ReLU) und als Pooling-Operation wird ein Durchschnitts-Pooling genutzt. Um die Performance zu steigern, können Schichten des Netzes miteinander verbunden werden. Dies ist bei benachbarten Schichten aus linearen Funktionen (in der Regel Schichten zwischen Aktivierungsfunktionen) möglich. Zusätzlich kann bei einer Klassifikation auch die Softmax-Funktion weggelassen werden, stattdessen wird einfach die Klasse mit dem höchsten Wert ausgewählt. Dadurch können keine Wahrscheinlichkeiten der Vorhersagen mitgegeben werden, die vorhergesagte Klasse bleibt dennoch gleich. CryptoNets nutzt eine eingeschränkt homomorphe Verschlüsselung, welche auf Gittern und algebraischen Ringen basiert. Die Anzahl an Berechnungen ist dabei an die Struktur und Tiefe des neuronalen Netzes gebunden. Abhängig von der Größe der Sicherheitsparameter werden tiefere neuronale Netze unterstützt. Die Autoren demonstrieren CryptoNets anhand eines neuronalen Netzes, welche eine Klassifikation der MNIST Datenmenge [20] vornimmt. Das neuronale Netz besitzt

dabei zwei Faltungsschichten (Convolutional Layer), die jeweils von einer Aktivierungsfunktion und Pooling Layer erweitert werden. Zusätzlich werden zwei vollständig verbundene Schichten (Fully Connected Layer) hinzugefügt, wobei eine davon als Output Schicht dient. Mit einer Batch Size von 4096 Bildern (28×28 Pixel), benötigt die reine Inferenz des Modells (auf einer Server-CPU aus dem Jahr 2012) 250 Sekunden. Da mehrere Batches parallel durch das Modell inferiert werden können, sind 3600 Batches pro Stunde möglich. Dies würde, bei voller Auslastung, fast 60.000 Vorhersagen pro Stunde entsprechen. Die Ver- und Entschlüsselung benötigen knapp 50 Sekunden zusätzlich, würde aber in einer Client-Server-Architektur vom Client übernommen werden. Zusätzlich müssten in der Client-Server-Architektur pro Batch 370 Megabyte an Daten übertragen werden.

Chabanne et. al. [68] verbesserten den Ansatz von CryptoNets, indem zusätzlich Fokus auf die Güte des Modells gelegt wird. Komplexere Modelle, die bei vielen Aufgaben eine verbesserte Performance aufweisen, sind nicht für CryptoNets geeignet. Dies liegt daran, dass komplexere Modelle mehr Aktivierungsfunktionen enthalten. Werden dafür Quadratfunktionen genutzt, könnte das Training instabil werden, da der Gradient der Quadratfunktion beliebig groß werden könnte. Der Ansatz nutzt anstelle von Quadratfunktionen demnach wieder die ReLU Funktion als Aktivierungsfunktion während des Trainings. Bei der Inferenz des Modells auf verschlüsselten Daten, wird die ReLU Funktion mittels einer Polynomfunktion approximiert. Diese Polynomfunktion wird im Voraus mittels Regression ermittelt. Der Grad der Polynomfunktion kann dabei die Genauigkeit der Approximation und auch die benötigte Rechenleistung beeinflussen. Der Grad 4 der Polynomfunktion zeigte bei der Evaluierung eine ausreichend gute Approximation, sodass Modelle verschlüsselt eine fast genauso gute Güte haben wie unverschlüsselt. Ein neuronales Netz mit 9 Schichten (eine davon ReLU) für die MNIST Datenmenge [20], welches unverschlüsselt eine Genauigkeit von 97,95% hat, würde mit dem Ansatz eine Genauigkeit von 97,84% erreichen. Bei tieferen neuronalen Netzen fällt die Differenz größer aus. Bei einem Netz mit 24 Schichten (sechs davon ReLU), fällt die Genauigkeit von 99.59% auf 97.91%. Der Ansatz von Chabanne et. al. [68] würde demnach das Training im Vergleich zu CryptoNets [67] stabilisieren, jedoch würde durch die Approximation Genauigkeit eingebüßt werden.

Inferenz mittels Funktionaler Verschlüsselung

Das CryptoNN Framework [62], welches in Kapitel 3.3.4 beschrieben wurde, enthält bereits den Forward-Pass eines neuronalen Netzes mit funktionaler Verschlüsselung, welcher der Inferenz des Modells entspricht. Dabei wird nur die erste Schicht des Netzes verschlüsselt berechnet, die restliche Berechnung erfolgt unverschlüsselt.

Dufour-Sans el at. [69] stellen eine alternative Möglichkeit vor, funktionale Verschlüsselung für neuronale Netze zu nutzen. Im Vergleich zu CryptoNN, kann die Inferenz des Modells bei diesem Ansatz ganz verschlüsselt berechnet werden. Dabei wird eine effiziente funktionale

Verschlüsselung genutzt, die jedoch nur Polynome des Grades 2 berechnen kann. Folglich werden nicht alle Operationen unterstützt, sondern nur lineare Schichten, Convolutions mit einem Durchschnitts-Pooling und Aktivierungsfunktionen, die sich mit Polynomen approximieren lassen. Anstelle einer Aktivierungsfunktion an der Output Schicht, wird der größte Wert als Vorhersage genutzt. Mit dieser Methode konnten die Autoren ein Modell trainieren, welches 97,54% Genauigkeit auf der MNIST Datenmenge [20] erreicht und die Inferenz eines Datensatzes auf einer CPU in knapp 13 Sekunden durchführt. Das Paper wurde in einer umfassenderen Version von Ryffel et. al. [70] veröffentlicht. Dieses enthält den Vorschlag, nur die ersten Schichten des Modells mit funktionaler Verschlüsselung zu berechnen. Der Vorteil dieser Variante ist es, dass die unverschlüsselten Schichten mehr Operationen unterstützen und dadurch die Güte des Modells verbessert werden kann.

Inferenz durch Secure Multi-Party Computation

DeepSecure [65], welches in Kapitel 3.3.5 beschrieben wurde, enthält bereits eine Möglichkeit, den Output des Modells mittels Yao's Garbled Circuits zu berechnen. Dabei wird das Modell als ein Boolescher Schaltkreis aus XOR und XNOR Gattern mit zwei Eingabegattern dargestellt, welches anschließend gemeinsam von Nutzer und Server ausgewertet werden.

Ein weiteres Framework zur Evaluation eines neuronalen Netzes mittels Secure Multi-Party Computation wurde von Riazi et. al. [71] mit dem Namen Chameleon vorgestellt. Dieses nutzt neben Yao's Garbled Circuits noch zwei weitere Methoden: das Goldreich-Micali-Wigderson (GMW) Protokoll und Secret Sharing, zu Deutsch Geheimnisteilung. Das GMW Protokoll ist eine Alternative zu Yao's Garbled Circuits, bei dem ebenfalls zwei Parteien gemeinsam, ohne Teilen der privaten Daten, eine Berechnung durchführen können. Ebenfalls wird die Berechnung als Boolescher Schaltkreis dargestellt, welcher aus XOR und AND Gattern besteht. Da die XOR Operation assoziativ ist, können diese Operationen im Voraus in einer sogenannten Offline Phase zusammengefasst werden. Lediglich die AND Gatter benötigen eine Kommunikation der beiden Parteien, was zusätzlich auch rechenintensiver ist. Demnach ist die Anzahl der AND Gatter entscheidend für die Performance. Gibt es wenige, so kann die Performance des GMW Protokolls besser sein, als die von Yao's Garbled Circuits. Bei dem sogenannten Secret Sharing wird ein Geheimnis unter mehreren Parteien aufgeteilt, sodass nur die Kombination aller Teile das korrekte Geheimnis rekonstruierbar macht. Keiner der Parteien kann demnach ohne die anderen Parteien das Geheimnis wiederherstellen. Chameleon nutzt Additives Secret Sharing, was bedeutet, die einzelnen Teile müssen addiert werden, um den ursprünglichen Wert zu erhalten. Zusätzlich findet das Secret Sharing auf einem algebraischen Ring $\mathbb{Z}$ mod 2^k statt, was dazu führt, dass Rechenoperationen mit bekannten Konstanten, ohne Kommunikation, auf einzelnen Teilen des Geheimnisses durchgeführt werden kann.

Das Chameleon Framework kombiniert diese Methoden, um eine effiziente, kryptografisch sichere Inferenz für neuronale Netze zu ermöglichen. Die Autoren demonstrieren das Framework anhand eines neuronalen Netzes für die MNIST Datenmenge. Das genutzte neuronale Netz hat dabei eine Convolutional Layer, gefolgt von einigen Fully Connected Layern. Convolution Layer und Fully Connected Layer werden dabei mittels Additive Secret Sharing und dem GMW Protokoll berechnet. Die Klasse des Datenpunktes wird dabei durch den höchsten Wert der letzten Sicht mittels Yao's Garbled Circuits extrahiert. Das dargestellte Modell benötigt knapp 2,5 Sekunden für die Inferenz eines Datensatzes. Dabei bleibt die Genauigkeit des Modells im Vergleich zur unverschlüsselten Inferenz gleich. Jedoch ist die Performance des Frameworks zusätzlich abhängig von der Netzwerkgeschwindigkeit und Stabilität, da beide Parteien eine Menge an Daten austauschen.

Neben den bereits beschriebenen Lösungen gibt es eine Reihe weiterer Frameworks zur kryptografischen Inferenz eines Modells. Diese setzen sich alle den bereits beschriebenen Methoden zusammen, aber kombinieren diese anders oder optimieren diese an einigen Stellen. So verknüpft das MiniONN Framework [72] die Techniken Oblivious Transfer, Yao's Garbled Circuits und Secret Sharing. Optional können bei MiniONN einige Schritte mit homomorpher Verschlüsselung angepasst werden, was die Last von Kommunikation auf Rechenleistung verlegt. XONN [73] zeigt, wie der Boolesche Schaltkreis eines neuronalen Netzes transformiert werden kann, sodass weniger rechenintensive Gatter vorkommen.

3.4.2 Kompression des Modells

Eigentlich dient die Kompression eines Modells dazu, den Speicherverbrauch zu minimieren und zusätzlich Rechenleistung bei der Vorhersage zu sparen. Jedoch gibt es auch einige Ansätze, wie Modellkompression genutzt werden kann, um die Vertraulichkeit der Daten zu sichern.

Ein Ansatz der Modellkompression ist es, ein Teacher-Modell zu trainieren und dieses dann dazu zu nutzen, ein Student-Modell zu trainieren. Die in Kapitel 3.3.2 beschriebene Methode PATE nutzt ebenfalls eine Teacher-Student-Architektur. Jedoch erfordert PATE eine Anpassung des Trainingsprozesses, indem verschiedene Teacher-Modelle trainiert werden. Andere Techniken können ein bestehendes Modell als Teacher nutzen.

Die Destillation eines Modells wurde erstmals von Hinton et. al. [74] vorgestellt. Dabei handelt es sich auch um eine Teacher-Student-Architektur, bei welcher ein einzelnes Modell, wie auch ein Ensemble an Modellen als Teacher genutzt werden kann. Das Student-Modell, welches eine ähnliche Architektur wie das Teacher-Modell hat, soll dabei lernen, die gleiche Wahrscheinlichkeitsverteilung wie das Teacher-Modell vorherzusagen. Anschließend wird nur

das Student-Modell genutzt, um Vorhersagen zu berechnen. Für das Training des Student-Modells kann der gleiche Trainingsdatenbestand genutzt werden, jedoch ist auch ein alternativer Datenbestand möglich. Als Label werden die Vorhersagen des Teacher-Modells, beziehungsweise die aggregierte Vorhersage des Teacher-Ensembles. Klassifikatoren haben in der Regel eine Softmax-Aktivierungsfunktion in der letzten Schicht, welche die Wahrscheinlichkeiten der einzelnen Klassen ausgibt. Die Softmax-Funktion hat dabei einen Parameter namens Temperatur, welcher die Entropie der Wahrscheinlichkeiten beeinflusst. Normalerweise ist der Wert der Temperatur auf 1 gesetzt, was dafür sorgt, dass die Wahrscheinlichkeit der vorhergesagten Klasse deutlich größer als die anderen Wahrscheinlichkeiten ist. Eine höhere Temperatur hat zur Folge, dass sich die Wahrscheinlichkeiten annähern und die Verteilung dadurch glatter wird. Modell Destillation nutzt eine höhere Temperatur im Teacher-Modell zum Labeln der Datensätze und die gleiche Temperatur während des Trainings des Student-Modells. Dies sorgt dafür, dass das Student-Modell die Verteilungen besser lernen kann, da so auch nicht vorhergesagte Klassen mehr Einfluss auf die Gradienten haben. Nach dem Training nutzt das Student-Modell wieder eine Temperatur von 1. Die Autoren zeigen, dass die Temperatur einen deutlichen Einfluss auf die Güte des Modells haben kann. Der Wert kann dabei zwischen 2,5 und 20 schwanken. Wang et. al. [75] zeigen, dass Modell Destillation in Kombination mit Differential Privacy genutzt werden kann, um ein Student-Modell zu erhalten, welches die Vertraulichkeit der Daten schützt. Dabei werden die Outputs der Softmax-Funktion mit hoher Temperatur des Teacher-Modells mit dem Gauß-Mechanismus verrauscht, bevor diese als Label für das Student-Modell genutzt werden.

Die Methode der Destillation wurde von Polino et. al. [76] durch die sogenannte Quantisierung erweitert. Ziel von Quantisierung ist, die Gewichte des Modells mit in einer festgelegten Bit-Länge anzugeben. Dabei werden die möglichen, kontinuierlichen Werte in den Wertebereich $[0, 1]$ projiziert und können anschließend in einen Zielwertebereich (mit festgelegter Bit-Länge) skaliert werden. Die Skalierung erfolgt dabei anhand einer Gleichverteilung. Dafür werden die kontinuierlichen Werte im Wertebereich $[0, 1]$ in Quantisierungsintervalle eingeteilt, wobei die Anzahl der Intervalle gleich der Anzahl an Bits im Zielbereich ist. Jeder Wert wird dem nächsten dieser Intervalle zugeordnet. Es ist dabei anzumerken, dass durch diese Intervalleinteilung ein Rundungsfehler entsteht. Dieser Fehler entspricht dem Rauschen einer Gauß Verteilung. Dies ähnelt dem Rauschen des Gauß-Mechanismus von Differential Privacy und könnte die Vertraulichkeit schützen. Die Autoren gehen aber nicht weiter auf das Thema ein und es gibt auch keine Berechnung eines Privacy Budgets. Quantisierung kann genutzt werden, um die Größe eines Modells zu reduzieren. Die Autoren zeigen, dass Quantisierung auch in Kombination mit Modell Destillation genutzt werden kann, wodurch das Student-Modell noch kleiner im Vergleich zum ursprünglichen Teacher-Modell wird, obwohl die Genauigkeit nahezu gleich bleibt.

3.5 Zusammenfassung der Methoden

Methoden zur Sicherung der Vertraulichkeit lassen sich in die verschiedenen Phasen des Trainingsprozesses einordnen (Kapitel 3.1):

- Vor dem eigentlichen Training: Aufbereitung des Datensatzes
- Anpassung des eigentlichen Trainings
- Nach dem eigentlichen Training: Anpassung und Betrieb des Modells

Anonymisierte Daten sind eine der bekanntesten Formen zur sicheren Datenaufbewahrung ohne direkte Identifikatoren. Sicher ist dabei unklar definiert, was jedoch durch verschiedene Maße quantifiziert werden soll. So beschreibt k-Anonymität eine Gruppierung anhand von Quasi-Identifikatoren, sodass jede dieser Gruppe mindestens k Elemente enthält. Mit l-Diversität kann zusätzlich noch beurteilt werden, ob sensible Attribut innerhalb eines Datensatzes oder auch eines k-Anonymität Blocks verschieden genug sind. t-Nähe erweitert diesen Ansatz, indem quantifiziert werden soll, wie die Verteilung der sensiblen Attribute in einem k-Anonymität Block im Verhältnis zu der Verteilung innerhalb des ganzen Datensatzes steht (Kapitel 3.2.1).

Differential Privacy ist ein anderes Maß, um zu beurteilen, wie stark eine Abfrage über zwei Datenmengen, die maximal einen unterschiedlichen Datensatz beinhalten, abweichen darf. Zusätzlich gibt es die Möglichkeit, mittels eines Rauschens über eine Abfrage, Differential Privacy mit einem festgelegten Privacy Budget ϵ zu erreichen. Gängig sind dabei der Laplace-Mechanismus und Gauß-Mechanismus als Rauschen über diskrete Werte und der Exponential-Mechanismus als Rauschen bei der Auswahl eines Objekts aus einer Menge. Differential Privacy wird in vielen anderen Methoden genutzt (Kapitel 3.2.2).

Synthetische Daten bieten eine Möglichkeit, Modelle zu trainieren, ohne echte Daten zu verwenden. Generative Adversarial Networks, oder auch GANs, sind optimal dazu geeignet, synthetische Daten aus den originalen Daten zu erzeugen. Dieses bietet unterschiedliche Erweiterungen, um die Verteilungen der beiden Datenmengen, synthetisch und original, miteinander zu vergleichen. Dazu zählt das Wassersetin GAN, welches mittels der gleichnamigen Wasserstein-Distanz dafür sorgen soll, dass nicht nur der häufigste Datensatz imitiert wird, sondern die gesamte Verteilung der Daten vom GAN gelernt wird. Zusätzlich kann ein GAN auch Differential Privacy nutzen, um die originalen Daten zu schützen. Neben GANs gibt es auch statistische Methoden, künstliche Daten zu erzeugen. NIST-MST ist eine dieser Methoden, welche Marginalverteilungen nutzt, um einen synthetischen Datensatz immer mehr dem originalen Datensatz anzugleichen (Kapitel 3.2.3).

Um Differential Privacy im Training zu nutzen, können die Gewichte eines neuronalen Netzes mittels DPSGD angepasst werden. Nach der Berechnung der Gradienten, werden diese mittels des Gauß-Mechanismus verrauscht, bevor die Gewichte angepasst werden. Mittels

der sogenannten Moment Berechnung kann das Privacy Budget nicht nur über einzelne Trainingsschritte, sondern den gesamten Trainingsprozess überwacht werden (Kapitel 3.3.1).

Homomorphe Verschlüsselung ist ein moderner kryptografischer Ansatz, welcher ermöglicht, Berechnungen auf verschlüsselten Daten durchzuführen. Somit kann der Plattformanbieter nicht mitlesen, wenn ein Modell auf seiner Plattform trainiert wird. Da vollständig homomorphe Verschlüsselung sehr rechenintensiv ist, wird oftmals mit teilweise homomorpher Verschlüsselung gearbeitet, die eine begrenzte Anzahl an Operationen zulässt. Mit ein paar Anpassungen genügt dies, um ein Modell zu trainieren (Kapitel 3.3.3).

Ein weiterer moderner Ansatz der Kryptografie ist funktionale Verschlüsselung. Dabei kann eine Funktion berechnet werden, indem nur der Geheimtext des eigentlichen Inputs eingegeben wird. Dies kann ebenfalls genutzt werden, um Modell auf einem Cloud Server zu trainieren, ohne dass der Provider mitlesen kann (Kapitel 3.3.4).

Secure Multi-Party Computation ist ein Gebiet der Kryptografie, welches die Berechnung einer Funktion von mehreren Teilnehmern ermöglicht, ohne die einzelnen Parameter der Teilnehmer zu teilen. Homomorphe Verschlüsselung und funktionale Verschlüsselung gehören ebenfalls zu diesem Gebiet. Ältere Methoden, wie Garbled Circuits, können jedoch auch genutzt werden, um Modelle in einer verteilten Umgebung zu trainieren. Dabei werden Berechnungen, wie der Forward-Pass, als Boolescher Schaltkreis dargestellt, welcher von mehreren Parteien gemeinsam ausgewertet werden kann. All diese Methoden lassen sich nicht nur beim Verteilten Lernen nutzen (Kapitel 3.3.5), sondern auch um ein bereits trainiertes Modell auf einer fremden Umgebung zu Nutzen, ohne die Daten dieser Umgebung preiszugeben. Dieser Schritt wird kryptografische Inferenz genannt (Kapitel 3.4.1).

Eine weitere Möglichkeit sicheres Verteiltes Lernen zu ermöglichen ist das sogenannte Distributed Selective SGD. Dabei laden Teilnehmer ein globales Modell und dessen Updates herunter, trainieren das Modell lokal mit den eigenen Daten und geben eine Auswahl der Updates verrauscht an das globale Modell zurück (Kapitel 3.3.5).

PATE ist eine Technik, bei der ein Teacher Modell genutzt wird, um ein Student Modell zu trainieren. Bei PATE gibt es nicht nur ein Teacher Modell, sondern ein ganzes Ensemble aus Teacher Modellen. Diese können dabei auf sensiblen Daten trainiert worden sein, wohingegen das Student Modell nur auf Vorhersagewahrscheinlichkeiten des Ensembles trainiert wird. Das Student Modell kann anschließend deployt werden und für Nutzer erreichbar sein (Kapitel 3.3.2). Die Modell Destillation beschreibt eine weitere Möglichkeit, um das Wissen eines Teacher Modells auf ein Student Modell zu übertragen. Dabei werden die Vorhersagewahrscheinlichkeiten des Teacher Modells durch Anpassung der Softmax-Funktion verändert, um das Training des Student Modells zu optimieren. Mittels Differential Privacy kann zusätzlich die Vertraulichkeit bei dem Wissenstransfer geschützt werden. Eine weitere Methode ist die Quantisierung des Modells. Dabei werden die Gewichte in eine festgelegte

Bit-Länge, die niedriger als im originalen Modell ist, übertragen. Dies ähnelt dabei dem Hinzufügen von Rauschen mittels des Gauß-Mechanismus von Differential Privacy (Kapitel 3.4.2).

Dieses Kapitel zeigt, dass es eine Reihe an Methoden gibt, die Vertraulichkeit von neuronalen Netzen zu schützen. Das nächste Kapitel bewertet diese Methoden und zeigt, wann es sinnvoll ist, eine Methodik zu nutzen und wie diese optimal eingesetzt wird.

Kapitel 4

Bewertung der Methoden

Nachdem Kapitel 3 einen Überblick über verschiedene Methoden zur Sicherung der Vertraulichkeit gibt, werden diese im folgenden Kapitel bewertet. Kriterien hierbei sind die technische Umsetzung, Schutz vor Angriffen und die generelle Eignung für neuronale Netze zum aktuellen Stand der Forschung.

4.1 Kategorisierung anhand technischer Grundlage

Kapitel 3 gliedert Methoden zur Sicherung der Vertraulichkeit in den Lebenszyklus eines Modells, von der Vorbereitung der Daten bis zum Betrieb des Modells, ein. Eine alternative Kategorisierung erfolgt anhand der technischen und mathematischen Grundlage der Methoden. Hier ergeben sich zwei Hauptkategorien: statistische Methoden und kryptografische Methoden.

Kryptografische Methoden, wie der Name bereits andeutet, nutzen Kryptografie, um Daten oder Berechnungen auf diesen Daten zu verschlüsseln. Die hier genutzten Basistechniken sind homomorphe Verschlüsselung, funktionale Verschlüsselung und andere Secure Multi-Party Computation Methoden wie Garbled Circuits. Diese ermöglichen es, den Trainingsprozess und die Vorhersageberechnung des Modells auf verschlüsselten Daten durchzuführen. Kapitel 4.2 zeigt, ob und wie diese Methoden sinnvoll genutzt werden können.

Statistische Methoden beeinflussen Daten, oder Berechnungen mit diesen, sodass die Vertraulichkeit dieser geschützt werden kann. Anonymisierung ist beispielsweise eine Technik, welche durch Gruppierungen von Merkmalen dafür sorgen kann, dass einzelne Datenpunkte nicht eindeutig zugeordnet werden können. Die Technik, die jedoch aktuell im Kontext Machine Learning am populärsten ist, ist Differential Privacy. Unternehmen wie Apple [77], Google [78], Meta [79] und Snapchat [80] nutzen Differential Privacy bereits in Kombination mit neuronalen Netzen. Aus diesem Grund wird Differential Privacy als Vertreter für statistische Methoden in Kapitel 4.3 bewertet.

4.2 Bewertung kryptografischer Methoden

Moderne kryptografische Techniken ermöglichen es, Berechnungen auf verschlüsselten Daten durchzuführen. Im Folgenden werden diese Methoden anhand von Beispielen, je eine Methode pro kryptografische Technik, analysiert. Dabei wird betrachtet, an welcher Stelle die Methoden angewendet werden können, wie die Rechenleistung und Güte der Modelle beeinflusst wird und ob die Methoden Vertraulichkeit schützen können.

Tabelle 4.1 zeigt die bereits vorgestellten kryptografischen Methoden, welche auf neuronale Netze angewendet werden können. Diese sind nach anhand der zugrundeliegenden Kryptografie gruppiert. Die Phase des Modells beschreibt dabei, ob die Methoden das Training oder nur die Inferenz unterstützen.

Art der Kryptografie	Phase des Modells	Methode
Homomorphe Verschlüsselung	Training + Inferenz	Eingeschränkte homomorphe Verschlüsselung nach Takabi et al. [60]
	Inferenz	CryptoNets [67]
	Inferenz	Eingeschränkt homomorphe Verschlüsselung nach Chabanne et al. [68]
Funktionale Verschlüsselung	Training + Inferenz	CryptoNN [62]
	Inferenz	Funktionale Verschlüsselung für Polynome mit Grad 2 nach Dufour-Sans et al. [69]
	Inferenz	Funktionale Verschlüsselung für Polynome mit Grad 2 nach Ryffel et al. [70]
Yao's Garbled Circuits	Inferenz	DeepSecure [65]
	Inferenz	Chameleon [71]
	Inferenz	MiniONN [72]

Tabelle 4.1: Übersicht kryptografischer Methoden

Phase des Modells

Jede der dargestellten Methoden unterstützt die Inferenz, jedoch nicht zwingend den Trainingsprozess selbst. Dies liegt daran, dass jeder der kryptografischen Methoden dafür sorgt, dass Berechnungen um ein Vielfaches komplexer werden. Ein Schritt des Trainingsprozesses ist wesentlich komplexer als die Vorhersage. Dies liegt unter anderem daran, dass jeder Trainingsschritt bereits eine Inferenz enthält. Zusätzlich muss für jedes Gewicht ein Gradient anhand der Verlustfunktion berechnet werden, anhand dessen jedes Gewicht angepasst

wird. Eine Nutzung von Kryptografie an dieser Stelle würde einen wesentlich höheren Mehraufwand, sowohl von der Komplexität als auch der Performance, bedeuten. Dieser ist sogar so hoch, dass es (mit aktuellen Techniken) nicht sinnvoll wäre, das Training kryptografisch zu sichern. Dies ist der Grund, warum sich kryptografische Methoden auf die Inferenz des Modells fokussieren.

Performance

Da kryptografische Methoden zusätzliche Schritte wie eine Ver- und Entschlüsselung beinhalten, ist es nachvollziehbar, dass Mehraufwand bei Berechnungen entsteht. Jedoch ist dieser verhältnismäßig hoch.

Ein Beispiel für homomorphe Verschlüsselung wäre hier CryptoNets [67]. Zur Evaluation wird ein relativ einfaches neuronales Netz genutzt, welches aus zwei Faltungsschichten, gefolgt von je einer Aktivierungsfunktion und Pooling Schicht, sowie zwei vollständig verbundenen Schichten besteht. Die Autoren zeigen zwar, dass eine Server-CPU in der Lage ist, bei Vollauslastung fast 60.000 Vorhersagen zu treffen, jedoch berücksichtigt dies nicht die Schritte der Ver- und Entschlüsselung. Die gleiche CPU ist in der Lage, 25.000 Datensätze (Bilder) pro Stunde zu verschlüsseln und 275.000 Ergebnisse zu entschlüsseln. Würde man alle Schritte kombinieren, benötigen die 60.000 Vorhersagen ungefähr 2 Stunden und 25 Minuten für die Verschlüsselung und 15 Minuten für die Entschlüsselung zusätzlich. Somit erhöht sich die Gesamtzeit auf 3 Stunden und 40 Minuten. Da ein Modell oftmals nicht unter Vollauslastung läuft, ist auch die Inferenzzeit eines einzelnen Datensatzes relevant. Die Verschlüsselung benötigt 44,5 Sekunde, die Vorhersage des Modells 250 Sekunden und die Entschlüsselung des Ergebnisses 3 Sekunden. Folglich dauert die Inferenz eines Datensatzes insgesamt 297,5 Sekunden.

Da die Autoren nicht spezifizieren, ob CryptoNets durch Grafikkarten (GPUs) beschleunigt werden könnte, wird ebenfalls mit einer CPU verglichen. Dafür wird ein vergleichbares Modell mit dem Framework PyTorch auf einer Desktop-CPU nachgebaut. Es ist anzumerken, dass die genutzte Desktop-CPU im Vergleich zu der von CryptoNets genutzten Server-CPU 6 Jahre neuer ist, 2 Kerne mehr hat und eine etwas höhere Taktrate besitzt. Die Differenz der Ergebnisse wären bei gleicher Hardware geringer, jedoch sind die Ergebnisse dennoch aussagekräftig. Die Klassifikation von 60.000 Datensätzen benötigt weniger als eine Minute und ein einzelnes Bild kann in weniger als einer Sekunden inferiert werden. Dementsprechend benötigt CryptoNets bei größeren Datenmengen unter Vollauslastung 150 Mal länger als ein unverschlüsseltes Modell. Bei einem einzigen Datensatz entspricht der Faktor sogar 300. Bei größeren Modellen würde sich der Faktor sogar erhöhen, was bedeutet, dass die kryptografische Inferenz mit steigender Modellgröße immer unpraktikabler wird.

CryptoNN [62], ein auf funktionaler Verschlüsselung basierendes Framework, bietet eine bessere Performance. Dies liegt daran, dass nicht das ganze Modell kryptografisch berechnet wird, sondern nur die erste Schicht des Modells. Der dadurch reduzierte Schutz des Modells wird im Laufe des Kapitels genauer beleuchtet. Zur Evaluation der Trainingszeit nutzten die Autoren LeNet-5 Modell, ein simples neuronales Netz mit zwei Faltungsschichten, welches mit der MNIST Datenmenge [20] und zwei Epochen trainiert wurde. Unverschlüsselt dauert das Training auf der Hardware der Autoren 4 Stunden und verschlüsselt 57 Stunden, somit über 14 Mal länger. Die Autoren geben keine Zeit für die reine Inferenz an, deshalb wird angenommen, dass der Multiplikationsfaktor der Zeit mindestens gleich groß ist.

Mit dem Chameleon Framework [71] wird das ganze Modell verschlüsselt von zwei Parteien mittels Oblivious Transfer und Garbled Circuits inferiert. Bei dem gleichen Modell, welches auch CrytoNets zur Evaluation nutzt, zeigen die Autoren, dass die Klassifikation eines Datensatzes 3 Sekunden dauert. Jedoch ist der Leistungsvorteil, wenn Daten gebündelt als Batch klassifiziert werden, kleiner als bei CryptoNN. Dies sorgt dafür, dass die Klassifikation von 60.000 Bildern, mit einer Batch Größe von 100, ungefähr 26 Stunden benötigen würde. Im Vergleich zum unverschlüsselten Modell mit PyTorch, benötigt die Klassifikation eines einzelnen Datensatzes 3 Mal mehr, von 60.000 Bilder jedoch 1560 Mal mehr. Zusätzlich ist das Chameleon auf eine schnelle und stabile Netzwerkverbindung angewiesen, was den Leistungsunterschied zu den normalen Modellen noch erhöhen könnte.

Qualität der Modelle

Neben dem Mehraufwand der Berechnungen, gibt es zusätzlich Einschränkungen bei der Erstellung des Modells, was auch die Güte des Modells beeinflussen könnte.

Um Leistung zu sparen, setzt CryptoNets [67] auf eine eingeschränkte homomorphe Verschlüsselung, welche nur eine begrenzte Anzahl an Operationen ermöglicht. CryptoNets ermöglicht nur Polynomberechnungen, was dafür sorgt, dass Funktionen, die sich nicht als Polynom abbilden lassen, approximiert oder ersetzt werden. Dazu zählen Pooling Schichten, aber auch Aktivierungsfunktionen wie ReLU. Bei dem Modell, welches oben beschrieben ist, erreichen die Autoren eine Genauigkeit von knapp 99 %. Somit hat die Approximation, zumindest bei kleinen neuronalen Netzen, kaum Auswirkungen auf die Güte. Da CryptoNN [62] nur die erste Schicht eines Modells verschlüsselt, muss das Framework nicht jede Art von Schicht unterstützen. Faltungsschichten und vollständig verbunden Schichten werden unterstützt, wodurch die Genauigkeit eines Modells nicht negativ beeinflusst wird. Das Framework Chameleon [71] hat ebenfalls keinen negativen Einfluss auf die Genauigkeit von Modellen. Das gleiche Modell wie bei CryptoNets, erreicht mit Chameleon ebenfalls eine Genauigkeit von 99 %.

Schutz der Vertraulichkeit

Public Clouds ermöglichen es, moderne und schnelle Hardware nach Belieben zu buchen und zu nutzen. Dies schließt ebenfalls GPUs ein, welche für das Training und die Vorhersage neuronaler Netze optimiert ist. Ein Problem, welches dabei entsteht, ist, dass Cloud Provider theoretisch in der Lage wären, Daten auf den Servern mitzulesen. Homomorphe Verschlüsselung könnte dieses Problem lösen, da sich weder die Eingabedaten, noch die Vorhersage eines Modells unverschlüsselt auf dem Server befinden. Der Modellbetreiber wäre ebenfalls nicht in der Lage, diese Daten einzusehen.

Da bei der funktionalen Verschlüsselung das Ergebnis jeder Schicht im Klartext vorliegt, bedeutet dies folgend, dass das Label der Vorhersage für den Serverbetreiber und den Modellbetreiber erkennbar sind. Das Modell, beziehungsweise die Anwendung um das Modell herum, kann so implementiert werden, dass der Serverbetreiber beispielsweise die Labels der Vorhersage nicht zuordnen können. Der Modellbetreiber hingegen könnte sogar das Ergebnis der Vorhersage mit Metadaten der Anfrage kombinieren und so dennoch Informationen über den Nutzer des Modells erhalten. Soll ein Modell beispielsweise anhand eines Bildes eine medizinische Diagnose vorhersagen, kann der Modellbetreiber erfahren, welcher Nutzer welche Krankheit hat, ohne jedoch das Bild zu kennen. Bei Garbled Circuits kann der Modellbetreiber ebenfalls das Label auslesen, was zu den gleichen Problemen führt, die auch funktionale Verschlüsselung hat.

Ein Vorteil der kryptografischen Methoden ist die mathematische Beweisbarkeit. Diese ermöglicht es, den Schutz gegen Angriffe zu quantifizieren, indem die Anzahl an Schritten angegeben werden kann, die ein Angreifer ausführen müsste, um die Verschlüsselung zu knacken. Zusätzlich kann die Schlüssellänge erhöht werden, sodass bei Bedarf die Sicherheit auch angepasst ist.

4.3 Bewertung von Differential Privacy

Differential Privacy, kurz DP, ist eine Methodik, welche einschränken kann, wie stark sich ein einzelner Datensatz auf eine Abfrage mit der gesamten Datenmenge auswirkt. Dadurch kann die Privatsphäre einzelner Datensätze geschützt werden und dennoch die Nützlichkeit von Abfragen gewährleistet werden.

Die gängigsten Methoden fügen ein statistisches Rauschen über ein Ergebnis, welches über festgelegte Parameter, das Privacy Budget, angepasst werden kann. Tabelle 4.2 listet Methoden, welche Differential Privacy nutzen, in Abhängigkeit der Modellzyklusphase auf. Im Folgenden werden diese Methoden bewerten, wobei ein besonderer Fokus auf der Wahl des Privacy Budget liegt, da dieser maßgeblich Eigenschaften der Modelle beeinflussen kann. Einige Ergebnisse aus Kapitel 5 fließen hier mit ein.

Phase des Modells	Methode
Vorverarbeitung	Verrauschen der Trainingsdaten
	Synthetischer Datensatz
Training	DPSGD
	PATE Architektur
Inferenz	Verrauschen des Ergebnisses
	Modell Kompression

Tabelle 4.2: Übersicht Differential Privacy Methoden

Phase des Modellzyklus

Differential Privacy lässt sich in jeder Entwicklungsphase eines Modells integrieren. Eine Eigenschaft von Differential Privacy ist die Resistenz gegen Nachbearbeitung. Diese besagt, dass mit Differential Privacy geschützte Datensätze im Nachgang nicht so bearbeitet werden können, dass die Privatsphäre einzelner Datenpunkte weniger geschützt ist. Dies bedeutet, dass bei der Nutzung einer Differential Privacy Methode, nicht nur das Resultat dieser geschützt ist, sondern auch alles was folgt. Konkret bedeutet dies, dass wenn Differential Privacy in der Vorverarbeitung der Daten genutzt wird, die Daten geschützt sind, aber auch das Modell und die Vorhersage mit diesem. Die Nutzung während des Trainingsprozesses sorgt nur für Schutz des Modells und der Vorhersagen. Eine Veröffentlichung des Modells ohne Verletzung der Vertraulichkeit wäre damit also möglich. Wird hingegen nur das Ergebnis verrauscht, ist das Modell weiterhin gegen Attacken, vor allem White-Box Angriffe, anfällig, wohingegen die Vorhersage geschützt ist.

Sollten Daten veröffentlicht werden, empfiehlt es sich, eine synthetische Datenmenge zu erzeugen und nur diese zu veröffentlichen. Bei der Erstellung von dieser, kann nach verschiedenen Metriken optimiert werden, sodass die synthetische Datenmenge gleiche latente Strukturen wie der echte Datenbestand aufweist, ohne jedoch Datensätze von diesem zu offenbaren. Außerdem ist ebenfalls durchaus denkbar, dass ein Modell nicht nur über eine API zur Verfügung gestellt wird, sondern als ganzes Modell. Dafür eignen sich demnach Differential Privacy Methoden, die bei der Vorverarbeitung oder im Training integriert werden.

Komplexität der Methoden

Die Komplexität, um die verschiedenen Methoden zu nutzen, variiert stark. Tabelle 4.3 zeigt eine Übersicht der Komplexität der Methoden. Im Folgenden wird der Wert der Komplexität jeder Methode, in absteigender Reihenfolge, begründen. Jedoch ist anzumerken, dass diese Werte je nach Use Case variieren können.

Methode	Komplexität
Verrauschen der Trainingsdaten	mittel
Synthetischer Datensatz	hoch
DPSGD	niedrig
PATE Architektur	hoch
Verrauschen des Ergebnisses	niedrig
Modell Kompression	niedrig - mittel

Tabelle 4.3: Komplexität Differential Privacy Methoden

Die höchste Komplexität hat dabei die Erzeugung eines synthetischen Datensatzes. Je nach Methode, benötigt dieser einen ganz eigenen Modelltrainingsprozess, welcher andere Differential Privacy Methoden integriert. Zusätzlich wird oftmals die Generative Adversarial Network Architektur benutzt, wie z. B. beim WGAN [45] oder DPGAN [46], welche sogar zwei Modelle trainiert. Die Optimierung dieser GANs kann dabei aufwendiger sein, als der tatsächliche Use Case, welcher umgesetzt werden soll. In Kapitel 5 werden Bilder von menschlichen Gesichtern mit der Auflösung von 224 mal 224 Pixeln genutzt und daraus Eigenschaften der Menschen ermittelt. Das StyleGAN von NVIDIA [81] würde eine passende, synthetische Bilderdatenmenge für den Use Case erzeugen. Jedoch sorgt die spezielle Architektur und die notwendigen Optimierungen dafür, dass die Synthetisierung bei weitem aufwendiger ist, als die Klassifizierung der Merkmale. Eine dieser Optimierungen ist es, dass verschiedene Generatoren trainiert werden, die eine ansteigende Anzahl an Pixeln im Output Bild berechnen. Modell für Modell wird die Anzahl der Pixel erhöht, bis die gewünschte Bildauflösung erreicht ist. Die Erzeugung von Synthetische Datenmengen empfiehlt sich also nur, wenn diese Daten auch veröffentlicht werden sollen. Ansonsten ist es sinnvoller, andere Methoden zu nutzen.

Die PATE-Architektur [51] weist ebenfalls einige Komplikationen auf. Eine Voraussetzung für die Architektur ist es, dass eine relativ große Datenmenge benötigt wird. Dies liegt daran, dass eine Vielzahl an Teacher-Modellen auf Teildatenmengen trainiert werden soll und jedes dieser Modelle eine akzeptable Genauigkeit aufweisen muss, damit das Student-Modell auch eine entsprechende Genauigkeit besitzt. Überwachung und Optimierung mehrere Modelle bring ebenfalls einen erhöhten Aufwand mit sich. Die Übertragung des Wissens der Teacher-Modelle auf das Student-Modell ist ebenso ein Prozess, welcher Komplexität bringt. PATE-G, die effektivste Methode um das Student-Modell zu trainieren, nutzt zusätzlich die GAN Architektur, wobei das Student-Modell der Diskriminator dabei ist. Jedoch erfordert die Methodik das Training eines zusätzlichen Generators. Aus diesen Gründen ist die PATE-Architektur keine empfohlene Methode bei der Integration von Differential Privacy.

Eine Differential Privacy Methode, welche nur mittlere Komplexität aufweist, ist das Verrauschen der Trainingsdaten bevor diese in das Modell gelangen. Technisch ist es relativ leicht die Methode einzusetzen, da moderne Bibliotheken und Frameworks Möglichkeiten

bieten, dies mit ein paar wenigen Zeilen Code zu implementieren. Das Problem entsteht jedoch durch die Fachlichkeit. Jedes Attribut einer Datenmenge muss dabei individuell betrachtet werden, wobei jeweils eine gesonderte Sensitivität und Privacy Budget ermittelt wird. Hier ist Balance ganz entscheidend, denn durch unterschiedlich starke Privacy Budgets können Zusammenhänge verloren gehen, was zu schlechterer Genauigkeit der Modelle führt. Außerdem addiert sich das Privacy Budget jeder Spalte, was dafür sorgt, dass die Privacy Budget Werte ϵ und δ höher sein können als bei anderen Methoden. Dies wiederum sorgt für einen schlechteren Schutz durch diese Methode. Dennoch kann die Methode, bei fachlicher Kompetenz, genutzt werden.

Die Kompression des Modells ist eine Möglichkeit, ein bereits trainiertes neuronales Netz anzupassen und so die Vertraulichkeit zu schützen. Die Modell Quantisierung sorgt beispielsweise dafür, dass die Dezimalzahlen der Gewichte, zu Ganzzahlen umgewandelt werden, was die Berechnung der Vorhersage erleichtert. Diese Umwandlung kann demnach sogar angewendet werden, bei Modellen, welche keine vertraulichen Daten nutzen, alleine aufgrund der verbesserten Leistung. Andere Methoden, wie die Modell Destillation, kann ebenfalls dafür sorgen, dass Modell zu vereinfachen, indem das Wissen eines Teacher-Modells (es sind auch mehrere Teacher-Modelle möglich) auf ein Student-Modell übertragen wird. Wenn das Student-Modell weniger Parameter als das Teacher Modell hat, benötigt die Vorhersage ebenfalls weniger Ressourcen. Zusätzlich kann bei der Destillation explizit Rauschen hinzugefügt werden [75]. Bedarf es einer Simplifizierung eines neuronalen Netzes, empfiehlt es sich eine Methode der Kompression zur Sicherung der Vertraulichkeit zu nutzen.

Eine einfache Methode ist das Verrauschen der Vorhersage. Anders als bei dem Verrauschen der Trainingsdaten muss hier oftmals nur ein einzelner Wert betrachtet werden. Bei der Vorhersage eines kontinuierlichen Werts kann der Laplace oder Gauß-Mechanismus genutzt werden, bei einer Klassifikation der Exponential-Mechanismus oder die Report Noisy Max Methode. Da ein Modell oftmals in zusätzlichen Code eingebunden wird, welcher auch die API bereitstellt, ist die Integration des Rauschens ohne großen Mehraufwand realisierbar. Ein weiterer Vorteil, den diese Methode bietet, ist die leichte Anpassung des Privacy Budgets. Da das Rauschen erst nach der Vorhersage des Modells hinzugefügt wird, muss ein Modell nicht neu trainiert werden, wenn sich das Rauschen verändert. Die Einfachheit dieser Methode ist einer der Gründe, Differential Privacy auf diese Weise zu integrieren.

Ebenfalls handelt es sich bei DPSGD um eine Methode mit niedriger Komplexität. Frameworks wie PyTorch bieten Bibliotheken an, welche nur wenige Zeilen Code erfordern, um DPSGD nutzen zu können. Der Trainingsprozess erfolgt ansonsten weitestgehend normal. Außerdem kann diese Methode bei jedem Use Case, egal ob tabellarische Daten oder Bilddaten, ohne spezifische Anpassungen angewendet werden. Kapitel 5 zeigt, wie eine moderne Implementierung dieser Methode aussieht. Die Flexibilität und Simplizität der Methode sorgen dafür, dass diese die bevorzugte Methode für Differential Privacy ist.

Qualität der Modelle

Die Nutzung von DPSGD verschlechtert die Güte eines neuronalen Netzes. Die Konfiguration der Trainingsvariablen, auch Hyperparameter genannt, haben dabei einen signifikanten Einfluss auf die Güte eines Modells. In Kapitel 5.4 wird experimentell gezeigt, wie diese Parameter optimal gewählt werden. Die exemplarischen Modelle erreichen bei einem ϵ-Wert von 1, zwischen 55 % und 96 % der Genauigkeit der Modelle ohne DPSGD. Bei einem ϵ-Wert von 10 erhöht sich der Bereich. Die Modelle erreichen zwischen 70 % und 97 % der Genauigkeit der Modelle ohne DPSGD.

Wenn ein Modell mit DPSGD nur 55 % der Genauigkeit des Modells ohne DPSGD erreicht, kann jedoch jeder Nutzen verloren gehen. Die Wahl des ϵ-Wertes sollte deshalb experimentell erfolgen. Eine Möglichkeit dazu ist es, ein Modell ohne DPSGD gegen Angriffe zu testen. Wenn die Effektivität eines Angriffs einen festgelegten Schwellwert überschreitet, kann DPSGD mit einem hohen ϵ-Wert genutzt werden. Sinkt die Effektivität des Angriffs nicht unter den festgelegten Schwellwert, kann der ϵ-Wert reduziert werden. Dies wird so lange wiederholt, bis die Effektivität des Angriffs unter dem festgelegten Schwellwert liegt.

Schutz vor Angriffen

Differential Privacy limitiert den Einfluss, welchen ein einzelner Datensatz auf einen Mechanismus hat. Dabei erhält der Mechanismus den ganzen Datenbestand als Eingabe.

Wird also DPSGD genutzt, wird der Einfluss eines einzelnen Datensatzes auf das Modell beschränkt. Einzelne Datensätze haben demnach keinen signifikanten Einfluss auf ein Modell, wodurch Angriffe wie die Membership Inference Attacke oder die Model Inversion Attacke entkräftet werden. Kapitel 5.5 zeigt jedoch, dass diese beiden Angriffe bereits gegen Modelle ohne DPSGD unwirksam sind. Demnach ist die Nutzung von DPSGD nur notwendig, wenn Modelle besonders gefährdet sind durch diese Angriffe. Dies muss experimentell evaluiert werden.

4.4 Zusammenfassung der Bewertungen

Methoden zur Sicherung der Vertraulichkeit können in kryptografische und statistische Methoden eingeteilt werden. Kryptografische Methoden lassen sich in drei Arten einteilen: homomorphe Verschlüsselung, funktionale Verschlüsselung und andere Secure Multi-Party Computation Protokolle. Die Methoden sind primär für die Inferenz von Modellen ausgelegt, da die kryptografischen Techniken für einen hohen Mehraufwand sorgen. Obwohl nur die Inferenz eines Modells verschlüsselt ausgeführt wird, ist der Mehraufwand erheblich. Je nach Wahl der Methode und Anpassung an das Modell dauert eine Inferenz hunderte bis

tausende Male so lang wie ohne Kryptografie. Außerdem steigt dieser Faktor mit Komplexität der Modelle. Ein Vorteil der kryptografischen Techniken ist jedoch der Schutz, den diese bringen können. Homomorphe Verschlüsselung ermöglicht es beispielsweise, ein neuronales Netz auf einem Cloud-Server zu nutzen, sodass weder der Anbieter der Anwendung, noch der Server-Provider die Möglichkeit haben, die persönlichen Daten unverschlüsselt zu lesen. Dadurch könnten also Modelle genutzt werden, ohne die eigenen Daten zu offenbaren, was sogar mathematisch beweisbar ist (Kapitel 4.2).

Als wichtigste statistische Methode zum Schutz der Vertraulichkeit in neuronalen Netzen wird Differential Privacy genauer bewertet. Dabei wird Differential Privacy in einer Vielzahl von Methoden genutzt, welche in unterschiedlichen Phasen des Modellzyklus genutzt werden können. DPSGD ist dabei eine der gängigsten Methoden, um Differential Privacy in Kombination mit neuronalen Netzen zu nutzen. Bei dieser werden die Gradienten der Modellparameter während des Trainingsprozesses verrauscht. Die Qualität der Modelle hängt dabei von der Wahl des ϵ-Werts und der Hyperparameter ab. Die Wahl von des ϵ-Werts beeinflusst ebenfalls die Sicherheit, die mit dieser Methode erzielt wird (Kapitel 4.2).

Die technische Nutzung von DPSGD, sowie die Wahl der Hyperparameter wird folgend in Kapitel 5 evaluiert. Zusätzlich werden ebenfalls zwei Angriffe, die Membership Inference Attacke sowie die Model Inversion Attacke, beleuchtet und deren Effektivität gegen exemplarische Use Cases bewertet.

Kapitel 5

Experimente

Im folgenden Kapitel wird detailliert auf die Nutzung von DPSGD für das Training von neuronalen Netzen eingegangen. Dafür werden zuerst die genutzten Use Cases beschrieben. Nachdem verglichen wird, welche technischen Lösungen, wie Bibliotheken und Frameworks, existieren, wird im Detail beschrieben, wie DPSGD mit Opacus und PyTorch genutzt wird.

Die tatsächliche Evaluierung besteht dabei aus zwei Teilen: der Nutzung von DPSGD und der Effektivität von Angriffen. Bei der Nutzung von DPSGD geht es darum, wie Hyperparameter angepasst werden können, um die bestmöglichen Ergebnisse zu erhalten. Die jeweiligen Ergebnisse werden, abhängig des ϵ-Werts, in Relation zueinander gesetzt. Gegen die resultierenden Modelle, mit und ohne DPSGD, werden anschließend Angriffe ausgeführt und die Effektivität dieser bewertet. Ziel hierbei ist es, herauszufinden, ob die Nutzung von DPSGD einen Einfluss auf die Effektivität der Angriffe hat. Der Code der Experimente befindet sich in einem öffentlichen GitHub-Repository[1].

5.1 Datensatz und Use Cases

Für die nachfolgenden Experimente werden zwei unterschiedliche Datenbestände genutzt: der CIFAR-10 Datenbestand [82] und der CelebA Datenbestand [83]. Bei dem CIFAR-10 Datenbestand handelt es sich um 60.000 Bilder, welche jeweils eine Auflösung von 32×32 Pixeln, sowie je 3 Farbkanäle, haben. Jedes Bild zeigt ein Objekt, welches genau einer von 10 Klassen zugeordnet werden kann. Bei diesen Klassen handelt es sich um Tierarten (beispielsweise "*Hund*"oder "*Katze*") oder um Fahrzeugtypen (beispielsweise "*Auto*" oder "*Flugzeug*"). Der Datenbestand ist bereits in "*Training*" und "*Test*" eingeteilt, wobei jede der 10 Klassen in beiden Datenmengen jeweils genau 10 % einnimmt. Der Use Case ist es nun, die Bilder der richtigen Klasse zuzuordnen. Dafür wird ein Modell genutzt, welches eine ResNet-Architektur mit 10 Schichten nutzt [84]. Dabei handelt es sich um ein neuronales Netz, welches großteils aus Faltungsschichten, sowie einigen Pooling-Schichten

[1] https://github.com/cyberlytics/PrivacyFlow

besteht. Eine Besonderheit ist jedoch, dass die Aktivierungen einiger Schichten nicht nur in der Folgeschicht genutzt werden, sondern auch in einer späteren Schicht. Dies wird als Skip Connections bezeichnet. Bei dem genutzten Modell, welches in Abbildung 5.1 zu sehen ist, gibt es zwei unterschiedliche Blöcke, welche abwechselnd genutzt werden. Die erste Art von Block besteht aus einer Faltungsschicht in Kombination mit einer Pooling-Schicht. Bei jedem dieser Blöcke wird die Anzahl an Kanälen erhöht (von 3 auf 64, anschließend jeweils verdoppelt), jedoch die Pixeldimensionen jeweils halbiert. Die zweite Blockart besteht jeweils aus zwei Faltungsschichten, wobei die Dimensionen der Eingaben unverändert bleiben. Die Aktivierungen der ersten Art von Block, wird jeweils im folgenden Block genutzt, sowie durch die Skip Connection im übernächsten Block. Dabei werden die Aktivierungen addiert, bevor diese in der nächsten Schicht ankommen. In Abbildung 5.1 ist dies durch die Addition zweier Pfeile dargestellt. Die letzte Schicht besteht aus 10 Neuronen, wobei jedes Neuron einer Klasse entspricht. Hier folgt eine Softmax-Aktivierungsfunktion, welche die Werte der Neuronen in die Wahrscheinlichkeiten der Klassenzugehörigkeit umwandelt. Der höchste Wert entspricht anschließend der vorhergesagten Klasse. Als Metrik zur Evaluierung eines Modells wird die Genauigkeit berechnet. Diese gibt das Verhältnis von richtigen Klassifikationen zu der Gesamtzahl an Klassifikationen an.

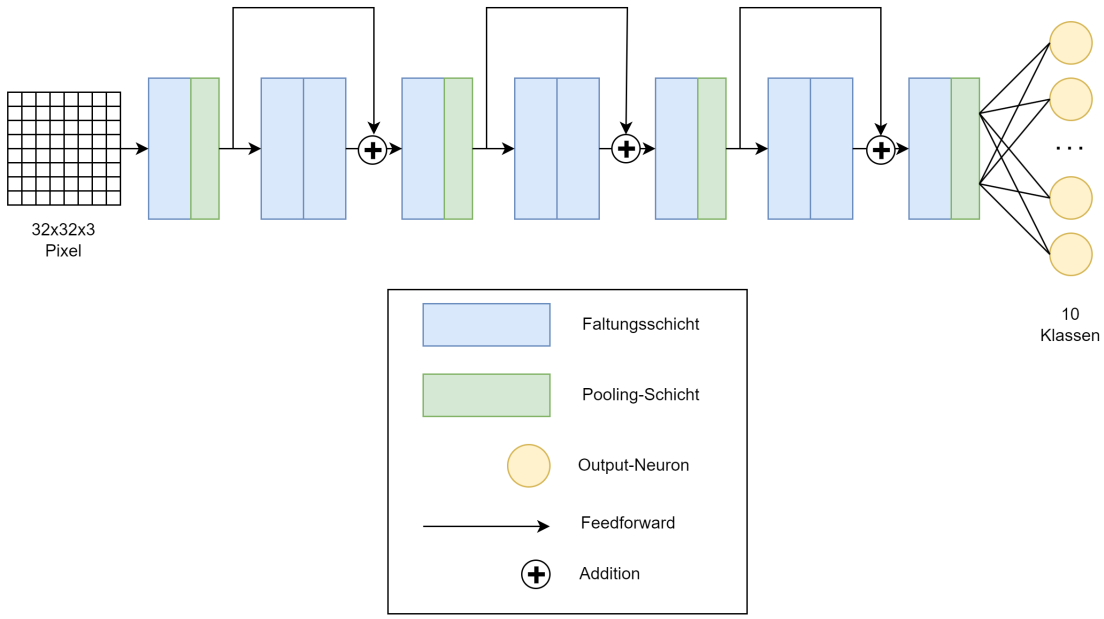


Abbildung 5.1: Neuronales Netz zur Klassifikation von CIFAR-10

Neben CIFAR-10, wird der CelebA Datenbestand [83] genutzt, welcher unter anderem über die Plattform Kaggle zur Verfügung steht. Dieser enthält 202.599 Bilder, auf welchen jeweils das Gesicht einer Person des öffentlichen Lebens zu sehen ist. Die Bilder haben jeweils eine Auflösung von 178×218 Pixeln mit je 3 Farbkanälen. Zu jedem Bild gibt es 40 Labels, wobei jedes davon eine Eigenschaft beschreibt. Dabei können jeweils die Werte -1 (Eigen-

schaft nicht vorhanden) und 1 (Eigenschaft vorhanden) angenommen werden. Ein Beispiel für eine Eigenschaft ist die Größe eines Organs sein (Label "*Big_Nose*"). Manche Eigenschaften können sich außerdem gegenseitig ausschließen. So gibt es beispielsweise mehrere Labels für Haarfarben (Label "*Black_Hair*" und Label "*Blond_Hair*"). Der Datenbestand ist bereits unterteilt in "*Training*", "*Validierung*" und "*Test*". Die vorgegebene Einteilung wird beibehalten, sodass die Güte des Modells immer auf der Testdatenmenge, welche nicht im Training genutzt werden, gemessen wird. Tabelle 5.1 zeigt die Anzahl der Datensätze in den jeweiligen Teildatenbeständen.

Datenbestand	Anzahl Datensätze	prozentualer Anteil
Training	162.770	$\approx 80\%$
Validierung	19.867	$\approx 10\%$
Test	19.962	$\approx 10\%$

Tabelle 5.1: Anzahl Datensätze

Der Use Case ist es dabei, alle 40 Label eines Bildes vorherzusagen. Dabei handelt es sich um ein sogenanntes Multi-Label Klassifizierung, also eine Klassifikation bei der mehrere Klassen (hier Eigenschaften) bestimmt werden können. Als Metrik zur Messung der Güte eines Modells wird dabei die Genauigkeit genutzt, also das Verhältnis von richtig vorhergesagten Labels zu der gesamten Anzahl an Labels.

Für diesen Use Case werden zwei unterschiedliche Modelle genutzt. Bei dem ersten Modell handelt es sich um ein ResNet-18 Modell [84]. Dieses hat eine ähnliche Architektur wie das bereits beschriebene Modell für die Klassifizierung der CIFAR-10 Daten. Jedoch besitzt dieses Modell 18 Faltungsschichten, wodurch die Anzahl der Parameter größer ist. Das zweite Modell ist das sogenannte Vision Transformer Modell [85], kurz ViT. Dieses Modell nutzt einige Elemente der Transformer-Architektur, welche ursprünglich aus dem Bereich der Computerlinguistik stammt [86]. Das Vision Transformer Modell unterteilt das Bild in mehrere, gleich große Teilbilder, welche Patches genannt werden. Dabei erhalten die Patches eine Position, welche von oben links nach unten rechts inkrementiert wird. Die Patches werden über sogenannte Embedding-Schichten in einen höherdimensionalen Vektorraum übertragen. Die entstandenen Vektoren werden Embeddings genannt. Im Laufe des Trainingsprozesses, werden die Parameter der Embedding-Schichten so angepasst, dass die Ähnlichkeit zweier Embeddings mit der Ähnlichkeit der Eingabebilder korreliert. Die Embeddings werden anschließend mit der Position der jeweiligen Patches als Eingabe für einen oder mehrere sequentiell verbundene Kodierer, auch Encoder genannt, genutzt. Diese bestehen aus jeweils mindestens einem Aufmerksamkeitsmechanismus, auch Attention genannt, und einigen vollständig verbundenen Schichten eines neuronalen Netzes. Die Aufmerksamkeitsmechanismen erlernen verschiedene Beziehungen der Patches zueinander, welche durch die jeweiligen Positionen in einer räumlichen Verbindung stehen. Die Ausgabe des letzten Kodierers wird anschließend als Eingabe für vollständig verbundene Schichten genutzt, wel-

che die Klassifikation durchführen. Neuronen der letzten Schicht repräsentieren jeweils eine Klasse.

5.2 Wahl des Frameworks

Viele der Methoden aus Kapitel 3 besitzen ein öffentliches Repository, in denen exemplarisch ein Beispiel umgesetzt und evaluiert wird. Einige dieser Repositories sind seit Jahren nicht mehr gepflegt, weshalb die Wahl der entsprechenden Bibliotheken entscheidend sein kann. Bei kryptografischen Methoden empfiehlt sich der Einsatz von bewährten, erforschten und gepflegten Bibliotheken. Für Homomorphe Verschlüsselung gibt es beispielsweise die SEAL Bibliothek [87], welche von Microsoft entwickelt wird, oder auch die HElib [88]. Diverse Secure Multi-Party Computation Methoden, wie Yao's Garbled Circuits, werden in einer Bibliothek namens MP-SPDZ [89] implementiert. Allerdings sind Bibliotheken für funktionale Verschlüsselung weniger gepflegt. So hat die Bibliothek CiFEr [90] seit mehr als zwei Jahren keine Aktualisierung erhalten.

Um Differential Privacy zu nutzen, ist nicht zwingend eine spezialisierte Bibliothek notwendig. Die populäre Python Bibliothek NumPy [91], die in vielen Machine Learning Projekten eingesetzt wird, bietet bereits Funktionalität, um beispielsweise Werte mittels einer Laplace-Verteilung oder Gauß-Verteilung zu verrauschen. Lediglich die Berechnung des Privacy Budgets fehlt in NumPy.

Die größten Frameworks für die Entwicklung von neuronalen Netzen sind PyTorch [92], TensorFlow [93] und JAX [94]. Tabelle 5.2 zeigt die Anzahl der Modelle, die jeweils mit dem entsprechenden Framework trainiert wurden und auf Hugging Face, einer Plattform zum Teilen von neuronalen Netzen, hochgeladen wurden [95]. Das Framework PyTorch wurde dabei für fast 7 Mal so viele Modelle genutzt, wie die anderen beiden Frameworks zusammen.

Framework	Anzahl Modelle auf Hugging Face
PyTorch	122.061
TensorFlow	9.326
JAX	8.560

Tabelle 5.2: Anzahl Modelle je Framework auf Hugging Face [95]

PyTorch und TensorFlow besitzen beide ein offizielles Modul, welches den Einsatz von Differential Privacy während des Trainings eines neuronalen Netzes ermöglicht. Diese heißen Opacus [96] und TensorFlow Privacy [97]. Für das Framework JAX gibt es keine offizielle Erweiterung für Differential Privacy, lediglich ein paar Open-Source-Bibliotheken wie

JAX-Privacy [98]. Für die nachfolgenden Experimente wird das Framework PyTorch mit der Erweiterung Opacus genutzt.

5.3 Technische Implementierung von DPSGD

Opacus ist eine Bibliothek, welches das Framework PyTorch erweitert, indem das Training eines neuronalen Netzes mittels DPSGD ermöglicht wird [96]. Der Großteil des PyTorch Codes bleibt dabei unverändert, wird jedoch um einige Wrapper-Klassen der Opacus Bibliothek erweitert.

```
1  import torch
2
3  model:torch.nn.Module = <...>
4  criterion:torch.nn.Module._Loss = torch.nn.CrossEntropyLoss()
5  optimizer:torch.optim.Optimizer = torch.optim.Adam(model.parameters(),
6                                                     lr=0.01)
7  train_dataloader:torch.utils.data.DataLoader = <...>
8
9  for epoch in range(10):
10     for inputs, labels in train_dataloader:
11         optimizer.zero_grad()
12         model_outputs = model(inputs)
13         loss = criterion(model_outputs, labels)
14         loss.backward()
15         optimizer.step()
```

Listing 5.1: Training einer Epoche in PyTorch

Listing 5.1 zeigt die wichtigsten Komponenten beim Training eines neuronalen Netzes mit PyTorch. Die Zeilen 3 bis 7 definieren die Objekte, die für das Training genutzt werden. Das Modell, welches trainiert werden soll, ist dabei vom Datentyp **torch.nn.Module** und wird in Zeile 3 als Platzhalter definiert. Zeile 4 definiert die Verlustfunktion, welche in diesem Beispiel **torch.nn.CrossEntropyLoss** ist. Als **Optimizer** wird **torch.optim.Adam** in Zeile 5 und 6 deklariert. Bei einem **Dataloader** handelt es sich um ein Objekt, welches Batches von Datensätzen und die zugehörigen Labels über einen Iterator ausgibt. Die Zeilen 10 bis 15 trainieren eine Epoche. Dabei wird jeder Batch, welche von dem **Dataloader** in Zeile 10 stammen, durch das Modell inferiert (Zeile 12). Mittels der Verlustfunktion werden die Gradienten der einzelnen Gewichte bestimmt (Zeile 13 und 14). Der **Optimizer** passt anschließend die Gewichte in die entgegengesetzte Richtung der Gradienten, skaliert mit der Lernrate, an (Zeile 15). Das Einbetten der Trainingslogik in eine For-Schleife ermöglicht das Training mehrerer Epochen hintereinander.

Opacus bietet Wrapper-Klassen, welche die Nutzung von DPSGD ermöglicht. Listing 5.2 zeigt, wie diese genutzt werden. Zuerst wird ein Objekt des Typs **PrivacyEngine** in Zeile 3

deklariert. Der Parameter "*rdp*" als "*accountant*" sorgt dafür, dass die Moments Berechnung aus Kapitel 3.3.1 verwendet wird. Die *make_private()*-Methode der **PrivacyEngine** akzeptiert das **Modell**, den **Optimizer** und den **Dataloader** und gibt diese jeweils in einer Wrapper-Klasse zurück. Das Modell ist anschließend vom Datentyp **GradSampleModule** der Opacus Bibliothek. Normale PyTorch Modelle, vom Typ **torch.nn.Module**, berechnen aggregiert die Gradienten pro Batch, wohingegen die Wrapper-Klasse von Opacus, **GradSampleModule**, für jeden Datensatz eines Batches die Gradienten berechnet. Grund dafür ist, dass die einzelnen Gradienten für das Clipping und das Verrauschen benötigt werden [43]. Das Clipping und Verrauschen übernimmt jedoch die Wrapper-Klasse des Optimizers. Diese ist vom Datentyp **DPOptimizer** der Opacus Bibliothek. Der PyTorch **Dataloader** wird zu einem Opacus **DPDataloader** transformiert, welcher anschließend Batches mittels des Poisson-Samplings zusammenstellt. Die *make_private()*-Methode benötigt noch zwei weitere Parameter: *max_grad_norm* und *noise_multiplier*. Bei dem Wert von *max_grad_norm* handelt es sich um die maximale Größe eines Gradienten, welche durch das Clipping beschränkt wird. Der *noise_multiplier* definiert dabei die Stärke des Rauschens. Diese wird mit der *max_grad_norm* multipliziert und anschließend als Standardabweichung σ des Gauß-Mechanismus, welcher für das Verrauschen der Gradienten zuständig ist, genutzt.

```
import opacus

privacy_engine = opacus.PrivacyEngine(accountant="rdp")
model, optimizer, data_loader = privacy_engine.make_private(
    module=model,
    optimizer=optimizer,
    data_loader=data_loader,
    noise_multiplier=1.0,
    max_grad_norm=1.0
)
#Hinterher normales Training des Modells
```

Listing 5.2: Opacus Wrapper für DPSGD

Nachdem die *make_private()*-Methode aufgerufen wurde und die entsprechenden Objekte von Opacus in Wrapper-Klassen umgewandelt wurden, wird das Modell mit gewöhnlichen PyTorch Code (Listing 5.1) trainiert. Um den ϵ-Wert des Privacy Budgets zu begrenzen, kann nach jedem Training eines Batches oder einer Epoche der aktuelle ϵ-Wert abgefragt werden und beim Überschreiten eines Grenzwerts das Training vorzeitig beendet werden. Listing 5.3 zeigt den entsprechenden Methodenaufruf. Dabei wird der δ-Wert des Privacy Budgets als Parameter mitgegeben. Dieser wird im Voraus definiert und sollte kleiner als die Inverse der Anzahl an Datensätzen des Trainingsdatenbestands sein.

```
privacy_engine.accountant.get_epsilon(delta=1e-5)
```

Listing 5.3: Berechnung von Epsilon

Ein Nachteil der Nutzung der *make_private()*-Methode besteht darin, dass ein Wert für das Rauschen im Voraus festgelegt werden muss. Bei einer zufälligen Wahl dieses Werts, ist unklar, wie viele Epochen trainiert werden können, bevor ein gewisses Privacy Budget überschritten wird. Um dieses Problem zu umgehen, kann als Alternative die Methode *make_private_with_epsilon()* genutzt werden. Diese ist in Listing 5.4 zu sehen. Anstatt dass hier eine Stärke des Rauschens übergeben wird, werden die gewünschte Anzahl an Epochen sowie das gewünschte Privacy Budget angegeben. Anschließend berechnet Opacus die Stärke des Rauschens, sodass in etwa nach der festgelegten Anzahl an Epochen das Privacy Budget erreicht ist. Jedoch kann der Wert des Privacy Budgets leicht abweichen. Dies liegt daran, dass durch das Poisson-Sampling die Anzahl an Datensätzen je Batch abweichen kann. Dadurch werden unterschiedlich viele Gradienten in jeder Epoche verrauscht. Die Abweichung ist deshalb bei einer kleinen Anzahl an Epochen am größten. Im Durchschnitt sollte die Größe der Batches jedoch der konfigurierten Batch-Größe entsprechen, sodass sich diese Abweichung über mehrere Epochen hinweg ausgleicht.

```
1  import opacus
2
3  privacy_engine = opacus.PrivacyEngine(accountant="rdp")
4  model, optimizer, data_loader = privacy_engine.make_private_with_epsilon(
5      module=model,
6      optimizer=optimizer,
7      data_loader=data_loader,
8      epochs=10,
9      target_epsilon=10,
10     target_delta=1e-5,
11     max_grad_norm=1.0
12 )
13 #Hinterher normales Training des Modells
```

Listing 5.4: Opacus Wrapper mit definiertem Epsilon

Einschränkungen der Modellarchitektur

Nicht jede Schicht eines Modells wird von DPSGD unterstützt und kann deshalb genutzt werden. Opacus bietet eine Klasse **ModuleValidator** an, welche überprüft, ob ein Modell kompatibel ist und bei Bedarf inkompatible Schichten austauscht. Listing 5.5 zeigt, wie diese Klasse genutzt werden kann. Die *validate()*-Methode aus Zeile 1 gibt dabei lediglich eine Liste mit inkompatiblen Schichten, sowie jeweils einer kurzen Erklärung dazu, zurück. Um die inkompatiblen Schichten mit kompatiblen, alternativen Schichten ausgetauscht werden, kann die *fix()*-Methode aufgerufen werden.

```
1  incompatible_layers = opacus.validators.ModuleValidator.validate(model)
2  model = opacus.validators.ModuleValidator.fix(model)
```

Listing 5.5: Opacus ModuleValidator

Ein Beispiel für eine nicht kompatible Schicht ist die sogenannte **BatchNorm**-Schicht. Diese Schicht berechnet Mittelwerte und Standardabweichung über mehrere Datensätze eines Batches und nutzt diese anschließend für die Normalisierung der Datensätze. Folglich wird ein Datensatz durch andere Datensätze in dem gleichen Batch beeinflusst, was jedoch nicht gewünscht ist. Dies liegt daran, dass laut DPSGD (Kapitel 3.3.1) ein Datensatz (ϵ,δ)-Differential Privacy in Bezug auf einen Batch erfüllen soll [43]. Alternative Schichten sind die **LayerNorm**-Schicht, die **InstanceNorm**-Schicht oder die **GroupNorm**-Schicht. All diese Schichten dienen ebenfalls der Normalisierung, jedoch nicht anhand anderer Datensätze eines Batches [99].

Laufzeit und Speicherverbrauch

Wie bereits erwähnt, berechnen normale PyTorch Modelle die Gradienten der Gewichte aggregiert über einen Batch. Der Grund dafür ist die Performance, denn die Berechnung von Gradienten einzelner Datensätze, ist beim Training ohne Differential Privacy nicht nötig. DPSGD benötigt jedoch genau diese Gradienten einzelner Datensätze, um diese per Clipping und Verrauschen zu verändern [43], weshalb diese in Opacus berechnet werden [96]. Jedoch sorgt dies für eine längere Laufzeit, sowie einen erhöhten Speicherverbrauch.

Der Faktor, um welchen sich die Laufzeit erhöht, hängt dabei primär von den Arten der Schichten eines neuronalen Netzes ab. Faltungsschichten, Normalisierungsschichten und sogenannte Multi-Head-Attention-Schichten sorgen für eine Erhöhung der Laufzeit um den Faktor 1,2 bis 2,9. Bei vollständig verbundenen Schichten und den sogenannten Embedding-Schichten, skaliert der Faktor der Laufzeit mit der Batch-Größe [96]. Der Speicherverbrauch skaliert ebenfalls mit der Batch-Größe. Dies liegt daran, dass für jedes Gewicht nicht nur ein Gradient im Speicher gehalten wird, sondern jeweils ein Gradient für jeden Datensatz eines Batches [96].

Das Poisson-Sampling von DPSGD sorgt dafür, dass die tatsächliche Batch-Größe von der festgelegten Batch-Größe abweichen kann. Dies kann jedoch zu einem Problem werden, wenn die festgelegte Batch-Größe so ausgewählt wurde, dass der zur Verfügung stehende Speicher der GPU nahezu vollständig ausgenutzt wird. Weicht die Batch-Größe nach oben ab, hat also mehr Datensätze als festgelegt, kann es sein, dass der Speicher nicht ausreicht und eine sogenannte **OutOfMemoryException** geworfen wird. Dies sorgt für einen Abbruch des Trainings, woraufhin der Speicher geleert werden müsste und das Training neu gestartet werden müsste. Um dies zu verhindert, bietet Opacus eine Lösung in Form der Klasse **BatchMemoryManager** an. Diese Klasse wrappt den Dataloader nochmals in eine Wrapper-Klasse, bei welcher ein Parameter "*max_physical_batch_size*" mitgegeben wird. Der neue, zurückgegebene Dataloader beschränkt die tatsächliche Batch-Größe auf die als Maximum festgelegte Batch-Größe, sodass ein Überlaufen des Speichers verhindert wird. Listing 5.6 zeigt, wie der BatchMemoryManager in den Trainingsprozess integriert wird.

Außerdem kann dieser auch genutzt werden, um mit virtuellen Batches zu trainieren. Dabei kann die Batch-Größe des Dataloaders höher konfiguriert werden, als die maximale Batch-Größe des BatchMemoryManagers. Dies sorgt dafür, dass die einzelnen Gradienten jeden Datensatzes anhand der kleineren Batches des BatchMemoryManagers berechnet werden. Jedoch erfolgt die Aggregierung der Gradienten und das Anpassen der Gewichte erst, wenn die konfigurierte Batch-Größe des Dataloaders erreicht wurde.

```
with BatchMemoryManager(
    data_loader=train_dataloader,
    max_physical_batch_size=64,
    optimizer=optimizer)
as memory_safe_data_loader:
        for model_inputs,labels in mmemory_safe_data_loader:
            # Training ...
```

Listing 5.6: Opacus BatchMemoryManager

Tabelle 5.3 zeigt, wie hoch der Speicherverbrauch bei den bereits beschriebenen Use Cases ist. Dabei wird der Speicher der Grafikkarte, auch VRAM genannt, betrachtet, da dieser oftmals ein limitierender Faktor ist. Für das CIFAR-10 Modell, welches eine ResNet-Architektur nutzt, steigt der Speicherverbrauch bei der Nutzung von DPSGD mit Opacus um den Faktor 1,8 bis 5,0. Für beide Modelle, welche Merkmale aus Gesichtern erkennen, wurde jeweils nur eine Batch-Größe betrachtet. Dabei ist die Batch-Größe eine Zweierpotenz, die so gewählt ist, dass der entsprechende Batch mit DPSGD und Opacus noch in den Speicher passt. Der maximale Speicher der genutzten Hardware liegt bei 24 GB VRAM (siehe Anhang A). Der Speicherverbrauch des ResNet-18-Modells steigt um den Faktor 2,8 bei einer Batch-Größe von 64. Für das Vision Transformer Modell kann jedoch nur eine Batch-Größe von 16 ausgewählt werden, da hier mit 20,8 GB Speicherauslastung fast das Limit der Hardware erreicht ist. Die relativ geringe Batch-Größe sorgt lediglich für eine Differenz von 1,6 Mal so viel Speicherauslastung, wie ohne DPSGD.

Modell	Batch-Größe	Speicherverbrauch ohne DPSGD	Speicherverbrauch mit DPSGD	Faktor Differenz
CIFAR-10 Modell	64	2,8 GB VRAM	5,1 GB VRAM	1,8
CIFAR-10 Modell	128	2,9 GB VRAM	10,2 GB VRAM	3,5
CIFAR-10 Modell	256	3,3 GB VRAM	16,4 GB VRAM	5,0
CelebA ResNet-18	64	4,6 GB VRAM	12,8 GB VRAM	2,8
CelebA ViT	16	12,8 GB VRAM	20,8 GB VRAM	1,6

Tabelle 5.3: Speicherverbrauch mit und ohne Differential Privacy

Tabelle 5.3 zeigt, wie die Trainingsdauer einer Epoche für die Modelle variieren kann. Dabei wurden die gleichen Batch-Größen gewählt, wie bei der Messung des Speicherverbrauchs. Die Trainingsdauer des CIFAR-10 Modells erhöht sich um einen Faktor von 1,3 bis 3,1.

Da eine Epoche jedoch nur wenige Sekunden benötigt, fällt die absolute Differenz relativ gering aus. Die Modelle der Merkmalserkennung sind demgegenüber komplexer und benötigen ohne DPSGD bereits mehrere Minuten für das Training einer Epoche. Jedoch erhöht sich der Faktor hier auf 4,5 oder 10,0. Dies liegt zum einen an den Arten der Schichten und zum anderen an einem Softwarefehler des Frameworks. Das ResNet-18-Modell nutzt einige Normalisierungsschichten, das Vision Transformer Modell nutzt einige Multi-Head-Attention Schichten für die Implementierung eines Aufmerksamkeitsmechanismus. Die zusätzliche Laufzeit dieser Schichten, skaliert bei Opacus mit der Batch-Größe, wodurch hier ein höherer Multiplikationsfaktor entstehen kann. Zusätzlich gibt es einen Softwarefehler des Dataloaders in Kombination mit Opacus, welcher dazu führt, dass das Laden von Daten nur in einem Thread ausgeführt werden kann und nicht zeitgleich in mehreren. Der Fehler wird im folgenden genauer beleuchtet. Der gleiche Fehler existiert auch bei den CIFAR-10 Modellen, fällt da jedoch weniger ins Gewicht, da das Laden der Bilder aufgrund der geringeren Auflösung weniger Zeit benötigt.

Modell	Batch-Größe	Dauer Epoche ohne DPSGD	Dauer Epoche mit DPSGD	Faktor Differenz
CIFAR-10 Modell	64	7 Sekunden	22 Sekunden	3,1
CIFAR-10 Modell	128	7 Sekunden	18 Sekunden	2,6
CIFAR-10 Modell	256	14 Sekunden	18 Sekunden	1,3
CelebA ResNet-18	64	3 Minuten	30 Minuten	10,0
CelebA ViT	16	11 Minuten	50 Minuten	4,5

Tabelle 5.4: Dauer des Trainings einer Epoche mit und ohne Differential Privacy

Softwarefehler

Da Opacus eine optionale Erweiterung von PyTorch ist, ist die Bibliothek weniger genutzt. Dies ist über die Statistiken von GitHub erkennbar. Während es auf GitHub fast 304.000 Repositories gibt, die PyTorch benutzen[2], sind es bei Opacus lediglich knapp 600[3]. Laut diesen Zahlen, nutzen nur ca. 0,2 % der Projekte mit PyTorch zusätzlich noch Opacus. Dies kann ein Grund sein, warum deutlich weniger Ressourcen in die Entwicklung von Opacus fließen. Bei PyTorch haben fast 3000 Personen über GitHub etwas zur Entwicklung des Frameworks beigetragen, wohingegen bei Opacus lediglich 55 Personen beteiligt waren. Dies könnte ein Grund sein, warum Opacus einige Softwarefehler hat, oder warum einige Features nicht vollumfänglich funktionieren.

Ein Softwarefehler, der die Trainingsdauer der Epoche erhöht, entsteht bei der Parallelisierung des Dataloaders. Der Dataloader kann über einen Parameter "*num_workers*" eine

[2] https://github.com/pytorch/pytorch
[3] https://github.com/pytorch/opacus

vorgegebene Anzahl an Threads starten. Jeder dieser Threads lädt dabei einen Batch an Datensätzen in den Speicher, sodass nach dem Training eines Batches bereits der nächste Batch zur Verfügung steht. Durch die Wrapper-Klasse in Opacus, welche das Poisson-Sampling implementiert, ist diese Parallelisierung nicht mehr möglich. Dies sorgt dafür, dass wenn die GPU einen Batch zum Training genutzt hat, es sein kann, dass der nächste Batch noch nicht geladen ist und somit die GPU auf diesen warten muss. Je komplexer die Daten und die Prozesse des Ladens sind, desto länger dauert das Laden eines Batches. Bei dem CelebA Datenbestand muss jedes Bild, welches jeweils eine Dimension von $178 \times 218 \times 3$ hat, von der Festplatte der Hardware geladen werden. Dies ist der Grund, warum das CelebA ResNet-18-Modell mit DPSGD die 10-fache Trainingsdauer einer Epoche erreicht, als im Vergleich zum Training ohne DPSGD.

Ein Beispiel für ein Feature, welches nicht vollumfänglich funktioniert, ist die Nutzung eines adaptiven Werts für das Clipping. Die Technik mit dem Namen AdaClip sorgt dafür, dass die Clipping-Norm variabel ist und sich den Gradienten anpasst [100]. Dadurch könnte der Wert möglichst gering sein, was auch das hinzugefügte Rauschen verkleinern würde. Die Klasse, welche die entsprechende Logik enthält, existiert in Opacus unter dem Namen **AdaClipDPOptimizer**. Jedoch gibt es zwei Probleme, welche die Nutzung der Klasse erschweren oder nicht möglich machen. Das erste Problem ist, dass die Übergabe der Parameter nicht korrekt funktioniert. Die *make_private()*-Methode kann über den Parameter "*clipping*" mit dem Wert "*adaptive*" so aufgerufen werden, dass der AdaClipDPOptimizer genutzt werden würde. Jedoch braucht diese Klasse zusätzliche Parameter, welche nicht über die *make_private()*-Methode weitergegeben werden können. Um dieses Problem zu umgehen, müssen die Wrapper-Klassen manuell konfiguriert werden. Das zweite Problem liegt in der Klasse selber. PyTorch hat einige Kontrollen integriert, welche sicherstellen sollen, dass das Training in einem vorgegebenen Rahmen ausgeführt wird. Eine dieser Kontrollen ist es, dass die Gradienten vor dem Training eines Batches zurückgesetzt werden, damit Gradienten nicht mehrfach genutzt werden. Bei Nutzung der Klasse AdaClipDPOptimizer kommt es jedoch zu einem Fehler dieser Kontrolle.

Obwohl die Behebung diverser Fehler und das Hinzufügen fehlender Funktionalität große Relevanz hat, ist dies im Rahmen der Arbeit nicht möglich. Deshalb werden mögliche Fehler umgangen. Die nicht mögliche Parallelisierung des Dataloaders schränkt die Funktionalität nicht ein, sondern erhöht nur die Laufzeit. Im Gegensatz dazu, ist die Nutzung der adaptiven Clipping-Norm nicht ohne entsprechenden Aufwand möglich, weshalb im folgenden lediglich konstante, festgelegte Werte für das Clipping betrachtet werden.

5.4 Hyperparameter von DPSGD

Die Güte eines Modells wird durch die Wahl der Hyperparameter beeinflusst. Dies gilt sowohl beim normalen Training, als auch beim Training mit DPSGD. Im Folgenden wird getestet, welche Parameter die Güte eines neuronalen Netzes bei der Nutzung von DPSGD beeinflussen. Dazu werden die drei Modelle, welche in Kapitel 5.1 beschrieben wurden, mit verschiedenen Hyperparametern trainiert und evaluiert. Dabei werden DPSGD spezifische Parameter, wie der ϵ-Wert und die Clipping-Norm, und auch allgemeine Parameter, wie die Batch-Größe und die Anzahl der Epochen, betrachtet. Die Lernrate wird dabei nicht konstant festgelegt, sondern über den **Adam**-Optimizer in PyTorch adaptiv konfiguriert. Der Adam-Optimizer passt die Lernrate anhand des Momentums und der Varianz der Gradienten an [101].

Bevor das eigentliche Modell trainiert wird, gibt es die Möglichkeit, die Datenmenge per Datenaugmentierung, auch Data Augmentation genannt, künstlich zu erweitern. Bei Bildern können die Werte einzelner Pixeln beeinflusst werden, indem der Farbraum der Bilder angepasst wird, oder Kontraste erhöht werden. Zusätzlich können Bilder über Rotation und Spieglung transformiert werden. Dabei können die Werte der Anpassung, z. B. die Verstärkung des Kontrasts, konfiguriert werden und an eine Wahrscheinlichkeit geknüpft werden. PyTorch bietet über AutoAugment eine Möglichkeit an, eine Kombination verschiedener Anpassungen zu nutzen [102]. Die Nutzung von AutoAugment, sowie dessen Einfluss auf die Güte der Modelle, wird folglich mitberücksichtigt.

Da das CIFAR-10 Modell die geringste Anzahl an lernbaren Parameter besitzt, wird die Anpassung verschiedener Parameter an diesem Modell zuerst evaluiert. Anschließend werden die Erkenntnisse auf das ResNet-18 Modell und das Vision Transformer Modell für den CelebA Datenbestand angewendet. Im Folgenden werden einige Auszüge aus den gesamten Ergebnissen zur Veranschaulichung herangezogen. Die vollständigen Tabellen aller Messwerte befinden sich in Anhang B.

5.4.1 Hyperparameter CIFAR-10 Modell

Für das CIFAR-10 Modell, welches kein DPSGD nutzt, werden folgende Hyperparameter betrachtet: Anzahl der Epochen, Batch-Größe und die Nutzung von AutoAugment. Dazu wurde das Modell mit einer Epochenanzahl zwischen 10 und 20 trainiert, jeweils mit einer Batch-Größe von 64, 128 oder 256. Dieser Vorgang wurde mit AutoAugment aktiviert und deaktiviert ausgeführt. Die höchste Genauigkeit wird erreicht, wenn das Modell mit Datenugmentierung und einer Batch-Größe von 64 über 15 Epochen hinweg trainiert wird. Die erreichte Genauigkeit, welche anhand der Test-Daten gemessen wurde, liegt bei 84,6 %.

Eine Batch-Größe von 64 und 128 resultieren in einer ähnlichen Güte des Modells. Jedoch sorgt eine Batch-Größe von 256 für eine geringere Genauigkeit des Modells. Dies liegt daran, dass einzelne Datensätze in der Aggregation der Gradienten untergehen können, wodurch einzelne Features der Daten weniger gut gelernt werden. Abbildung 5.2 zeigt die Güte der Modelle, welche mit den unterschiedlichen Batch-Größen trainiert wurden. Jede Linie steht dabei für eine unterschiedliche Anzahl an trainierten Epochen.

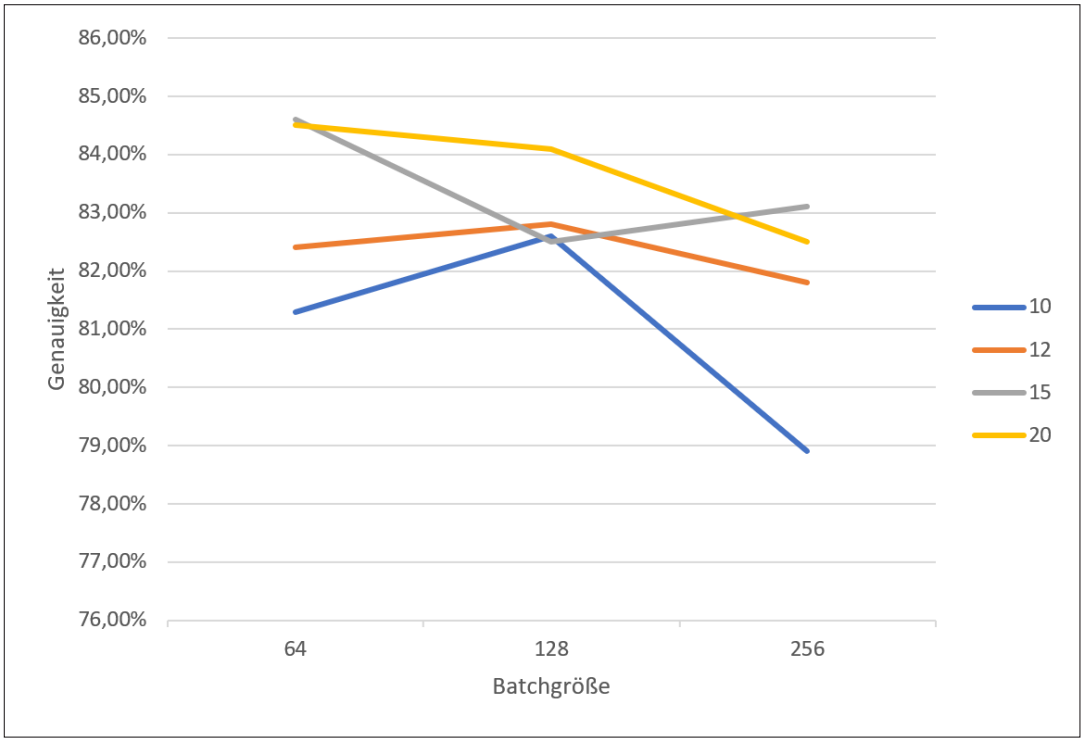


Abbildung 5.2: Auswirkung Batch-Größe auf CIFAR-10 Modell ohne DPSGD und mit Datenaugmentierung

Zusätzlich zeigt Abbildung 5.2, dass eine höhere Anzahl an Epochen mit einer höheren Genauigkeit korreliert. Dies liegt an der Nutzung von Datenaugmentierung. Wird diese nicht genutzt, liegt die optimale Anzahl an Epochen zwischen 12 und 15. Werden mehr Epochen trainiert, sinkt die Genauigkeit wieder, was auf ein Overfitting des Modells zurückzuführen ist. Generell sinkt die Genauigkeit der Modelle, wenn AutoAugment nicht genutzt wird. Durch die Datenaugmentierung wird der Datensatz künstlich erweitert, was dafür sorgt, dass die Trainingsdatenmenge vergrößert wird.

Für das CIFAR-10 Modell ohne DPSGD resultieren folgendende Hyperparameter in einer optimalen Güte des Modells:

- **Batch-Größe:** Batch-Größe kann variablen gewählt werden, sollte jedoch nicht zu groß sein. Ein Wert von 64 oder 128 ist wirksam.

- **Anzahl Epochen:** Die optimale Anzahl an Epochen ist um 15 herum. Wird Datenaugmentierung genutzt, kann die Anzahl höher sein, wenn nicht, dann sollte die Anzahl geringer sein.
- **Datenaugmentierung:** Datenaugmentierung verbessert die Güte des Modells und ermöglicht das Training mehrerer Epochen, wobei Overfitting vermieden wird. Datenaugmentierung sollte deshalb genutzt werden.

Wird das CIFAR-10 Modell mit DPSGD trainiert, müssen zusätzlich noch der ϵ-Wert des Privacy Budgets und die Clipping-Norm betrachtet werden. Der δ-Wert des Privacy Budgets wird konstant auf 10^{-5} festgelegt. Um die Hyperparameter zu evaluieren, gibt es drei unterschiedliche Versuche. Alle drei Versuche trainieren eine Vielzahl an Modellen mit DPSGD. Dabei werden ϵ-Werte von 1 bis 50, Epochenanzahlen von 1 bis 30 und Batch-Größen von 64 bis 256 genutzt. Lediglich bei der Clipping-Norm und bei der Nutzung von Datenaugmentierung unterscheiden sich die drei Versuche. Um den festgelegten ϵ-Wert nach einer festgelegten Anzahl an Epochen zu erreichen, wurde die *make_private_with_epsilon()*-Methode genutzt.

Der erste Versuch nutzt Datenaugmentierung und hat einen nicht optimierten Wert der Clipping-Norm. Dieser ist auf 1,0 festgelegt. Dieser Versuch zeigt, dass die Genauigkeit der Modelle, mit der Größe des ϵ-Werts zunimmt. Bei einem ϵ-Wert von 1 wird maximal eine Genauigkeit von 36,9 % erreicht, wohingegen bei einem ϵ-Wert von 50 eine Genauigkeit von 59,1 % erreicht wird. Ebenfalls steigt die Genauigkeit der Modelle mit der Anzahl an trainierten Epochen. Dies liegt daran nicht nur daran, dass Datenaugmentierung genutzt wird, sondern dass auch das Rauschen Overfitting verhindert. Sogar 30 Epochen können trainiert werden, ohne dass die Genauigkeit stagniert. Anders als bei dem Modell ohne DPSGD, hat die Batch-Größe eine deutliche Auswirkung auf die Genauigkeit der Modelle. Eine höhere Batch-Größe sorgt für eine verbesserte Genauigkeit der Modelle. Das Rauschen, welches den einzelnen Gradienten hinzugefügt wird, gleicht sich bei einer höheren Batch-Größe aus und nähert sich dem Erwartungswert an. Da der Erwartungswert den echten Gradienten entspricht, bedeutet dies, dass eine höhere Batch-Größe dafür sorgt, dass die verrauschten Gradienten der Gewichte, näher an den tatsächlichen Gradienten der Gewichte liegen. Abbildung 5.3 zeigt diesen Zusammenhang grafisch. Jede Linie entspricht einem Modell, welches über 15 Epochen, mit unterschiedlichem ϵ-Wert, trainiert wird.

Der zweite Versuch ähnelt dem ersten Versuch, jedoch wird die Datenaugmentierung nicht genutzt. Dies sorgt dafür, dass die Genauigkeit der Modelle steigt. Abbildung 5.4 zeigt diesen Zusammenhang. Die Modelle, dessen Genauigkeit dargestellt werden, werden jeweils 15 Epochen lang, mit einer Batch-Größe von 256, trainiert. Obwohl die Datenaugmentierung im zweiten Versuch nicht genutzt wird, kann das Modell 20 Epochen lang trainiert werden und hat dennoch eine steigende Genauigkeit. Erst im Bereich zwischen 20 und 30 Epochen stagniert die Genauigkeit. Im Vergleich zu dem Modell ohne DPSGD sind dies dennoch einige

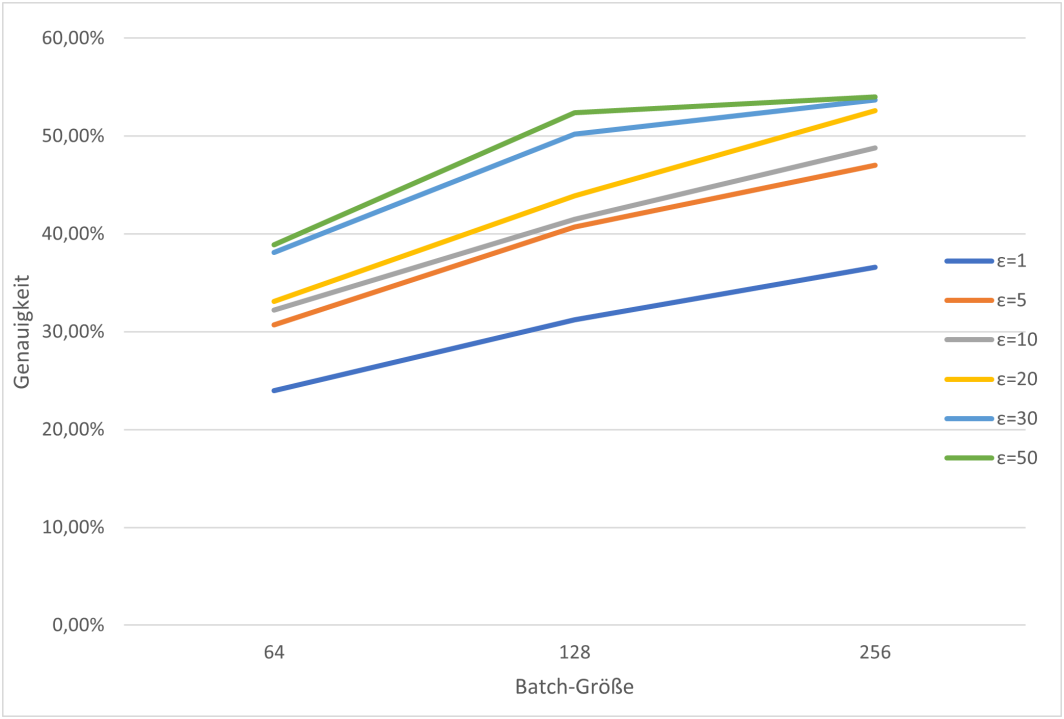


Abbildung 5.3: Auswirkung der Batch-Größe auf die Genauigkeit von CIFAR-10 Modellen

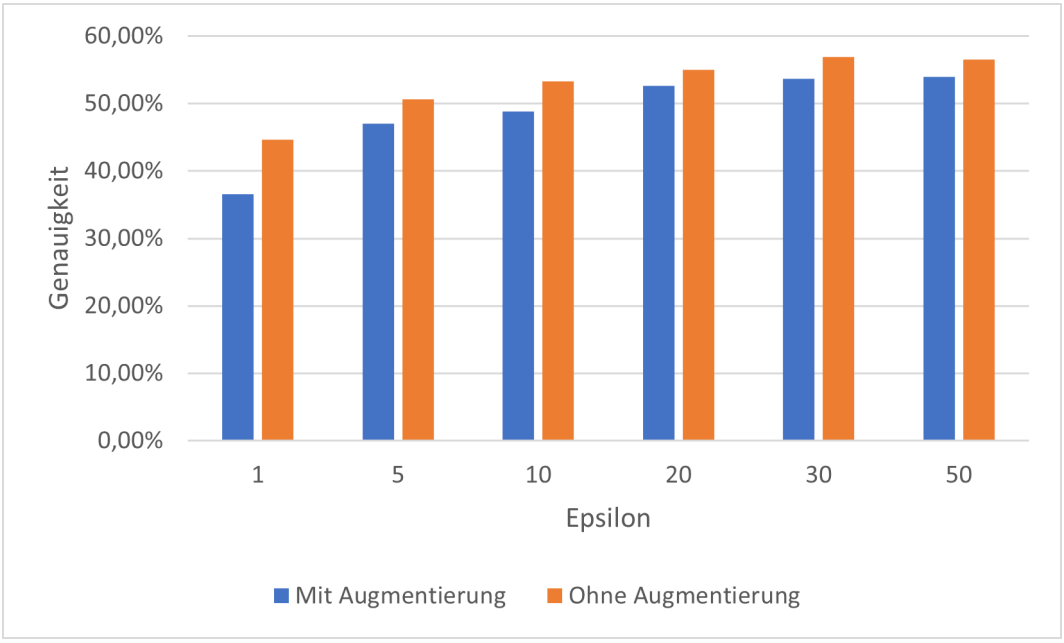


Abbildung 5.4: Auswirkung von AutoAugment auf die Genauigkeit von CIFAR-10 Modellen

Epochen mehr. Dies liegt daran, dass das Rauschen, genau wie die Datenaugmentierung, das Overfitting reduziert.

Der dritte Versuch gleicht dem zweiten Versuch, wobei hier der Wert der Clipping-Norm angepasst wurde. Da die Clipping-Norm die Stärke des Rauschens beeinflusst, kann ein

geringer Wert die Güte eines Modells erhöhen. Wird der Wert jedoch zu gering gewählt, kann dies auch für den gegenteiligen Effekt sorgen, denn die Gradienten werden zu klein, damit das Modell in den geplanten Trainingsschritten konvergiert. Um eine geeignete Clipping-Norm zu finden, wird das CIFAR-10 Modell über 5 Epochen trainiert, ohne Rauschen, jedoch mit unterschiedlichen Werten für das Clipping. Dabei werden Zehnerpotenzen genutzt. Der Wert wird immer um eine Zehnerpotenz reduziert, solange bis das Clipping einen leichten negativen Einfluss auf die Güte des Modells hat. Die resultierende Zehnerpotenz ist in diesem Fall 10^{-5}. Da diese einen leichten negativen Einfluss auf das Modell hat, bedeutet dies, dass eine Vielzahl an Gradienten durch diesen Wert begrenzt wird. Die Wahl der Clipping-Norm sollte auf diesen Wert fallen, oder einen minimal höheren Wert, welcher keinen Einfluss auf das Modell hat. Durch die Wahl dieses geringen Werts wird auch die Stärke des Rauschens reduziert. In diesem dritten Versuch wird die Clipping-Norm von 10^{-5} evaluiert.

Die Anpassung der Clipping-Norm wirkt sich nicht bei einer niedrigen Anzahl an Epochen aus. Bei den CIFAR-10 Modellen, welche 15 Epochen lang trainiert werden, ist die Genauigkeit in etwa gleich, egal ob die Clipping-Norm bei 1 oder bei 10^{-5} liegt. Abbildung 5.5 zeigt den soeben beschriebenen Zusammenhang.

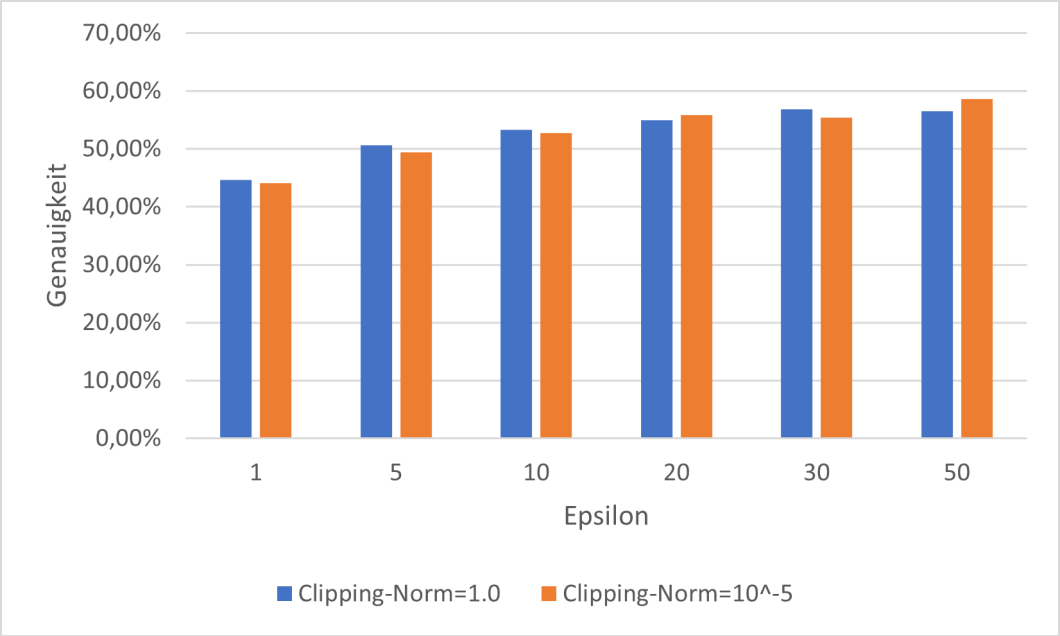


Abbildung 5.5: Auswirkung der Clipping-Norm auf die Güte von CIFAR-10 Modellen bei 15 Epochen Training

Jedoch ermöglicht die geringere Clipping-Norm, mehr Epochen zu trainieren, ohne dass die Genauigkeit der Modelle stagniert. Dies liegt daran, dass die Gradienten mit kleiner Clipping-Norm kleiner sind, als die Gradienten mit großer Clipping-Norm. Somit werden mehr Trainingsschritte benötigt, um zu konvergieren. Außerdem ist die Stärke des Rauschens geringer, was ebenfalls die Güte des Modells verbessert. Abbildung 5.6 zeigt den

Vergleich zwischen einer hohen und einer niedrigen Clipping-Norm, wenn die Modelle jeweils 30 Epochen trainiert werden.

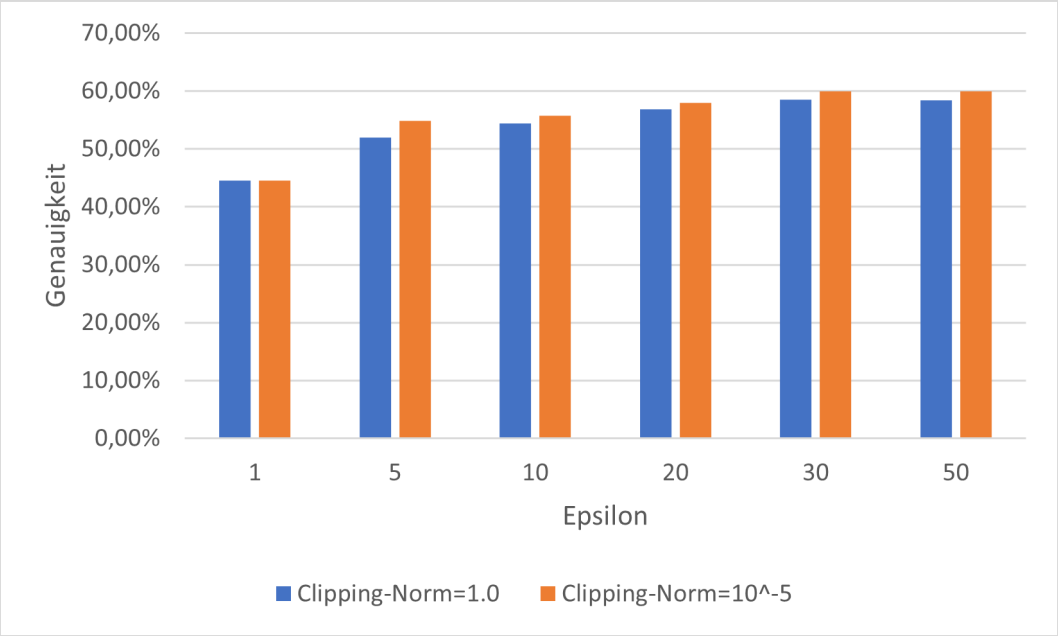


Abbildung 5.6: Auswirkung der Clipping-Norm auf die Güte von CIFAR-10 Modellen bei 30 Epochen Training

Für das CIFAR-10 Modell mit DPSGD resultieren folgende Hyperparameter in einer optimalen Güte des Modells:

- **ϵ-Wert des Privacy Budgets:** Der ϵ-Wert korreliert mit der Güte des Modells. Kleine Werte schränken die Genauigkeit der Modelle ein, können aber die Vertraulichkeit besser schützen. Kapitel 4.3 gibt dabei einen Überblick über den Schutz der Modelle.
- **Batch-Größe:** Die Batch-Größe sollte möglichst groß gewählt werden. Hier kann ebenfalls der BatchMemoryManager von Opacus helfen, welcher eine virtuelle Batch-Größe ermöglicht, ohne den Speicher zu überlasten. Eine Batch-Größe von 512 bietet jedoch keinen Vorteil mehr gegenüber einer Batch-Größe von 256.
- **Anzahl Epochen:** Die Anzahl an Epochen ist höher als bei einem Modell ohne DPSGD. Dies liegt daran, dass das Rauschen Overfitting abschwächt. Wird ein geringer Wert für die Clipping-Norm gewählt, können sogar noch mehr Epochen trainiert werden. Eine Epochenanzahl von 30 bietet immer noch eine steigende Genauigkeit, wohingegen ein Modell ohne DPSGD bereits nach 15 Epochen stagniert.
- **Datenaugmentierung:** Datenaugmentierung ist nicht notwendig und kann die Güte des Modells sogar verschlechtern.
- **Clipping-Norm:** Der Wert der Clipping-Norm sollte möglichst gering gewählt werden. Dies ermöglicht das Training von mehreren Epochen, wobei das Rauschen verringert wird.

Tabelle 5.5 zeigt, welche Genauigkeit mit verschiedenen ϵ-Werten erreicht wird, wenn die Hyperparameter nach obigen Regeln optimiert werden.

Epsilon	Anzahl Epochen	Batch-Größe	Clipping-Norm	Genauigkeit
1	30	512	10^{-5}	46,7 %
5	30	512	10^{-5}	55,0 %
10	30	512	10^{-5}	59,4 %
20	30	512	10^{-5}	60,5 %
30	30	512	10^{-5}	61,9 %
50	30	512	10^{-5}	63,2 %

Tabelle 5.5: Genauigkeit von CIFAR-10 Modellen mit DPSGD

5.4.2 Hyperparameter ResNet-18 Modell

Das originale ResNet-18 Modell ist für die Klassifikation des ImageNet Datenbestands trainiert. Da das Modell jedoch dafür genutzt werden soll, Merkmale aus Gesichtsbildern der CelebA Datenmenge zu erkennen, wird das Modell angepasst. Dafür wird die ursprünglich letzte, vollständig verbundene Schicht durch eine neue, vollständig verbundene Schicht ausgetauscht, welche nur 40 Neuronen enthält. Jedes Neuron steht dabei für eines der Merkmale.

Das ResNet-18 Modell bietet die Möglichkeit, die vortrainierten Gewichte zu nutzen, oder die Gewichte zufällig zu initialisieren. Deshalb wird der Parameter "Vortrainiert" als zusätzlicher Hyperparameter evaluiert. Dabei ist anzumerken, dass das Modell mit dem ImageNet Datenbestand vortrainiert ist, welches Bilder von Objekten enthält, jedoch nicht primär von Gesichtern.

Das vortrainierte ResNet-18 Modell ohne DPSGD erreicht eine Genauigkeit von 92,4 % nach 10 Epochen mit Datenaugmentierung. Wird stattdessen das nicht-vortrainierte Modell genommen, liegt die Genauigkeit bei 92,3 %, also minimal darunter. Das Deaktivieren der Datenaugmentierung hat jedoch einen größeren negativen Einfluss. Dadurch würde die Güte des Modells auf 91,0 % für das vortrainierte und auf 90,1 % für das nicht-vortrainierte Modell sinken.

Die Auswirkung der Hyperparameter des ResNet-18 Modells mit DPSGD ist gleich wie bei dem CIFAR-10 Modell. Höhere Batch-Größen, eine niedrige Clipping-Norm und keine Nutzung von Datenaugmentierung sorgen für eine Verbesserung der Genauigkeit des Modells. Ob das Modell dabei vortrainiert oder nicht-vortrainiert ist, spielt eine untergeordnete Rolle und wirkt sich kaum auf die Genauigkeit der Modelle aus. Anhang B enthält Tabellen

Epsilon	Clipping-Norm	Batch-Größe	Anzahl Epochen	Daten-augmentierung	Genauigkeit
∞	∞	128	10	ja	92,3 %
1	10^{-5}	256	10	nein	83,6 %
5	10^{-5}	256	10	nein	86,4 %
10	10^{-5}	256	10	nein	86,8 %

Tabelle 5.6: Übersicht der besten ResNet-18 Modelle mit DPSGD

mit detaillierten Messungen zu verschiedenen Hyperparametern von DPSGD. Im Folgenden werden nur die besten Modelle betrachtet.

Tabelle 5.6 zeigt dabei, was für eine Genauigkeit von Modellen erreicht werden kann, abhängig von unterschiedlichen ϵ-Werten. Die virtuelle Batch-Größe ist bei den DPSGD Modellen auf 256 gesetzt, wohingegen die physische Batch-Größe bei 64 liegt. Der Wert für das Clipping wird auf 10^{-5} gesetzt, welcher einen leicht negativen Einfluss auf das Modell hat, bei einem ϵ-Wert von unendlich. Datenaugmentierung verschlechtert die Güte des Modells bei der Nutzung von DPSGD, weshalb diese nicht genutzt wird. Bei einem ϵ-Wert von 1 erreicht das Modell eine Genauigkeit von 83,6 % und liegt damit absolut 8,7 % unter dem Modell ohne DPSGD. Wird der ϵ-Wert auf 10 erhöht, steigt die Genauigkeit auf 86,8 % an. Damit hat dieses Modell nur 5,5 Prozentpunkte weniger Genauigkeit, als das Modell ohne DPSGD.

5.4.3 Hyperparameter Vision Transformer Modell

Das Vision Transformer Modell in PyTorch ist standardmäßig ebenfalls für die ImageNet-Datenmenge vortrainiert. Dies bedeutet, dass hier ebenfalls die letzte, vollständig verbundene Schicht ausgetauscht wird, mit einer Schicht, welche 40 Neuronen, ein Neuron pro Merkmal, enthält.

Das vortrainierte Vision Transformer Modell ohne DPSGD erreicht eine Genauigkeit von 82,9 % nach 10 Epochen mit Datenaugmentierung. Ohne Datenaugmentierung liegt die Genauigkeit bei 83,0 % und damit minimal höher als mit Datenaugmentierung. Wird jedoch das nicht-vortrainierte Vision Transformer Modell genutzt, sinken die Genauigkeiten auf 82,4 % für das Modell ohne Datenaugmentierung und 82,3 % für das Modell mit Datenaugmentierung. Dies lässt schlussfolgern, dass die Nutzung von Datenaugmentierung kaum Einfluss auf das Modell ohne DPSGD hat. Außerdem wird deutlich, dass das Vision Transformer Modell schlechter geeignet ist für die Merkmalserkennung aus Gesichtern, als das ResNet-18 Modell.

Die Wahl der Hyperparameter hat außerdem bei diesem Modell weniger Einfluss auf die Genauigkeit, als bei den beiden vorherigen Modellen. Tabelle 5.7 zeigt, welche Genauigkeit

Epsilon	Clipping-Norm	Batch-Größe	Anzahl Epochen	Daten-augmentierung	Genauigkeit
1	1,0	16	5	ja	79,4 %
1	10^{-5}	128	5	nein	80,0 %
5	1,0	16	5	ja	78,8 %
5	10^{-5}	128	5	nein	80,0 %
10	1,0	16	5	ja	79,4 %
10	10^{-5}	128	5	nein	80,3 %

Tabelle 5.7: Vergleich der Hyperparameter bei Vision Transformer Modellen

die Vision Transformer Modelle mit DPSGD erreichen, wenn die Hyperparameter optimiert werden. Dabei wird jeweils das vortrainierte Vision Transformer Modell genutzt. Die unoptimierten Hyperparameter sind dabei eine Batch-Größe von 16, Nutzung von Datenaugmentierung und eine Clipping-Norm von 1,0. Das Optimieren der Parameter erhöht die Batch-Größe auf 128, reduziert die Clipping-Norm auf 10^{-5} und deaktiviert die Nutzung von Datenaugmentierung. Zusätzlich ist anzumerken, dass die DPSGD Modelle maximal für 5 Epochen trainiert wurden. Dies liegt daran, dass eine Epoche ca. 50 Minuten Zeit benötigt, was die Evaluierungsmöglichkeiten einschränkt. Zudem gibt es keine signifikante Verbesserung der Genauigkeit bei einem Training des Vision Transformer Modells mit DPSGD von 3 Epochen und 5 Epochen. Dies lässt andeuten, dass es ebenfalls keine signifikante Verbesserung der Genauigkeit bei 5 Epochen und 10 Epochen gibt. Das Optimieren der Hyperparameter sorgt für eine absolute Verbesserung der Genauigkeit von 0,6, 1,2 und 0,8 Prozentpunkte, je nach Wahl von ϵ.

Epsilon	Clipping-Norm	Batch-Größe	Anzahl Epochen	Daten-augmentierung	Genauigkeit
∞	∞	64	10	nein	83,0 %
1	10^{-5}	128	5	nein	80,0 %
5	10^{-5}	128	5	nein	80,0 %
10	10^{-5}	128	5	nein	80,3 %

Tabelle 5.8: Übersicht der besten Vision Transformer Modelle mit DPSGD

Tabelle 5.8 stellt die Genauigkeit des besten Vision Transformer Modells ohne DPSGD in Relation zu den Genauigkeiten der Vision Transformer Modelle mit DPSGD und optimierten Hyperparametern. Die Genauigkeit des Modells ohne DPSGD ist bis zu 3 Prozentpunkte höher, als bei den Modellen mit DPSGD. Damit fällt Differenz der Genauigkeit von Modellen ohne DPSGD und mit DPSGD geringer aus, als bei den beiden vorherigen Modellarten.

5.4.4 Handlungsempfehlung für die Wahl der Hyperparameter

Die folgende Handlungsempfehlung zeigt, wie die Hyperparameter optimiert werden müssen, um die bestmögliche Güte eines Modells mit DPSGD zu erhalten. Der ϵ-Wert wird hierbei außer Betracht gelassen. Datenaugmentierung gilt es ebenfalls zu vermeiden.

Batch-Größe

Da das Training mit DPSGD von Opacus mehr Speicher als das gleiche Training ohne DPSGD benötigt, muss zuerst eine geeignete Batch-Größe herausgefunden werden. Dabei gibt es eine physische Batch-Größe, die Batch-Größe, die wirklich gleichzeitig in den Speicher passt, und eine virtuelle Batch-Größe, die größer sein kann. Um die physische Batch-Größe zu finden, müssen die entsprechenden PyTorch Klassen in die Opacus Wrapper-Klassen eingebunden werden. Die Parameter bzgl. dem Rauschen und Clipping können dabei ignoriert werden. Anschließend kann ein Batch genutzt werden, um die Gradienten zu berechnen. An dieser Stelle ist es möglich, den Speicherverbrauch zu überwachen und so lange die Batch-Größe anzupassen, bis der Speicher optimal ausgenutzt ist. Die virtuelle Batch-Größe kann größer gewählt werden, als die physische Batch-Größe. Eine relativ große Batch-Größe sorgt dafür, dass die Werte des Rauschens sich ausgleichen. Eine zu große Batch-Größe hat jedoch auch negative Folgen. Es empfiehlt sich eine Batch-Größe von 128, 256 oder 512.

Anzahl der Epochen

Um die Stärke des Rauschens zu setzen, ist es wichtig, die geplante Anzahl an Epochen im Voraus zu kennen. Die Anzahl der Epochen, welche bei einem Training mit DPSGD benötigt werden, können von dem Modell ohne DPSGD abgeleitet werden. Da das Hinzufügen von Rauschen als eine Art Regularisierung betrachtet werden kann, wird Overfitting durch DPSGD vermieden. Somit könnten einige Epochen mehr trainiert werden, als bei dem Modell ohne DPSGD. Die Evaluierung des ResNet-18 Modells und des Vision Transformer Modells haben jedoch kaum Verbesserungen bei einer hohen Anzahl an Epochen gezeigt. Ein guter Richtwert ist es also, dass das Modell mit DPSGD für die gleiche Anzahl an Epochen trainiert wird, wie das Modell ohne DPSGD.

Clipping-Norm

Der Wert des Clippings hat direkten Einfluss auf die Stärke des Rauschens und sollte deshalb klein gesetzt werden. Ist der Wert jedoch zu klein, werden die Gradienten ebenfalls zu klein und das Modell wird nicht konvergieren. Um eine ideale Clipping-Norm zu ermitteln, kann ein Modell für eine kleine Anzahl an Epochen mit DPSGD und Opacus trainiert werden.

Der Wert des Rauschens wird dabei auf 0 gesetzt, wodurch sich ein ϵ-Wert von unendlich ergibt. Anschließend werden mehrere Modelle trainiert, welche jeweils eine unterschiedliche Zehnerpotenz als Clipping-Norm haben. Die Clipping-Norm wird jeweils um eine Zehnerpotenz reduziert, solange bis ein Modell eine signifikant schlechtere Güte zeigt. Dies bedeutet, dass diese Zehnerpotenz als Clipping-Norm eine signifikante Menge an Gradienten begrenzt und somit ein geeigneter Wert ist.

5.5 Angriffe

Das Ziel von Differential Privacy in Kombination mit neuronalen Netzen, ist es, Angriffe abzuschwächen. Im Folgenden wird anhand zweier Angriffe evaluiert, welchen Effekt die Nutzung von DPSGD hat. Bei den zwei genutzten Angriffen handelt es sich um die Membership Inference Attacke und die Model Inversion Attacke.

5.5.1 Membership Inference Attacke

Die Membership Inference Attacke soll zeigen, ob ein Datensatz in der Trainingsdatenmenge eines Modells enthalten ist oder nicht. Dazu werden Shadow Modelle genutzt, welche das anzugreifende Modell nachahmen. Die Vorhersagen der Shadow Modelle werden genutzt, um einen Meta-Klassifikator zu trainieren, welcher bestimmt, ob ein Datensatz bei einem Modell zum Training genutzt wurde.

Eine Besonderheit dieses Angriffs ist es, dass sowohl eine Black-Box, als auch eine White-Box Variante möglich ist. Bei der White-Box Variante kennt der Angreifer die genaue Architektur des anzugreifenden Modells und kann die Architektur der Shadow Modell nachbilden. In der Black-Box Variante, wird lediglich versucht, ähnliche Architekturen zu kreieren. Dieses Experiment nutzt den White-Box Ansatz. Dies liegt daran, dass es bei einem Black-Box Ansatz sein könnte, dass der Angreifer, durch Glück, eine nahezu identische Architektur der Shadow Modelle wählt. Somit kann ein Black-Box Ansatz theoretisch genauso effektiv sein, wie der White-Box Ansatz.

Membership Inference Attacke gegen das CIFAR-10 Modell

Die CIFAR-10 Datenmenge besteht bereits aus einer Trainingsdatenmenge und einer Testdatenmenge, welche in Summe aus 60000 Datensätzen bestehen. Für das Training der Shadow Modelle werden die beiden Datenmengen kombiniert. Anschließend wird für jedes Shadow Modell zufällig eine Datenmenge, bestehend aus 50000 Datensätzen, für das Training ausgewählt. Somit dienen die ausgewählten Daten nicht nur für das Training, sondern haben zusätzlich das Label "included", wohingegen die nicht ausgewählten Daten das Label

"excluded" erhalten. Die Labels dienen im Anschluss dazu, die Datenmenge für den Meta-Klassifikator zu konstruieren. Damit die Anzahl der beiden Labels in der Datenmenge gleich ist, werden von der genutzten Trainingsdatenmenge jedes Shadow Modells nur 10000 Datensätze gelabelt. Somit entstehen pro Shadow Modell 20000 gelabelte Datensätze, jeweils 10000 mit dem Label "included" und 10000 mit dem Label "excluded".

Der Meta-Klassifikator ist ein neuronales Netz, welches nur aus vollständig verbundenen Schichten besteht. Dabei werden 10 Neuronen als Eingabe genutzt, je ein Neuron pro Vorhersagewahrscheinlichkeit einer Klasse, und ein Neuron als Ausgabe, welches das Label "included" oder "excluded" bestimmt. Die Vorhersagewahrscheinlichkeiten, welche als Eingabe dienen, werden von den Shadow Modellen durch die Softmax-Funktion in den Wertebereich (0,1) übertragen, wobei eine Klasse einen Wert nahe 1 hat, die restlichen Klassen einen Wert von 0.

Die Trainingsdatenmenge und Testdatenmenge des CIFAR-10 Datenbestands könnte genutzt werden, um die Membership Inference Attacke zu bewerten, jedoch haben die Datenmengen eine ungleiche Anzahl an Datensätzen. Würde der Meta-Klassifikator immer das Label "included" bestimmen, unabhängig des Inputs, dann entspräche dies einer Genauigkeit von ca. 83,3 %, da die Anzahl der Datensätze in den beiden Datenmengen das gleiche Verhältnis haben. Deshalb werden aus der Trainingsdatenmenge zufällig 10000 Datensätze ausgewählt. Diese, sowie die 10000 Datensätze der Testdatenmenge, dienen dann zur Evaluierung des Angriffs. Dazu werden die Datensätze durch das anzugreifende Modell inferiert und die Vorhersagewahrscheinlichkeiten als Eingabe des Meta-Klassifikators genutzt.

Die Modelle, an welchen die Membership Inference Attacke getestet wird, sind die Modelle, welche in Kapitel 5.4 trainiert werden. Dabei wird das beste Modell ohne DPSGD genutzt, sowie jeweils das Modell mit DPSGD, welches mit optimierten Hyperparametern trainiert wurde.

Tabelle 5.9 zeigt die Genauigkeit, welche der Meta-Klassifikator bei der Ausführung der Membership Inference Attacke gegen das CIFAR-10 Modell ohne DPSGD hat. Dabei wurden eine unterschiedliche Anzahl an Shadow Modellen evaluiert, welche jeweils für 15 Epochen trainiert wurden und dadurch eine ähnliche Genauigkeit wie das originale Modell haben. Die Genauigkeit des Angriffs ist bei 8,16 und 32 Shadow Modellen ähnlich. Diese beträgt zwischen 57,4 % und 58,9 %.

Epsilon	Anzahl Shadow Modelle	Genauigkeit des Angriff
∞	8	58,0 %
∞	16	58,9 %
∞	32	57,4 %

Tabelle 5.9: Membership Inference Angriff gegen CIFAR-10 Modell ohne DPSGD

Tabelle 5.10 zeigt, wie sich die Genauigkeit des Angriffs durch die Nutzung von DPSGD ändert. Bei allen Modellen mit DPSGD, hat die Membership Inference Attacke eine Genauigkeit zwischen 49,8 % und 50,2 %. Dadurch ist der Angriff weniger effektiv, als ohne die Nutzung von DPSGD.

Epsilon	Anzahl Shadow Modelle	Genauigkeit des Angriff
∞	32	57,4 %
1	32	49,9 %
5	32	49,8 %
10	32	50,1 %
20	32	50,2 %
30	32	50,2 %
50	32	50,0 %

Tabelle 5.10: Membership Inference Angriff gegen CIFAR-10 Modelle mit DPSGD

Das Problem des Angriffs ist jedoch, dass dieser nur minimal effektiv gegen ein Modell ohne DPSGD ist. Würde der Meta-Klassifikator zufällig ein Label wählen, würde dies einer Genauigkeit von ca. 50 % entsprechen. Gegen ein Modell ohne DPSGD, erreicht der Meta-Klassifkator eine Genauigkeit von bis zu 58,9 %, was eine Verbesserung ist. Jedoch ermöglicht dies nicht, eine sachdienliche Aussage zu treffen. Zusätzlich hat der Angreifer in der Regel keine Möglichkeit, den Angriff zu evaluieren. Dies liegt daran, dass der Angreifer keine Informationen bzgl. der genutzten Trainingsdatenmenge hat. Die geringe Effektivität des Angriffs, sogar in der White-Box Variante, macht die Nutzung von DPSGD entbehrlich.

Membership Inference Attacke gegen das ResNet-18 Modell

Die Membership Inference Attacke wird nur gegen das ResNet-18 Modell evaluiert, jedoch nicht gegen das Vision Transformer Modell. Dies liegt an den benötigten Ressourcen, eine Vielzahl an Shadow Modellen zu trainieren, welche jeweils die Vision Transformer Architektur nutzen.

Das ResNet-18 Modell erkennt 40 Merkmale anhand von Gesichtsbildern des CelebA Datenbestand. Im Gegensatz zu dem CIFAR-10 Modell wird nicht eine Klasse vorhergesagt, sondern 40 unabhängige Merkmale. Dabei handelt es sich um eine sogenannte Multi-Label Klassifizierung, also einer Klassifikation bei welcher mehrere Labels vorhergesagt werden können. Für das Modell sind diese 40 Labels unabhängig, fachlich sind jedoch einige dieser Labels verbunden. So gibt es beispielsweise unterschiedliche Labels, welche für unterschiedliche Haarfarben stehen. Da jede Person in dem Datenbestand eine Haarfarbe hat, hat maximal eines der Haarfarben-Labels den Wert 1, wohingegen die restlichen Haarfarben den Wert 0 haben.

Die Membership Inference Attacke gegen die ResNet-18 Modelle werden simultan zu den Angriffen gegen die CIFAR-10 Modelle durchgeführt. Tabelle 5.11 zeigt die Genauigkeit des Angriffs gegen Modelle mit einem unterschiedlichen ϵ-Wert. Es wurden jeweils 8 Shadow Modelle trainiert und zum Labeln genutzt. Der Angriff gegen das Modell ohne DPSGD hat eine Genauigkeit von 52,1 % und ist damit nur minimal besser, als bei einer zufälligen Vorhersage. Gegen Modelle die DPSGD benutzten ist der Angriff noch weniger effektiv und erreicht eine Genauigkeit von maximal 50,6 %.

Epsilon	Genauigkeit des Angriff
∞	52,1 %
1	50,2 %
5	50,6 %
10	50,1 %

Tabelle 5.11: Membership Inference Angriff gegen ResNet-18 Modelle mit DPSGD

5.5.2 Model Inversion Attacke

Bei der Model Inversion Attacke werden die Ausgaben eines Modells genutzt, um Rückschlüsse zu den Trainingsdaten zu ziehen. Der hier evaluierte Angriff wird auch Reconstruction Attacke genannt. Ziel ist es, einen Datensatz aus dem Trainingsdatenbestand nachzubilden, lediglich anhand des Modells. Hierfür wird ein Startbild in das Modell gegeben, welches zu Beginn lediglich eine Ansammlung von zufälligen Pixeln ist. Die Vorhersage des Modells wird anschließend mit einer Verlustfunktion durch das Modell backpropagiert, jedoch werden nicht die Gewichte des Modells angepasst, sondern die Werte des Startbildes. Dadurch verändert sich das Bild, bis es das gewünschte Label zeigt. Der Vorgang wird dabei iterativ wiederholt. Zusätzlich wird in jedem Schritt das Bild mittels eines Autoencoders entrauscht, sodass sich die zufällige Ansammlung von Pixeln zu einem möglichst realistischen Bild umwandeln. Der iterative Vorgang kann nach einer festgelegten Anzahl an Wiederholungen gestoppt werden, oder bis das rekonstruierte Bild den gewünschten Detailgrad erreicht hat.

Model Inversion Attacke gegen CIFAR-10 Modell

Um die Model Inversion Attacke durchzuführen, wurde zu Beginn ein Autoencoder trainiert. Dieser soll Bilder entrauschen, d. h., das Rauschen aus Bildern entfernen und diese realistischer werden lassen. Als Ausgabe des Modells werden daher die Bilder des CIFAR-10 Trainingsdatenbestand genutzt, wobei die Eingabe jeweils eine verrauschte Version dieses Bildes ist. Die verrauschten Bilder können erzeugt werden, indem ein zufälliges Rauschen über jedes Pixel hinzugefügt wird.

Der Autoencoder hat die gleiche Eingabedimension wie Ausgabedimension, welche $32 \times 32 \times 3$ Pixeln entspricht. In einer verdeckten Schicht ist die Dimension jedoch niedriger, welche dafür sorgt, dass das Modell in dieser nur einen Teil der Bildinformationen zwischenspeichern kann. Zum Verkleinern der Dimensionen werden Faltungsschichten und Pooling-Operationen genutzt. Das Vergrößern der Dimension wird mit transponierten Faltungsschichten ausgeführt. Transponierte Faltungsschichten funktionieren simultan zu normalen Faltungsschichten, jedoch wird zuvor das Bild um interpolierte Pixel erweitert. Abbildung 5.7 zeigt, welchen Effekt der Autoencoder auf verrauschte Bilder ausübt. Der linke Teil der Abbildung zeigt ein originales Bild aus der CIFAR-10 Datenmenge. Dieses ist der Klasse "Schiff" untergeordnet. Der mittlere Teil der Abbildung zeigt dabei die zufällig verrauschte Version dieses Bildes. Das Schiff ist dabei kaum zu erkennen, da viele Pixel des Bildes eine andere Farbe angenommen haben. Der rechte Teil der Abbildung zeigt, wie die Ausgabe des Autoencoders bei Eingabe des verrauschten Bildes aussieht. Das Schiff ist wieder erkennbar und das Bild ähnelt dem originalen Bild.

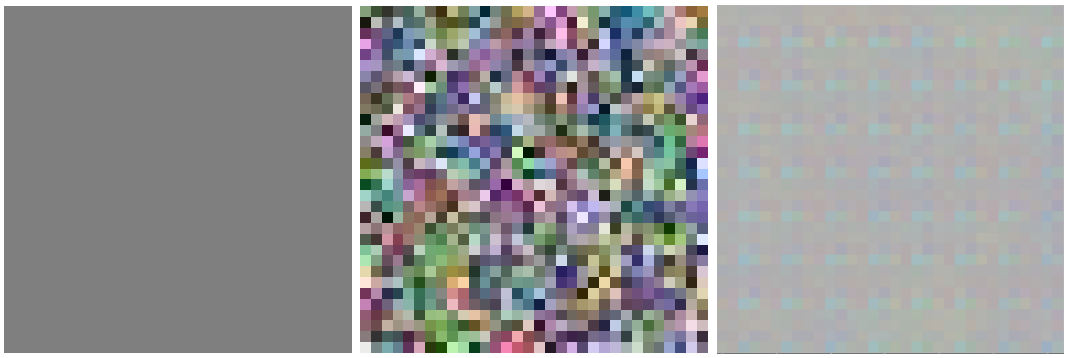
(a) Originales Bild (b) Verrauschtes Bild (c) Ausgabe Autoencoder

Abbildung 5.7: Autoencoder CIFAR-10 Datenbestand

Wird die Model Inversion Attacke gegen das CIFAR-10 Modell ohne DPSGD ausgeführt, kann kein Bild rekonstruiert werden. Parameter wie der Optimizer, die Lernrate oder die Verlustfunktion wurden angepasst, jedoch führte dies zu keiner Verbesserung des Angriffs Zusätzlich wurden unterschiedliche Methoden der Initialisierung des Startbildes gewählt. Ebenfalls wurden evaluiert, den Angriff einige Iterationen ohne den Autoencoder durchzuführen. Keine der Anpassungen sorgte für eine Verbesserung des Angriffs.

Abbildung 5.8 zeigt, das Ergebnis einer Model Inversion Attacke, wenn ein Bild mit dem Label "Schiff" nachgebildet werden soll. Das linke Bild entspricht dem initialen Startbild, bei dem alle Farbwerte jeweils mit dem Wert 0,5 befüllt werden. Dies entspricht einem grauen Bild. Das mittlere Bild zeigt, wie sich das Bild nach einigen Iterationen des Angriffs ohne Autoencoder verändert hat. Dabei handelt es sich um zufällig aussehende Pixel, jedoch sagt das CIFAR-10 Modell bei diesem Bild das gewünschte Label "Schiff" vorher. Das rechte Bild zeigt das Ergebnis des Angriffs, nachdem das Bild für weitere Iterationen angepasst wurde und pro Iteration mit dem Autoencoder angepasst wurde. Das entstandene Bild

besteht primär aus gräulichen Pixeln, welche eine Musterung aufweisen, die vermutlich vom Autoencoder stammt. Es ist kein Schiff zu erkennen.

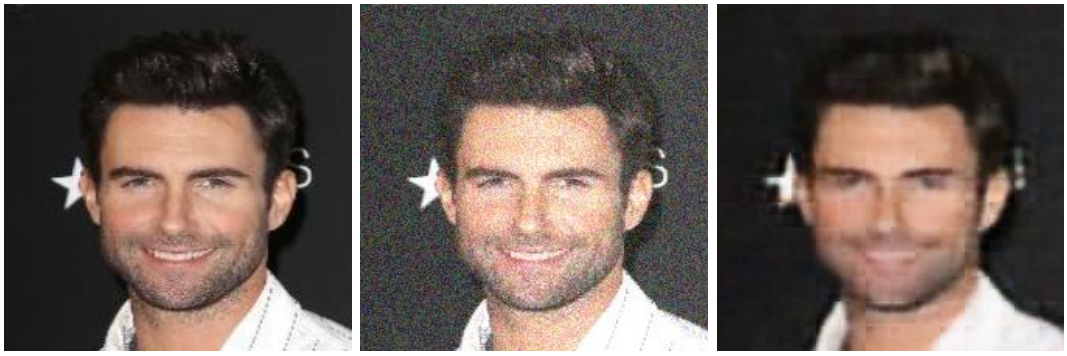


(a) Initiales Bild jeder Wert=0,5 (b) Anpassung ohne Autoencoder (c) Anpassung mit Autoencoder

Abbildung 5.8: Model Inversion Attacke gegen CIFAR-10

Model Inversion Attacke gegen ResNet-18 Modell

Die Model Inversion Attacke gegen das ResNet-18 Modell erfolgt simultan zu der Attacke gegen das CIFAR-10 Modell. Zuerst wird ein Autoencoder trainiert, welcher das Rauschen von Bildern der CelebA Datenmenge entfernt. Abbildung 5.9 zeigt, wie ein verrauschtes Bild mittels des Autoencoders wieder entrauscht werden kann. Der Output des Autoencoders, also das entrauschte Bild, zeigt das Gesicht einer Person, jedoch ist das Bild weniger scharf als das originale Bild.

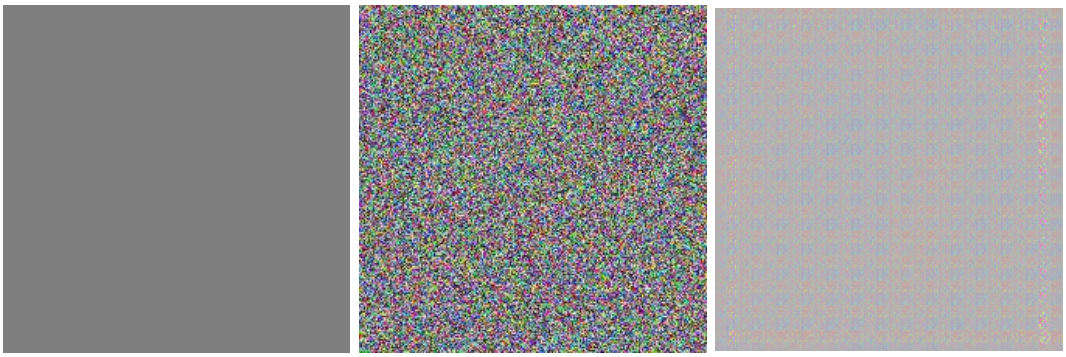


(a) Originales Bild (b) Verrauschtes Bild (c) Ausgabe Autoencoder

Abbildung 5.9: Autoencoder CelebA Datenbestand

Die Model Inversion Attacke ist ebenfalls gegen das ResNet-18 Modell ohne DPSGD nicht effektiv. Abbildung 5.10 zeigt dabei das Ergebnis einer Rekonstruktion. Die Labels, nach welchen das Bild rekonstruiert werden soll, entstammen dem Beispielbild aus Abbildung 5.9. Als Startbild dient hier wieder ein Bild, bei welchem jeder Farbwert auf den Wert 0,5 gesetzt

ist. Dieses wurde für einige Iterationen durch die Backpropagation des Modells angepasst, jedoch vorerst ohne Autoencoder. Hier entsteht eine zufällige Ansammlung an Pixeln, welche jedoch vom Modell die gewünschten Labels vorhergesagt bekommt. Nach einigen weiteren Iterationen, sowie zusätzlichen Anpassungen mittels des Autoencoders, entsteht wieder ein relativ gräuliches Bild. Dieses besitzt ebenfalls eine Musterung, welche vermutlich durch den Autoencoder entstanden ist.



(a) Initiales Bild jeder Wert=0,5

(b) Anpassung ohne Autoencoder

(c) Anpassung mit Autoencoder

Abbildung 5.10: Model Inversion Attacke gegen ResNet-18 Modell

5.6 Zusammenfassung der Experimente

PyTorch stellt mit Opacus eine Bibliothek zur Verfügung, welche die Nutzung von DPSGD durch einige Wrapper-Klassen ermöglicht. Opacus bietet über die Klasse **PrivacyEngine** einen simplen Weg, die PyTorch Objekte entsprechend umzuwandeln. Anschließend erfolgt das Training mit normalem PyTorch Code, weshalb die Nutzung von DPSGD mit Opacus kein aufwendiges Refactoring benötigt. Da jedoch die Funktionsweise von PyTorch durch Opacus angepasst wird, steigt der Speicherverbrauch, sowie die Trainingsdauer einer Epoche (Kaptiel 5.3).

Durch die richtige Wahl von Hyperparametern kann die Güte von neuronalen Netzen mit DPSGD optimiert werden. Dazu sollte eine möglichst große Batch-Größe, ein geringer Wert der Clipping-Norm und eine verhältnismäßig hohe Anzahl an Trainingsepochen gewählt werden. Ein höherer ϵ-Wert sorgt ebenfalls für eine bessere Güte des Modells. Die Hyperparameter wurden anhand von drei Modellen evaluiert:

- **CIFAR-10 Modell**: Das CIFAR-10 Modell ist ein neuronales Netz, welches eine ResNet-Architektur mit 10 Schichten nutzt und den CIFAR-10 Datenbestand in 10 unterschiedliche Klassen einteilt. Bei der ResNet-Architektur handelt es sich um ein neuronales Netz mit Faltungsschichten, welches zusätzlich sogenannte Skip Connections nutzt.

- **ResNet-18 Modell**: Das ResNet-18 Modell ist ein neuronales Netz mit 18 Schichten, welches ebenfalls die ResNet-Architektur nutzt. Dieses Modell erkennt 40 Merkmale aus Bildern von menschlichen Gesichtern.
- **Vision Transformer Modell**: Das Vision Transformer Modell erkennt ebenfalls 40 Merkmale aus Bildern von menschlichen Gesichtern. Jedoch basiert dieses auf der Transformer-Architektur, welche Embeddings, Aufmerksamkeitsmechanismen und Encoder nutzt.

Die Differenz der Genauigkeit des CIFAR-10 Modells mit DPSGD und ohne DPSGD fällt dabei am größten aus. Bei einem ϵ-Wert von 10, sinkt die Genauigkeit von 82,9 % ohne DPSGD auf eine Genauigkeit von 59,4 %. Die Differenz fällt bei dem ResNet-18 Modell und bei dem Vision Transformer Modell geringer aus. Bei dem ResNet-18 Modell sinkt die Genauigkeit bei einem ϵ-Wert von 10 um 5,5 Prozentpunkte. Die Genauigkeit des Vision Transformer Modells sinkt bei einem ϵ-Wert von 10 nur um 2,7 Prozentpunkte (Kapitel 5.4).

Die Wahl von ϵ und der damit einhergehende Schutz ist abhängig von der benötigten Sicherheit. Jedoch wird gezeigt, dass die Membership Inference Attacke und die Model Inversion Attacke bei den genutzten Use Cases nicht effektiv ist. Sogar bei den Modellen ohne DPSGD sind die Angriffe unwirksam (Kapitel 5.5).

Kapitel 6

Abgeleitetes Handlungsmodell

Da Methoden zur Sicherung der Vertraulichkeit einen Mehraufwand verursachen und möglicherweise sogar die Qualität von Modellen verschlechtern, sollten diese Methoden nur eingesetzt werden, wenn diese sachgerecht erscheint. Abbildung 6.1 zeigt einen Entscheidungsbaum, welcher bei dieser Fragestellung unterstützt. Der Entscheidungsbaum wird aus Sicht eines Unternehmens beschrieben, welches Daten sammelt und damit eine Anwendung entwickelt und bereitstellt. Die einzelnen Kriterien sind dennoch auf Einzelpersonen, öffentliche Open-Source-Projekte oder andere Use Cases anwendbar. Zudem werden nur die in den vorherigen Kapiteln beschrieben Methoden im Entscheidungsbaum berücksichtigt. Grundlegende Sicherheitsmechanismen der Softwareentwicklung, wie z. B. Authentifizierung, sollten zusätzlich genutzt werden.

Um den benötigten Schutz von Daten zu ermitteln, sollten Daten in unterschiedliche Vertraulichkeitsstufen klassifiziert werden. Die Vertraulichkeitsstufen können dabei für jedes Unternehmen individuell sein. Laut dem Bundesamt für Sicherheit in der Informationstechnik werden Daten häufig wie folgt abgestuft [103]:

- **Offen**: Diese Daten sind öffentlich bekannt. Diese finden sich beispielsweise auf der öffentlichen Website eines Unternehmens.
- **Intern**: Interne Daten sind Daten, welche unter normalen Mitarbeitern des Unternehmens bekannt und einsehbar sind. Diese könnten beispielsweise in einem Intranet vorhanden sein. Die Daten und Informationen sollten möglichst im Unternehmen bleiben, jedoch würde kein signifikanter Schaden entstehen, wenn diese öffentlich bekannt werden.
- **Vertraulich**: Daten, die vertraulich klassifiziert sind, stehen nur einer definierten Personenmenge zur Verfügung. Eine Veröffentlichung dieser würde dem Unternehmen schaden. Dabei kann der öffentliche Ruf des Unternehmens geschädigt werden, aber auch finanzielle und rechtliche Folgen sind möglich.
- **Streng vertraulich**: Streng vertrauliche Daten stehen ebenfalls nur einer definierten Personenmenge zur Verfügung, wobei die Größe der Personenmenge möglichst gering gehalten wird. Werden diese Daten öffentlich, entsteht ein großer Schaden. Dieser hat

in der Regel sowohl rechtliche als auch finanzielle Folgen und schädigt das Unternehmen nachhaltig.

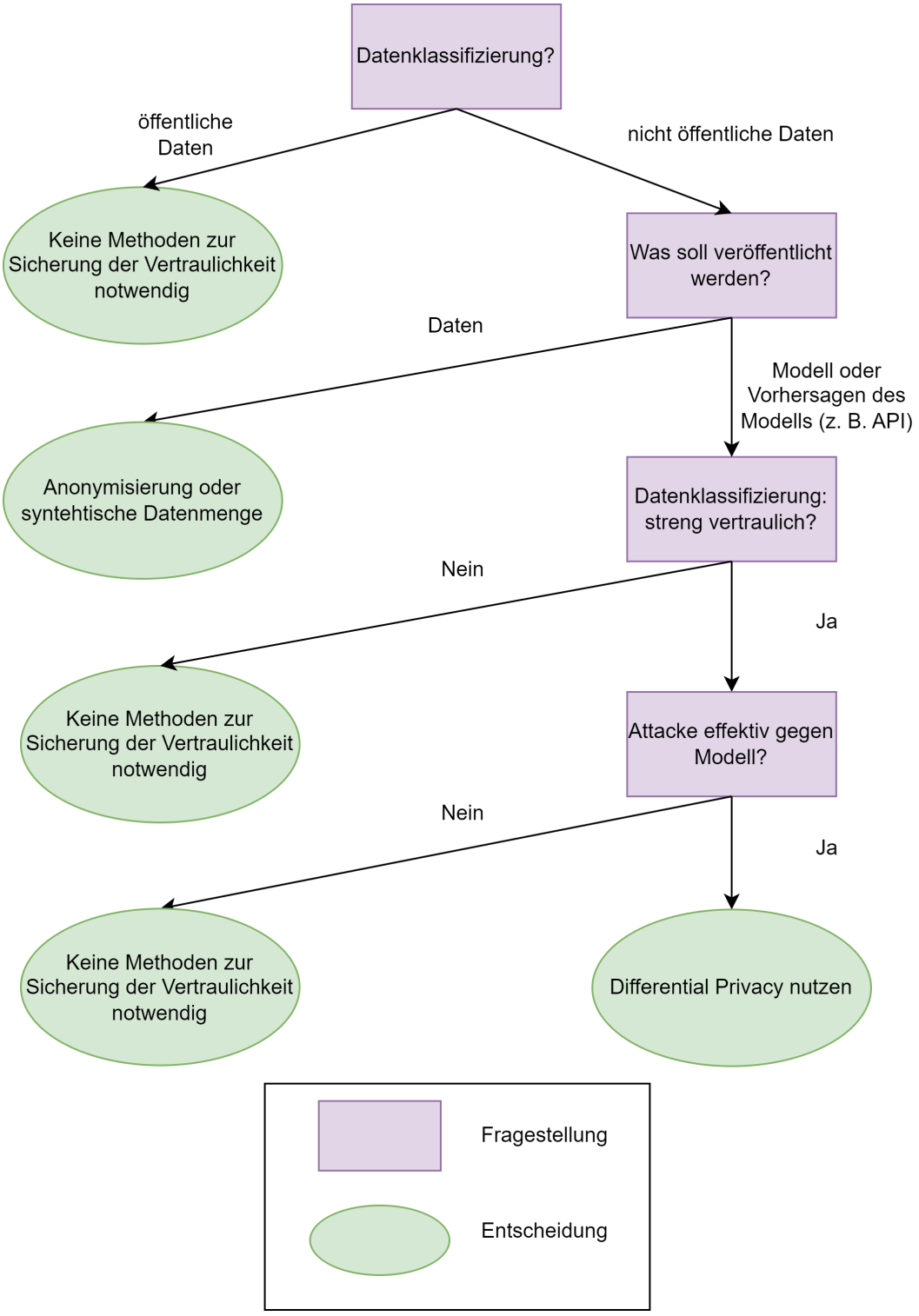


Abbildung 6.1: Abgleitetes Handlungsmodell

Wird eine Anwendung auf Basis öffentlicher Daten entwickelt, ist es nicht notwendig, Methoden zur Sicherung der Vertraulichkeit zu nutzen. Sind die Daten jedoch nicht öffentlich, hängt die Entscheidung von weiteren Faktoren ab. Sollen die Daten selbst veröffentlicht werden, dann muss jede Art von nicht-öffentlichen Daten anonymisiert werden. Bei vertraulichen oder streng vertraulichen Daten, wie beispielsweise Kundendaten, gibt es die Möglichkeit synthetische Datenmengen zu erzeugen und nur diese zu veröffentlichen.

Werden nicht die Daten veröffentlicht, sondern nur das Modell oder die Vorhersagen des Modells, dann benötigen interne und vertrauliche Informationen ebenfalls keine Methoden zur Sicherung der Vertraulichkeit. Dies liegt daran, dass die Angriffe gegen neuronale Netze nicht sehr effektiv sind. Experimente in Kapitel 5.5 zeigen dies und zusätzlich wird in Kapitel 7.1 die Effektivität von Angriffen diskutiert.

Da die Nutzung von streng vertrauliche Daten für ein enormes Risiko sorgt, sollten Modelle, die diese Daten nutzen, experimentell angegriffen werden. Dies gleicht einem Penetrationstest, wobei nicht nur Schwachstellen im Anwendungscode und der Infrastruktur evaluiert werden, sondern zusätzlich noch das Modell selbst. Die Evaluation der Angriffe kann beispielsweise automatisiert in die Deployment-Pipeline integriert werden. Wird das Modell veröffentlicht, sollte mindestens ein White-Box Angriff evaluiert werden. Wird das Modell lediglich über eine API bereitgestellt, reicht es, einen Black-Box Angriff zu evaluieren. Mögliche Angriffe sind hier die Membership Inference Attacke als Black-Box Angriff und die Model Inversion Attacke als White-Box Angriff.

Bei der Effektivität der Membership Inference Attacke kann ein Schwellwert von beispielsweise 80 % als Ziel gesetzt werden. Diesen Wert gilt es zu unterschreiten. Sollte ein Modell weniger anfällig für diesen Angriff sein, bedarf es keiner weiteren Methoden. Liegt die Effektivität des Angriffs jedoch über dem Schwellwert von 80 %, sollte Differential Privacy genutzt werden. Hier empfiehlt sich die Nutzung von DPSGD oder das Verrauschen der Ausgabe. Die Hyperparameter von DPSGD können anhand von Kapitel 5.4 optimiert werden, wobei zunächst ein hoher ϵ-Wert von 50 gewählt wird. Liegt die Effektivität des Angriffs anschließend immer noch über dem Schwellwert, sollte der ϵ-Wert sukzessiv verkleinert werden.

Um eine Model Inversion Attacke zu evaluieren, benötigt es eine Möglichkeit, zu evaluieren, ob ein rekonstruierter Datensatz einem originalen Datensatz ähnelt. Dies könnte über mathematische Metriken, z. B. mittels der Kosinus-Ähnlichkeit zweier Vektoren, bewertet werden, ohne von menschlichen Fachexperten beurteilt werden. Mathematische Metriken bieten den Vorteil, dass hier ein Schwellwert festgelegt werden kann, wohingegen dies bei einer menschlichen Beurteilung nicht möglich ist. Ist die Effektivität des Angriffs hoch, sollte hier ebenfalls Differential Privacy genutzt werden.

Kapitel 7

Diskussion und Ausblick

Ziel dieser Arbeit ist es, den Schutz der Vertraulichkeit in neuronalen Netzen zu gewährleisten, obwohl sensible Daten genutzt werden. Die Vertraulichkeit kann dabei von einer ganzen Reihe an Angriffen bedroht werden. Kapitel 5.5 zeigt jedoch, dass weder die Membership Inference Attacke, noch die Model Inversion Attacke effektiv gegen exemplarische Use Cases sind. Im Folgenden wird deshalb diskutiert, welche Gefahr durch diese Angriffe wirklich entstehen kann.

Außerdem sind neuronale Netze ein aktiver Forschungsbereich. Neue Modelle mit neuen Architekturen werden regelmäßig veröffentlicht. Dies kann dafür sorgen, dass sich auch die Angriffe weiterentwickeln. Dies wird ebenfalls im Folgenden betrachtet.

7.1 Effektivität der Angriffe

Die Diskussion zur Effektivität der Angriffe basiert auf der Membership Inference Attacke und auf der Model Inversion Attacke, da beide Angriffe in dieser Arbeit bereits evaluiert werden. Shokri et. al. [22], die Forscher, welche die Membership Inference Attacke erstmals vorstellten, zeigen, dass der Angriff effektiv sei, wenn das Modell viele unterschiedliche Klassen vorhersagt und zusätzlich auf die Trainingsdaten overfitted ist. Overfitting bedeutet, dass ein Modell sehr stark an die Trainingsdaten angepasst ist und deshalb nicht mehr gut generalisiert. Overfitting zeigt sich, wenn die Genauigkeit auf Trainingsdaten sehr hoch ist, jedoch die Genauigkeit für ungenutzte Daten, wie eine Testdatenmenge, signifikant niedriger ist. Tabelle 7.1 zeigt die Effektivität der Membership Inference Attacke, welche von Shokri et. al. [22] evaluiert wurde. Die konkreten Use Cases sind dabei zweitrangig. Viel wichtiger ist jedoch die Anzahl der Klassen und das Verhältnis zwischen der Genauigkeit gemessen anhand der Trainingsdaten und der Genauigkeit gemessen anhand ungesehener Testdaten.

Der Use Case mit der Datenmenge "Einkommen" besitzt zwei unterschiedliche Klassen und die Genauigkeit anhand der Trainingsdaten ist vergleichbar mit der Genauigkeit der Testdaten. Es werden also wenige Klasse vorhergesagt und das Modell ist nicht von Overfitting

Datenmenge	Anzahl an Klassen	Genauigkeit Trainingsdaten	Genauigkeit Testdaten	Effektivität Membership Inference Angriff
Einkommen	2	84,8 %	84,2 %	50,3 %
MNIST	10	98,4 %	92,8 %	51,7 %
Standort	30	100,0 %	67,3 %	67,8 %
Einkauf	2	99,9 %	98,4 %	50,5 %
Einkauf	10	99,9 %	86,6 %	55,0 %
Einkauf	20	100,0 %	78,1 %	59,0 %
Einkauf	50	100,0 %	69,3 %	86,0 %
Einkauf	100	99,9 %	65,9 %	93,5 %

Tabelle 7.1: Effektivität Membership Inference Attacke nach [22]

betroffen. Somit ist die Membership Inference Attacke nicht effektiv, was sich an einer Effektivität von 50,3 % zeigt. Ähnlich sieht es bei dem Use Case mit der Datenmenge "MNIST" aus. Hier werden 10 Klasse vorhergesagt, wobei die Genauigkeit auf den Testdaten etwas geringer ist, als die Genauigkeit der Trainingsdaten. Jedoch ist die Membership Inference Attacke mit 51,7 % ebenfalls nicht effektiv. Der Use Case mit der Datenmenge "Standort" unterscheidet zwischen 30 verschiedenen Klassen. Zusätzlich ist das Modell von Overfitting betroffen, was daran erkennbar ist, dass die Genauigkeit anhand der Trainingsdaten bei vollen 100 % liegt, die Genauigkeit auf Testdaten lediglich bei 67,3 %. Dies sorgt für eine höhere Effektivität der Membership Inference Attacke von 67,8 %.

Shokri et. al. [22] evaluieren zusätzlich die Membership Inference Attacke mit der Datenmenge "Einkauf", bei welcher die Anzahl an vorhergesagten Klassen bei jedem Modell erhöht wird. Jedes der Modelle ist von Overfitting betroffen, was an der Genauigkeit erkennbar ist. Diese liegt bei den Trainingsdaten bei 99,9 % oder 100,0 %, wohingegen diese bei den Testdaten geringer ist und bis auf 65,9 % abfällt. Die Effektivität des Membership Inference Attacke bei dem Modell, welches 20 Klassen vorhersagt, liegt bei 59,0 % und ist damit immer noch ineffektiv. Jedoch sind die Modelle, welche 50 und 100 Klassen vorhersagen, von der Membership Inference Attacke bedroht. Bei diesen liegt die Effektivität des Angriffs bei 86,0 % und 93,5 %.

Ein weiterer Faktor dieses Angriffs ist es, welche Ressourcen benötigt werden, um diesen Angriff durchzuführen. Die benötigten Ressourcen hängen primär von der Anzahl der Shadow Modelle ab. Da jedes Shadow Modell das originale Modell nachahmen soll, ist das Training eines Shadow Modells ressourcentechnisch vergleichbar mit dem des originalen Modells. Shokri et. al. [22] nutzen zwischen 10 und 100 Shadow Modellen, was eine Membership Inference Attacke demnach 10 bis 100 Mal so ressourcenintensiv macht, wie das Training des originalen Modells.

Das Handlungsmodell aus Kapitel 6 empfiehlt, die Membership Inference Attacke nur bei streng vertraulichen Daten zu evaluieren. Dies liegt an den benötigten Ressourcen, um die Evaluierung durchzuführen. Der Aufwand, den Angriff bei Daten mit einer niedrigen Vertraulichkeitsklassifizierung zu evaluieren, ist zu hoch. Dies liegt außerdem daran, dass Overfitting bei jedem Modell nach Möglichkeit vermieden werden soll, was die Membership Inference Attacke ineffektiver macht.

Die Model Inversion Attacke, beziehungsweise die Reconstruction Attacke wurde von Fredrikson et. al. [15] vorgestellt. Kapitel 2.3 beschreibt, wie das Bild einer Person rekonstruiert werden kann. Zur Evaluierung nutzten Fredrikson et. al. [15] einen Datenbestand, welcher jeweils 10 Bilder von 40 unterschiedlichen Personen zeigt [104]. Demnach hat jede klassifizierbare Person lediglich 10 Bilder, wobei diese Bilder ähnlich aussehen. Abbildung 2.1 zeigt, wie das Bild einer Person rekonstruiert wurde. Jedoch könnte das rekonstruierte Bild auch ein anderes Bild dieser Person zeigen, da die Trainingsbilder der gleichen Person ähnlich sind. Ein Indiz für diese Vermutung ist der Mund des rekonstruierten Bildes. Im rekonstruierten Bild ist der Mund geöffnet, wohingegen in dem dargestellten originalen Bild der Mund geschlossen ist. Abbildung 7.1 zeigt das rekonstruierte Bild von Fredrikson et. al. [15] mit mehreren Datensätzen der Trainingsdatenmenge.

Abbildung 7.1: Mögliche rekonstruierte Datensätze einer Model Inversion Attacke [15] [104]

Das neuronale Netz, welches die Klassifikation vornimmt, besteht dabei aus drei vollständig verbunden Schichten, wobei die erste Schicht die Eingabeschicht ist und die letzte Schicht die Ausgabeschicht. Somit gibt es nur eine Hidden Layer. Es werden auch keine Faltungsschichten oder andere komplexere Techniken genutzt. Die Ähnlichkeit und die geringe Anzahl von Trainingsbildern, sowie eine simple Modellarchitektur sind demnach zwei mögliche Erklärungsmöglichkeiten für die Effektivität der Model Inversion Attacke.

Die Modell Inversion Attacke in Kapitel 5.5, evaluierte den Angriff anhand des CIFAR-10 Datenbestand und des CelebA Datenbestands. Jede Klasse des CIFAR-10 Datenbestands hat 6000 unterschiedliche Bilder [82], wobei die Bilder selbst mehr Unterschiede aufweisen, als der von Fredrikson et. al. genutzte Datenbestand. Dies liegt daran, dass die Bilder farbig sind und die Objekte in unterschiedlichen Situationen darstellen. Der CelebA Datenbestand zeigt ebenfalls nur Gesichter an, jedoch gibt es für jedes Merkmal mindestens 4547 Beispiele im Trainingsdatenbestand [83]. Sowohl das Modell für den CIFAR-10 Datenbestand, als

auch die beiden Modelle für den CelebA Datenbestand waren deutlich komplexer als das Modell von Fredrikson et. al. [15].

Die Erweiterung der Model Inversion Attacke von Zhang et. al. [16] verbessert den Autoencoder und zeigt den Angriff, wenn ein Angreifer bereits einen Teil des Datensatzes, z. B. einen Bildausschnitt, besitzt. Zhang et. al. [16] evaluierten den Angriff auf dem CelebA Datenbestand, welcher auch in Kapitel 5.5 genutzt wird. Den Autoren gelingt es, fehlende Bildausschnitte eines Datensatzes mit dem Angriff zu befüllen. Dabei werden auch komplexere neuronale Netze genutzt.

Da es sich bei der Model Inversion Attacke um einen White-Box Angriff handelt, empfiehlt das Handlungsmodell aus Kapitel 6 diesen Angriff nur zu evaluieren, wenn das ganze Modell veröffentlicht wird. Um den Angriff bestmöglich zu evaluieren, sollte auch Fälle getestet werden, bei denen der Angreifer über Hintergrundwissen oder Teile von Datensätzen verfügt.

7.2 Weiterentwicklung von Angriffen durch moderne Modelle

ChatGPT ist ein Large Language Modell, welches eine Vielzahl von Aufgaben übernehmen kann. Die Generierung von Bildern gehört bislang nicht dazu. Jedoch hat OpenAI, das Unternehmen hinter ChatGPT, ein anderes Modell, welches dies kann. Dieses heißt DALL-E. DALL-E 2 ist seit 2022 verfügbar und die neuste Version, DALL-3, wird planmäßig im Oktober 2023 veröffentlicht[1]. DALL-E 3 wird voraussichtlich in die ChatGPT Anwendung integriert werden. Stable Diffusion, bzw. Stable Diffusion XL[2], ist eine Open-Source Alternative zu DALL-E.

Sowohl DALL-E als auch Stable Diffusion erzeugen aus einer Texteingabe ein hochauflösendes Bild. Die Texteingabe kann dabei den Inhalt des Bildes, als auch den Stil von diesem beschreiben. Um das Bild zu erzeugen, wird die Texteingabe zuerst in einen latenten Vektorraum konvertiert [105]. Diese Konvertierung erfolgt mittels eines Sprachmodells. Dieser Vektor wird anschließend mittels eines neuronalen Netzes zu einem Bild hochskaliert. Die Hochskalierung entspricht dabei dem Decoder eines Autoencoders [105]. Stable Diffusion wurde auf dem LAION-5B Datenbestand trainiert, welcher 5,85 Milliarden Bilder mit einer zugehörigen Beschreibung enthält [106]. Sowohl DALL-E als auch Stable Diffusion können qualitativ hochwertige und hochauflösende Bilder erzeugen. Abbildung 7.2 zeigt vier Bilder, welche von Stable Diffusion XL generiert sind[3]. Die zugehörige Texteingabe lautet: "*Panda bear eating a bowl of japanese ramen in an anime style*".

[1] https://openai.com/dall-e-3
[2] https://stability.ai/blog/stable-diffusion-sdxl-1-announcement
[3] https://clipdrop.co/stable-diffusion

Abbildung 7.2: Stable Diffusion XL Beispiele

Diese Modelle zeigen, wozu moderne generative Modelle in der Lage sind. Diverse Angriffe, wie beispielsweise die Model Inversion Attacke, nutzen ebenfalls generative Modelle. Dies ermöglicht verbesserte Versionen des Angriffs. Beispielsweise könnte die Hochskalierung einer Model Inversion Attacke nicht mehr von einem eigens trainierten Autoencoder übernommen werden, sondern von einem riesigen, öffentlichen Modell wie Stable Diffusion. Zusätzlich könnten auch neue Arten von Angriffen entwickelt werden. Eine denkbare Alternative einer Model Inversion Attacke ist es, mittels Stable Diffusion Bilder zu erzeugen, welche vom anzugreifenden Modell klassifiziert werden. Die Vorhersagewahrscheinlichkeiten des anzugreifenden Modells können anschließend genutzt werden, um die Eingabetexte von Stable Diffusion iterativ anzupassen. Dies wird so lange wiederholt, bis das Modell die gewünschte Klasse vorhersagt. Dabei handelt es sich um einen hypothetischen Angriff, welcher aktuell noch nicht erforscht ist.

Generell besteht das Risiko, dass Fortschritte in der Forschung von neuronalen Netzen dafür sorgen, dass neue Angriffe entstehen und bestehende Angriffe optimiert werden. Wird beispielsweise eine automatische Evaluierung von Angriffen in einer Deployment-Pipeline eines Modells vorgenommen, sollte darauf geachtet werden, dass aktuelle Angriffe mit fortschrittlichen Angreifermodellen genutzt werden.

Zusätzlich gibt es die Möglichkeit Penetrationstests von Anwendungen und Infrastruktur auszuweiten oder anzupassen, sodass auch Machine Learning Modelle evaluiert werden. Dabei werden Angriffe gegen die Vertraulichkeit bei neuronalen Netzen evaluiert, jedoch zusätzlich auch andere Angriffe, welche z. B. das Ziel haben, die Güte des Modells zu verschlechtern. Diese Art von Penetrationstests ist aktuell nicht weit verbreitet, allerdings steigt die Relevanz eines solchen Tests für neuronale Netze.

Kapitel 8

Zusammenfassung

Neuronale Netze benötigen für das Training eine große Menge an Daten. Ja nach Use Case, handelt es sich dabei um sensible Daten, welche geschützt werden müssen. Jedoch gibt es eine Vielzahl an Angriffen, welche zeigen, dass ein relevantes Risiko besteht, dass die Vertraulichkeit von Daten bei der Anwendung von neuronalen Netzen gefährdet ist.

Angriffe können dabei in White-Box und Black-Box Attacken eingeteilt werden. Diese unterscheiden sich dadurch, ob ein Angreifer Zugriff auf das gesamte Modell hat (White-Box) oder nur die Vorhersagen von diesem kennt (Black-Box). Die Angriffe ermöglichen es, Eigenschaften der Trainingsdaten zu ermitteln, einzelne Trainingsdatensätze zu rekonstruieren oder festzustellen, ob ein spezieller Datensatz im Training eines Modells genutzt wurde. Die Effektivität dieser Angriffe variiert dabei, je nachdem, welcher Trainingsdatenbestand und welche Modellarchitektur genutzt werden. Dies kann sogar so weit reichen, dass Angriffe unwirksam sind, selbst wenn ein Modell keine speziellen Techniken nutzt, um die Vertraulichkeit zu sichern. Da sich jedoch nicht nur neuronale Netze weiterentwickeln, sondern auch die Angriffe gegen diese, empfiehlt es sich, diese Angriffe dennoch zu evaluieren. Dies ist besonders wichtig, wenn die genutzten Daten als "*streng vertraulich*" klassifiziert werden.

Zur Sicherung gegen Angriffe gibt es eine Reihe an Maßnahmen, welche sich in kryptografische und statistische Methoden einteilen lassen. Kryptografische Methoden können Berechnungen auf verschlüsselten Daten ausführen, wodurch das Modelltraining und die Inferenz verschlüsselt möglich sind. Jedoch erzeugen diese Methoden einen signifikanten Mehraufwand, welche Berechnungen um ein Vielfaches verlangsamt. Die meisten Methoden fokussieren deshalb die Inferenz eines Modells, nicht das Training. Dennoch ist der Mehraufwand alleine bei der Inferenz so hoch, dass eine Vorhersage hunderte bis tausende Male länger benötigt, wie eine Vorhersage ohne die Nutzung von Kryptografie. Zusätzlich steigt dieser Faktor mit der Komplexität und der Anzahl an Parametern eines Modells, wodurch große Modelle wie ChatGPT nicht praktikabel betrieben werden könnte.

Statistische Methoden haben weniger Mehraufwand, weshalb diese Methoden derweil bevorzugt werden. Differential Privacy ist hier die populärste und wichtigste Technik. Diese Technik ermöglicht es, den Einfluss eines einzelnen Datensatzes auf Berechnungen mit dem

ganzen Datenbestand quantitativ zu begrenzen. Dies geschieht in der Regel durch ein Verrauschen des ursprünglichen Ergebnisses. Differential Privacy ermöglicht es demnach, den Einfluss eines einzelnen Datensatzes auf ein neuronales Netz zu begrenzen, weshalb Angriffe abgeschwächt werden können. Der Einfluss eines Datensatzes wird durch das Privacy Budget, welches aus den Parametern ϵ und δ besteht, quantifiziert. Die gängigste Variante, um Differential Privacy in Kombination mit neuronalen Netzen zu nutzen, ist eine Technik namens DPSGD. Dabei werden die Gradienten der Gewichte eines neuronalen Netzes während des Trainings verrauscht.

Um die Vertraulichkeit in einem neuronalen Netz zu schützen, sollte evaluiert werden, wie effektiv Angriffe gegen dieses Modell sind. Sind diese Angriffe bereits ineffektiv, so bedarf es keiner zusätzlichen Techniken. Sind die Angriffe jedoch sehr effektiv, so sollte Differential Privacy in Form von DPSGD genutzt werden. Das Privacy Budget wird dabei zunächst hoch gewählt, sodass die Nützlichkeit des Modells bestmöglich erhalten bleibt. Anschließend werden die Angriffe erneut evaluiert. Der Wert des Privacy Budgets wird dabei iterativ gesenkt, solange bis die Angriffe ineffektiv sind. Dies sorgt für ein optimales Gleichgewicht zwischen der Nützlichkeit des Modells und der Sicherheit der Vertraulichkeit.

Abbildungsverzeichnis

2.1 Rekonstruktion eines Bildes [15] . . . 7
2.2 Permutation eines neuronalen Netzes [19] . . . 10
2.3 Verteiltes Lernen Topologien [32] . . . 15
2.4 Rekonstruktion durch Deep Leakage [34] . . . 17

3.1 Training eines neuronalen Netzes . . . 22
3.2 Einfluss von ϵ . . . 30
3.3 Beispiel Differential Privacy . . . 31
3.4 Generative Adversarial Network nach [17] . . . 37
3.5 Synthetische MNIST Datenmenge mittels DPGAN [46] . . . 38
3.6 PATE-Architektur nach [51] . . . 42
3.7 PATE-G . . . 44
3.8 Gruppenhomomorphismus nach [53] . . . 45
3.9 Homomorphe Verschlüsselung . . . 46

5.1 Neuronales Netz zur Klassifikation von CIFAR-10 . . . 74
5.2 Auswirkung Batch-Größe auf CIFAR-10 Modell ohne DPSGD und mit Datenaugmentierung . . . 85
5.3 Auswirkung der Batch-Größe auf die Genauigkeit von CIFAR-10 Modellen . . . 87
5.4 Auswirkung von AutoAugment auf die Genauigkeit von CIFAR-10 Modellen . . . 87
5.5 Auswirkung der Clipping-Norm auf die Güte von CIFAR-10 Modellen bei 15 Epochen Training . . . 88
5.6 Auswirkung der Clipping-Norm auf die Güte von CIFAR-10 Modellen bei 30 Epochen Training . . . 89
5.7 Autoencoder CIFAR-10 Datenbestand . . . 98
5.8 Model Inversion Attacke gegen CIFAR-10 . . . 99
5.9 Autoencoder CelebA Datenbestand . . . 99
5.10 Model Inversion Attacke gegen ResNet-18 Modell . . . 100

6.1 Abgleitetes Handlungsmodell . . . 104

7.1 Mögliche rekonstruierte Datensätze einer Model Inversion Attacke [15] [104] . . . 109
7.2 Stable Diffusion XL Beispiele . . . 111

Tabellenverzeichnis

3.1 Nicht-Anonymisierter Titanic Datensatz [38] . . . 24
3.2 k-Anonymity Titanic Datensatz [37][38] . . . 25
3.3 Angreifbare Abwandlung von Tabelle 3.2 . . . 25
3.4 Beispiel Marginalverteilungen . . . 36

4.1 Übersicht kryptografischer Methoden . . . 64
4.2 Übersicht Differential Privacy Methoden . . . 68
4.3 Komplexität Differential Privacy Methoden . . . 69

5.1 Anzahl Datensätze . . . 75
5.2 Anzahl Modelle je Framework auf Hugging Face [95] . . . 76
5.3 Speicherverbrauch mit und ohne Differential Privacy . . . 81
5.4 Dauer des Trainings einer Epoche mit und ohne Differential Privacy . . . 82
5.5 Genauigkeit von CIFAR-10 Modellen mit DPSGD . . . 90
5.6 Übersicht der besten ResNet-18 Modelle mit DPSGD . . . 91
5.7 Vergleich der Hyperparameter bei Vision Transformer Modellen . . . 92
5.8 Übersicht der besten Vision Transformer Modelle mit DPSGD . . . 92
5.9 Membership Inference Angriff gegen CIFAR-10 Modell ohne DPSGD . . . 95
5.10 Membership Inference Angriff gegen CIFAR-10 Modelle mit DPSGD . . . 96
5.11 Membership Inference Angriff gegen ResNet-18 Modelle mit DPSGD . . . 97

7.1 Effektivität Membership Inference Attacke nach [22] . . . 108

A.1 Übersicht der Hardware . . . 131
A.2 Übersicht der Software . . . 131
A.3 Übersicht der wichtigsten Python Bibliotheken . . . 131

B.1 CIFAR-10 Modell mit AutoAugment ohne Differential Privacy . . . 133
B.2 CIFAR-10 Modell ohne Augmentierung ohne Differential Privacy . . . 133
B.3 CIFAR-10 Modell, DPSGD mit Clipping bei 1,0 und mit AutoAugments . . . 134
B.4 CIFAR-10 Modell, DPSGD mit Clipping bei 1,0 und ohne Augmentation . . . 135
B.5 CIFAR-10 Modell, DPSGD mit Clipping bei 0,0001 und ohne Augmentation . . . 136
B.6 Resnet-18 Modell mit AutoAugment ohne Differential Privacy . . . 137
B.7 Resnet-18 Modell ohne Augmentation ohne Differential Privacy . . . 137
B.8 Resnet-18-Modell mit Augmentation, Batch-Größe=64 und Clipping bei 1,0 . . . 138

B.9 Resnet-18-Modell mit Augmentation, Batch-Größe=256 und Clipping bei 10^{-5} . 138
B.10 Resnet-18-Modell mit und ohne Augmentation 139
B.11 Vision Transformer Modell mit AutoAugment ohne Differential Privacy 140
B.12 Vision Transformer Modell ohne Augmentation ohne Differential Privacy 140
B.13 Vision Transformer Modell mit Augmentation, Batch-Größe=16 und Clipping bei 1,0 141
B.14 Vision Transformer Modell ohne Augmentation, virtuelle Batch-Größe=128 und Clipping bei 1,0 141
B.15 Vision Transformer Modell ohne Augmentation, virtuelle Batch-Größe=128 und Clipping bei 10^{-5} 142
B.16 Membership Inference Angriff gegen CIFAR-10 Modelle 143

Quellcodeverzeichnis

5.1 Training einer Epoche in PyTorch 77
5.2 Opacus Wrapper für DPSGD 78
5.3 Berechnung von Epsilon 78
5.4 Opacus Wrapper mit definiertem Epsilon 79
5.5 Opacus ModuleValidator 79
5.6 Opacus BatchMemoryManager 81

Literaturverzeichnis

[1] D. Milmo. „ChatGPT reaches 100 million users two months after launch.“ (Feb. 2023), Adresse: https://www.theguardian.com/technology/2023/feb/02/chatgpt-100-million-users-open-ai-fastest-growing-app (besucht am 23. 09. 2023).

[2] L. Ouyang, J. Wu, X. Jiang u. a., *Training language models to follow instructions with human feedback*, 2022.

[3] Twitter Inc., *Twitter's Recommendation Algorithm*, 22. Mai 2023.

[4] Siri Team. „Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant.“ (2017), Adresse: https://machinelearning.apple.com/research/hey-siri (besucht am 23. 09. 2023).

[5] comma.ai, *Openpilot*, Version 0.9.2, 24. Mai 2023.

[6] I. Caswell und B. Liang. „Recent Advances in Google Translate.“ (2020), Adresse: https://ai.googleblog.com/2020/06/recent-advances-in-google-translate.html (besucht am 23. 09. 2023).

[7] K. Tunyasuvunakool, J. Adler, Z. Wu u. a., „Highly accurate protein structure prediction for the human proteome,“ *Nature*, Jg. 596, Nr. 7873, S. 590–596, 2021.

[8] FirstMark. „The 2023 MAD (ML/AI/Data) Landscape.“ (2023), Adresse: https://mad.firstmark.com/ (besucht am 23. 09. 2023).

[9] C. Cadwalladr und E. Graham-Harrison, *Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach*, März 2018.

[10] J. Corbet. „Class action against GitHub Copilot.“ (Nov. 2022), Adresse: https://lwn.net/Articles/914150/ (besucht am 23. 09. 2023).

[11] Bundesamt für Sicherheit in der Informationstechnik, „IT-Grundschutz-Kompendium,“ in Bundesamt für Sicherheit in der Informationstechnik, Bonn, 2023, S. 42.

[12] A. Pakmehr, A. Aßmuth, C. P. Neumann und G. Pirkl, „Security Challenges for Cloud or Fog Computing-Based AI Applications,“ in *IARIA Cloud Computing 2023: 14th International Conference on Cloud Computing, GRIDs, and Virtualization*, Nice, France, Juni 2023, S. 21–29.

[13] J. Bennett und S. Lanning, „The netflix prize,“ in *Proceedings of KDD cup and workshop*, New York, 2007, S. 35.

[14] A. Narayanan und V. Shmatikov, „Robust De-anonymization of Large Sparse Datasets," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*, 2008, S. 111–125.

[15] M. Fredrikson, S. Jha und T. Ristenpart, „Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Ser. CCS '15, Denver, Colorado, USA: Association for Computing Machinery, 2015, S. 1322–1333.

[16] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li und D. Song, „The Secret Revealer: Generative Model-Inversion Attacks Against Deep Neural Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Juni 2020.

[17] I. Goodfellow, J. Pouget-Abadie, M. Mirza u. a., „Generative adversarial networks," *Communications of the ACM*, Jg. 63, Nr. 11, S. 139–144, 2020.

[18] G. Ateniese, L. V. Mancini, A. Spognardi, A. Villani, D. Vitali und G. Felici, „Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers," *International Journal of Security and Networks*, Jg. 10, Nr. 3, S. 137–150, 2015.

[19] K. Ganju, Q. Wang, W. Yang, C. A. Gunter und N. Borisov, „Property Inference Attacks on Fully Connected Neural Networks Using Permutation Invariant Representations," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, Ser. CCS '18, Toronto, Canada: Association for Computing Machinery, 2018, S. 619–633.

[20] Y. LeCun, C. Cortes und C. Burges, „MNIST handwritten digit database," *ATT Labs*, Jg. 2, 2010.

[21] D. Gopinath, H. Converse, C. Pasareanu und A. Taly, „Property Inference for Deep Neural Networks," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, S. 797–809.

[22] R. Shokri, M. Stronati, C. Song und V. Shmatikov, „Membership inference attacks against machine learning models," in *2017 IEEE symposium on security and privacy (SP)*, IEEE, 2017, S. 3–18.

[23] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis und F. Tramer, „Membership inference attacks from first principles," in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, S. 1897–1914.

[24] C. A. Choquette-Choo, F. Tramer, N. Carlini und N. Papernot, „Label-Only Membership Inference Attacks," in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila und T. Zhang, Hrsg., Ser. Proceedings of Machine Learning Research, Bd. 139, PMLR, Juni 2021, S. 1964–1974.

[25] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos und D. Song, „The secret sharer: Evaluating and testing unintended memorization in neural networks," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, S. 267–284.

[26] N. Carlini, F. Tramer, E. Wallace u. a., „Extracting Training Data from Large Language Models.," in *USENIX Security Symposium*, Bd. 6, 2021.

[27] OWASP Foundation, „OWASP Top 10 for LLM Applications," Publication, Version 1.0.1, Aug. 2023.

[28] B. Biggio, B. Nelson und P. Laskov, *Poisoning Attacks against Support Vector Machines*, 2013.

[29] C. Yang, Q. Wu, H. Li und Y. Chen, *Generative Poisoning Attack Method Against Neural Networks*, 2017.

[30] J. Guo und C. Liu, „Practical Poisoning Attacks on Neural Networks," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox und J.-M. Frahm, Hrsg., Cham: Springer International Publishing, 2020, S. 142–158.

[31] F. Tramèr, R. Shokri, A. S. Joaquin u. a., *Truth Serum: Poisoning Machine Learning Models to Reveal Their Secrets*, 2022.

[32] T. Li, A. K. Sahu, A. Talwalkar und V. Smith, „Federated learning: Challenges, methods, and future directions," *IEEE signal processing magazine*, Jg. 37, Nr. 3, S. 50–60, 2020.

[33] L. Melis, C. Song, E. De Cristofaro und V. Shmatikov, „Exploiting Unintended Feature Leakage in Collaborative Learning," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, S. 691–706.

[34] L. Zhu, Z. Liu und S. Han, „Deep Leakage from Gradients," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox und R. Garnett, Hrsg., Bd. 32, Curran Associates, Inc., 2019.

[35] B. Zhao, K. R. Mopuri und H. Bilen, „idlg: Improved deep leakage from gradients," *arXiv preprint arXiv:2001.02610*, 2020.

[36] B. Hitaj, G. Ateniese und F. Perez-Cruz, „Deep models under the GAN: information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, S. 603–618.

[37] L. Sweeney, „k-anonymity: A model for protecting privacy," *International journal of uncertainty, fuzziness and knowledge-based systems*, Jg. 10, Nr. 05, S. 557–570, 2002.

[38] W. C. Jessica Li, *Titanic - Machine Learning from Disaster*, 2012.

[39] A. Machanavajjhala, D. Kifer, J. Gehrke und M. Venkitasubramaniam, „L-Diversity: Privacy beyond k-Anonymity," *ACM Trans. Knowl. Discov. Data*, Jg. 1, Nr. 1, 3–es, März 2007.

[40] N. Li, T. Li und S. Venkatasubramanian, „t-Closeness: Privacy Beyond k-Anonymity and l-Diversity," in *2007 IEEE 23rd International Conference on Data Engineering*, 2007, S. 106–115.

[41] C. Dwork, „Differential Privacy," in *Automata, Languages and Programming*, M. Bugliesi, B. Preneel, V. Sassone und I. Wegener, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, S. 1–12.

[42] C. Dwork und A. Roth, „The Algorithmic Foundations of Differential Privacy," *Found. Trends Theor. Comput. Sci.*, Jg. 9, Nr. 3–4, S. 211–407, Aug. 2014.

[43] M. Abadi, A. Chu, I. Goodfellow u. a., „Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, S. 308–318.

[44] M. Hardt, K. Ligett und F. Mcsherry, „A Simple and Practical Algorithm for Differentially Private Data Release," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou und K. Weinberger, Hrsg., Bd. 25, Curran Associates, Inc., 2012.

[45] M. Arjovsky, S. Chintala und L. Bottou, „Wasserstein Generative Adversarial Networks," in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup und Y. W. Teh, Hrsg., Ser. Proceedings of Machine Learning Research, Bd. 70, PMLR, Aug. 2017, S. 214–223.

[46] L. Xie, K. Lin, S. Wang, F. Wang und J. Zhou, „Differentially private generative adversarial network," *arXiv preprint arXiv:1802.06739*, 2018.

[47] J. Yoon, J. Jordon und M. van der Schaar, „PATE-GAN: Generating Synthetic Data with Differential Privacy Guarantees," in *International Conference on Learning Representations*, 2019.

[48] R. McKenna, G. Miklau und D. Sheldon, „Winning the NIST Contest: A scalable and general approach to differentially private synthetic data," *arXiv preprint arXiv:2108.04978*, 2021.

[49] R. Mckenna, D. Sheldon und G. Miklau, „Graphical-model based estimation and inference for differential privacy," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri und R. Salakhutdinov, Hrsg., Ser. Proceedings of Machine Learning Research, Bd. 97, PMLR, Juni 2019, S. 4435–4444.

[50] R. Bassily, A. Smith und A. Thakurta, „Private Empirical Risk Minimization, Revisited," in *ICML 2014 Workshop on Learning, Security and Privacy*, Beijing, China, Juni 2014.

[51] N. Papernot, M. Abadi, Ú. Erlingsson, I. Goodfellow und K. Talwar, *Semi-supervised Knowledge Transfer for Deep Learning from Private Training Data*, 2017.

[52] B. Van der van der Waerden, E. Artin und E. Noether, *Moderne Algebra*, Ser. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg, 1937, §10.

[53] X. Yi, R. Paulet und E. Bertino, „Homomorphic Encryption," in *Homomorphic Encryption and Applications*. Cham: Springer International Publishing, 2014, S. 27–46.

[54] A. Acar, H. Aksu, A. S. Uluagac und M. Conti, „A Survey on Homomorphic Encryption Schemes: Theory and Implementation," *ACM Comput. Surv.*, Jg. 51, Nr. 4, Juli 2018.

[55] C. Gentry, „Fully Homomorphic Encryption Using Ideal Lattices," in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, Ser. STOC '09, Bethesda, MD, USA: Association for Computing Machinery, 2009, S. 169–178.

[56] M. van Dijk, C. Gentry, S. Halevi und V. Vaikuntanathan, „Fully Homomorphic Encryption over the Integers," in *Advances in Cryptology – EUROCRYPT 2010*, H. Gilbert, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 24–43.

[57] Z. Brakerski und V. Vaikuntanathan, „Efficient fully homomorphic encryption from (standard) LWE," *SIAM Journal on computing*, Jg. 43, Nr. 2, S. 831–871, 2014.

[58] C. Gentry, A. Sahai und B. Waters, „Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti und J. A. Garay, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, S. 75–92.

[59] B. Pulido-Gaytan, A. Tchernykh, J. M. Cortés-Mendoza u. a., „Privacy-preserving neural networks with Homomorphic encryption: Challenges and opportunities," *Peer-to-Peer Networking and Applications*, Jg. 14, Nr. 3, S. 1666–1691, Mai 2021.

[60] H. Takabi, E. Hesamifard und M. Ghasemi, „Privacy preserving multi-party machine learning with homomorphic encryption," in *29th Annual Conference on Neural Information Processing Systems (NIPS)*, 2016.

[61] D. Boneh, A. Sahai und B. Waters, „Functional Encryption: Definitions and Challenges," in *Theory of Cryptography*, Y. Ishai, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 253–273.

[62] R. Xu, J. B. Joshi und C. Li, „Cryptonn: Training neural networks over encrypted data," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2019, S. 1199–1209.

[63] R. Shokri und V. Shmatikov, „Privacy-Preserving Deep Learning," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Ser. CCS '15, Denver, Colorado, USA: Association for Computing Machinery, 2015, S. 1310–1321.

[64] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar und H. Ludwig, „Hybridalpha: An efficient approach for privacy-preserving federated learning," in *Proceedings of the 12th ACM workshop on artificial intelligence and security*, 2019, S. 13–23.

[65] B. D. Rouhani, M. S. Riazi und F. Koushanfar, „Deepsecure: Scalable provably-secure deep learning," in *Proceedings of the 55th annual design automation conference*, 2018, S. 1–6.

[66] K. Bonawitz, V. Ivanov, B. Kreuter u. a., „Practical Secure Aggregation for Privacy-Preserving Machine Learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, Ser. CCS '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, S. 1175–1191.

[67] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig und J. Wernsing, „CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy," in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan und K. Q. Weinberger, Hrsg., Ser. Proceedings of Machine Learning Research, Bd. 48, New York, New York, USA: PMLR, Juni 2016, S. 201–210.

[68] H. Chabanne, A. de Wargny, J. Milgram, C. Morel und E. Prouff, *Privacy-Preserving Classification on Deep Neural Network*, Cryptology ePrint Archive, Paper 2017/035, 2017.

[69] E. Dufour-Sans, R. Gay und D. Pointcheval, „Reading in the dark: Classifying encrypted digits with functional encryption," *Cryptology ePrint Archive*, 2018.

[70] T. Ryffel, E. Dufour-Sans, R. Gay, F. Bach und D. Pointcheval, „Partially encrypted machine learning using functional encryption," *arXiv preprint arXiv:1905.10214*, 2019.

[71] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider und F. Koushanfar, „Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, Ser. ASIACCS '18, Incheon, Republic of Korea: Association for Computing Machinery, 2018, S. 707–721.

[72] J. Liu, M. Juuti, Y. Lu und N. Asokan, „Oblivious Neural Network Predictions via MiniONN Transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, Ser. CCS '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, S. 619–631.

[73] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. E. Lauter und F. Koushanfar, „XONN: XNOR-based Oblivious Deep Neural Network Inference.," in *USENIX Security Symposium*, 2019, S. 1501–1518.

[74] G. Hinton, O. Vinyals und J. Dean, *Distilling the Knowledge in a Neural Network*, 2015.

[75] J. Wang, W. Bao, L. Sun, X. Zhu, B. Cao und P. S. Yu, *Private Model Compression via Knowledge Distillation*, 2018.

[76] A. Polino, R. Pascanu und D. Alistarh, „Model compression via distillation and quantization," *arXiv preprint arXiv:1802.05668*, 2018.

[77] Apple Differential Privacy Team. „Learning with Privacy at Scale." (2017), Adresse: https://machinelearning.apple.com/research/learning-with-privacy-at-scale (besucht am 23. 09. 2023).

[78] S. Pichai. „Keeping your private information private." (2020), Adresse: https://blog.google/technology/safety-security/keeping-private-information-private/amp/ (besucht am 23. 09. 2023).

[79] A. Herdağdelen, A. Dow, B. State, P. Mohassel und A. Pompe. „Protecting privacy in Facebook mobility data during the COVID-19 response." (2020), Adresse: https://research.facebook.com/blog/2020/06/protecting-privacy-in-facebook-mobility-data-during-the-covid-19-response/ (besucht am 23. 09. 2023).

[80] Snapchat. „Differential Privacy at Snapchat." (2022), Adresse: https://eng.snap.com/differential-privacy-at-snapchat (besucht am 23. 09. 2023).

[81] T. Karras, S. Laine und T. Aila, „A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, S. 4401–4410.

[82] A. Krizhevsky, G. Hinton u. a., „Learning multiple layers of features from tiny images," S. 32–35, 2009.

[83] Z. Liu, P. Luo, X. Wang und X. Tang, „Deep Learning Face Attributes in the Wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, Dez. 2015.

[84] K. He, X. Zhang, S. Ren und J. Sun, „Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, S. 770–778.

[85] A. Dosovitskiy, L. Beyer, A. Kolesnikov u. a., „An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[86] A. Vaswani, N. Shazeer, N. Parmar u. a., „Attention is all you need," *Advances in neural information processing systems*, Jg. 30, 2017.

[87] Microsoft, *Microsoft SEAL*, Version 4.1.1, 11. Jan. 2023.

[88] S. Halevi und V. Shoup, „Design and implementation of HElib: a homomorphic encryption library," *Cryptology ePrint Archive*, 2020.

[89] M. Keller, „MP-SPDZ: A Versatile Framework for Multi-Party Computation," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[90] FENTEC, *CiFEr*, Version 4.1.1, 11. Jan. 2023.

[91] C. R. Harris, K. J. Millman, S. J. van der Walt u. a., „Array programming with NumPy,“ *Nature*, Jg. 585, Nr. 7825, S. 357–362, Sep. 2020.

[92] A. Paszke, S. Gross, F. Massa u. a., „PyTorch: An Imperative Style, High-Performance Deep Learning Library,“ in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, S. 8024–8035.

[93] Martín Abadi, Ashish Agarwal, Paul Barham u. a., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015.

[94] J. Bradbury, R. Frostig, P. Hawkins u. a., *JAX: composable transformations of Python+NumPy programs*, Version 0.3.13, 2018.

[95] Hugging Face. „Hugging Face Models.“ (2023), (besucht am 23. 09. 2023).

[96] A. Yousefpour, I. Shilov, A. Sablayrolles u. a., „Opacus: User-Friendly Differential Privacy Library in PyTorch,“ *arXiv preprint arXiv:2109.12298*, 2021.

[97] G. Andrew, S. Chien und N. Papernot, *TensorFlow Privacy*, Version 0.8.10, Juni 2023.

[98] B. Balle, L. Berrada, S. De, J. Hayes, S. L. Smith und R. Stanforth, *JAX-Privacy: Algorithms for Privacy-Preserving Machine Learning in JAX*, Version 0.2.0, 2022.

[99] A. Sablayrolles, A. Yousefpour, K. Prasad u. a. „Differential Privacy Series Part 3 | Efficient Per-Sample Gradient Computation for More Layers in Opacus.“ (Nov. 2022), Adresse: https://pytorch.medium.com/differential-privacy-series-part-3-efficient-per-sample-gradient-computation-for-more-layers-in-39bd25df237 (besucht am 23. 09. 2023).

[100] G. Andrew, O. Thakkar, B. McMahan und S. Ramaswamy, „Differentially private learning with adaptive clipping,“ *Advances in Neural Information Processing Systems*, Jg. 34, S. 17 455–17 466, 2021.

[101] D. P. Kingma und J. Ba, „Adam: A method for stochastic optimization,“ *arXiv preprint arXiv:1412.6980*, 2014.

[102] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan und Q. V. Le, „Autoaugment: Learning augmentation strategies from data,“ in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, S. 113–123.

[103] Bundesamt für Sicherheit in der Informationstechnik, „BSI-Standard 200.2 IT-Grundschutz-Methodik,“ in Bundesamt für Sicherheit in der Informationstechnik, Bonn, 2017, S. 52–54.

[104] AT&T Laboratories Cambridge, *The Database of Faces*, 1994.

[105] J. Alammar. „The Illustrated Stable Diffusion.“ (2022), Adresse: https://jalammar.github.io/illustrated-stable-diffusion/ (besucht am 23. 09. 2023).

[106] C. Schuhmann, R. Beaumont, R. Vencu u. a., „Laion-5b: An open large-scale dataset for training next generation image-text models,“ *Advances in Neural Information Processing Systems*, Jg. 35, S. 25 278–25 294, 2022.

Anhang A

Übersicht von Hardware und Software

Die wichtigsten Komponenten, welche für die Experimente in Kapitel 5 genutzt wurden, werden im Folgenden dargestellt. Tabelle A.1 zeigt die relevantesten Hardware-Komponenten.

Art der Hardware	Hardware
Prozessor - CPU	Intel Core i9-13900KF mit 24 Kernen
Grafikkarte - GPU	Nvidia GeForce RTX 4090 mit 24GB Videospeicher/VRAM
Arbeitsspeicher - RAM	64GB DDR5 mit 6000MHz
Festplatte	SSD mit 2TB, Lesegeschwindigkeit 7000MB/s

Tabelle A.1: Übersicht der Hardware

Die Übersicht der genutzten Software ist in zwei Teile untergliedert. Der erste Teil, welcher in Tabelle A.2 dargestellt ist, umfasst grundlegende Software, welche zur Ausführung eines Python-Programms notwendig ist. Der zweite Teil setzt sich aus den wichtigsten Python Bibliotheken, sowie deren jeweiliger Versionsnummer zusammen. Dieser ist in Tabelle A.3 zu sehen.

Art der Software	Software
Betriebsystem	Microsoft Windows 11 Pro, Version 10.0.22621
Interpreter	Python 3.10.4
Treiber Grafikkarte	CUDA Toolkit, Version 11.8

Tabelle A.2: Übersicht der Software

Python Bibliothek	Version	Release der Version
PyTorch	2.0.1 für CUDA 11.8	08.05.2023
Torchvision	0.15.2 für CUDA 11.8	08.05.2023
Opacus	1.4.0	24.03.2023
NumPy	1.24.4	26.06.2023
IPython	8.14.0	02.06.2023

Tabelle A.3: Übersicht der wichtigsten Python Bibliotheken

Anhang B

Ergebnisse im Detail

Im Folgenden befinden sich die Messungen der Experimente in ausführlicher Fassung. Einige Werte wurden bereits in Kapitel 5 beschrieben.

B.1 Hyperparametertuning DPSGD CIFAR-10 Modell

Tabelle B.1 zeigt die Genauigkeit des CIFAR-10 Modells ohne die Nutzung von Differential Privacy. Dabei werden verschiedene Batch-Größen und Anzahl an trainierten Epochen betrachtet. Zusätzlich ist Datenaugmentierung mittels AutoAugment aktiviert, welche in B.2 nicht genutzt wird. In Tabelle B.3 wird DPSGD benutzt, jedoch ohne eine Anpassung von Parametern. Die Clipping-Norm ist auf 1,0 gesetzt und Datenaugmentierung wird in Form von AutoAugment genutzt. Die Datenaugmentierung fällt in Tabelle B.4 jedoch weg. Tabelle B.5 zeigt die Genauigkeit des Modells mit einer angepassten Clipping-Norm von 10^{-5}.

Epsilon	Anzahl Epochen	Genauigkeit mit Batch-Größe=64	Genauigkeit mit Batch-Größe=128	Genauigkeit mit Batch-Größe=256
∞	10	81,3 %	82,6 %	79,8 %
∞	12	82,4 %	82,8 %	81,8 %
∞	15	84,6 %	82,5 %	83,1 %
∞	20	84,5 %	84,1 %	82,5 %

Tabelle B.1: CIFAR-10 Modell mit AutoAugment ohne Differential Privacy

Epsilon	Anzahl Epochen	Genauigkeit mit Batch-Größe=64	Genauigkeit mit Batch-Größe=128	Genauigkeit mit Batch-Größe=256
∞	10	82,2 %	78,3 %	75,7 %
∞	12	79,6 %	81,9 %	80,6 %
∞	15	78,8 %	80,2 %	80,1 %
∞	20	79,5 %	82,9 %	80,9 %

Tabelle B.2: CIFAR-10 Modell ohne Augmentierung ohne Differential Privacy

CIFAR-10 Modell mit Delta=10^{-5}, mit AutoAugments und Clipping bei 1,0				
Epsilon	Anzahl Epochen	Genauigkeit mit Batch-Größe=64	Genauigkeit mit Batch-Größe=128	Genauigkeit mit Batch-Größe=256
1	1	13,9 %	16,6 %	24,7 %
1	5	23,3 %	28,1 %	36,0 %
1	10	18,1 %	29,8 %	34,9 %
1	15	24,0 %	31,2 %	36,6 %
1	20	18,4 %	28,4 %	34,9 %
1	30	17,6 %	30,9 %	36,9 %
5	1	18,8 %	28,7 %	27,1 %
5	5	30,6 %	38,9 %	41,7 %
5	10	26,1 %	40,1 %	43,3 %
5	15	30,7 %	40,7 %	47,0 %
5	20	29,2 %	39,3 %	45,4 %
5	30	32,7 %	41,4 %	47,4 %
10	1	17,8 %	26,3 %	30,6 %
10	5	31,7 %	36,7 %	42,9 %
10	10	34,4 %	39,7 %	46,6 %
10	15	32,2 %	41,5 %	48,8 %
10	20	29,1 %	42,0 %	50,6 %
10	30	32,5 %	43,1 %	52,9 %
20	1	27,6 %	31,3 %	19,4 %
20	5	34,6 %	39,3 %	43,1 %
20	10	33,9 %	44,5 %	48,1 %
20	15	33,1 %	43,9 %	52,6 %
20	20	36,2 %	45,9 %	51,4 %
20	30	37,2 %	48,9 %	55,7 %
30	1	27,5 %	29,2 %	29,3 %
30	5	37,7 %	40,6 %	46,0 %
30	10	36,7 %	42,7 %	50,8 %
30	15	38,1 %	50,2 %	53,7 %
30	20	36,1 %	49,2 %	55,5 %
30	30	40,2 %	50,0 %	56,9 %
50	1	33,0 %	30,2 %	23,2 %
50	5	40,4 %	43,6 %	48,1 %
50	10	38,7 %	48,8 %	53,3 %
50	15	38,9 %	52,4 %	54,0 %
50	20	42,7 %	53,8 %	56,6 %
50	30	45,1 %	52,3 %	59,1 %

Tabelle B.3: CIFAR-10 Modell, DPSGD mit Clipping bei 1,0 und mit AutoAugments

CIFAR-10 Modell mit Delta=10^{-5} und Clipping bei 1,0				
Epsilon	Anzahl Epochen	Genauigkeit mit Batch-Größe=64	Genauigkeit mit Batch-Größe=128	Genauigkeit mit Batch-Größe=256
1	1	24,7 %	27,7 %	32,6 %
1	5	30,2 %	35,9 %	43,4 %
1	10	35,2 %	40,7 %	44,9 %
1	15	34,3 %	41,0 %	44,6 %
1	20	32,6 %	41,0 %	43,9 %
1	30	26,5 %	39,5 %	44,5 %
5	1	34,5 %	30,6 %	34,5 %
5	5	38,8 %	40,9 %	47,5 %
5	10	40,0 %	44,3 %	48,7 %
5	15	40,9 %	47,1 %	50,6 %
5	20	40,5 %	46,1 %	51,9 %
5	30	40,5 %	44,9 %	52,0 %
10	1	32,6 %	38,5 %	35,7 %
10	5	39,6 %	43,3 %	48,6 %
10	10	39,7 %	47,1 %	50,6 %
10	15	40,4 %	47,1 %	53,3 %
10	20	39,9 %	49,8 %	54,0 %
10	30	41,7 %	50,8 %	54,4 %
20	1	34,3 %	37,6 %	37,5 %
20	5	44,0 %	47,3 %	48,7 %
20	10	45,4 %	48,5 %	53,2 %
20	15	45,8 %	51,1 %	55,0 %
20	20	45,6 %	52,4 %	56,1 %
20	30	45,2 %	54,0 %	56,9 %
30	1	34,0 %	35,8 %	39,3 %
30	5	40,9 %	47,4 %	49,9 %
30	10	46,3 %	49,5 %	54,1 %
30	15	46,9 %	52,1 %	56,9 %
30	20	46,9 %	52,6 %	58,4 %
30	30	45,7 %	55,3 %	58,5 %
50	1	41,0 %	37,8 %	35,5 %
50	5	43,7 %	49,8 %	50,1 %
50	10	47,7 %	52,5 %	56,5 %
50	15	49,0 %	52,4 %	56,5 %
50	20	48,5 %	53,0 %	60,1 %
50	30	46,1 %	56,3 %	58,4 %

Tabelle B.4: CIFAR-10 Modell, DPSGD mit Clipping bei 1,0 und ohne Augmentation

CIFAR-10 Modell mit Delta=10^{-5} und Clipping bei 0,0001					
Epsilon	Anzahl Epochen	Genauigkeit Batch-Größe 64	Genauigkeit Batch-Größe 128	Genauigkeit Batch-Größe 256	Genauigkeit Batch-Größe 512
1	1	26,8 %	30,1 %	30,9 %	30,3 %
1	5	32,3 %	37,4 %	42,2 %	42,7 %
1	10	34,3 %	40,0 %	42,5 %	46,0 %
1	15	34,1 %	41,1 %	44,1 %	46,3 %
1	20	29,2 %	41,1 %	43,9 %	47,0 %
1	30	31,4 %	40,4 %	44,5 %	46,7 %
5	1	28,9 %	34,9 %	38,5 %	30,3 %
5	5	38,6 %	44,9 %	45,9 %	48,4 %
5	10	41,1 %	47,0 %	47,8 %	50,4 %
5	15	40,3 %	46,0 %	49,4 %	53,7 %
5	20	37,0 %	46,8 %	53,6 %	53,7 %
5	30	41,7 %	50,2 %	54,8 %	55,0 %
10	1	31,3 %	31,0 %	33,4 %	20,4 %
10	5	41,6 %	43,0 %	46,9 %	47,2 %
10	10	44,4 %	49,3 %	50,9 %	54,5 %
10	15	42,3 %	48,6 %	52,8 %	54,4 %
10	20	47,0 %	52,9 %	55,3 %	54,9 %
10	30	45,5 %	52,1 %	55,7 %	59,4 %
20	1	33,0 %	37,7 %	35,1 %	25,9 %
20	5	44,6 %	46,9 %	49,5 %	49,2 %
20	10	46,5 %	50,5 %	52,0 %	56,4 %
20	15	45,0 %	51,1 %	55,8 %	57,7 %
20	20	47,4 %	55,8 %	58,2 %	58,0 %
20	30	50,6 %	56,3 %	58,0 %	60,5 %
30	1	37,0 %	36,4 %	38,7 %	25,2 %
30	5	45,0 %	45,2 %	51,5 %	49,6 %
30	10	46,7 %	51,5 %	55,1 %	55,7 %
30	15	49,1 %	53,3 %	55,4 %	59,1 %
30	20	50,6 %	54,4 %	58,1 %	59,6 %
30	30	51,2 %	56,9 %	59,9 %	61,9 %
50	1	36,6 %	40,5 %	38,6 %	28,8 %
50	5	45,1 %	49,1 %	51,3 %	51,3 %
50	10	48,7 %	52,1 %	56,3 %	57,0 %
50	15	49,2 %	54,8 %	58,6 %	60,4 %
50	20	53,4 %	55,7 %	57,4 %	60,5 %
50	30	53,7 %	56,7 %	60,0 %	63,2 %

Tabelle B.5: CIFAR-10 Modell, DPSGD mit Clipping bei 0,0001 und ohne Augmentation

B.2 Hyperparametertuning DPSGD CelebA ResNet-18 Modell

In Tabelle B.6 wird die Genauigkeit von zwei ResNet-18 Modellen gezeigt, wobei eines dieser Modelle zu Beginn vortrainiert war. Dabei wurden die Daten über AutoAugment erweitert, was in Tabelle B.7 jedoch entfernt wurde. Tabelle B.8 zeigt, wie sich die Genauigkeit dieser Modelle bei Nutzung von DPSGD verändert. Hierbei sind jedoch noch keine Parameter optimiert. Dies ist erst bei Tabelle B.9 der Fall, wo die virtuelle Batch-Größe erhöht ist und die Clipping-Norm verkleinert ist. Jedoch ist Datenaugmentierung über AutoAugment zusätzlich integriert. Tabelle B.10 vergleicht, wie sich die Nutzung von Datenaugmentierung auf das Modell auswirkt.

Epsilon	Anzahl Epochen	vortrainiert, mit AutoAugment Batch-Größe=128	nicht-vortrainiert, mit AutoAugment Batch-Größe=128
∞	1	86,6 %	88,6 %
∞	3	89,8 %	89,7 %
∞	5	91,5 %	91,4 %
∞	10	92,4 %	92,3 %

Tabelle B.6: Resnet-18 Modell mit AutoAugment ohne Differential Privacy

Epsilon	Anzahl Epochen	vortrainiert, ohne Augmentation Batch-Größe=128	nicht vortrainiert, ohne Augmentation Batch-Größe=128
∞	1	87,1 %	86,0 %
∞	3	89,9 %	90,7 %
∞	5	91,1 %	90,8 %
∞	10	91,0 %	90,1 %

Tabelle B.7: Resnet-18 Modell ohne Augmentation ohne Differential Privacy

CelebA ResNet-18-Modell mit Delta=10^{-6}, mit Augmentation, Batch-Größe=64 und Clipping bei 1,0			
Epsilon	Anzahl Epochen	vortrainiert	nicht vortrainiert
1	1	79,9 %	79,4 %
1	3	79,5 %	78,5 %
1	5	79,4 %	79,5 %
1	10	79,4 %	79,3 %
5	1	80,1 %	80,2 %
5	3	80,5 %	80,2 %
5	5	80,7 %	80,5 %
5	10	79,7 %	80,0 %
10	1	81,2 %	80,8 %
10	3	81,4 %	81,4 %
10	5	80,8 %	81,5 %
10	10	80,9 %	81,3 %

Tabelle B.8: Resnet-18-Modell mit Augmentation, Batch-Größe=64 und Clipping bei 1,0

CelebA ResNet-18-Modell mit Delta=10^{-6}, mit Augmentation, physische Batch-Größe=64, virtuelle Batch-Größe=256 und Clipping bei 10^{-5}			
Epsilon	Anzahl Epochen	vortrainiert	nicht vortrainiert
1	1	81,0 %	80,2%
1	3	82,0 %	81,8 %
1	5	81,1 %	82,0%
1	10	82,2 %	82,1%
5	1	82,7 %	81,3%
5	3	84,5 %	83,7 %
5	5	85,1 %	84,4%
5	10	85,3%	84,4%
10	1	82,1 %	82,1%
10	3	84,9 %	85,0 %
10	5	85,2%	84,3%
10	10	85,7%	85,1%

Tabelle B.9: Resnet-18-Modell mit Augmentation, Batch-Größe=256 und Clipping bei 10^{-5}

CelebA ResNet-18-Modell mit Delta=10-6, nicht vortrainiert, physische Batch-Größe=64, virtuelle Batch-Größe=256 und Clipping bei 10^{-5}			
Epsilon	Anzahl Epochen	mit AutoAugments	ohne Augmentation
1	3	81,8 %	83,6 %
1	10	82,1 %	83.6 %
5	3	83,7 %	85,0 %
5	10	84,4%	86,4 %
10	3	85,0 %	85,7 %
10	10	85,1 %	86,8 %

Tabelle B.10: Resnet-18-Modell mit und ohne Augmentation

B.3 Hyperparametertuning DPSGD CelebA Vision Transformer Modell

Tabelle B.11 zeigt die Genauigkeit der Vision Transformer Modelle, ohne die Nutzung von DPSGD, jedoch mit Datenaugmentierung. Die Datenaugmentierung ist in Tabelle B.12 jedoch deaktiviert. Tabellen B.13 zeigt, wie sich die Genauigkeit bei der Nutzung von DPSGD ohne Parameteroptimierung verändert. Die Batch-Größe ist dabei auf 16 gesetzt, da die nächstgrößere Zweierpotenz als Batch-Größe den Speicher der Grafikkarte überladen würde. In Tabelle B.14 wird deshalb eine virtuelle Batch-Größe von 128 genutzt. Außerdem wurde die Datenaugmentierung deaktiviert. Tabelle B.15 reduziert zusätzlich die Clipping-Norm auf einen Wert von 10^{-5}.

Epsilon	Anzahl Epochen	vortrainiert, mit AutoAugment Batch-Größe=64	nicht-vortrainiert, mit AutoAugment Batch-Größe=64
∞	1	80,0 %	80,0 %
∞	3	80,8 %	80,6 %
∞	5	81,7 %	81,4 %
∞	10	82,9 %	82,4 %

Tabelle B.11: Vision Transformer Modell mit AutoAugment ohne Differential Privacy

Epsilon	Anzahl Epochen	vortrainiert, mit AutoAugment Batch-Größe=64	nicht-vortrainiert, mit AutoAugment Batch-Größe=64
∞	1	80,8 %	79,1 %
∞	3	81,3 %	80,2 %
∞	5	81,8 %	81,0 %
∞	10	83,0 %	82,3 %

Tabelle B.12: Vision Transformer Modell ohne Augmentation ohne Differential Privacy

Celeb A Vision Transformer Modell mit Delta=10^{-6}, mit Augmentation, Batch-Größe=16 und Clipping bei 1,0			
Epsilon	Anzahl Epochen	vortrainiert	nicht-vortrainiert
1	1	79,4 %	78,6 %
1	3	79,1 %	78,1 %
1	5	79,4 %	78,8 %
5	1	78,3 %	78,6 %
5	3	78,1 %	78,9 %
5	5	78,8 %	78,7 %
10	1	79,4 %	79,6 %
10	3	79,8 %	78,9 %
10	5	79,4 %	79,4 %

Tabelle B.13: Vision Transformer Modell mit Augmentation, Batch-Größe=16 und Clipping bei 1,0

Celeb A Vision Transformer Modell mit Delta=10^{-5}, ohne Augmentation, virtuelle Batch-Größe=128 und Clipping bei 1,0			
Epsilon	Anzahl Epochen	vortrainiert	nicht-vortrainiert
1	1	79,4 %	79,8 %
1	3	79,4 %	80,0 %
1	5	79,5 %	79,9 %
5	1	79,4 %	79,1 %
5	3	79,9 %	79,4 %
5	5	79,9 %	79,5 %
10	1	79,6 %	80,0 %
10	3	79,9 %	78,6 %
10	5	80,1 %	79,8 %

Tabelle B.14: Vision Transformer Modell ohne Augmentation, virtuelle Batch-Größe=128 und Clipping bei 1,0

Celeb A Vision Transformer Modell mit Delta=10^{-6}, ohne Augmentation, virtuelle Batch-Größe=128 und Clipping bei $10^{-}5$			
Epsilon	Anzahl Epochen	vortrainiert	nicht-vortrainiert
1	1	80,1 %	80,0 %
1	3	79,9 %	80,1 %
1	5	80,0 %	79,7 %
5	1	79,3 %	79,1 %
5	3	79,5 %	79,2 %
5	5	80,0 %	79,7 %
10	1	79,8 %	78,4 %
10	3	80,0 %	79,9 %
10	5	80,1 %	80,3 %

Tabelle B.15: Vision Transformer Modell ohne Augmentation, virtuelle Batch-Größe=128 und Clipping bei 10^{-5}

B.4 Membership Inference Attacke CIFAR-10 Modell

Tabelle B.16 zeigt die Effektivität der Membership Inference Attacke gegen das CIFAR-10 Modell. Dabei werden unterschiedliche ϵ-Werte, sowie eine steigende Anzahl an Shadow Modellen betrachtet

Epsilon	Anzahl Shadow Modelle	Genauigkeit des Angriff
∞	8	58,0 %
∞	16	58,9 %
∞	32	57,4 %
1	8	49,9 %
1	16	50,0 %
1	32	49,9 %
5	8	49,7 %
5	16	49,8 %
5	32	49,8 %
10	8	50,1 %
10	16	50,0 %
10	32	50,1 %
20	8	50,0 %
20	16	50,2 %
20	32	50,2 %
30	8	50,3 %
30	16	49,9 %
30	32	50,2 %
50	8	50,0 %
50	16	50,2 %
50	32	50,0 %

Tabelle B.16: Membership Inference Angriff gegen CIFAR-10 Modelle