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: 23. August 2023

Unterschrift:

Kurzdarstellung

Hier steht eine deutsche Zusammenfassung der Arbeit

Abstract

This is the location of the abstract

Inhaltsverzeichnis

1 Einleitung **1**

2 Angriffe gegen Machine Learning Anwendungen **3**

2.1 De-Anonymisierung und Re-Identifikation 3
2.2 Model Inversion Attacke 4
2.3 Property Inference Attacke 5
2.4 Membership Inference Angriff 7
2.5 Data Extraction Attacke 8
2.6 Poisoning Attacke 10
2.7 Angriffe gegen Verteiltes Lernen 11
2.8 Zusammenfassung der Angriffe 14

3 Methoden zur Sicherung der Vertraulichkeit **17**

3.1 Übersicht der Pipeline 17
3.2 Aufbereitung des Datenbestands 19
3.2.1 Anonymisierung 19
3.2.2 Differential Privacy 24
3.2.3 Synthetische Daten 31
3.3 Training des Modells 35
3.3.1 Training mit Differential Privacy 35
3.3.2 Private Aggregation of Teacher Ensembles 37
3.3.3 Homomorphe Verschlüsselung 39
3.3.4 Funktionale Verschlüsselung 43
3.3.5 Verteiltes Lernen 44
3.4 Anpassung und Betrieb des Modells 48
3.4.1 Kryptografische Inferenz 49
3.4.2 Kompression des Modells 53
3.5 Zusammenfassung der Methoden 54

4 Bewertung der Methoden **57**

4.1 Kategorisierung anhand technischer Grundlage 57
4.2 Bewertung kryptografischer Methoden 58
4.3 Bewertung von Differential Privacy 61
4.4 Spezialfall Large Language Models 65

4.5 Zusammenfassung der Bewertungen 65

5 Experimente 67
5.1 Vergleich technischer Lösungen 67
5.2 Datensätze und Use Cases 67
5.3 Angriffe 67
5.4 Hyperparameter DPSGD 67

6 Diskussion 69
6.1 Handlungsempfehlung anhand der Ergebnisse 69
6.2 Offene Forschungbereiche 69
6.3 Zusammenfassung der Ergebnisse 69

Abbildungsverzeichnis 71

Tabellenverzeichnis 73

Auflistungen 75

Literatur 77

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. Im Hintergrund von ChatGPT läuft ein sogenanntes Large Language Model, welches anhand von menschlichem Feedback optimiert wurde [1].

Jedoch ist Machine Learning bereits seit Jahren in vielen Produkten verankert. So sortieren beispielsweise Soziale Medien Beiträge mittels Machine Learning nach der Beliebtheit [2], Sprachassistenten nutzen Neuronale Netze, um schneller auf Nutzereingaben zu reagieren [3], autonomes Fahren wird auf Basis von Machine Learning erforscht [4] und für Übersetzungen wird Machine Learning genutzt [5]. Sogar im Bereich der Medizin und Biologie wird Machine Learning genutzt, um die Forschung voranzutreiben. So gibt es AlphaFold [6], ein Neuonales 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, darunter Google Cloud Platform[2], Amazon Web Services[3] und Microsoft Azure[4]. Die Venture Capital Gesellschaft FirstMark gibt ein Überblick über Unternehmen, die im Machine Learning Umfeld tätig sind und Produkte, die diese Technologien unterstützen [7].

Eine Voraussetzung für diese Anwendungen sind Daten, viele Daten. Teilweise sind diese Daten privat. Diese werden von Unternehmen gesammelt und dazu genutzt, Produkte zu verbessern oder sogar neue Services zu entwickeln und anzubieten. Beispielsweise nutzen Soziale Medien die Interessen der Nutzer, um die Reihenfolge von Beiträgen zu sortieren [2]. Jedoch sind diese Daten nicht immer sicher. So wurde beispielsweise über 50 Millionen Profile des Sozialen Netzwerks Facebook ausgelesen, und anschließend wurden diese Personen

[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

in Bezug auf die US Wahl 2016 manipuliert [8]. Solche Datenlecks oder auch Sammelklagen gegen Machine Learning Modelle, wie das Beispiel GitHub Copilot [9], 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 [10]: *"Vertraulichkeit ist der Schutz vor unbefugter Preisgabe von Informationen. Vertrauliche Daten und Informationen dürfen ausschließlich Befugten in der zulässigen Wiese 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, beispielsweise durch die gewöhnliche Nutzung einer Anwendung. Damit sollen nun Modelle trainiert werden, die die Anwendung 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. 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 wurden nicht geteilt und sollen deshalb 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 können dabei Daten schützen, die in einem Modell gelernt wurden, als auch Daten auf Systemen dritter unlesbar machen. Mehrerer dieser Maßnahmen werden anschließend in einem Framework namens PrivacyFlow gebündelt. Dieses hat das Ziel, die Entwickeln Neuronaler Netze mit Methoden der Vetraulichkeitssicherung zu kombinieren und diese über eine einfache Konfiguration für Entwickler zu ermöglichen. Die Konzeption und Implementierung dieses Frameworks wird detailliert geschildert. Anhand einiger Beispieldatensätze wird gezeigt, wie sich verschiedene Methoden auf die Genauigkeit von Modellen und auf die Trainingsdauer auswirken. Diese Evaluation wird genutzt, um eine Handlungsempfehlung bezüglich der Auswahl der Methoden zu geben. Aktuelle Probleme, wie der Grad der Privatsphäre eines Modells im Verhältnis zur Leistung, werden zum Abschluss der Arbeit diskutiert. Zusätzlich wird ein Ausblick gegeben, welcher versucht, neuste Entwicklungen in der Modellarchitektur Neuronaler Netze mit den beschriebenen Methoden zu verbinden.

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 Vertraulichkeit bei Neuronalen Netzen bedrohen.

2.1 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 private, Daten an die Öffentlichkeit oder in die Hände eines Angreifers gelangen. Ein Beispiel hierfür wäre das Datenleck von Facebook, bei welchem 50 Millionen Nutzerprofile von dem Datenanalyse-Unternehmen Cambridge Analytica ausgelesen wurden. Diese Daten wurden verwendet, um die US-Wahl 2016 zu beeinflussen [8].

Allerdings kommt es auch vor, dass Unternehmen freiwillig Daten veröffentlichen. Netflix veröffentlichte 2006 einen Datenbestand, welcher Filmbewertungen von knapp 500.000 Nutzern enthält [11]. 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 [12] 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 [12] beschreiben, dass sogar politische oder religiöse Informationen herausgefunden werden können. Hierzu werden beispielsweise positive Bewertungen von religiösen Dokumentationsfilmen, die privat auf Netflix abgegeben werden, den entsprechenden öffentlichen Profilen auf IMDb zugeordnet.

2.2 Model Inversion Attacke

Bei der Model Inversion Attacke, versucht ein Angreifer, durch bestimmte Eingaben in das Modell, Rückschlüsse auf die Trainingsdaten zu ziehen. Dies kann soweit führen, dass einzelne Datensätze nachgebildet werden können [13].

Fredrikson et al. [13] 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 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 [13]

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 oder zufälligen Werten initialisiert werden. In jedem Schritt des Algorithmus wird zuerst 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 Netzen, 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 anzupassen, sodass der Wert der Verlustfunktion sinkt. Zusätzlich nutzen die Autoren einen Autoencoder, welcher das Bild zusätzlich harmonisiert. 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. 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 identisch zu dem eingegebenen Bild. Der Autoencoder von Fredrikson et al. [13] 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 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 überschreitet oder sich eine definierte Anzahl an Schritten der Wert der Verlustfunktion nicht reduziert.

Zhang et al. [14] 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 [15]. 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 beschriebenen Vorgehensweise.

2.3 Property Inference Attacke

Bei der sogenannten Property Inference Attacke versucht ein Angreifer, bestimmte Eigenschaften, Attributwerte, über die Daten eines Modells herauszufinden, welche nur indirekt von einem Modell gelernt wurden und auch nicht bewusst veröffentlicht wurden [16].

Ateniese et al. [16] zeigten erstmals, wie so ein Angriff bei Machine Learning Modellen, wie einer Support Vektor Maschine, funktionieren kann. Dabei handelt es sich um einen

White-Box Angriff. Es wird gezeigt, dass es möglich ist, herauszufinden, ob das angegriffene Modell eine bestimmte Eigenschaft der Daten gelernt hat. 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 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. [16] konnten mit diesem Angriff zeigen, dass es möglich ist, herauszufinden, ob ein Sprachmodell mit Daten der Eigenschaft "Sprecher mit indischem Dialekt "trainiert wurde.

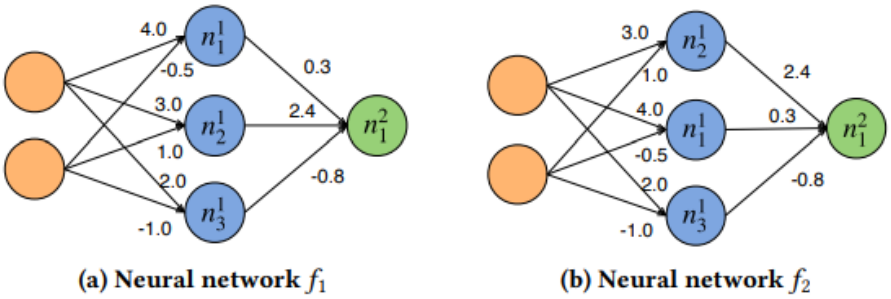

Abbildung 2.2: Permutation eines Neuronalen Netzes [17]

Die Methode von Ateniese et al. [16] 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. [17] 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 Netz basiert auf der Eigenschaft von Neuronalen Netzen, dass die Reihenfolge der Neuronen vertauscht werden kann. Abbildung 2.2 zeigt 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. [17] zeigen anhand des MNIST Datenbestands [18], dass beide Repräsentationsformen die Genauigkeit des Meta-Klassifikators erhöhen können.

Gopinath et al. [19] 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. [19] 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 Neuronenaktivierug zu nutzen, um einen Meta-Klassifikator zu trainieren.

2.4 Membership Inference Angriff

Bei der Membership Inference Attacke versucht ein Angreifer herauszufinden, ob ein Datenpunkt Bestandteil der Trainingsdatenmenge eines Modells ist. Dies bedroht die Vertraulichkeit, da beispielsweise herausgefunden werden kann, ob eine bestimmte Person Teil eines Datenmenge für die Diagnose einer Krankheit ist und folglich auch mit der entsprechenden Krankheit diagnostiziert ist [20].

Shokri et al. [20] 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 nichts ersichtlich sind und damit wie eine Art Black Box. Die Attacke wird durchgeführt, indem eine Reihe Modelle trainiert wird, die ähnlich dem Angriffsziel-Modell sind. Diese Modelle werden auch Shadow Modelle genannt. 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 und schätzen. Als Trainingsdaten kann ein Angreifer öffentliche Datenbestände nutzen. Für jedes Shadow Modell wird ein eigener Trainingsdatenbestand zusammengesetzt, wobei ein zu untersuchender Record in manchen dieser Bestände drinnen ist und in anderen wiederum

nicht. Somit haben einige der trainierten Shadow Modelle den zu untersuchenden Record in den Trainingsdaten, andere hingegen nicht. Anschließend wird ein binärer Meta-Klassifikator trainiert (ähnlich zu der Property Inference Attacke in Kapitel 2.3), welcher anhand der Vorhersagen der Shadow Modelle, Label und Confidence Score, lernt, ob ein Record im Training des entsprechenden Modells genutzt wurde. Wird in diesen Meta-Klassifikator die Vorhersage des angegriffenen Modells als Input genutzt, so erhält man die Antwort, ob der Record für das Modelltraining genutzt wurde oder nicht. Laut Shokri et al. [20] 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 Record enthalten, ein Muster gelernt werden, welches auch auf andere Modelle anwendbar ist. Overfitting eines Modells und eine hohe Anzahl an Klassen die vorhergesagt werden, erhöhen die Wahrscheinlichkeit eines erfolgreichen Angriffs auf ein Modell.

Carlini et al. [21] zeigen eine Alternative des Angriffs, bei welcher kein Meta-Klassifikator genutzt wird. Die Shadow Modelle werden analog zu [20] trainiert. Anstatt mit den Vorhersagen dieser Modelle nun einen Meta-Klassifikator zu trainieren, werden zwei Gaußverteilungen gebildet, jeweils über die Confidence Scores der Modelle, wo der Datenpunkt im Training enthalten war oder nicht. Mittels eines Likelihood-Quotienten-Tests wird anschließend vorhergesagt, in welcher Verteilung der Confidence Score des angegriffenen Modells wahrscheinlicher liegt. Ein Vorteil dieser Methode liegt darin, dass weniger Shadow Modelle trainiert werden müssen, da bereits mit relativ wenig Werten eine Gaußverteilung modelliert werden kann.

Eine Voraussetzung bei Shokri et al. [20] ist es, dass der Confidence Score mit ausgegeben wird. Choquette-Choo et al. [22] wandeln den Angriff ab, sodass dieser Score nicht mehr benötigt wird. Der Angriff funktioniert simultan zu [20], jedoch wird nicht nur der Record 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 Records, 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 Record nicht im Trainingsdatenbestand des anzugreifenden Modells ist.

2.5 Data Extraction Attacke

Bei der Data Extraction Attacke versucht ein Angreifer, Informationen eines Modells zu extrahieren, die gelernt wurden, obwohl dies (oftmals) nicht der Fall sein sollte [23]. Der

Angriff unterscheidet sich von der Model Inversion Attacke (Kapitel 2.2) und von der Property Inference Attacke (Kapitel 2.3), indem Daten direkt aus dem Modell extrahiert werden und nicht anhand der Vorhersage des Modells nachgebildet werden.

Carlini et al. [23] beschreiben, dass konkrete Zeichenketten oder Werte, wie eine Kreditkartennummer oder eine Sozialversicherungsnummer, von einem ungewollt Sprachmodell gelernt werden können. Um dies zu evaluieren, wurde ein Sprachmodell mit der Penn Treebank Datenmenge trainiert, welche ca. 5MB 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. [23] 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 [24], 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 Email-Adressen sind in der erhaltenen Tokenabfolge vorhanden. 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. [24] 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 bei 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.

2.6 Poisoning Attacke

Bei der sogenannten Poisoning Attacke werden manipulierte Records 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. [25] 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. [26] 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. [26] 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 [15] entspricht.

Guo und Liu [27] 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. [26] die Gradienten des anzugreifenden Modells nutzt, um Daten für das anzugreifende Modell zu verschlimmern. Guo und Liu [27] gehen davon aus, dass das angegriffene Modell noch optimiert wurde und deshalb eine bessere Feature Erkennung als die öffentlich vortrainierten Modelle hat. 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. [28] zeigen jedoch, dass manipulierte Daten dafür sorgen können, dass andere Angriffe, welche die Vertraulichkeit angreifen, effektiver werden. Durch Ändern des Labels eines Records 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 Records, würde eine Memberhsip Inferenze Attacke auf diesen verbessern.

2.7 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 mehr 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 [29].

Abbildung 2.3: Verteiltes Lernen Topologien [29]

Melis et al. [30] zeigen, dass Verteiltes Lernen anfällig für Membership Inference Attacken (siehe Kapitel 2.4) und Property Inference Attacken (siehe Kapitel 2.3) sind. 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 Datenpunkt oder einem Batch aus Datenpunkten 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 der diese 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 enthält. Bei jedem Update des gemeinsam gelernten Modells kann dieser Meta-Klassifikator genutzt werden.

Zhu et al. [31] 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 genutzt hat. Mit dieser Attacke, konnten die Autoren zeigen, dass die rekonstruierten Bilder nahezu (bis auf einige Pixel) identisch zu den Originalbildern sind. Die Attacke entspricht dabei einer Model Inversion Attacke 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. [32] optimierten den Deep Leakage Angriff, indem ein zusätzlicher Schritt in den Algorithmus eingefügt wird. Dieser beinhaltet, dass zuerst das Label des Records 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.

Abbildung 2.4: Rekonstruktion durch Deep Leakage [31]

Hitaj et al. [33] zeigen einen alternativen Angriff. Bei diesem Szenario einigen sich die Trainingsteilnehmer darauf, von welchen Klassen sie jeweils Daten haben und entsprechend auch welche Sie davon im gemeinsamen Modell trainieren. Klassen können sich unter den Teilnehmern überlappen, sodass mehrere Teilnehmer diese Klasse trainieren können. Jedoch braucht der Angreifer eine Klasse, die außer ihm niemand hat. 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 nun, Daten einer Klasse zu rekonstruieren, die nur von anderen Teilnehmern bespielt wurde. Dazu wird ein Generative Adversarial Network [15] 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. Der Generator wird vom Angreifer trainiert, indem er das gemeinsame Modell auch durch den Generator backpropagiert. Diese erzeugten Daten, kann der Angreifer der Klasse zuordnen, die außer ihm niemand trainiert. Daduruch werden andere 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, sondern nur Daten erzeugt, welche der gleichen Klasse zugeordnet werden können. Die Autoren zeigen jedoch, dass bei Bildern eine erkennbare Ähnlichkeit vorhanden ist.

2.8 Zusammenfassung der Angriffe

Ein Datenleck ist eines der größten Probleme von datengetriebenen Anwendungen. Selbst wenn diese anonymisiert wurden, können diese oftmals zu den Originaldaten zurückgeführt werden. Dies erfolgt oftmals statistisch mit einem weiteren (teils öffentlichem) Datensatz (Kapitel 2.1).

Jedoch können Daten auch rekonstruiert werden, wenn nur das Modell bereitgestellt wird. Dies wird Model Inversion Attacke genannt. Dabei versucht ein Angreifer, die Vorhersage eines Modells zu nutzen, um seinen Datenpunkt näher an den Originaldatenpunkt zu bringen. Dies kann über einen iterativen Prozess funktionieren oder auch komplexere, generative Modelle wie Autoencoder nutzen (Kapitel 2.2).

Wenn das Ziel nicht einzelne Datenpunkte sind, sondern Eigenschaften, die für einen ganzen Datensatz 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 einem Datensatz trainiert wurden, der die zu untersuchende Eigenschaft besitzt und andere wiederum nicht. Die Shadow Modelle werden genutzt, um einen Klassifikator zu bauen, der bestimmt, ob die Eigenschaft im genutzten Datensatz vorkommt oder nicht. Wird dies auf das originale Modell angewendet, kann ein Angreifer Aussagen zu dieser Eigenschaft treffen (Kapitel 2.3).

Die Membership Inference Attacke nutzt ebenfalls Shadow Modelle, um einen Klassifikator zu trainieren, jedoch diesmal um herauszufinden, ob ein spezifischer Datenpunkt im Training genutzt wurde. Alleine dieser Fakt könnte sensibel sein (Kapitel 2.4).

Sprachmodelle, welche das nächste Wort vorhersagen, sind besonders von der Data Extraction Attacke betroffen. Durch lange, fortlaufende Nutzung des Modells wurde ein riesiger Datensatz aus Text gesammelt, welcher nur vom Modell stammt. Dabei fanden sich sensible Informationen, die das Modell nur indirekt zum Training nutzte. Durch effizientere Sammelmethoden kann die Anzahl an sensiblen Information sogar noch erhöht werden (Kapitel 2.5).

Bei der Poisoning Attacke werden manipulierte Daten in den Datensatz integriert, 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 2.6).

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 Gradienten auch ganze Datensätze rekonstruieren (Kapitel 2.7).

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 sicher. 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 2, welche das Training des Modells umfasst. Das Ende der Pipeline, Phase 3, 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 ist die Vorverarbeitung der Daten. Die Daten können dabei aus einer Datenbank oder einem Filesystem stammen, die im Voraus gesammelt wurden. Die Vorverarbeitung der Daten ist recht individuell und kann je nach Anwendungsfall variieren. Typische Handlungen sind:

- Vereinheitlichung des Datenformats
- Fehler und starke Ausreißer werden entfernt
- Normalisierung der Daten
- Erstellung neuer Daten durch Transformationen
- Kodierung von (kategorialen) Variablen und Labels
- Aufteilung in Trainings-, Validierungs- und Testdatensätze

Die Abbildung 3.1 stellt die Vorverarbeitung in einem Schritt dar, bei dem 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.

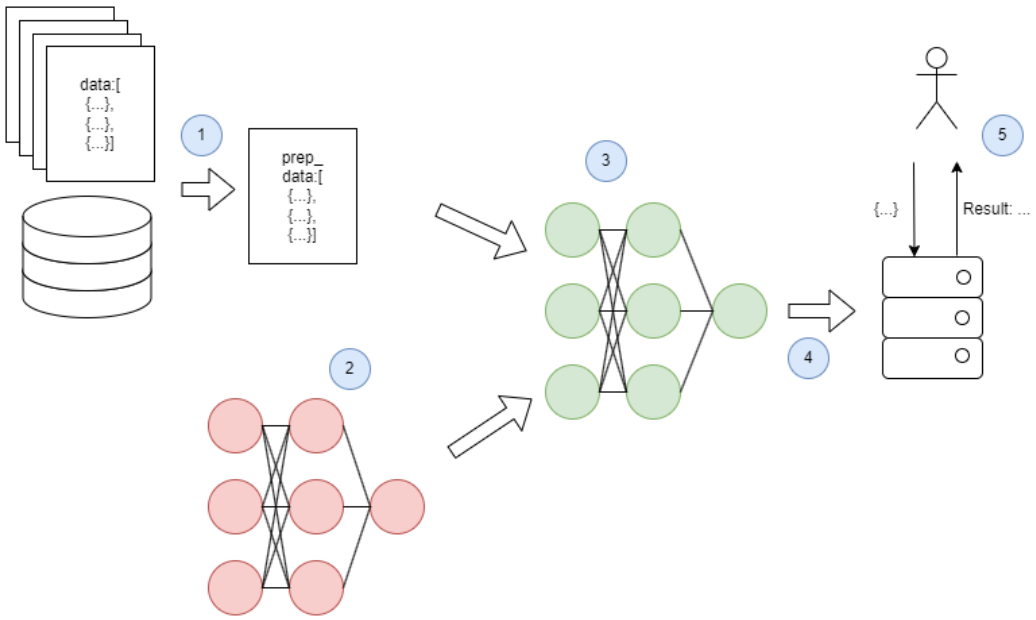

Abbildung 3.1: Training eines Neuronalen Netzes

Schritt 2 umfasst die Architektur eines Modells. Je nach Anwendungsfall und Komplexität der Aufgabe, werden unterschiedliche Konfigurationen der einzelnen Schichten vorgenommen. Dabei werden die Gewichte zufällig oder anhand einer Verteilung initialisiert. In der Abbildung wird dies durch ein rotes Modell angezeigt, da Gewichte des Modell zufällig (oder anhand einer Verteilung) initialisiert sind, und noch keine sinnvolle Vorhersage getroffen werden kann. 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. **Feedforward:** Ein Datenpunkt oder mehrere Datenpunkte, auch Batch genannt, werden durch das Modell gegeben und die Vorhersage berechnet.
2. **Backpropagation:** Die Abweichung von der tatsächlichen Vorhersage zu dem Label des Datenpunktes wird mittels einer Verlustfunktion quantifiziert. Mittels des Wertes der Verlustfunktion lassen sich Gradienten für die Gewichte des Neuronalen Netzes berechnen. Diese können dazu genutzt werden, die aktuellen Gewichte anzupassen.

Ein Durchgang der obigen Schritte mit jedem Record wird dabei Epoche genannt. Um den Trainingsfortschritt zu beobachten, kann eine Epoche zusätzlich auch eine Evaluierung mittels eines Validierungsdatenbestands enthalten. Grafisch wird das Training durch den Verbund des Dokuments und des untrainierten, roten Modells dargestellt. Es entsteht ein grünes Modell, welches in der Lage ist, sinnvolle Vorhersagen zu erzeugen. In der Praxis gibt es hier einige Iterationen des Training, welche zusätzliche Vorgänge, wie beispielsweise

eine Validierung, enthalten. 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. Dieses soll für vorgesehene Anwender erreichbar sein. 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. In 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 3 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 unkenntbar zu machen. Dabei soll dennoch ein gewisser Nutzen erhalten bleiben.

Sweeney [34] stellt eine Methode der Anonymisierung namens k-Anonymität (im Englischen k-Anonymity) vor. Die Methode wird im folgenden mittels eines Auszugs der Titanic Datenmenge [35] 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 gehen wir davon aus, dass es sich bei dem Einstiegsort um eine private Information handelt, die den Wohnort verraten könnte.

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 Variablen, welche Information über einen Datenpunkt preisgeben, aber nicht direkt auf diesen schließen lassen. Um mit Quasi-Identifikatoren einen Datenpunkt zu identifizieren, braucht es in der Regel mehr Informationen, beispielsweise einen

[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 [35]

zusätzlichen Datenbestand. In diesem Beispiel könnte also jede Spalte ein Quasi-Identifikator sein. Da der Name bereits ein direkter Identifikator ist, wird dieser anders 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 Alter dienen. Tabelle 3.2 zeigt wie diese Gruppierung aussieht.

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 [34][35]

Es ist zu sehen, dass die Identifikatoren, hier nur der Name, entfernt wurden. Geschlecht und Buchungsklasse sind auch 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. [36] zeigen anhand von 2 Attacken, dass k-Anonymität nicht ausreichend ist. Bei der Homogenitätsattacke kann die Eigenschaft, dass sensible Attribute 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 auch 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, mit dem gewisse sensible Attribute ausgeschlossen werden können oder zumindest unwahrscheinlicher machen könnten. 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. [36] 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 den ganzen Datensatz gelten. Ein Block ist dabei l-divers, wenn die Werte des sensiblen Attributes *"gut repräsentiert"* sind, wobei l eine Zahl zugeorndet 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 [36]:

- **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 das sensible Attribute 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 das häufigste sensible Attribut eines Blocks, seltener vorkommt, als die Anzahl der restlichen Attribute, 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 immernoch erfüllt ist.

Je nach Datensatz, kann es Sinn ergeben, einige Ausnahmen zu erlauben. So könnte es sein, dass ein Datensatz von einer Variable 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. Auf der anderen Seite, gibt es Attribute, die besonders geschützt werden sollten.

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. [37] 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 ist. Dabei sind 99% der Personen gesund und lediglich 1% der Personen infiziert. Die Verteilung des Attributs ist stark schief. Hat jetzt ein Block, der durch k-Anonymität entsteht, ein 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 das gleiche 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. [37] 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, müssen diese erstmal sortiert werden. Sofern es sich um eine ungleiche Anzahl an Werten handelt, können die 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:

Verteilung 1 = {20, 30, 40}
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 erhält man:

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, was bedeutet, wenn man einer Person diesem Block zuordnet könnte, ist dennoch kaum Informationsgewinnung möglich.

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 [38] vorgestellt wurde. Ziel dabei ist es, Zugriff auf einen Datensatz zu ermöglichen, der sowohl nützliche Erkenntnisse zulässt, als auch die Privatsphäre eines einzelnen Datenpunktes schützt.

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

- **Vorbereitung 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 [38] sorgt dafür, dass ein Mechanismus, auch Abfrage oder Funktion, welche eine Menge an Datensätzen als Eingabe akzeptiert, keinen konstanten Wert mehr zurück gibt. Stattdessen wird dem Mechanismus zufälliges Rauschen mit einer festgelegten Intensität hinzugefügt. Der Mechanismus gibt also eine Stichprobe einer Verteilung zurück, bei der das tatsächlichen Ergebnis dem Erwartungswert entspricht. Demnach können eindeutige Feststellungen über Eigenschaften nicht getroffen werden. Ziel der Verrauschung 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. Dabei kann der Wert ϵ, welcher auch Privacy Budget genannt wird, festgelegt werden.

Formal lautet die Definition von ϵ-Differential Privacy wie folgt [38]:
Ein randomisierter Mechanismus M, welche eine Menge an Datensätzen D auf einen Wertebereich 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 [39] fügten der Definition noch einen Parameter δ hinzu, welcher erlaubt, dass die Voraussetzungen 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 [39]:
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 Verteilung 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 [39]. 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 entscheident 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 niedirger 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.

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 Un-

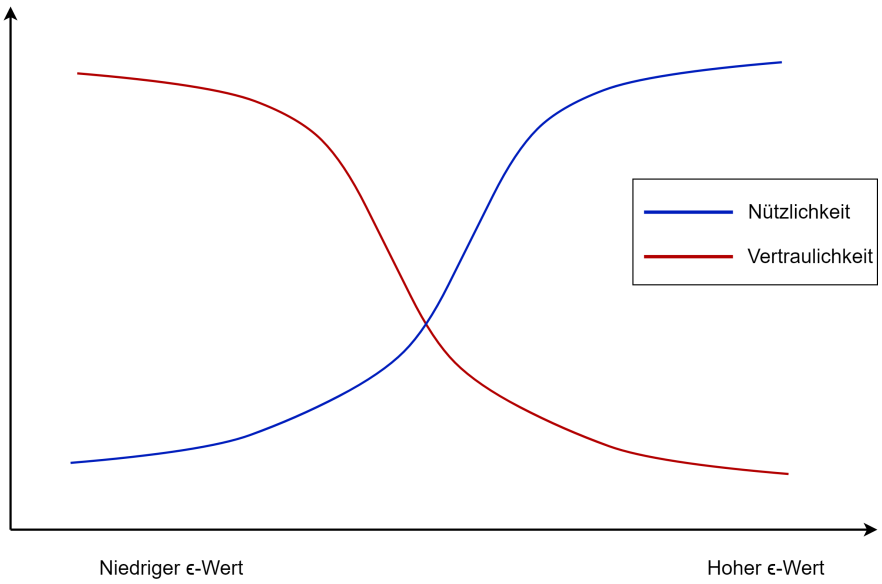

Abbildung 3.2: Einfluss von ϵ

ternehmens 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.

Algorithmen für Differential Privacy

Die Art des Rauschens, welche über die Ausgabe einer Abfrage gelegt wird, kann unterschiedlichster Herkunft sein. Dwork und Roth [39] 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

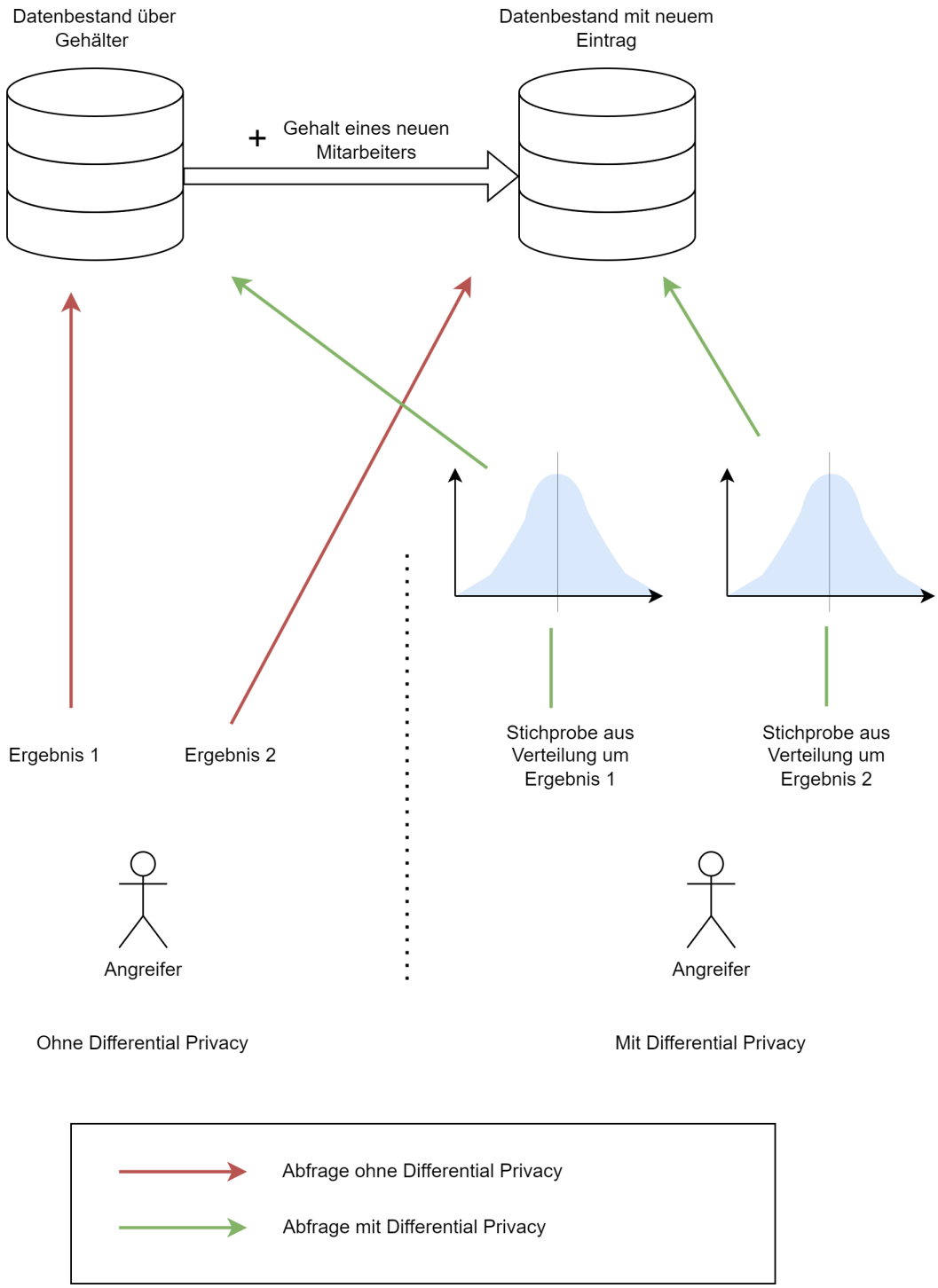

Abbildung 3.3: Beispiel Differential Privacy

Rauschen dieser Methode. Wird beispielsweise eine Person gefragt, ob sie eine gewisse Aktivität durchgeführt hat, wird mit 25-prozentiger Wahrscheinlichkeit die falsche Antwort gegeben. 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 bei X=0, lautet

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

wobei b die Steigung der Funktion beeinflusst. Um nun ein geeignetes b 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 gesehen, gibt es jedoch eine Gehaltsobergrenze, sodass fachlich eine Sensitivität festgelegt werden muss. 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 [39]:

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

Eine Abwandlung des Laplace-Mechanismus ist es, anstatt der Laplace-Verteilung, eine Gaußverteilung zu nutzen. Die Dichtefunktion dieser lautet:

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

Der Gauß-Mechanismus liefert (ϵ,δ)-Differential Privacy, wenn $b = \Delta f \frac{\ln(1/\delta)}{\epsilon}$ ist [39].

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 $u : DxR \rightarrow \mathbb{R}$, 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 Ausgabewahrscheinlichkeiten. 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 [39]:

- **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 Nachbearbeitung:** Das Ergebnis eines Mechanismus, welcher Differential Privacy nutzt, ist geschützt und es gibt keinen Mechanismus, welcher dies umkehren könnte.
- **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 [39]. 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 [39]:
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) [40]. 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 Feedforward Schritt wird demnach nicht auf den echten Daten ausgeführt, sondern auf den jeweils geschützten 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. [41] stellen solch ein Verfahren vor, welches MWEM genannt wird. Dieses basiert auf Marginalverteilungen 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[35]

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 [15]. Bei dieser Architektur gibt es 2 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.

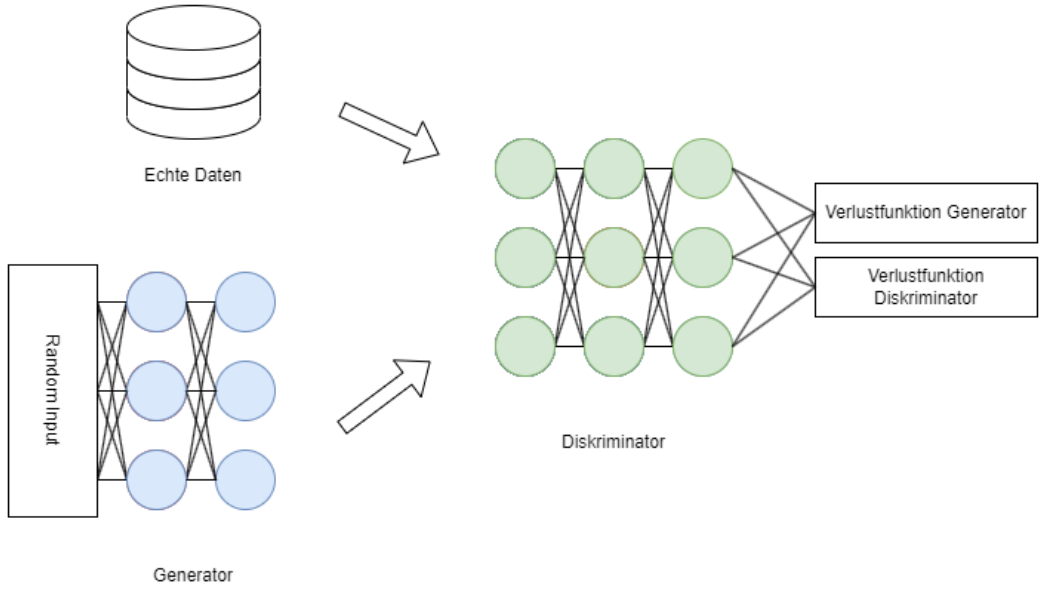

Abbildung 3.4: Generative Adversarial Network nach [15]

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 [42]. Standardmäßig kann bei GANs der Fall auftreten, dass die erzeugten Daten nicht jeden Teil einer Verteilung abbilden, sondern nur einen (z. B. den Häufigsten). 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. [43] 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 des MNIST Datensatzes [18]

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.

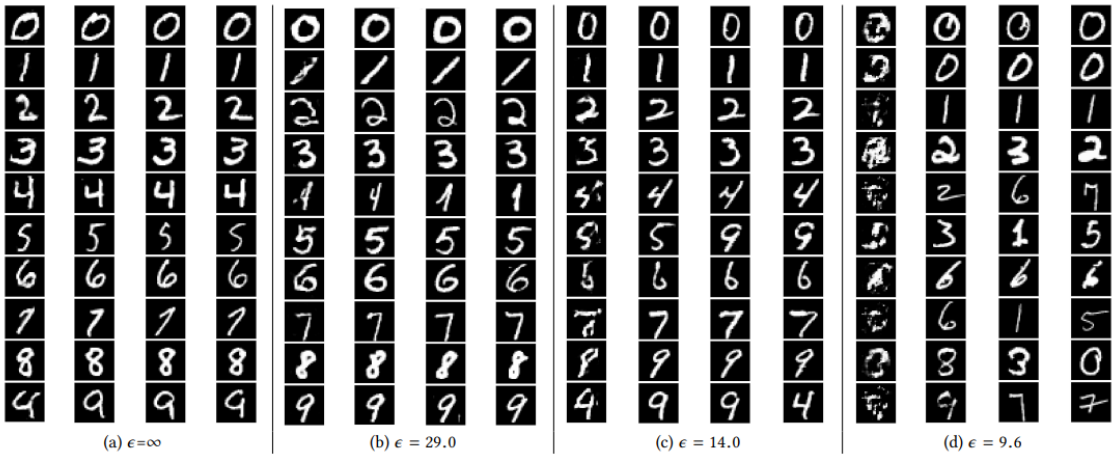

Abbildung 3.5: Synthetische MNIST Datenmenge mittels DPGAN [43]

Jordon et al. [44] 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. [45]. 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 [41], auf Marginalverteilungen. MST besteht dabei aus 3 Schritten [45]:

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 [46], 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 [46] 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. [40] 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 Feedforward Schritt genutzt. Dabei kommt jeder Datenpunkt mit der gleichen Wahrscheinlichkeit in einem Batch vor. Anders als bei einem normalen Training könnten einzelne Datensätze auch mehrmals oder gar nicht innerhalb einer Epoche zum Trainieren genutzt werden.
2. Die Gradienten werden mittels der Verlustfunktion berechnet. Dies gleicht dem Training ohne Differential Privacy.
3. Die maximale Größe der Gradienten wird beschränkt. Dies liegt daran, dass Gradienten potenziell beliebig groß werden könnten, was dafür sorgen könnte, dass das berechnete Privacy Budget nicht eingehalten werden könnte.
4. Anschließend werden die Gradienten durch den Gauß-Mechanismus verrauscht.
5. Die Anpassung der Gewichte erfolgt in die umgekehrte Richtung der Gradienten, skaliert mit einer Lernrate. Dies gleicht ebenfalls dem normalen Trainingsvorgehen.

Ein wichtiger Teil der Methode ist jedoch die Berechnung des ϵ-Werts und des δ-Werts über den Trainingsprozess hinweg. Hierbei wird für jeden Batch berechnet, welchen Einfluss dieser über die Gradienten auf die Gewichte des Modells hat. Das Rauschen in Schritt 4 wird dabei so gewählt, dass eine Anpassung der Gewichte durch einen Batch (ϵ,δ)-Differential Privacy erfüllt. Dadurch, dass ein Batch aus zufälligen Datenpunkten des Datensatzes besteht, kann das sogenannte Privacy Amplification Theorem genutzt werden [47]. 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 Datensatzgröße 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 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 [39]. 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. [40] 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. [48] erstmals vorgestellt. Dabei handelt es sich um eine sogenannte Wissenstransfer-Architektur (Knowledge Transfer), bei welcher mindesten ein Modell genutzt wird, um ein weiteres Modell zu trainieren.

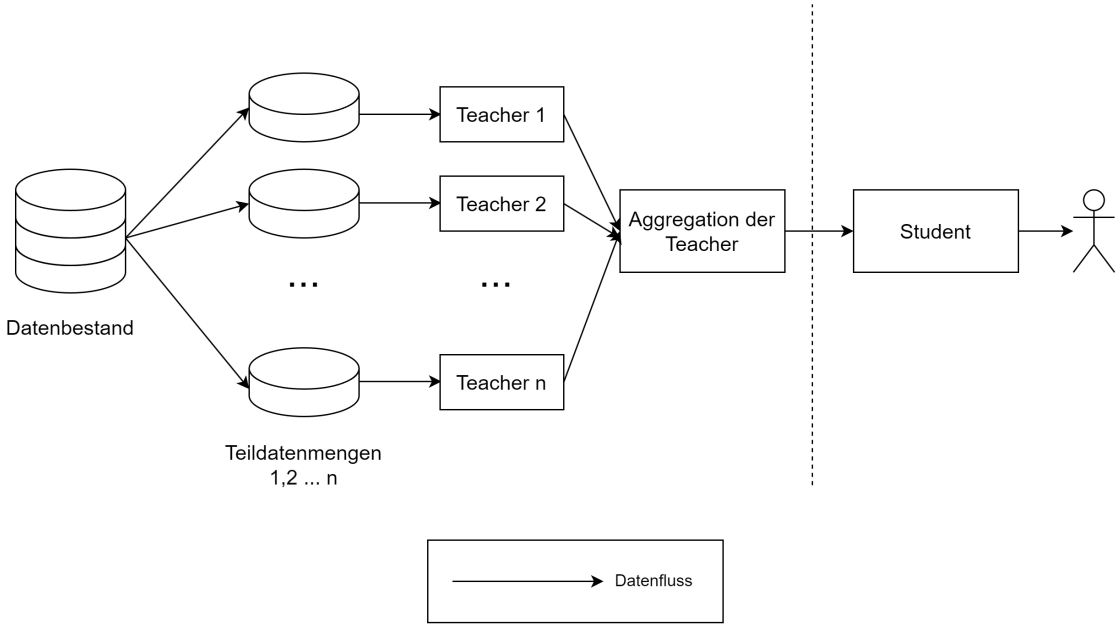

Abbildung 3.6: PATE-Architektur nach [48]

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 [40] 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 [48]. 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 wer-

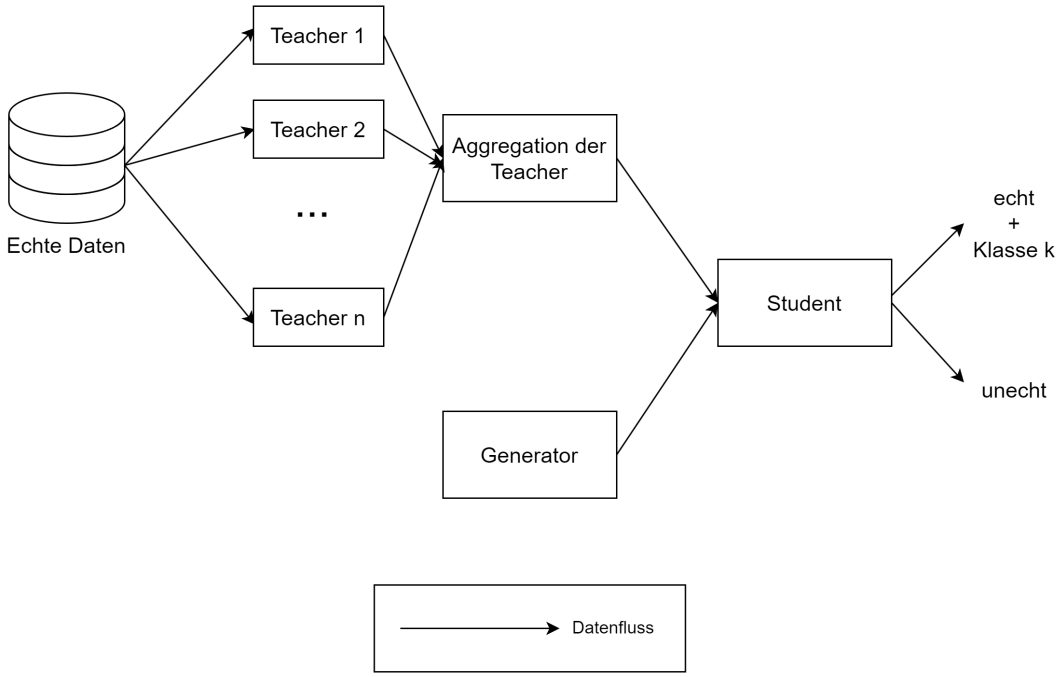

Abbildung 3.7: PATE-G

den, 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 Training 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 [49].

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 :

$G \rightarrow H$ gibt, die Elemente der Gruppe G auf die Gruppe H abbildet und dabei für alle Elemente $g_1, g_2 \in G$ gilt [50]:

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

Abbildung 3.8 zeigt eine grafische Darstellung dieses Gruppenhomomorphismus.

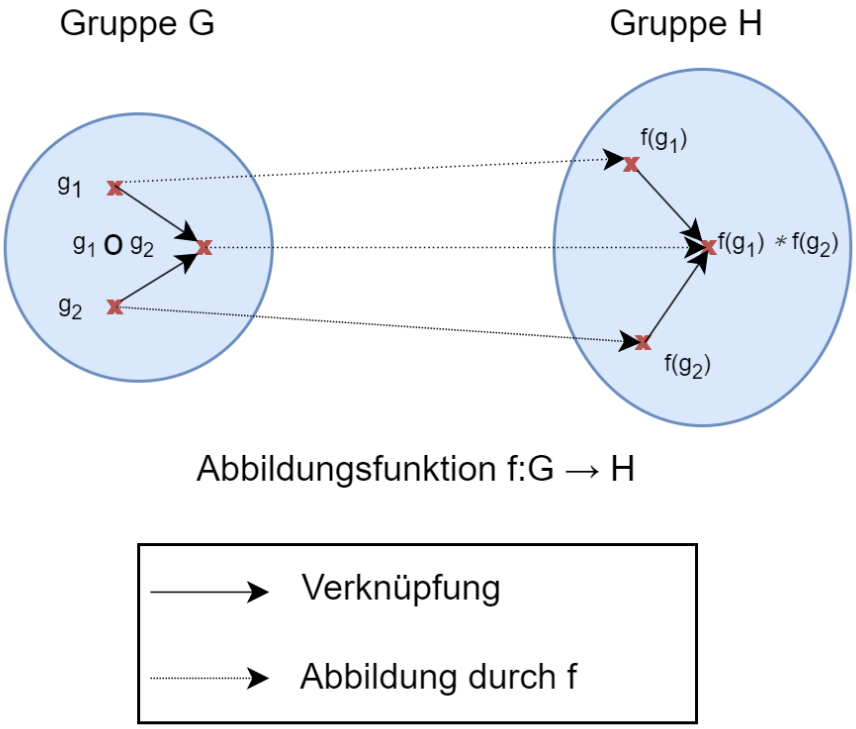

Abbildung 3.8: Gruppenhomomorphismus nach [50]

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 $Enc_k : P \rightarrow C$ und der Entschlüsselungsalgorithmus $Dec_k : C \rightarrow P$ mit einem Schlüssel $k \in K$ [50]. 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.

Homomorphe Verschlüsselungen lassen sich dabei in 3 Kategorien einteilen, je nachdem welche Verknüpfungen innerhalb der Gruppen möglich sind [51]:

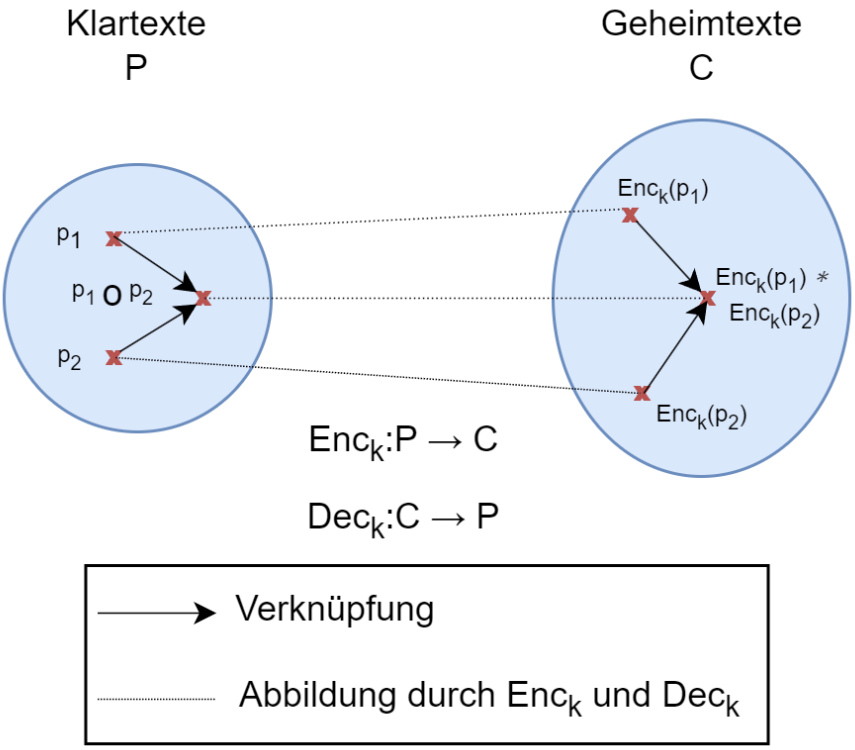

Abbildung 3.9: Homomorphe Verschlüsselung

- **Teilweise homomorphe Verschlüsselung (partially):** Entweder Multiplikation oder Addition 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

Gentry [52] 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 Boot-

strapping zu einem vollständig homomorphen Verschlüsselungssystem umgewandelt werden. So nutzen beispielsweise Van Dijk et al. [53] diesen Fakt aus, und ersetzten die auf Gittern basierte Verschlüsselung durch eine auf Ganzzahlen basierte, eingeschränkt homomorphe Verschlüsselung aus.

Brakerski und Vaikuntanathan [54] 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. [55] 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 (Inferenze, Berechnen der Verlustfunktion, Gradientenberechnung, Anpassen der Gewichte), welche mit der Größe des Neuronalen Netzes ansteigt. So wird bereits bei einem Neuronales Netz mit 7 Schichten, Faltungsschichten und vollständig verbundene Schichten, die Trainingszeit auf einer gewöhnlichen CPU, von ungefähr einer Stunde auf ein ganzes Jahr erhöht [56]. Eine Alternative ist es, keine vollständige sondern nur eingeschränkte homomorphe Verschlüsselung zu nutzen. Takabi et al. [57] 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 Verschlüsselungssytem genutzt werden muss, können performantere, teilweise homomorphe Verschlüsselungssyteme genutzt werden.

Neben dem Training eines Modells, gibt es eine Vielzahl an Techniken, die Homomorphe Verschlüsselung nur bei der Inferenz der Neuronalen Netze nutzt. Diese werden in Kapitel 3.4.1 genauer beschrieben. Alternativ kann 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. [58] 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. Dies geschieht in vier Schritten [58]:

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 ersten Ansätze um 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. [59] vorgestellt. Dieses ermöglicht, ein Modell auf einem fremden Server (Cloud) zu trainieren, ohne dass der Provider 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 Feedforward Schritt 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 Feedforward Schritt erfolgt unverschlüsselt. Wenn der Client auch das Label verschlüsselt, kann die Berechnung der Verlustfunktion für jedes Output Neuron ebenfalls verschlüsselt erfolgen. Abbildung 3.10 zeigt das CryptoNN Framework.

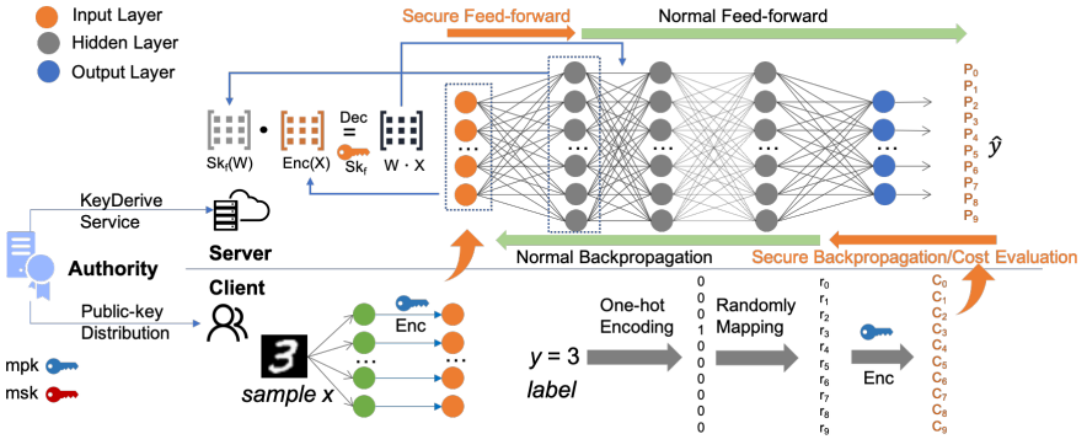

Abbildung 3.10: CryptoNN Framework [59]

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

Das Verteilte Lernen bietet einige besondere Herausforderungen, die bereits in Kapitel 2.7 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 die 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 auf das Verteilte Lernen ausgerichtet sind. Im Folgenden werden einige davon genauer beschrieben.

Distributed Selective SGD

Shokri und Shmatikov [60] 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 mit in der Größe zu begrenzen oder Rauschen mittels Differential Privacy hinzuzufügen. Abbildung 3.11 zeigt die Architektur von Distributed Selective SGD.

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 [40] das 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. [57] 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

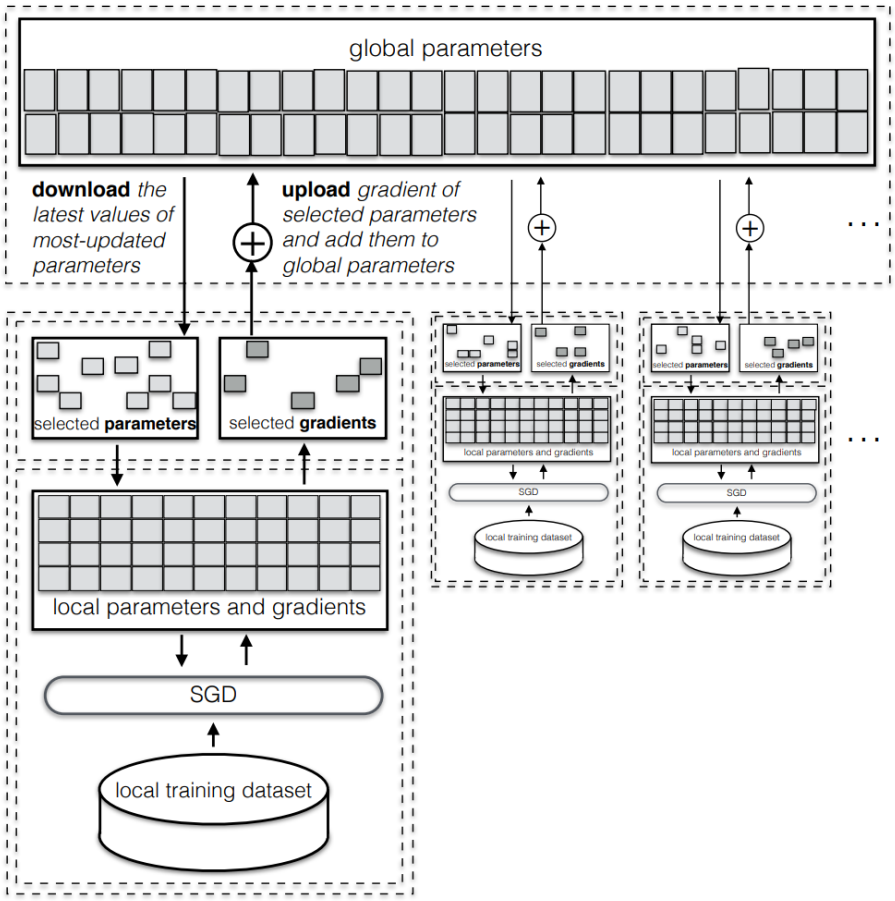

Abbildung 3.11: Distributed Selective SGD [60]

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 [59], 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 Verteiltes Lernen, welches auf Funktionaler Verschlüsselung basiert, wurde von Xu et al. [61] 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 eige-

nen 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 des MNIST Datensatze [18], 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. Methoden dieses kryptografischen Forschungsgebiets können auch für Neuronale Netze genutzt werden.

Rouhani et al. [62] 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 Feedforward Schritt 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 der Gradienten 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 Feedforward Schritt ungefähr 2 Gigabyte an Daten übertragen.

Aggregation

Eine alternative Methode wird von Bonawitz et al. [63] 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 bei Bedarf genutzt werden, um grundlegende Sicherheit zu gewährleisten. 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. [57], 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 [64] 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 [18] 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. [65] 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 dieser Funktionen 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 den MNIST Datensatz [18], 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. [65] würde demnach das Training im Vergleich zu CryptoNets [64] stabilisieren, jedoch würde durch die Approximation Genauigkeit eingebüßt werden.

Inferenz mittels Funktionaler Verschlüsselung

Das CryptoNN Framework [59], welches in Kapitel 3.3.4 beschrieben wurde, enthält bereits einen den Feedforward Schritt 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. [66] 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 dem MNIST Datensatz [18] 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. [67] 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 [62], welches in Kapitel 3.3.5 beschrieben wurde, enthält bereits einen 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. [68] 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 Vorraus 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.

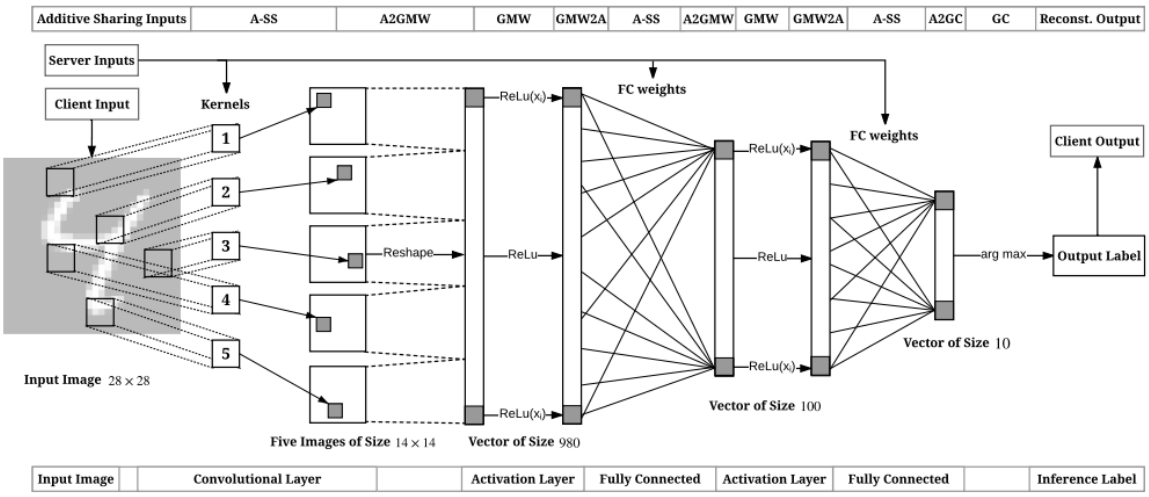

Abbildung 3.12: Chameleon Framework [68]

Das Chameleon Framework kombiniert diese Methoden, um eine effiziente, kryptografisch sichere Inferenz für Neuronale Netze zu ermöglichen. Abbildung 3.12 zeigt, wie das Framework an einem Modell des MNIST Datensatzes [18] angewendet wird. 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 [69] 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 [70] 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. [71] 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 Studen-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. [72] 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. [73] 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 nähesten 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 Datensätze, die maximal einen unterschiedlichen Datenpunkt 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 Datensätze, 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 Datenpunkt 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).

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 Feedforward Schritt, 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).

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 [74], Google [75], Meta [76] und Snapchat [77] 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. [57]
	Inferenz	CryptoNets [64]
	Inferenz	Eingeschränkt homomorphe Verschlüsselung nach Chabanne et al. [65]
Funktionale Verschlüsselung	Training + Inferenz	CryptoNN [59]
	Inferenz	Funktionale Verschlüsselung für Polynome mit Grad 2 nach Dufour-Sans et al. [66]
	Inferenz	Funktionale Verschlüsselung für Polynome mit Grad 2 nach Ryffel et al. [67]
Yao's Garbled Circuits	Inferenz	DeepSecure [62]
	Inferenz	Chameleon [68]
	Inferenz	MiniONN [69]

Tabelle 4.1: Übersicht kryptografischer Methoden

Phase des Modells

Jede der dargestellten Methoden unterstützt die Inferenz, doch 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 dieser 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 [64]. 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. Dabei 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 [59], 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[18] 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 [68] 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 Leistungunterschied 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 [64] 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, nur approximiert werden können. 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 [59] 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 [68] 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 ist eine Methodik, welche einschränken kann, wie stark sich ein einzelner Datensatz auf eine Abfrage an des gesamten Datenmenge auswirkt. Dadurch kann die Privatsphäre einzelner Datenpunkte 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, sind die Daten geschützt, 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 synthetischen 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 [42] oder DPGAN [43], 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 [78] 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. Synthetische Datensätze erzeugen empfiehlt sich also nur, wenn diese Daten auch veröffentlicht werden sollen. Ansonsten ist es sinnvoller, andere Methoden zu nutzen.

Die PATE-Architektur [48] 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 und gesonderte Sensitivitäten und Privacy Budgets ermittelt werden. 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 die 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 [72]. 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.

Schutz vor Angriffen

Wahl Epsilon

Qualität der Modelle

Hyperparameter Tuning aus Experimenten

4.4 Spezialfall Large Language Models

4.5 Zusammenfassung der Bewertungen

Kapitel 5

Experimente

5.1 Vergleich technischer Lösungen

PyTorch Opacus vs. TensorFlow Privacy vs. Jax Libraries

5.2 Datensätze und Use Cases

5.3 Angriffe

5.4 Hyperparameter DPSGD

Kapitel 6

Diskussion

6.1 Handlungsempfehlung anhand der Ergebnisse

Sicherheit versus Performance/Accuracy

6.2 Offene Forschungbereiche

6.3 Zusammenfassung der Ergebnisse

Abbildungsverzeichnis

2.1 Rekonstruktion eines Bildes [13] 4
2.2 Permutation eines Neuronalen Netzes [17] 6
2.3 Verteiltes Lernen Topologien [29] 11
2.4 Rekonstruktion durch Deep Leakage [31] 13

3.1 Training eines Neuronalen Netzes 18
3.2 Einfluss von ϵ 26
3.3 Beispiel Differential Privacy 27
3.4 Generative Adversarial Network nach [15] 33
3.5 Synthetische MNIST Datenmenge mittels DPGAN [43] 34
3.6 PATE-Architektur nach [48] 37
3.7 PATE-G 39
3.8 Gruppenhomomorphismus nach [50] 40
3.9 Homomorphe Verschlüsselung 41
3.10 CryptoNN Framework [59] 44
3.11 Distributed Selective SGD [60] 46
3.12 Chameleon Framework [68] 52

Tabellenverzeichnis

3.1 Nicht-Anonymisierter Titanic Datensatz [35] 20
3.2 k-Anonymity Titanic Datensatz [34][35] 20
3.3 Angreifbare Abwandlung von Tabelle 3.2 21
3.4 Beispiel Marginalverteilungen[35] 32

4.1 Übersicht kryptografischer Methoden 58
4.2 Übersicht Differential Privacy Methoden 62
4.3 Komplexität Differential Privacy Methoden 63

Auflistungen

Literatur

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

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

[3] S. Team, *Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant*, https://machinelearning.apple.com/research/hey-siri, 2017, [Online; Abgerufen 19.Juni 2023].

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

[5] I. Caswell und B. Liang, *Recent Advances in Google Translate*, https://ai.googleblog.com/2020/06/recent-advances-in-google-translate.html, 2020, [Online; Abgerufen 19.Juni 2023].

[6] 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.

[7] FirstMark, *The 2023 MAD (ML/AI/Data) Landscape*, https://mad.firstmark.com/, 2023, [Online; Abgerufen 09.Mai 2023].

[8] C. Cadwalladr und E. Graham-Harrison, *Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach*, https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election, März 2018, [Online; Abgerufen 08.Mai 2023].

[9] J. Corbet, *Class action against GitHub Copilot*, https://lwn.net/Articles/914150/, Nov. 2022, [Online; Abgerufen 09.Mai 2023].

[10] Bundesamt für Sicherheit in der Informationstechnik, „IT-Grundschutz-Kompendium," in Bundesamt für Sicherheit in der Informationstechnik, Bonn, 2023, S. 42, ISBN: 978-3-8462-0906-6.

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

[12] 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.

[13] 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, ISBN: 9781450338325.

[14] 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.

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

[16] 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.

[17] 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, ISBN: 9781450356930.

[18] Y. LeCun, C. Cortes und C. Burges, „MNIST handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, Jg. 2, 2010.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos und D. Song, *The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks*, 2019.

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

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

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

[27] 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, ISBN: 978-3-030-58583-9.

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

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

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

[36] 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, ISSN: 1556-4681.

[37] 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.

[38] 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, ISBN: 978-3-540-35908-1.

[39] 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, ISSN: 1551-305X.

[40] 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.

[41] 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.

[42] 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.

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

[44] 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.

[45] 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.

[46] 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.

[47] R. Bassily, A. Smith und A. Thakurta, „Private empirical risk minimization, revisited," *rem*, Jg. 3, S. 19, 2014.

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

[49] B. Van der van der Waerden, E. Artin und E. Noether, *Moderne Algebra*. Springer Berlin Heidelberg, 1937, §10, ISBN: 9783662356043.

[50] X. Yi, R. Paulet und E. Bertino, „Homomorphic Encryption," in *Homomorphic Encryption and Applications*. Cham: Springer International Publishing, 2014, S. 27–46, ISBN: 978-3-319-12229-8.

[51] 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, ISSN: 0360-0300.

[52] 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, ISBN: 9781605585062.

[53] 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, ISBN: 978-3-642-13190-5.

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

[55] 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, ISBN: 978-3-642-40041-4.

[56] 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, ISSN: 1936-6450.

[57] 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.

[58] 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, ISBN: 978-3-642-19571-6.

[59] 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.

[60] 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, ISBN: 9781450338325.

[61] 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.

[62] 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.

[63] 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, ISBN: 9781450349468.

[64] 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.

[65] 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.

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

[67] 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.

[68] 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, ISBN: 9781450355766.

[69] 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, ISBN: 9781450349468.

[70] 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.

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

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

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

[74] A. D. P. Team, *Learning with Privacy at Scale*, https://machinelearning.apple.com/research/learning-with-privacy-at-scale, 2017, [Online; Abgerufen 13.August 2023].

[75] S. Pichai, *Keeping your private information private*, https://blog.google/technology/safety-security/keeping-private-information-private/amp/, 2020, [Online; Abgerufen 13.August 2023].

[76] A. Herdağdelen, A. Dow, B. State, P. Mohassel und A. Pompe, *Protecting privacy in Facebook mobility data during the COVID-19 response*, https://research.facebook.com/blog/2020/06/protecting-privacy-in-facebook-mobility-data-during-the-covid-19-response/, 2020, [Online; Abgerufen 13.August 2023].

[77] Snapchat, *Differential Privacy at Snapchat*, https://eng.snap.com/differential-privacy-at-snapchat, 2022, [Online; Abgerufen 13.August 2023].

[78] 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.