Fakultät Elektrotechnik, Medien und Informatik

Konzeption und Implementierung eines Frameworks zur Sicherung der Vertraulichkeit bei 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:

Konzeption und Implementierung eines Frameworks zur Sicherung der Vertraulichkeit bei 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: 12. Juni 2023

Unterschrift:

Kurzdarstellung

Hier steht eine deutsche Zusammenfassung der Arbeit

Abstract

This is the location of the abstract

Inhaltsverzeichnis

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. Die Venture Capital Gesellschaft FirstMark gibt ein Überblick über Unternehmen, die im Machine Learning Umfeld tätig sind und Produkte, die diese Technologien unterstützen [2].

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 Nutzerdaten, 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 in Bezug auf die US Wahl 2016 manipuliert [3].

Das Bundesamt für Sicherheit in der Informationstechnik, kurz BSI, definiert das Schutzziel der Vertraulichkeit wie folgt [4]: *"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. Datenlecks, wie das Beispiel von Facebook [3], oder auch Sammelklagen gegen Machine Learning Modelle, wie das Beispiel GitHub Copilot [5], zeigen, dass hier allerdings noch Nachholbedarf besteht.

[1] https://openai.com/blog/chatgpt
[2] https://github.com/twitter/the-algorithm

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. Anschließend wird eine Reihe von Maßnahmen aufgeführt, die das Ziel haben, die vorher genannten Angriffe unwirksam zu machen. Eine Menge dieser Maßnahmen wird anschließend in einem Framework namens PrivacyFlow gebündelt. Die Konzeption und Implementierung dieses Frameworks wird detailliert geschildert. Nach Zusammenfassen der Ergebnisse, kann eine Handlungsempfehlung erstellt werden. Zum Abschluss der Arbeit werden offene Probleme der Forschung genannt und wie diese möglicherweise in Zukunft gelöst werden.

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 worden sind. Diese wurden genutzt, um die US-Wahl 2016 zu beeinflussen [3].

Allerdings kommt es auch vor, dass Unternehmen freiwillig Daten veröffentlichen. Netflix veröffentlichte 2006 einen Datensatz, welcher Filmbewertungen von knapp 500.000 Nutzern enthält [6]. Um nicht absichtlich private Daten zu veröffentlichen, war der Datensatz anonymisiert. Neben einem Wettbewerb steht dieser Datensatz zu Forschungszwecken öffentlich zur Verfügung. Narayanan und Shmatikov [7] zeigen jedoch, dass die Anonymisierung des Datensatzes nicht ausreichend war, um private Informationen zu schützen. Die Bewertungen der anonymisierten Benutzer, wurden mit den Bewertungen der öffentlichen Filmdatenbank IMDb abgeglichen. Dabei genügt es, wenn Präferenzen in Korrelation gesetzt werden können, die genauen Wert sind nicht notwendig. Narayanan und Shmatikov [7] beschreiben, dass sogar politische oder religiöse Informationen herausgefunden werden können. Hierzu werden beispielsweise positive Bewertungen von religiösen Dokumentationsfilmen, die privat auf Netflix abgegeben werden, werden öffentlichen Profilen auf IMDb zugeordnet.

2.2 Property Inference Attacke

Bei der sogenannten Property Inference Attacke versucht ein Angreifer, bestimmte Eigenschaften über die Daten eines Modells herauszufinden, welche nur indirekt von einem Modell gelernt wurden und auch nicht bewusst veröffentlicht wurden [8].

Ateniese et al. [8] zeigten erstmals, wie so ein Angriff bei Machine Learning Modellen ,wie einer Support Vektor Maschine, funktionieren kann. 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 Datensätze, in denen die zu untersuchende Eigenschaft vorhanden ist oder nicht. Anschließend werden diese Datensätze genutzt, um verschiedene Modelle zu trainieren, welche die gleiche Architektur wie das angegriffene Modell nachbilden. Der Schlüssel dieser Methode ist es, nun 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 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, die gleiche Architektur haben und dadurch auch zum gleichen Format transformiert werden können. Ateniese et al. konnten mit diesem Angriff zeigen, dass es möglich ist, herauszufinden, ob ein Sprachmodell mit Daten der Eigenschaft "Sprecher mit indischem Dialekt "trainiert wurde.

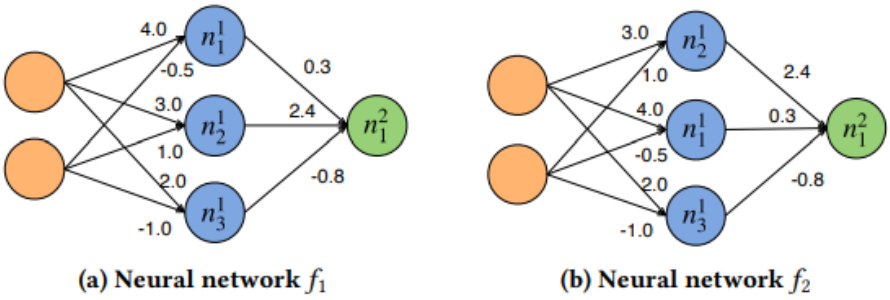

Abbildung 2.1: Permutation eines Neuronalen Netzes [9]

Die Methode von Ateniese et al. [8] 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. [9] passten den Angriff auf Neuronale Netze an, indem zwei geeignete Repräsentation für diese Art von Modellen genutzt werden kann. Die erste Repräsentation eines Neuronalen Netz basiert auf der Eigenschaft von Neuronalen Netzen, dass die Reihenfolge der Neuronen vertauscht werden kann. Abbildung 2.1 zeigt zwei Neuronale Netze, bei denen die Reihenfolge der Knoten in der ersten Hidden Layer vertauscht ist, jedoch das Modell die gleiche Funktion berechnet. Diese Eigenschaft

kann nun genutzt werden, um jede Schicht zu sortieren und dadurch eine einheitliche Matrix für Permutationen des gleichen Modells zu erhalten.

Die zweite Repräsentation beruht darauf, die Schichten eines Neuronalen Netzes nicht als Vektor darzustellen. Im Gegensatz zu einem Vektor hat ein Set keine feste Reihenfolge oder Ordnung, sondern ist lediglich eine Menge von Objekten, bzw. hier von Knoten. Dies sorgt ebenfalls dafür, dass Permutationen des gleichen Modells, in das gleiche Format übertragen werden können. Ganju et al. [9] zeigen anhand des MNIST Datensatzes, dass beide Repräsentationsformen die Accuracy des Meta-Klassifikators erhöhen können.

Gopinath et al. [10] zeigen, dass viele Informationen eines Neuronalen Netzes in der Aktivierung der Neuronen steckt. Dabei reicht es, einen Wert in **on** und **off** zu unterteilen, wobei on einem Wert > 0 entspricht. Gopinath et al. [10] nutzten diese Darstellung für eine gutwillige Informationsgewinnung über ein Modell, jedoch könnte die Darstellung auch für Property Inference Angriffe genutzt werden.

2.3 Model Inversion Attacke

Bei der Model Inversion Attacke, versucht ein Angreifer, durch bestimmte Eingaben in das Modell, Rückschlüsse zu den Trainingsdaten zu ziehen. Dies kann soweit führen, dass einzelne Datenpunkte nachgebildet werden können [11].

Fredrikson et al. [11] zeigen anhand eines Gesichterkennungsmodells, dass es möglich ist, das Bild einer Person zu rekonstruieren. Diese Person muss lediglich von dem Modell klassifiziert werden können, folglich also auch in den Trainingsdaten vorhanden sein. Abbildung 2.2 zeigt, wie sehr das rekonstruierte Bild (links) dem Originalbild (rechts) ähnelt.

Abbildung 2.2: Rekonstruktion eines Bildes [11]

Der Angriff, auch Reconstruction Attacke genannt, wird durch einen generativen, iterativen Algorithmus durchgeführt. Zu Beginn wird ein Bild als Startbild gesetzt, welches jedem Pixel den Wert 0 zuweist. In jedem Schritt des Algorithmus wird zuerst der Wert einer

Verlustfunktion bestimmt. Der Wert dieser ist, sofern das Modell nicht mehr Details angibt, lediglich der Confidence Score, dass es sich bei dem eingegebenen Bild nicht um das gesuchte Label handelt. Konkret bedeutet dies, wenn das Bild bereits die zu rekonstruierende Person zeigt, der Confidence Score des Labels der Person nahe 1 ist und damit unsere Verlustfunktion nahe 0. Anschließend wird das Bild des vorherigen Durchlaufs, durch einen Autoencoder geschickt, welcher ein Bild für den nächsten Schritt des Algorithmus erzeugt. 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 den Input transformiert. Input und Output eines Autoencoders haben somit den gleichen Rang, hier wird also ein Bild zu einem anderen Bild transformiert. Zum Training des Autoencoders werden Bilder genutzt, wobei der Input auch gleich dem 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. Dieser Algorithmus läuft so lange, bis die maximal konfigurierte Anzahl an Schritten erreicht wurde, die Verlustfunktion einen bestimmten Wert überschritten hat oder sich eine definierte Anzahl an Schritten lang nichts verbessert hat.

Zhang et al. [12] erweitern diesen Angriff, indem die Qualität des generativen Modells (Autoencoder) verbessert wird. Zum einen wird der Autoencoder so erweitert, dass zusätzliche Informationen mitgegeben werden können. Diese zusätzlichen Informationen sind beispielsweise verschwommene oder zensierte Bilder. 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 [13]. Mithilfe des Diskriminators, kann der Autoencoder verbessert werden, wodurch die Qualität der einzelnen Bilder erhöht wird und folglich auch die Qualität des rekonstruierten Bildes steigt.

He et al. [14] zeigen eine White Box Version des Angriffs. White Box bedeutet, dass das Modell vollumfänglich in den Händen des Angreifers ist. Dies ist der Fall, wenn ein Modell öffentlich geteilt wird (die Trainingsdaten jedoch nicht). Die Fähigkeit, das Modell vollumfänglich zu nutzen, erlaubt es, das zu rekonstruierende Bild mittels Backpropagation anzupassen. Dazu wird initial ein Bild genutzt, bei jedem jeder Pixel auf einen einheitlichen Farbwert, z. B. 0.5, gesetzt wird. Dieses Bild wird nun durch das Modell interferiert, und der Wert der Verlustfunktion wird backpropagiert. Alle Gewichte und Bias des Modells werden dabei unverändert gelassen, jedoch werden die Pixel des Bildes Gradienten bestimmt und mit einer konfigurierten Lernrate angepasst. Dieser Vorgang wird anschließend so lange wiederholt, bis eine maximale Anzahl an Iterationen durchgelaufen ist.

2.4 Membership Inference Angriff

Bei der Membership Inference Attacke versucht ein Angreifer herauszufinden, ob ein Datenpunkt Bestandteil des Trainingsdatensatzes eines Modells ist. Dies bedroht die Vertraulichkeit, da beispielsweise herausgefunden werden kann, ob eine bestimmte Person Teil eines Trainingsdatensatzes für die Diagnose einer Krankheit ist und folglich auch mit der entsprechenden Krankheit diagnostiziert ist [15].

Shokri et al. [15] führen eine Membership Inference Attacke durch, 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 Funktion der Modelle, als auch der Trainingsdatensatz, mit dem angegriffenen Modell vergleichbar sind. Einige dieser trainierten Modelle enthalten den zu untersuchenden Datenpunkt im Trainingsdatensatz, andere hingegen nicht. Nun wird ein binärer Meta-Klassifikator trainiert (analog zu der Property Inference Attacke in Kapitel 2.2), welcher anhand der Vorhersagen der Shadow Modelle, Label und Confidence Score, lernt, ob ein Datensatz 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 Datenpunkt für das Modelltraining genutzt wurde oder nicht. Laut Shokri et al. [15] funktioniert dieser Angriff, da ähnliche Modelle, die mit einem ähnlichen Datensatz trainiert wurden, sich auch ähnlich verhalten. Somit kann bei den selbst trainierten Modellen, welche den Datensatz enthalten, ein Muster gelernt werden, welches auch auf andere Modelle anwendbar ist. Overfitting und eine komplexe Modellarchitektur erhöhen die Wahrscheinlichkeit eines erfolgreichen Angriffs.

Carlini et al. [16] zeigen eine Alternative des Angriffs, bei welcher kein Meta-Klassifikator genutzt wurde. Die Shadow Modelle werden analog zu [15] 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. [15] ist es, dass der Confidence Score mit ausgegeben wird. Choquette-Choo et al. [17] wandeln den Angriff ab, sodass dieser Score nicht mehr benötigt wird. Der Angriff funktioniert simultan zu [15], jedoch wird nicht nur der Datenpunkt 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 Datenpunkten, 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.

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

Carlini et al. [18] beschrieben, dass konkrete Zeichenketten oder Werte, wie eine Kreditkartennummer oder eine Sozialversicherungsnummer, in einem Sprachmodell gelernt werden. Um dies zu evaluieren, wurde ein Sprachmodell mit dem Penn Treebank Datensatz trainiert, welcher ca. 5MB groß ist. Zusätzlich wurde ein Satz eingefügt, welcher mit *"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. [18] zeigen, dass dieses Modell die Zahlenfolge ungewollt gelernt hatte als mögliche Vorhersage ausgibt, wenn der oben genannte Satz als Input genutzt wird.

In einem anderen Forschungsprojekt, ebenfalls unter der Leitung von Carlini, [19] 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 Modell anwendbar wäre. Zur Durchführung des Angriffs wird lediglich ein Starttoken in das Modell eingegeben und anschließend vielfach das vorgeschlagene Folgewort gesammelt. Wird dies lang genug gemacht, erhält man eine lange Tokenabfolgen, also quasi Sätze, 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 vorhanden. Diese Variation des Angriffs kann in gewissem Maße funktionieren, liefert jedoch oftmals gleiche Wortabfolgen und hat auch eine hohe False-Positive Rate. Carlini et al. [19] variierten deshalb die Methodik, wie 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. Wird der Softmax Funktion ein Hyperparameter namens Temperatur > 1 mitgegeben, wird das Modell unsicherer und erhöht dadurch die Diversität der Vorhersagen des Modells. Neben dieser Temperatur wird eine zweite Verbesserung vorgeschlagen. Anstatt nur einen Starttoken zu nutzen, werden die ersten Wörter von verschiedenen, öffentlichen Datenquellen genutzt. Mit diesen zwei Verbesserungen konnten mehr unterschiedliche Arten von Texten, die das Modell gelernt hat, extrahiert werden. Neben Newsartikeln oder Forumsbeiträgen, befanden sich auch Kontaktdaten einiger Privatpersonen in diesen Tokenabfolgen.

2.6 Poisoning Attacke

Bei der sogenannten Poisoning Attacke werden manipulierte Datensätze in den Trainingsdatensatz 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. [20] zeigt, dass einige modifizierte Datenpunkte in der Nähe der Entscheidungsgrenze genügen, um die gelernte Funktion deutlich negativ zu beeinflussen.

Yang et al. [21] zeigen ein Verfahren, bei dem eine Poisoning Attacke auf Neuronalen Netzen angewendet wird. Ziel hierbei ist es, die Daten mit einem falschen Label so zu wählen, dass der Wert der Verlustfunktion möglichst groß ist. Dadurch werden die Gradienten größer, was auch bedeutet, dass die negative Beeinflussung des infizierten Datenpunktes größer wird. Um diese Daten zu erzeugen, nutzen Yang et al. [21] einen Autoencoder, der Daten so transformiert, dass diese vom Modell als echte Daten erkannt werden, jedoch aufgrund der falschen Labels, das Modell möglichst stark negativ beeinflussen. Dieser Autoencoder wurde trainiert, indem die Verlustfunktion des angegriffenen Modells auch durch den Autoencoder backpropagiert wurde.

Guo und Liu [22] 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. [21] die Gradienten durch das Generator Modell backpropagieren. Guo und Liu [22] 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 verschieden sind.

Poisoning Attacken verschlechtern in der Regel lediglich die Performance eines Modells und sorgen für falsche Vorhersagen. Tramèr et al. [23] zeigen jedoch, dass manipulierte Daten

dafür sorgen können, dass andere Angriffe, welche die Vertraulichkeit angreifen, effektiver werden. Durch Ändern des Labels eines Datenpunktes 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.

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 [24].

Abbildung 2.3: Verteiltes Lernen Topologien [24]

Melis et al. [25] zeigen, dass Verteiltes Lernen anfällig für Membership Inference Attacken (siehe Kapitel 2.4) und Property Inference Attacken (siehe Kapitel 2.2) 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 einem Datensatz, welcher die zu untersuchende Eigenschaft enthält und einmal mit einem Datensatz 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 die zu untersuchende Eigenschaft enthält. Bei jedem Update des gemeinsam gelernten Modells kann dieser Meta-Klassifikator genutzt werden.

Zhu et al. [26] 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 interferiert, 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. Abbildung 2.4 zeigt einer Gegenüberstellung der Bilder. Die Bilder können ebenfalls rekonstruiert werden, wenn mehrere Bilder in Batches zum Training genutzt werden.

Zhao et al. [27] optimierten den Deep Leakage Angriff, indem ein zusätzlicher Schritt in den Algorithmus eingefügt wird. Dieser beinhaltet, dass zuerst das Label des Datenpunktes herausgefunden wird, von dem die Gradienten stammen. Dies kann dadurch ermittelt werden, dass die Kostenfunktion eines Outputs bei der richtigen Klasse den Wertebereich (-1,0) annimmt und bei den falschen Klassen den Wertebereich (0,1), sofern keine Quadrierung stattfindet. Der initiale Eingabevektor hat nun von Beginn an das richtige Label. Dadurch werden weniger Iterationen benötigt, um Daten zu rekonstruieren.

Hitaj et al. [28] zeigen einen alternativen Angriff. Bei diesem Szenario einigen sich die Trainingsteilnehmer darauf, von welchen Klassen sie jeweils Daten haben und entsprechend auch

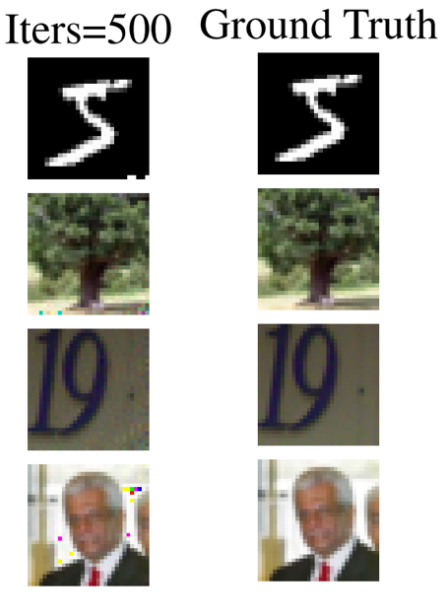

Abbildung 2.4: Rekonstruktion durch Deep Leakage [26]

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 [13] genutzt. Dabei handelt es sich um eine Architektur, welche sich aus zwei Modellen, einem Generator und einem Discriminator, zusammensetzt. Der Discriminator entscheidet dabei, ob die Daten, die vom Generator erzeugt werden, echt oder unecht sind. In diesem Angriff wird als Discriminator 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.

Kapitel 3

Methoden zur Sicherung der Vertraulichkeit

Neben den beschriebenen Angriffen aus Kapitel 2 gibt es eine Vielzahl an Methoden, diese abzuschwächen oder sogar ganz zu unterbinden. Dieses Kapitel stellt zuerst eine typische Pipeline zum Training eines Neuronalen Netzes vor und teilt diese in 3 Phasen ein. Phase 1 ist dabei die Aufbereitung des Datensatzes, 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.

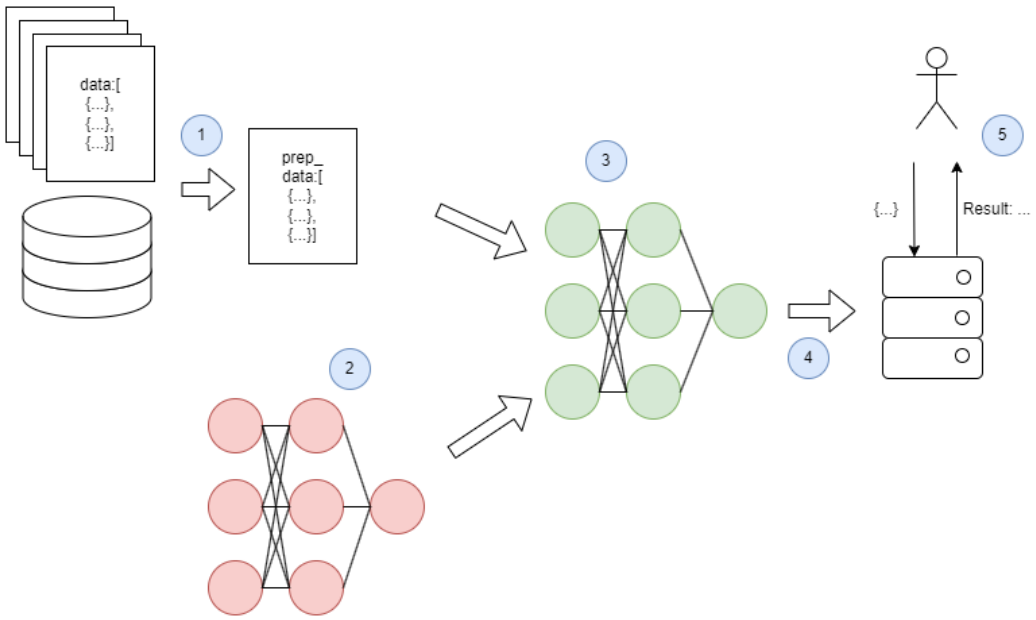

Abbildung 3.1: Training eines Neuronalen Netzes

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 ein mehrere Datenpunkte (Batch) werden durch das Modell gegeben und die Vorhersage berechnet. Die Abweichung von der tatsächlichen Vorhersage zu dem Label des Datenpunktes wird mittels einer Verlustfunktion quantifiziert.
2. **Backpropagation:** 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 Datenpunkt wird dabei Epoche genannt. Um den Trainingsfortschritt zu beobachten, kann eine Epoche zusätzlich auch eine Evaluierung mittels eines Validierungsdatensatzes 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 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 Datensatzes

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 [29] stellt eine Methode der Anonymisierung namens k-Anonymität (im Englischen **k**-Anonymity) vor. Die Methode wird im folgenden mittels eines Auszugs des Titanic Datensatzes [30] erläutert. Tabelle 3.1 zeigt einen Auszug aus diesem Datensatz. 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 [30]

Datensatz. 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, eine 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 [29][30]

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. [31] 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. [31] 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 [31]:

- **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. [32] 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. [32] 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 2 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 zu 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 [33] 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 laut Dwork [33] sorgt dafür, dass die gleiche Anfrage auf zwei unterschiedlichen Datensätze, die in höchstens einem Datenpunkt voneinander abweichen, keine signifikant unterschiedlichen Ergebnisse aufweise. Um dies zu erreichen, wird den Anfragen ein zufälliges Rauschen hinzugefügt. Dadurch handelt es sich bei Abfragen um randomisierte Funktionen. Die Größe des Unterschied der gleichen Abfrage auf zwei Datensätze, die sich in höchstens einem Eintrag unterscheiden, kann durch einen Wert ϵ dargestellt werden. Dieser Wert wird auch Privacy Budget genannt.

Formal lautet die Definition von ϵ-Differential Privacy wie folgt [33]:
Eine randomisierte Funktion M, welche einen Datensatz D auf einen Wertebereich R abbildet, weist ϵ-Differential Privacy auf, wenn für alle Datensätze D_1 und D_2 die sich in höchstens einem Datenpunkt unterscheiden, gilt:

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

Dwork und Roth [34] fügten der Definition noch einen Parameter δ hinzu, welcher erlaubt, dass die Voraussetzungen zu einem definierten Grad verletzt werden können. Die damit angepasste Definition von (ϵ,δ)-Differential Privacy lautet [34]:
Eine randomisierte Funktion M, welche einen Datensatz D auf einen Wertebereich R abbildet, erfüllt (ϵ,δ)-Differential Privacy, wenn für alle Datensätze D_1 und D_2, die sich in höchstens einem Datenpunkt unterscheiden, gilt:

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

Konkret sagen die Definitionen aus, dass eine randomisierte Funktion, auf beiden Datensätzen angewendet, nahezu die gleichen Ergebnisse liefert. Dabei bestimmten ϵ und δ wie stark sich die Ergebnisse unterscheiden dürfen. Kleinere Werte bedeuten dabei, dass die Privatsphäre besser geschützt ist, was sich jedoch negativ auf die Nützlichkeit der Abfragen auswirkt.

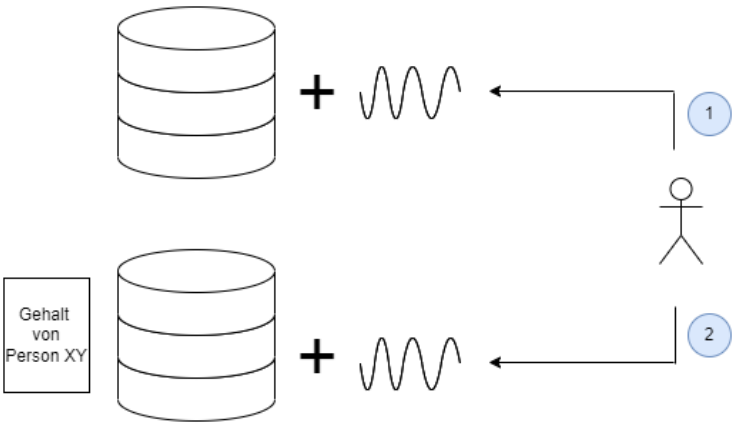

Abbildung 3.2: Beispiel Differential Privacy

Abbildung 3.2 zeigt exemplarisch, wie Differential Privacy anhand einer Abfrage des Durchschnittsgehalts eines Unternehmens aussehen könnte. Dabei führt ein Angreifer 2 Anfragen aus, auf die gleiche Datenbank, welche aber bei Anfrage 2 um einen zusätzlichen Gehaltseintrag eines neuen Mitarbeiters erweitert wurde. Würde kein Differential Privacy genutzt werden, könnte anhand der Anzahl der Mitarbeiter und der beiden durchschnittlichen Gehältern das exakte Gehalt des neuen Mitarbeiters berechnet werden. Wird nun Differential Privacy genutzt, wird ein zufälliges Rauschen über das Ergebnis der Anfrage gelegt, wodurch Anfrage 1 und 2 ein nahezu identisches Ergebnis ausliefern, abhängig der Parameter ϵ und δ.

Eigenschaften von Differential Privacy

Das Ziel von Differential Privacy, die Vertraulichkeit eines Datenpunktes zu schützen und dennoch die Nützlichkeit des Datensatzes zu bewahren, wurde bereits zu Beginn des Kapitels geschildert. Jedoch bringt Differential Privacy eine Reihe weiterer nützlicher Eigenschaften mit sich[34]:

- **Gruppen Privacy:** Differential Privacy betrachtet zwei Datensätze, die sich in einem Punkt unterscheiden. Jedoch gelten die gleichen Regeln für Datensätze, die sich in mehreren Datenpunkten unterscheiden, wobei ϵ dabei mit der Anzahl der unterschiedlichen Datenpunkte multipliziert wird.

- **Resistenz gegen Nachbearbeitung:** Eine Anfrage, welche Differential Privacy nutzt, kann nicht im Nachgang so bearbeitet werden, dass die Privatsphäre weniger geschützt sei.
- **Quantifizierung der Vertraulichkeit:** Mit den Parametern ϵ und δ kann angegeben werden, wie stark die Vertraulichkeit der Daten geschützt wird.
- **Zusammensetzung von Berechnungen:** Durch die Quantifizierung der Vertraulichkeit, können auch zusammengesetzte Berechnungen bewertet werden.

Algorithmen für Differential Privacy

Die Art des Rauschens, welche über die Ausgabe einer Abfrage gelegt wird, kann unterschiedlichster Herkunft sein. Dwork und Roth [34] 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. 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 $Pr[Antwort = "Ja" \mid Wahrheit = "Ja"] = 3/4$ und $Pr[Antwort = "Nein" \mid Wahrheit = "Ja"]$. Die jeweils ausgegebene Antwort könnte dadurch glaubhaft abgestritten werden.

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

Formel 3.3 zeigt nicht nur die Anwendung der randomisierten Antwort, sondern auch, dass es in diesem Fall um (ln 3, 0)-Differential Privacy handelt.

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, so kann ein neuer Datenpunkt eine Änderung um den Wert 1 erzeugen, folglich ist $\Delta f = 1$. Auch bei Erzeugung eines Histogramms kann ein neuer Datenpunkt maximal eine Änderung um den Wert 1 erreichen. 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:

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

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.

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 Datensatz, oder eine Teilmenge eines Datensatzes, liefert also keine Zahl, sondern ein Datenpunkt aus diesem. Möglichen Ausgabeoptionen sind dabei R und die Bewertungsfunktion ist $u : DxR \to \mathbb{R}$ mit einer Sensitivität von Δu. 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 eine Anfrage genutzt werden, die ausgeben soll, ob Krankheit A oder B häufiger vorkommt. Somit wären die möglichen Optionen R=A,B, die Bewertungsfunktion $u(D, r) = \text{Count(r in D)}$ und die Sensitivität $\Delta u = 1$. Für jede der Optionen in R, würde die Ausgabewahrscheinlichkeit mit Formel 3.7 berechnet werden. Anschließend wird anhand dieser Wahrscheinlichkeiten, entweder A oder B ausgegeben.

Vorverarbeitung der Trainingsdaten

Bisher wurden nur einzelne Anfragen in Bezug auf Differential Privacy betrachtet. Es gibt jedoch auch Verfahren, welche es ermöglichen, einen ganzen Datensatz zu veröffentlichen, welcher mit Differential Privacy geschützt wird. Hardt et al. [35] stellen solch ein Verfahren vor. Dabei ist anzumerken, dass hier eine Variable des Datensatzes betrachtet wird. Um einen Datensatz umzuwandeln, wird D' mit der gleichen Anzahl an Elementen wie D initialisiert, wobei jedoch die Werte mittels einer Gleichverteilung über alle potenziellen Werte ermittelt wird. Das Verfahren besteht aus 3 Schritten, welche iterativ wiederholt werden können, um die künstliche Verteilung D' an die echte Verteilung D anzugleichen:

1. **Wahl einer Anfrage:** Aus allen potenziell möglichen Anfragen an den Datensatz wird mittels Exponential-Mechanismus die Anfrage gewählt, welche den größten Unterschied zwischen den Eingaben D' und D besitzt.
2. **Verrauschen:** Mittels Laplace-Mechanismus wird nun das Ergebnis der gewählten Anfrage aus Schritt 1 mit dem originalen Datensatz D verrauscht.
3. **Anpassen von D':** Abhängig von dem verrauschten Ergebnis der gewählten Anfrage, werden die Werte von D' nach oben oder nach unten angepasst.

Nach Beendigung des Algorithmus, ist D' ein synthetischer Datensatz, welcher den originalen Datensatz D mit Differential Privacy abbildet.

3.2.3 Synthetische Daten

Die Erzeugung synthetischer Daten wurde im Jahre 2016 maßgeblich durch die Generative Adversarial Architektur, auch GAN genannt, beeinflusst [13]. 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, die abwechselnd von dem echten Datensatz und dem Generator kommen. Im Gegensatz dazu, versucht der Generator, Daten zu erzeugen, die den Diskriminator täuschen. Der Diskriminator kann dabei 2 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 Daten generiert. Im besten Falle, erzeugt der Generator nach dem Training Daten, die so echt wirken, dass der Diskriminator diese nicht mehr unterscheiden kann. Folglich erzeugt der Generator synthetische Daten, die latente Strukturen der originalen Daten enthalten. Abbildung 3.3 die bereits beschriebene Architektur.

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 [36]. 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 Datenpunkten gearbeitet, sondern jeweils mit Batches trainiert wird. Der Austausch der Verlustfunktion sorgt dafür, dass der der Generator Daten aus der ganzen Verteilung der Daten nachstellen muss und nicht nur aus einem Teil dieser.

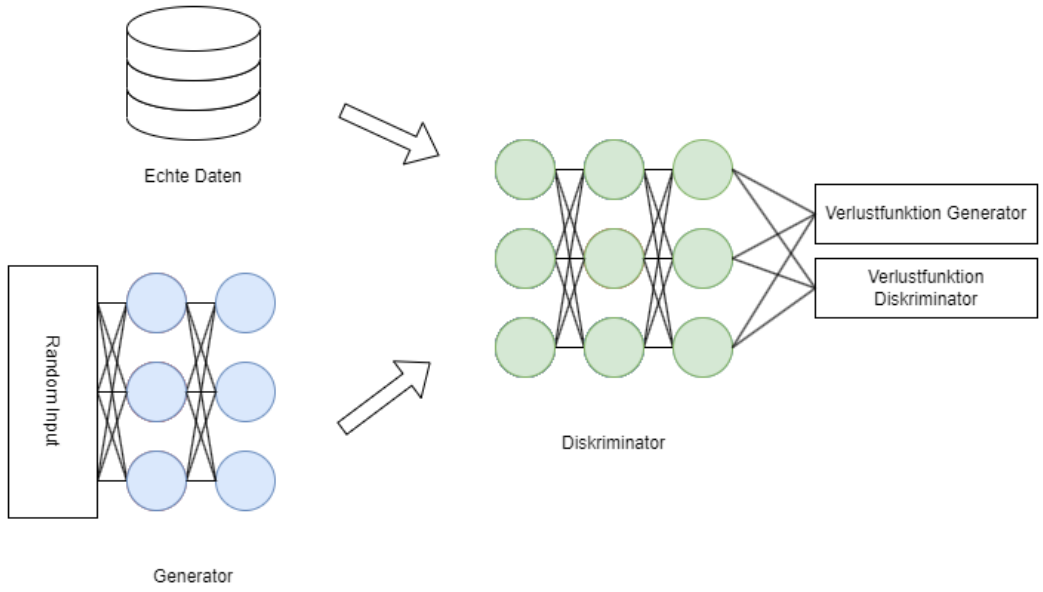

Abbildung 3.3: Generative Adversarial Network nach [13]

Xie et al. [37] 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 zu erzeugen. Abbildung 3.4 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.

Jordon et al. [38] stellten eine alternative Form des GANs vor, welches ebenfalls synthetische Daten erzeugt. Dieses nutzt das Private Aggregation of Teachen Ensembles Framework, kurz PATE, welches in Kapitel 3.3.2 im Detail beleuchtet wird. Bei der PATE Architektur, werden die Daten in verschiedene Teildatensätze unterteilt. Verschiedene Modelle, die sogenannten Lehrer oder Teacher, lernen die Klassifikation jeweils an einem unterschiedlichen Teildatensatz. Ein weiteres Modell, welches Schüler oder Student genannt wird, kann nun mittels des Exponential-Mechanismus aus 3.2 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

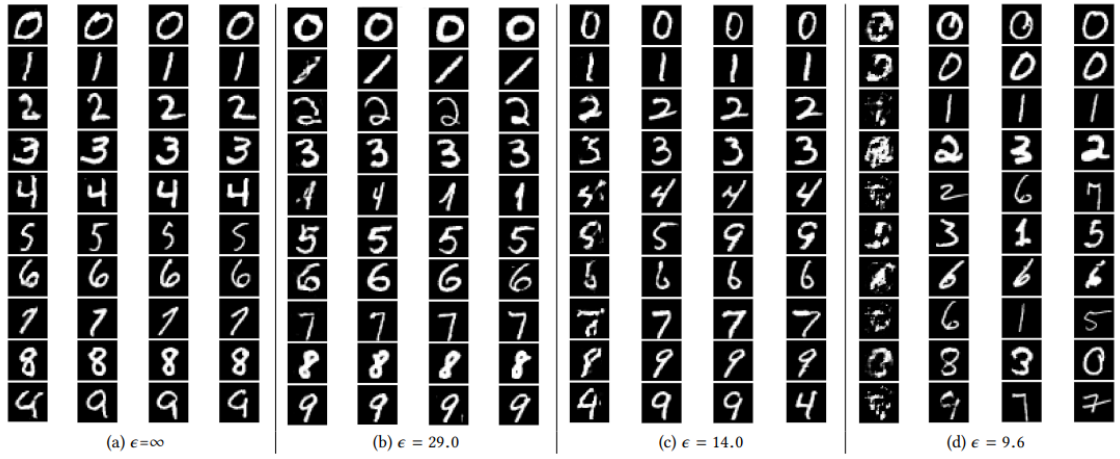

Abbildung 3.4: Syntehtischer MNIST Datensatz mittels DPGAN [37]

PATE-GAN die Resistenz gegenüber Nachbearbeitungen von Differential Privacy aus, um Differential Privacy auch für die synthetischen Daten zu garantieren.

Ein Algorithmus, welcher kein GAN zur Erzeugung künstlicher Daten nutzt, ist NIST-MST von McKenna et al. [39]. 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. MST besteht dabei aus 3 Schritten [39]:

1. **Wahl von Marginalverteilungen:** Bei Marginalverteilungen handelt es sich um Anzahl der Elemente von zwei Variablen (Spalten) einem Datensatz in Abhängigkeit und Kombination zueinander. Variablen können dabei mehrfach genutzt werden. Aus dem Datensatz, über den synthetische Daten erzeugt werden, können mehrere dieser Marginalverteilungen gewählt werden. Dabei sollten die wichtigsten Zusammenhänge und Abhängigkeiten des Datensatzes in diesen vorkommen. Es wird deshalb empfohlen, dass ein Fachexperte die Marginalverteilungen aussucht.
2. **Zählen der Marginalverteilungen:** Marginalverteilungen entahlten die Anzahl einer Variable in Abhängigkeit von der anderen Variable enthält. 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 [40], welches von den gleichen Autoren stammt, um einen künstlichen Datensatz zu erzeugen. Dabei sollen die Marginalverteilungen des künstlichen Datensatzes möglichst nahe an den gemessenen Marginalverteilungen liegen.

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. [41] 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. Die Gradienten werden mittels der Verlustfunktion berechnet. Dies gleicht dem Training ohne Differential Privacy.

2. Die maximale Größe der Gradienten wird beschränkt. Dies liegt daran, dass Gradienten potenziell beliebig groß werden könnten, jedoch für die Berechnung des Privacy Budget die maximale Änderung der Funktion, die Sensitivität Δf, benötigt wird. Diese entspricht der maximalen Größe der Gradienten quadriert.

3. Anschließend werden die Gradienten durch den Gauß-Mechanismus verrauscht.

4. Die Anpassung der Gewichte erfolgt in die umgekehrte Richtung der Gradienten, skaliert mit einer Lernrate. Dies gleicht ebenfalls dem normalen Trainingsvorgehen.

Entsprechend der Parameter des Gauß-Mechanismus (Formel 3.6 in Kapitel 3.2.2), kann gezeigt werden, dass jeder Schritt (ϵ,δ)-Differential Privacy erfüllt. Zusätzlich integrieren die Autoren eine Methode, Moments Accountant, welche den Verlust des Privacy Budgets über den Trainingsprozess überwacht.

3.3.2 Private Aggregation of Teacher Ensembles

Die Private Aggregation of Teacher Ensembles (Pate) Architektur wurde 2017 von Papernot et al. [42] erstmals vorgestellt. Dabei handelt es sich um eine sogenannte Wissenstransfer-Architektur (Knowledge Transfer), bei welcher ein Modell genutzt wird, um ein weiteres Modell zu trainieren.

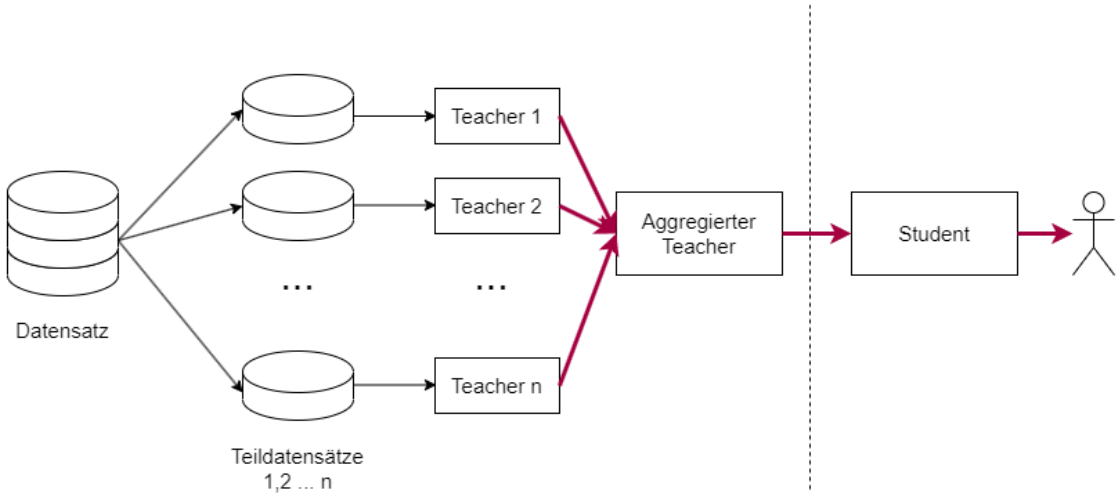

Abbildung 3.5: PATE Architektur nach [42]

Abbildung 3.5 zeigt eine Übersicht der PATE Architektur. Der Datensatz wird dabei zuerst in verschiedene Teildatensätze aufgeteilt. Für jeden dieser Teildatensätze wird anschließend ein sogenanntes Teacher oder Lehrer Modell trainiert. Wenn das gesamte Modell eine Klassifikation aus 10 unterschiedlichen Klassen ist, dann muss jedes Teacher Modell ebenfalls diese Klassen abbilden. Hierbei ist es ebenfalls wichtig, dass die Teildatensätze nicht zu klein sind, da ansonsten die Teacher Modelle nicht gut funktionieren würden. Die Vorhersagen aller Teacher Modelle können aggregiert werden, indem die Vorhersagen der Teacher Modelle addiert werden. An dieser Stelle werden die Anzahlen jeder Klasse mittels des Laplace-Mechanismus verrauscht. Diese Anzahlen können nun an das sogenannte Student oder Schüler Modell weitergegeben werden, welches die finale Vorhersage an den Nutzer weitergibt. Die roten Pfeile der Abbildung zeigen dabei den Fluss einer Predction. Für den Nutzer ist nur der rechte Teil der gestrichelten Linie (Student Modell) erreichbar, der linke Teil nicht.

Theoretisch wäre es möglich, die Aggregation der Teacher Modelle direkt an den Nutzer zu geben. Ein Student Modell könnte jedoch in die Hände eines Angreifers gelangen und wäre dennoch resistent gegen White Box Angriffe, wohingegen White Box Angriffe bei den Teacher Modellen problematischer wäre. Zusätzlich bietet Student Modelle weitere Vorteile. So ist es beispielsweise möglich, die Teacher Modelle nur auf sensiblen Daten zu trainieren und nicht sensible (öffentliche) Daten mit in das Student Modell zu übergeben.

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 [43].

Ein typisches Beispiel ist der Homomorphismus zwischen zwei Gruppen $(G, \circ)$ und $(H, *)$. Die Beziehung der beiden Gruppen wird Gruppenhomomorphismus genannt, wenn es eine Funktion $f : G \to H$ gibt, die Elemente der Gruppe G auf die Gruppe H abbildet und dabei für alle Elemente $g_1, g_2 \in G$ gilt [44]:

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

Abbildung 3.6 zeigt eine grafische Darstellung dieses Gruppenhomomorphismus.

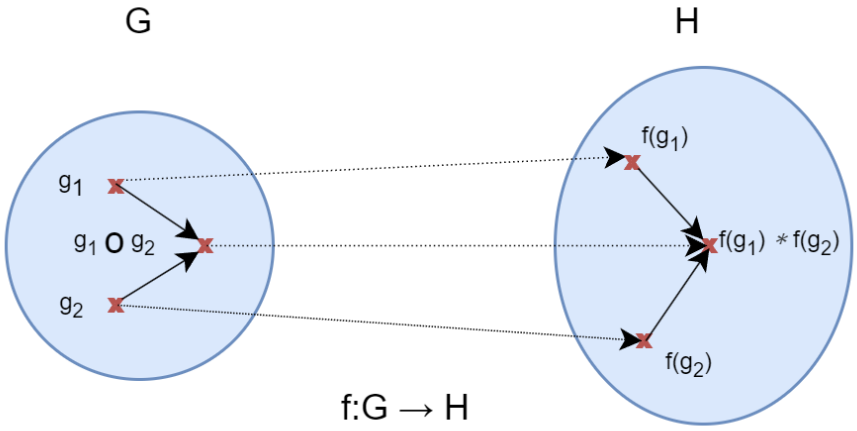

Abbildung 3.6: Gruppenhomomorphismus nach [44]

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 $E_k : P \to C$ und der Ent-

schlüsselungsalgorithmus $D_k : C \to P$ mit einem Schlüssel $k \in K$ [44]. Daraus lässt sich ableiten, dass folgende Bedingungen erfüllt sind:

$$E_k(p_1 \circ p_2) = E_k(p_1) * E_k(p_2) \text{ und } D_k(c_1 * c_2) = D_k(c_1) \circ D_k(c_2)$$

Abbildung 3.7 zeigt, wie besagter Homomorphismus aussieht.

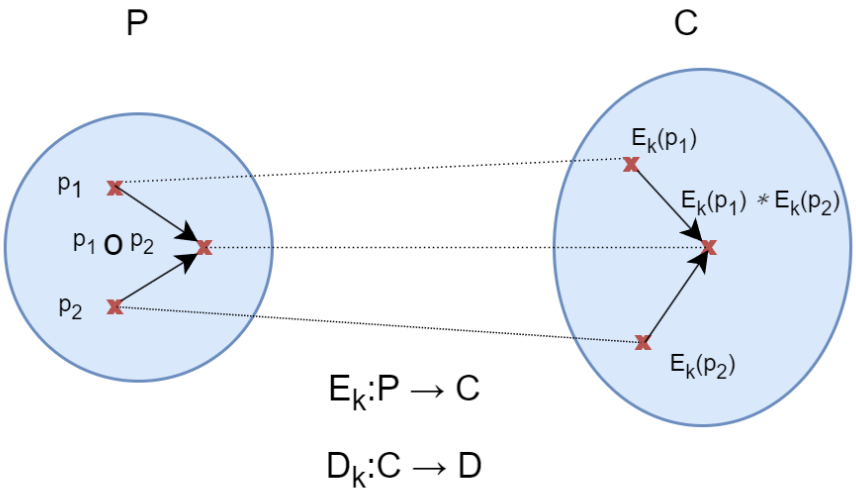

Abbildung 3.7: Homomorphe Verschlüsselung

Homomorphe Verschlüsselungen lassen sich dabei in 3 Kategorien einteilen, je nachdem welche Verknüpfungen innerhalb der Gruppen möglich sind [45]:

- **Teilweise homomorphe Verschlüsselung (partially):** Entweder Multiplikation oder Addition möglich, jedoch nicht beides.
- **Eingeschränkte homomorphe Verschlüsselung (somewhat):** 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 [46] 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 den Verschlüsselungssystems zurückgesetzt und eine weitere Berechnung ist möglich. Kann das ursprünglich eingeschränkte homomorphe Verschlüsselungssystem die homomorphe Entschlüsselung und eine weitere Operation durchführen, dann kann es mittels Bootstrapping zu einem vollständig homomorphen System umgewandelt werden. So tauschten Van Dijk et al. [47] die auf Gittern basierte Verschlüsselung durch eine auf Ganzzahlen basierte, eingeschränkt homomorphe Verschlüsselung aus.

Brakerski und Vaikuntanathan [48] verbesserten die Effizienz des bereits geschilderten Ansatzes. Sie nutzen eine eingeschränkte homomorphe Verschlüsselung auf Basis des Lernen mit Fehlern (Learning with errors) Problems zusätzlich zu einem Relinearisierungsschritt. Dieser zusätzliche Schritt reduziert die Größe des Geheimtextes, wodurch die homomorphe Entschlüsselung beim Bootstrapping vereinfacht wird.

Gentry et al. [49] 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 Konvolutions und insgesamt 7 Schichten, die Trainingszeit auf einer gewöhnlichen CPU, von ungefähr einer Stunde auf ein ganzes Jahr erhöht[50]. Eine Alternative ist es, keine vollständige sondern nur eingeschränkte homomorphe Verschlüsselung zu nutzen. Takabi et al. [51] 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. Dadurch wird das Rauschen zurückgesetzt, ohne dass Bootstrapping nötig ist. Dies ermöglicht es, andere Verschlüsselungssysteme zu nutzen, die performanter sind.

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 3.4.3 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.[52] vorgestellt. Funktionale Verschlüsselung erlaubt es, eine Funktion f eines Klartextes zu berechnen, wobei nur der Geheimtext als Input der Funktion genutzt wird. Dies geschieht in vier Schritten[52]:

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 3.4.3 genauer betrachtet werden. Ein Framework für das Training des Modells mit dem Namen CryptoNN wurde jedoch von Xu et al. [53] 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 3.8 zeigt das CryptoNN Framework.

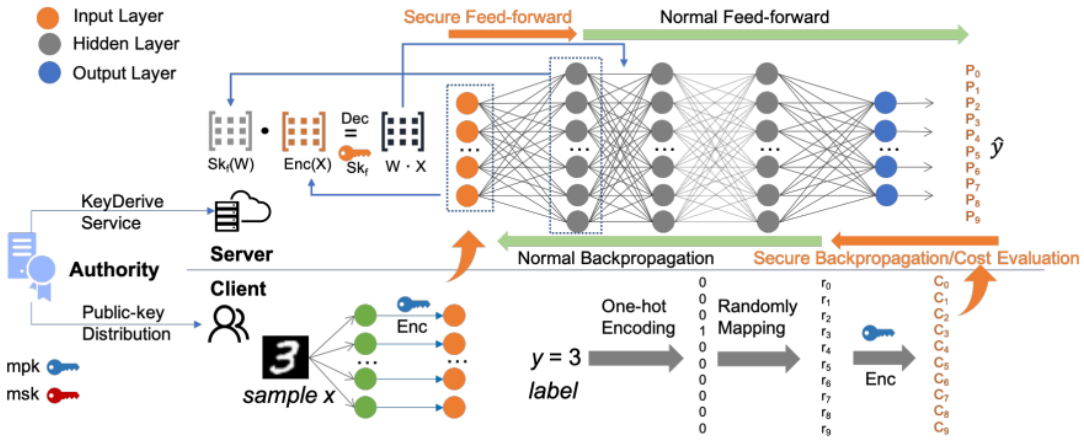

Abbildung 3.8: CryptoNN Framework [53]

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. 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 [54] 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, oder möglichst nach Größe sortiert werden. Alle Teilnehmer wählen die zu teilenden Gradienten mit der gleichen Strategie aus und behalten diese 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. 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 eher Richtung globales Minimum konvergiert. Zwei Parameter steuern dabei die Performance des Modells beim Training mit Distributed Selective SGD: sas Privacy Budget ϵ und die Anzahl der zu teilenden Gradienten. Werden mehr Gradienten nach jedem Schritt von jedem Teilnehmer geteilt, kann das Privacy Budget ϵ niedriger angesetzt werden, um dennoch eine nahezu identische Güte im Vergleich zu einem normal trainierten Modell zu erhalten.

Aggregation

[55]

Kryptografische Methoden

Takabi et al. [51] nutzen Homomorphe Verschlüsselung, um ein Modell zu trainieren, welches Daten von mehreren Teilnehmern nutzen kann. Die Funktionsweise der Methode mit einem Teilnehmer wurde bereits in Kapitel 3.3.3 beschrieben. Diese lässt sich problemlos auf mehrere Teilnehmer erweitern, indem abwechselnd Daten jedes Teilnehmers verschlüsselt an den Server übertragen wird und diese für das Training des Modells mittels Homomorpher Verschlüsselung genutzt wird. Da Daten jeweils verschlüsselt sind, ist es nicht möglich, Daten anderer Teilnehmer zu extrahieren.

Auch das auf Funktionaler Verschlüsselung basierende Framework CryptoNN[53], 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 übernommen werden muss. Anschließend können auch bei dieser Methode abwechselnd Daten von verschiedenen Teilnehmern zum Trainieren genutzt werden.

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. [56] 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 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 wenden 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.

3.4 Betrieb des Modells

Nachdem ein Neuronales Netz trainiert wurde, muss dieses irgendwie für Nutzer zugänglich gemacht werden. So könnte beispielsweise eine API zur Verfügung stehen, über welches ein Modell auf einem Server erreichbar ist. Alternativ kann auch das ganze Modell ausgeliefert werden. Auch hierbei gibt es zusätzlich Möglichkeiten, die Vertraulichkeit der Trainingsdaten zu schützen.

3.4.1 Limitierungen

3.4.2 Differential Privacy der Vorhersage

3.4.3 Kryptografische Inferenz

Wird ein Modell auf einen fremden Server (beispielsweise in einer Cloud) deployed, 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.

Inferenz mittels Homomorpher Verschlüsselung

Inferenz mittels Funktionaler Verschlüsselung

3.4.4 Model Transformation

3.4.5 Model Compression

Kapitel 4

Konzeption PrivacyFlow

4.1 Vergleich zu anderen Frameworks/Lösungen

4.2 Wahl der Methoden

Kapitel 5

Implementierung PrivacyFlow

5.1 Architektur

5.2 Datensatz

5.3 Implementierung der Methoden

5.4 Evaluation + Vergleich mit Vanilla ML Pipeline - Performance & Technik

Kapitel 6

Evaluierung

6.1 Vergleich PrivacyFlow vs. normaler ML-Workflow

6.2 Schutz der Vertraulichkeit

Kapitel 7

Diskussion

7.1 Zusammenfassung der Ergebnisse

7.2 Handlungsempfehlung anhand der Ergebnisse

7.3 Offene Probleme + Ausblick auf zukünftige Forschung

Abbildungsverzeichnis

Tabellenverzeichnis

Auflistungen

Literatur

[1] L. Ouyang, J. Wu, X. Jiang u. a., *Training language models to follow instructions with human feedback*, 2022. arXiv: 2203.02155 [cs.CL].

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

[3] 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, [Online; Abgerufen 08.Mai 2023], März 2018.

[4] 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.

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

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

[7] 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. DOI: 10.1109/SP.2008.33.

[8] 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. DOI: 10.1504/IJSN.2015.071829.

[9] 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. DOI: 10.1145/3243734.3243834.

[10] 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. DOI: 10.1109/ASE.2019.00079.

[11] 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. DOI: 10.1145/2810103.2813677.

[12] 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.

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

[14] Z. He, T. Zhang und R. B. Lee, „Model Inversion Attacks against Collaborative Inference," in *Proceedings of the 35th Annual Computer Security Applications Conference*, Ser. ACSAC '19, San Juan, Puerto Rico, USA: Association for Computing Machinery, 2019, S. 148–162, ISBN: 9781450376280. DOI: 10.1145/3359789.3359824.

[15] R. Shokri, M. Stronati und V. Shmatikov, „Membership Inference Attacks against Machine Learning Models," *CoRR*, Jg. abs/1610.05820, 2016. arXiv: 1610.05820.

[16] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis und F. Tramèr, „Membership Inference Attacks From First Principles," *CoRR*, Jg. abs/2112.03570, 2021. arXiv: 2112.03570.

[17] 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.

[18] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos und D. Song, *The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks*, 2019. arXiv: 1802.08232 [cs.LG].

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

[20] B. Biggio, B. Nelson und P. Laskov, *Poisoning Attacks against Support Vector Machines*, 2013. arXiv: 1206.6389 [cs.LG].

[21] C. Yang, Q. Wu, H. Li und Y. Chen, *Generative Poisoning Attack Method Against Neural Networks*, 2017. arXiv: 1703.01340 [cs.CR].

[22] 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.

[23] F. Tramèr, R. Shokri, A. S. Joaquin u. a., *Truth Serum: Poisoning Machine Learning Models to Reveal Their Secrets*, 2022. arXiv: 2204.00032 [cs.CR].

[24] 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.

[25] 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. DOI: 10.1109/SP.2019.00029.

[26] 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.

[27] B. Zhao, K. R. Mopuri und H. Bilen, „iDLG: Improved Deep Leakage from Gradients," *CoRR*, Jg. abs/2001.02610, 2020. arXiv: 2001.02610.

[28] B. Hitaj, G. Ateniese und F. Pérez-Cruz, „Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning," *CoRR*, Jg. abs/1702.07464, 2017. arXiv: 1702.07464.

[29] 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.

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

[31] 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. DOI: 10.1145/1217299.1217302.

[32] 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. DOI: 10.1109/ICDE.2007.367856.

[33] 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.

[34] 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. DOI: 10.1561/0400000042.

[35] 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.

[36] 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.

[37] L. Xie, K. Lin, S. Wang, F. Wang und J. Zhou, „Differentially Private Generative Adversarial Network," *CoRR*, Jg. abs/1802.06739, 2018. arXiv: 1802.06739.

[38] 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.

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

[40] 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.

[41] 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.

[42] N. Papernot, M. Abadi, Ú. Erlingsson, I. Goodfellow und K. Talwar, *Semi-supervised Knowledge Transfer for Deep Learning from Private Training Data*, 2017. arXiv: 1610.05755 [stat.ML].

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

[44] 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. DOI: 10.1007/978-3-319-12229-8_2.

[45] 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. DOI: 10.1145/3214303.

[46] 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. DOI: 10.1145/1536414.1536440.

[47] 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.

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

[49] 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.

[50] 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. DOI: 10.1007/s12083-021-01076-8.

[51] 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.

[52] 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.

[53] R. Xu, J. B. D. Joshi und C. Li, „CryptoNN: Training Neural Networks over Encrypted Data,“ *CoRR*, Jg. abs/1904.07303, 2019. arXiv: 1904.07303.

[54] 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. DOI: 10.1145/2810103.2813687.

[55] 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. DOI: 10.1145/3133956.3133982.

[56] B. D. Rouhani, M. S. Riazi und F. Koushanfar, „DeepSecure: Scalable Provably-Secure Deep Learning,“ *CoRR*, Jg. abs/1705.08963, 2017. arXiv: 1705.08963.

Glossar

library A suite of reusable code inside of a programming language for software development. i

shell Terminal of a Linux/Unix system for entering commands. i