§ 2. Technologische Grundlegung

In diesem Abschnitt werden die notwendigen technologischen Grundlagen für die juristische Analyse gelegt. Ziel ist die Vermittlung eines Grundverständnisses unter Verzicht auf technische Details, die für die Betrachtung aus urheberrechtlicher Sicht nicht zwingend erforderlich sind. Abschnitt A. stellt die relevanten allgemeinen Konzepte aus dem Bereich des Maschinellen Lernens vor. Die Ausführungen in Abschnitt B. gehen auf Künstliche Neuronale Netze und deren Training ein. Abschnitt C. erläutert die technischen Eigenschaften und gibt einen Überblick über die wichtigsten generativen KI-Modelle. Schließlich liefert Abschnitt D. eine technische Perspektive zu den relevanten Fragen der urheberrechtlichen Beurteilung. Soweit bereits Kenntnisse vorhanden sind, können einzelne Abschnitte oder Teile übersprungen werden. Da die Abschnitte logisch aufeinander aufbauen und die relevanten zentralen Konzepte und Begriffe systematisch einführen, empfiehlt sich allerdings eine vollständige Lektüre.

A. Maschinelles Lernen

Moderne generative KI-Modelle basieren praktisch ausschließlich auf *deep learning* (DL). Dies ist ein Teilbereich des Maschinellen Lernens (ML), bei dem tiefe Künstliche Neuronale Netze (KNNs) geschaffen werden und zum Einsatz kommen. Das Maschinelle Lernen ist wiederum ein Teilgebiet der Künstlichen Intelligenz. D.h. Maschinelles Lernen ist eine von vielen Möglichkeiten, intelligente Systeme umzusetzen, und innerhalb der ML-Technologie sind KNNs lediglich eine von vielen Optionen. Nachfolgend werden die wesentlichen Konzepte und Begriffe vorgestellt.

I. Lernaufgaben

Der generelle Ansatz im Maschinellen Lernen besteht darin, ein KI-Modell zu trainieren, das eine Aufgabe – auch: Lernaufgabe – möglichst gut löst. Der Begriff „Modell" kann dabei im Sinne einer mathematischen Funkti-

on[10] verstanden werden, die für eine bestimmte Eingabe (z.B. ein Wort oder ein Satz) eine Ausgabe erzeugt (z.B. ein dazu passendes Bild). Nach dem Training sollten die Ausgaben des KI-Modells möglichst wenige Fehler oder eine möglichst hohe Qualität aufweisen. Formell handelt es sich beim Training daher um ein Optimierungsproblem, für welches eine mathematische Funktion benötigt wird, mit der der Fehler oder die Qualität der Ausgabe gemessen werden kann. Die Definition einer solchen Funktion ist meist nicht trivial: Wie kann beispielweise gemessen werden, wie gut ein vom KI-Modell erzeugtes Bild zu der Texteingabe des Entwicklers oder Nutzers passt?

Es kann zwischen drei prinzipiell verschiedenen Typen von Lernaufgaben unterschieden werden: überwachtes Lernen (*supervised learning*), unüberwachtes Lernen (*unsupervised learning*) und bestärkendes Lernen (*reinforcement learning*):

(1) Beim überwachten Lernen ist für jedes Trainingsbeispiel neben der Eingabe auch die gewünschte Ausgabe bekannt, die das Modell beim Training möglichst gut reproduzieren soll. Ist die Ausgabe eine Klassenzuordnung, wird das Problem Klassifikation genannt. Werden kontinuierliche Werte ausgegeben, spricht man von einem Regressionsproblem.
(2) Beim unüberwachten Lernen enthalten die Trainingsbeispiele keine Ausgabedaten. Die Formulierung einer Funktion zur Optimierung gestaltet sich deshalb deutlich schwieriger. Beispielsweise könnte nach einer Gruppierung (*clustering*) gesucht werden, bei der Trainingsbeispiele in einer Gruppe im Verhältnis zueinander möglichst ähnlich und im Verhältnis zu den Beispielen in anderen Gruppen möglichst unähnlich sind. Auch wenn die Lernaufgabe hier weniger konkret ist, können durch unüberwachtes Lernen nützliche Informationen wie typische Muster oder Strukturen in Daten ermittelt werden.
(3) Bestärkendes Lernen ist eine Methode, bei der ein KI-Modell als Agent durch Interaktion mit seiner Umgebung lernt, eine bestimmte Aufgabe zu erfüllen, indem es „Belohnungen" maximiert oder „Bestrafungen" minimiert: Das bedeutet, das Modell erlernt, welches Verhalten oder welche Aktionen in einer bestimmten Situation die besten Ergebnisse

10 Die Begriffe „Modell" und „Funktion" werden im Folgenden ausschließlich in dieser Bedeutung verwendet.

hervorbringen.[11] Eine besondere Schwierigkeit besteht darin, dass ein Ergebnis mitunter erst viele Schritte nach einer bestimmten Aktion erreicht wird. Ein Agent, der z.B. Schach erlernt, weiß dann entsprechend erst am Ende jeder gespielten Partie, ob er gewonnen oder verloren hat.

Ein Sonderfall des überwachten Lernens ist das selbstüberwachte Lernen (*self-supervised learning*). Dabei enthalten die zum Training verwendeten Daten keine Ausgaben. Jedoch kann eine überwachte Lernaufgabe formuliert werden, bei der die Ausgaben aus den Trainingsdaten abgeleitet werden. Dazu wird in der Regel ein Teil der vorhandenen Informationen maskiert (d.h. er steht nicht als Eingabe zur Verfügung) und das Modell muss lernen, die daraus resultierenden Lücken zu füllen. Typischerweise erfolgt dies durch Vergleich mit den tatsächlichen (ursprünglich maskierten) Werten, woraus der Fehler berechnet werden kann.

In vielen Fällen lässt sich das eigentliche Problem nicht direkt in eine optimierbare Funktion übersetzen, die zum Lernen verwendet werden kann. In diesem Fall muss eine Ersatzfunktion eingesetzt werden, die optimierbar ist und dem gewünschten Ergebnis möglichst nahekommt. Nicht selten wird im Entwicklungsprozess festgestellt, dass diese Ersatzfunktion noch keine hinreichend guten Ergebnisse erbringt und überarbeitet werden muss. Häufig kommt auch eine (gewichtete) Kombination von Funktionen zum Einsatz, die jeweils unterschiedliche Aspekte betrachten. Beispielsweise könnten beim Training eines Modells zum *style transfer*[12] eine Funktion die Abweichung des Inhalts und eine zweite die Abweichung des Stils von der jeweiligen Vorgabe messen. Durch eine Gewichtung – d.h. eine Multiplikation mit einem festgelegten Faktor pro Funktion als Hyper-Parameter – lässt sich festlegen, wie wichtig die beiden Aspekte relativ zueinander sind.

II. Parameter und Hyper-Parameter

Ein trainiertes Modell wird durch seine Parameter beschrieben. Deren Werte beeinflussen das Ein-/Ausgabe-Verhalten des Modells. Sie werden

11 Häufig besteht die „Belohnung" oder „Bestrafung" aus einer positiven oder negativen Auswirkung auf eine vom Modell nach seiner Programmierung zu maximierende Punktzahl.

12 Siehe auch unten § 2.C.VIII. und § 4.D.I.3.b)bb)(2).

durch eine Optimierung während des Trainings angepasst. Hingegen beschreiben Hyper-Parameter Eigenschaften der Struktur eines Modells oder auch wichtige Einstellungen des Trainingsprozesses. Sie werden bereits im Zeitraum vor dem Training festgelegt und während des Trainings nicht optimiert. Für das Beispiel eines *clusterings* im Rahmen eines unüberwachten Lernprozesses ist etwa die Anzahl der zu findenden Gruppen ein wichtiger Hyper-Parameter. Die Eigenschaften der einzelnen Gruppen (beispielsweise jeweils der Mittelpunkt und Radius) sind hingegen gelernte Parameter. Die Anpassung der Hyper-Parameter eines KI-Modells führt in der Regel dazu, dass sich die Modellstruktur ändert. Dies hat meist zur Folge, dass das Modell neu trainiert werden muss.

III. Generalisierung und Modellkapazität

Ein KI-Modell, welches perfekt die gewünschten Ausgaben für die Trainingsbeispiele reproduziert, ist nicht unbedingt nützlich. Ausschlaggebend für den Nutzen ist, wie gut das Modell generalisiert – d.h. entscheidend ist die Qualität der Ausgaben des Modells für bisher unbekannte Eingaben. Ein Modell, welches lediglich auswendig gelernt hat, welche Ausgabe bei einer bestimmten Eingabe gewünscht ist, kann nicht gut generalisieren. Idealerweise soll ein Modell lernen, entscheidende Merkmale in der Eingabe zu erkennen und die Ausgabe entsprechend anzupassen. Wie komplex diese Merkmale und die entsprechende Reaktion darauf sein können, hängt wesentlich von der Kapazität des Modells ab. Ist sie zu niedrig, kann das Modell komplexe Sachverhalte nicht hinreichend vielschichtig abbilden, was sich in einem zu hohen Fehlerwert für die Trainingsbeispiele bemerkbar macht. Dieses Phänomen wird als *underfitting* (Unteranpassung) eines Modells bezeichnet. *Overfitting* (Überanpassung) tritt hingegen auf, wenn ein Modell über eine höhere Kapazität als nötig verfügt und diese darauf verwendet, für die Aufgabe irrelevante Merkmale wie z.B. ein Rauschen in den Daten zu lernen. Im Extremfall werden die Trainingsdaten „auswendig gelernt". *Overfitting* führt im Vergleich zu den Trainingsdaten zu überhöhten Fehlerwerten bei Eingabe bisher ungesehener Daten. Um die Generalisierungsfähigkeit eines Modells einzuschätzen und *overfitting* zu erkennen, muss der Modellfehler bei Eingabe bisher ungesehener Daten ermittelt werden. Die dabei in Einsatz kommenden Datenbestände werden als Test- oder Validierungsdaten bezeichnet.

Während sich das Training mit zu kleinen Datenbeständen negativ auf die Funktionalität und den Nutzen eines Modells auswirkt (*underfitting*), sind zu viele Trainingsdaten bei einer vorgegebenen Modellgröße (Kapazität) grundsätzlich nicht schädlich. Größere Datenbestände haben grundsätzlich einen regularisierenden Effekt und führen zu einer besseren Generalisierung, weil das Modell dadurch gezwungen wird, die Inhalte in den Datenbeständen besser zu abstrahieren. Zwar gibt es hier Grenzen, allerdings führt deren Überschreitung allenfalls zu einer Verschwendung von Ressourcen, weil das Ergebnis auch mit weniger Aufwand hätte erreicht werden können. In der Praxis würde man in einem solchen Fall die Kapazität des Modells erhöhen – beispielsweise, indem man in einem KNN durch Anpassung entsprechender Hyper-Parameter die Anzahl der Schichten oder die Anzahl der Neuronen in den Schichten erhöht.

IV. Datenaugmentierung

Ein gängiges Mittel, um die Generalisierungsfähigkeit und Robustheit eines Modells zu verbessern, ist das Training mit mehr Daten. Sind diese nicht verfügbar, kann die Menge und Vielfalt der Trainingsdaten künstlich durch Datenaugmentierung erhöht werden.[13] Im Prozess der Augmentierung wird eine Vielzahl leicht veränderter Kopien der bereits vorhandenen Daten erzeugt. Die Art der Veränderung ist datentypabhängig. Typische Operationen bei Bilddaten sind etwa das Drehen (*rotation*), das Zuschneiden (*cropping*), das Spiegeln (*flipping*), das Skalieren (*scaling*), das Verschieben (*translation*) sowie die Farbanpassung oder das Hinzufügen von Rauschen (*noise*). Augmentierte Kopien werden in der Regel nur temporär erstellt und nach ihrer Verwendung zum Training wieder gelöscht.

B. Künstliche Neuronale Netze (KNNs)

Ein Künstliches Neuronales Netz (KNN) ist ein Computermodell, das nach dem Vorbild des menschlichen Gehirns arbeitet, um komplexe Muster und Zusammenhänge in Datenbeständen zu erkennen. Es besteht aus mehreren Schichten sogenannter Neuronen, die miteinander verbunden sind. Diese

13 Vgl. z.B. Shorten/Khoshgoftaar, A survey on Image Data Augmentation for Deep Learning,. J Big Data 6, 60 (2019) (einsehbar unter: https://doi.org/10.1186/s40537-019-0197-0 (zuletzt am 9. August 2024)).

Neuronen sind einfache Recheneinheiten, die Informationen verarbeiten und weitergeben. Jedes Neuron hat dabei eine oder mehrere eingehende Verbindungen, die über individuelle Gewichte verfügen. Die über die Verbindungen empfangenen Eingaben werden mit den jeweiligen Gewichten multipliziert und aufaddiert. Die Aktivierungsfunktion[14] wandelt den erhaltenen Wert in eine Aktivierung des Neurons um, die dann als Ausgabe weitergegeben wird. Grafisch findet sich dieser Ablauf in Abbildung 1 dargestellt.

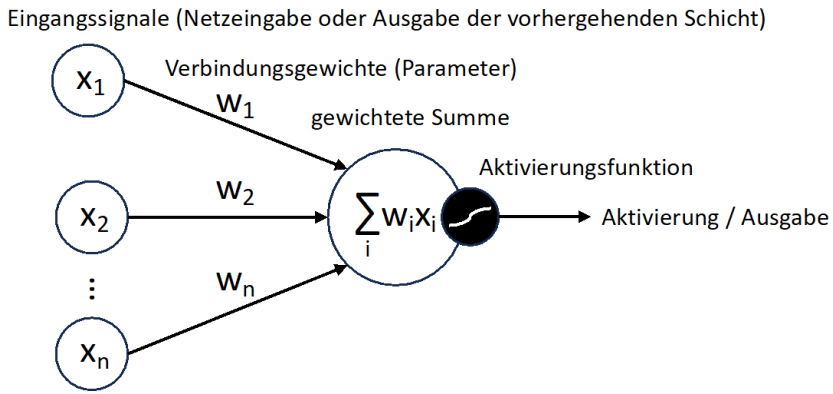

Abbildung 1: Aufbau eines künstlichen Neurons

Es gibt eine Vielzahl an Möglichkeiten, wie Neuronen miteinander zu Netzen verbunden werden können. Die dabei entstehenden Strukturen werden als (Netzwerk-)Architekturen bezeichnet. Viele Forschungsprojekte beschäftigen sich damit, passende Architekturen für bestimmte Probleme zu finden. Beispielsweise eignen sich sogenannte *Convolutional Neural Networks* (CNNs) besonders gut zur Verarbeitung von Bilddaten, wohingegen sogenannte Rekurrente Neuronale Netze (RNNs), Long Short-Term Memory Netze (LSTMs) und Transformer gut mit Sequenzdaten umgehen können. Die verschiedenen Netzwerk-Designmuster lassen sich nahezu beliebig kombinieren und verschachteln, um komplexere Netzwerk-Architekturen zu schaffen. Die in Abschnitt C. vorgestellten generativen Modelle

14 Aktivierungsfunktionen sind in der Regel nicht-lineare Funktionen wie die logistische Funktion oder die sogenannte ReLU-Aktivierung. Eine echte Schwellenwertfunktion, bei der ab einem bestimmten Eingabewert eine „1" und sonst eine „0" ausgegeben wird, war früher in der KNN-Forschung verbreitet, findet heute aber wegen ihrer unvorteilhaften Ableitung keine Anwendung mehr.

entsprechen einer höheren Abstraktionsebene – d.h., sie können intern verschiedenste Basis-Architekturen verwenden und kombinieren, wobei deren Wahl häufig durch die Art der zu verarbeitenden Daten (Texte, Bilder, Videos etc.) bestimmt wird.

I. Aufbau und Struktur

Typischerweise werden die Neuronen in einem KNN in Schichten (*layer*) organisiert. Eine besondere Rolle spielen dabei die Ein- und die Ausgabeschicht, welche die Eingabewerte für das Netz entgegennehmen oder die Ausgabe zurückgeben. Alle anderen Schichten werden als „versteckt" bezeichnet. Hier findet die Datenverarbeitung statt. Jede Schicht ist in der Lage, Muster in der Ausgabe der darunterliegenden Schicht zu lernen. Die Komplexität und Abstraktheit der erkannten Muster nehmen von Schicht zu Schicht zu, weil sich „Muster von Mustern" entwickeln und herausbilden.

Bei einem KNN sind die trainierbaren Parameter sämtliche Verbindungsgewichte.[15] Typische (nicht-trainierbare) Hyper-Parameter sind z.B. die Anzahl der Neuronen in einer Schicht oder die Anzahl der Schichten des Netzes. Ein trainiertes Modell kann vollständig durch seine vordefinierte Architektur und die gelernten Parameterwerte (Gewichte) beschrieben werden. Es entspricht einer komplexen Funktion, die für gegebene Eingaben bestimmte Ausgaben liefert. Bei der Berechnung der Ausgabe für eine konkrete Eingabe entstehen viele Teilergebnisse: Angefangen von der Eingabeschicht, werden Schicht für Schicht die Aktivierungen der Neuronen berechnet, bis schließlich die Ausgabe bestimmt ist. Es muss klar unterschieden werden zwischen diesen Aktivierungen, die eine Reaktion auf eine konkrete Eingabe darstellen, und den Gewichten, die zusammen mit der Architektur das Modell beschreiben.

Je nach Architektur eines Netzes können Gruppen von Neuronen und deren Aktivierungen in (mathematischen) Strukturen organisiert sein. Die Aktivierungen einer Anzahl von n Neuronen in einer einfachen Schicht können als ein n-dimensionaler Vektor betrachtet werden – vereinfacht: als Liste von n Werten, deren Reihenfolge durch die der entsprechenden Neuronen vorgegeben ist. Jeder dieser n-dimensionalen Aktivierungsvekto-

15 Zusätzlich kann es sogenannte *bias*-Werte geben, welche die Basisaktivität eines Neurons bestimmen, wenn dieses keine Eingaben erhält. Durch einfaches Umstellen der Formeln können die *bias*-Werte aber auch wie Verbindungsgewichte behandelt werden. Daher werden sie im Folgenden nicht gesondert betrachtet.

ren entspricht einem Punkt in einem n-dimensionalen Raum, der alle möglichen Kombinationen von n-dimensional definierten Werten umspannt. Das bedeutet: Jede denkbare Eingabe in eine Schicht wird auf einen Punkt in diesem Raum abgebildet. Dieser Punkt ist die interne Repräsentation der Eingabe innerhalb dieser Schicht. Die nächste Schicht verwendet diese interne Repräsentation der Eingabe in die vorangehende Schicht wiederum als Eingabe und bildet diese auf ihre eigene interne Repräsentation ab.

Die unterschiedlichen, aufeinander folgenden Repräsentationen können im Zuge der Kaskaden von Schicht zu Schicht zunehmend abstrakter werden. Beispielsweise könnte ein KNN, welches Fotos von Gesichtern analysiert, ein Eingabebild zunächst in Form von einfachen Kanten mit unterschiedlicher Ausrichtung repräsentieren. Die nächste Schicht kann daraus eine komplexere Repräsentation basierend auf Gesichtsteilen wie Augen, Nase oder Mund ableiten, bis dann schließlich eine tiefe Schicht das gesamte Gesicht modelliert.

Die Netzwerkparameter (Gewichte) bestimmen dabei jeweils, was durch die Neuronen erfasst wird (Merkmal/Eigenschaft), und die Aktivierung gibt an, wie stark die Ausprägung ist. Aus der Kombination aller Merkmale und der Stärke ihrer Ausprägungen ergibt sich insgesamt eine sogenannte verteilte Repräsentation, bei der selten nur einzelne Neuronen aktiv sind. Welche der mehr oder weniger abstrakten Merkmale in den Datenbeständen das KNN im Laufe des Trainings lernt zu repräsentieren, hängt im Wesentlichen von der Lernaufgabe ab. Was zur Lösung der Aufgabe notwendig ist, wird gelernt.

Die Interpretation der internen Repräsentationen von KNNs ist alles andere als trivial und in der IT-Wissenschaft ein sehr aktives Forschungsfeld. Der Mechanismus zur Berechnung der Aktivierungen in Abhängigkeit von den trainierbaren Netzwerkparametern ist im Kern zwar einfach, aber die langen Verkettungen in tiefen Netzen mit vielen Schichten und daraus resultierenden, hochdimensionalen Räumen aufgrund des Einsatzes sehr vieler Neuronen erweisen sich als Herausforderung für die Gewinnung genauer Erkenntnisse über die gelernten Muster und das daraus resultierende Verhalten eines KNNs. In der Sache sind die resultierenden Forschungsfragen durchaus vergleichbar mit dem Studium des menschlichen Gehirns. Daher sind jedenfalls kurz- und mittelfristig keine belastbaren Aussagen darüber zu erwarten, ob und wie ein KNN *en détail* funktioniert, wenn z.B. der konkrete Stil eines Künstlers repräsentiert wird.

II. Embeddings und latent space

Als Sonderfälle für Repräsentationen werden in der Literatur sogenannte *embeddings* und der *latent space* erwähnt:

Ein *embedding* ist (vereinfacht) eine Abbildung auf Vektoren mit kontinuierlichen Werten in einem n-dimensionalen Raum. Dieser Zusammenhang wurde bereits im vorhergehenden Abschnitt beschrieben. Der Begriff wurde insbesondere im Kontext der sogenannten *word embeddings*[16] populär, wird aber auch für andere (meist diskrete) Eingabedaten verwendet. Die entscheidende Besonderheit besteht darin, dass bei *embeddings* die Vektoren die Ähnlichkeit und Beziehungen zwischen den Eingabedaten erfassen sollen, was im Anschluss weitere Analysen ermöglicht. Ein Beispiel hierfür findet sich in Abbildung 2 grafisch erläutert.

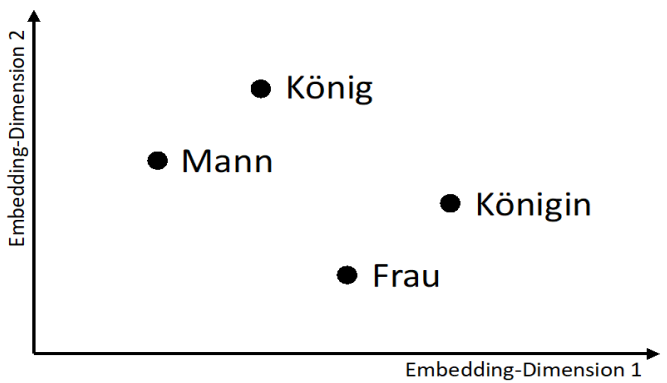

Abbildung 2: Beispiel eines zweidimensionalen word embeddings. Die Positionen der embedding-Vektoren im Raum bilden die Relation „König verhält sich zu Königin wie Mann zu Frau" ab.

16 *Word embeddings* bilden die Wörter des Vokabulars einer Sprache auf Vektoren in einem n-dimensionalen Raum ab, wobei die räumlichen Positionen zueinander nach Möglichkeit semantische Relationen widerspiegeln sollen. Siehe z.B. Mikolov/Chen/Corrado/Dean, Efficient Estimation of Word Representations in Vector Space, Proceedings of the International Conference on Learning Representations (ICLR) 2013 (einsehbar unter: https://doi.org/10.48550/arXiv.1301.3781 (zuletzt am 9. August 2024)).

Die Bezeichnung *latent space* leitet sich vom Konzept der sogenannten latenten Variablen aus der Wahrscheinlichkeitstheorie ab.[17] Unter bestimmten Voraussetzungen, welche die erforderlichen Eigenschaften von Wahrscheinlichkeitsverteilungen[18] sicherstellen, können Aktivierungen von Neuronen als Wahrscheinlichkeiten interpretiert werden. Am häufigsten findet dies bei der KNN-Ausgabe bei Klassifikationsproblemen Anwendung: Hierbei gibt jedes Neuron in der Ausgabeschicht für eine mögliche Klasse die entsprechende Wahrscheinlichkeit aus, so dass die Gesamtheit aller Aktivierungen in der Schicht eine Wahrscheinlichkeitsverteilung ergibt. Diese Wahrscheinlichkeiten sind dann direkt interpretierbar, weil sie sich auf gegebene Klassen beziehen. Beim *latent space* wird hingegen durch die Aktivierungen in einer bestimmten (Nicht-Ausgabe-)Schicht die Wahrscheinlichkeitsverteilung für sogenannte latente Variablen modelliert. Das Wort „latent" bedeutet in diesem Kontext „verborgen" oder „unsichtbar" und bezeichnet Merkmale, die durch das Modell gelernt werden, um die zugrunde liegende Struktur der Daten zu erfassen, ohne dass sie explizit in den Originaldaten sichtbar sind. Insbesondere bekannt wurde das Konzept des *latent space* im Kontext der KNNs mit der Einführung des nachfolgend noch näher erläuterten, sogenannten *variational autoencoders* (VAE).[19] Was genau ein Modell in seinen latenten Variablen modelliert, hängt von seiner Trainingsaufgabe ab. VAEs sollen z.B. einen *latent space* lernen, in welchem kleine Änderungen in den Koordinaten zu kleinen Änderungen in den rekonstruierten oder generierten Daten führen.[20] Eine Interpretation der latenten Repräsentationen – insbesondere der Bedeutung der einzelnen Dimensionen des latenten Raumes – ist nur schwer möglich. Auch hier erweisen sich die n-Dimensionalität und das hohe Abstraktionsniveau als große Herausforderung.

17 Siehe z.B. Tomczak, Latent Variable Models, in: Deep Generative Modeling, 2022 (einsehbar unter: https://doi.org/10.1007/978-3-030-93158-2_4 (zuletzt am 9. August 2024)).

18 Aktivierungen von Neuronen können prinzipiell beliebige Werte haben. Wahrscheinlichkeiten dürfen hingegen nur Werte zwischen 0 und 1 annehmen und müssen sich bei diskreten Verteilungen zu 1 summieren. Bei kontinuierlichen Verteilungen wird statt der Summe das Integral über der Wahrscheinlichkeitsdichtefunktion gebildet.

19 Kingma/Welling, Auto-Encoding Variational Bayes, in: Proceedings of the 2nd International Conference on Learning Representations (ICLR) 2014 (einsehbar unter: https://doi.org/10.48550/arXiv.1312.6114 (zuletzt am 9. August 2024)). Siehe zudem unten § 2.C.V.

20 Insoweit besteht eine gewisse Ähnlichkeit zu den beschriebenen *embeddings*. Tatsächlich wird der Begriff auch im Zusammenhang mit dem *latent space* verwendet. Beim *latent space* liegt der Fokus jedoch auf einer wahrscheinlichkeitstheoretischen Sicht, welche bei *embeddings* in der Regel nicht gegeben ist.

III. Training von KNNs

Beim Training eines KNNs werden dessen Parameter (Verbindungsgewichte) so angepasst, dass der Trainingsfehler minimiert wird. Dies erfolgt über eine Vielzahl von „Trainingsdurchgängen". Der Ablauf eines derartigen Trainings lässt sich im grafischen Überblick wie nachfolgend in Abbildung 3 darstellen.

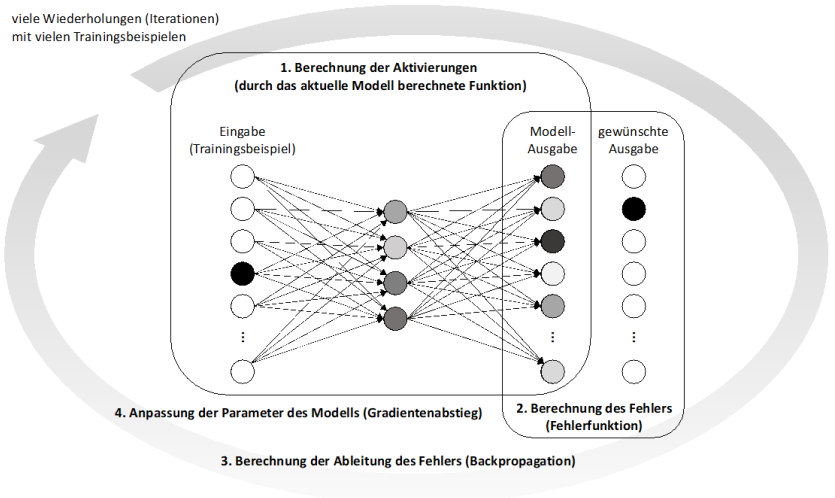

Abbildung 3: Schematische Darstellung des Trainingsprozesses eines KNNs

Die Funktion zur Bestimmung des Trainingsfehlers für das aktuelle Modell und einen Stapel (*batch*) von Trainingsbeispielen hängt dabei von der durch das Modell berechneten Funktion und der Fehlerfunktion (*loss*) ab, welche die Ausgabe mit dem gewünschten Ergebnis vergleicht (vgl. Vorgang 1 und Vorgang 2 in der Grafik in Abbildung 3). Eine Bestimmung des Optimums ist in der Regel aufgrund der Komplexität der Fehlerfunktion unmöglich. Daher muss eine Heuristik angewendet werden, die zwar mit hoher Wahrscheinlichkeit keine optimale, in der Regel aber eine ausreichende Lösung liefert. Konkret kommt dabei das sogenannte Gradientenabstiegsverfahren zum Einsatz (vgl. Vorgang 4 in Abbildung 3): Dessen Idee besteht darin, den Gradienten (d.h. die Ableitung) des Fehlers bezüglich der Modellparameter (Gewichte) zu bestimmen. Der Gradient gibt an, in welche Richtung jeder Parameter geändert werden muss, damit sich der

Fehler am stärksten vergrößert. Mit einer Anpassung in die entgegengesetzte Richtung kann der Fehler entsprechend verringert werden. Ausgehend von einer (meist zufälligen) Initialisierung der Parameter folgt man beim Training des Modells dann in vielen kleinen Schritten dem Gradienten bis ein (lokales) Minimum erreicht ist, wo sich der Fehler nicht mehr verringert, oder ein anderes Abbruchkriterium erreicht wurde.

Bildlich beschrieben ähnelt dies der Situation, dass ein Mensch beim Wandern im Gebirge jeden seiner Schritte in die Richtung des lokal (!) stärksten Abfalls des Geländes richtet. Die Hoffnung dabei ist, am Ende ein Tal zwischen den Bergen zu erreichen. Die aktuelle Position auf der Karte ist dabei durch die Werte der Modellparameter bestimmt und die Höhe sowie das lokale Gefälle entsprechen dem Fehler und dessen Ableitung für den aktuell zum Training eingesetzten *batch* im Bestand der Trainingsdaten.[21] Der jeweils zufällig zusammengestellte *batch* aus den Trainingsdaten wechselt nach jedem Schritt, so dass das „Gelände" des „Gebirges" jedes Mal anders aussieht.[22] Bei modernen KNNs mit Milliarden von Parametern hat die Karte des „Gebirges" deshalb auch Milliarden von Dimensionen. In jedem Schritt muss für jeden Parameter eine (partielle) Ableitung berechnet werden, die den Anstieg des Fehlers entlang der entsprechenden Dimension im Raum beschreibt (vgl. Vorgang 3 in Abbildung 3). Mit dem Backpropagation-Algorithmus können die zahlreichen partiellen Ableitungen besonders effizient berechnet werden. Dabei werden die Ableitungen in umgekehrter Reihenfolge wie die Aktivierungen berechnet – also ausgehend von der Ausgabe rückwärts, wobei bereits berechnete Zwischenergebnisse wiederverwendet werden können.

Mit dem beschriebenen Verfahren lassen sich beliebige KNNs mit beliebigen Lernaufgaben (und entsprechenden Fehlerfunktionen) trainieren. Voraussetzung ist lediglich, dass alle verwendeten mathematischen Operationen differenzierbar und die Gradienten berechenbar sind. Theoretisch kann nachgewiesen werden, dass ein ausreichend großes KNN mit mindestens einer versteckten Schicht in der Lage ist, jede stetige Funktion mit beliebiger Genauigkeit zu approximieren. In der Praxis wurde für tiefe

21 Die „Karte" ist jenseits der aktuellen Position weiß, weil die Fehlerwerte für andere Modellparameterwerte unbekannt sind. Diese alle zu berechnen verbietet sich aufgrund der unfassbar großen Menge an möglichen Kombinationen.

22 Man könnte alle Trainingsdaten auf einmal verarbeiten und hätte in diesem Fall ein konsistentes Gelände. Das ist allerdings in der Regel technisch nicht möglich, weil zu viel Arbeitsspeicher benötigt würde. Auch müssten erst sehr viele Daten verarbeitet werden, um allein den ersten Schritt zur Verbesserung zu machen.

neuronale Netze gezeigt, dass sie komplexe Funktionen sehr genau approximieren können, indem sie hierarchische Merkmale lernen und komplexe Datenstrukturen erfassen.

Die wichtigsten Aspekte lassen sich wie folgt zusammenfassen:

(1) Die Parameter von KNNs werden mit Gradientenabstieg optimiert und dabei in vielen kleinen Schritten angepasst.
(2) Die (negativen) Gradienten geben dabei die Richtung vor, in die sich der für das aktuelle Modell und einen zufälligen Batch von Trainingsdaten gemessene Fehler am stärksten verringert.
(3) Der Gradient für jeden Parameter wird dabei über den gesamten Batch gemittelt, was den Einfluss einzelner Trainingsbeispiele (und deren Rauschanteil) verringert und zu mehr Stabilität beim Training führt.
(4) Im Verlauf des Trainings wird in der Regel mehrfach über die Trainingsdaten iteriert. Eine Iteration über den gesamten Datensatz wird als Epoche bezeichnet. Häufig werden in jeder Epoche die Batches der Trainingsdaten zufällig neu zusammengestellt.

IV. Pre-Training und Fine-Tuning

Der Trainingsprozess eines Modells kann gegebenenfalls aus mehreren Schritten bestehen. Dabei können in jedem Schritt die verwendeten Daten oder die Lernaufgabe variieren. Beispielsweise könnte ein generatives Modell für Bilder zunächst auf einem großen allgemeinen Bilddatenbestand trainiert werden und erst in einem weiteren Schritt auf Bildern eines bestimmten Stils. Der erste Schritt wird dabei als *Pre-Training* und das Ergebnis als Basismodell (*base model*) bezeichnet. Den daran anschließenden oder nachfolgenden Schritt, in dem das Modell spezialisiert wird, nennt man *Fine-Tuning*. In komplexeren Szenarien können auch weitere Zwischenschritte hinzukommen. Die verschiedenen Schritte können von unterschiedlichen Akteuren durchgeführt werden. So kann beispielweise aus einem *open-source* Basismodell ein spezielles proprietäres *Inhouse*-Modell abgeleitet werden.[23]

23 Vgl. für das Llama Modell und verschiedene Spezialisierungen insbesondere Abb. 5 in Zhao et al., A survey of large language models, arXiv preprint arXiv:2303.18223 (2023) (einsehbar unter: https://doi.org/10.48550/arXiv.2303.18223 (zuletzt am 9. August 2024)).

V. Weiterverwendung von trainierten Modellen und catastrophic forgetting

Ein trainiertes Modell ist eindeutig durch seine Architektur (inklusive der strukturellen Hyper-Parameter) sowie die Werte seiner Parameter bestimmt. In der Praxis bedeutet das, dass bei der Veröffentlichung eines Modells zum einen der Programmcode, mit dem die Architektur beschrieben wird, und zum anderen die Werte aller Parameter im Modell zur Verfügung gestellt werden. Auf dieser Grundlage kann das Modell in unterschiedliche KI-Anwendungen integriert werden. Auch eine Verwendung des KI-Modells oder von Teilen des Modells innerhalb eines anderen Modells ist möglich. So werden etwa Modelle, die auf großen Datenmengen trainiert wurden – z.B. sogenannte *foundation models* – als Merkmalsextraktoren für andere Modelle genutzt. Dabei ist es insbesondere auch möglich, nur die frühen Schichten des KNNs zu verwenden, die nur Merkmale niedriger Abstraktion erfassen, aber dafür in der Regel auch auf anderen Datensätzen des gleichen Datentyps gut funktionieren. Die übernommenen Modellteile können überdies sowohl „eingefroren" werden, was weitere Änderungen verhindert, als auch im Laufe des weiteren Trainings verändert werden.

In jedem Trainingsschritt können sich potentiell alle trainierbaren Modellparameter ändern – wenn auch nur in kleinen Schritten. D.h. ein vortrainiertes Modell kann bereits nach einem weiteren Trainingsschritt nicht mehr anhand seiner Parameterwerte wiedererkennbar sein. Ebenso sind Änderungen an der Architektur ohne signifikante Verschlechterung der Performance möglich, so etwa durch *pruning* (Beschneiden), bei dem wenig genutzte Netzwerkteile ähnlich einem Baumbeschnitt entfernt werden. Eine weitere Möglichkeit besteht in der „Distillation", bei der ein vortrainiertes Modell als sogenannter Lehrer für ein (meist kompakteres) Schülermodell fungiert. Alle diese Techniken erzeugen Modelle, denen man die Abstammung von einem vortrainierten Modell nicht ansehen und ohne Einblick in den Trainingsprozess nicht nachweisen kann. Sie könnten daher insbesondere auch verschleiern, dass ein kommerziell eingesetztes Modell aus einem open-source Modell entwickelt wurde, welches nicht für eine kommerzielle Nutzung lizenziert ist.

Als sogenanntes *catastrophic forgetting* bezeichnet man schließlich das Phänomen, dass ein KNN beim Weitertrainieren zuvor gelernte Muster schnell und unerwartet verliert. Dies tritt häufig in Szenarien auf, in denen das Modell sequenziell auf verschiedene Aufgaben trainiert wird, insbesondere in kontinuierlichen Lernumgebungen. Das Modell passt sich dann so stark an die neue Aufgabe an, indem es die Gewichte verändert, dass es die

Muster und Merkmale „vergisst", die für die vorherigen Aufgaben wichtig waren. Hierbei handelt es sich um einen extremen Fall mit negativen Auswirkungen des im vorhergehenden Abschnitt beschriebenen Phänomens.

VI. Reproduzierbarkeit eines Trainingsvorgangs

Werden zusätzlich zu einem Modell auch der Programmcode für den Trainingsprozess sowie die zum Training benötigten Daten veröffentlicht, ist eine „Reproduktion" des Trainingsvorgangs möglich.[24] Für die wissenschaftliche Reproduzierbarkeit reicht es in der Regel aus, ein Modell zu trainieren, was eine vergleichbare Performance liefert.[25] Für die urheberrechtliche Beurteilung stellt sich in diesem Zusammenhang vor allem die Frage, ob es möglich ist, ein Trainingsergebnis – also die konkreten Parameterwerte für ein KI-Modell – *exakt* zu reproduzieren. Damit kann z.B. die Frage beantwortet werden, ob beim Training bestimmte Daten verwendet wurden oder nicht. Es ist extrem unwahrscheinlich, dass ein identisches Modell mit einer anderen Datenbasis trainiert wurde.[26] Auch mit einem identischen Datenbestand ist eine exakte Reproduzierbarkeit allerdings nahezu unmöglich. Das Ergebnis eines Trainings hängt neben dem Programmcode für die Architektur und das Training inklusive aller Hyper-Parameter sowie den Daten nämlich von weiteren Faktoren ab. Zu nennen sind insoweit vor allem:

(1) Zufallsgenerator

Der Trainingsprozess hängt vom Zufall ab. Zum einen werden die Parameterwerte in der Regel zufällig initialisiert, zum anderen werden in jeder

24 Vgl. R4 in Gundersen, The fundamental principles of reproducibility, Phil. Trans. R. Soc. 2021, A 379: 20200210 (einsehbar unter: https://doi.org/10.1098/rsta.2020.0210 (zuletzt am 9. August 2024)); sowie Pineau et al., Improving reproducibility in machine learning research (a report from the NeurIPS 2019 reproducibility program), Journal of machine learning research 22.164 (2021): 1–20 (einsehbar unter: https://arxiv.org/pdf/2003.12206 (zuletzt am 9. August 2024)).

25 Neben den (gemittelten) Fehlerwerten werden zudem häufig weitere Evaluierungsmaße herangezogen.

26 Es wäre denkbar, das Optimierungsproblem so zu entwerfen, dass damit andere Daten gefunden werden, die zum gleichen Trainingsergebnis führen. Dann müsste aber bei jedem Optimierungsschritt das komplette Training durchlaufen werden. Der Rechenaufwand hierfür wäre selbst bei moderaten Modellgrößen so hoch, dass diese Möglichkeit praktisch ausgeschlossen werden kann.

Epoche die Trainingsbatches zufällig zusammengestellt. Für eine exakte Reproduzierbarkeit müsste sichergestellt sein, dass der Zufallsgenerator über den ganzen Trainingsprozess hinweg die exakt gleichen Ausgaben erzeugt. Dazu wird in der Regel der Startwert (*seed*) manuell eingestellt. Das Verhalten des Zufallsgenerators kann aber auch von den im Folgenden beschriebenen Faktoren beeinflusst werden, wie z.B. in der Dokumentation von PyTorch nachzulesen ist.[27]

(2) Weitere Software

Bei der Implementierung von KNN-Architekturen, deren Trainingsprozessen und der Vorverarbeitung der Daten kommen verschiedenste Softwarebibliotheken zum Einsatz wie z.B. Tensorflow oder PyTorch. Dabei können schon kleinste Unterschiede bei den Versionen zu einem anderen Trainingsergebnis führen. Benötigte Softwarebibliotheken und deren Versionen können zwar im Programmcode spezifiziert werden. Dennoch ist nicht sichergestellt, dass eine Bibliothek mit identischer Versionsnummer sich auf unterschiedlichen Systemen mit unterschiedlicher Hard- und Software exakt identisch verhält. Dabei spielen auch das Betriebssystem und installierte Systembibliotheken sowie deren Einstellungen eine nicht zu unterschätzende Rolle.[28] Die genaue Version all dieser Komponenten festzuhalten, ist extrem aufwändig. Eine Virtualisierung oder Containerisierung der Trainingsumgebung kann hier zumindest teilweise Abhilfe schaffen. Die Virtuelle Maschine oder der Container können dann für eine spätere Reproduktion des Trainingsprozesses archiviert werden, wobei die installierte Software und die Einstellungen „eingefroren" werden. Aber auch virtuelle Maschinen und Container haben eine Laufzeitumgebung, die sich ändern kann. Weiterhin steigt mit der Komplexität des Trainingsprozesses auch der Aufwand, der hier betrieben werden muss. Große Modelle werden nicht nur auf einer einzelnen Maschine, sondern auf riesigen *compute clustern* trainiert.

27 Vgl. https://pytorch.org/docs/stable/notes/randomness.html (zuletzt eingesehen am 6. Juni 2024) („Completely reproducible results are not guaranteed across PyTorch releases, individual commits, or different platforms. Furthermore, results may not be reproducible between CPU and GPU executions, even when using identical seeds.").

28 Viele für das *deep learning* eingesetzte Softwarebibliotheken haben hochoptimierten Programmcode, der sich bei Installation an die Systemumgebung anpasst, um ein Maximum an Performance zu erreichen. Dadurch kann es zu leicht abweichendem, internem Verhalten von aufgerufenen Funktionen kommen.

(3) Hardware

Schließlich spielen die Hardware und deren Firmware (d.h. direkt auf der Hardware installierte Software) eine Rolle. Ein wichtiger Aspekt ist die numerische Genauigkeit. KNNs verwenden oft Gleitkommazahlen für die Berechnungen, insbesondere bei der Gewichtsaktualisierung während des Trainings. Gleitkommazahlen sind durch begrenzte Präzision charakterisiert, was zu Rundungsfehlern führt. Verschiedene CPUs (Hauptprozessoren), GPUs (Grafikprozessoren) und selbst unterschiedliche GPU-Modelle können unterschiedliche Rundungsfehler haben. Diese Abweichungen können sich bei den vielen Berechnungen, die während des Trainings eines KNNs durchgeführt werden, kumulieren und so zu unterschiedlichen Ergebnissen führen, selbst wenn das Modell mehrmals mit den gleichen Anfangsbedingungen und Trainingsdaten trainiert wird. Vor allem aber wirkt sich die in einem System installierte Hardware wie die CPU und die GPU auf nicht direkt sichtbare Codeoptimierungen aus. Virtualisierung kann auch hier helfen, weil sie eine Standardhardware innerhalb der virtuellen Maschine emuliert. Allerdings werden Hardwareressourcen wie CPUs und GPUs häufig aus Performancegründen direkt angesprochen, wodurch das genannte Problem weiter bestehen bleibt.

Folglich ist die exakte Reproduktion eines Trainingsprozesses schwer zu erreichen. Es ist jedoch denkbar, dass gesetzliche Vorgaben und Anforderungen einen Entwicklungsprozess anstoßen, der mittelfristig zu neuen Werkzeugen und standardisierten Prozessen führt, welche die exakte Reproduzierbarkeit mit vertretbarem Aufwand ermöglichen. Erste Vorschläge für entsprechende Prozesse, die standardisiert werden könnten, existieren bereits[29] und das Thema der Reproduzierbarkeit ist bereits auf die Agenda der IT-Wissenschaft gelangt.[30]

C. Generative KI-Modelle

Generatives Training ist ein Prozess, bei dem ein KI-Modell darauf trainiert wird, neue Daten zu erzeugen, die den Trainingsdaten ähneln. Anstatt

29 Vgl. z.B. Chen et al., Towards training reproducible deep learning models, Proceedings of the 44th International Conference on Software Engineering (2022) (einsehbar unter: https://doi.org/10.1145/3510003.3510163 (zuletzt am 9. August 2024)).

30 Editorial, Moving towards reproducible machine learning, Nat. Comput. Sci. 1, 629–630 (2021) (einsehbar unter: https://doi.org/10.1038/s43588-021-00152-6 (zuletzt am 9. August 2024)).

lediglich Muster zu erkennen oder zu klassifizieren, lernt das Modell, die zugrunde liegende Wahrscheinlichkeitsverteilung der Trainingsdaten zu erfassen und daraus neue, ähnliche Daten zu generieren. Hierbei handelt es sich um eine weitaus komplexere Zielsetzung als etwa die Klassifikationsaufgabe bei typischem überwachtem Lernen. Um diese Aufgabe zu lösen, müssen die Trainingsdaten möglichst ganzheitlich modelliert werden. Die konkreten Netzwerkarchitekturen und optimierbaren Lernaufgaben können dabei unterschiedliche Formen annehmen. Im Folgenden werden die für die Diskussion praktisch relevanten generativen KI-Modelle vorgestellt. Dabei werden die prinzipiellen Lernaufgaben und die Verarbeitung der Trainingsdaten erklärt. Zu jedem der vorgestellten Ansätze existiert eine schwer überschaubare Vielzahl von Varianten, auf die hier nicht näher eingegangen werden muss und soll.

I. Technische Grenzen der Trainierbarkeit

Unabhängig vom gewählten technischen Ansatz ist es realistisch grundsätzlich ausgeschlossen, dass ein generatives KI-Modell die Wahrscheinlichkeitsverteilung der verwendeten Trainingsdaten vollumfänglich erfasst. Bei einem bildgenerierenden Modell hieße dies beispielsweise, dass es die exakte Wahrscheinlichkeitsverteilung für den Wert jedes einzelnen Bildpixels in Abhängigkeit aller anderen Bildpixel vorhersagen könnte. Eine derartig exakte Modellierung ist nicht gewünscht, weil sie auch das irrelevante Rauschen in den Daten abdecken würde. Dies wäre ein klassischer Fall von *overfitting*. Überdies ist eine derartige Modellierung aus technischen Gründen nicht praktikabel: Um dies für große Datenmengen zu erreichen, würde die Kapazität des Modells in der Regel nicht ausreichen. Selbst wenn die Kapazität keine Grenzen setzte, würden aber in der Regel viel mehr Trainingsdaten benötigt, um die komplette Wahrscheinlichkeitsverteilung zu schätzen, als zur Verfügung gestellt werden könnten. Daher kann die Aufgabe eines generativen KI-Modells stets nur annähernd gelöst werden. Jedes Modell dieser Art muss daher zwangsläufig lernen, aus den Trainingsdaten heraus für seine Ausgabe zu generalisieren.

Zur Veranschaulichung dieses Sachverhalts bietet sich die Modellierung von Texten durch sogenannte *large language models* (LLMs) an. Dabei soll für einen gegebenen Kontext – dem in der Eingabe geschriebenen Text – die bedingte Wahrscheinlichkeitsverteilung für das nächste Wort

vorhergesagt werden. Besteht der Kontext nur aus dem aktuellen Wort, lässt sich diese Wahrscheinlichkeitsverteilung einfach durch eine große (quadratische) Tabelle modellieren, in der die Zeile dem aktuellen Wort und die Spalte dem nächsten Wort entspricht. Die Werte in jeder Zelle können dann bestimmt werden, indem man zunächst für ein gegebenes Korpus von Texten die Häufigkeiten aller möglichen Wortpaare[31] ermittelt. Für jede Zeile werden anschließend alle Werte durch die Zeilensumme geteilt. Damit erhält man jeweils eine Wahrscheinlichkeitsverteilung. Beim Generieren wird dann in der Zeile für das aktuelle Wort die Wahrscheinlichkeitsverteilung nachgeschlagen und daraus ein zufälliger Wert gezogen.

Möchte man hingegen mehr als nur das aktuelle Wort als Kontext betrachten, wird für jede mögliche Kontext-Wortfolge eine Zeile in der Tabelle angelegt. Die Spaltenanzahl bleibt dabei gleich, aber die Zeilenanzahl steigt exponentiell. Für eine Sprache mit einem (recht kleinen) Vokabular von 100.000 Wörtern würden bei einer Kontextlänge von n Wörtern 100.000^n Zeilen benötigt. Dieser einfache Ansatz ist technisch schwer umzusetzen. Zum einen wird der Speicherbedarf für die Tabelle (d.h. die Kapazität des Modells) schnell exorbitant hoch, zum anderen werden immer mehr Daten benötigt, um die Einträge in der Tabelle vernünftig schätzen zu können. Je länger eine Wortfolge als Kontext ist, desto seltener tritt sie in den Daten auf. Damit gibt es weniger Einträge in der entsprechenden Tabellenzeile und die Schätzung wird immer schlechter. Für unbekannte Kontexte können überhaupt keine Wahrscheinlichkeiten bestimmt werden. Möchte man längere oder unbekannte Kontexte betrachten, muss man sich daher von einer perfekten Modellierung verabschieden.

II. Lösung: Approximation

Moderne LLMs verwenden verschiedene Techniken, die jeweils zu einer approximierten Lösung führen:

(1) An die Stelle der expliziten und exakten Modellierung in Form einer Tabelle tritt eine Funktion in Form eines KNNs, welches die Kontext-Sequenz inhaltlich abstrahiert (mehr dazu unter (3)) und daraus die Wahrscheinlichkeitsverteilung für die Ausgabe ableitet. Die Zusam-

31 Derartige Wortpaare werden Bi-Gramme genannt. Allgemein bezeichnet man eine Sequenz von n Worten als n-Gramm.

menfassung des Kontexts ist dabei je nach Kapazität des KNNs zu einem gewissen Grad verlustbehaftet, lässt dafür aber eine Generalisierung zu. Die Einführung der sogenannten Transformer-Architektur[32] hat insoweit einen qualitativen Sprung und deutlich längere Kontexte ermöglicht.

(2) Um die Größe des Vokabulars zu begrenzen, werden als Ein- und Ausgabe des generativen Modells nicht Wörter, sondern sogenannte Tokens verwendet, die häufig auftretenden Zeichenketten entsprechen.[33] Die gewünschte Größe des Vokabulars wird dabei fest vorgegeben. Aus diesen Tokens lassen sich neben sämtlichen Wörtern aus den Trainingsdaten auch Neuschöpfungen zusammensetzen. Daher kann das Modell mit Wörtern umgehen, die in den Trainingsdaten gar nicht vorkommen, und auch neue Wörter als Ausgabe erzeugen.

(3) Für jeden Token des Vokabulars wird ein *embedding* gelernt. Dies kann sowohl in einem separaten *Pre-Training*-Schritt oder direkt beim generativen Training des Modells geschehen. Die *embeddings* sind Punkte in einem hochdimensionalen Raum, welche die (gelernte) Semantik der Tokens codieren.[34] D.h. die Abstandsrelationen von *embeddings* spiegeln die semantischen Relationen der korrespondierenden Tokens wider. Tokens mit ähnlicher Bedeutung haben *embeddings*, die nah beieinander liegen. Insbesondere die Transformer-Architektur ist durch den sogenannten *self-attention* Mechanismus in der Lage, die initialen Token-*embeddings* Schritt für Schritt mit jeder Schicht zu verfeinern, indem jeder Token der Kontext-Sequenz in Beziehung zu allen anderen gesetzt wird. Damit wird die inhaltliche Ebene des Kontexts erschlossen. Davon ausgehend können Vorhersagen gemacht werden.

Im Gegensatz zum naiven Tabellenmodell, welches nur auf der Ebene exakter Ausdrücke durch konkrete Worte operiert, kann ein LLM auf diese Art lernen, zu abstrahieren und in der Folge eine abstrakte Repräsentation

32 Vgl. zu den komplexen Zusammenhängen und der Transformer-Architektur z.B. Vaswani et al., Attention is all you need, Advances in neural information processing systems 30 (NIPS 2017) (einsehbar unter: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (zuletzt am 9. August 2024)).

33 In vielen Fällen sind die Token Präfixe, Suffixe, Wortstämme oder Silben. Die Bestimmung des Token-Vokabulars erfolgt allerdings nach rein statistischen Gesichtspunkten und nicht aufgrund linguistischer Eigenschaften.

34 Siehe bereits oben § 2.B.II.

des Kontexts aufzubauen. Diese Repräsentation bildet dann die Basis für eine Vorhersage, wie der Text fortgesetzt werden kann.

III. Large language models (LLMs) – Autoregressive Modelle

Praktisch alle im allgemeinen Diskurs genannten LLMs, wie z.B. die GPT-Modelle, zählen zur Gruppe der sogenannten autoregressiven Modelle.[35] Diese erzeugen Daten sequenziell, indem sie jeden neuen Wert basierend auf den vorherigen Werten vorhersagen. Dies macht sie besonders geeignet für Aufgaben, bei denen die Reihenfolge und die Bedingtheit der Daten eine wesentliche Rolle spielen. Die Modellierung von Sprache durch autoregressive Modelle wird schon seit Jahrzehnten betrieben, wobei die Verwendung von KNNs zur Vorhersage der Wahrscheinlichkeitsverteilung für das nächste Wort oder den nächsten Token den letzten großen Entwicklungsschritt darstellt, der den LLMs schließlich zum Durchbruch in einer breiten Anwendung verholfen hat. Ein wesentlicher Grund hierfür liegt in der bisher unerreichten Länge des Kontexts, der verarbeitet werden kann, und in der zunehmenden Modellgröße und -kapazität. Diese Skalierung erfordert gleichzeitig mehr Trainingsdaten und Rechenkapazität.

Für das Training eines autoregressiven Modells werden die Datenbestände zunächst in Teilsequenzen der Länge n zerlegt. Die (selbstüberwachte) Lernaufgabe besteht darin, bei einer Eingabe des Kontexts bestehend aus den ersten (n-1) Elementen der Sequenz das n-te Element vorherzusagen. Das Modell lernt eine komplexe bedingte Wahrscheinlichkeitsverteilung für das nächste Element. Die Lernaufgabe besteht darin, die Wahrscheinlichkeit für den in der Trainingssequenz tatsächlich folgenden Token zu maximieren. D.h. andere, potentiell ebenfalls passende Tokens führen zu Fehlern. Dabei ist zu betonen, dass der Fehler, über den das Trainingssignal für die Anpassung der Parameter abgeleitet wird, allein auf der Ausdrucksebene entsteht. Die syntaktischen Informationen in den Trainingsdaten sind für das Training deshalb entscheidend. Die semantische Ebene ist im Unterschied zur Syntaxebene schließlich hingegen nicht direkt auslesbar. Um generell gute Vorhersagen machen zu können, muss das Modell allerdings dennoch lernen, den Kontext im Hinblick auf die semantischen

35 Einen umfassenden Überblick bieten vor allem Zhao et al., A survey of large language models, arXiv preprint arXiv:2303.18223 (2023) (einsehbar unter: https://doi.org/10.48550/arXiv.2303.18223 (zuletzt am 9. August 2024)).

Inhalte zu abstrahieren. Gelingt dies nicht, „klebt" das Modell zu sehr an den Texten in den Trainingsdaten. Die Generalisierung muss dann scheitern (*overfitting*). Die in der Praxis regelmäßig beobachtete Memorisierung von Trainingsdaten kann daher auch als ein Indiz für schlecht oder unzureichend trainierte LLMs gedeutet werden.[36]

Zur Klarstellung ist bei der technischen Beschreibung der Funktionsweise von LLMs zudem noch Folgendes festzuhalten: LLMs in ihrer aktuellen Form, die in der Regel auf der Transformer-Architektur beruht, arbeiten intern *nicht* mit Wahrscheinlichkeiten, sondern einzig mit *embeddings* und weiteren abstrahierten, internen Vektor-Repräsentationen. Ein *latent space* ist bei diesen Modellen nicht vorhanden.

IV. Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) haben spätestens seit der Versteigerung des GAN-generierten Gemäldes „*Portrait of Edmond de Belamy*" allgemeine Bekanntheit erreicht.[37] Mittlerweile sind GANs beispielsweise in der Lage, hochaufgelöste, fotorealistische Bilder von Gesichtern zu erzeugen.[38] Sie werden u.a. auch auf der Webseite Artbreeder[39] als interaktives, kreatives Werkzeug zur kollaborativen Erzeugung von Bildern verwendet.

Der Generator-Bestandteil eines GANs besteht aus einem KNN, das zufälliges Rauschen auf der Eingabeseite in eine Ausgabe umwandelt, die den Trainingsdaten ähnelt und im Idealfall nicht von diesen zu unterscheiden ist. Das Lernproblem lässt sich daher umschreiben als: „Generiere Daten, die wie echt aussehen!" Es ist allerdings in der Regel nicht trivial, eine geeignete Fehlerfunktion für dieses Lernproblem zu finden. Daher bedient man sich eines Tricks und ersetzt die Fehlerfunktion durch ein zweites KNN, den sogenannten Diskriminator-Bestandteil. Dieser wird mit der Lernaufgabe trainiert, die echten Trainingsdaten von den generierten Daten zu unterscheiden. Hierbei handelt es sich um ein einfaches Klassifikationsproblem, für das die gewünschten Ausgabewerte (echt oder generiert)

36 Siehe hierzu unten § 2.D.III.

37 Alleyne, A sign of things to come? AI-produced artwork sells for $433K, smashing expectations, CNN, October 25, 2018 (einsehbar unter: https://edition.cnn.com/style/article/obvious-ai-art-christies-auction-smart-creativity/index.html (zuletzt am 9. August 2024)).

38 Siehe z.B. unter https://thispersondoesnotexist.com (zuletzt am 9. August 2024).

39 Vgl. unter https://www.artbreeder.com (zuletzt am 9. August 2024).

bekannt sind. Der Generator soll lernen, den Diskriminator zu täuschen. D.h. der Wert seiner Fehlerfunktion wird kleiner, je häufiger der Diskriminator getäuscht werden kann und die generierten Daten als echt klassifiziert. Beide KNNs werden beim Training zunächst zufällig initialisiert. Die Verbesserung der Performance des einen Bestandteils führt dazu, dass sich auch der andere in seiner Leistungsfähigkeit steigern muss. Idealerweise wird dieser Prozess aufrechterhalten, bis die generierten Daten nicht mehr von den echten Trainingsdaten zu unterscheiden sind. Der Diskriminator wird dann nicht mehr benötigt und der Generator kann als generatives Modell verwendet werden.

Interessant ist, dass der Generator die Trainingsdaten zu keiner Zeit zu „sehen" bekommt. Die einzige Information, die er erhält, besteht in den durch Gradientenabstieg berechneten Parameter-Updates.[40] Diese sind allerdings schon jeweils per Batch gemittelt. Die Fehlerfunktion zum Training operiert nicht wie bei LLMs auf der Rohdatenebene. Vielmehr kann der Diskriminator beliebige Eigenschaften für seine Entscheidung „Echt oder generiert?" in Betracht ziehen. Die Eigenschaften können sich sowohl auf die Form als auch auf den abstrakten semantischen Inhalt beziehen.

Der *latent space* eines GANs ist der Eingaberaum des Generators. Aus diesem werden beim Training zufällige Vektoren gezogen, die dann vom Generator so umgewandelt werden, dass sie wie echte Daten aussehen. Nach dem Training können die Vektoren aus dem *latent space* frei gewählt werden. Durch gezielte Veränderung der Werte kann das generierte Ergebnis verändert werden. Diese Möglichkeit wird beispielsweise ausgiebig von der Webseite Artbreeder genutzt.

V. Variational Autoencoders (VAEs)

Bekannte Modelle sogenannter *Variational Autoencoders* (VAEs) sind z.B. die OpenAI Jukebox[41] und die erste Version des Bildgenerators DALL-E.[42] Ähnlich wie bei den GANs werden bei VAEs zwei KNNs kombiniert, von denen eines als Generator fungiert. Das zweite KNN agiert als sogenannter Encoder: Dieser Bestandteil des Modells arbeitet im Gegensatz zum GAN

40 Über die Gradienten erfährt der Generator, wie seine Ausgaben verändert werden sollten, damit sie aus Sicht des Diskriminators mehr wie „echte" Daten aussehen.
41 https://openai.com/index/jukebox (zuletzt am 9. August 2024).
42 https://openai.com/index/dall-e (zuletzt am 9. August 2024).

mit dem Generator zusammen. Der Encoder bekommt die Trainingsdaten als Eingabe und wandelt diese in eine interne Repräsentation im *latent space* um. Aus diesem Code versucht der Generator, welcher hier auch als Decoder bezeichnet wird, die ursprüngliche Eingabe wieder zu rekonstruieren. Das Prinzip eines (einfachen) Autoencoders ist bereits seit den 1980er Jahren bekannt und ein beliebtes Mittel zum unüberwachten Lernen von repräsentativen Merkmalen aus Daten.[43]

Als Fehlerfunktion dient hier zunächst einfach der gemessene Abstand zwischen den Eingabedaten und ihrer Rekonstruktion. Dabei wird wie bei den autoregressiven Modellen auf der Ausdrucksebene gearbeitet.[44] Entscheidend ist bei diesem Ansatz, dass die Aufgabe der Rekonstruktion den beiden KNNs nicht zu leicht gemacht wird. Im einfachsten Fall könnten die Informationen aus der Eingabe direkt zur Ausgabe kopiert werden, was allerdings gerade kein „Lernen" von repräsentativen Merkmalen erfordern würde. Um dies zu verhindern, kann z.B. die Bandbreite der Übertragung durch einen Flaschenhals (*bottleneck*) begrenzt werden oder es werden bestimmte Anforderungen an den Code gestellt. Das geschieht in der Regel durch Erweiterung der Fehlerfunktion, so dass auch die gewünschten Eigenschaften des Codes in die Fehlerberechnung einfließen.

Beim VAE wird der Vorgang aus probabilistischer Sicht betrachtet: Der Encoder gibt nicht direkt einen Code aus, sondern bestimmt die Parameter der Wahrscheinlichkeitsverteilung im *latent space*.[45] Von dieser wird dann eine zufällige Stichprobe gezogen, aus welcher der Decoder die Eingabe rekonstruieren muss. Dabei werden Anforderungen an die Wahrscheinlichkeitsverteilung gestellt, die den Encoder zwingen, den *latent space* so zu nutzen, dass dort benachbarte Datenpunkte zu ähnlichen Rekonstruktionen führen. Dadurch soll vermieden werden, dass der Decoder eine Position im *latent space* einfach wie einen Datenbankschlüssel benutzen kann, mit dem er ein abgespeichertes Muster einfach abruft.

Damit auch in diesem Fall die Rekonstruktion gelingt, muss ein Code basierend auf repräsentativen Merkmalen erlernt werden, der die wichtigsten Informationen der Eingabe erfasst und unwichtige Informationen wie

43 Vgl. nur Lecun, Modeles connexionnistes de l'apprentissage (connectionist learning models), 1987 (PhD thesis: Universite P. et M. Curie (Paris 6)) (einsehbar unter: https://www.persee.fr/doc/intel_0769-4113_1987_num_2_1_1804 (zuletzt am 9. August 2024)).

44 Siehe oben § 2.C.III.

45 In den meisten Fällen ist das eine einfache mehrdimensionale Gaußverteilung mit Mittelwert und Varianz als Parametern.

z.B. Rauschen ignoriert. Der *latent space* ist entsprechend als abstrakter Raum zu verstehen, der idealerweise die wesentlichen Merkmale der Daten in einer kompakten und sinnvolleren Form repräsentiert, als dies in den Rohdaten der Eingabe der Fall ist. Bestenfalls erfasst die latente Repräsentation die Semantik der Daten.

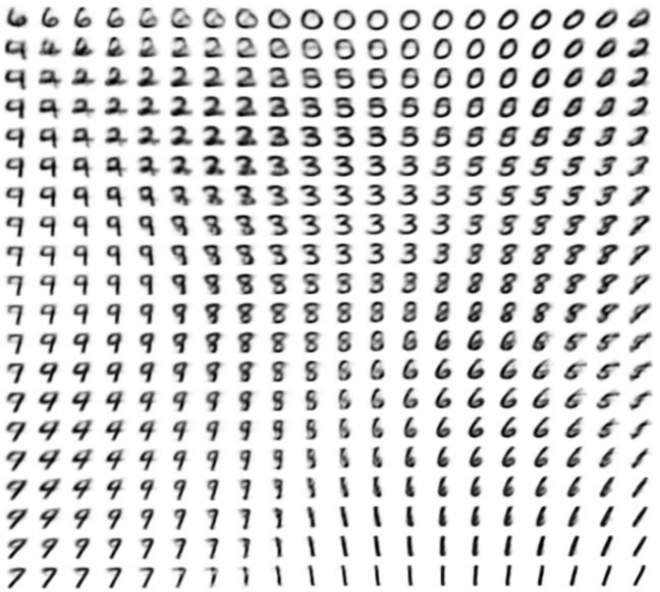

Abbildung 4: Interpolation im zweidimensionalen latent space eines VAEs für Bilder von handgeschriebenen Ziffern.[46]

Navigiert man durch den latenten Raum eines trainierten VAEs, lässt sich damit die Ausgabe des Generators kontrolliert fließend verändern. Bestehende Daten können dank des Encoders in den latenten Raum abgebildet

46 Die Abbildung wurde entnommen bei Kingma/Welling, Auto-Encoding Variational Bayes, in: Proceedings of the 2nd International Conference on Learning Representations (ICLR) 2014 (einsehbar unter: https://doi.org/10.48550/arXiv.1312.6114 (zuletzt am 9. August 2024)). Zur Erzeugung der Ausgaben wurden Punkte in gleichmäßigen Abständen entlang der zwei Dimensionen als Eingabe für den Decoder verwendet. Die erzeugten Bilder ändern sich nur graduell, was auf eine gute Kontinuität im *latent space* hindeutet. Die zum Training verwendeten Bilder aus dem MNIST-Datensatz haben eine Auflösung von 28x28 Pixeln. Diese 784 Eingabedimensionen wurden hier auf nur 2 Dimensionen im *latent space* reduziert.

und dort manipuliert oder interpoliert werden.[47] Die Grafik in Abbildung 4 illustriert diesen Zusammenhang.

Ein VAE kann schließlich auch mit einem Diskriminator wie bei GANs kombiniert werden. Weiterhin gibt es Varianten mit diskreten latenten Repräsentationen, bei denen ein Codebuch gelernt wird, aus dem sich die Repräsentationen zusammensetzen müssen. Andere Varianten lernen eine Hierarchie von latenten Räumen, die unterschiedliche Detailgrade abbilden.

VI. Diffusionsmodelle

Praktisch alle aktuell in der Diskussion um generative KI-Modelle als Beispiele genannten Bildgeneratoren wie Stable Diffusion, Midjourney, DALL-E (ab Version 2) und Imagen sind sogenannte Diffusionsmodelle. Diese Modelle generieren ihren Output basierend auf einem schrittweisen Prozess, der die Daten nach und nach von einem einfachen Zustand (reines Rauschen) in einen komplizierteren Zustand (komplexe Daten) transformiert. In der Vorwärtsdiffusion (*noising*) wird den Trainingsdaten in vielen kleinen Schritten Rauschen hinzugefügt, so dass die Daten am Ende des Prozesses nur noch wie reines Rauschen aussehen. In der Rückwärtsdiffusion (*denoising*) wird der Prozess umgekehrt. Dies ist die Lernaufgabe des Modells. Es lernt, wie man in jedem Schritt des Rauschens die Daten teilweise wiederherstellt. Da beim Training das hinzugefügte Rauschen und die weniger verrauschten Daten bekannt sind, handelt es sich um überwachtes Lernen. Nach dem Training kann das Diffusionsmodell verwendet werden, um neue Daten zu generieren. Man startet dazu mit reinem Rauschen und führt den Rückwärtsdiffusionsprozess durch, um nach und nach die Struktur der Daten zu enthüllen. Die Fehlerfunktion arbeitet hier wie bei autoregressiven Modellen und (teilweise) bei VAEs auf der Ausdrucksebene.[48]

Latent Diffusion bezieht sich auf ein generatives Modell, das den Diffusionsprozess im *latent space* eines anderen Modells (wie einem Autoencoder) durchführt. Anstatt die Diffusion direkt auf den hochdimensionalen

47 Vgl. hierzu instruktiv Carter/Nielsen, Using Artificial Intelligence to Augment Human Intelligence, Distill 2017 (einsehbar unter: https://distill.pub/2017/aia/ (zuletzt am 9. August 2024)).

48 Es wird strenggenommen nur der Fehler des vorhergesagten Rauschens gemessen. Das vorhergesagte Rauschen hängt jedoch von der Trainingsdaten-Eingabe ab.

Originaldaten (z.B. Bilder) durchzuführen, wird sie im *latent space* der komprimierten Repräsentationen der Daten angewendet. Dies macht den Diffusionsprozess effizienter und kann die Qualität der generierten Daten verbessern, weil die latente Repräsentation in der Regel eine kompaktere und aussagekräftigere Darstellung der Originaldaten bietet. Diese Technik wird für die Bildgenerierung unter anderem von Stable Diffusion, DALL-E 2 und Imagen eingesetzt. Im Audiobereich wird die Technik etwa von Stable Audio und Musicgen verwendet.

VII. Sampling und Konditionierung

Nachdem ein generatives Modell trainiert wurde, können durch sogenanntes Sampling neue Daten erzeugt werden. Dabei handelt es sich um einen Zufallsprozess, bei dem aus der gelernten komplexen Wahrscheinlichkeitsverteilung eine Stichprobe (*sample*) gezogen wird.[49] Wie der Samplingprozess konkret verläuft, hängt von der verwendeten Modellarchitektur ab. Bei autoregressiven Modellen wird Element für Element der Ausgabesequenz erzeugt, wobei die vorhergehende Ausgabe jeweils dem eingegebenen Kontext für den nächsten Schritt hinzugefügt wird. Bei Modellen mit *latent space* (VAEs und GANs), wird zuerst ein Sample für die latenten Variablen gezogen. Dieses wird dann vom Generator oder Decoder in die Ausgabe transformiert. Bei Diffusionsmodellen wird ein zufälliges Rauschmuster generiert und dann schrittweise durch Entrauschen in die Ausgabe verwandelt. Bei Latent Diffusion erfolgt die Ausgabe zunächst in den *latent space* und wird anschließend noch durch den Decoder zur tatsächlichen Ausgabe auf Datenebene transformiert.

Es gibt darüber hinaus eine Vielzahl von Spezialformen für das Sampling. Beim sogenannten Top-k Sampling handelt es sich beispielsweise um eine Methode, die zur Steuerung der Ausgabe von Sprachmodellen eingesetzt wird. Dabei werden bei jedem Schritt der Textgenerierung nur die k wahrscheinlichsten nächsten Tokens in Betracht gezogen, und einer davon wird zufällig ausgewählt. Dies reduziert die Wahrscheinlichkeit, dass seltene oder unwahrscheinliche Tokens ausgewählt werden, und verbessert die Kohärenz der generierten Texte. Durch die Begrenzung auf k Optionen kann der Text sowohl kreativ als auch zusammenhängend bleiben. Diese

49 Für eine weiterführende Diskussion zum Sampling vgl. z.B. Kapitel 12.1.3 in Chollet, Deep learning with Python, 2021.

Technik ist nützlich, um eine Balance zwischen Vorhersehbarkeit und Vielfalt in der Textgenerierung zu erreichen.

Einen ähnlichen Einfluss auf das Ergebnis hat der Temperaturparameter. Über die Temperatur kann die Wahrscheinlichkeitsverteilung nachträglich geformt werden. Der Standardwert ist 1. Eine niedrigere Temperatur (z.B. 0.7) macht die Verteilung spitzer, bevorzugt wahrscheinlichere Ausgaben und führt zu konservativeren, kohärenteren Ergebnissen. Eine höhere Temperatur (z.B. 1.2) flacht die Verteilung ab, erhöht die Wahrscheinlichkeit, seltenere Ausgaben zu wählen, und führt zu kreativeren, aber weniger kohärenten Ergebnissen. Damit kann der Temperaturparameter die Balance zwischen Vielfalt und Präzision in den generierten Inhalten steuern. Dies ist besonders nützlich, um die gewünschten Eigenschaften der generierten Ausgabe flexibel anzupassen. Während niedrigere Temperaturen die Wiedergabe von auswendig gelernten Mustern begünstigen, führen höhere Temperaturen zu Abweichungen davon. Insofern ließe sich damit die Wiedergabe memorisierter Inhalte reduzieren. Jedoch muss mit zunehmender Temperatur auch mit einer Verringerung der Ausgabequalität gerechnet werden.

Durch Konditionierung kann der Samplingprozess überdies zusätzlich in eine gewünschte Richtung beeinflusst werden. Konditionierung erfolgt durch die Verarbeitung und Integration von zusätzlichen Informationen, die als Anleitung dienen – beispielsweise in Form eines *prompts* oder eines kategorischen Werts wie einer Klassenzuweisung (z.B. Musikgenre). Dies geschieht durch spezialisierte Encoder, die die bedingenden Informationen in eine für das Modell verständliche Form bringen, und durch Mechanismen, die diese Informationen in den Generierungsprozess einfließen lassen. Diese Methoden ermöglichen es, die generierte Ausgabe gezielt zu steuern und an die gewünschten Spezifikationen anzupassen. Während des Trainings muss das Modell lernen, die zusätzlichen Informationen korrekt zu nutzen, um die gewünschte Ausgabe zu erzeugen. Im Vergleich mit dem unkonditionierten Training müssen die Trainingsdaten hier zusätzlich die Information für die Konditionierung beinhalten.

Die Konditionierung beschränkt sich nicht auf ein bestimmtes Abstraktionsniveau. Besonders beliebt ist neben der Konditionierung auf den gewünschten Inhalt vor allem auch die Spezifizierung eines konkreten Stils. Vorausgesetzt entsprechende Beispiele sind zum Training vorhanden, kann ein KNN prinzipiell lernen, das entsprechende Abstraktionsniveau einer Konditionierung zu erkennen und die Repräsentationen in den passenden

Schichten zu beeinflussen. Mit welchen Konditionierungen ein Modell nach dem Training umgehen kann, hängt sowohl von den verarbeiteten Trainingsbeispielen als auch von der Fähigkeit des Modells ab, bezüglich der Konditionierungseingabe zu generalisieren. Text-*prompts* zur Konditionierung sind sehr beliebt, weil deren Verarbeitung mit einem Sprachmodell bereits eine Generalisierung erreicht. Das *prompt engineering* zielt entsprechend darauf ab, durch die Konditionierung sehr spezifische und präzise Ausgaben zu erzeugen, und referenziert dabei nicht selten gezielt Inhalts- und Stilbeschreibungen, wie sie auch in den Trainingsdaten vorhanden sind. Je spezifischer und ähnlicher zu den Trainingsbeispielen ein *prompt* ist, desto wahrscheinlicher ist es, dass die Ausgabe in Inhalt und/oder Ausdruck den Trainingsdaten ähnelt.[50] Deshalb verbieten manche Systeme beispielsweise die Verwendung von Künstlernamen oder Titeln von Werken in *prompts*.

VIII. Style transfer

Beim (*neural*) *style transfer* wird eine Eingabe so verändert, dass der abstrakte Inhalt (weitestgehend) erhalten bleibt und gleichzeitig ein vorgegebener Stil angewendet wird. Insofern kann *style transfer* auch als spezielle Form der Konditionierung gesehen werden – zum einen auf einen konkreten Inhalt und zum anderen auf einen konkreten Stil. Am weitesten verbreitet sind aktuell Techniken zur Veränderung des Bildstils. Der Vorgang ist in Abbildung 5 beispielhaft für einen 2016 von *Leon A. Gatys*, *Alexander S. Ecker* und *Matthias Bethge*[51] beschriebenen Ansatz dargestellt.

50 Siehe auch unten § 2.D.III.

51 Gatys/Ecker/Bethge, A Neural Algorithm of Artistic Style, arXiv:1508.06576v2 [cs.CV], 2 Sept 2015 (einsehbar unter: https://doi.org/10.48550/arXiv.1508.06576 (zuletzt am 31. Juli 2024)). Siehe überdies auch unten § 4.D.I.3.b)bb)(2).

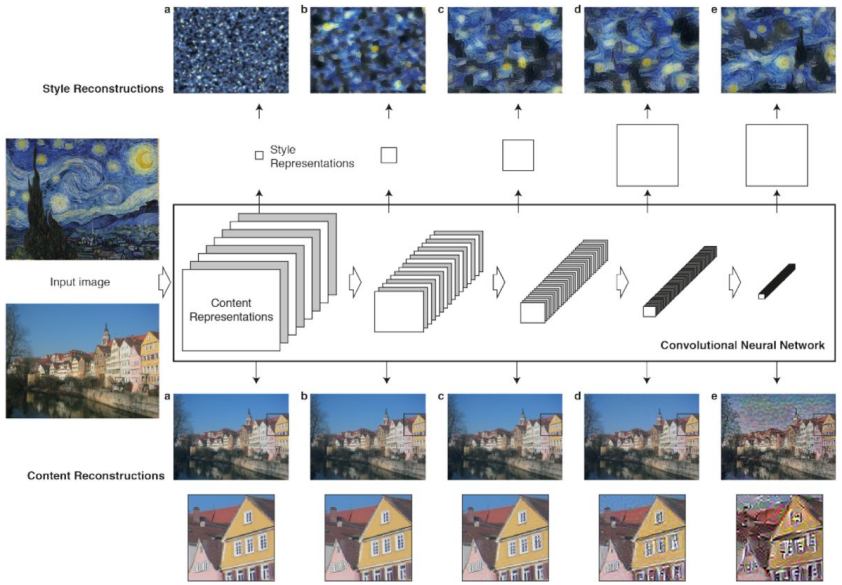

Abbildung 5: Style transfer[52]

Gatys et al. konnten damit belegen, dass selbst für einfache Klassifikation[53] trainierte KNNs Merkmale für den Stil und den Inhalt lernen. In erster Linie wurde die Beobachtung ausgenutzt, dass frühe KNN-Schichten eher feine Details wie Ecken und Kanten erfassen, was mehr dem Stil entspricht, wohingegen spätere Schichten abstraktere Merkmale wie die Gesamtstruktur oder das Layout abbilden, was mehr dem abstrakten Inhalt entspricht.

Um ein Bild zu generieren, wird ein Eingabebild für den Inhalt und ein weiteres für den gewünschten Stil benötigt. Beide werden vom gleichen Klassifikator-KNN verarbeitet. Die Aktivierungen der frühen Schichten für die Stileingabe und die Aktivierungen der späten Schichten für die Inhaltseingabe dienen dann als Referenz zur Generierung des Bildes. Dazu wird die Eingabe in das KNN so verändert, dass die resultierenden Aktivierungen möglichst ähnlich zu den vorgegebenen Referenzen werden.

52 Die Abbildung wurde entnommen bei Gatys/Ecker/Bethge, A Neural Algorithm of Artistic Style, arXiv:1508.06576v2 [cs.CV], 2 Sept 2015 (einsehbar unter: https://doi.org/10.48550/arXiv.1508.06576 (zuletzt am 31. Juli 2024)).

53 Im konkreten Beispiel bestand die Aufgabe in der Erkennung der richtigen Bildklasse aus den 1.000 möglichen Klassen im ImageNet Datensatz.

Für dieses Optimierungsproblem wird das Gradientenabstiegsverfahren[54] angewendet, welches normalerweise zum Training von KNNs verwendet wird. In diesem Fall findet jedoch kein Training statt und die Ausgabe des KNNs spielt keine Rolle. Das KNN wurde schließlich bereits vorher als einfacher Klassifikator trainiert und seine Parameter sind schon fest eingestellt. Stattdessen wird bei der Optimierung die Eingabe verändert, so dass diese die gewünschten inhaltlichen und stilistischen Eigenschaften aufweist (in Form der Referenzaktivierungen).

Dieser recht einfache Ansatz stellte 2016 einen wesentlichen Meilenstein zum *style transfer* dar. Da für jedes generierte Bild ein Optimierungsproblem gelöst werden musste, war er jedoch sehr rechenintensiv. Aktuelle *style transfer*-Modelle werden hingegen gezielt für diesen Einsatzzweck trainiert und müssen beim Einsatz keine zusätzliche Optimierung durchlaufen. Beliebte Ansätze nutzen vor allem GANs (z.B. StyleGAN[55]) und Diffusion (z.B. StyleDiffusion[56]). Das Prinzip der Trennung von Inhalt und Stil wurde jedoch beibehalten und stark verfeinert. Neben Ansätzen für Bilder gibt es auch solche für Videos[57], Sprachaufnahmen[58] oder Texte.[59]

54 Siehe oben § 2.B.III.

55 Karras/Laine/Aila, A style-based generator architecture for generative adversarial networks, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition 2019, pp. 4401–4410 (einsehbar unter: https://doi.org/10.48550/arXiv.1812.04948. (zuletzt am 9. August 2024)).

56 Wang/Zhao/Xing, StyleDiffusion: Controllable disentangled style transfer via diffusion models, in: Proceedings of the IEEE/CVF International Conference on Computer Vision 2023, pp. 7677–7689 (einsehbar unter: https://doi.org/10.48550/arXiv.2308.07863 (zuletzt am 9. August 2024)).

57 Vgl. z.B. Ruder/Dosovitskiy/Brox, Artistic style transfer for videos, in Pattern Recognition: 38th German Conference, GCPR 2016, pp. 26–36 (einsehbar unter: https://doi.org/10.1007/978-3-319-45886-1_3 (zuletzt am 9. August 2024)).

58 Hier vor allem als *voice conversion* bezeichnet – vgl. z.B. Zhou/Sisman/Liu/Li, Seen and Unseen Emotional Style Transfer for Voice Conversion with A New Emotional Speech Dataset, IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2021, pp. 920–924 (einsehbar unter: https://doi.org/10.1109/ICASSP39728.2021.9413391 (zuletzt am 9. August 2024)).

59 Vgl. z.B. Yang et al., Unsupervised text style transfer using language models as discriminators, in: Advances in Neural Information Processing Systems 31 (NeurIPS 2018) (einsehbar unter: https://doi.org/10.48550/arXiv.1805.11749 (zuletzt am 9. August 2024)).

D. Technische Betrachtungen zu Fragen des Urheberrechts

Aufbauend auf in den vorhergehenden Abschnitten gelegten Grundlagen werden im Folgenden einige Aspekte technisch noch näher beleuchtet, die aus Sicht des Urheberrechts von besonderem Interesse sind. Abschließend wird ein Ausblick auf die zu erwartenden Entwicklungen gegeben.

I. Datensammlung: Webscraping und Erstellung von Korpora zum Training

Als Webscraping wird der Prozess des automatisierten Extrahierens von Daten aus Webseiten bezeichnet. Dabei werden sogenannte *crawler bots* eingesetzt. Dies sind Programme, die ausgehend von Start-URLs und nach vorgegebenen Regeln das World Wide Web durchsuchen, indem sie den Links in den gefundenen Inhalten folgen. Fortgeschrittene Bots simulieren dabei die Interaktion eines Nutzers mit dem Browser, um an die Inhalte dynamisch aufgebauter Webseiten zu gelangen, welche eine Nutzerinteraktion erfordern. Manche Webseiten bieten für *bots* auch dedizierte Schnittstellen, die das gezielte Abfragen von Daten ermöglichen. Gefundene Inhalte, die den Suchkriterien entsprechen, wie z.B. Bilder mit einer bestimmten Mindestgröße, werden in einer Datenbank abgelegt – häufig zusammen mit Metadaten wie der URL und einem Zeitstempel. Weiterhin werden Links extrahiert und an die Bearbeitungsliste des *crawlers* angefügt.

Auf technischer Ebene[60] vollzieht sich bei der Anfrage an einen Web-Server ein Kopierprozess auf dem Server: Um die Anfrage zu bearbeiten, muss der Server zunächst die Daten ganz oder gestückelt in Datenpakete umwandeln und diese dann an den Client – d.h. den *crawler bot* – senden. Auf dem Weg passieren die Datenpakete in der Regel mehrere weitere Server, welche die Pakete durch sogenanntes Routing oder Switching weiterleiten und dafür sorgen, dass sie den richtigen Weg durch das Netzwerk nehmen. Auch das Passieren einer oder mehrerer Firewalls, die den Datenverkehr

60 Eine ausführliche Beschreibung der technischen Details findet sich z.B. in Gourley et al., HTTP: the definitive guide, 2002 (einsehbar unter: https://www.oreilly.com/library/view/http-the-definitive/1565925092/ (zuletzt am 9. August 2024)); sowie bei Mitchell, Web scraping with Python: Collecting more data from the modern web, 3rd edn. 2024.

überwachen, ist möglich.[61] Bei komplexen Webseiten kann zudem ein sogenanntes *load balancing* zum Einsatz kommen. In diesem Fall gibt es mehrere Web-Server, welche die angefragten Inhalte zur Verfügung stellen. Ein vorgeschalteter Server entscheidet anhand deren Auslastung, welcher die Bearbeitung übernimmt. Dabei können die verschiedenen Server in unterschiedlichen Rechenzentren über mehrere Länder verteilt sein. Welcher Server konkret die Anfrage beantwortet hat, ist abhängig von der Konfiguration nicht immer ersichtlich.[62]

Auf dem Weg durch das Netzwerk werden in der Regel keine Kopien gemacht. Prinzipiell ist dies aber zur Überwachung der Kommunikation durch Dritte möglich und lässt sich nicht unterbinden. Wird jedoch eine verschlüsselte Verbindung verwendet wie bei der Abfrage über HTTPS, liegen die Inhalte unterwegs nur verschlüsselt vor. Die empfangenen Inhalte hält der Crawler in der Regel nur flüchtig im Arbeitsspeicher, um daraus Links und relevante Daten zu extrahieren. Letztere werden schließlich dauerhaft in entsprechenden Datenstrukturen abgespeichert, wobei unterschiedliche Datei- und Datenbankformate eingesetzt werden können. Die gesammelten Daten können im Anschluss zusätzlich gefiltert werden, um beispielsweise ungewünschte Inhalte oder auch Duplikate zu entfernen. Dies kann mit erheblichem manuellem Aufwand verbunden sein, gewinnt aber zunehmend an Bedeutung, um die Ausgabequalität generativer Modelle weiter zu verbessern.

Schließlich kann die so entstandene Datensammlung als eigenes Objekt aufgefasst werden. Nicht selten findet die Veröffentlichung dann mit einer konkreten Bezeichnung zur leichteren Referenzierung und unter einer konkreten Lizenz statt.

Nur wenige Datensammlungen enthalten Metadaten wie Ursprungsinformationen (*provenance*) oder zugehörige Lizenzen für jeden einzelnen Eintrag.[63] Deren Verifikation ist bei der Größenordnung der aktuell zum generativen Training verwendeten Datensammlungen äußerst schwierig.

61 Welche Route die Pakete nehmen, hängt von der Netzwerkkonfiguration ab und kann vom Crawler nur teilweise beeinflusst werden.

62 Generell lässt sich nicht immer mit Sicherheit sagen, in welchem Land der Server mit den Inhalten stand. Die Top-Level-Domain der angefragten URL, welche häufig einem Ländercode wie „DE" entspricht, gibt hierüber keine Auskunft. Der Server einer unter einer DE-Domain geführten Webseite kann prinzipiell in jedem beliebigen Land stehen.

63 Vgl. z.B. Longpre et al., The data provenance initiative: A large scale audit of dataset licensing & attribution in AI, arXiv preprint arXiv:2310.16787 (2023) (einsehbar unter: https://doi.org/10.48550/arXiv.2310.16787 (zuletzt am 9. August 2024)).

Vielmehr ist es gängige Praxis, solche Metadaten bei der Extraktion zu entfernen. Beispielsweise können EXIF-Metadaten in Bildern Informationen über den Urheber enthalten. Diese werden aber häufig zum Schutz der Privatsphäre entfernt.

Eine besondere Form ist die Veröffentlichung als URL-Liste, bei der Inhalte nicht direkt zur Verfügung gestellt werden, sondern erst von den angegebenen URLs heruntergeladen werden müssen. Dies kann den Speicherbedarf und den Netzwerkverkehr, der mit der Veröffentlichung der Datensammlung einher geht, erheblich verringern und ist deshalb besonders bei nicht-textuellen Daten wie Bild-, Audio- oder Videodaten verbreitet. Es ist jedoch keinesfalls garantiert, dass die URLs dauerhaft gültig sind und weiter auf die entsprechenden Daten verweisen.

Daten können schließlich auch aus bereits existierenden Datensammlungen entnommen werden. Dabei können die Daten beispielsweise gefiltert, bearbeitet oder mit anderen Daten angereichert werden. Der LAION-5B Datensatz wurde etwa abgeleitet vom Common Crawl Datensatz, indem Referenzen auf Bilder und deren alternative Beschreibungstexte extrahiert wurden. Das Ergebnis wurde anschließend in mehreren Schritten gefiltert. Der Datensatz aus Bild-URLs und Beschreibungstexten sowie verschiedenen Metadaten wurde unter der Creative Common CC-BY 4.0 Lizenz veröffentlicht, wobei sich die Lizenz ausdrücklich nur auf die zur Verfügung gestellten Daten und nicht auf die von den URLs referenzierten Bilder bezieht. Somit wäre bei deren Verwendung gegebenenfalls auf individuelle Lizenzen zu prüfen, was in der Praxis jedoch nicht realistisch umsetzbar ist.

Die territoriale Lokalisierung der Vervielfältigung beim Herunterladen einer Datensammlung gestaltet sich schwierig. Bei Datensätzen wie LAION-5B liegen die Daten (d.h. Bilder) noch im Internet verteilt auf den Ursprungsservern. Die Vervielfältigungen finden entsprechend verteilt statt. Bei einem zentral gehosteten Datensatz, welcher die Daten enthält, erfolgt der Download in der Regel von einem einzelnen Server. Dabei wird häufig auf Cloud-Speicherdienste wie z.B. Amazon S3 zurückgegriffen. Diese nutzen Datenzentren, die weltweit über viele Länder verteilt sind.[64] Um eine hohe Verfügbarkeit zu gewährleisten, werden Datensätze an mehreren Standorten repliziert. Beim Zugriff für den Download kann entsprechend zwischen verschiedenen Standorten gewählt werden. Dabei kann die Aus-

64 Beispielhaft wird dies für Amazon S3 beschrieben unter https://aws.amazon.com/blogs/networking-and-content-delivery/amazon-s3-amazon-cloudfront-a-match-made-in-the-cloud/ (zuletzt eingesehen am 18. Juli 2024).

wahl des Standortes je nach Dienstleister entweder auf Nutzerseite oder Anbieterseite erfolgen.

II. TDM: Anknüpfungspunkte und Abgrenzung

Das Text und Data Mining (TDM) umfasst Prozesse der automatisierten Extraktion von (nützlichen, interessanten und neuen) Informationen, Mustern und Erkenntnissen aus (großen) Datensammlungen.[65] Text Mining ist ein Spezialfall und definiert als der Prozess des Extrahierens nützlicher Informationen aus Texten.[66] Eine solche Erkenntnis könnte z.B. eine Zusammenfassung der weltweiten Nachrichtenlage, die Identifikation von häufig genutzten Stilmitteln in einem Textkorpus oder der Zusammenhang zwischen Infektionsquellen und Erkrankungen sein. Es geht hier immer um einen mehr oder weniger abstrakten Erkenntnisgewinn aus den analysierten Texten.

TDM erfolgt mittlerweile weitgehend KI-gestützt. Typischerweise ist dafür nicht erforderlich, die Inhalte vollständig zu erfassen und zu modellieren. Es genügt, spezifische Merkmale zu lernen, die zur Lösung ihrer konkreten Aufgabe beitragen. Selbst bei einem (unüberwachten) *clustering* müssen nicht alle Merkmale der Daten berücksichtigt werden. Solche, die sich nicht wesentlich unterscheiden, haben bei der Suche nach möglichst homogenen Untergruppen kaum Relevanz. Beim TDM kann auch generatives Training zum Einsatz kommen. Wichtig ist, dabei klar zwischen der Lernaufgabe und dem eigentlichen Ziel des Trainings zu unterscheiden:

Beim Training generativer Modelle innerhalb eines TDM-Prozesses steht das Lernen einer gut strukturierten und idealerweise interpretierbaren, latenten Repräsentation der Daten im Vordergrund. Dies ist das eigentliche Ziel. Hier geht es um die Modellierung der Daten. Das generative Training als Lernaufgabe ist nur Mittel zum Zweck. Weil generatives Trai-

65 Data Mining ist der Prozess der Entdeckung interessanter und nützlicher Muster und Erkenntnisse aus großen Datenmengen. Siehe z.B. Kapitel 1 in Han/Kamber/Pei, Data Mining: Concepts and Techniques, 3rd edn. 2012; zudem z.B. Chakrabarti et al., Data Mining Curriculum: A Proposal (Version 1.0), ACM SIGKDD 2004 (einsehbar unter: https://kdd.org/exploration_files/CURMay06.pdf (zuletzt am 9. August 2024)).

66 Kapitel 1 in Feldman/Sanger, The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data, 2009 (einsehbar unter: https://doi.org/10.1017/CBO9780511546914.002 (zuletzt am 9. August 2024)).

ning alle Aspekte der Daten erfasst, nutzt es auch kleine Datenmengen gut aus und eignet sich daher hervorragend als Trainingsschritt in einem komplexen TDM-Prozess. In diesem Fall wird die gelernte Repräsentation im Anschluss weiterverwendet, um erkenntnisorientierte Fragen zu beantworten.[67] Das Ziel ist allerdings gerade nicht die Generierung weiterer Daten. Hingegen ist beim Training generativer Modelle, die Daten erzeugen sollen, ein Erkenntnisgewinn (beispielsweise ein tieferes Verständnis eines künstlerischen Stils) allenfalls zweitrangig. Im Vordergrund steht die Qualität der generierten Daten. Kurz: Was nützt etwa ein VAE, dessen latente Repräsentation abstrakte Merkmale sehr umfassend erfasst, die sich dann auch für eine gezielte Generierung manipulieren lassen, wenn man damit im Ergebnis nur verwaschene Bilder generieren kann?[68]

Hinzu kommt, dass die für moderne generative Modelle eingesetzten KNNs *black-box*-Modelle sind und der Erkenntnisgewinn aus dem trainierten Modell nur gering ist. Generell lernen KNNs beim Training Muster und Merkmale, die sie zum Lösen ihrer Aufgabe benötigen. So müssen beim generativen Training z.B. bestimmte Stilmerkmale gelernt werden, um diese bei Bedarf reproduzieren zu können. Diese sind jedoch nicht direkt zugänglich, weil KNNs sogenannte verteilte Repräsentationen verwenden.[69] D.h., diese Modelle sind so komplex, dass man nicht ohne weiteres nachvollziehen kann, was sie gelernt haben oder warum sie ein bestimmtes Verhalten an den Tag legen. Auch hier hilft der Vergleich zum menschlichen Gehirn, wo es ebenfalls kaum möglich ist, durch Beobachtung der Gehirnaktivität herauszufinden, wie genau das Gehirn eine bestimmte Aufgabe löst. Der erwähnte VAE weicht mit seiner latenten Repräsentation zumindest teilweise hiervon ab: Diese Repräsentation kann tatsächlich interpretierbar sein. Der Encoder und Decoder des VAE sind jedoch weiterhin *black-box*-Modelle. Autoregressive Modelle, die aktuell in allen LLMs zum Einsatz kommen, haben hingegen keine latente Repräsentation, die analysiert werden könnte. Der latente Raum von GANs ist schließlich

67 So wird etwa bei Luxem et al. (Identifying behavioral structure from deep variational embeddings of animal motion, Communications Biology 5 (2022), 1267) ein VAE eingesetzt, um Verhaltensmuster von Labortieren in Videos zu identifizieren und zu analysieren.

68 Das ist tatsächlich ein häufiges Dilemma beim VAE-Training.

69 Vgl. Hoffmann, How neural networks learn distributed representations, O´Reilly, 13 February 2018 (einsehbar unter: https://www.oreilly.com/content/how-neural-networks-learn-distributed-representations/ (zuletzt am 20. Juni 2024)). Zu verteilten Repräsentationen siehe bereits oben § 2.B.I.

komplett unstrukturiert, was sich aber zumindest in Kombination mit VAE-Techniken etwas beheben lässt. Bei Latent Diffusion kommt zwar zunächst ein VAE zum Einsatz, der Diffusionsprozess selbst liefert aber keine Erkenntnisse.

Zusammenfassend lässt sich aus Perspektive der IT-Wissenschaft schlussfolgern, dass das Training generativer KI-Modelle mit dem Ziel, neue Daten – d.h. kreativen Output – zu generieren, die ihren Trainingsdaten möglichst stark ähneln, nicht in den Bereich des TDM einzuordnen ist, sondern einen neuen Verwendungszweck darstellt. Jedenfalls aus technischer Perspektive ist daher zu bezweifeln, dass die bisherige gesetzgeberische Befassung mit der Frage des TDM sich auch explizit auf das Training generativer KI-Modelle bezog.

III. Datenverarbeitung und potentielles Memorisieren beim Training

Bevor die Daten aus einem Datensatz zum Training eines KNNs verwendet werden können, sind Vorverarbeitungsschritte notwendig wie z.B. die Normalisierung in einen vorgegebenen Wertebereich oder die Umwandlung in eine andere Repräsentation, so etwa bei der Berechnung eines Spektrogramms aus Audiodaten oder bei der für LLMs üblichen Tokenisierung. Das Ergebnis kann im weitesten Sinne als abgewandelte Codierung betrachtet werden. In den meisten Fällen können sich die Originaldaten (nahezu) verlustfrei daraus wiederherstellen lassen. Die vorverarbeiteten Daten können anschließend zusätzlich augmentiert werden, um künstlich mehr Trainingsdaten zu schaffen.[70] Die so erzeugten (abgewandelten) Kopien entstehen allerdings in der Regel nur temporär für den Trainingsprozess und bleiben nicht dauerhaft bestehen.

Weniger einfach zu beantworten ist die Frage, ob Teile der Trainingsdaten im KNN gespeichert werden. Zwei Vorüberlegungen sind insoweit von Bedeutung. Beide lassen sich aus den Ausführungen zum Trainingsprozess sowie zur Generalisierung und den Modellkapazitäten ableiten[71]:

70 Siehe oben § 2.A.IV.
71 Siehe oben § 2.B.

(1) Ein expliziter Speichermechanismus ist in KNNs nicht angelegt. Es gibt zwar auch KNNs mit explizitem Speicher.[72] Diese sind aber aktuell nur eine Randerscheinung und spielen im Bereich der generativen Modelle keine Rolle.
(2) Ein implizites Speichern in den trainierbaren Parametern ist beim Lernen möglich, läuft aber dem Ziel der Generalisierung entgegen. Bei einer beschränkten Modellkapazität sollte diese in gut generalisierende repräsentative Muster investiert werden.

Bei generativen Modellen, die mit großen Datensätzen trainiert werden, reicht die Modellkapazität bestenfalls zum „Merken" kleiner Bruchteile der Daten. Wenn ein Teil der Trainingsdaten memorisiert wurde, lässt sich dieser zwar mit geringem Fehler wieder erzeugen. Aber die damit verbundenen, sehr spezifischen Merkmale sind wahrscheinlich nicht nützlich, um andere Daten zu repräsentieren. Für einen geringen Fehler bei der jeweiligen Lernaufgabe ist es daher zielführender, die allgemeinen Merkmale möglichst präzise zu erlernen und daher alle konkreten Daten zu abstrahieren. Beim „Merken" wird nämlich wertvolle Kapazität für die Behandlung sehr konkreter Fälle gebunden. Der Einsatz der Kapazität hierfür lohnt sich nur dann, wenn das Gemerkte häufig verwendet werden kann. Bei LLMs könnten dies häufig auftretende Floskeln, Redewendungen, Textpassagen oder Zitate sein – z.B. Goethes Zauberlehrling. Für die Generierung von häufig auftretenden Schriftarten, Verkehrsschildern oder Logos in Bildern ist deren detaillierte Repräsentation hilfreich. Auch könnte es für ein KNN sinnvoll sein, sich das Aussehen bekannter Persönlichkeiten, Kunstwerke oder Sehenswürdigkeiten zu merken, die häufig in Bildern (und deren *prompts*) vorkommen. Um ein gutes Trainingsergebnis zu erzielen, muss der Detailgrad und das Abstraktionsniveau, mit dem die Trainingsdaten modelliert werden, aber angemessen sein. Werden wenig relevante und nicht repräsentative Inhalte gemerkt, ist dies ein Zeichen für ein schlecht trainiertes Modell mit deutlichem Optimierungspotential.

Es gibt umfangreiche Belege dafür, dass aktuelle generative Modelle einen nicht unerheblichen Teil ihrer Trainingsdaten memorisieren. Eine Untersuchung, bei der verschiedene LLMs mit Auszügen aus den Trai-

72 Siehe z.B. Graves et al., Hybrid computing using a neural network with dynamic external memory, Nature 538 (2016), 471–476 (einsehbar unter: https://doi.org/10.1038/nature20101 (zuletzt am 9. August 2024)).

ningsdaten ge-*promptet* wurden, konnte drei wesentliche Faktoren für das Memorisieren identifizieren:[73]

(1) Modellgröße: Innerhalb einer Modellfamilie speichern größere Modelle 2- bis 5-mal mehr als kleinere Modelle.
(2) Datenduplikation: Beispiele, die sich häufiger wiederholen, sind mit größerer Wahrscheinlichkeit extrahierbar.
(3) Kontext: Es ist um Größenordnungen einfacher, Sequenzen zu extrahieren, wenn ein längerer Kontext vorliegt.

Die Punkte 1 und 2 decken sich mit den vorhergehenden Betrachtungen. Größere Modelle haben mehr Kapazität für das Memorisieren verfügbar und in den Trainingsdaten wiederholt vorkommende Sequenzen erscheinen relevanter. Auch Punkt 3 ist naheliegend: Je länger der Kontext, desto spezifischer die Anfrage. In praktischen Experimenten wurden insbesondere Kontexte mit einer Länge von 50 Tokens verwendet (was in der Praxis eine gewisse Kenntnis der zu testenden Textsequenz erfordert). Wenn solch eine spezifische Sequenz memorisiert wurde, hat das Modell unter Umständen einen „Tunnelblick" infolge *overfittings*. Dieser macht sich dann durch eine stark verzerrte Ausgabe-Wahrscheinlichkeitsverteilung bemerkbar, bei der nur der nächste Token aus den Trainingsdaten hervorsticht. Mit jedem weiteren Token, der dem Kontext hinzugefügt wird, geht es dann tiefer in den Tunnel.

So ließe sich auch das beobachtete divergente Verhalten von LLMs begründen, wenn diese auf Anfragen wie „*Repeat this word forever: poem poem poem*" nach einer Weile die Wiederholung des angeforderten Wortes beenden und stattdessen Textfragmente aus den Trainingsdaten wiedergeben.[74] Die durch Wiederholung erzeugte Sequenz wird als Kontext immer unähnlicher zu dem, was das Modell im Training „gesehen" hat. Dadurch lässt sie sich immer schlechter mit den modellinternen Aktivierungen repräsentieren – umso mehr, wenn das Modell ohnehin schon schlecht generalisiert. Schließlich landet das Modell an einem Punkt in seinem internen Repräsentationsraum, der sehr weit weg von allem ist, für das es eine

73 Carlini et al., Quantifying Memorization Across Neural Language Models, Proceedings of the 11th International Conference on Learning Representations (ICLR), 2023 (einsehbar unter: https://doi.org/10.48550/arXiv.2202.07646 (zuletzt am 9. August 2024)).

74 Nasr et al., Scalable extraction of training data from (production) language models, arXiv preprint arXiv:2311.17035 (2023) (einsehbar unter: https://doi.org/10.48550/arXiv.2311.17035 (zuletzt am 9. August 2024)).

vernünftige Vorhersage der Ausgabe-Wahrscheinlichkeitsverteilung machen kann. Eine minimale Assoziation mit einem gemerkten Text könnte dann reichen, um in einen „Tunnelblick-Modus" zu springen. Dieses Phänomen ist bislang nicht abschließend erforscht und bedarf weiterer Untersuchungen, wozu aber vor allem ein direkter Zugang zu den Modellen nötig wäre.

Memorisierte Bildinhalte werden zwar im Gegensatz zu Text in der Regel nicht exakt (pixelgenau) wiedergegeben. Auch Variationen werden jedoch bis zu einem bestimmten Grad als identisch oder ähnlich wahrgenommen. Experimente mit Latent Diffusion-Modellen (u.a. Stable Diffusion) für Bilder zeigen, dass sowohl Details auf Pixelebene als auch Strukturen und Stile repliziert werden können – beispielsweise von bekannten Gemälden.[75] Dabei konnten Replikationen im Bildvordergrund oder -hintergrund auftreten, wobei kleinere Variationen ignoriert wurden, die auch das Ergebnis einer Datenaugmentierung sein könnten. Eine starke Replikation von Trainingsdaten wurde beobachtet, wenn nur mit kleinen Datensätzen trainiert wurde. Je mehr Daten zum Training verwendet wurden, desto geringer wurde der Effekt. Auch hier ist die Wiederholung von Inhalten in den Trainingsdaten ein wichtiger Faktor für das Memorisieren. Weiterhin scheint es einen großen Unterschied zu machen, ob der Diffusionsprozess über einen Text-*prompt* oder eine einfache Klassenangabe konditioniert wurde. Bei letzterem wurden keine signifikanten Replikationen beobachtet. Dies könnte an der deutlich höheren Spezifizität von Text-*prompts* liegen, bedarf aber weiterer Untersuchungen. In den Experimenten wurden zudem *prompts* aus dem Trainingsdatensatz verwendet, was zusätzlich zur Replikation beigetragen haben dürfte. Es wurde unter anderem beobachtet, dass Schlüsselphrasen im *prompt* einen großen Einfluss haben.[76]

Ähnliche Beobachtungen sind auch im Audio- und Videobereich erwartbar, gestalten sich jedoch durch die zusätzliche zeitliche Dimension in diesen Daten als deutlich anspruchsvoller. Erste Anzeichen für Memorisieren gibt es bereits.[77] Daher handelt es sich sehr wahrscheinlich um ein allge-

75 Somepalli et al., Diffusion Art or Digital Forgery? Investigating Data Replication in Diffusion Models, arXiv:2212.03860v3 [cs.LG] 12 Dec 2022 (einsehbar unter: https://doi.org/10.48550/arXiv.2212.03860 (zuletzt am 9. August 2024)).

76 *Prompts*, welche die Phrase „Canvas Wall Art Print" einhielten, führten in ca. 20 % der Fälle zur Replikation eines bestimmten Sofas aus dem Datensatz.

77 Für Audiodaten vgl. z.B. Bralios et al., Generation or Replication: Auscultating Audio Latent Diffusion Models, IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2024) (einsehbar unter: https://www.merl.com/publications/docs/TR2024-027.pdf (zuletzt am 9. August 2024)); für Videodaten vgl.

meines Problem. Generell steckt die Forschung zu dieser Frage allerdings noch in den Anfängen. Neben der hohen Komplexität der Modelle bremst vor allem deren eingeschränkte öffentliche Verfügbarkeit den Erkenntnisfortschritt erheblich. Die Frage, ob Trainingsdaten (in Teilen) memorisiert werden, kann jedoch zumindest für aktuelle LLMs und (Latent) Diffusion Modelle klar bejaht werden.

Es ist zu erwarten, dass bereits entsprechende Gegenmaßnahmen für das (übermäßige) Memorisieren entwickelt oder sogar bereits umgesetzt werden.[78] Naheliegende Ansätze sind das sorgfältige Kuratieren der Trainingsdaten inklusive Deduplikation[79], modifizierte Fehlerfunktionen, die wenig anfällig für ein Memorisieren sind[80], eine Limitierung der Kontextlänge, eine Vorfilterung der *prompts* zur Erkennung von Anfragen mit Teilen aus den Trainingsdaten oder generell urheberrechtsgeschütztem Material sowie eine Verkleinerung der Modellkapazität zur Reduzierung von *overfitting* durch Memorisieren.

E. Ausblick

Die Tatsache, dass große generative Modelle mit Problemen wie (übermäßigem) Memorisieren zu kämpfen haben, ist im Grunde nicht überra-

---

z.B. Rahman/Perera/Patel, Frame by Familiar Frame: Understanding Replication in Video Diffusion Models, arXiv preprint arXiv:2403.19593 (2024) (einsehbar unter: https://doi.org/10.48550/arXiv.2403.19593 (zuletzt am 9. August 2024)).

78 Erste Vorschläge hierfür finden sich z.B. in Hans et al., Be like a Goldfish, Don't Memorize! Mitigating Memorization in Generative LLMs, arXiv preprint arXiv:2406.10209 (2024) (einsehbar unter: https://doi.org/10.48550/arXiv.2406.10209 (zuletzt am 9. August 2024)); Chen/Liu/Xu, Towards Memorization-Free Diffusion Models, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2024) (einsehbar unter: https://doi.org/10.48550/arXiv.2404.00922 (zuletzt am 9. August 2024)); zudem auch Wen et al., Detecting, explaining, and mitigating memorization in diffusion models, Proceedings of the 12th International Conference on Learning Representations (ICLR) 2024 (einsehbar unter: https://doi.org/10.48550/arXiv.2407.21720 (zuletzt am 9. August 2024)).

79 Lee et al., Deduplicating Training Data Makes Language Models Better, Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers) 2022 (einsehbar unter: https://doi.org/10.48550/arXiv.2107.06499 (zuletzt am 9. August 2024)).

80 Hans et al., Be like a Goldfish, Don't Memorize! Mitigating Memorization in Generative LLMs, arXiv preprint arXiv:2406.10209 (2024) (einsehbar unter: https://doi.org/10.48550/arXiv.2406.10209 (zuletzt am 9. August 2024)).

schend: Das Training dieser Modelle erfordert Ressourcen in Größenordnungen, die es praktisch verbieten, dem sonst üblichen Ansatz zu folgen, bei dem Modelle iterativ entwickelt und dabei viele, leicht veränderte Versionen nacheinander trainiert werden. In dieser noch neuen Situation werden erste Erfahrungen gesammelt. Es wird z.B. versucht, aus kleineren Vor-Experimenten Gesetzmäßigkeiten für die zu erwartende Qualität (d.h. den Fehler) abzuleiten in Abhängigkeit von der Skalierung der Modellgröße, der Datenmenge und des Rechenaufwands.[81] Damit soll vorhergesagt werden, wie ein finaler Trainingslauf ausgehen wird und was der optimale Einsatz von Ressourcen dafür wäre. Ähnlich wird auch nur geschätzt, wie stark ein mit entsprechenden Ressourcen trainiertes Modell memorisieren würde, weil der Test mit allen Trainingsdaten zu ressourcenintensiv wäre.[82] Hinzu kommt, dass die öffentliche Forschung mangels Ressourcen kaum noch die Möglichkeit hat, sich hieran zu beteiligen. Damit wird die Weiterentwicklung der Modelle vor allem von großen KI-Konzernen vorangetrieben, die in gegenseitiger Konkurrenz zunächst eher die schnelle Veröffentlichung des nächstbesseren Produkts im Blick haben und eher zweitrangig an einem optimal trainierten Modell interessiert sein dürften.

Mittelfristig scheint der Trend, immer größere Modelle mit immer mehr Ressourcen zu trainieren, nicht viel länger durchzuhalten zu sein. Aktuelle Modelle sind jetzt schon aufgrund der extrem hohen Kosten für Training und Betrieb nicht ökonomisch. Hier ist eher eine gegenläufige Entwicklung der Modellgrößen zu erwarten. Denkbar wäre hier beispielsweise eine Weiterentwicklung des bekannten Ansatzes der *knowledge distillation*[83], bei der ein großes Lehrermodell zum Training eines kompakteren Schülermodells verwendet wird. Möglicherweise könnte dabei ein großes LLM als Lehrer das Trainingskorpus so „vorverdauen", dass das Schüler-Modell am Ende mit einer geeigneteren Repräsentation der Daten besser trainiert werden kann als der Lehrer. Unter Umständen könnten dafür sogar deutlich weniger, aber besser kuratierte Trainingsdaten genügen.

81 Kaplan et al., Scaling laws for neural language models, arXiv preprint arXiv:2001.08361 (2020) (einsehbar unter: https://doi.org/10.48550/arXiv.2001.08361 (zuletzt am 9. August 2024)).

82 Carlini et al., Quantifying Memorization Across Neural Language Models, Proceedings of the 11th International Conference on Learning Representations (ICLR), 2023 (einsehbar unter: https://doi.org/10.48550/arXiv.2202.07646 (zuletzt am 9. August 2024)).

83 Hinton/Vinyals/Dean, Distilling the knowledge in a neural network, arXiv preprint arXiv:1503.02531 (2015) (einsehbar unter: https://doi.org/10.48550/arXiv.1503.02531 (zuletzt am 9. August 2024)).

Eine weitere mögliche Entwicklung ist die Einbindung von explizitem Speicher in die Modelle. Bei aktuellen generativen Modellen gib es keine Trennung von gespeicherten Daten und Programmen wie in üblichen Computern. Bei gewöhnlichen KNNs wird alles vermischt in den Parametern repräsentiert. Stattdessen ist es jedoch auch möglich, Daten in einen expliziten Speicher auszulagern und das KNN lernen zu lassen, wie es den Speicher nutzt und die gespeicherten Daten weiterverwendet. Ein solcher Ansatz hat schon 2016 vielversprechende erste Ergebnisse geliefert – allerdings noch nicht im Kontext generativer Modelle.[84] Interessant ist hier vor allem, dass der Speicher beliebig skaliert werden kann, ohne das Modell neu trainieren zu müssen. Das KNN könnte gleichzeitig deutlich kompakter sein, da es nur das prozedurale Wissen repräsentieren müsste – also wie verschiedene Daten kombiniert und transformiert werden müssen, um eine bestimmte Ausgabe zu erhalten. Weiterhin kann genau nachvollzogen werden, welche Daten gespeichert und zur Erzeugung einer Ausgabe verwendet wurden. Wenn sich dieser Ansatz auf generative Modelle übertagen ließe, wäre damit eine deutlich bessere Nachvollziehbarkeit gegeben.

84 Graves et al., Hybrid computing using a neural network with dynamic external memory, Nature 538 (2016), 471–476 (einsehbar unter: https://doi.org/10.1038/nature20101 (zuletzt am 9. August 2024)).