GAN - Generative Adversarial Networks

Eine weitere Architektur

Von

Luca Ritz

BFH - Berner Fachhochschule

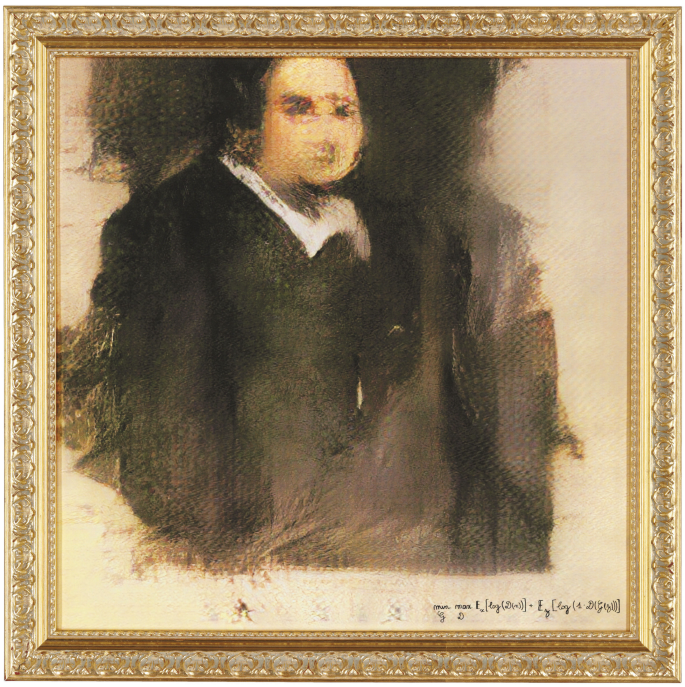

Abbildung 1: Portrait of Edmond Belamy [GAN18]

2021
PUBLISHER

Inhaltsverzeichnis

Kapitel 1

Einführung

Am 25. Oktober 2018 wurde bei Christies das erste Gemälde verkauft, welches durch künstliche Intelligenz erschaffen wurde [GAN18]. Auf etwa 10'000 USD geschätzt, brachte es dem Anbieter doch über 432'500 USD ein. Es zeigt das Porträt einer fiktiven Person namens Edmond Belamy, trainiert mit über 15'000 Bildnissen verschiedener Epochen [Fel18]. Ob dieses „Gemälde" nun als kreatives Werk angesehen werden kann, ist sehr umstritten und wird auch nicht weiter behandelt. Es steht viel mehr der Künstler im Mittelpunkt. Der Künstler namens GAN.

Doch wieso ist so ein GAN überhaupt von Interesse? Leser mit Vorkentnissen im Bereich der künstlichen neuronalen Netzwerke haben wahrscheinlich schon gemerkt, dass es nicht trivial ist, neue Daten aufgrund von Bestehenden zu generieren. Für alle, welche nicht Wissen, was gemeint ist, sei gesagt, dass ein KNN Daten klassifizieren kann, aber nicht unbedingt neue Daten erzeugen. Das Generieren erfordert weitaus mehr. Es soll ein kleines Beispiel behandelt werden, um aufzuzeigen, was gemeint ist. Die Idee zu diesem Gedankenexperiment stammt von „Computerphile" [Com17]. Ein neuronales Netz lernt auf Basis von Input-Daten ein Modell. In diesem Fall gibt es deren drei Input-Daten, welche in der Abbildung 1.1 dargestellt werden. Wie erkannt werden kann, sind diese Datenpunkte sehr zufällig gewählt.

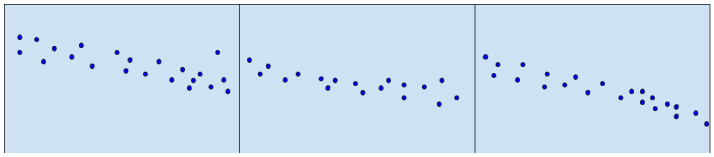

Abbildung 1.1: Input-Daten für KNN

Das Modell, welches das KNN lernt, kann wie in Abbildung 1.2 ersichtlich visualisiert werden. Es bildet eine möglichst gute Annäherung an alle Datenpunkte ab. Doch wie kann nun ein

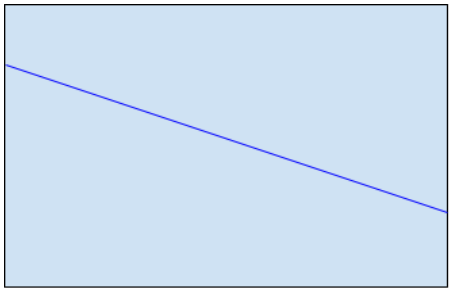

Abbildung 1.2: Gelerntes Modell

neues Sample, welches möglichst mit den Input-Daten korreliert, erzeugt werden? Das einzige, was das Netzwerk wahrscheinlich kann, sind zufällige Datenpunkte auf der Modellgeraden bestimmen. In der Abbildung 1.3 kann entnommen werden, dass dies nicht wirklich natürlich

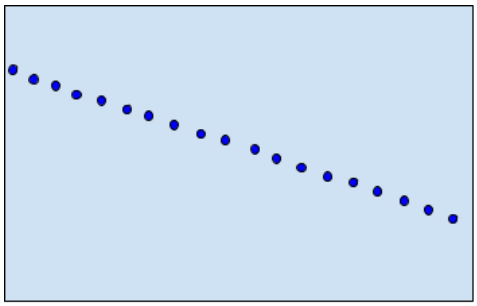

Abbildung 1.3: Generiertes Sample aufgrund von gelerntem Modell

wirkt und nicht mit den Input-Daten übereinstimmt. Die Frage stellt sich nun, wie also solche natürlich wirkenden Samples erzeugt werden können? Eine Antwort darauf liefert GAN. Die vorliegende Arbeit geht der Fragestellung nach, wie diese „Generative Adversarial Networks“ funktionieren.

Kapitel 2

GAN

Ein „Generative Adversarial Net“ besteht aus zwei Teilstücken. Der erste Teil wird „generative model G“ genannt und generiert auf Basis eines originalen Datensets neue Datensamples. Der zweite Teil nennt sich „discriminative model D“ und schätzt die Wahrscheinlichkeit, ob ein solches Sample, welches G generiert, aus dem originalen Datenset stammt. D selbst hat als Output also eine reele Zahl. [Cre+18] In den Worten der Autoren ausgedrückt ist ein GAN ein Spiel zwischen Geldfälschern und Polizisten: «*The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistiguishable from the genuine articles*» [Cre+18]. Anders ausgedrückt versucht G immer besser zu werden, um D möglichst zu überlisten. D hingegen sieht sich einem immer besser werdenden G gegenüber gestellt und muss seine eigenen Schwachstellen ausbessern, um G entgegenzuhalten. Eine Architektur, die so aufgebaut ist, dass sie sich gegenseitig immer weiter verbessert. Aus diesm Grund auch „Generative **Adversarial** Net“ genannt.

Im Endeffekt besteht das Ziel also darin, dass D nicht mehr sagen kann, ob ein Sample von G oder aus dem originalen Datenset stammt. Somit sollte D für alle Samples immer $\frac{1}{2}$ ausgeben. Dies bedeutet, dass ein Sample dieselbe Wahrscheinlichkeit hat aus dem originalen Datenset oder aus dem generierten Set zu stammen und diese beiden Datensets daher nicht mehr unterscheidbar sind. Der Trainingsprozess für G kann weiterhin als Maximierung der Fehlerwahrscheinlichkeit für D angegeben werden. Dies entspricht also einem Zwei-Spieler Minimax-Algorithmus, auf welchen in dieser Arbeit ebenfalls noch eingegangen wird.[Cre+18]

Das generative Modell G sowie das unterscheidende Modell D bestehen jeweils aus zwei Multilayer-Perceptronen. Dies hat den Vorteil, dass die Modelle mittels Backpropagation trainiert werden können [Cre+18].

2.1 Neuronale Netze - Multilayer Perceptron

Da die Architektur des GAN auf zwei neuronalen Netzen beruht, wird in diesem Kapitel kurz darauf eingegangen. Dieser Einschub gibt einen groben Überblick, hat aber nicht die Absicht, ein vertieftes Wissen zu vermitteln.

Die neuronalen Netze sind, wie so manches, über die Zeit herangereift. Die Intention besteht nicht darin, das menschliche Gehirn nachzubauen, sondern eher eine Art Modell für die Informationsverarbeitung zu beschreiben [Unk20b]. Wie in der Biologie versucht man dies in diesen Netzen über Neuronen, wo auch der Name herrührt. Ein solches Neuron bekommt als Eingabe eine Summe von gewichteten Werten, welche es durch eine sogenannte „Aktivierungsfunktion“ in einen Ausgabewert umwandelt.

Am Anfang dieser Netzwerke stand das Perzeptron, welches nur ein einziges künstliches Neuron beinhaltet und von Frank Rosenblatt beschrieben wurde.[Unk20c] Danach hat man damit begonnen, diese Neuronen über Layer miteinander zu verbinden, wobei man den ersten Layer als Input-Layer, die Layer dazwischen als Hidden-Layer und den letzten Layer aus Output-Layer kennt. Der Output-Layer beinhaltet in einem einfachen KNN so viele Neuronen, wie es Klassen gibt. Jedes dieser Output-Neuronen gibt die Wahrscheinlichkeit an, ob eine Eingabe zu der jeweiligen Klasse gehört [Unk20b].

Ein solches neuronales Netz ist eigentlich eine Optimierungsaufgabe. In dem Fall geht es darum, am Ende eine Fehlerfunktion, welche den Abstand des Ist- und Sollzustands beschreibt, zu minimieren. Je geringer der Fehler, desto besser klassifiziert das Netz. Die Variablen, welche dabei angepasst werden können, sind die Gewichte, welche jeweils auf einer Verbindung zwischen zwei Neuronen liegen. Wie wahrscheinlich jeder aufmerksame Leser erkennt, werden dies sehr schnell sehr viele Variablen, weswegen normale Verfahren wie z.B. der Gradientenabstieg zwar eingesetzt werden können, aber sehr ineffizient sind. Der Durchbruch dieser Netzwerke wurde daher erst mit der Beschreibung des Backpropagation-Algorithmus erzielt, welcher ein Spezialfall des Gradientenabstiegverfahrens darstellt und die Gewichte aufgrund ihres Anteils am Fehler anpasst [Unk21a]. Heute gibt es sehr viele verschieden Arten und Architekturen von künstlichen neuronalen Netzen, auf welche nun nicht weiter eingegangen wird.

2.2 Funktionsweise

Formell ausgedrückt besteht ein GAN aus zwei ableitbaren Funktionen $G(z;\theta_g)$ und $D(x,\theta_d)$, welche durch zwei Multilayer-Perceptronen dargestellt werden. Dabei bilden jeweils θ_g sowie θ_d die Parameter der Netzwerke. Um die generierten Daten p_g über ein Sample x aus dem originalen Datenset p_{data} zu trainieren, wird eine Input-Noise-Funktion $p_z(z)$ definiert. $G(z;\theta_g)$ bildet dabei das Mapping zwischen diesen Noise-Daten und dem originalen Datenset p_{data}. Aus dem Noise wird also ein Sample erzeugt, welches möglichst mit den Samples aus p_{data} korreliert. $D(x)$ wiederum gibt einen einzelnen Skalar aus, der die Wahrscheinlichkeit beschreibt, ob ein Sample x eher aus dem originalen Datenset p_{data} als aus dem generierten Datenset p_g (beinhaltet alle Samples von G) stammt. D wird nun so trainiert, um die Wahrscheinlichkeit zu maximieren, das korrekte Label einem Input-Sample x zu geben. Dabei gibt es zwei Labels: „Stammt aus original Datenset“ und „Stammt aus generiertem Datenset“. Simultan dazu wird G trainiert, um die Funktion $\log(1 - D(G(z)))$ zu minimieren[Cre+18, p. 1]. In Worten: $G(z)$ generiert ein Sample aus den Noise-Daten. Davon wird die Wahrscheinlichkeit berechnet, ob dieses Sample aus dem originalen Datenset stammt. Wird dieser Wert gross, das heisst, D hat das Gefühl, dass das generierte Sample aus dem originalen Datenset stammt, so geht der Wert innerhalb des Logarithmus gegen 0 und damit gegen $-\infty$. Dadurch wird also

über diese Minimierung erzielt, dass G möglichst Daten generiert, die D nicht dem generierten Datenset sondern dem originalen Datenset selbst zuweist. Die Autoren verweisen hier auf die Tatsache, dass die beiden neuronalen Netze D und G ein Zwei-Spieler minimax-Spiel mit der folgenden Value-Funktion $V(G,D)$ spielen [Cre+18, p. 2]:

$$\min_G \max_D V(D,G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.1}$$

Diese Value-Funktion definiert die Fehlerfunktion von D. Nachfolgend werden einige Gedanken des Autors dazu aufgeführt.
Fall 1 - Discriminator macht alles richtig:

$$D(x) = 1, D(G(z)) = 0 \Rightarrow \log(1) + \log(1-0) = \log(1) + \log(1) = 0 \tag{2.2}$$

Fall 2 - Discriminator weist $G(z)$ dem Datenset zu:

$$D(x) = 1, D(G(z)) = 1 \Rightarrow \log(1) + \log(1-1) = \log(1) + \log(0) = -\infty \tag{2.3}$$

Fall 3 - Discriminator kann nicht mehr unterscheiden:

$$D(x) = \frac{1}{2}, D(G(z)) = \frac{1}{2} \Rightarrow \log(\frac{1}{2}) + \log(1 - \frac{1}{2}) = \log(\frac{1}{2}) + \log(\frac{1}{2}) = -\log(4) = -1.3862 \tag{2.4}$$

Nach den Autoren bildet der Fall 3 das globale Minimum, welches eben nur erzielt werden kann, wenn der Discriminator nicht mehr zwischen den beiden Datensets unterscheiden kann [Cre+18, p. 4-5]. Es gilt: $p_{data} = p_g$

Nun stellt sich noch die Frage, wie denn genau G lernt. Nach „Computerphile“ erhält G Informationen von D. G kennt also die Schwächen von D und kann diese besser ausnützen. Dies geschieht über die Gradienten von D. [Com17]
TODO: Mapping besser von Noise z nach x darstellen

2.2.1 Einschub - Minimax

Um zu verstehen, wieso es sich hierbei um einen solchen Minimax-Algorithmus handelt, wird in diesem Abschnitt ebendieser mit Verweis auf die Gleichung 1 eingeführt. Der Algorithmus geht auf den Versuch zurück, für ein Zwei-Spieler-Spiel eine schlaue KI zu erschaffen, die möglichst weit in die Zukunft sehen und abschätzen kann, was der Gegener tun wird. Dabei geht man davon aus, dass der Gegner immer so agiert, damit er seine Gewinnwahrscheinlichkeit maximiert und die der KI minimiert.[Unk19] Die Züge, welche gemacht werden können, werden über einen sogenannten Game-Tree dargestellt. In Abbildung 2.1 ist der Game-Tree für ein GAN gegeben. Jeder Layer stellt einen Zug in diesem Spiel dar. Innerhalb eines Zu-

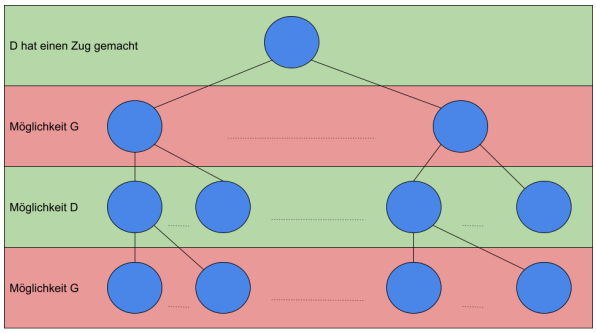

Abbildung 2.1: Leerer GAN-Game-Tree

ges gibt es viele Möglichkeiten. Jede dieser Möglichkeiten wird über einen Node dargestellt. Für ein GAN sind dies alle möglichen Zustände der Variablen, des neuronalen Netzwerks. Der Branching-Faktor ist also sehr gross. In der Gleichung 1 geht es um die Maximierung für D, weswegen die grünen Layer zu D gehören. In diesen Zügen soll der Algorithmus die Value-Funktion $V(G,D)$ für einen Node aus seinen Nachfolgern maximal wählen. Hingegen soll G minimiert werden, weswegen die roten Layer zu G gehören. Innerhalb dieser Layer wird der minimale Wert der nachfolge States bevorzugt. Ist eine gewisse Tiefe erreicht worden, oder es gibt keine Nachfolge-States mehr, dann wird die Value-Funktion $V(G,D)$ für diese States aufgerufen und für den jeweiligen Status der Wert berechnet. Abbildung 2.2 zeigt einen möglichen ausgefüllten Game-Tree über drei Stufen hinweg. Der rote Pfad bildet nach aktuellem Kent-

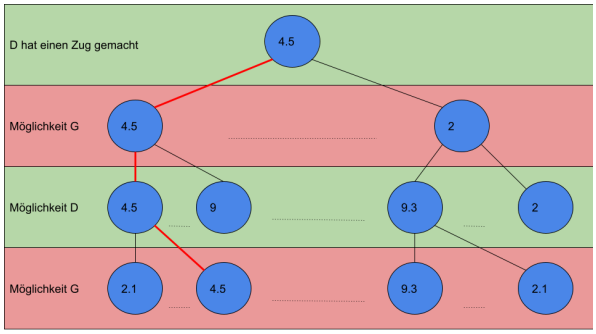

Abbildung 2.2: Berechneter GAN-Game-Tree

nisstand der beste Weg für D, und wird diesen wählen unter der Annahme, dass G wiederum

den Weg wählen wird, welcher für G am besten ist.

2.3 Resultate

Beschreibe einige Ergebnisse auch aus dem Abstract.

2.4 Im Vergleich - Vor- und Nachteile

Weitere möglichkeiten, Samples aus Daten zu generieren aufzeigen und evtl. Vergleich zu GAN.

Kapitel 3

Fazit

Nicht wirklich abschliessendes Urteil, aber Meinung des Autors, ob kreativ oder nicht. Vielleicht kann dies auch gar nicht so pauschal gesagt werden.

Kapitel 4

Anhang

Diese Kapitel fällt wahrscheinlich weg.

Abbildungsverzeichnis

Literatur

[Com17] Computerphile. *Generative Adversarial Networks (GANs) - Computerphile*. [Online; accessed 28.02.2021]. 2017. URL: https://www.youtube.com/watch?v=Sw9r8CL98N0.

[Cre+18] A. Creswell u. a. «Generative Adversarial Networks: An Overview». In: *IEEE Signal Processing Magazine* 35.1 (2018), S. 1–3. DOI: 10.1109/MSP.2017.2765202.

[Fel18] Felix Philipp Ingold. *Kunst oder bloss technischer Schmiereffekt? Das Porträt von Edmond Belamy – geschaffen von künstlicher Intelligenz*. [Online; accessed 26.02.2021]. 2018. URL: https://www.nzz.ch/feuilleton/kuenstliche-intelligenz-und-statt-autorschaft-ld.1435762.

[GAN18] GAN. *Portrait of Edmond Belamy*. [Online; Zugriff 24.02.2021]. 2018. URL: https://www.christies.com/lot/lot-edmond-de-belamy-from-la-famille-de-6166184/?from=salesummery&intObjectID=6166184&sid=18abf70b-239c-41f7-bf78-99c5a4370bc7.

[Unk19] Unknown. *Game Theory — The Minimax Algorithm Explained*. [Online; accessed 28.02.2021]. 2019. URL: https://towardsdatascience.com/how-a-chess-playing-computer-thinks-about-its-next-move-8f028bd0e7b1.

[Unk20a] Unknown. *Generative Adversarial Networks*. [Online; accessed 26.02.2021]. 2020. URL: https://de.wikipedia.org/wiki/Generative_Adversarial_Networks.

[Unk20b] Unknown. *Künstliches neuronales Netz*. [Online; accessed 28.02.2021]. 2020. URL: https://de.wikipedia.org/wiki/K%C3%BCnstliches_neuronales_Netz.

[Unk20c] Unknown. *Perzeptron*. [Online; accessed 28.02.2021]. 2020. URL: https://de.wikipedia.org/wiki/Perzeptron.

[Unk21a] Unknown. *Multilayer perceptron*. [Online; accessed 28.02.2021]. 2021. URL: https://de.wikipedia.org/wiki/Backpropagation.

[Unk21b] Unknown. *Multilayer perceptron*. [Online; accessed 28.02.2021]. 2021. URL: https://en.wikipedia.org/wiki/Multilayer_perceptron.

Glossar

GAN Generative Adversarial Networks - Zwei neuronale Netze, die ein Nullsummenspiel durchführen. Eines erstellt Kandidaten, das andere bewertet sie.[Unk20a].

Multilayer-Perceptron Ein Multilayer-Perceptron ist eine Klasse von künstlichen neuronalen Netzen. Es besteht aus mindestens drei Layern: dem Input-Layer, einem Hidden-Layer und einem Output-Layer. Vielfach wird es auch als „vanilla“ neuronales Netz bezeichnet.[Unk21b].

KNN Siehe Multilayer-Perceptron.

Backpropagation Die Backpropagation beschreibt ein Verfahren, um neuronale Netze zu trainieren. Es basiert auf dem mittleren quadratischen Fehler und gehört zu der Gruppe der überwachten Lernverfahren.[Unk21a].