Eine Einführung in Generative Adverserial Network(GAN)

Andreas Wiegand

Seminar KI: gestern, heute, morgen
Angewandte Informatik, Universität Bamberg

Zusammenfassung. Generative Adverserial Network(GAN) ist ein künstliches neuronales Netzwerk(ANN) aus dem Bereich der generativen Modelle. Die Aufgabe des GANs ist es, die Wahrscheinlichkeitsverteilung von Trainingsdaten zu erlernen und dadurch anschließend neue Samples aus dieser Wahrscheinlichkeitsverteilung zu generieren. Man erhofft sich, den hohen Datenaufwand beim trainieren von ANN zu umgehen und durch GANS neue Trainingsdaten zu generieren. Im Folgenden wird auf die Theorie von GANs eingegangen, wie deren Aufbau und deren zugrunde liegendes mathematische Modell. Es wird aufgezeigt wie GANs trainiert werden. Des weiteren sollen Vorteile gegenüber anderen generativen Modelle aufgezeigt werden und derzeitige Schwachstellen von GAN, sowie Möglichkeiten diese zu verbessern.

1 Einleitung

In den letzten Jahren haben sich im Deep Learning Bereich besonders die discriminativen Modelle hervorgehoben, welche Input Daten wie Bilder, Audio oder Text Daten zu bestimmte Klassen zuordnen. Der Grund für das steigende Interesse liegt in der niedrigen Fehlerrate bei discriminativen Aufgaben, im Vergleich zu anderen Maschine Learning Ansätzen, wie Desicion Trees oder Markov Chains(Goodfellow, Bengio, Courville, & Bengio, 2016). Die Modelle lernen eine Funktion welche Input Daten X auf ein Klassen Label Y abbildet. Die Modelle werden dabei von ANN repräsentiert. Man kann auch sagen das Model lernt die bedingte Wahrscheinlichkeitsverteilung $P(Y|X)$ (Ng & Jordan, 2002). Generative Modelle haben die Aufgabe die Wahrscheinlichkeitsverteilung von Trainingsdatendaten zu erlernen. Es lernt eine multivariate Verteilung $P(X, Y)$, was dank der Bayes Regel auch zu $P(Y|X)$ umgeformt werden kann und somit kann das Modell auch für discriminativen Aufgaben herangezogen werden. Gleichzeitig können aber neue (x,y) Paare erzeugt werden, was zu dem Ergebnis von neuen Datensätzen führt welche nicht Teil der Trainingssample sind (Ng & Jordan, 2002). In diesem Paper wird speziel auf GAN, aus der Vielzahl von generativen Modellen eingegangen. Diese wurden von Goodfellow(Goodfellow et al., 2014) entwickelt und ebneten den Weg für Variationen, welche auf der Grundidee von GANs aufbauen. 2017 wurden alleine 227 neue Paper zu diesem Thema vorgestellt. Ein Grund weshalb GAN an Popularität gewinnt ist der, dass neuronale

Netzwerke mit Zunahme der Trainingsdatenanzahl eine Erhöhung der Performance für die sie Trainiert werden zeigen. Was bedeutet,dass sich mit Zunahme der Datenanzahl die Chance auf eine bessere Performance der Neuronalen Netzwerke ergibt (Halevy, Norvig, & Pereira, 2009). An diesem Punkt erhofft man sich durch GANS mehr qualitative Daten zu erzeugen und somit das Trainingsergebnis zu discriminativen Modelle zu verbessern.

2 Grundlagen

Im folgenden Kapitel wird auf theoretische Grundlagen eingegangen, welche zum Verständnis von GANs benötigt werden. Zunächst werden generative Modelle im Allgemeinen vorgestellt, welche den Grundgedanken der Datengeneration für GANs aufzeigen. Darauf folgend werden Convolutional Neuronale Netzwerke vorgestellt, aus welchen GANs aufgebaut werden können, um mit Bild Daten zu arbeiten.

2.1 Generative Modelle

Generative Modelle haben das Ziel eine Wahrscheinlichkeitsverteilungen zu erlernen. Anschließend kann diese als ein Modell genutzt werden und Samples zu erzeugen. Die Modelle können dabei beispielsweise auf ANN oder Markov Chains trainiert werden(Goodfellow et al., 2016). Im folgenden liegt der Fokus auf ANNs. Ein mögliches Anwendungsbeispiele wären es dem generativen Model, Bilder von bestimmten Objekten zunächst als Trainingsdaten zu geben. Anschließend können von er erlernten hochdimensionalen Wahrscheinlichkeitsverteilung Samples gezogen werden und neue Bilder erzeugt werden, welche nicht im Trainingsdatensatz vorhanden waren. Allgemein gehalten können jegliche Typen von Daten wie Text, Bild oder Audiodateien für generative Modelle herangezogen werden. Es gibt unterschiedliche Typen von generativen Modellen, welche sich vom Aufbau des Neuronalen Netzwerk und der Zielfunktion unterscheiden. Beispiele dafür sind Boltzmann Maschine, Autoencoder oder Deep Belief Networks(Goodfellow et al., 2016). Diese Arbeit beschäftigt sich mit einen anderen Vertreter, dem Generativ Adverserial Network(GAN). In den letzten Jahren konnte sich das GAN als best practice Ansatz bei den generativen Modellen herausarbeiten was Performancegründe bei der Trainierbarkeit und Qualität der generierbaren Daten zu Grunde liegt(Goodfellow et al., 2016). Die Modelle arbeiten nach der Maximum Likelihood Schätzverfahren(ML-Schätzer) in dem die Parameter θ dahingegen angepasst werden, dass die unsere beobachtetet Daten am ehesten passen. Man kann ML-Schätzer als Kulback-Leibler(KL) Divergenz darstellen und das generative Modelle das Ziel haben die KL Divergenz zwischen den Trainingsdaten $\mathrm{P_r}$ und den generierten Daten $\mathrm{P_g}$ zu minimieren. In Abbildung 1 ist dieser Prozess dargestellt. Diese Modelle versuchen dann die diese Cost Funktion

$$KL(P_\mathrm{r}||P_\mathrm{g}]) = \int_x P_\mathrm{r} log \frac{P_\mathrm{r}}{P_\mathrm{g}} dx$$

zu minimieren. Wenn nun beide Verteilungen $P_r = P_g$ sind, hat das Model sein Minimum Loss erreicht und Θ muss nicht mehr angepasst werden. Interessant wird es, wenn $P_r \neq P_g$. Wenn $P_r > P_g$ führt, das dazu dass das Integral schnell gegen unendlich konvergiert. Was dazuführt, das hohe Kosten entstehen wenn die Verteilung von generativen Modell erzeugt, nicht die Daten abdeckt. Wenn nun $P_r < P_g$ ist, dann bedeutet das, dass x eine niedrige Wahrscheinlichkeit hat aus unseren Trainingsdaten zu kommen aber eine hohe Wahrscheinlichkeit von den Generator erzeugt zu werden. Dann würde sich die KL gegen 0 konvergieren. Was zur Folge hat, dass der Generator Falsch ausschauende Daten geniert aber keine Kosten dafür erzeugt werden und im Umkehrschluss es zu keinen Veränderungen unseren Θ kommt. Man vermutet das dies der Grund für die Problematiken von generativen Modellen wie Autoencoder und CO sind. Dies ist aber noch kein abgeschlossenen Problem und wird weiterhin erforscht(Salimans et al., 2016). Unter GAN werden wir auf dieses Problem erneut aufgreifen und es wird gezeigt inwiefern sich GAN dieses Problem angeht. Generative Modelle gehören zu einem Bereich des unüberwachten Lernens, da keine Labels für die Trainingsdaten gebraucht werden. Probleme welche diese Modelle haben sind beispielsweise, dass Autoencoder zwar mit wenig Trainingsaufwand trainiert werden können jedoch sind die generierten Bilder sehr trüb. Allgemein haben Autoencoder und Co. Vorteile im Lernen des latenten Raums von Objektlassen, weisen aber Probleme beim Generieren von neuen Daten auf. Da gezeigt wurde das im Deep Learning Bereich die discrimativen Modelle mit Zunahme der Daten stark an Leistung zunehmen. Und GANs Stärke in der Datengeneration in guter Qualität liegt. Kann es seine Vorteil gegenüber den anderen Modellen ausspielen(Salimans et al., 2016).

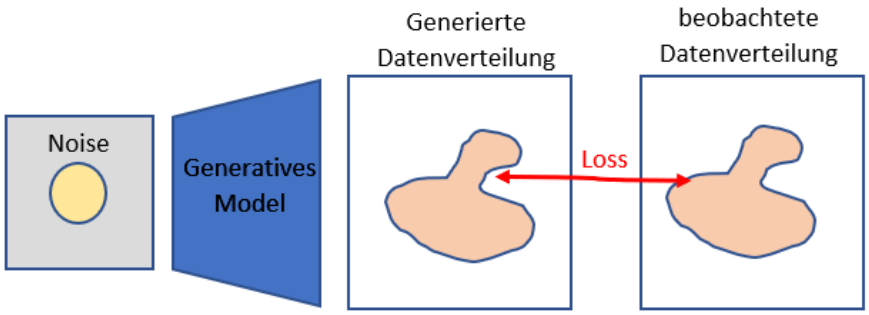

Abb. 1. Generative Modelle

2.2 Convolution Neural Networks

Convolution Neural Netzworks(CNN) sind eine besondere Art von künstlichen neuronalen Netzwerken, sie sind dafür konzipiert auf Datensätzen zu arbeiten welche in eine Matrix Form gebracht worden sind. Der Input eines CNN können beispielsweise Bilder sein welche durch die Matrix A = $\begin{bmatrix} a_1 & a_1 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ \vdots & n_n & \ddots & \vdots \\ x_1 & x_2 & x_3 & x_n \end{bmatrix}$ dargestellt werden. Jedes Element x_{ij} stellt einen Pixel eines Bildes da, wobei x_n $\in$ [0,255] . Die Matrix $A^{w \cdot b \cdot c}$ stellt w·b·c = N dimensionale Matrix da. Wobei w die Länge und b Breite des Bildes entspricht. c sind die Farbspektren eines Bildes und sind in einen RGB-Farbraum 3 beziehungsweiße in einen schwarz-weiß Bild 1. Nachdem der Input eines CNN definiert ist kommt nun der Aufbau. CNN setzen sich aus mehrere Schichten von Convolution Layern zusammen. Ein Netzwerk kann mehrere N-Layer haben. Wobei jeder Layer aus mehreren Convolution oder auch Kernels genannt, zusammengesetzt ist. Ein Aufbau kann aus Abbildung 2 entnommen werden(Goodfellow et al., 2016).

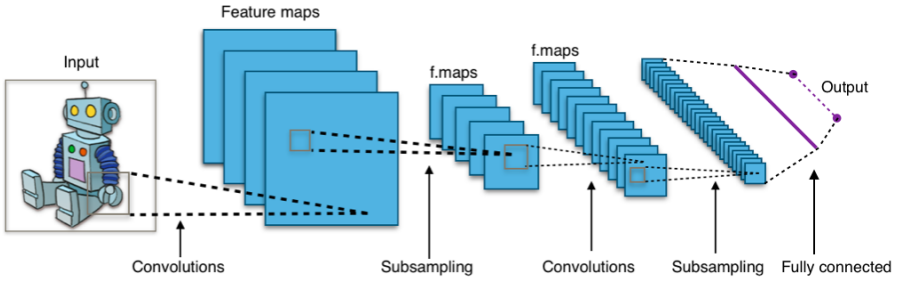

Abb. 2. Convolutional Neural Network

Die Kernels, also die einzelnen Filter, von den jeder der N-Layer k besitzt sind $K^{n \cdot n}$ Matrizen jedes k_{ij} in einem Filter einspricht einen aus der üblichen Neuronalen Netzwerk Architektur bekannten Gewichte. Diese Gewichte werden dann durch den Backprobagation-Algorithmus in der Trainingsphase des Netzwerkes angepasst um den Verlust der Loss-Funktion durch bestimmten des Gradianten zu minimieren. Das durch die Abbildung 2 dargestellte Subsampling ist der Output aus den Convolutional Layern(Goodfellow et al., 2016).

Da Input und Kernel unterschiedliche Größen haben und man den gesamten Input mit den Kernel abdecken möchte, bewegt sich der Filter um s Position auf den Input und führt erneut einen Berechnungschritt durch. Dieser Vorgang wird Stride genannt. An jeder Position wird das Produkt von jeden x_{ij} des Input und k_{ij} des Kernel durchgeführt. Anschließend werden alle Produkte aufsummiert. In Abbildung 3 ist dieser Vorgang verdeutlicht. Zusätzlich gibt es die

Möglichkeit für das sogenannte Zero Padding P. Dabei werden mehrere 0 um die Input Feature Map, am Anfang und Ende der Axen anfügt. Dies ist notwendig wenn Kernel und Input Größe nicht kompatible zueinander sind. Die Anzahl der möglichen Positionen ergeben sich aus Kernel Größe und den Input des jeweiligen Kernel sowie des Strides. Die Output Größ W kann berechnet werden durch W = (W-F+2P)/s+1. Wobei F für die Größe des Kernel steht (Dumoulin & Visin, 2016).

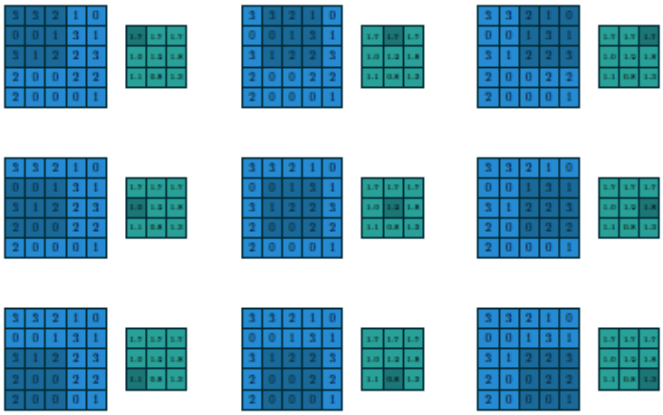

Abb. 3. Convolution Beispiel (Dumoulin & Visin, 2016)

Um besser zu verstehen welche Auswirkungen die Anzahl der Kernels in Layer n auf die Größe des Outputs von n und die Anzahl der Kernels in Layer n+1 für den nächsten Layer haben, wird ein Beispiel aufgezeigt. Der erste Layer hat 20 Kernels mit der Größe 7x7 und Stride 1. Der Input A für einen Kernel K ist ein 28x28 Matrix. Der Output aus diesen Filter sind 20 22X22 Feature Maps. Würde der Input ein 28x28x3 Bild mit 3 RGB Channels sein, der Output 60 22x22 Feature Maps. Allgemein kann Convolution Layer als Supsampling gesehen werden und Stride gibt an wieviele Dimensionen bei diesen Prozess pro Convolution Layer entfernt werden soll. Der letzte Layer ist ein fully-connected Layer welcher den typischen Anforderungen von ANN entspricht (Dumoulin & Visin, 2016).

Transposed Convolution, auch genannt Fractionally Strided Convolution oder Deconvolution ist eine Umkehrfunktion von der üblichen Convolution. Es verwendet die gleichen Variablen wie Convolution. Dabei wird ein Kernel K mit der Größe N x N definiert der Input I mit der Größe N x N und Stride s = 1. Deconvolution kann wie Convolution angesehen werden mit Zero Padding auf dem Input. Das in Abbildung 4 gezeigte Beispiel zeigt einen deconvolution Vorgang mit eine 3x3 Kernel über einen 4x4 Input. Dies ist gleich mit einen Convolution

Schritt mit einen 3x3 kernel auf einen 2x2 Input und einer 2x2 Zero Padding Grenze. Convolution ist Supsampling und mit Deconvolution wird Upsampling betrieben. Durch diesen Schritt kommt es zu einer Dimensionserhöhung des Inputs. Die Gewichte der Kernels bestimmen wie der Input transfomiert wird. Durch mehrere Schichten von Deconvolution Layer kann von einer Input Größe NxN auf eine Output Größe KxK, wobei K > N mit Abhängigkeit von Kernel und Stride abgebildet werden(Dumoulin & Visin, 2016).

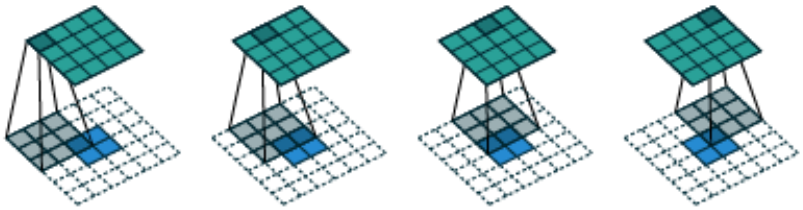

Abb. 4. Deconvolution Beispiel(Dumoulin & Visin, 2016)

3 Generative Adversarial Network

Ein GAN besteht besteht aus zwei ANN, dem Discriminator D und dem Generator G. Das Ziel des G ist es, Daten x zu erzeugen, welche nicht von Trainingsdaten y unterschieden werden können. Dabei wird eine vorangegangene Input Noise Variable $p_z(z)$ verwendet, welche eine Abbildung zum Datenraum $G(z;\Phi_g)$ herstellt. Dabei sind Φ_g die Gewichte des neuronalen Netzwerkes von G. Der Discriminator hat die Aufgabe zu unterscheiden, ob der jeweilige Datensatz von G erzeugt wurde und somit ein fake Datensatz ist, oder von Trainingsdaten y stammt(Goodfellow et al., 2014). Die Zusammensetzung zwischen den beiden Netzwerken kann aus Abbildung 5 entnommen werden.

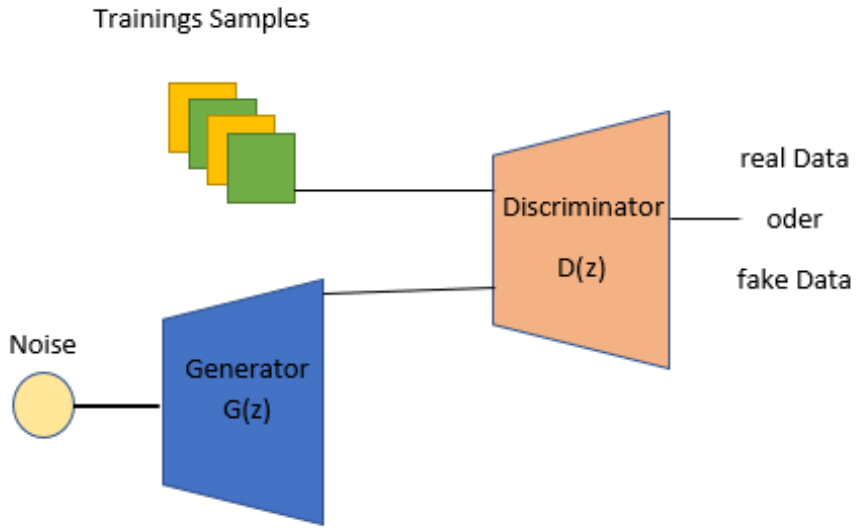

Abb. 5. Generativ Adverserial Network

Der Discriminator ist definiert durch $D(x;\Phi_d)$. Wobei Φ_d die Gewichte des Descriminators sind und D(x) die Wahrscheinlichkeit ist, dass x von den Trainingsdaten stammt und nicht von p_g. Die Wahrscheinlichkeitsverteilung für unsere Trainingsdaten ist p_r. Im Training werden dann Φ_d so angepasst, dass die Wahrscheinlichkeit Trainingsbeispiele richtig zu klassifizieren maximiert wird. Und Φ_g wird dahingehen trainiert die Wahrscheinlichkeit zu minimieren, so dass D erkennt dass Trainingsdatensatz x von G erzeugt wurde. Mathematisch ausgedrückt durch log(1 - D(G(z))). Die gesamte Loss-Funktion des vanilla GAN

ist definiert als

$$min_{\mathrm{G}} max_{\mathrm{D}} V(D,G) = E_{\mathrm{x}\sim \mathrm{p_{data}(x)}}[logD(x)] + E_{\mathrm{z}\sim \mathrm{p_z(z)}}[log(1-D(G(z))]$$

diese beschreibt ein Minmax Spiel zwischen G und D. Welches das globale Optimum erreicht hat wenn $\mathrm{p_g} = \mathrm{p_r}$. Das heißt, wenn die Datenverteilung, welche von G erzeugt wird, gleich der unserer Trainingsdaten ist(Goodfellow et al., 2014). Das Training erfolgt durch den folgenden Algorithmus:

Algorithm 1: Minibatch stochastic gradient descent Training für Generative Adversarial Networks. Die Anzahl der Schritte welche auf den Discriminator angewendet wird ist k

for *Anzahl von Training Iterationen* **do**
  for *k Schritte* **do**
    •Sample minibatch von m noise Samples $\mathrm{z}^{(1)}$,...,$\mathrm{z}^{(\mathrm{m})}$von noise $\mathrm{p_g(z)}$
    • Sample minibatch von m Beispielen $\mathrm{x}^{(1)}$,...,$\mathrm{x}^{(\mathrm{m})}$von Daten Generationsverteilung $\mathrm{p_{data}(x)}$
    • Update den Discriminator zum aufsteigenden stochastischen Gradianten:
    $\nabla_{\Phi_{\mathrm{d}}} \frac{1}{m} \sum_{i=1}^{m} [logD(x^{(\mathrm{i})}) + log(1 - D(G(z^{(\mathrm{i})})))]$
  end
  • Sample minibatch von m noise Samples $\mathrm{z}^{(1)}$,...,$\mathrm{z}^{(\mathrm{m})}$ von noise $\mathrm{p_g(z)}$
  • Update den Generator mit den absteigenden stochastischen Gradianten:
  $\nabla_{\Phi_{\mathrm{g}}} \frac{1}{m} \sum_{i=1}^{m} log(1 - D(G(z^{(\mathrm{i})})))$
end

Beim Training wird ein stochastischer Minibatch von mehren Trainingsdaten gleichzeitig erstellt. Dies soll dabei helfen, dass der Generator sich nicht auf bestimmte Gewichte fest fährt und auf Trainingssätze kollabiert. So weisen die erzeugten Daten mehr Variationen auf (Salimans et al., 2016). D wird zunächst in einer inneren Schleife auf n Trainingsätzen trainiert, womit man Overfitting von D vermeiden will, was zur Folge hätte, dass D nur den Trainingsdatensatz kopieren würde. Deshalb wird k mal D optimiert und ein mal G in der äußeren Schleife.

Ein möglicher Aufbau von GAN wird in Abbildung 6 dargestellt. Dies ist das sogenannte Deep Convolution GAN(DC GAN), welches dafür konzipiert wurde auf Bilddaten zu arbeiten. Dabei besteht der Generator aus mehren Schichten von Deconvolution Layern. Welche den Input Noise Variable $p_z(z)$ auf y abbildet. D besteht aus mehren Schichten von Convolution Layern und bekommt als Input die Trainingsdaten, oder die von G erzeugten Y, und entscheidet über die Klassifikation(Radford, Metz, & Chintala, 2015).

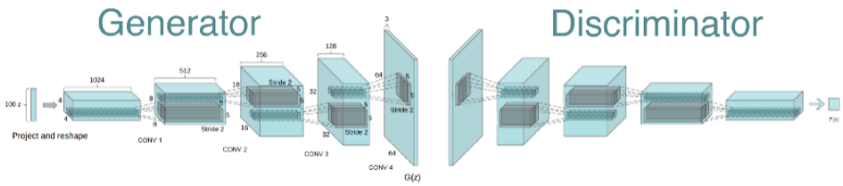

Abb. 6. Deep Convolutional GAN

Wie unter Generativen Modellen gezeigt wurde kann das asymmetrische Verhalten der KV Divergenz zu schlechten Trainingsergebnissen führen. Goodfellow (Goodfellow et al., 2014) zeigte, dass sich die MinMax Loss-Funktion des GAN auch als Jensen-Shannon Divergenz(JS Divergenz) darstellen lässt. Diese ist definiert als

$$D_{\mathrm{JS}}(P_{\mathrm{r}}||P_{\mathrm{g}}]) = \tfrac{1}{2}D_{\mathrm{KL}}(P_{\mathrm{r}}||\tfrac{P_{\mathrm{g}}+P_{\mathrm{r}}}{2}]) + D_{\mathrm{KL}}(P_{\mathrm{q}}||\tfrac{P_{\mathrm{g}}+P_{\mathrm{r}}}{2}])$$

wobei P_r die Wahrscheinlichkeitsverteilung der Trainingsdaten ist und P_g die des Generators. Huzár (Huszár, 2015) zeigte, dass durch das symmetrische Verhalten der JS Divergenz ein potentiell besseres Trainingsergebnis entstehen kann, im Vergleich zu der KL Divergenz. Damit zeigte weshalb GANs im Vorteil gegenüber anderen generativen Modellen sind. Abbildung 7 veranschautlicht dieses Konzept. Der linke Graph zeigt 2 Normal Verteilungen. In der Mitte wird die KV Divergenz der beiden Normal Verteilungen dargestellt. Rechts ist die JS Divergenz der Beiden dargestellt. Man sieht sehr gut das asymmetrische Verhalten der KV und das symmetrische der JS. Dadurch lassen sich aussagekräftigere Gradianten, bestimmen welche zum Optimieren von D und G benötigt werden(Huszár, 2015).

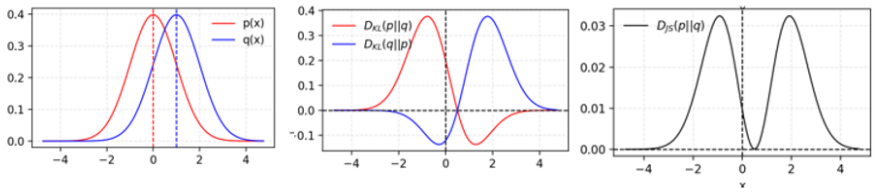

Abb. 7. KL Divergenz und JS Divergenz

3.1 Probleme mit Generative Adverserial Networks

Wie auch die anderen generativen Modelle haben auch GANs noch Schwächen bezüglich der Trainingsabläufe und der Qualität der generierten Daten. Im Folgenden wird auf einige Probleme eingegangen.

Equilibrium D und G betreiben ein MinMax Spiel. Beide versuchen das Nash Equilibrium zu finden. Dies ist der bestmögliche Endpunkt in einen nicht kooperativen Spiel. Wie in dem Fall von GAN wäre das wenn $p_g = p_r$. Es wurde gezeigt, dass das Erreichen dieses Punktes sehr schwierig ist, da durch die Updates der Gewichte mit den Gradianten der Loss-Funktion starke Schwingungen der Funktion enstehen können. Dies kann zur Instabilität für das laufende Training führen (Salimans et al., 2016).

Vanishing gradient Dies beschreibt das Problem, wenn D perfekt trainiert ist mit $D(x) = 1, \forall \in p_r$ und $D(x)=0 \; \forall x \in p_g$. Die Loss-Funktion würde in diesem Fall auf 0 fallen und es gäbe keinen Gradianten, für den die Gewichte von G angepasst werden können. Dies verlangsamt den Trainingsprozess bis hin zu einem kompletten Stopp des Trainings. Würde D zu schlechten trainiert mit $D(x) = 0, \forall \in p_r$ und $D(x)=1 \; \forall x \in p_g$. Bekommt G kein Feedback über seine Leistung bei der Datengeneration hat er keine Möglichkeit p_r zu erlernen (Salimans et al., 2016).

Mode Collapse Während des Trainings von GAN kann es dazu kommen, dass der Generator möglicherweise auf eine Einstellung seiner Gewichte fixiert wird und es zu einem sogenannten Mode Collapse führt. Was zur Folge hat, dass der Generator sehr ähnliche Samples produziert (Arjovsky, Chintala, & Bottou, 2017).

Keine aussagegräftigen Evaluations Metriken Die Loss Funktion der GANs liefert keine aussagekräftigen Evaluationsmöglichkeit über den Fortschritt des Trainings. Bei discriminativen Modellen im üblichen Maschine Learning besteht die Möglichkeit Validierungsdatensätze zu verwenden und an diesen die Genauigkeit des Modells zu testen. Diese Möglichkeit besteht bei GANs nicht(Huang et al., 2018).

3.2 Lösungsansätze für Generative Adverserial Networks Probleme

Nun werden einige Techniken aufgezeigt, welche die unter Abschnitt Probleme mit GAN genannten Schwierigkeiten angehen und zu einem effizienteren Training führen, damit eine schnellere Konvergenz während des Trainings erreicht wird.

Feature matching Dies soll die Instabilität von GANS verbessern und gegen das Problem des Vanishing Gradient angehen. G bekommt eine neue Loss-Funktion und ersetzt die des üblichen Vanilla GAN. Diese soll G davon abhalten, sich an D über zu trainieren und sich zu sehr darauf zu fokussieren, D zu täuschen und gleichzeitig auch versuchen die Datenverteilung der Trainingsdaten abzudecken(Salimans et al., 2016).

Minibatch discrimination Um das Problem des Mode Collapse zu umgehen, so dass es nicht zu einem Festfahren der Gewichten von G kommt, wird beim Trainieren die Nähe von den Trainingsdatenpunkten gemessen. Anschließend wird die Summe über der Differenz aller Trainingspunkte genommen und dem Discriminator als zusätzlicher Input beim Training hinzugegeben (Salimans et al., 2016).

Historical Averaging Beim Training werden die Gewichte von G und D aufgezeichnet und je Trainingsschritt i verglichen. Anschließend wird an die Lossfunktion je Trainingschritt die Veränderung zu i-1 an die Loss-Funktion addiert. Damit wird eine zu starke Veränderung bei den jeweiligen Trainingsschritten bestraft und soll gegen ein Model Collapse helfen (Salimans et al., 2016).

One-sided Label Smoothing Die üblichen Label für den Trainingsdurchlauf von 1 und 0 werden durch die Werte 0.9 und 0.1 ersetzt. Dies führt zu besseren Trainingsergebnissen. Es gibt derzeit nur empirische Belege für den Erfolg, jedoch nicht weshalb diese Technik besser funktioniert(Salimans et al., 2016).

Adding Noises Noise an den Input von D zu hängen kann gegen das Problem des Vanishing gradienten helfen und das Training verbessern(Salimans et al., 2016).

Use Better Metric of Distribution Similarity Die JS Divergenz von vanilla GAN sorgt für bessere Trainingsergebnisse im Vergleich zu der KL Divergenz von anderen generativen Modelle. Jedoch weißt die JS immernoch Probleme auf. Es wird vorgeschlagen diese durch die Wasserstein Metric zu ersetzen, da diese bessere Ergebnisse bei disjunkten Wahrscheinlichkeitsverteilungen liefern kann(Salimans et al., 2016).

4 Zusammenfassung und Ausblick

Ein GAN besteht aus zwei ANN, dem Discriminator und dem Generator. Das Ziel des G ist es Daten zu erzeugen, welche nicht von Trainingsdaten unterschieden werden können. Der Discriminator hat die Aufgabe zu unterscheiden ob der jeweilige Datensatz von G erzeugt wurde und somit ein fake Datensatz ist, oder von den Trainingsdaten stammt(Goodfellow et al., 2014). GANs werden derzeit noch erforscht. Es gibt noch einige offene Fragen, beispielsweise bezüglich der Performance hochauflösender Bilder(Wang et al., 2017). Es wurden in diesem Paper einige Probleme, welche beim Trainieren von GAN auftreten können und mögliche Lösungsansätze, vorgestellt. Es gibt derzeit einige praktische Ansätze, welche in der Anwendung auf GANs zurück greifen. Beispielsweise durch Textbeschreibung eigene Bilder als Output generiert werden (Reed et al., 2016), oder 3D Daten erzeugt werden(Wu, Zhang, Xue, Freeman, & Tenenbaum, 2016). GANs finden Anwendung in unterschiedlichen Bereichen des Deep Learnings, da sie als Lösung des Problems angesehen werden, dass Neuronale Netzwerke eine hohe Menge an Trainingsdaten benötigen und GANs dieses Problem durch ihre Fähigkeit, neue Daten zu generieren, umgehen. GANs lernen eine Art "versteckeRepräsentation von Klassen, was dazu beitragen kann auch Modelle von komplexen Prozessen zu erlernen. Es gibt erste Ansätze bei denen im Reinforcement Learning durch GANs versucht wird Modelle von der Umwelt eines Agenten zu erlernen, welche dann auf Grundlage dieser Modelle Vorhersagen über zukünftige Ereignisse treffen kann(Pinto, Davidson, Sukthankar, & Gupta, 2017).

Literatur

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1). MIT press Cambridge.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680).

Halevy, A., Norvig, P., & Pereira, F. (2009). The unreasonable effectiveness of data. *IEEE Intelligent Systems*, *24*(2), 8–12.

Huang, G., Yuan, Y., Xu, Q., Guo, C., Sun, Y., Wu, F., & Weinberger, K. (2018). *An empirical study on evaluation metrics of generative adversarial networks.* Retrieved from `https://openreview.net/forum?id=Sy1f0e-R`
-

Huszár, F. (2015). How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv preprint arXiv:1511.05101*.

Ng, A. Y., & Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems* (pp. 841–848).

Pinto, L., Davidson, J., Sukthankar, R., & Gupta, A. (2017). Robust adversarial reinforcement learning. *arXiv preprint arXiv:1703.02702*.

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. (2016). Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). Improved techniques for training gans. In *Advances in neural information processing systems* (pp. 2234–2242).

Wang, T.-C., Liu, M.-Y., Zhu, J.-Y., Tao, A., Kautz, J., & Catanzaro, B. (2017). High-resolution image synthesis and semantic manipulation with conditional gans. *arXiv preprint arXiv:1711.11585*.

Wu, J., Zhang, C., Xue, T., Freeman, B., & Tenenbaum, J. (2016). Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in neural information processing systems* (pp. 82–90).