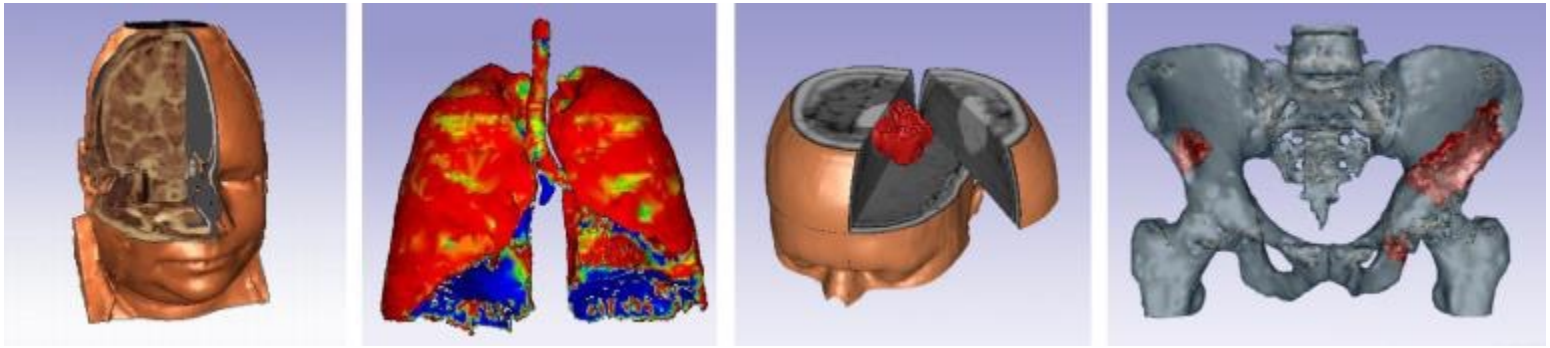
Modell- und KI-basierte Bildverarbeitung in der Medizin



Autoencoder und Generative Adversarial Networks

Dr.-Ing. Marian Himstedt

Institut für Medizinische Informatik

Universität zu Lübeck

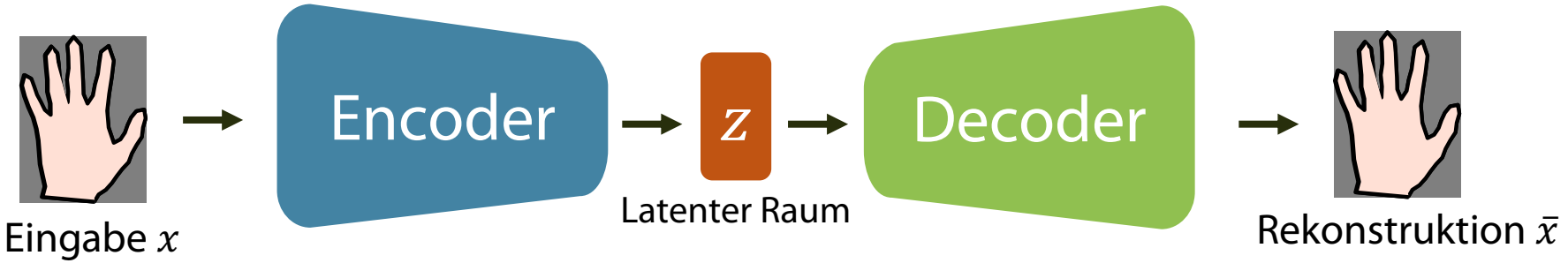
Autoencoder

Einführung

- Künstliches neuronales Netz
- Unüberwachte Methode des maschinellen Lernens
  - d.h. es sind **keine** annotierten Daten für das Training erforderlich
- Ziele:
  - Komprimierung und Visualisierung von hochdimensionalen Daten

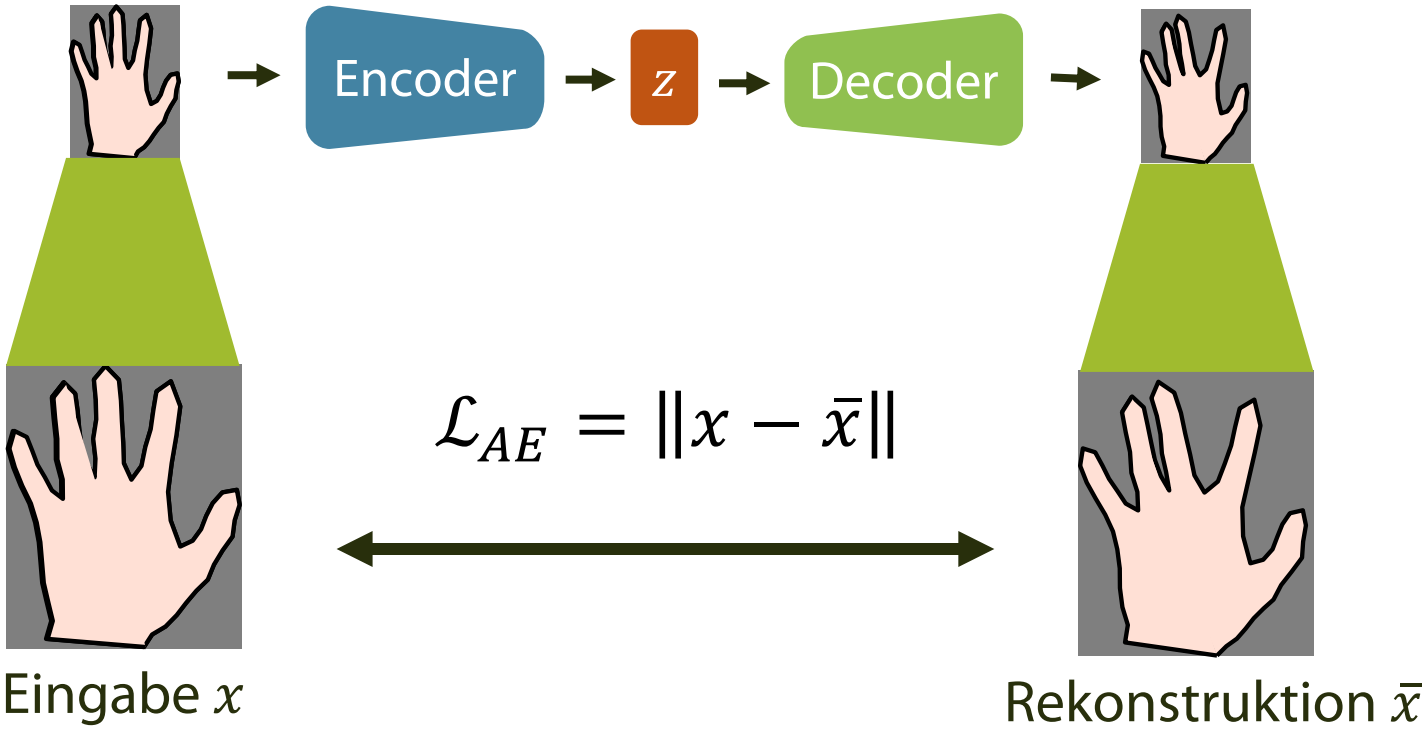
- Grundidee
  - Dimensionsreduzierte Repräsentation hochdimensionaler Eingabedaten in einem niedrigdimensionalen Raum (**Encoding**) so, dass
  - aus der dimensionsreduzierten Repräsentation die originären Eingabedaten möglichst genau rekonstruiert (**Decoding**) werden können.

Aufbau



Encoder	Latenter Raum	Decoder
- Neuronales Netz (NN) - Abbildung des Eingabe-vektors x in einen niedrigdimensionalen Vektor z (encoding) - Reduktion der Dimension, $\dim(x) > \dim(z)$	Kodierte Repräsentation der Eingabe - Bottleneck des Autoencoders - z liegt in einem latenten (unbekann-ten) Raum	- Neuronales Netz - Generiert aus z die Rekonstruktion $\bar{x}$ mit möglichst geringem Rekonstruktionsfehler (decoding)

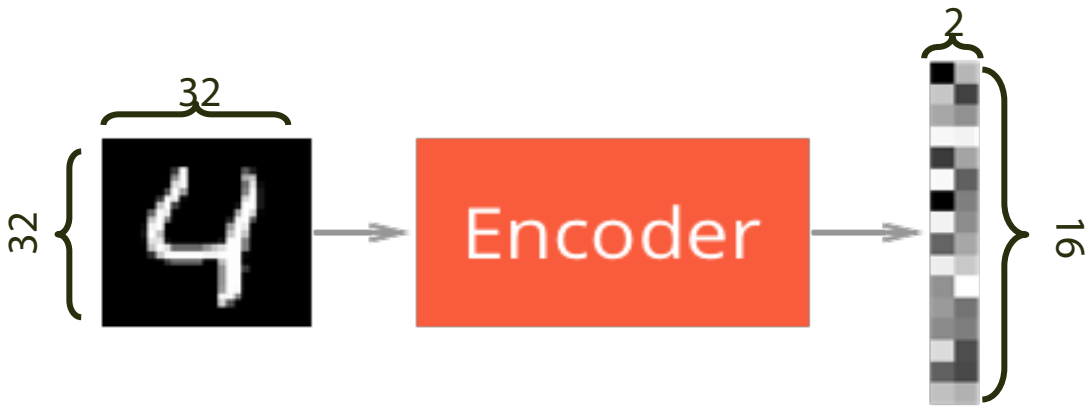
Fehlerfunktion des Trainings



- Fehlerfunktion $\mathcal{L}_{AE}$ ist Maß für die Ähnlichkeit der Eingabe und der Rekonstruktion z.B. L2-Loss, Dice-Loss,…

Lernen von wesentlichen Merkmalen

- Im Encoder-Schritt wird eine Kompression der Eingabedaten durchgeführt.
- Man erhält eine dimensionsreduzierte Repräsentation z hochdimensionaler Eingabedaten x in dem niedrigdimensionalen, latenten Raum ***Z***.
  - Hierdurch wird eine reduzierte Anzahl an Merkmalen gelernt, auch ***„Meaningful Features" genannt,*** die die Eingabedaten möglichst gut beschreiben.



Dimension: $d_1 = 32 * 32 = 1024$ > $d_2 = 2 * 16 = 32$

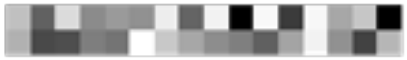
https://github.com/udacity/deep-learning/blob/master/autoencoder/Simple_Autoencoder_Solution.ipynb

Anwendungsbeispiel: Komprimiertes Speichern

1. Zu speicherndes Bild

Encoder

2. Komprimierte Repräsentation



Nur der 32-dimensionale z-Vektor wird gespeichert.
→ Verringerung des benötigten Speichers

Decoder

3. Rekonstruktion (z.B. zur Darstellung auf dem Monitor)

Spezialfall: Denoising Autoencoder

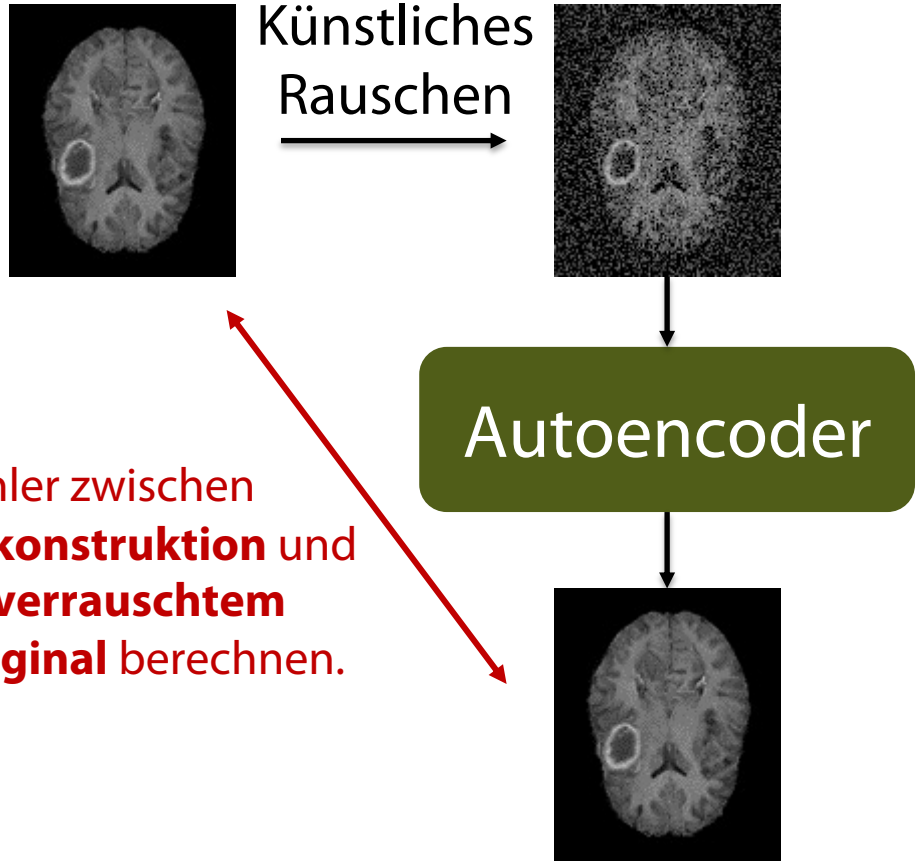
- **Ziel**
  - Trainieren eines Autoencoders, der aus verrauschten Bildern rauschreduzierte Bilder generiert.

- **Methode**
  - Die Eingabebilddaten werden während des Trainings mit künstlichem Rauschen überlagert.
  - Man hat somit von jedem Bild zwei Versionen:
    - Originalbilddaten + künstlich verrauschte Bilddaten.
  - Der Autoencoder reduziert zur Bildrepräsentation im latenten Raum Z die Dimension.
  - Nach dem Training generiert der Autoencoder aus dieser niedrigdimensionalen Bildrepräsentation eine rauschreduzierte Version des verrauschten Bildes.
  - Man erhält eine neuronalen Glättungsfilter.

- **Eigenschaften**
  - Wenn die Dimension des latenten Raums nur wenig kleiner ist als die der Eingabedaten, kann der Autoencoder eine Identitätsfunktion approximieren.
    - Die Dimensionsreduktion ist dann gering und der Autoencoder lernt keine „meaningful features“.
    - Die Rauschreduktion ist dann entsprechend gering.
  - Bei geeigneter Wahl der Dimension des latenten Raumes erlernt der Autoencoder die wesentlichen Features zur Repräsentation der Bildstrukturen (ohne Rauschen)
    - Dann wird durch die Anwendung des trainierten Autoencoders aus einem verrauschten Bild ein rauschreduziertes Bild generiert.

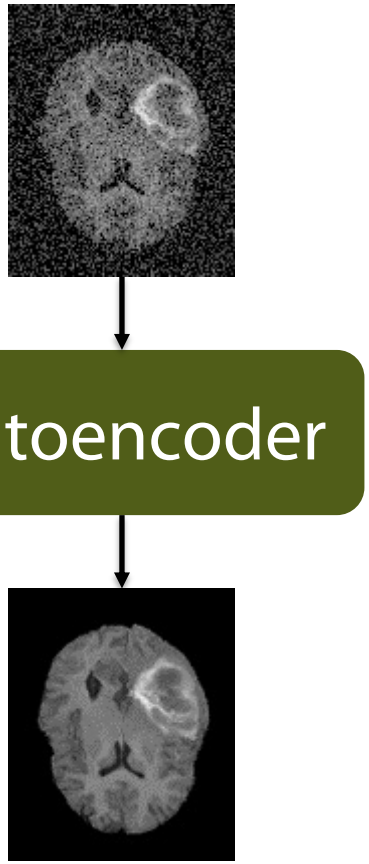
Denoising Autoencoder

Training



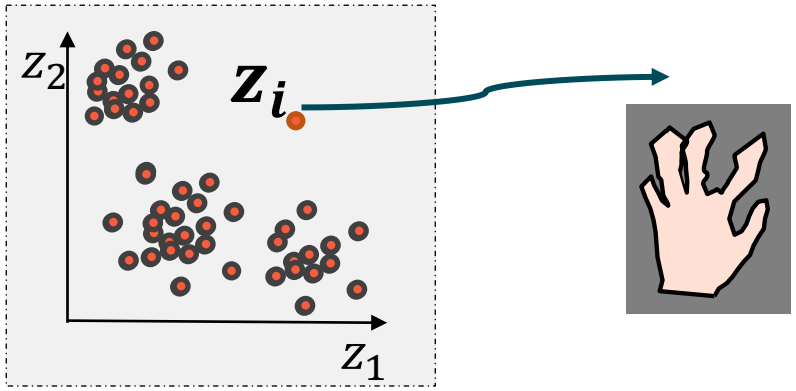
Fehler zwischen **Rekonstruktion** und **unverrauschtem Original** berechnen.

Anwendung



Autoencoder als Generatives Modell?

- Vorbemerkung: Im latenten Raum ist ein Bild durch einen niedrigdimensionalen z-Vektor repräsentiert.
- Frage: Kann ich durch zufällig gewählte Vektoren z_i aus dem latenten Raum realistische synthetische Bilder einer Klasse generieren?
- Beispiel: Handbilder, dim(z)=2

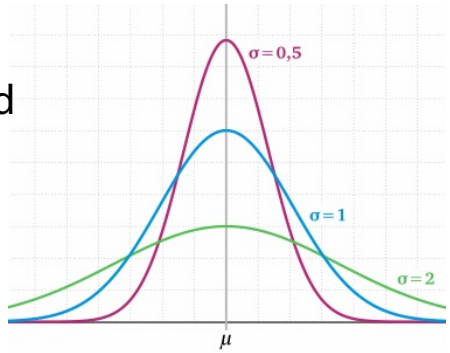


Unplausible Rekonstruktion!

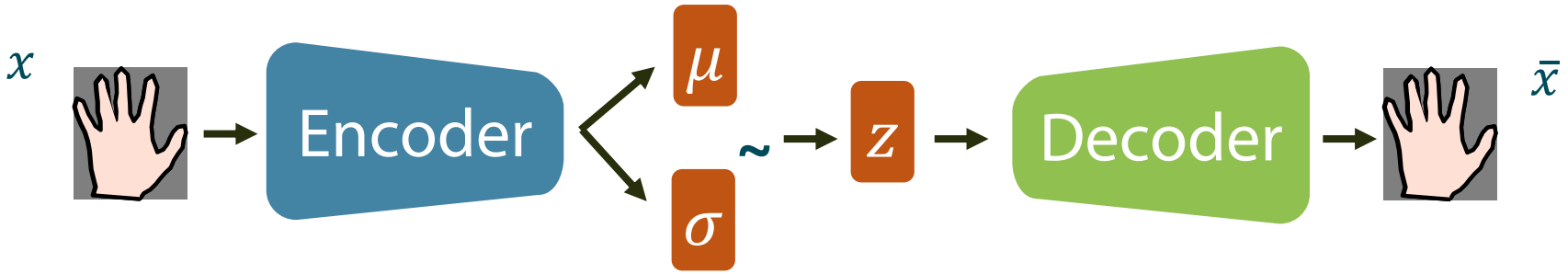
- Problem: Die Verteilung der Vektoren im latenten Raums ist unbekannt. Daher hat mein keine Information, ob ein z-Vektor mit hoher oder geringer Wahrscheinlichkeit zu einer Bildklasse (z.B. Handbilder) gehört. Es ist keine sinnvolle Interpolation im z-Raum möglich.
- Lösungsansatz : Die Verteilung der z-Vektoren des latenten Raums einschränken und z.B. durch Normalverteilungen beschreiben, die bildklassenspezifisch erlernt werden.
  → Variationeller Autoencoder

Variationeller Autoencoder (VAE)

- Grundlegender Aufbau entspricht einem Autoencoder.
- Unterschiede:
  - Die Verteilung der z-Vektoren einer Bildklasse im latenten Raum wird durch eine Normalverteilung $\boldsymbol{N}(\mu, \sigma)$, auch Gaußverteilung genannt, charakterisiert.
    - Die Normalverteilung ist durch den Erwartungswert (Mittelwert) μ und Standardabweichung (empirische Standardabweichung) σ eindeutig beschrieben.
  - Im Training werden aus den Eingabedaten einer Bildklasse zunächst die Verteilungsparameter μ und σ generiert.
  - Vektor z im latenten Raum wird dann zufällig aus dieser Gauß-Verteilung $\boldsymbol{N}(\mu, \sigma)$ generiert und variiert somit in jedem Trainingsdurchlauf.



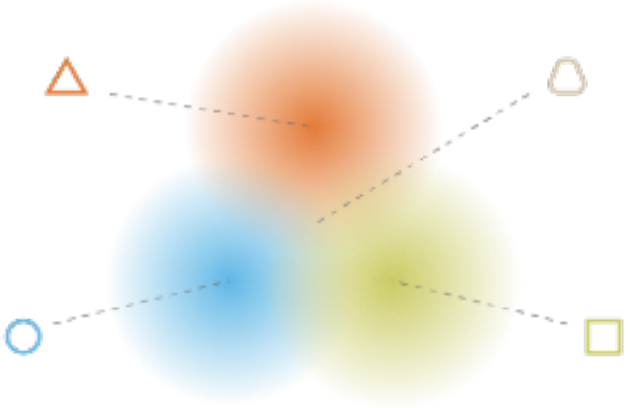
Bsp.: 1-dim. Normalverteilung



Modellierung des latenten Raumes

- Bei n Bildklassen werden die Verteilungen im Raum durch n multivariate Normalverteilungen beschrieben.
- Sie werden durch n-dimensionale Mittelwertvektoren und Standardabweichungsvektoren eindeutig festgelegt. Korrelationen zwischen den Merkmalen werden vernachlässigt, d.h. gleich 0 gesetzt.

Irregulärer latenter Raum eines Autoencoders
- Nicht kontinuierlich
- Keine sinnvolle Interpolation möglich



Durch Normalverteilungen modellierter latenter Raum eines variationellen Autoencoders
- Kontinuierlich
- Sinnvolle Interpolation möglich

Fehlerfunktion für das Training des VAEs

$$\mathcal{L}_{VAE} = \|x - \bar{x}\| + D_{KL}(N(\mu, \sigma), N(0,1))$$

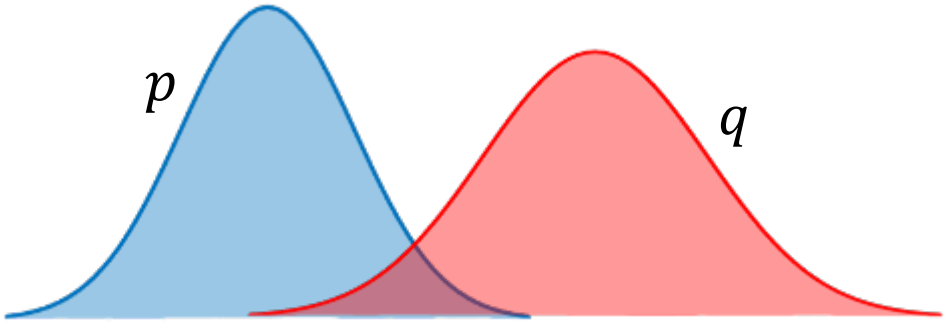
- Minimieren der Distanz zwischen Eingabe x und Rekonstruktion $\bar{x}$
  - Wie bei anderen Autoencodern
- Minimieren der Unterschiedlichkeit (Divergenz) zwischen der kodierten $N(\mu, \sigma)$ und der gewünschten $N(0,1)$ Verteilung.
  - Durch dies Zusatzkriterium soll die Verteilung $N(\mu, \sigma)$ sich der Einheitsnormalverteilung während des Trainingsprozesses annähern.
- Man erhält im latenten Raum eine approximativ einheitsnormalverteilte Repräsentation der Bilddaten.

https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Kullback-Leibler Divergenz

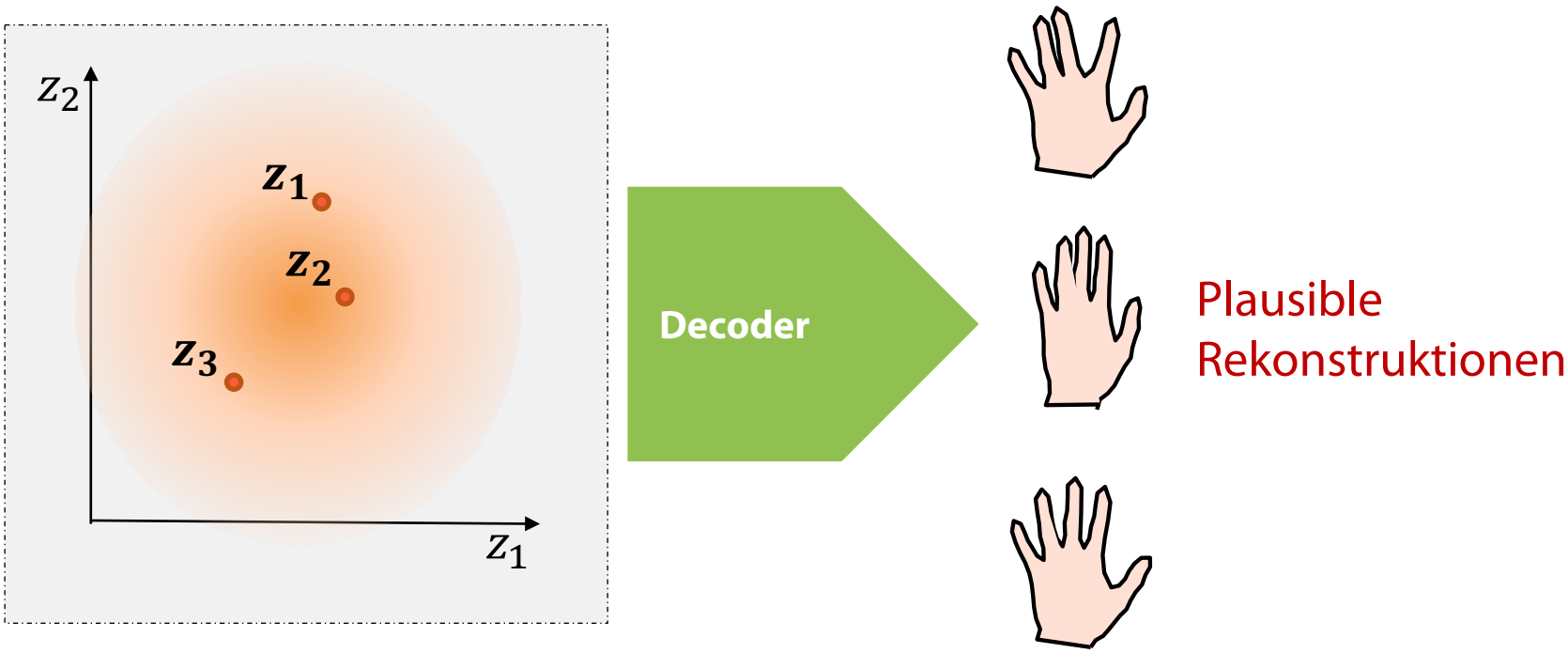
- Die Kullback-Leibler Divergenz misst die Abweichung zweier Verteilungen p und q (Achtung! nicht die Distanz)
- D_{KL} ist ein Maß dafür, wie hoch der erwartete Informationsverlust ist, wenn man die Verteilung p durch die Verteilung q approximiert.

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i)(\log p(x_i) - \log q(x_i)) = \sum_{i=1}^{N} p(x_i) \frac{\log p(x_i)}{\log q(x_i)}$$
$$= E[\log p(x) - \log q(x)]$$



Generierung synthetischer Bilddaten

1. Ziehe zufällige Stichproben z_i aus der (approximativ) $N(0,1)$ - verteilten Repräsentation der Klasse (z.B. Handbilder) im latenten Raum
2. Rekonstruiere mittels des Decoders in der Bilddomäne die zugehörigen Bilder zu z_i



Zusammenfassung

- Durch Reduktion der Dimension der Ausgangsdaten werden im latenten Raum des Autoencoders die wichtigsten Features der Eingabedaten repräsentiert.
- Der konventionelle Autoencoder kann nicht als generatives Modell genutzt werden, da die Daten nur diskret den latenten Raum repräsentiert und keine Informationen vorliegen, die hier nicht explizit repräsentiert sind.
- Ein variationeller Autoencoder lernt im Training eine kontinuierliche Gauß-Verteilung, nach der die dimensionsreduzierten Daten im latenten Raum verteilt sind.
- Mittels Stichproben aus dieser Gauß-Verteilung lassen sich nun neue, plausible Daten bzw. Rekonstruktionen generieren.

Generative Adversarial Network - GAN

Einführung GANs

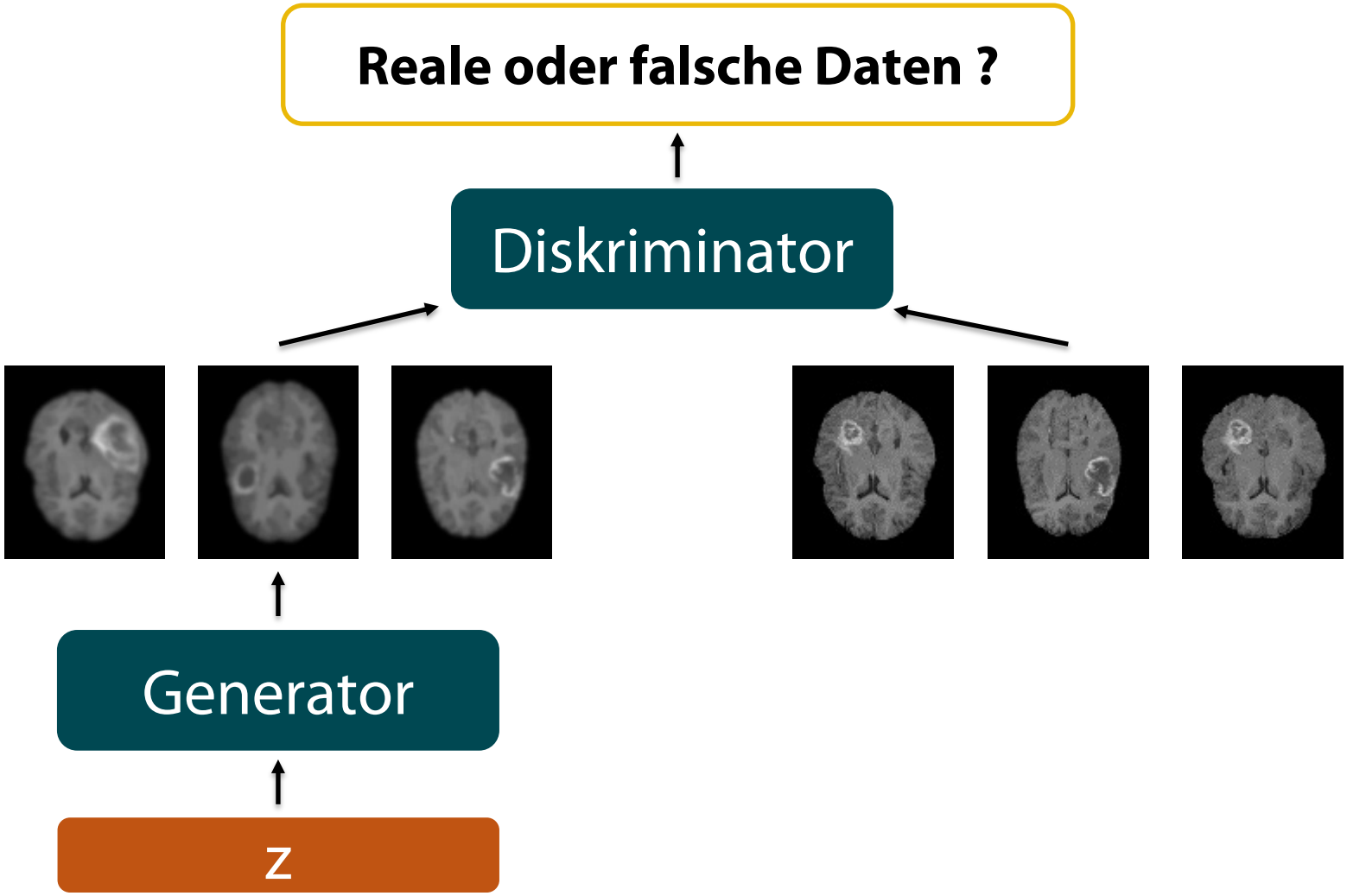
- **G**enerative **A**dversarial **N**etworks
  - Abk.: GANs
- GANs bestehen aus **zwei künstlichen neuronalen Netzwerken**, einem Generator- und einem Diskriminator-Netzwerk
- **Generator-Netzwerk**
  - Ähnlich wie ein Autoencoder erzeugt der Generator aus Vektoren aus einem latenten Raum Repräsentationen in dem gewünschten Ergebnisraum (z. B. künstliche Bilder)
  - Ziel des Generators: Lernen, Ergebnisse nach einer bestimmten Verteilung zu erzeugen.
- **Diskriminator-Netzwerk**
  - Der Diskriminator wird darauf trainiert, die Ergebnisse des Generators von den Daten aus der echten, vorgegebenen Verteilung von den künstlichen Daten, die der Generator erzeugt, zu unterscheiden. Er ist ein Klassifikator-Netzwerk mit 2 Klassen (echt oder fake).
- **Anwendungen:**
  - Generierung realistischer synthetischer Bilder
    - Beispiele siehe rechts
  - Modellierung von Bewegungsmustern
  - …
- Vorgestellt von Ian Goodfellow in 2014
  - Goodfellow, Ian, et al. "Generative adversarial nets." ***Advances in neural information processing systems***. 2014.

https://www.christies.com/features/A-collaboration-between-two-artists-one-human-one-a-machine-9332-1.aspx

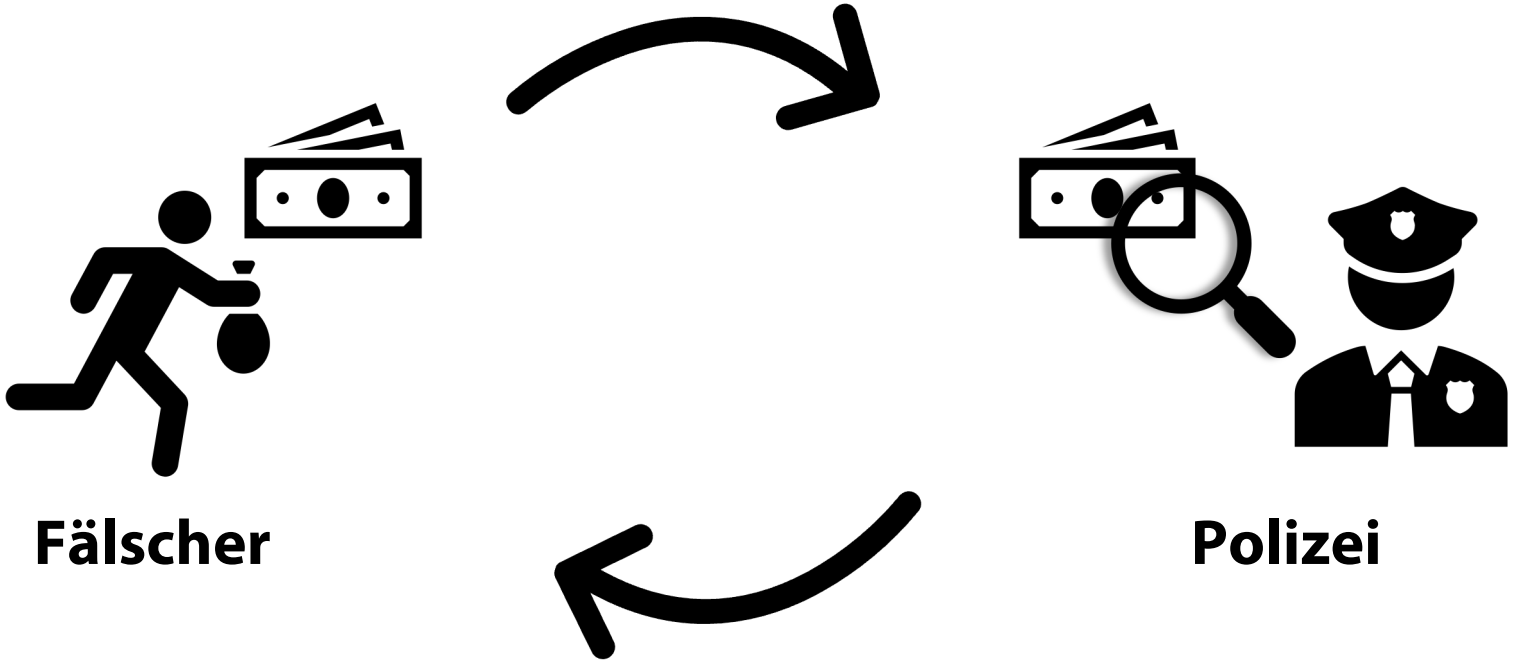
https://developers.google.com/machine-learning/gan

GAN-generierte, künstliche Malerei und künstliche Bilder von Gesichtern

Funktionsweise

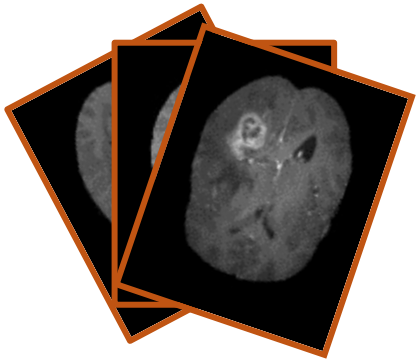


Analogie



Ziel des Generator-Netzwerks

- Generation realistischer, künstlicher Daten
  - Daten werden durch hochdimensionale Vektoren **z** repräsentiert.
  - Anforderung für die Generierung realistischer künstlicher Daten: Kenntnis der Wahrscheinlichkeitsverteilung p(**z**) der Vektoren **z**
    - z.B. Normalverteilung mit gegebenem Erwartungswertvektor und Kovarianzmatrix



Trainingsdaten

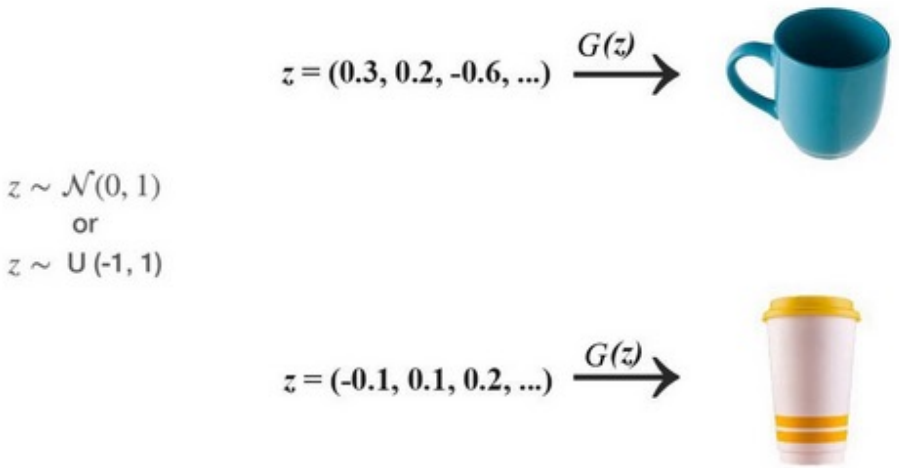
Normalverteilte oder gleichverteilte Vektoren

Komplexe nicht-lineare Funktion

Generierte Daten

Generator: Beschreibung

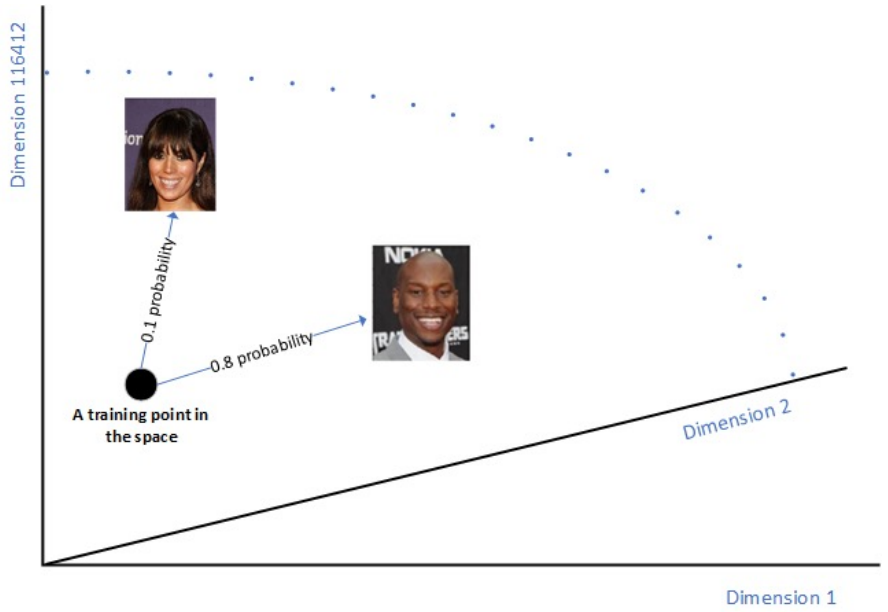
- Ein Generator realisiert eine Funktion, die ein Sample **z** aus einer bekannten Verteilung p(**z**) auf eine Ausprägung einer komplexen, hochdimensionalen Verteilung, z.B. bestimmter Bilder, pdata(x) abbildet.
- Während des Training erlernt der Generator den Zusammenhang zwischen den Vektoren **z** und der als Trainingsmenge verfügbaren realen Bildern bzw. zwischen der Verteilung p(**z**) und $p_{data}(x)$.



https://medium.com/@jonathan_hui/gan-whats-generative-adversarial-networks-and-its-application-f39ed278ef09

Beispiel: Porträtbilder im Generator

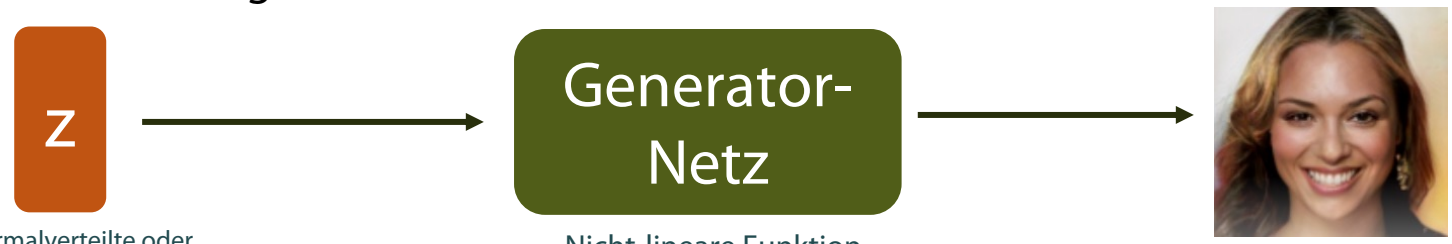
- Beispiel: Gesichtsbilder mit 218x178 Pixeln und 3 Farbkanälen (RGB).
- Jedes Bild wird als 218*178+3=116412 dimensionaler Vektor repräsentiert.
  - Im der Grafik sind nur 3 Komponenten aus dem 116412-dimensioanlen Raum dargestellt, wo jedes Bild durch einen Punkt repräsentiert ist.
- Die zugehörige, unbekannte Verteilungsfunktion $p_{data}(x)$ der Porträtbilder ordnet jedem Bild eine nicht-negativen Zahl zu, so dass die Summe aller dieser Zahlen 1 ergibt.
- Bem.: In diesem 116412-dim. Raum gibt es Bilder, die den im Beispiel betrachten Porträtbildern ähnlich sind, andere sind ihnen vollkommen unähnlich. Die Ähnlichkeit von Bildern zueinander wird durch $p_{data}(x)$ modelliert.



https://towardsdatascience.com/generative-adversarial-network-gan-for-dummies-a-step-by-step-tutorial-fdefff170391

Grundidee des Generators

- Generiere eine bekannte, einfache Verteilung p(**z**)
  - Bsp. Porträtbilder: Generiere 116412 gleichverteilte oder normalverteilte, unkorrelierte Zufallsvariablen und erhalte so 116412-dim. Vektoren **z**
- Suche nun mithilfe des Generators in dem Lernprozess nach einer Transformation zwischen den Ausprägungen **z** der bekannten Verteilung und Ausprägungen der unbekannten Verteilung $p_{data}(x)$ der zu generierenden Bilder.
- Hierbei bilden die Bilder der Trainingsmenge eine Stichprobe aus $p_{data}(x)$
  - Bsp.: Nach dem Training kann der Generator aus dem Input einer einfachen 116412-dimensionalen, gleichverteilten Zufallsvariablen eine andere 116412-dimensionale Zufallsvariable generieren, die der Verteilung der Porträtbilder P(X) folgt.

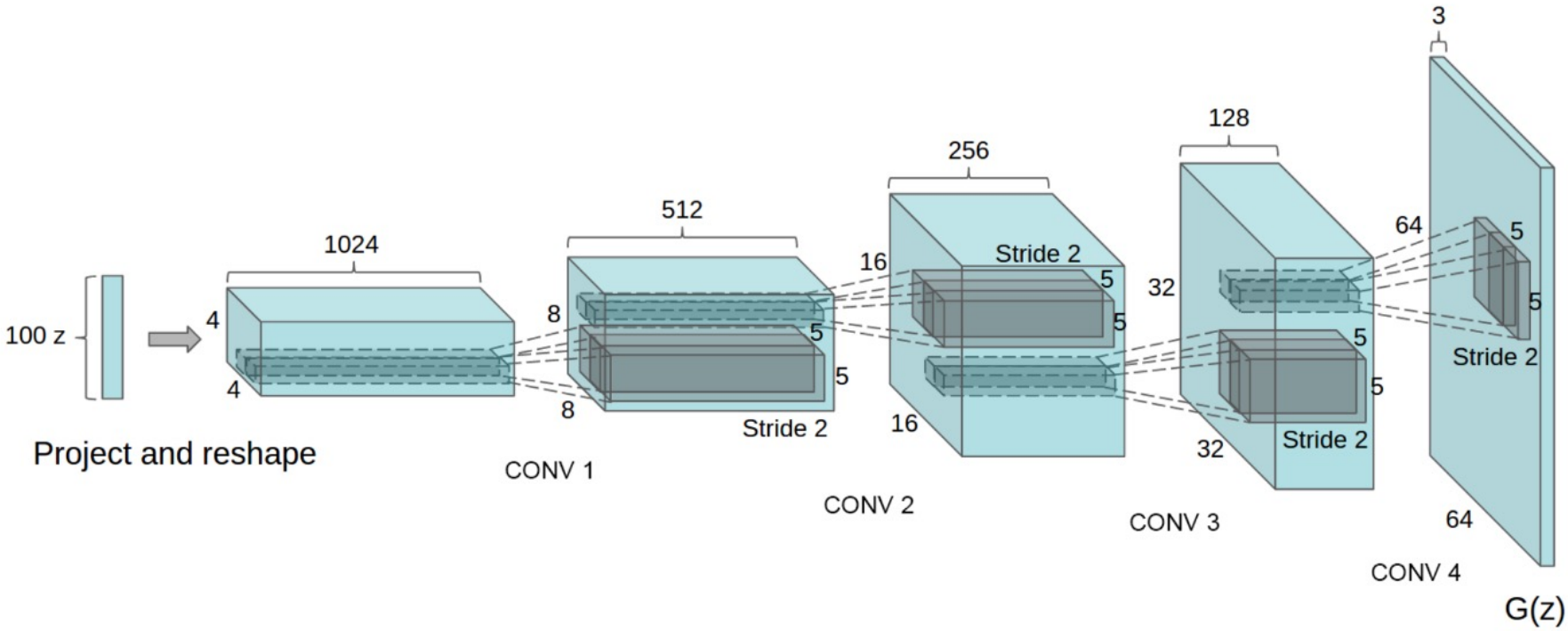


Normalverteilte oder gleichverteilte Vektoren

Nicht-lineare Funktion

P(X): Verteilung der zu generierenden Bilder

Beispiel: Generator-Netzwerk eines GANs (DCGAN)



Quelle: https://arxiv.org/pdf/1511.06434.pdf

Training eines GANs

- Während des Trainings sind der Generator G und Diskriminator D aus Sicht der Spieltheorie Gegner in einem Minimax Spiel
  - Annahme: Gegner hat optimale Strategie
  - Nash-Equilibrium: G generiert Daten, die den Trainingsdaten so sehr ähneln, dass D nur raten kann.
- Ziele:
  - Der Generator wird so trainiert, dass der finale Klassifikationsfehler des Diskriminators bei der Erkennung von echten und künstlich generierten Bildern **maximal** wird.
  - Der Diskriminator wird so trainiert, dass der Klasssifikationsfheler **minimiert** wird.
    - Bemerkung: Aufgrund dieser gegenläufigen Zielsetzungen des generators udn des Disktriminatoros man von **Adversarial Networks** (dt.: gegnerische Netzwerke)

Training des GANs: Zielfunktion

- Generator und Diskriminator werden abwechselnd, aber in einem Lernprozess trainiert.
  - Separates Training eines Generators würde nur zufällige Bilder generieren.
  - Der Generator profitiert von der Rückkopplung durch den Diskriminator, durch den gesteuert wird, welche Art von Bildern generiert werden sollen.

- Zielfunktion bzw. Optimierungskriterium

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

- Generator G , Diskriminator D
- Reale Trainingsdaten (Bilder) $\boldsymbol{x}$ der Verteilung p_{data}
- Eingabedaten des Generators $\boldsymbol{z}$ der bekannten Verteilung p_z

Zielfunktion des Trainings

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

Ergebnis des Diskriminators für reale Daten

Ergebnis des Diskriminators für generierte Daten $G(z)$

- Diskriminator maximiert: $D(x) \to 1, D\big(G(z)\big) \to 0$
- Diskriminator minimiert: $D\big(G(z)\big) \to 1$

Training

- Das Training von Generator und Diskriminator findet abwechselnd statt.
- Der Diskriminator maximiert (Gradientenaufstieg):

$$E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log 1 - D(G(z))]$$

- Der Generator minimiert (Gradientenabstieg):

$$E_{z\sim p_z(z)}[\log 1 - D(G(z))]$$

- Bem.: Der vordere Teil $E_{x\sim p_{data}(x)}[\log D(x)]$ entfällt bei der Optimierung von G, da nicht von G abhängig

Trainingsalgorithmus

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

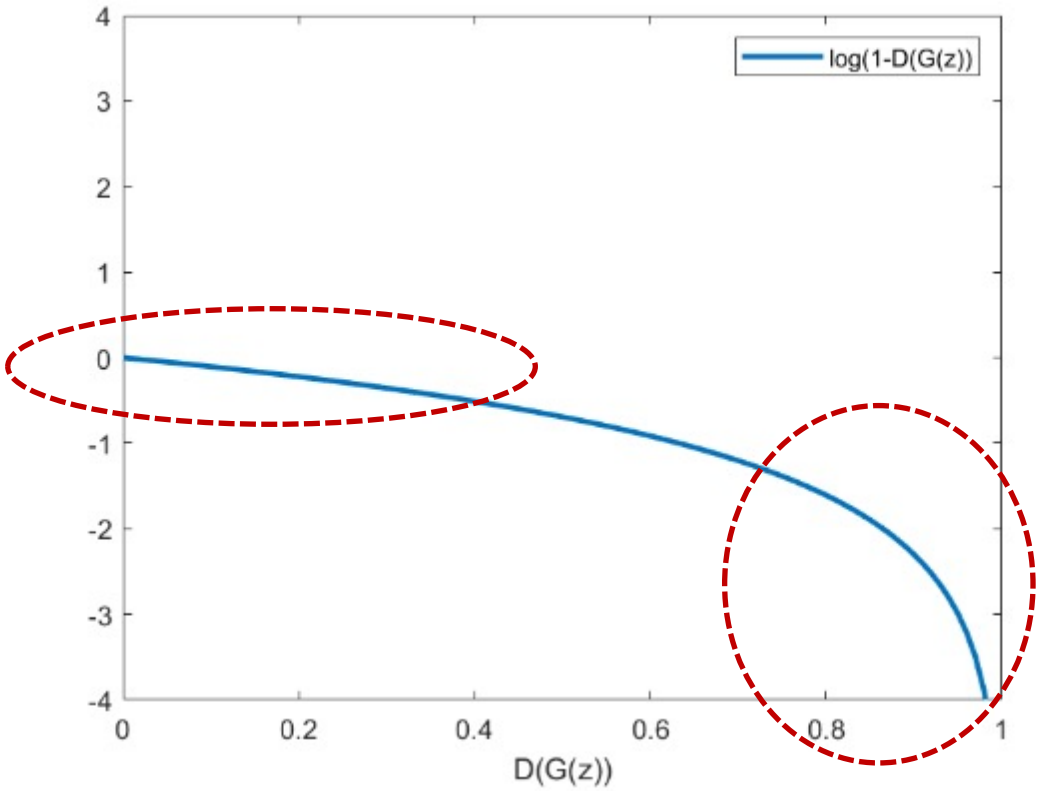
end for

Goodfellow, Ian. "NIPS 2016 tutorial: Generative adversarial networks." *arXiv preprint arXiv:1701.00160* (2016).

Fehlerfunktion des Generators

- Ist für Umsetzung ungeeignet → „Vanishing Gradient“ Problem

Flacher Gradient, wenn Generator schlecht ist.
→ **G lernt nicht!**

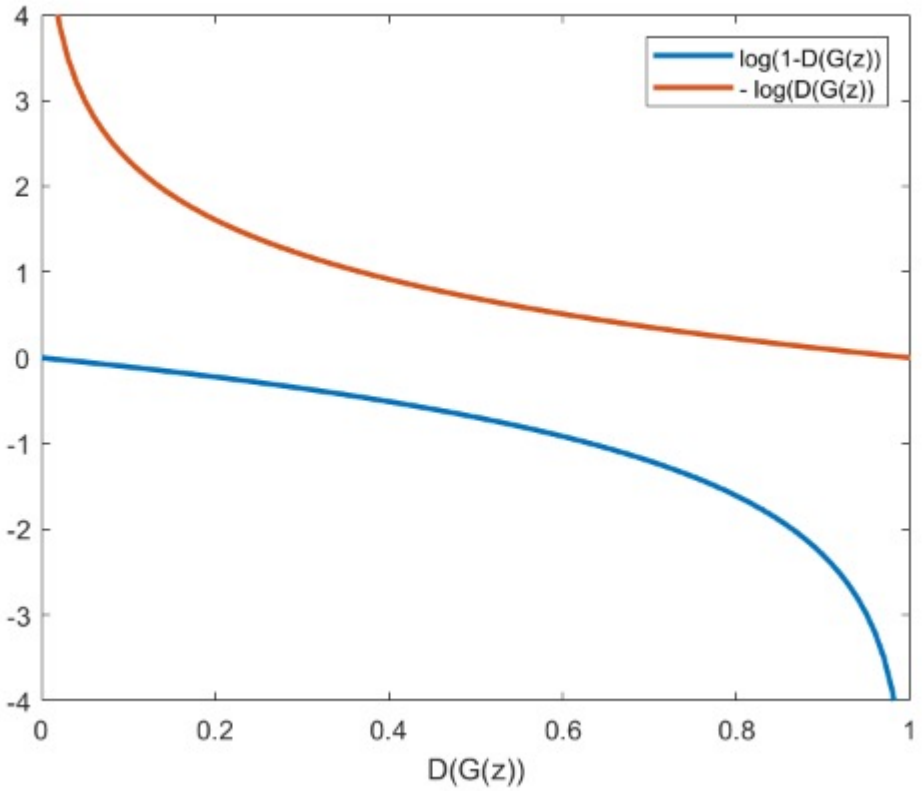


Generator ist bereits gut .
→ **starker Gradient „verschwindet“**

Li,F. et al. „Generative Models“ Convolutional Neural Networks for Visual Recognition (2017).

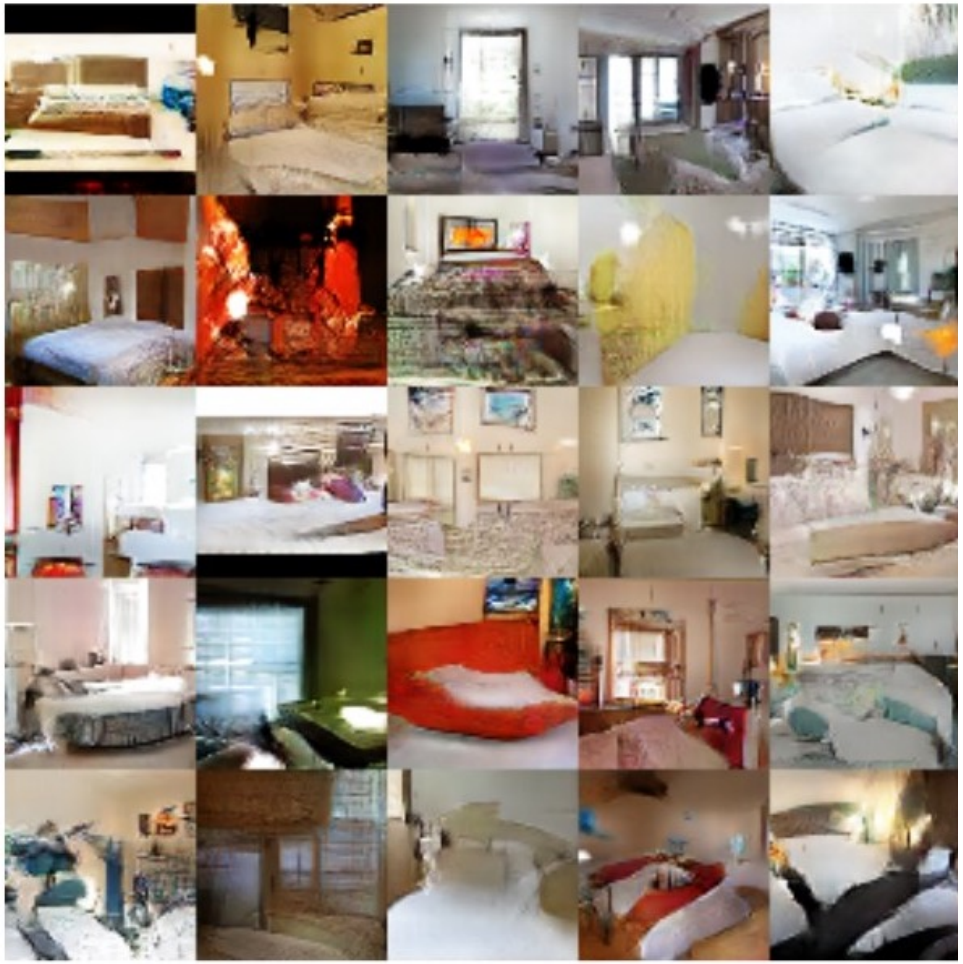
Fehlerfunktion des Generators

- Stattdessen: Maximieren von $E_{z\sim p_z(z)}\left[D\left(G(z)\right)\right]$



Li,F. et al. „Generative Models“ Convolutional Neural Networks for Visual Recognition (2017).

Beispiel Ergebnisse



[1] Goodfellow, Ian. "NIPS 2016 tutorial: Generative adversarial networks." *arXiv preprint arXiv:1701.00160* (2016).
[2] Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." *arXiv preprint arXiv:1511.06434* (2015).

Probleme

- „Gradient Vanishing“ Problem:
  - Diskriminator ist „zu gut“.
  - Effekt: der Diskriminator gibt nicht genügend Informationen an den Generator (Backpropagation) zurück, sodas dieser nichts Wesentliches dazu lernt.

- „Mode Collaps“:
  - Generator lernt ausschließlich einen „Mode“, welchen der Diskriminator am ehesten als real einstuft.

- „Non-Convergence“:
  - Oszillation des Gradienten destabilisiert das Modell.

Mode Collapse

- Daten sind multimodal, d.h. Daten haben eine finite Anzahl an Ausprägungen bzw. Modes
  - Z.B. Datenset MNIST hat 10 Modes (Ziffern 0 bis 9)

Alle 10 Modes werden generiert

Nur Mode „6“ wird generiert

Bei einem „Mode Collapse“ generiert G nur wenige bzw. einen Mode.

https://cedar.buffalo.edu/~srihari/CSE676/22.3-GAN%20Mode%20Collapse.pdf

Mode Collapse

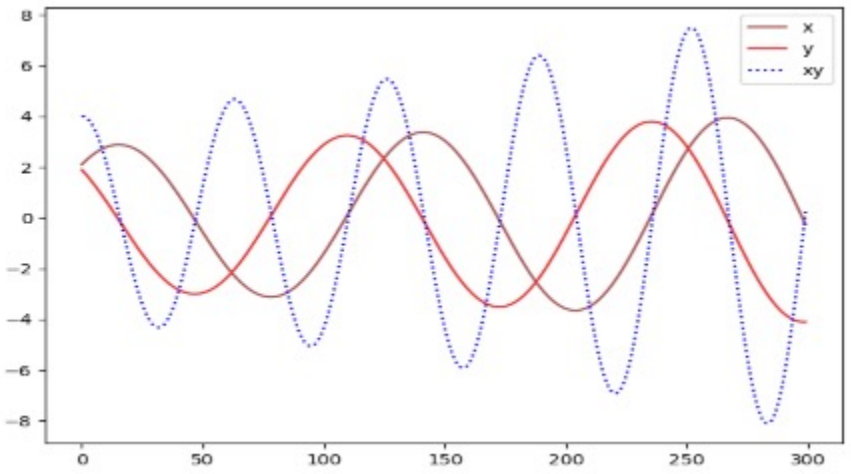
- Wenn der Generator trainiert wird, ohne dass der Diskriminator ge-updated wird, konvergiert der Generator zu einem Mode, welchen er besonders real darstellen kann und überlistet so den Diskriminator.
- Wird nun der Diskriminator ge-updated, lernt er nicht mehr, ob die generierten Daten real bzw. falsch sind , sondern nur das dieser bestimmte Mode falsch ist.

  → alle anderen Modes werden als real eingestuft.
- Der Generator lernt einfach den nächsten Mode um so wieder den Diskriminator zu überlisten.

Katz-und-Maus-Spiel von G und D statt Konvergenz zum Nash-Equilibrium

Non-Convergence

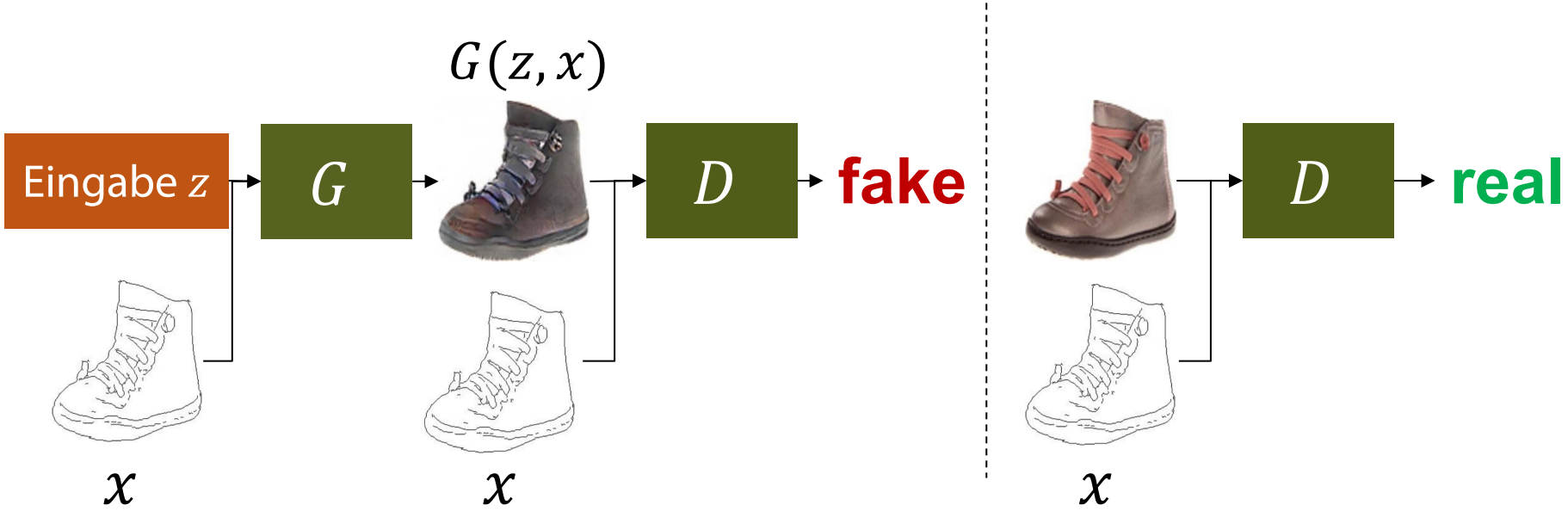
- Kommt der Diskriminator in die Nähe des gesetzten Ziels (50/50 raten), kann kein sinnvolles Feedback (Backpropagation) an den Generator gegeben werden.
  - Wird der Generator dennoch weiter trainiert, kann dieser sich wieder verschlechtern, worauf der Diskriminator wieder sicherere Entscheidungen treffen kann, usw.
  - Das kann sich unendlich wiederholen, so dass eine Instabilität des Modells gegeben ist.
  - Der Lernprozess konvergiert nicht.



Oszillation der Zielfunktion (hier xy)
→ keine Konvergenz

Conditional GAN

- Zusätzlich zu z erhält der Generator G eine **Kondition** (z.B. Kantenbild)
- Der Generator lernt die Kantenbilder als Grundlage für das **eigentliche Ergebnis**, hier ein fotorealistisches Objekt, zu nutzen.
- Der Diskriminator D lernt reale **Paare** (Eingabe + Kondition) von generierten Paaren zu unterscheiden.



Isola, P. et. al.: "Image-to-Image Translation with Conditional Adversarial Networks",ICCV, 2017

Pix2Pix

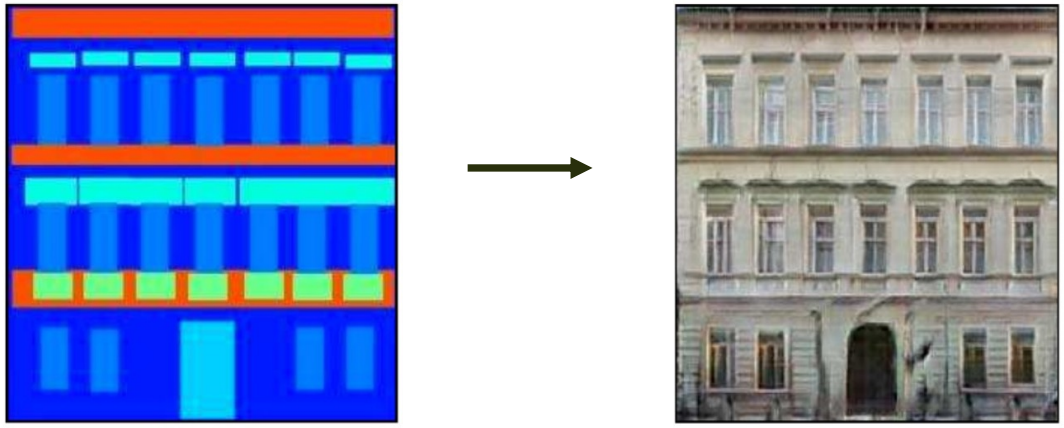
- Setzen die „Image-to-Image-Translation“ mittels Conditional GANs um.
- Der Generator lernt eine Funktion um eine Bilddomäne (z.B. Kantenbild) auf eine andere (z.B. Foto) abzubilden.
- Die Autoren verwenden als Generator das U-Net und als Diskriminator ein PatchGAN.

https://towardsdatascience.com/pix2pix-869c17900998

Pix2Pix

Fotorealistische Aufnahme → Karte

Labelbild → fotorealistische Fassade



Isola, P. et. al.: "Image-to-Image Translation with Conditional Adversarial Networks",ICCV, 2017

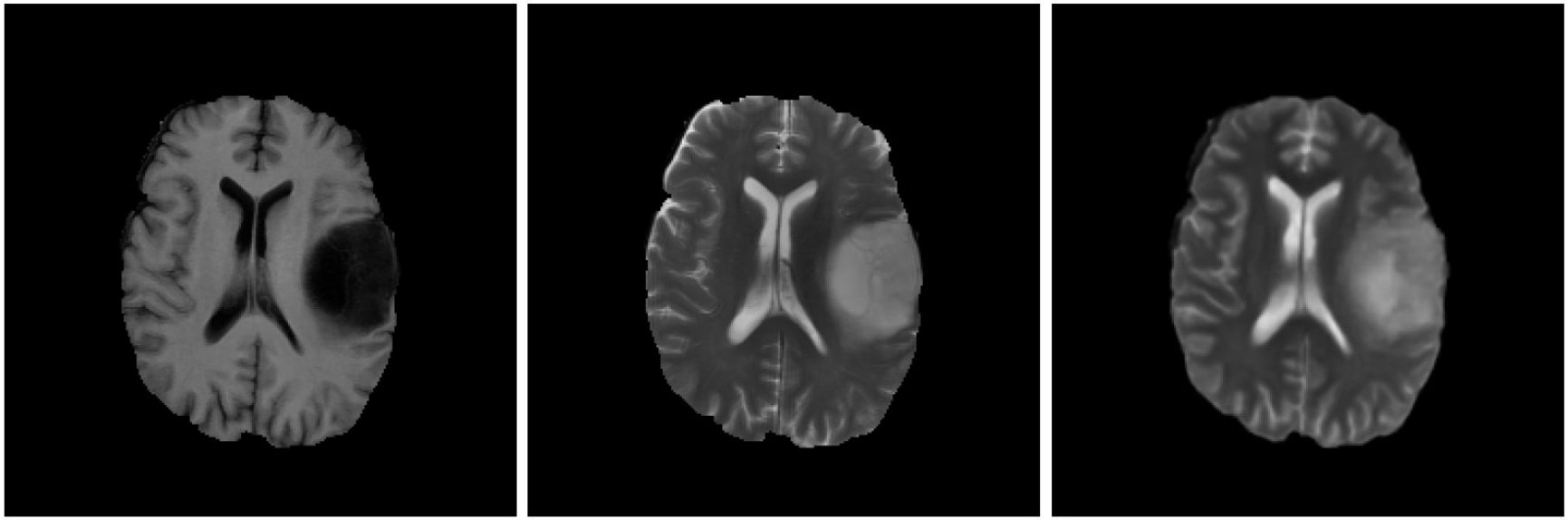
GANs: Medizinische Anwendung

- Beispiel: Translation von T1- gewichteten MRT-Bildern zu T2-gewichteten MRT-Bildern zur Ergänzung fehlender Bildkanäle in multispektralen MRT-Bilddaten
- Methode: Schaltung mehrerer GANs hintereinander um iterativ hochdimensionale Daten zu generieren.

Eingabe: T1-gewichtetes MR-Bild

Grundwahrheit: Reales T2-gewichtetes MR-Bild

Generiertes, synthetisches T2-gewichtetes MR-Bild



Uzunova, Hristina, et al. "Multi-scale GANs for Memory-efficient Generation of High Resolution Medical Images." International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer, Cham, 2019.

Cycle-GAN

- Das Cycle GAN ist in der Lage “Image-to-Image-Translation” (oder auch “Domain Transfer”) **ohne korrespondierende** Trainingspaare zu erlernen.

Fotorealistische Aufnahme

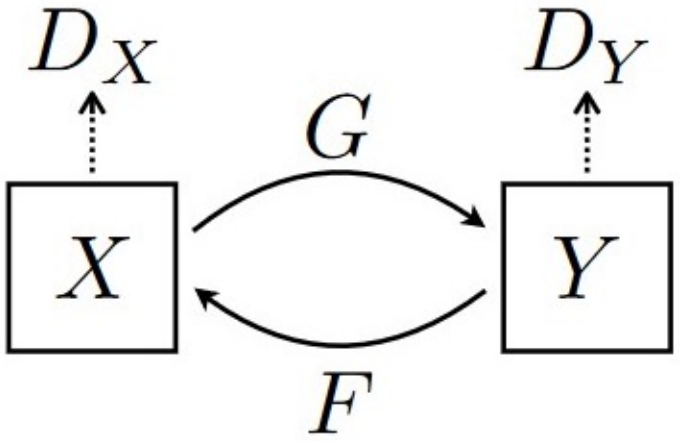
The Seine at Argenteuil, Claude Monet,1873

Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." *Proceedings of the IEEE international conference on computer vision*. 2017.

Cycle-GAN

- Besteht aus:
  - Generator G bildet Daten der Domäne X auf die Domäne Y ab
  - Generator F bildet Daten der Domäne Y auf die Domäne X ab
  - Diskriminatoren D_X bzw. D_Y unterscheiden zwischen echten und falschen Daten der jeweiligen Domäne

- Zusätzlich zum GAN-Loss wird ein Consistency Loss“ optimiert
  - L1-Norm um Rekontruktionsfehler von $x \rightarrow y \rightarrow \hat{x}$ und $y \rightarrow x \rightarrow \hat{y}$ zu minimieren



$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1]$$

Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." *Proceedings of the IEEE international conference on computer vision*. 2017.

Zusammenfassung

- GAN ist ein generatives Modell, welches durch das Spiel zweier Gegner lernt.
- Der Generator versucht den Diskriminator zu betrügen.
- Der Diskriminator versucht die falschen Daten zu enttarnen.
- Das Training von GANs ist anspruchsvoll und ist Mittelpunkt weiterer Forschung.