Vorlesung 09: Deep Generative Models

BA-INF 153: Einführung in Deep Learning für Visual Computing

Prof. Dr. Reinhard Klein

Nils Wandel

Informatik II, Universität Bonn

26.06.2024

Wiederholung

Letzte Woche: Autoencoder

1. Architektur
2. Datenkompression und Information Retrieval
3. Regularisierte Autoencoder
4. Denoising Autoencoder
5. Variational Autoencoder
6. (U-Net)

Mit dem VAE haben wir unser erstes "echtes" generatives Modell kennengelernt.
$\Rightarrow$ Heute betrachten wir weitere wichtige Konzepte für generative Modelle.

Motivation: This Person Does Not Exist

Die Website https://thispersondoesnotexist.com/ zeigt bei jedem Refresh ein anderes fotorealistische Bild eines Gesichts, dass mit einem Neuronalen Netzwerk generiert wurde.

Figure: Bild eines Gesichts, das mit StyleGAN [**karras2019style**] erzeugt wird [Quelle].

Die Bilder sind mit eine Generative Adverserial Network (GAN) erzeugt worden. Das Paper "A Style-Based Generator Architecture for Generative Adversarial Networks" findet ihr [hier].

Motivation: Dall-E 2

"DALL-E 2 kann originelle, realistische Bilder und Grafiken aus einer Textbeschreibung erstellen. Es kann Konzepte, Attribute und Stile kombinieren." [Quelle]

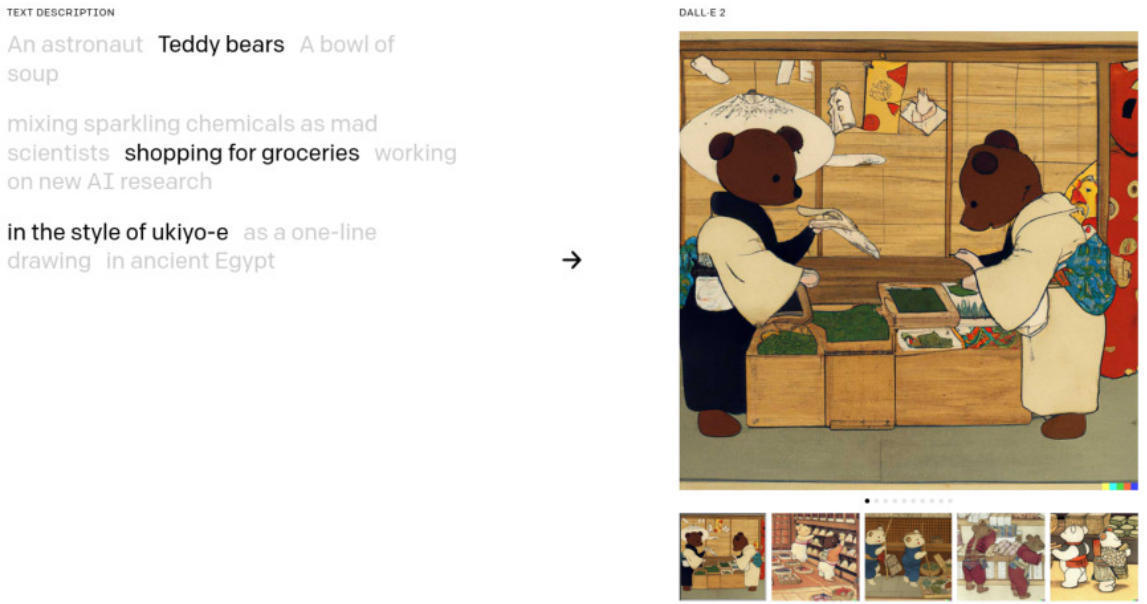


Figure: Eine Textbeispiel und dass von DALL-E 2 generierte Bild. [Quelle]

DALL-E 2 kombiniert eine Neuronale Netzwerk zur Textverarbeitung mit einem Diffusion-Modell um Bilder zu generieren.

Inhalt dieser Vorlesung

In dieser Vorlesung werden die Grundlagen von weiteren aktuellen generativen Modellen vorgestellt.

- Autoregressive Modelle
- Generative Adversarial Networks
- Flow-based Modelle
- Diffusion Modelle

Die grundlegenden Konzepte finden in unterschiedlichen Forschungsbereichen Anwendung und natürlich auch im Bereich Visual Computing. In dieser Vorlesung beschränken wir uns auf die Anwendung zum generieren von Bildern. [1]

[1] Themen die wir hier nicht besprechen: Energy-Based Models, Restricted Boltzmann Machines und fortgeschrittene Architekturen/Varianten der obigen Methoden.

Abschnitt 1

Autoregressive Modelle

Autoregressive Modelle

Bei einem Autoregressive Modell wird ein Bild $\mathbf{x} \in D^{n \times n}$ als eine Sequenz $x_1, ..., x_{n^2} \in D$ der einzelnen Pixeln interpretiert. Die Menge D sind alle Werte die von einem Pixel angenommen werden kann. Für die übliche Farbtiefe von 8-bits für ein Graustufenbild wäre die Menge $D := \{0, 1, ..., 255\}$.

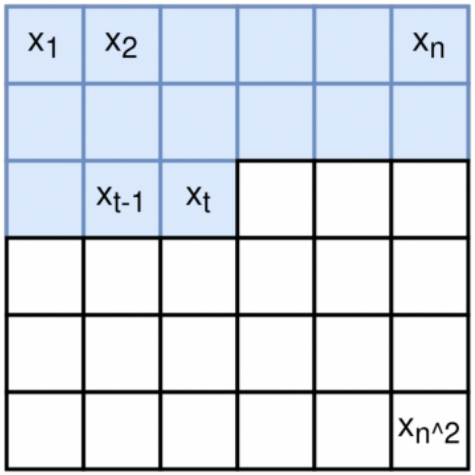


Figure: Bei autoregressive Modellen werden die Bilder Zeile für Zeile als Sequenz aus Pixeln darstellt.

Die Grundannahme bei autoregressive Modellen ist dass der Wert eines Pixels x_t nur von seinen Vorgängern $x_{t-1}, x_{t-2}, ..., x_1$ abhängt.

Training - Autoregressive Modelle

Mit dieser Annahme kann die Wahrscheinlichkeit $p(x)$ der Observationen $\{x^{(i)}\}_{i\in[1:m]}$ effizient als bedingte Wahrscheinlichkeit berechnet werden

$$p(x) = \prod_{t=1}^{n^2} p(x_t \mid x_{t-1}, x_{t-2}, ..., x_1).$$

Angenommen wir haben ein Netzwerk $f_\theta : D^{n\times n} \to \mathbb{R}^{n\times n\times 256}$, bei dem der Output $f_\theta(x)_{ij}$ nur von bereits generierten Pixeln x_{kl} mit $k \cdot n + l < i \cdot n + j$ abhängt. Wenn auf den Output eines solchen Netzwerks f_θ die Softmax-Aktivierung angewendet wird, erhalten wir die Verteilung $p_\theta(x_t \mid x_{t-1}, x_{t-2}, ..., x_1)$ über die Pixelwerte für alle Pixel x_t wobei $t \in [1 : n^2]$. Das Modell wird dann trainiert indem die negative log-likelihood minimiert wird:

$$\theta = \underset{\theta}{\operatorname{argmin}} \; -\sum_{t=1}^{n^2} \log p_\theta(x_t \mid x_{t-1}, x_{t-2}, ..., x_1).$$

PixelCNN

Ein Netzwerk $f_\theta : D^{n \times n} \to \mathbb{R}^{n \times n \times 256}$ lässt sich mit Hilfe von Masken mit Konvolutions-Layern darstellen. Allerdings führt die Verwendung von Masken bei einer hierarchischen Architektur zu blinden Flecken. In der Literatur hat sich das Aufteilen des Receptive Fields in einen "vertikalen" und "horizontalen" Teil etabliert.

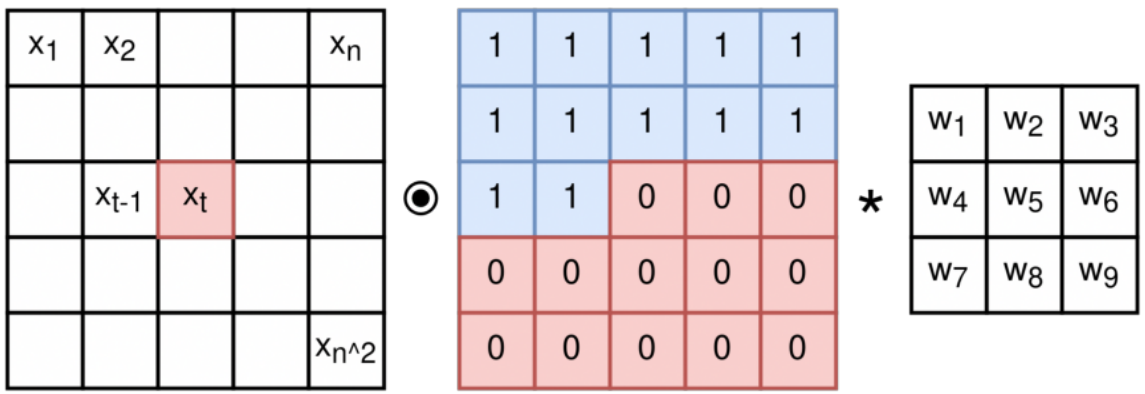


Figure: Illustration der Konvolution mit Masken, die die Autoregressive Eigenschaft aufrecht erhalten. Beachte: Es wäre sehr ineffizient, während des Trainings für jeden Pixel das gesamte Netzwerk mit einer separaten Maske auszuwerten. ⇒ Können wir die Maskierung direkt in das CNN einbauen?

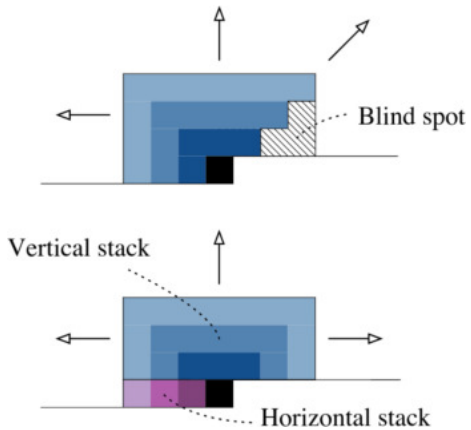


Figure: Illustration der Blindspots bei naiver CNN-Maskierung (oben) und der verikalen / horizotalen Konvolutionen in einem PixelCNN (unten) [**van2016conditional**]

Gated PixelCNN

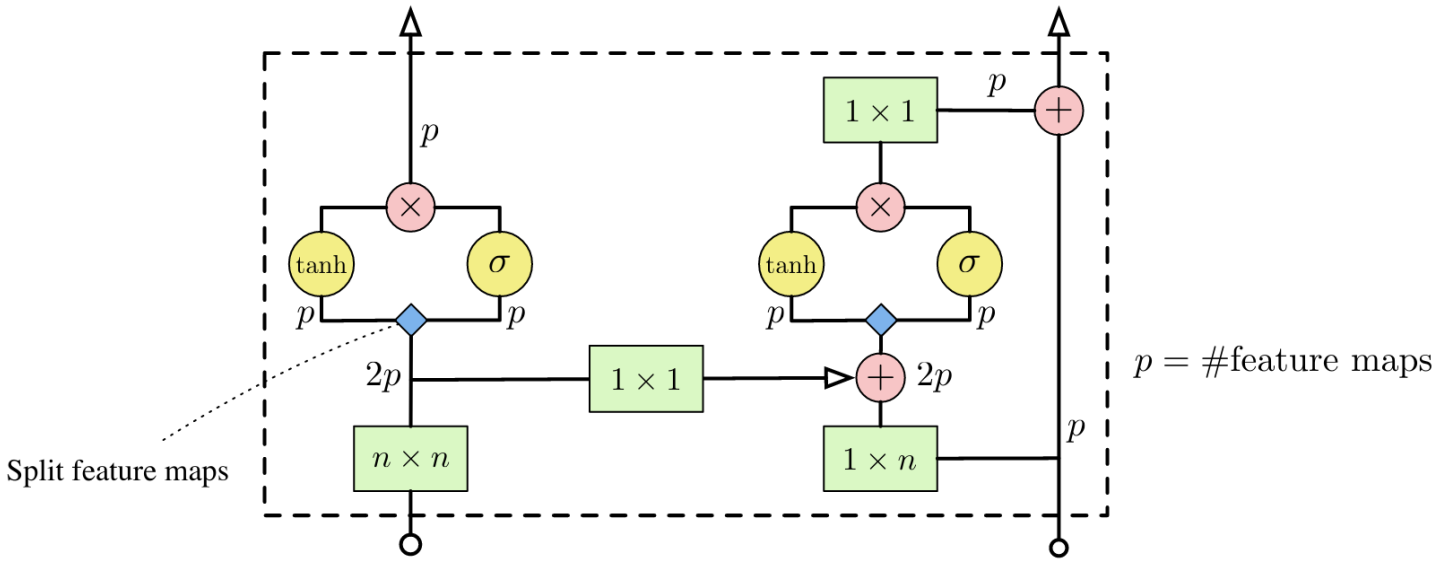


Figure: Architektur einer gated PixelCNN Unit mit vertical (links) und horizontal (rechts) stacks. [**van2016conditional**]

Die gated PixelCNN Architektur erlaubt dabei den Informationsfluss vom vertikalen zum horizontalen Teil (aber nicht umgekehrt damit die autoregressive Eigenschaft erhalten bleibt). Eine weitere Verbesserung wird durch Verwendung von Gated Convolution Units erzielt:

$$\tanh(W_{k,f} * x) \cdot \sigma(W_{k,g} * x)$$

wobei $W_{k,f}, W_{k,g}$ zwei Filter sind und $*$ die diskrete Kreuzkorrelation darstellt.

Inferenz - Autoregressive Modelle

[pmlr-v48-oord16]

Im Gegensatz zum Training des Netzwerks, erfolgt das genieren eines neuen Bildes sequentiell (also Pixel für Pixel).

Algorithm Pseudocode zum generieren Bilder mit einem Autoregressive Model

1: Initialisiere das Bild $\boldsymbol{x}$ mit 0.
2: **for** $i \in \{1, ..., n^2\}$ **do**
3: Berechne die Wahrscheinlichkeiten $p(\cdot \mid x_{t-1}, ..., x_1)$ mit PixelCNN.
4: Ziehe den Pixelwert $x_t \sim p(\cdot \mid x_{t-1}, ..., x_1)$.
5: **end for**

Bei Farbbildern werden die Farbkanäle eine Pixels ebenfalls als Sequenz dargestellt und sequentiell generiert.

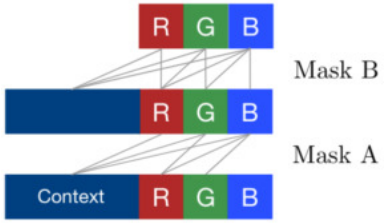


Figure: Verknüpfung der Farbkanäle eine Pixels in PixelCNN. **[pmlr-v48-oord16]**

Gated PixelCNN

Figure 7: Class-Conditional (multi-scale 64×64) samples from the Conditional Pixel CNN.

Figure: Results produced by PixelCNN. [**van2016conditional**]

Vorteile und Nachteile [2]

Vorteile

1. Kann sowohl für diskrete als auch kontinuierliche Daten verwendet werden.
2. Erlaubt es die Wahrscheinlichkeit, dass ein Bild zu dem Datensatz gehört, direkt zu berechnen.
3. Das Training eins PixelCNNs ist stabil.

Nachteile

1. Bilder müssen sequentiell generiert werden (sehr rechenintensiv).
2. Die Architektur muss die autoregressive Eigenschaft während des Training sicherstellen.

[2] Diese Einordnung der Methode bezieht auf die grundlegende Methode und ist nur eine grobe Einordnung. Es wird an allen vorgestellten Modellen aktiv geforscht.

Abschnitt 2

Generative Adversarial Networks (GAN)

Chapter 20.10.4 of [**Goodfellow-et-al-2016-Book**]

Generative Adversarial Networks (GAN)

GANs basieren auf einem spieltheoretischen Szenario, bei dem zwei Netzwerke miteinander konkurrieren.

Generator-Netzwerk

Erzeugt Stichproben $\boldsymbol{x} = G_{\theta_g}(\boldsymbol{z})$ aus zufälligen Rauschproben $\boldsymbol{z}$.

Discriminator-Netzwerk

Versucht, Stichproben aus den "echten" Trainingsdaten von "gefälschten" Generator-Stichproben zu unterscheiden und gibt $D_{\theta_d}(\boldsymbol{x})$ eine Wahrschlichkeit aus mit der $\boldsymbol{x}$ real ist.

Training a GAN

Objective

Die Zielfunktion des GAN ist die Binäre Entropie:

$$J_{\text{GAN}} = \log D_{\theta_d}(\boldsymbol{x}) + \log\left[1 - D_{\theta_d}(G_{\theta_g}(\boldsymbol{z}))\right].$$

Während des Trainings wird zwischen der Optimierung des Discriminators und Generators alterniert:

$$\theta_d{}^* = \underset{\theta_d}{\operatorname{argmax}}\ J_{\text{GAN}} \qquad \text{and} \qquad \theta_g{}^* = \underset{\theta_g}{\operatorname{argmin}}\ J_{\text{GAN}}$$

Konvergenz

Das Ziel der binären Entropie ermutigt den Diskriminator, den Unterschied zwischen echten und gefälschten Proben zu lernen, und den Generator, zu versuchen, den Diskriminator zu überlisten.

Das Netz konvergiert, wenn die Generatorproben nicht mehr von den realen Datenproben unterscheidbar sind, *d.h.* der Diskriminator immer $d = \frac{1}{2}$ ausgibt

Training a GAN

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

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:

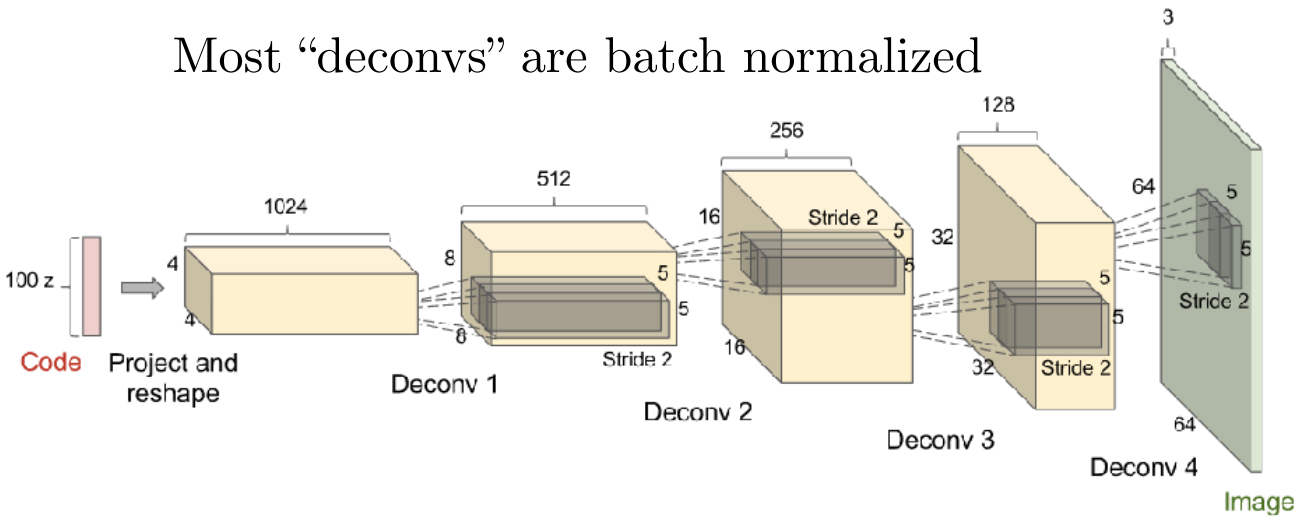
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Deep Convolutional GAN

DCGAN Architecture



(Radford et al 2015)

(Goodfellow 2016)

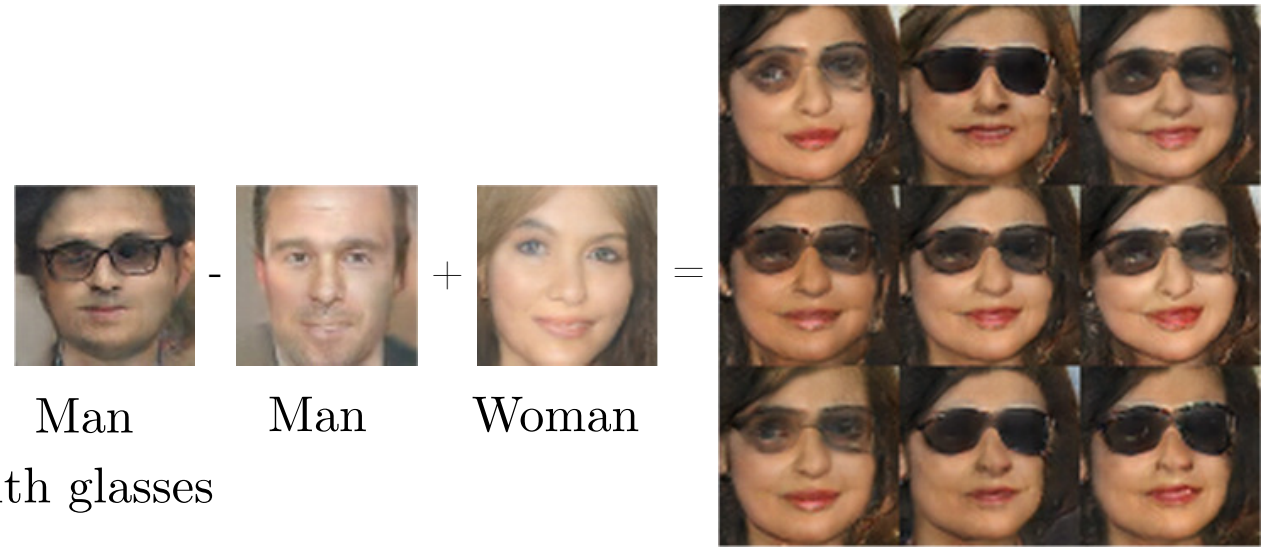
Figure: from Goodfellow

Synthesizing Images

Figure: Top: Samples generated by a GAN during training on CIFAR after 300, 900 and 5700 iterations. Bottom: Real images (left) and images generated by a GAN after 182000 training iterations (right). Images taken from [here], with excellent intuitive explanation of GANs and source code.

Deep Convolutional GAN

Vector Space Arithmetic



(Radford et al, 2015)

(Goodfellow 2016)

Figure: from Goodfellow

Vorteile und Nachteile [3]

Vorteile

- Erlaubt ein effizientes Generieren neuer Bilder.
- Erzeugt Bilder mit einem hohen Grad an Realismus.
- Das Konzept des "adversarial Loss" findet auch in zahlreichen anderen generativen Modellen Anwendung, um z.B. eine höhere Bildqualität zu erzielen.

Nachteile

- Höhere Trainingzeiten da notwendigerweise zwei Netzwerke trainiert werden.
- Durch die minimax Zielfunktion ist das Training eines GANs häufig instabil.
- GANs neigen zu mode collapse (niedrige Vielfältigkeit innerhalb der generierten Bilder).

[3] Diese Einordnung der Methode bezieht auf die grundlegende Methode und ist nur eine grobe Einordnung. Es wird an allen vorgestellten Modellen aktiv geforscht.

Abschnitt 3

Flow-based Modelle

Flow-based Generative Modelle

Idee in 1D

Können wir ein Netzwerk trainieren, welches eine Prior-Verteilung $p_\theta(z)$ direkt auf die Verteilung unserer zu generierenden Daten $p_\theta(x)$ abbildet?

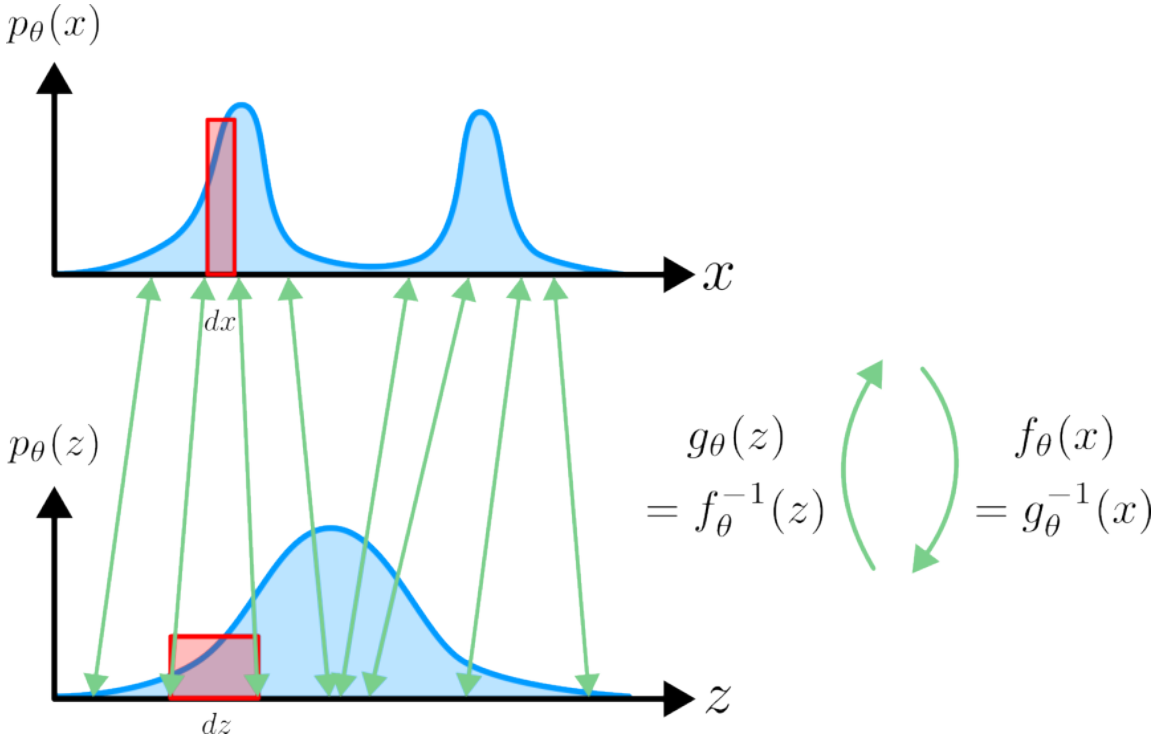


Figure: Transformation von Zufallsvariablen in 1D. $g_\theta(z)$ "generiert" neue Samples aus $p_\theta(x)$, indem es Samples von z auf x abbildet. Es ist wichtig, dass g_θ invertierbar ist ($g_\theta^{-1} = f_\theta$ bzw $f_\theta^{-1} = g_\theta$).

Flow-based Generative Modelle

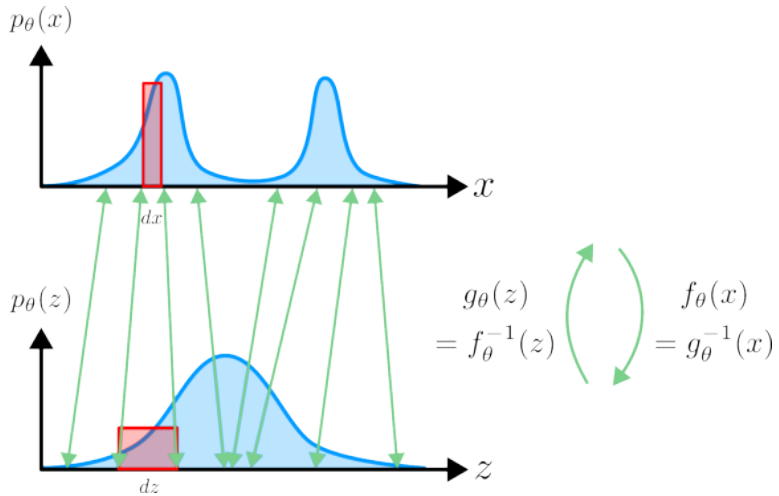


Figure: Transformation von Zufallsvariablen in 1D. Intuition: Die roten Flächen müssen gleich groß sein ($p_\theta(x)dx = p_\theta(z)dz$).

Die Prior-Verteilung können wir z.B. durch eine Normalverteilung vorgeben:

$$p_\theta(z) = \mathcal{N}(z|0, \mathbb{I})$$

Dadurch ergibt sich für die Verteilung über x durch das Theorem zur Transformation von Zufallsvariablen:

$$p_\theta(x) = \underbrace{p_\theta(z = f_\theta(x)) \cdot |f'_\theta(x)|}_{\text{in } 1D\ (f'_\theta(x) \approx \frac{dz}{dx})} = \underbrace{p_\theta(z = f_\theta(x)) \cdot |det(J_{f_\theta}(x))|}_{\text{in } nD\text{: verwende Determinante der Jacobian von } f_\theta}$$

Flow-based generative Modelle

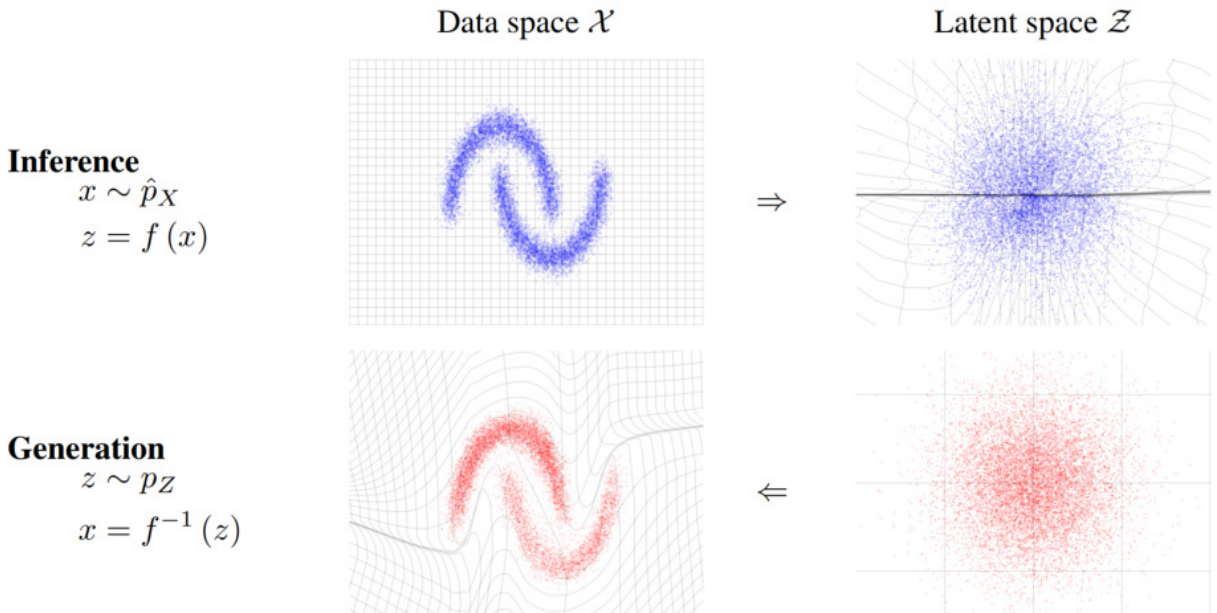


Figure 1: Real NVP learns an invertible, stable, mapping between a data distribution $\hat{p}_X$ and a latent distribution p_Z (typically a Gaussian). Here we show a mapping that has been learned on a toy 2-d dataset. The function $f(x)$ maps samples x from the data distribution in the upper left into approximate samples z from the latent distribution, in the upper right. This corresponds to exact inference of the latent state given the data. The inverse function, $f^{-1}(z)$, maps samples z from the latent distribution in the lower right into approximate samples x from the data distribution in the lower left. This corresponds to exact generation of samples from the model. The transformation of grid lines in $\mathcal{X}$ and $\mathcal{Z}$ space is additionally illustrated for both $f(x)$ and $f^{-1}(z)$.

Figure: Transformation von Zufallsvariablen in 2D. Beachte: Bei Bilddaten wird die Dimensionalität von x und z viel höher ($x, z \in \mathbb{R}^{h \times w}$). Beispiel aus "Density Estimation using Real NVP" by Dinh et al.

Flow-based generative Modelle

Das Ziel ist nun, die Evidenz der Daten x_i zu maximieren:

$$
\begin{aligned}
\theta^* &= \underset{\theta}{\operatorname{argmax}} \prod_i p_\theta(x_i) \\
&= \underset{\theta}{\operatorname{argmax}} \log(\prod_i p_\theta(x_i)) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_i \log(p_\theta(x_i)) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_i \log(p_\theta(z = f_\theta(x_i))) + \log(|det(J_{f_\theta}(x_i))|)
\end{aligned}
$$

$\Rightarrow$ Wie können wir ein invertierbares neuronales Netz erstellen, um g_θ bzw f_θ zu erhalten?
$\Rightarrow$ Und wie können wir damit effizient die Jacobi-Determinante $det(J_{f_\theta}(x))$ berechnen?

Flow-based Generative Modelle

[DBLP:journals/corr/DinhKB14]

Ein flow-based generatives Modell verwendet einen "Normalizing-Flow", der die Transformation der Prior Verteilung $p(z)$ in die Zielverteilung $p(x)$ modelliert. Ein "Normalizing-Flow" f ist eine Sequenz von invertierbaren Abbildungen f_i (Layern) zwischen Verteilungen:

- Der Flow ist eine Komposition $f = f_1 \circ f_2 \circ ... \circ f_n$.
- $f_i : \mathbb{R}^d \to \mathbb{R}^d$ sind invertierbar $\Rightarrow$ f ist invertierbar
- die Determinanten der Jacobi-Matrizen J_{f_i} der einzelnen Layer f_i sollten effizient berechnet werden können:
  $\Rightarrow det(J_f) = det(J_{f_1} \cdot ... \cdot J_{f_n}) = det(J_{f_1}) \cdot ... \cdot det(J_{f_n})$

Ein flow-based Model machte die Berechnung der Wahrscheinlichkeit $p_\theta(x_i)$ einer Observation $x^{(i)}$ handhabbar, in dem diese durch den "Normalizing-Flow" bijektiv auf eine "latent variable" z abgebildet wird, deren Verteilungen durch den Prior $p(z)$ definiert ist. Damit ergibt sich für θ^*:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_i \left(\log(p_\theta(z = f_\theta(x_i))) + \sum_j \log(|det(J_{f_{j,\theta}}(x_i))|) \right)$$

Affine Coupling Layer - Definition

Wie können wir ein invertierbares Netzwerk-Layer entwerfen, dessen Jacobian leicht berechenbar ist?

Sei $x \in \mathbb{R}^n$ mit Partition $x = (x_1, x_2)$ und seien $s, t : \mathbb{R}^d \to \mathbb{R}^d$ beliebige Funktionen sein. Die affine Kopplung $f : \mathbb{R}^n \to \mathbb{R}^n$ ist definiert als:

$$y_1 = x_1$$
$$y_2 = x_2 \odot \exp(s(x_1)) + t(x_1)$$

Das Affine Coupling Layer modelliert die Funktionen s, t mit einem kleinen Konv. Netzwerk.

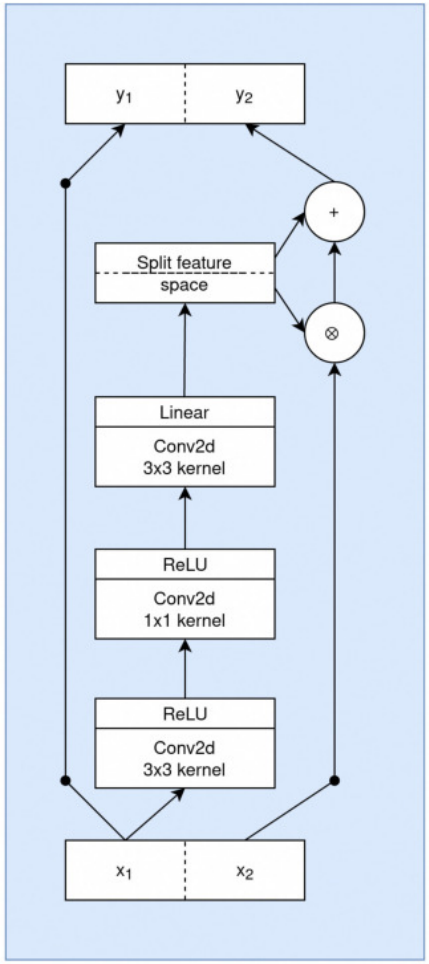


Figure: Diagramm des Affine Coupling Layer

Affine Coupling Layer - Inverse

Inverse

Die zweite Partition ist eine affine Transformation der Eingabe x_2 durch die Werte s, t, die nur von Partition x_1 abhängen. Beim Berechnen der Umkehrfunktion bzgl. der zweiten Partition, können s, t als konstanten behandelt werden. Da die erste Partition die Identität ist (und offensichtlich bijektiv) können s, t auch mit y_1 berechnet werden. Die Umkehrfunktion des Affinen Coupling Layer ist dementsprechend:

$$f : \mathbb{R}^n \to \mathbb{R}^n$$

$$\begin{aligned} y_1 &= x_1 \\ y_2 &= x_2 \odot \exp(s(x_1)) + t(x_1) \\ &= x_2 \odot \exp(s(y_1)) + t(y_1) \end{aligned}$$

$$f^{-1} : \mathbb{R}^n \to \mathbb{R}^n$$

$$\begin{aligned} x_1 &= y_1 \\ x_2 &= (y_2 - t(y_1)) \odot \exp(-s(y_1)) \end{aligned}$$

Affine Coupling Layer - Determinante

Determinante

Die Jacobimatrix eines Affine Coupling Layer lässt sich als Blockmatrix schreiben:

$$J_f = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} I & 0 \\ \frac{\partial y_2}{\partial x_1} & \text{diag}\left(\exp(s(x_1))\right) \end{bmatrix}$$

. Für eine 2x2-Blockmatrix $\det\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}\right)$ bei der entweder B=0 oder C = 0 folgt aus dem verallgemeinerten Entwicklungssatz für die Determinante:

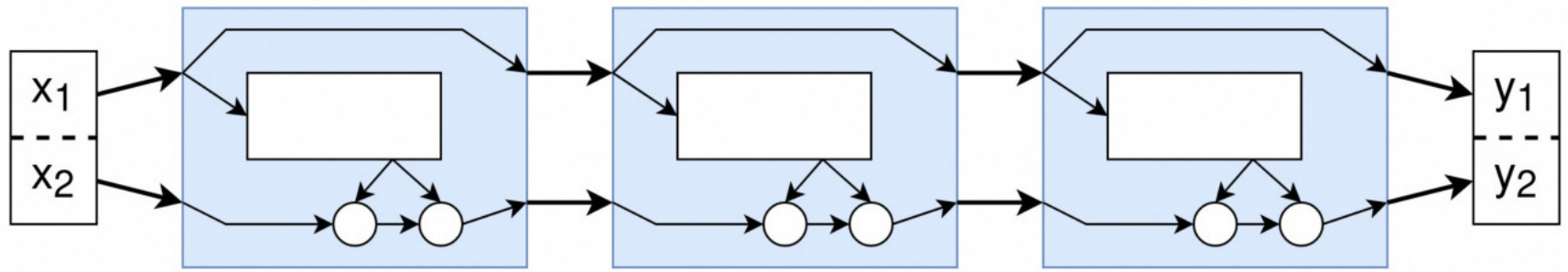
$$\det\left(\begin{bmatrix} A & 0 \\ C & D \end{bmatrix}\right) = \det\left(\begin{bmatrix} A & B \\ 0 & D \end{bmatrix}\right) = \det(A) \cdot \det(D)$$

Die Determinante der Identitätsmatrix (A) ist 1 und die Determinante einer Diagonalmatrix (D) das Produkt ihrer Diagonaleinträge. Dementsprechend ist Determinante der Jacobian des Affine Coupling Layer:

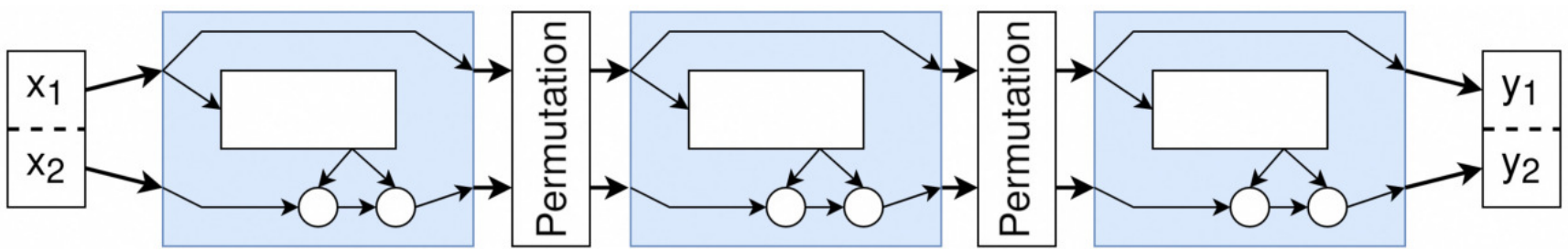
$$\det(J_f) = 1 \cdot \exp\left(\sum_j s(x_1)_j\right).$$

Coupling layer - IV

Das Netzwerk $f = f_1 \circ f_2 \circ ... \circ f_k$ ist eine Komposition der Coulping Layer $f_1, ..., f_k$. Allerdings führt ein einfaches Aneinanderreihen der Layer dazu, dass der "Normalizing-Flow" eine Identitätsfunktion bezüglich der ersten Partition x_1 ist.



Eine Permutation der Partitionen zwischen Layern verhindert, dass das Netzwerk für die Hälfte der Eingabe nur die Identitätsfunktion repräsentieren kann. Die Permutationen sind entweder nicht parametrische Operationen oder werden mit 1×1 Konvolutionen gelernt.



Inferenz von Flow-based Modellen

[kingma2018glow]

Die Inferenz neuer Bilder erfolgt in zwei Schritten:

1. Ziehe z entsprechend der a-priori Verteilung.
2. Generiere das Bild mit Hilfe das "Normalizing-Flows" $x = g(z) = f^{-1}(z)$.

Zur Verbesserung der Bildqualität kann aus der a-priori Verteilung mit reduzierter Temperatur $T \leq 1$ gesampled werden. Anstatt für $p_\theta(z)$ eine normierte Normalverteilung $\mathcal{N}(z|0,1)$ zu verwenden, wird dann eine Normalverteilung mit kleinerer Standardabweichung $\mathcal{N}(z|0,T)$ genommen.

Figure: Generierte Bilder für Temperaturen $0.25, 0.6, 0.7, 0.8, 0.9, 1.0$.

Vorteile und Nachteile [4]

Vorteile

- Erlaubt ein effizientes Sampling neuer Bilder.
- Gute Repräsentation der Modi der Datenverteilung (gesamplte Bilder weisen eine hohe Vielfältigkeit auf).
- Erlaubt das Berechnen von Latent Repräsentationen der Daten.
- Kann in andere Modelle integriert werden um einen gelernten Prior zu ermöglichen.

Nachteile

- Setzt voraus, dass die Netzwerkarchitektur invertierbar ist.
- Die Bildqualität ist geringer als bei GAN und Diffusion Modellen.
- Aktuellen Architekturen können nicht mit GANs konkurrieren obwohl sie deutlich mehr Parameter haben.

[4] Diese Einordnung der Methode bezieht auf die grundlegende Methode und ist nur eine grobe Einordnung. Es wird an allen vorgestellten Modellen aktiv geforscht.

Abschnitt 4

Diffusion-Modelle

Diffusion-Modelle

Der Grundgedanke von Diffusions-Modellen besteht darin, die Struktur einer Verteilung $p(x_0)$ durch einen iterativen Diffusionsprozess $q(x_t|x_{t-1})$ systematisch und langsam so zu verändern, dass sie zu einer (normalverteilten) Prior-Verteilung konvergiert. Gleichzeitig wird ein Netzwerk $p_\theta(x_{t-1}|x_t)$ trainiert, welches den umgekehrten Diffusionsprozess $q(x_{t-1}|x_t)$ lernen soll um die Struktur in den Daten iterativ wiederherzustellen.

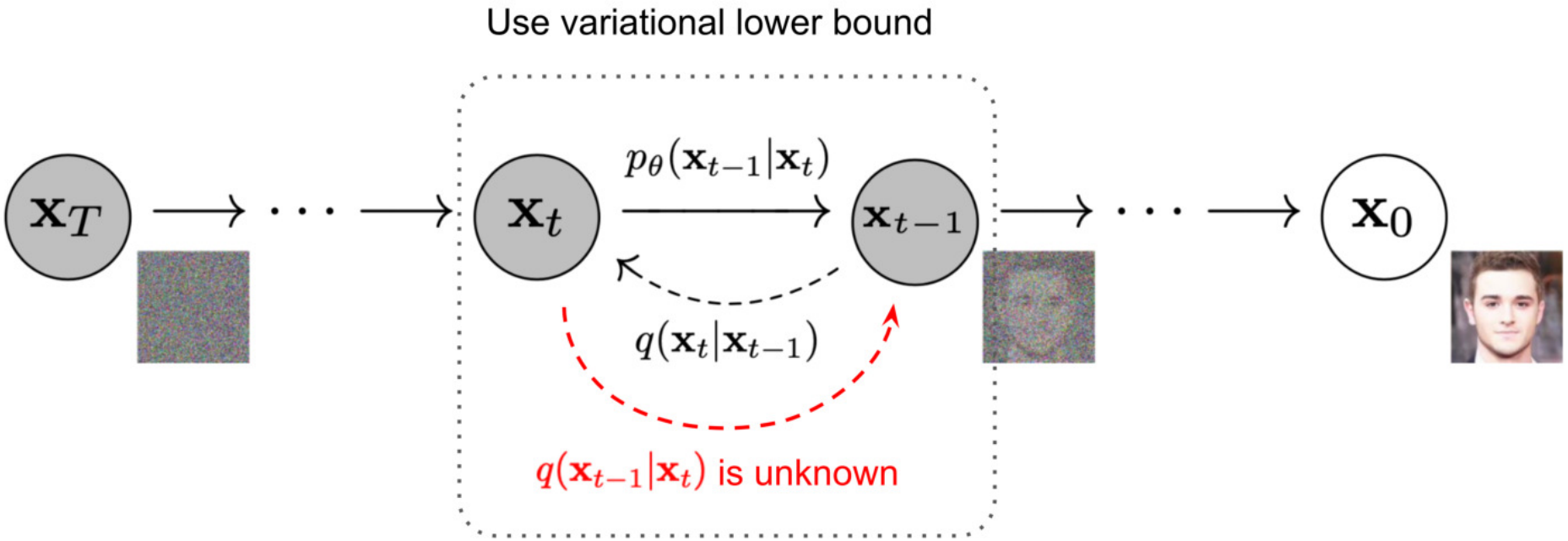


Figure: Beispielhafte Darstellung des Vorwärts und Rückwärts Diffusions Prozess [Quelle]

Vorwärts Diffusionsprozess

Ausgehend von einer Observation $x_0 \sim p(x)$ erzeugt der Vorwärtsdiffusionsprozess eine Folge zunehmend verrauschter Stichproben $x_1, ..., x_T$. Dabei wird bei jedem der T Schritte der Mittelwert verringert und normalverteiles Rauschen angewendet. Die Schrittgrößen des Diffusionprozesses werden durch einen "variance schedule" $\beta_t \in (0,1)_{t\in[1:T]}$ definiert, wobei β_t mit grösseren t ansteigt.

$$q(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1}),\ q(x_t \mid x_{t-1}) = \mathcal{N}(x_t \mid \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

Eine praktische Eigenschaft dieses Diffusionsprozesses ist, dass wir beliebig viele Schritte in einen Diffusionsschritt zusammenführen können: Sei $\alpha_t = 1 - \beta_t$ und $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$, dann ist: $q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$
Im fortschreitenden Prozess verliert die Stichprobe x_0 allmählich ihre erkennbaren Merkmale. Im Limit $T \to \infty$ geht $\bar{\alpha}_T \to 0$ und somit entsprechen Stichproben von x_T einer Standardnormalverteilung $\mathcal{N}(x|0, I)$.

Rückwärts Diffusionsprozess

Nun möchten wir den Diffusionsprozess umkehren und eine Stichprobe aus $q(x_{t-1} \mid x_t)$ nehmen. Damit können wir iterativ ein neues Sample $x_0 \sim p(x)$ aus einer latenten Variable $x_T \sim \mathcal{N}(0, I)$ generieren. Wenn β_t klein genug ist, wird $q(x_{t-1} \mid x_t)$ ebenfalls normalverteilt sein. Leider können wir $q(x_{t-1} \mid x_t)$ nicht einfach schätzen, da der gesamte Datensatz verwendet werden müsste.
In einem Diffusionmodell wird daher ein Neuronales Netzwerk $p_\theta(x_{t-1}|x_t)$ trainiert um diese bedingten Wahrscheinlichkeiten $q(x_{t-1}|x_t)$ anzunähern. Mit diesem Modell lässt sich dementsprechend der umgekehrten Diffusionsprozess wie folgt durchgeführen:

$$p_\theta(x_{0:T}) = p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t), \text{ mit: } p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1} \mid \mu_\theta(x_t, t), \sigma_t^2 I)$$

Hierbei wird $\mu_\theta(x_t, t)$ durch ein neuronales Netz gelernt und die Varianz σ_t^2 auf einen vorgegebenen Wert gesetzt (z.B. $\sigma_t^2 = \beta_t$), sodass sie nicht gelernt werden muss.

Zielfunktion

Das Ziel ist nun wieder die Evidenz $p_\theta(x_0)$ der Daten im Datenset zu maximieren, beziehungsweise den negativen logarithmus davon zu minimieren. Dabei können wir - wie bereits bei VAE gesehen - wieder auf den ELBO zurückgreifen:

$$\begin{aligned} -\log p_\theta(x_0) &\leq -\log p_\theta(x_0) + D_{KL}(q(x_{1:T}|x_0), p_\theta(x_{1:T}|x_0)) \\ &= -\log p_\theta(x_0) + \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log\left(\frac{q(x_{1:T}|x_0)}{p_\theta(x_{1:T}|x_0)}\right)\right] \end{aligned}$$

Hier können wir nun den Satz von Bayes anwenden:

$$p_\theta(x_{1:T}|x_0) = \frac{p_\theta(x_0|x_{1:T})p_\theta(x_{1:T})}{p_\theta(x_0)} = \frac{p_\theta(x_{0:T})}{p_\theta(x_0)}$$

$$\ldots = -\log p_\theta(x_0) + \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log(p_\theta(x_0)) + \log(q(x_{1:T}|x_0)) - \log(p_\theta(x_{0:T}))\right]$$

$$= \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log(\underbrace{q(x_{1:T}|x_0)}_{=\prod_{t=1}^{T} q(x_t|x_t-1)}) - \log(\underbrace{p_\theta(x_{0:T})}_{=p_\theta(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)})\right]$$

Zielfunktion

$$
\begin{aligned}
\ldots &= \mathbb{E}_{q(x_{1:T}|x_0)} \left[-\log(p_\theta(x_T)) - \sum_{t=1}^{T} \log(p_\theta(x_{t-1}|x_t) + \sum_{t=1}^{T} \log(q(x_t|x_{t-1})) \right] \\
&= \mathbb{E}_{q(x_{1:T}|x_0)} \left[-\log(p_\theta(x_T)) - \sum_{t=1}^{T} \log(\frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}) \right] \\
&= \mathbb{E}_{q(x_{1:T}|x_0)} \left[-\log(p_\theta(x_T)) - \sum_{t=2}^{T} \log(\frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}) - \log(\frac{p_\theta(x_0|x_1)}{q(x_1|x_0)}) \right]
\end{aligned}
$$

... in der Übung werden wir sehen, dass $q(x_t|x_{t-1}) = \frac{q(x_{t-1}|x_t,x_0)q(x_t|x_0)}{q(x_{t-1}|x_0)}$ ist.

$$
\begin{aligned}
\ldots = \mathbb{E}_{q(x_{1:T}|x_0)} [&-\log(p_\theta(x_T)) - \sum_{t=2}^{T} \log(\frac{p_\theta(x_{t-1}|x_t)q(x_{t-1}|x_0)}{q(x_{t-1}|x_t,x_0)q(x_t|x_0)}) \\
&- \log(\frac{p_\theta(x_0|x_1)}{q(x_1|x_0)})
\end{aligned}
$$

Zielfunktion

... in der Übung werden wir sehen, dass dies entspricht:

$$\ldots = \mathbb{E}_{q(x_{1:T}|x_0)} \left[\underbrace{-\log(\frac{p_\theta(x_T)}{q(x_T|x_0)})}_{A} \underbrace{-\sum_{t=2}^{T} \log(\frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t, x_0)})}_{B} \underbrace{-\log(p_\theta(x_0|x_1))}_{C} \right]$$

Lasst uns nun die Terme A, B, C genauer betrachten:

- A: Da $p_\theta(x_T) = \mathcal{N}(x_T|0, I)$ entspricht, ist dieser Term unabhängig von θ und kann ignoriert werden.
- C: Dieser Term entspricht einer negative log-likelihood für eine Normalverteilung und wird (wie bereits gezeigt) einen Least Square Loss für die Vorhersage des Means $\mu_\theta(x_1, 1)$ ergeben.
- B: Hier können wir die Summe aus dem Erwartungswert ziehen und erhalten:

$$B = \sum_{t=2}^{T} \underbrace{\int q(x_{1:T}|x_0)}_{\mathbb{E}_{q(x_{1:T}|x_0)}[\ldots]} \left(-\log(\frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t, x_0)}) \right) dx_{1:T}$$

Zielfunktion

- B: Hier können wir die Summe aus dem Erwartungswert ziehen und erhalten:

$$B = \sum_{t=2}^{T} \int q(x_{1:T}|x_0) \left(-\log \left(\frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t, x_0)} \right) \right) dx_{1:T}$$

Nun können wir $q(x_{1:T}|x_0)$ zerlegen in $q(x_{1:t-2}, x_{t+1:T}|x_{t-1}, x_t, x_0)q(x_t|x_0)q(x_{t-1}|x_t, x_0)$. Das Integral über $dx_{1:t-2}dx_{t+1:T}$ kann nun separat betrachtet werden und ergibt 1. Somit verbleiben wir mit:

$$\begin{aligned} B &= \sum_{t=2}^{T} \int q(x_t|x_0)q(x_{t-1}|x_t, x_0) \log \left(\frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} \right) dx_{t-1}dx_t dx_0 \\ &= \sum_{t=2}^{T} \int q(x_t|x_0) \left(\int q(x_{t-1}|x_t, x_0) \log \left(\frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} \right) dx_{t-1} \right) dx_t dx_0 \\ &= \sum_{t=2}^{T} \mathbb{E}_{q(x_t|x_0)} \left[D_{KL}(q(x_{t-1}|x_t, x_0), p_\theta(x_{t-1}|x_t)) \right] \end{aligned}$$

Zielfunktion

Die Kullback-Leibler Divergenz lässt sich analytisch lösen, da $q(x_{t-1}|x_t, x_0)$ und $p_\theta(x_{t-1}|x_t)$ normalverteilt sind:

- $q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I)$ mit $\tilde{\mu}(x_t, x_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t$ und $\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$
- $p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1} \mid \mu_\theta(x_t, t), \sigma_t^2 I)$ (siehe oben)

$$\Rightarrow D_{KL}(q(x_{t-1}|x_t, x_0), p_\theta(x_{t-1}|x_t)) = \frac{1}{2\sigma_t^2} ||\tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t)||^2 + \text{const}$$

Um nun den Erwartungswert $\mathbb{E}_{q(x_t|x_0)} \left[D_{KL}(q(x_{t-1}|x_t, x_0), p_\theta(x_{t-1}|x_t)) \right]$ zu samplen, können wir $x_t(x_0, \epsilon) = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ mit $\epsilon \approx \mathcal{N}(0, I)$ samplen. Durch Auflösen von x_t nach x_0 und Einsetzen dieses x_0 in $\tilde{\mu}_t(x_t, x_0)$ erhalten wir dann:

$$\tilde{\mu}_t(x_t, x_0) = \tilde{\mu}_t(x_t, \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t}\epsilon)) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon \right)$$

$$\Rightarrow B = \sum_{t=2}^{T} \mathbb{E}_{x_0, \epsilon} \left[\frac{1}{2\sigma_t^2} || \frac{1}{\sqrt{\alpha_t}} \left(x_t(x_0, \epsilon) - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon \right) - \mu_\theta(x_t(x_0, \epsilon), t) ||^2 \right]$$

Parametrisierung des Netzwerks

Anstatt $\mu_\theta(x_t, t)$ direkt den Mittelwert von $\tilde{\mu}_t(x_t, x_0)$ vorhersagen zu lassen, können bessere Ergebnisse erzielt werden, wenn das Netzwerk lernt, das hinzuaddierte Rauschen $\epsilon \sim \mathcal{N}(0, I)$ von $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$ zu entfernen.

$$\tilde{\mu}_t(x_t, x_0) = \tilde{\mu}_t(x_t, \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1 - \bar{\alpha}_t}\epsilon)) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}}\epsilon\right)$$

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right)$$

Das Netzwerk lernt also nicht mehr direkt $\mu_\theta(x_t, t)$ sondern $\epsilon_\theta(x_t, t)$.

Zielfunktion

Mit diesen Werten für $\tilde{\mu}_t$ und μ_θ vereinfacht sich B wie folgt:

$$B = \sum_{t=2}^{T} \mathbb{E}_{x_0,\epsilon} \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1-\bar{\alpha}_t)} \|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\|^2 \right]$$

Ho et al. (2020) haben empirisch festgestellt, dass das Training des Diffusionsmodells mit einem vereinfachten Loss, das den Gewichtungsterm ignoriert, besser funktioniert:

$$B_{\text{simple}} := \sum_{t=2}^{T} \mathbb{E}_{t,x_0,\epsilon} \left[\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\|^2 \right]$$

Training und Inferenz

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Figure: Trainings Algorithmus für Diffusion-Modelle [**ho2020denoising**]

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Figure: Inferenz Algorithmus für Diffusion-Modelle [**ho2020denoising**]

Architektur

Das Diffusion Modell hat keine außergewöhnlichen Anforderungen an die Architektur des Neuronalen Netzwerks. Das Neuronale Netzwerk muss lediglich von einem Bild auf ein Bild abbilden können $f_\theta : \mathbb{R}^{n \times n \times 3} \to \mathbb{R}^{n \times n \times 3}$. Eine mögliche Architektur für ein Diffusion-Modell ist ein U-Net.

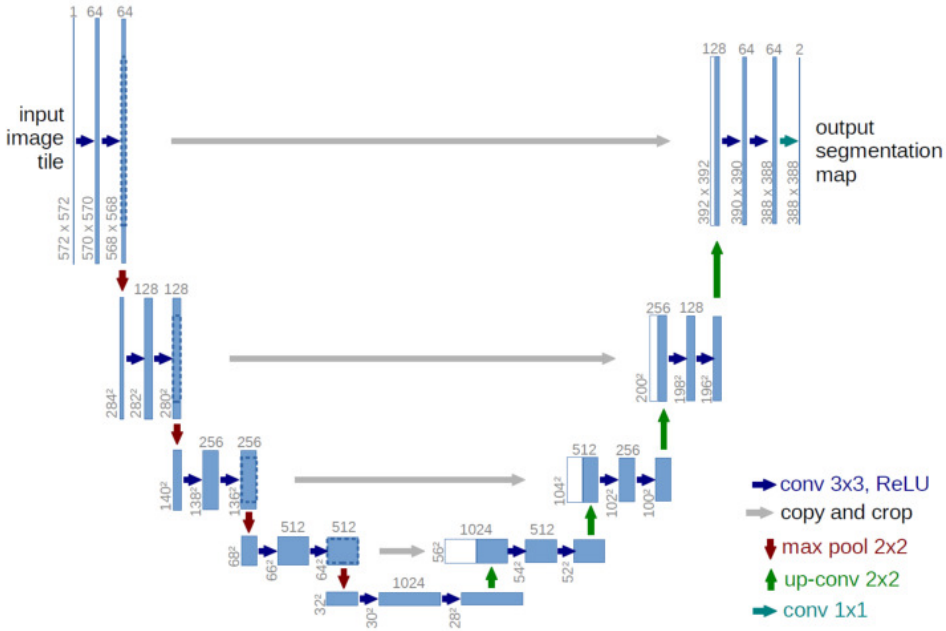


Figure: Architektur eines U-Net erweitert eine Konv. Autoencoder um Skip-Connetions auf unterschiedlichen Abstraktionsgraden.[Quelle]

Resultate

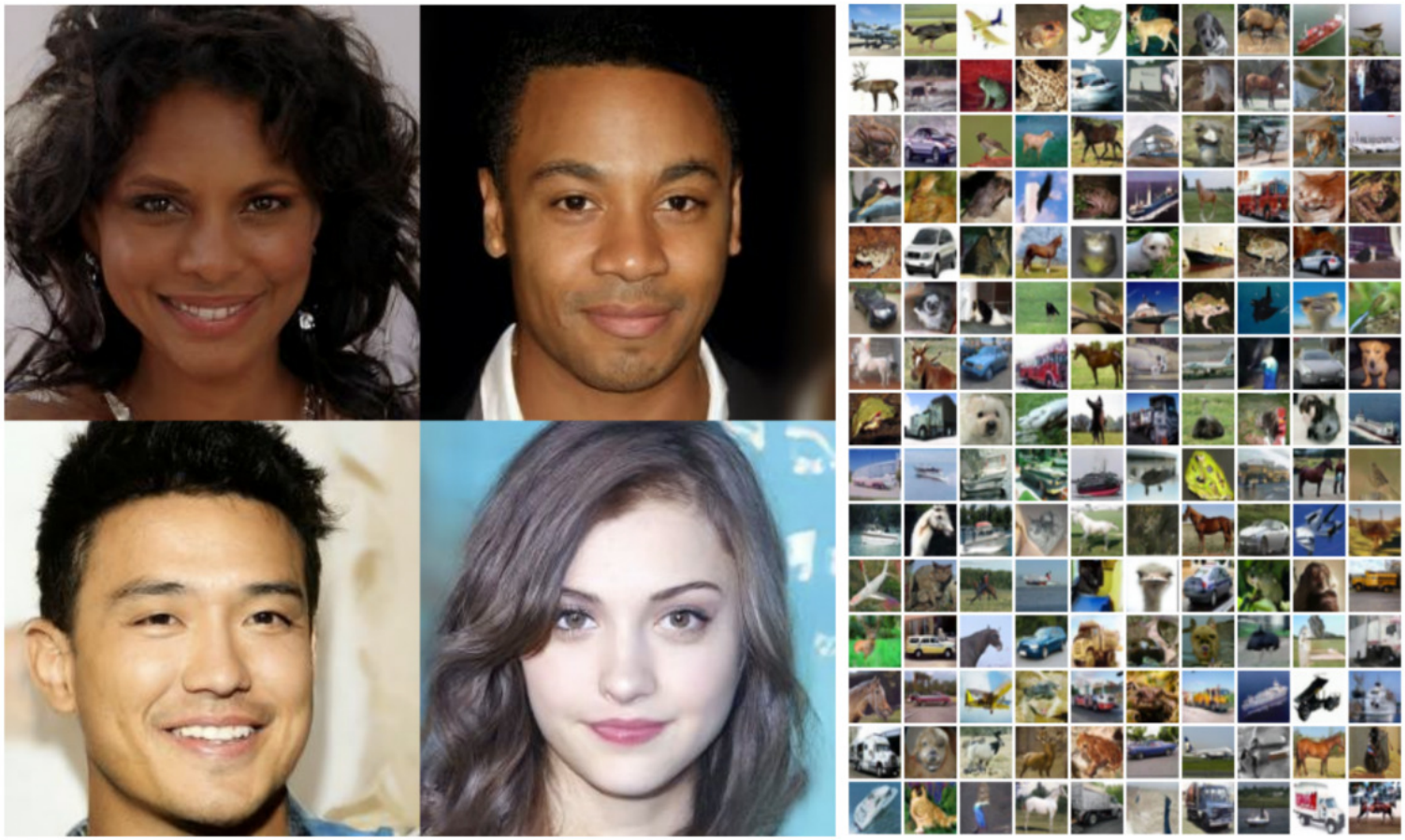


Figure 1: Generated samples on CelebA-HQ 256 × 256 (left) and unconditional CIFAR10 (right)

Figure: Resultate aus "Denoising Diffusion Probabilistic Models" (Ho et Al., 2020)

Vorteile und Nachteile [5]

Vorteile

- Aktuelle State-of-the-Art Methode zur Generation realistischer Bilder.
- Gute Repräsentation der Modi des Datensatzes (die generierten Bilder weisen eine hohe Vielfältigkeit auf).
- Keine besonderen Anforderungen an die Netzwerkarchitektur.

Nachteile

- Da der Diffusionsprozess iterativ ist, und für jeden Schritt das gesamte Modell ausgewertet werden muss, dauert das Training aber vor allem aber das Erzeugen neuer Bilder vergleichsweise lange.

... Diffusion-Modelle bilden eine wichtige Grundlage für generative Modelle zur Superreolution (z.B. SR3) oder zur Bildsynthese aus Texteingaben (z.B. Dall-E2).

[5] Diese Einordnung der Methode bezieht auf die grundlegende Methode und ist nur eine grobe Einordnung. Es wird an allen vorgestellten Modellen aktiv geforscht.