Deep Learning

Grundlagen Grundbegriffe Grundübungen

Alfatraining Kurs 2025-08 Dozent: Karsten Flügge

Themen des Tages

Tag 10

Methoden der kreativen Bilderzeugung
Generative Adversarial Networks (GAN)
Deepfakes
Diffusionsmodelle

Generative Adversarial Networks (GAN)

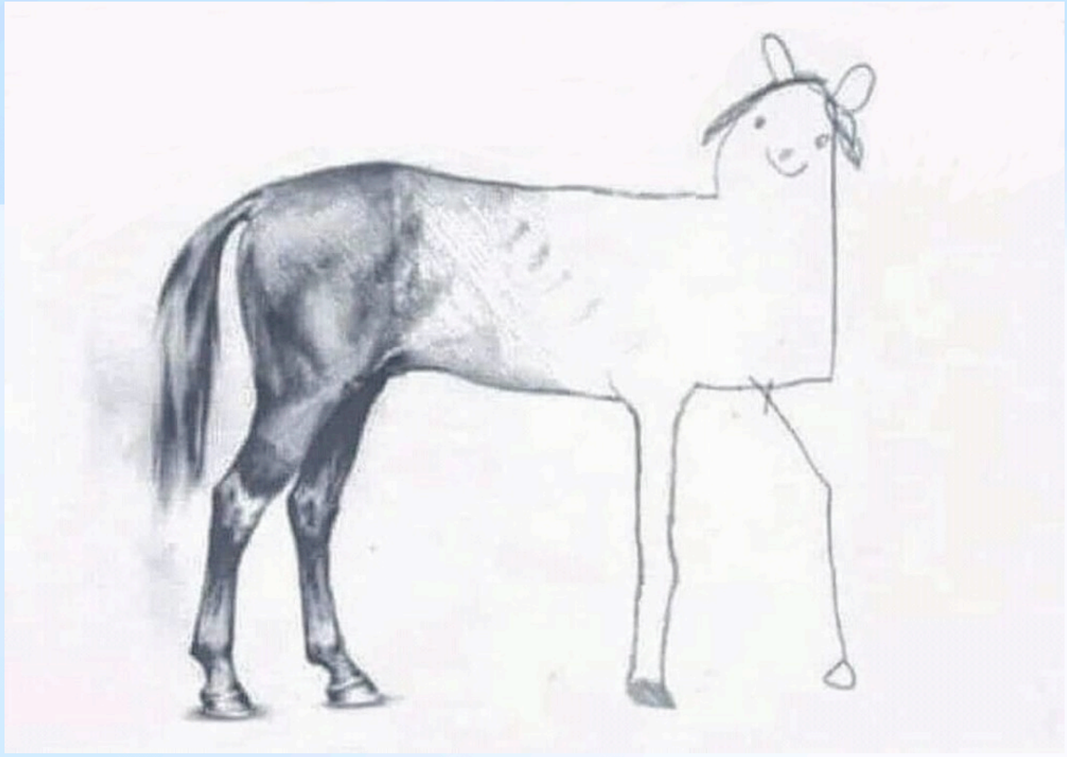
Zwei konkurrierenden Modelle:

- **Generator erzeugt Daten**
- **Diskriminator bewertet** die erzeugten Daten

GAN

Ian Goodfellow (2014)

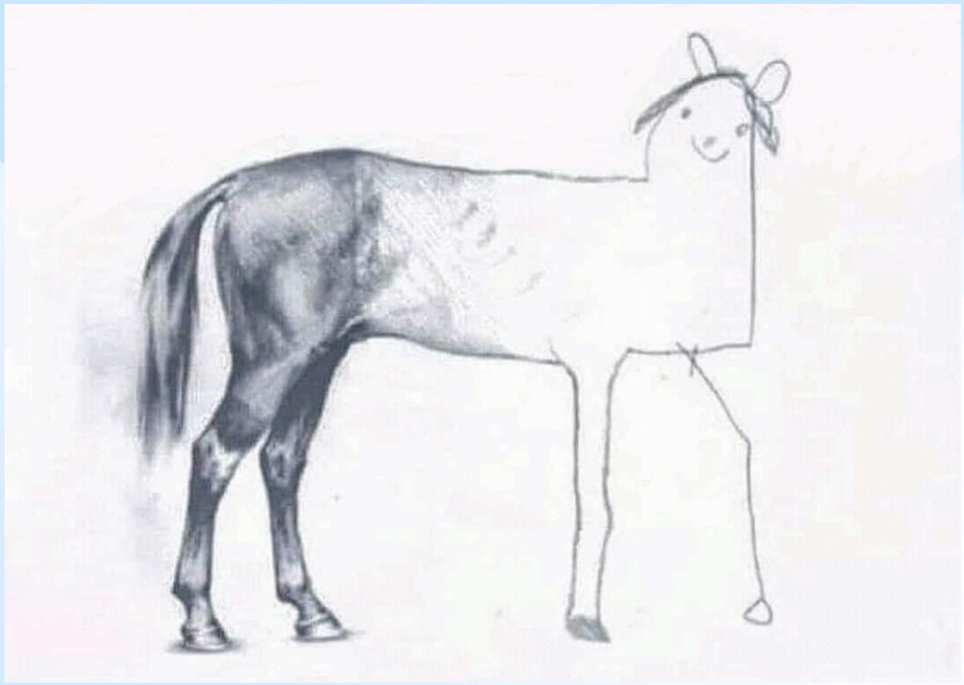
Wettbewerb: Beide Netzwerke werden gleichzeitig trainiert. **Der Generator versucht, den Diskriminator zu täuschen**, indem er **realistischere Daten** erzeugt, während der Diskriminator versucht, die echten Daten von den **gefälschten zu unterscheiden**. Dieses Spiel mit zwei Spielern führt dazu, dass der Generator immer besser wird und schließlich sehr realistische Daten erzeugen kann.



GAN

Szenario 1:
Generator: Dies ist ein Picasso!
Diskriminator: Dies ist kein Picasso!

Resultat: **Generator muss nachbessern**, Diskriminator wird belohnt.
Heisst: Anhand von Fehlersignalen werden wieder die Gewichte korrigiert

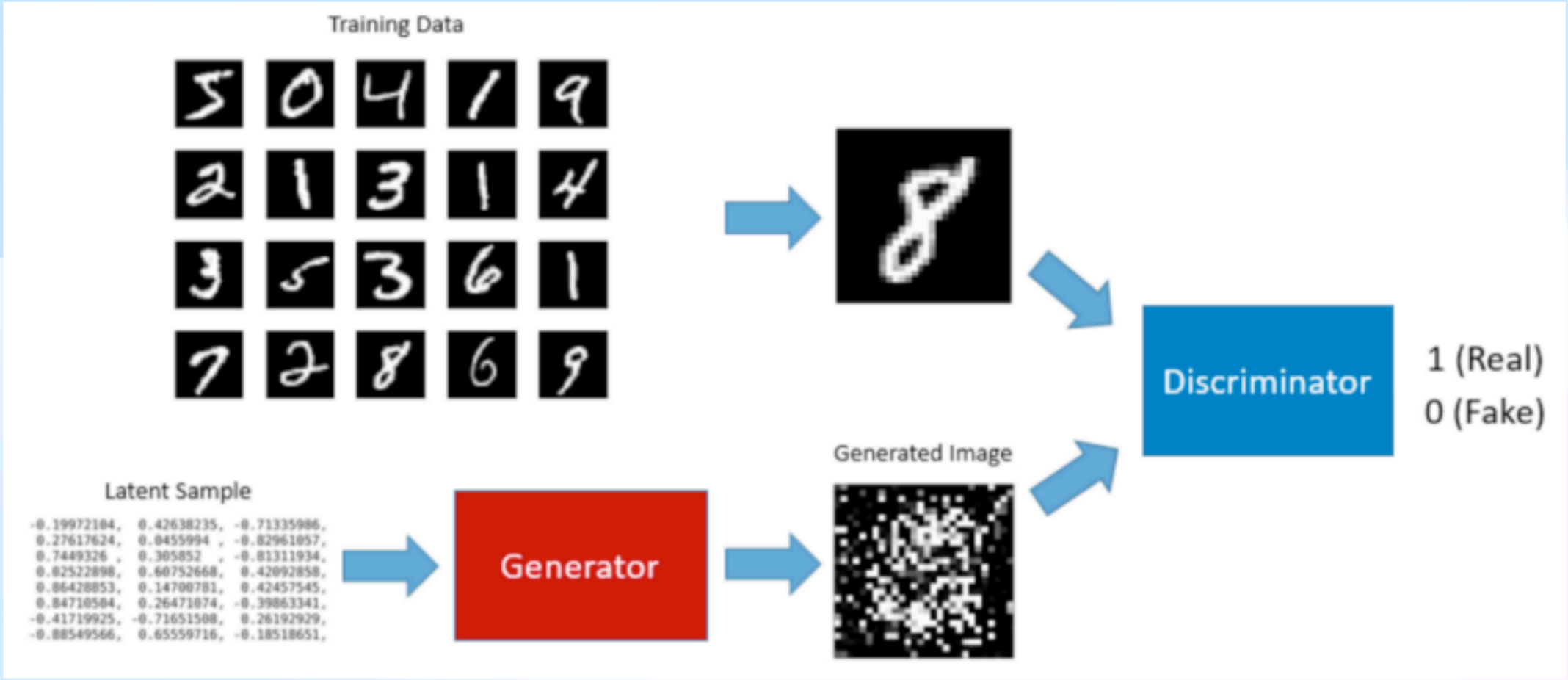


Beispiele und Anwendungen

- Bilderzeugung (z.B. DeepArt, Deepfake)
- Text Erzeugung
- **Style Transfer** (make it Picasso)
- Datenerweiterung (Data Augmentation : neu?)
- **Text-zu-Bild**-Generierung
- Spielentwicklung (Level-Design)
- neu: **Video** Erzeugung

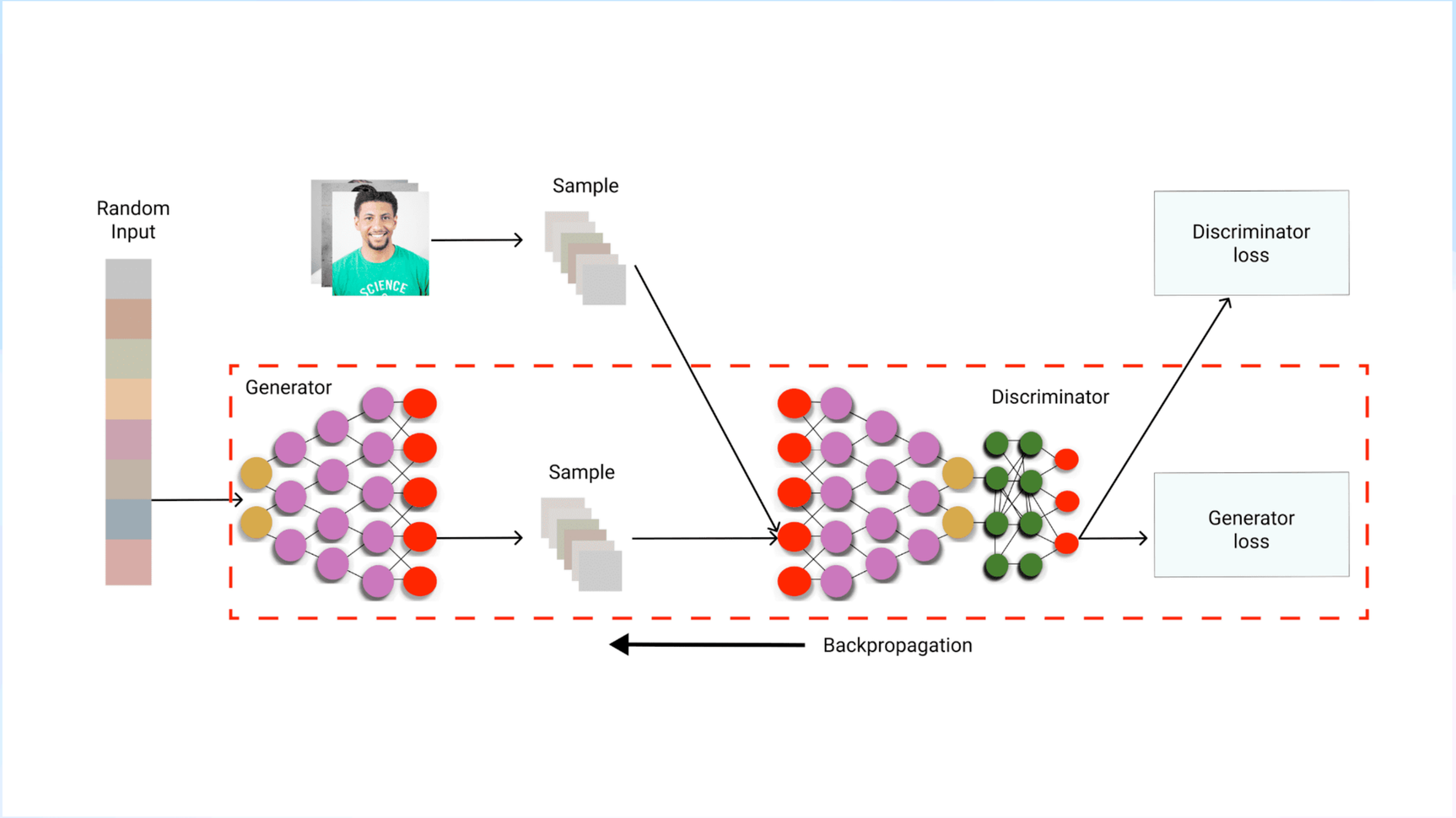
MNIST

Generator: Startet als Auto-encoder der Zahlen-Bilder
Diskriminator: Lernt zunächst normale Label



GAN

Generator: Startet als Decoder (wie beim Auto-encoder) **der Bilder erzeugt**
Diskriminator: Lernt zunächst normale Label



MNIST Pseudocode

Initialisierung:

Generator G neuronales Netz welches **Rauschen** im Latent Space Vektor / **Embedding** als Input bekommt und **Bild(größe) als output**
Discriminator D normales neuronales Netz mit **Bildgröße** als Input shape und ein Wert als output **(0,1 'fake'/'real')**

Training schleife:

Lade eine Batch **echte**r **Bilder** (real_images)
Berechne die **Ausgabe des Diskriminators für echte Bilder**
Berechne den **Verlust** für echte Bilder (D_loss_real) **also wie oft Vorhersage mit Wahrheit übereinstimmte**

Erzeuge **gefälschte Bilder** mit dem Generator

Berechne die **Ausgabe des Diskriminators** für gefälschte Bilder (output_fake)
Berechne den **Verlust für gefälschte Bilder** (D_loss_fake) **wie oft wurden gefälschte Bilder nicht enttarnt**

Rückwärtspropagation und Update der Gewichte des Diskriminators
Berechne den **gesamten Verlust** des Diskriminators (D_loss = D_loss_real + D_loss_fake)
Berechne **Gradienten** von D_loss
Aktualisiere die Gewichte des Diskriminators

 # Training des Generators
Berechne den **Verlust des Generators** (G_loss) :
wie oft die Fake-Bilder als 'Real' klassifiziert

Rückwärtspropagation und Update der Gewichte des Generators
Berechne **Gradienten** von G_loss
Aktualisiere die Gewichte des Generators

MNIST code

```
for i, (images, _) in enumerate(data_loader):
    real_images = Variable(images.view(-1, image_size))

    #  echte Bilder '1' und gefälschte Bilder '0'
    real_labels = Variable(torch.ones(batch_size, 1)) # info für Diskriminator: alle MNIST Bilder sind echt
    fake_labels = Variable(torch.zeros(batch_size, 1)) # info für Diskriminator: alle generierten Bilder sind fake

    outputs = D(real_images) # sage Label für echte Bilder voraus
    # Überprüfe wie oft die Bilder richtig klassifiziert wurden (alle sollten 1 also 'Real' sein)
    D_loss_real = criterion(outputs, real_labels)

    # Generiere zufälliges Rauschen um Fake-Bilder zu generieren
    z = Variable(torch.randn(batch_size, latent_size))
    fake_images = G(z) # generiere Fake-Bilder
    outputs = D(fake_images) # sage Label für Fake-Bilder voraus
    # Überprüfe wie oft die Bilder richtig klassifiziert wurden (alle sollten 0 also 'Fake' sein)
    D_loss_fake = criterion(outputs, fake_labels)

    D_loss = D_loss_real + D_loss_fake

    # Training Generator
    # Überprüfe wie oft die Fake-Bilder als 'Real' klassifiziert wurden
    G_loss = criterion(outputs, real_labels)
```

GANs Metakriterien

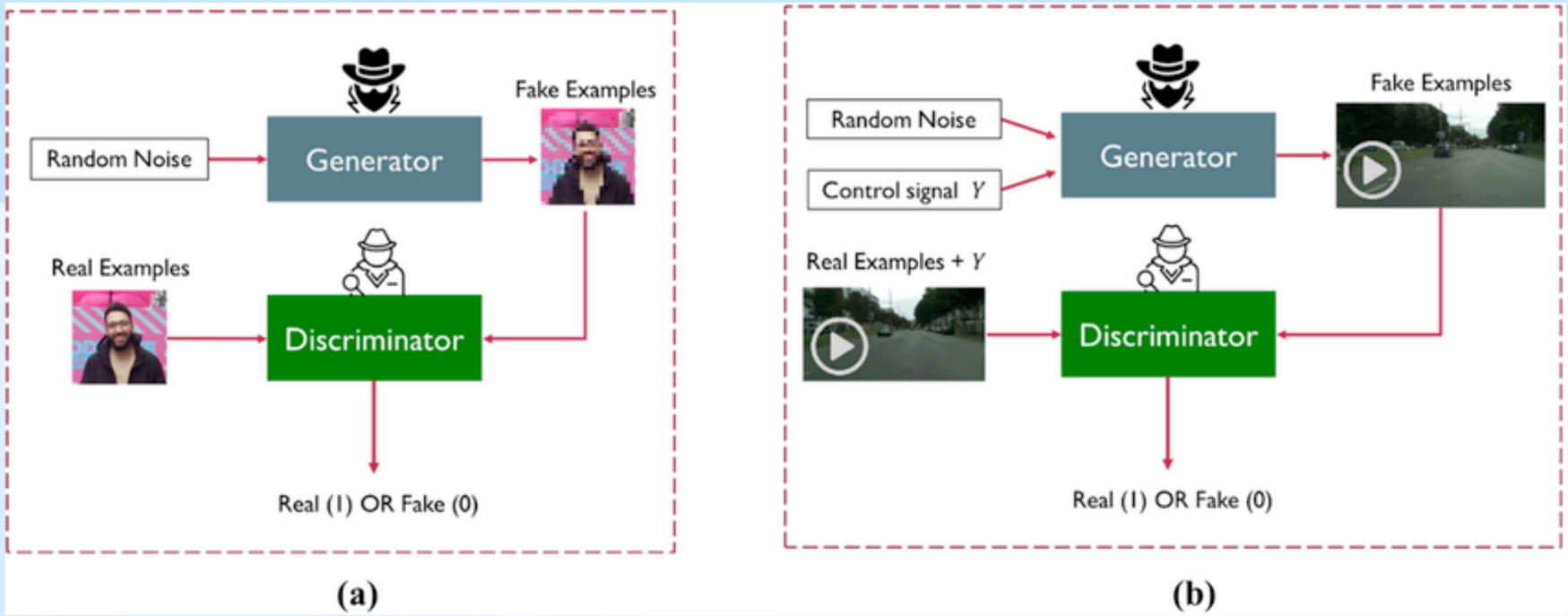
- **GANs** sind etwas **hard zu trainieren**:
- Diskriminator und Generator müssen '**austariert**' sein

- idealer **Latent space** ist **steuerbar**, dass heisst, sagt man 'erzeuge 9' dann soll er eine 9 malen
- oder 'Features' erzeuge **dicke** 9 …

Conditional GANs

- Bisher erzeugt unser Generator nur zufällige Zahlen es wäre natürlich schön wenn wir **gezielt Beispielbilder** erzeugen können.
- Dazu geben wir unseren Generator noch das **Label** mit für welches ein Bild erzeugt werden soll

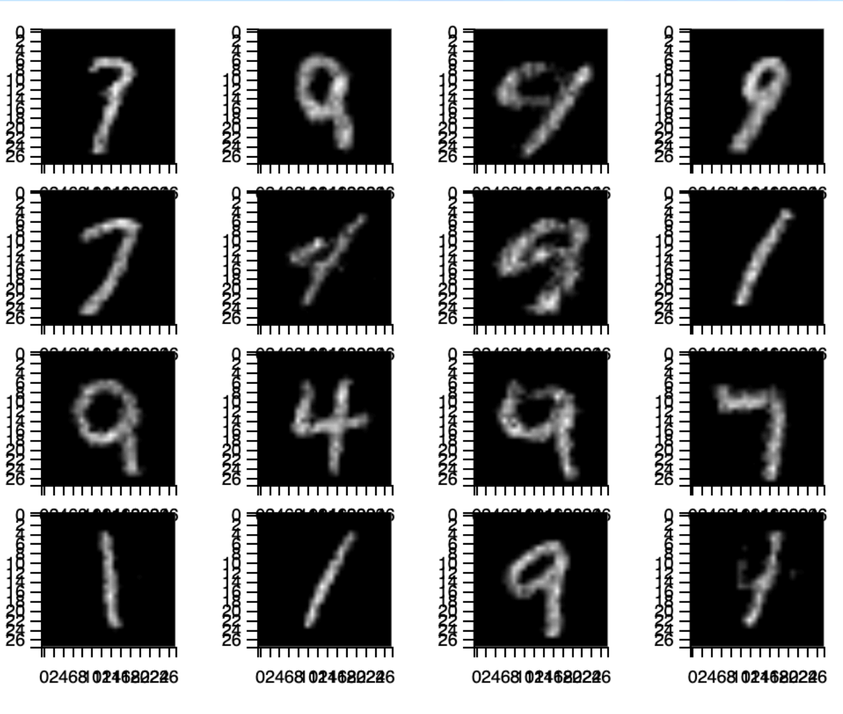
Diese Zusatzinformationen nennt sich **Condition**, daher Conditional GANs



Statt das Label explizit mitzugeben kann man es aber auch über das 'Rauschen' steuern:

Aufgaben

- Lasst den Beispiel GAN min durchlaufen:
- Optimiert mnist_gan.py zu mnist_gan_**min**.py
  so, dass es bei Euch schneller läuft (also hidden space, latent space kleiner)



- während GAN im Hintergrund trainiert:

Autoencoder sind wichtig (und einfach!)
trivial tag8/mnist_autoencoder.py selbst implementieren
Ausnahmsweise ohne GPT
erst Lösung anschauen und verstehen, dann aus Gedächtnis selbst…
Ggf in Lösung spicken, aber schon versuchen von Grund auf selbst zu machen

Zusatzaufgabe: interpoliere zwischen 1 und 7

Latent Space

- Bisher haben wir dem Generator ein bisschen rausstellen mitgegeben damit er eine Grundlage hat um Bilder zu erstellen. Die Menge dieser Rausch Vektoren nennt man auch **Latent Space** (≈Nebenraum).

- Den **Latent Space** kann man sich zu Nutze machen um **gezielter Daten zu erzeugen**:
  - Ähnliche Eingaben führen zu **ähnlichen Ausgaben** (automatische Clusterung)
  - Man kann zu einer gegebenen Ausgabe eine passende Eingabe **rekonstruieren**
  - Man kann zwischen verschiedenen Ausgaben **interpolieren !**

- Man kann den Latent Space mit einem anderen Netz **co-trainieren** (z.B. **Text-to-Image**)

Latent Space co-training

- Den **Latent Space** kann man sich zu Nutze machen um **gezielter Daten zu erzeugen**:
- wähle **output embedding** von einem Netz als **input embedding** für den Generator

Text Embedding Model: Erzeugt für beliebigen Text einen **Vektor**.
"**zahl 9**" **-> Vektor** -> MNIST Generator mit Zusatzinformation: Klasse 9 -> male 9 via GAN

Latent Space

- Den **Latent Space** kann man **visualisieren,** indem man
- **clustered:** (extra code t-SNE.py an Tag 8, ausserhalb von Deep Learning)
- **sampled**: zufällig einen Vektor z auswählen und dann das generierte Bild G(z) anzeigen
- **interpoliert**: zwischen Zwei Bildern und deren zugrunde liegendem Embedding z1,z2 :
- z = α*z1 + (1-α)*z2
- G(z) kann dann je nach α so aussehen wie
- das erste Bild (α=1), das zweite (α=0) oder alles dazwischen (α=1/2 …)

- **backtracking**: finde besten **Latent Vektor** (*Embedding)* für gegebenes output **Bild**:

Fange mit zufälligem Latent Vektor **z** an und optimiere dann so, dass
die Differenz² vom generierten Bild **G(z)** zum gegebenen Bild (target) minimiert wird (wie beim Autoencoder) :

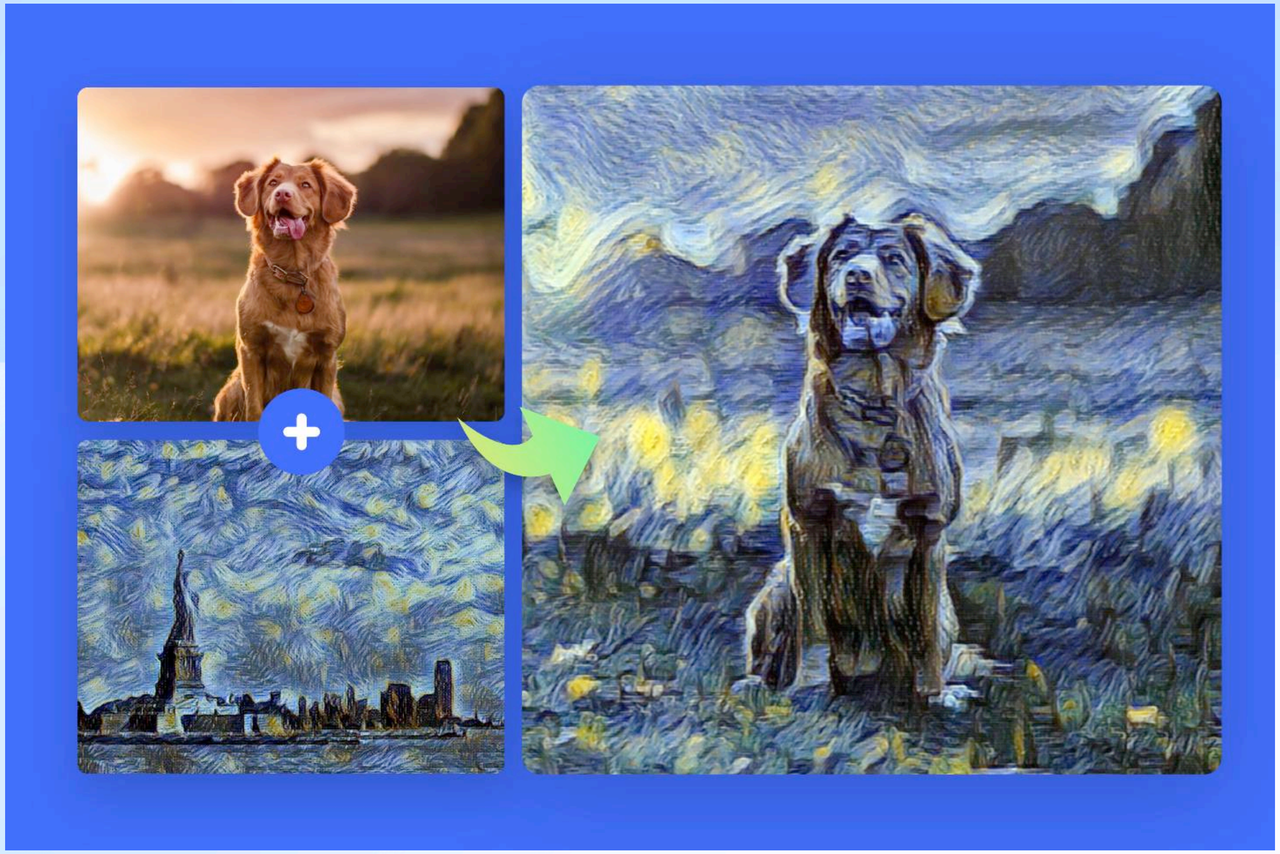
$$\text{Loss}(\mathbf{z}) = \|G(\mathbf{z}) - \text{target}\|^2$$

dann kann man zum Beispiel **in der Nähe** vom gefundenen z **ähnliche Bilder** erzeugen

"closest representative embedding" siehe **find_latent_vector.py**

Style transfer

Vorausschau: Modifikation des latenten Raumes,
um **Stil Aspekte** von einem Bild zum anderen zu **übertragen**
(primitiver Ansatz: addiere Embeddings, heute oft realisiert mit **de-convolution** Schichten)



Moderne GANs

Allgemeine Techniken (CNN, Attention) finden natürlich auch bei GANs Verwendung:

de-Convolution: gehe Convolution Schritt 'rückwärts': erzeuge aus feature maps Bild Segmente

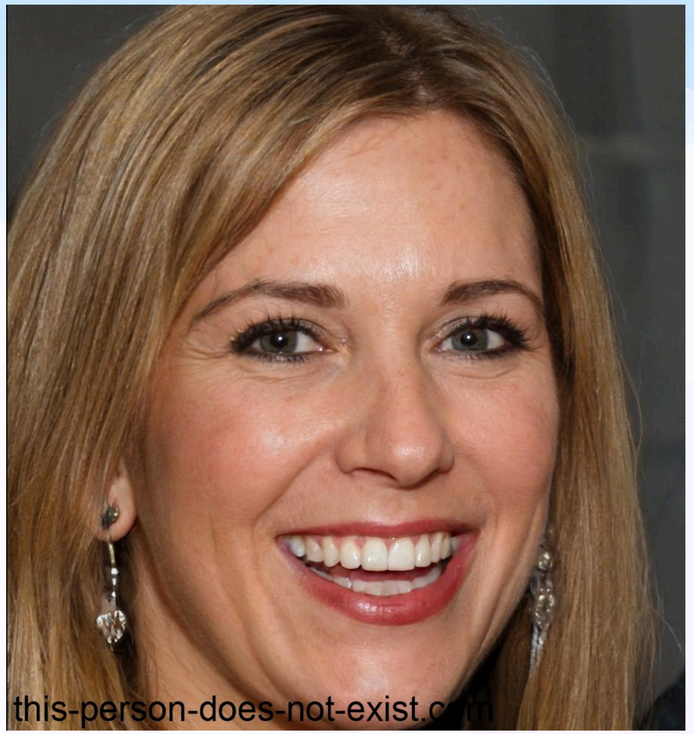
1. **CycleGAN**:
   - Ermöglicht das Training ohne gepaarte Daten, indem es zwei Generatoren und zwei Discriminatoren verwendet, um Bild-zu-Bild-Übersetzungen durchzuführen. Ein Beispiel ist die Umwandlung von Sommer- in Winterlandschaften.
2. **GANs with Conditional Inputs (cGANs)**:
   - Erweiterung von GANs, die Konditionen wie **Labels** oder Text verwenden, um die Kontrolle über den generierten Output zu erhöhen. Anwendungen umfassen die Erzeugung spezifischer Bildkategorien oder Merkmale.
3. **DCGAN (Deep Convolutional GAN)**:
   - Einführung von Convolutional Layers in GAN-Architekturen, was zu besseren Ergebnissen bei der Generierung hochauflösender Bilder führt. Dies war ein Schlüsselschritt für die Nutzung von GANs in der Bildgenerierung.
4. **Self-Attention GAN (SAGAN)**:
   - Integriert Self-Attention Mechanismen, um lange Reichweitenabhängigkeiten in Bildern zu modellieren. Dies führt zu einer Verbesserung der Kohärenz und Konsistenz in großen Bildern.
5. **Mixture of Experts GAN (MoE-GAN)**:
   - Verwendet mehrere Generatoren, die als Experten für verschiedene Bildbereiche fungieren. Diese Kombination verbessert die generierte Bildqualität, insbesondere bei komplexen Datensätzen.
6. **Pix2Pix**:
   - Eine Architektur, die auf gepaarten Bilddaten trainiert wird, um präzise Bildübersetzungen zu ermöglichen, z.B. Skizzen in Fotos umzuwandeln. Sie eignet sich für Anwendungen in der Bildbearbeitung und -umwandlung.
7. **StarGAN**:
   - Ermöglicht das Training eines einzigen Modells für die bedingte Bilderzeugung über mehrere Domänen hinweg. Es ist besonders nützlich für Multi-Domain-Bildmanipulationen, wie z.B. das Ändern von Gesichtsattributen.

Deep Fakes

- **Sampling**
- Wenn man aus dem **Latent Space** einen Vektor auswählt welcher nicht beim Trainieren verwendet wurde, ist das erzeugte Bild tatsächlich von neuartiger Natur.
- auf diese Weise kann man zum Beispiel Gesichter erzeugen welche **vorher nicht existiert** haben
- Gerade durch Steuerung mit **Text-to-Image, Text-to-Audio** oder gar **Text-to-Video**
- lassen sich so **gefälschte Medien Inhalte** erstellen

https://this-person-does-not-exist.com/

this … does not exist



Deep Fakes

Vorteile:

- Kreative Anwendungen (z.B. in der Medienindustrie)
- Bildungs- und Forschungstools.

Nachteile:

- Potenzial für Missbrauch und Verbreitung von Falschinformationen.
- Bedrohung der Privatsphäre und Sicherheit. (Impersonation!)

Beispiele und Anwendungen:

- Konservierung der eigenen Stimme (Nach Operation / Tod)
- Ausdruck eigener Ideen
- Fälschung von Politikerreden.
- Austausch von Schauspielergesichtern in Filmen.
- Erstellung von realistischen Stimmen für Sprachassistenten.
- … !

Weitere Aspekte

- **Schwieriges Training** (Instabilität, Mode Collapse)
- Hoher Rechenaufwand
- Erweiterungen und Varianten (z.B. Conditional GANs, CycleGANs)
- Ethische Überlegungen (Missbrauchsmöglichkeiten, Fake-News)
- Aktuelle Forschungsfragen (Stabilisierung des Trainings, Verbesserung der Qualität)

Weitere Aspekte

Mode Collapse

Generator erzeugt nur eine Klasse von Bildern!

Mini-Batch Discrimination: Sie misst, wie ähnlich sich die Bilder eines Batchs sind, und stellt diese Statistik dem Diskriminator zur Verfügung, sodass er problemlos einen gan- zen Batch mit künstlichen Bildern verwerfen kann, wenn diese nicht abwechslungsreich genug sind.

Andere Gegenmittel: aktive Forschung &

Diffusionsmodelle

lassen sich nicht nur viel einfacher trainieren als GANs, die generierten Bilder sind auch viel diverser und von noch höherer Qualität, allerdings mit höherem Rechenaufwand.

rianzm

Pause

prompt = "scrooge mcduck as pirate with eyepatch riding a green camel"

Diffusionsmodelle

Man beginnt mit Trainings **Bilder**n und fügt diesen **schrittweise Rauschen** hinzu.

Wir trainieren unser **Diffusionmodell** so, dass es den **Rückwärts Schritt lernt** also aus einem Bild mit viel Rauschen ein Bild mit weniger Rauschen zu generieren. Daher bezeichnet man sie auch als

(Stacked) Denoising Auto Encoder (anti image augmentation;)

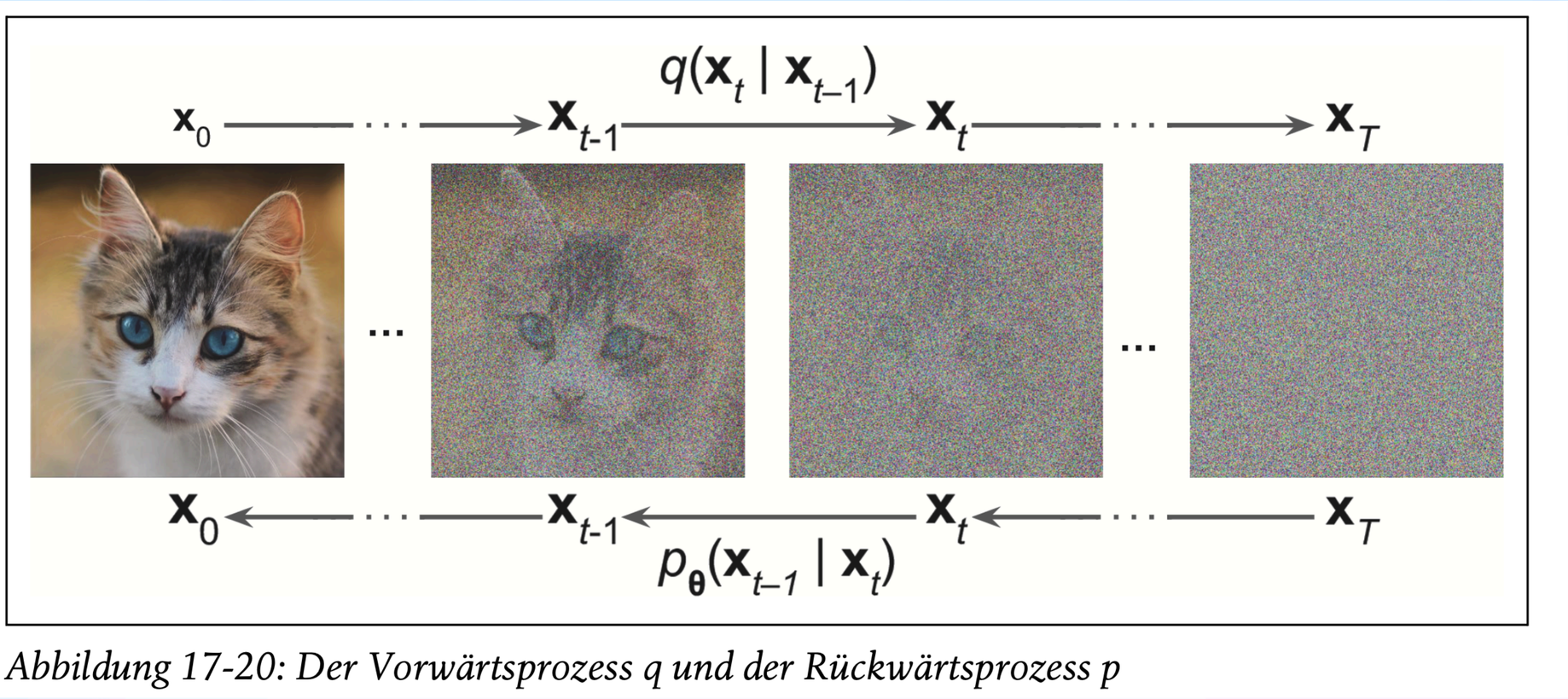
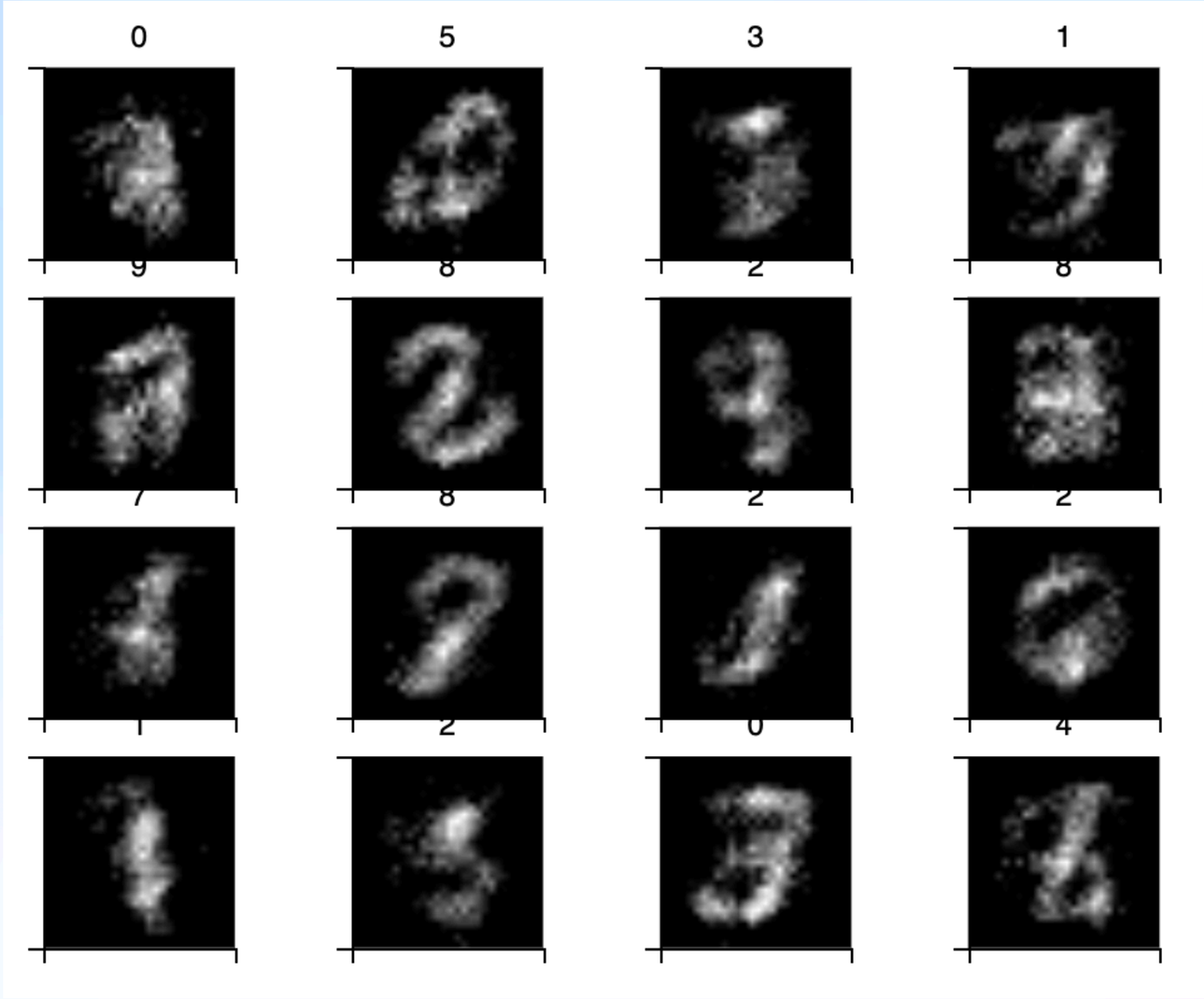


Abbildung 17-20: Der Vorwärtsprozess q und der Rückwärtsprozess p

Diffusionsmodelle



Diffusionsmodelle

Es gibt viele Arten von Rauschen und viele Arten wie die Verteilung der Bildpunkte mit dem Rauschen Inter agieren deswegen ist die Implementierung dieser **im Grunde einfachen Idee** in der Praxis etwas komplizierter:

Man versucht nach bestimmten recht komplizierten(!) Formeln die Varianz in bestimmten Bereichen zu lenken.

Alternativer **Fortschritt**: Diffusionsprozess **im latenten Raum** statt im Pixelraum (mit leistungsfähigem Autoencoder)

Dies beschleunigt die Bilderzeugung erheblich und reduziert Zeit und Kosten für das Training drastisch.

Darüber hinaus ist die Qualität der erzeugten Bilder beeindruckend.

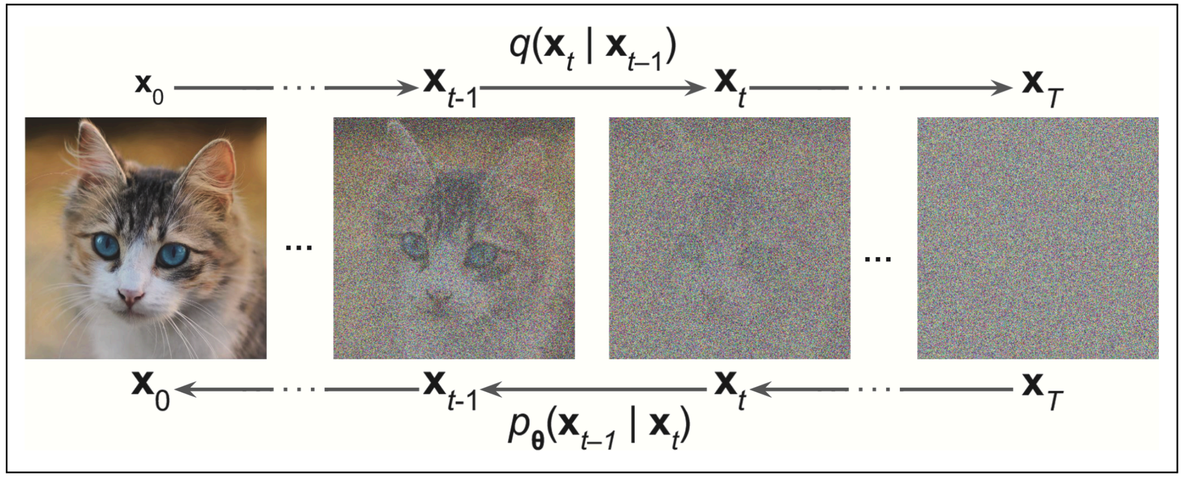


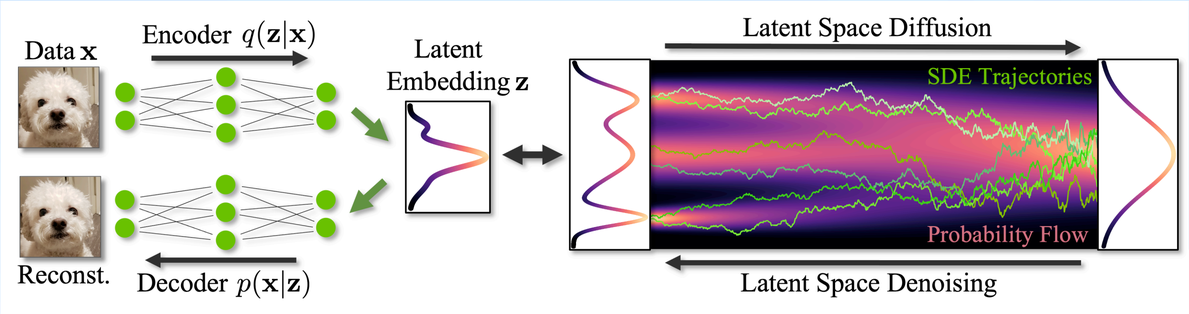
Abbildung 17-20: Der Vorwärtsprozess q und der Rückwärtsprozess p

Latent Diffusion

Latent Diffusion: addiere Rauschen nicht zum Bild, sondern zur Mittelschicht ("Embedding, Latent space") von einem **Denoising** Auto-Encoder

Encoder Komprimiert Bild in **Embedding** Bottleneck

Decoder Dekomprimiert Embedding zu Bild



<u>Stacked</u> Denoising Auto-Encoder

- Statt das Netzwerk versuchen zu lassen,
- das gesamte Rauschen in einem Schritt zu entfernen,
- ist es einfacher (fürs Netz), sukzessiv das Rauschen in
- mehreren Schritten zu entfernen.
- Analog beim Trainieren nur von einer Rausch-Stufe
- zur nächsten trainieren

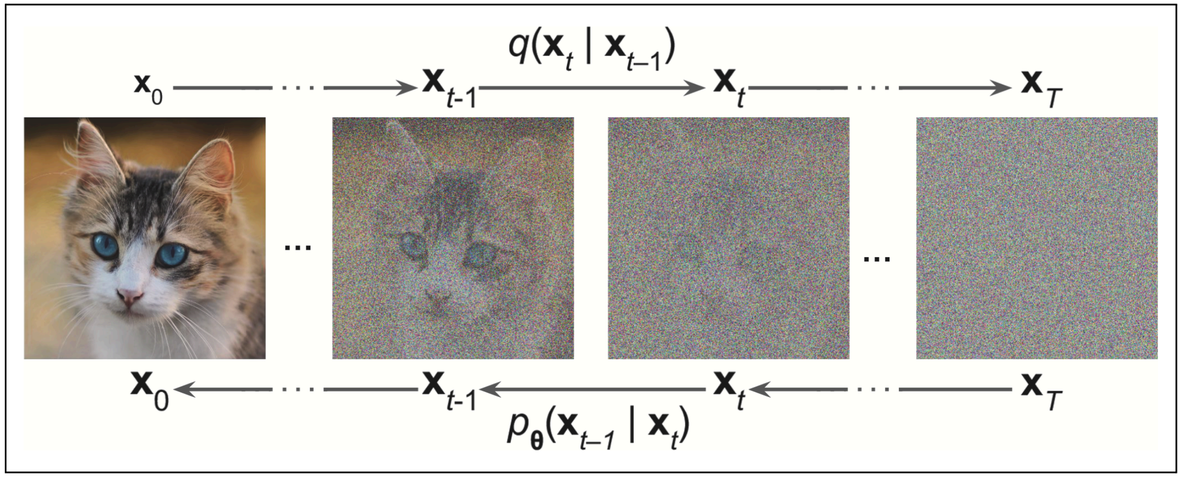


Abbildung 17-20: Der Vorwärtsprozess q und der Rückwärtsprozess p