Deep Learning - Übungsblatt 5

Generative Modelle und Fortgeschrittenes Deep Learning

Fachhochschule Südwestfalen

23. Oktober 2025

Voraussetzungen

- Alle vorherigen Übungsblätter sollten erfolgreich bearbeitet worden sein
- Solides Verständnis von CNNs, RNNs und Optimierungsverfahren
- Grundkenntnisse in Wahrscheinlichkeitstheorie und Informationstheorie
- Vertrautheit mit komplexeren Deep Learning Konzepten

Lernziele

Nach erfolgreicher Bearbeitung dieser Übung können Sie:

- Autoencoder-Architekturen verstehen und implementieren
- Variational Autoencoders (VAEs) mathematisch herleiten
- Generative Adversarial Networks (GANs) konzipieren und trainieren
- Diffusion Models grundlegend verstehen
- Transfer Learning und Fine-Tuning praktisch anwenden
- Moderne Optimierungsverfahren implementieren

1 Autoencoders

1.1 Grundlegende Autoencoder-Mathematik

Aufgabe 1.1: Gegeben sei ein einfacher Autoencoder für MNIST-Daten:
Architektur:

$$\text{Encoder: } \mathbf{z} = f_{\text{enc}}(\mathbf{x}) = \sigma(\mathbf{W}_e\mathbf{x} + \mathbf{b}_e) \tag{1}$$

$$\text{Decoder: } \hat{\mathbf{x}} = f_{\text{dec}}(\mathbf{z}) = \sigma(\mathbf{W}_d\mathbf{z} + \mathbf{b}_d) \tag{2}$$

$$\text{Loss: } L(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2 \tag{3}$$

Dimensionen: - Input: $\mathbf{x} \in \mathbb{R}^{784}$ (28×28 Pixel, flattened) - Latent: $\mathbf{z} \in \mathbb{R}^{32}$ (Bottleneck) - Output: $\hat{\mathbf{x}} \in \mathbb{R}^{784}$ (Rekonstruktion)

(a) Bestimmen Sie die Dimensionen aller Gewichtsmatrizen und Bias-Vektoren

(b) Berechnen Sie die Gesamtanzahl der Parameter

(c) Gegeben sei $\mathbf{x} = (1, 0, 1, 0, \ldots)^T$ (vereinfacht auf 4D) und:

$$\mathbf{W}_e = \begin{pmatrix} 0.5 & 0.2 & -0.1 & 0.3 \\ 0.1 & -0.4 & 0.6 & 0.2 \end{pmatrix}, \quad \mathbf{b}_e = \begin{pmatrix} 0.1 \\ -0.2 \end{pmatrix} \tag{4}$$

$$\mathbf{W}_d = \begin{pmatrix} 0.4 & 0.1 \\ -0.2 & 0.5 \\ 0.3 & -0.1 \\ 0.2 & 0.4 \end{pmatrix}, \quad \mathbf{b}_d = \begin{pmatrix} 0.05 \\ -0.1 \\ 0.15 \\ 0.0 \end{pmatrix} \tag{5}$$

(d) Berechnen Sie $\mathbf{z}$ und $\hat{\mathbf{x}}$ (verwenden Sie $\sigma(x) = \frac{1}{1+e^{-x}}$)

(e) Berechnen Sie den Rekonstruktionsfehler $L(\mathbf{x}, \hat{\mathbf{x}})$

(f) Interpretieren Sie: Was repräsentiert der latente Raum $\mathbf{z}$?

1.2 Denoising Autoencoders

Aufgabe 1.2: Noise Robustness

(a) Erklären Sie das Konzept von Denoising Autoencoders

(b) Gegeben sei ein Input $\mathbf{x}$ und Noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2\mathbf{I})$

(c) Die noisy Eingabe ist $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}$

(d) Loss-Funktion: $L = \|\mathbf{x} - f(\tilde{\mathbf{x}})\|^2$

(e) Diskutieren Sie: Wie lernt der Autoencoder robuste Repräsentationen?

(f) Implementieren Sie verschiedene Noise-Typen (Gaussian, Salt-and-Pepper, Dropout)

2 Variational Autoencoders (VAEs)

2.1 VAE-Mathematik

Aufgabe 2.1: Variational Inference

VAE-Gleichungen:

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_\phi(\mathbf{x}), \boldsymbol{\sigma}^2_\phi(\mathbf{x})) \quad \text{(Encoder)} \tag{6}$$

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}_\theta(\mathbf{z}), \mathbf{I}) \quad \text{(Decoder)} \tag{7}$$

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad \text{(Prior)} \tag{8}$$

ELBO (Evidence Lower Bound):

$$\mathcal{L} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) \tag{9}$$

(a) Leiten Sie die KL-Divergenz zwischen zwei Gaussschen Verteilungen her:

$$\mathrm{KL}(\mathcal{N}(\mu_1, \sigma_1^2) \| \mathcal{N}(\mu_2, \sigma_2^2))$$

(b) Für den VAE-Fall mit $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, zeigen Sie:

$$\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})) = \frac{1}{2}\sum_{i=1}^{d}(1 + \log\sigma_i^2 - \mu_i^2 - \sigma_i^2)$$

(c) Erklären Sie den **Reparameterization Trick**: $\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$ mit $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

(d) Diskutieren Sie die zwei Terme der ELBO:

- Reconstruction Term: Was wird optimiert?
- Regularization Term: Welche Rolle spielt dieser?

2.2 β-VAE

Aufgabe 2.2: Disentangled Representations

(a) Die β-VAE Loss-Funktion ist:

$$\mathcal{L}_\beta = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta \cdot \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$$

(b) Analysieren Sie den Einfluss von β:

- $\beta = 1$: Standard VAE
- $\beta > 1$: Stärkere Regularisierung
- $\beta < 1$: Schwächere Regularisierung

(c) Erklären Sie das Konzept von **Disentangled Representations**

(d) Diskutieren Sie das Trade-off zwischen Reconstruction Quality und Disentanglement

3 Generative Adversarial Networks (GANs)

3.1 GAN-Grundlagen

Aufgabe 3.1: Minimax-Spiel
GAN-Zielfunktion:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z})))] \tag{10}$$

(a) Erklären Sie die Rollen von Generator G und Discriminator D

(b) Zeigen Sie, dass der optimale Discriminator für einen festen Generator G ist:

$$D^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}$$

(c) Beweisen Sie, dass für den optimalen Discriminator D^*:

$$V(D^*, G) = -\log(4) + 2 \cdot \mathrm{JS}(p_{\text{data}}\|p_g)$$

wobei JS die Jensen-Shannon Divergenz ist

(d) Interpretieren Sie: Warum minimiert der Generator die JS-Divergenz?

(e) Diskutieren Sie das **Mode Collapse** Problem

3.2 Wasserstein GANs (WGANs)

Aufgabe 3.2: Wasserstein Distance

(a) Die Wasserstein-1 Distance (Earth-Mover Distance) ist definiert als:

$$W(p, q) = \inf_{\gamma \in \Pi(p,q)} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\gamma}[\|\mathbf{x} - \mathbf{y}\|]$$

(b) Unter der Kantorovich-Rubinstein Dualität:

$$W(p, q) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\mathbf{x}\sim p}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{y}\sim q}[f(\mathbf{y})]$$

(c) WGAN-Zielfunktion:

$$\min_G \max_{D\in\mathcal{D}} \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_z}[D(G(\mathbf{z}))]$$

(d) Erklären Sie die **Lipschitz-Constraint** auf den Discriminator

(e) Diskutieren Sie: Warum sind WGANs stabiler als Standard GANs?

(f) Beschreiben Sie den **Gradient Penalty** Ansatz (WGAN-GP)

4 Diffusion Models

4.1 Denoising Diffusion Probabilistic Models (DDPMs)

Aufgabe 4.1: Forward und Reverse Process

Forward Process (Noise-Addition):

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{11}$$

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{12}$$

Reverse Process (Denoising):

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \tag{13}$$

(a) Zeigen Sie, dass der Forward Process direkt von $\mathbf{x}_0$ zu $\mathbf{x}_t$ führt:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

mit $\alpha_t = 1 - \beta_t$ und $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$

(b) Erklären Sie das **Training-Ziel** für DDPMs:

$$L = \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon},t}[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2]$$

(c) Beschreiben Sie den **Sampling-Prozess** zur Generierung

(d) Diskutieren Sie: Wie unterscheiden sich Diffusion Models von VAEs und GANs?

5 Programmieraufgaben

5.1 Autoencoder Implementation

Aufgabe 5.1: MNIST Autoencoder

Listing 1: Autoencoder für MNIST

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt

class Autoencoder(keras.Model):
    def __init__(self, encoding_dim):
        super(Autoencoder, self).__init__()
        self.encoding_dim = encoding_dim

        # Encoder
        self.encoder = keras.Sequential([
            # TODO: Implementieren Sie den Encoder
            # - Flatten für MNIST (28, 28) -> (784,)
            # - Dense Layer(s) mit Aktivierung
            # - Bottleneck mit encoding_dim
        ])

        # Decoder
        self.decoder = keras.Sequential([
            # TODO: Implementieren Sie den Decoder
            # - Dense Layer(s) von encoding_dim zurück zu 784
            # - Reshape zurück zu (28, 28)
            # - Sigmoid-Aktivierung für Pixel-Werte
        ])

    def call(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded

def plot_reconstructions(model, test_data, n=10):
    """Visualisiert Original vs. Rekonstruktion"""
    predictions = model.predict(test_data[:n])

    plt.figure(figsize=(20, 4))
    for i in range(n):
        # Original
        ax = plt.subplot(2, n, i + 1)
        plt.imshow(test_data[i].reshape(28, 28), cmap='gray')
        plt.title("Original")
        plt.axis('off')

        # Rekonstruktion
```

```
        ax = plt.subplot(2, n, i + 1 + n)
        plt.imshow(predictions[i].reshape(28, 28), cmap='gray')
        plt.title("Rekonstruiert")
        plt.axis('off')
    plt.show()

def main():
    # Daten laden
    (x_train, _), (x_test, _) = keras.datasets.mnist.load_data()

    # TODO: Datenvorverarbeitung
    # - Normalisierung auf [0, 1]
    # - Reshape falls nötig

    # TODO: Modell erstellen und trainieren
    autoencoder = Autoencoder(encoding_dim=32)

    # TODO: Compilation und Training
    # TODO: Visualisierung der Ergebnisse

if __name__ == "__main__":
    main()
```

Teilaufgaben:

(a) Implementieren Sie einen einfachen Autoencoder mit 32-dimensionalem Latent Space

(b) Erweitern Sie auf Convolutional Autoencoder

(c) Implementieren Sie Denoising-Funktionalität

(d) Visualisieren Sie den latenten Raum mit t-SNE oder PCA

(e) Experimentieren Sie mit verschiedenen Latent Space Größen (2, 8, 32, 128)

5.2 VAE Implementation

Aufgabe 5.2: Variational Autoencoder

Listing 2: VAE für MNIST

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np

class VAE(keras.Model):
    def __init__(self, latent_dim):
        super(VAE, self).__init__()
        self.latent_dim = latent_dim

        # Encoder
        self.encoder = keras.Sequential([
            # TODO: Encoder-Architektur
```

```
        ])

        # Latent space - separate outputs für  und log(²)
        self.mu_layer = layers.Dense(latent_dim)
        self.log_var_layer = layers.Dense(latent_dim)

        # Decoder
        self.decoder = keras.Sequential([
            # TODO: Decoder-Architektur
        ])

    def encode(self, x):
        """Encoded Input zu  und log(²)"""
        h = self.encoder(x)
        mu = self.mu_layer(h)
        log_var = self.log_var_layer(h)
        return mu, log_var

    def reparameterize(self, mu, log_var):
        """Reparameterization Trick"""
        # TODO: Implementierung
        # z =  +  *  , wobei  ~ N(0, I)
        pass

    def decode(self, z):
        """Dekodiert latente Repräsentation"""
        return self.decoder(z)

    def call(self, x):
        mu, log_var = self.encode(x)
        z = self.reparameterize(mu, log_var)
        return self.decode(z), mu, log_var

def vae_loss(x, x_reconstructed, mu, log_var):
    """VAE Loss: Reconstruction + KL Divergence"""
    # TODO: Implementieren Sie ELBO
    # reconstruction_loss = ...
    # kl_loss = ...
    # return reconstruction_loss + kl_loss
    pass

def generate_images(model, n=10):
    """Generiert neue Bilder durch Sampling aus dem Prior"""
    # TODO: Sample z aus N(0, I) und dekodiere
    pass

def plot_latent_space(model, test_data, test_labels):
    """Visualisiert den 2D latenten Raum (nur für latent_dim=2)"""
    # TODO: Encode test_data und plotte nach Labels eingefärbt
    pass

```

```python
def main():
    # TODO: VAE Training und Evaluation
    pass

if __name__ == "__main__":
    main()
```

Teilaufgaben:

(a) Implementieren Sie den Reparameterization Trick

(b) Vervollständigen Sie die VAE Loss-Funktion

(c) Trainieren Sie das VAE auf MNIST

(d) Implementieren Sie Bildgenerierung durch Prior-Sampling

(e) Für 2D Latent Space: Visualisieren Sie die Verteilung verschiedener Ziffern

(f) Experimentieren Sie mit β-VAE ($\beta > 1$)

5.3 GAN Implementation

Aufgabe 5.3: Simple GAN

Listing 3: GAN für MNIST

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np

def build_generator(latent_dim):
    """Erstellt Generator-Netzwerk"""
    model = keras.Sequential([
        # TODO: Generator-Architektur
        # - Input: Latent Vector (z)
        # - Dense Layer(s) mit ReLU
        # - Output: 28x28x1 Bild mit tanh-Aktivierung
    ])
    return model

def build_discriminator():
    """Erstellt Discriminator-Netzwerk"""
    model = keras.Sequential([
        # TODO: Discriminator-Architektur
        # - Input: 28x28x1 Bild
        # - Conv2D oder Dense Layer(s)
        # - Output: Einzelne Probability (echt/fake)
    ])
    return model

class GAN:
    def __init__(self, latent_dim=100):
```

```
28          self.latent_dim = latent_dim
29
30          self.generator = build_generator(latent_dim)
31          self.discriminator = build_discriminator()
32
33          # Optimizers
34          self.generator_optimizer = keras.optimizers.Adam(1e-4)
35          self.discriminator_optimizer = keras.optimizers.Adam(1e-4)
36
37          # Loss function
38          self.cross_entropy = keras.losses.BinaryCrossentropy()
39
40      def discriminator_loss(self, real_output, fake_output):
41          """Discriminator Loss"""
42          # TODO: Implementierung
43          # real_loss = cross_entropy(ones, real_output)
44          # fake_loss = cross_entropy(zeros, fake_output)
45          # return real_loss + fake_loss
46          pass
47
48      def generator_loss(self, fake_output):
49          """Generator Loss"""
50          # TODO: Implementierung
51          # Generator will Discriminator "täuschen" -> Labels = 1
52          pass
53
54      @tf.function
55      def train_step(self, real_images, batch_size):
56          """Ein Trainingsschritt für beide Netzwerke"""
57          noise = tf.random.normal([batch_size, self.latent_dim])
58
59          with tf.GradientTape() as gen_tape, tf.GradientTape() as
                disc_tape:
60              # TODO: Forward Pass
61              # generated_images = self.generator(noise)
62              # real_output = self.discriminator(real_images)
63              # fake_output = self.discriminator(generated_images)
64
65              # TODO: Loss-Berechnung
66              # gen_loss = self.generator_loss(fake_output)
67              # disc_loss = self.discriminator_loss(real_output,
                    fake_output)
68              pass
69
70          # TODO: Gradient-Berechnung und Update
71
72      def generate_images(self, n=16):
73          """Generiert n Bilder"""
74          noise = tf.random.normal([n, self.latent_dim])
75          generated_images = self.generator(noise)
76          return generated_images
```

```
77
78 def main():
79     # TODO: GAN Training
80     pass
81
82 if __name__ == "__main__":
83     main()
```

Teilaufgaben:

(a) Implementieren Sie Generator und Discriminator

(b) Vervollständigen Sie die Loss-Funktionen

(c) Implementieren Sie den alternierenden Trainingsprozess

(d) Visualisieren Sie generierte Bilder während des Trainings

(e) Experimentieren Sie mit verschiedenen Architekturen

(f) Implementieren Sie WGAN-Verlust (optional)

6 Transfer Learning und Fine-Tuning

6.1 Feature Extraction vs. Fine-Tuning

Aufgabe 6.1: ImageNet Pre-trained Models

(a) Laden Sie ein pre-trained ResNet50 (ImageNet) ohne Top-Layer

(b) Implementieren Sie **Feature Extraction**:

- Alle ConvNet-Layer einfrieren (trainable=False)
- Nur neue Classifier-Layer trainieren

(c) Implementieren Sie **Fine-Tuning**:

- Erst Feature Extraction für wenige Epochen
- Dann ausgewählte obere Layer äuftauen"
- Mit sehr kleiner Learning Rate weitertrainieren

(d) Vergleichen Sie beide Ansätze auf einem kleinen Dataset (z.B. CIFAR-10)

(e) Diskutieren Sie: Wann verwenden Sie welchen Ansatz?

6.2 Domain Adaptation

Aufgabe 6.2: Cross-Domain Transfer

(a) Trainieren Sie ein CNN auf MNIST

(b) Evaluieren Sie auf Fashion-MNIST ohne Re-Training

(c) Implementieren Sie Domain Adaptation Techniken:

- Gradual Unfreezing
- Discriminative Learning Rates
- Data Augmentation für Ziel-Domain

(d) Messen Sie Performance-Verbesserungen

(e) Analysieren Sie: Welche Features sind transferierbar?

7 Moderne Optimierungsverfahren

7.1 Adaptive Learning Rate Methods

Aufgabe 7.1: Optimizer-Vergleich

(a) Implementieren Sie von Grund auf:

- **SGD with Momentum**: $v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$
- **AdaGrad**: $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla_\theta J(\theta)$
- **Adam**: Kombination aus Momentum und adaptive Learning Rates

(b) Testen Sie alle Optimizer auf demselben Problem (z.B. MNIST-Klassifikation)

(c) Plotten Sie Convergence-Curves und Learning Rate Schedules

(d) Diskutieren Sie Vor- und Nachteile jedes Verfahrens

7.2 Learning Rate Scheduling

Aufgabe 7.2: Advanced Scheduling

(a) Implementieren Sie verschiedene LR-Schedules:

- **Step Decay**: LR halbieren alle N Epochen
- **Exponential Decay**: $\text{LR} = \text{LR}_0 \cdot e^{-kt}$
- **Cosine Annealing**: $\text{LR} = \text{LR}_{\min} + \frac{1}{2}(\text{LR}_{\max} - \text{LR}_{\min})(1 + \cos(\frac{T_{\text{cur}}}{T_{\max}}\pi))$
- **Warm Restarts**: Periodische LR-Resets

(b) Vergleichen Sie die Auswirkungen auf Training-Stabilität und finale Performance

(c) Diskutieren Sie: Wann ist welcher Schedule geeignet?

8 Verständnisfragen

Aufgabe 8.1: Generative Models Comparison

(a) Vergleichen Sie VAEs, GANs und Diffusion Models bezüglich:

- **Training-Stabilität**
- **Sample-Qualität**
- **Mode Coverage**
- **Latent Space Interpretierbarkeit**
- **Computational Complexity**

(b) Diskutieren Sie spezifische Anwendungsfälle für jedes Modell

(c) Erklären Sie das **Mode Collapse** Problem in GANs und Lösungsansätze

(d) Beschreiben Sie das **Posterior Collapse** Problem in VAEs

Aufgabe 8.2: Theoretical Foundations

(a) Erklären Sie die Information-Theoretic Perspektive auf Autoencoders:

- Information Bottleneck Principle
- Mutual Information zwischen Input und Latent Space
- Rate-Distortion Theory

(b) Diskutieren Sie die Verbindung zwischen:

- VAE ELBO und Maximum Likelihood Estimation
- GAN Minimax-Spiel und Optimal Transport
- Diffusion Models und Score Matching

9 Zusatzaufgaben (Optional)

Aufgabe 9.1: Conditional Generation

(a) Implementieren Sie Conditional VAE (CVAE) für MNIST

(b) Erweitern Sie Ihr GAN zu einem Conditional GAN (cGAN)

(c) Vergleichen Sie beide Ansätze für kontrollierte Generierung

(d) Implementieren Sie Style Transfer mit pre-trained Networks

Aufgabe 9.2: Advanced Architectures

(a) Recherchieren Sie StyleGAN-Architektur

(b) Implementieren Sie Progressive Growing für GANs

(c) Experimentieren Sie mit Self-Attention in Generative Models

(d) Implementieren Sie Spectral Normalization für GAN-Stabilität

Hinweise und Tipps

Zu den Mathematik-Aufgaben

- **Wahrscheinlichkeitstheorie:** Verstehen Sie KL-Divergenz und andere Distanzmaße
- **Reparameterization Trick:** Essentiell für gradient-based Optimization in VAEs
- **Numerical Stability:** Verwenden Sie log-space Berechnungen wo möglich
- **Minimax-Spiele:** Verstehen Sie die Dynamik zwischen Generator und Discriminator

Zu den Programmieraufgaben

- **Training Instability:** GANs sind notorisch schwer zu trainieren - experimentieren Sie mit Hyperparametern
- **Model Monitoring:** Tracken Sie multiple Metriken (Loss, FID, IS, etc.)
- **Computational Resources:** Generative Models benötigen viel Rechenzeit
- **Evaluation:** Qualitative Evaluation ist oft wichtiger als quantitative Metriken

Häufige Fehler vermeiden

- **VAE KL-Collapse:** Balance zwischen Reconstruction und KL-Term
- **GAN Mode Collapse:** Verschiedene Techniken zur Diversität
- **Learning Rate:** GANs benötigen oft verschiedene LRs für G und D
- **Data Normalization:** Konsistente Normalisierung zwischen Training und Generation

Weiterführende Ressourcen

- **Bücher:**
  - "Deep Learning Goodfellow et al., Kapitel 14, 16, 20
  - "Probabilistic Machine Learning Kevin Murphy
- **Fundamental Papers:**
  - Äuto-Encoding Variational Bayes Kingma & Welling
  - "Generative Adversarial Networks Goodfellow et al.
  - "Denoising Diffusion Probabilistic Models Ho et al.
  - "Wasserstein GAN Arjovsky et al.
- **Online:**

- ”Tutorial on Variational Autoencoders Carl Doersch
- ”GAN Lab Interactive GAN Visualization
- CS236: Deep Generative Models (Stanford)
- Distill.pub: ”Visualizing Neural Networks with the Grand Tour”