HSLU Hochschule Luzern

Computer Vision

Generative AI: Variational Autoencoders,
Generative Adversarial Networks, Diffusion Models

Dr. Vagia Tsiminaki, Dr. Umberto Michelucci, Dr. Aygul Zagidullina

The GenAI Era

HSLU Hochschule Luzern

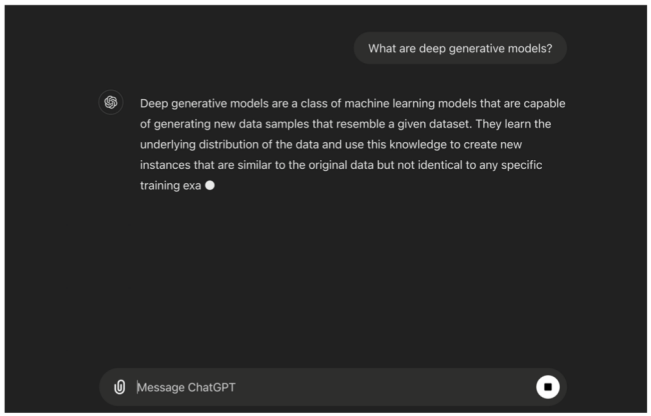
Text to Image Generation

Generated by Stable Diffusion 3 Medium.
Prompt: teddy bear teaching a course, with "generative models" written on blackboard

Text to Video Generation

Generated by Sora

Chatbot and natural language conversation



Text to 3D Model Generation

HSLU Hochschule Luzern

What are Generative Models?

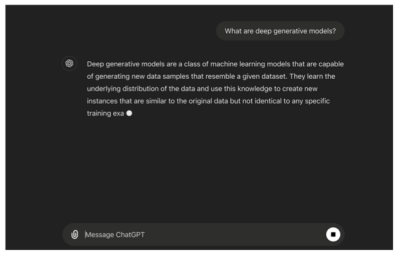
HSLU Hochschule Luzern

What do these scenarios have in common?

- There are **multiple** predictions given one input.
- Some predictions are **more plausible**.
- Predictions are **mode complex, more informative** than input.

Generated by Stable Diffusion 3 Medium.
Prompt: teddy bear teaching a course, with "generative models" written on blackboard

Generated by Sora

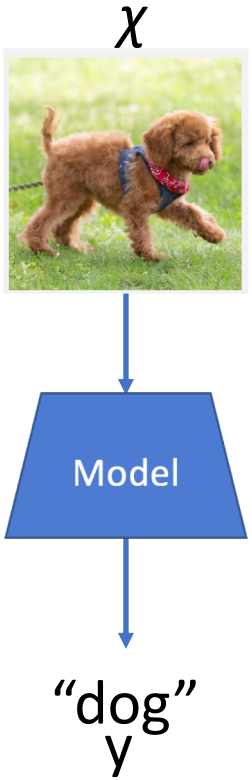


Discriminative vs. Generative Models

HSLU Hochschule Luzern

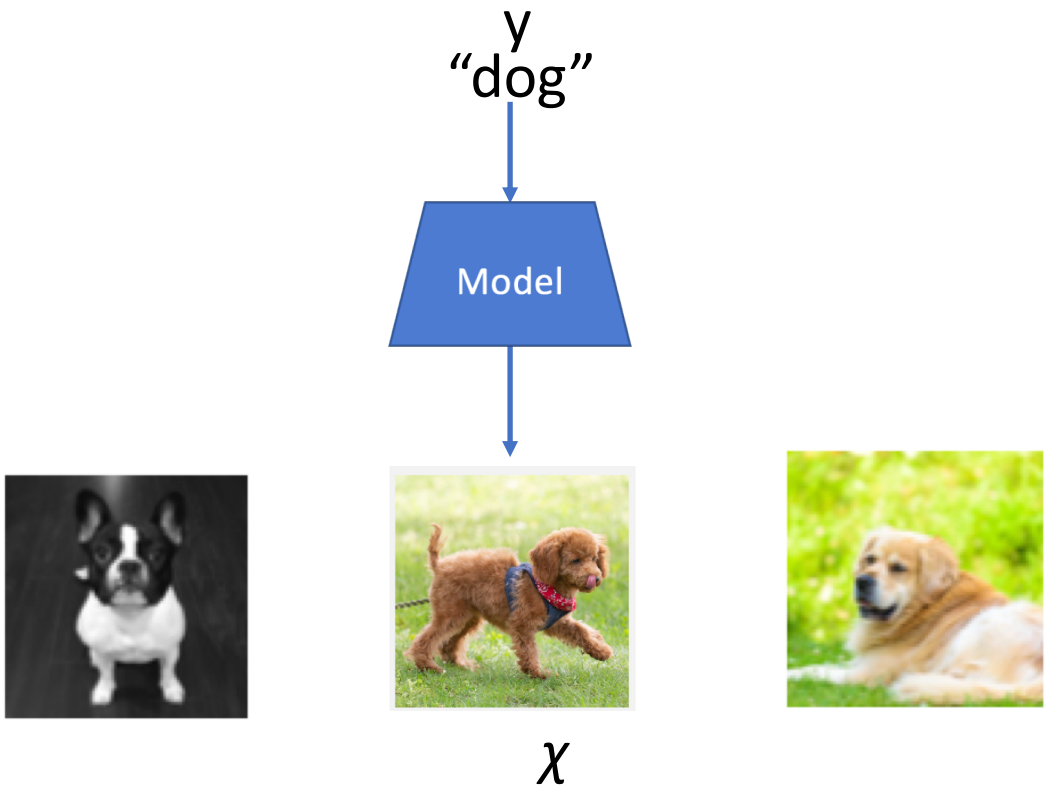
Discriminative

- sample χ => label
- One desired output



Generative

- label y => sample χ
- many possible outputs

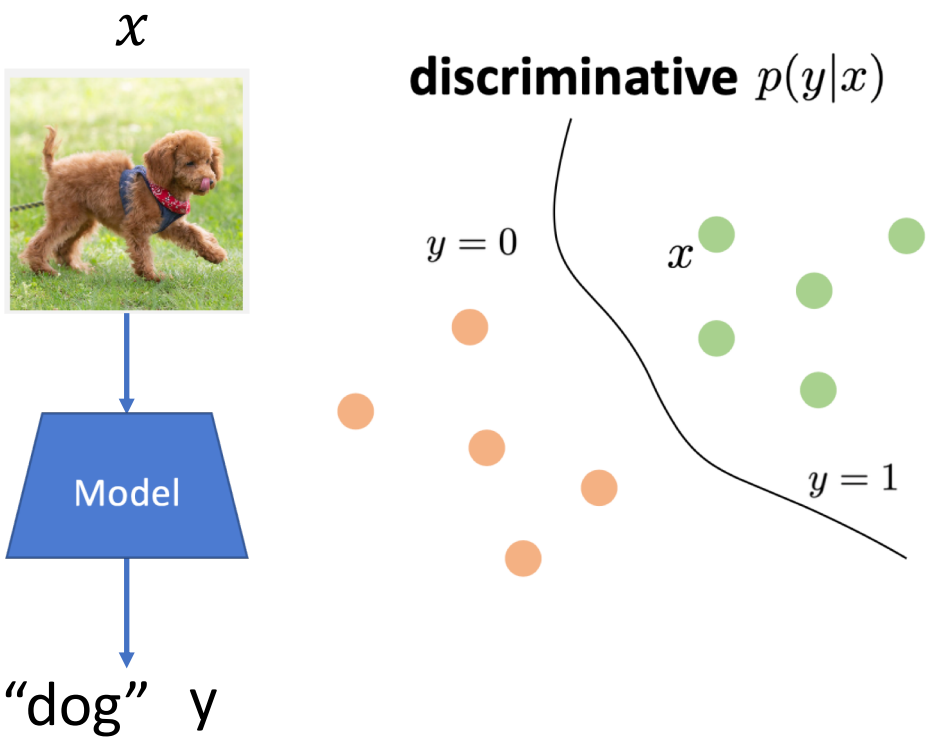


Discriminative vs. Generative Models

HSLU Hochschule Luzern

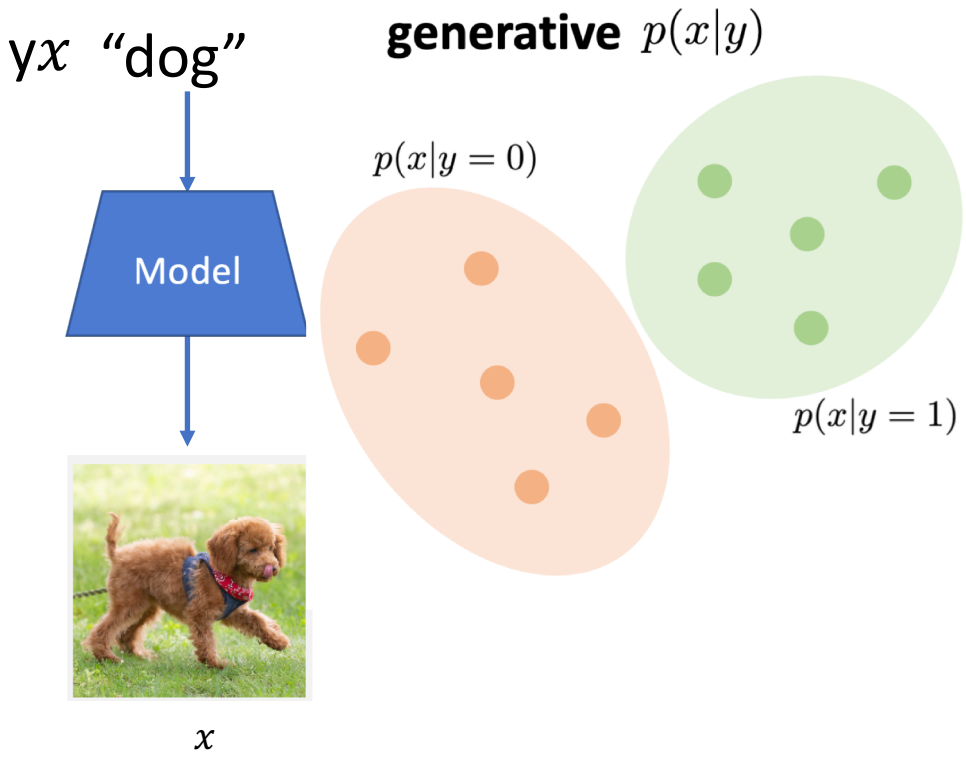
Discriminative

- sample χ => label
- One desired output



Generative

- label y => sample χ
- many possible outputs



Generative Models with Probabilistic Modeling

HSLU Hochschule Luzern

- Assuming underlying distributions of data generation process

Bayes' rule

discriminative

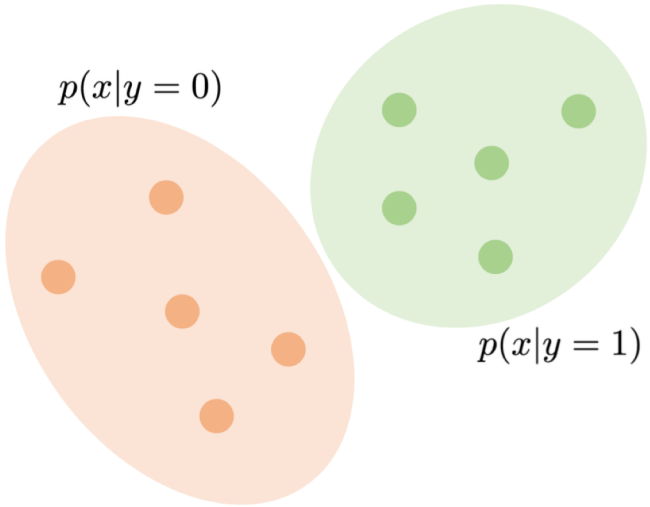
$$p(y|x) = p(x|y)\frac{p(y)}{p(x)}$$

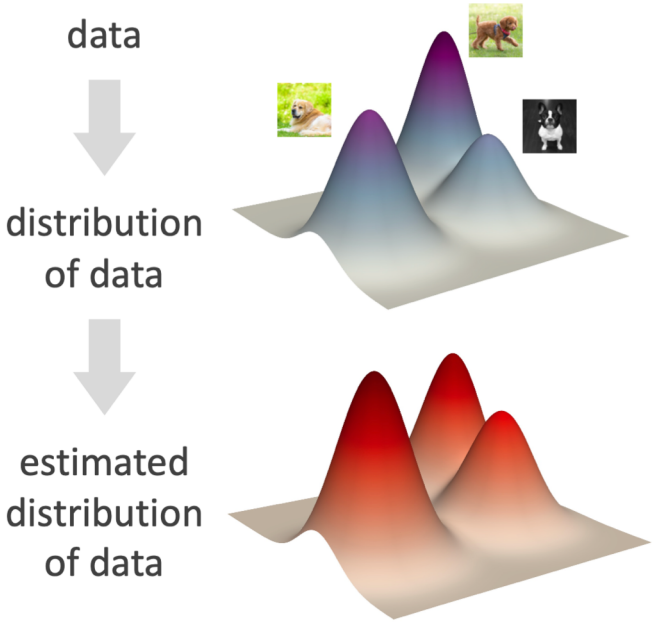
generative

assuming known prior

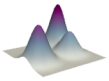
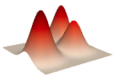
constant for given x

generative $p(x|y)$





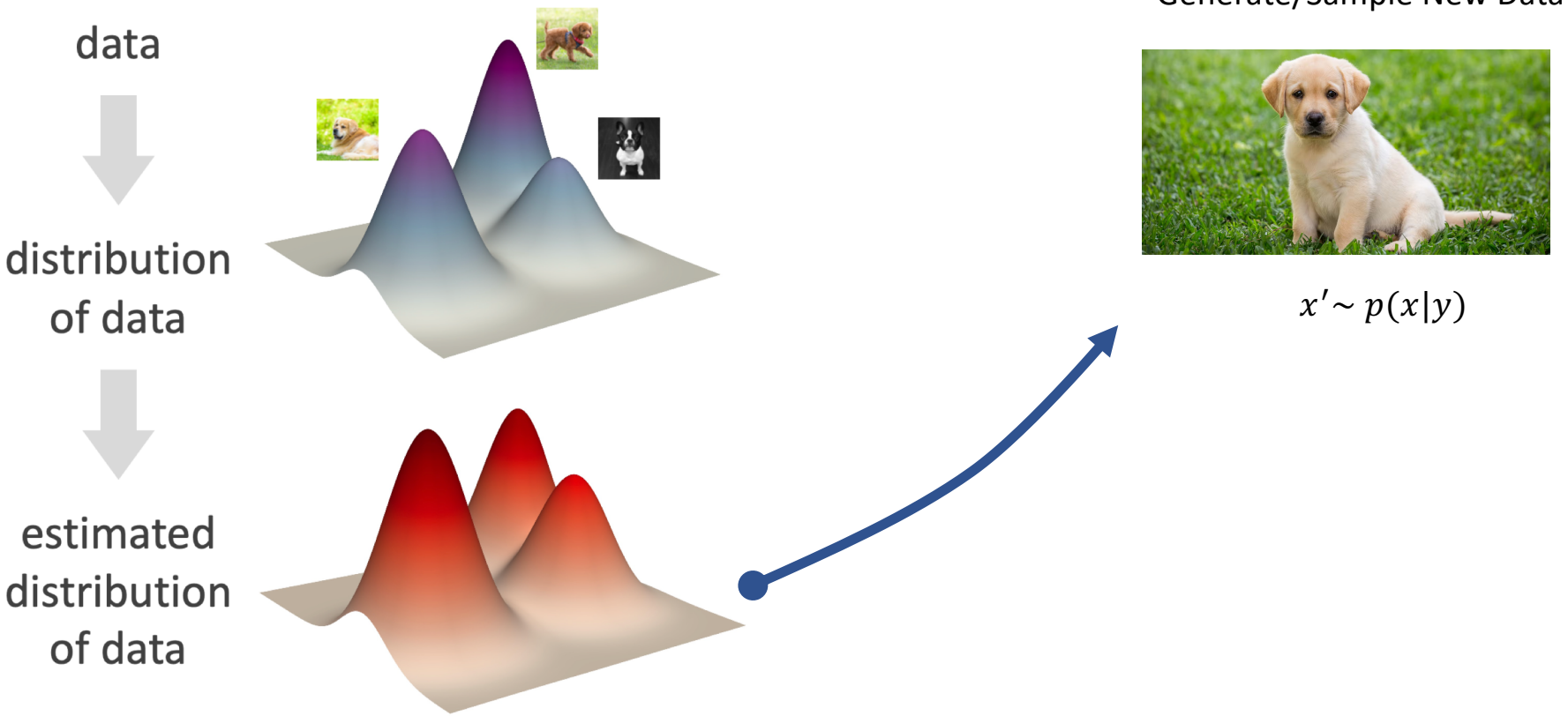
- Optimize a loss function

$\mathcal{L}($  ,  $)$

Generative Models with Probabilistic Modeling

HSLU Hochschule Luzern

- Assuming underlying distributions of data generation process

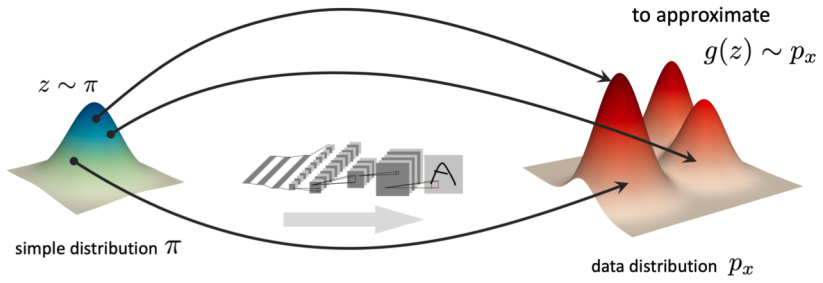


Deep Generative Models

HSLU Hochschule Luzern

- Deep Learning is **representation learning.**
- Learning to represent data instances.
  - Mapping data to feature spaces: $x \rightarrow f(x)$
  - Minimize the loss with the target: $\mathcal{L}(y, f(x))$

- Learning to represent probability distributions.
  - Map a simple distribution to a complex one: $\pi \rightarrow g(\pi))$
  - Minimize the loss with data distribution $\mathcal{L}(p_x, g(\pi))$



Deep Generative Models: Key Components

HSLU Hochschule Luzern

- Formulate the problem as **probabilistic modeling** $p(x|y)$.
  - y is the label, more abstract, less informative.
  - x the data, sample, observation, more concrete, more informative.

- **Deep neural networks** to represent data and their probability distributions.

- Define the **Loss function** to measure how good the predictions is

- **Optimize** the parameters of the network.

- **Sampling** step to generate new data.

Deep Generative Models for Computer Vision

HSLU Hochschule Luzern

Probabilistic Modeling

$p(x|y)$

Text to Image Generation

Generated by Stable Diffusion 3 Medium.
Prompt: teddy bear teaching a course, with "generative models" written on blackboard

y: *Input text prompt*

x: *Generated Visual* Content

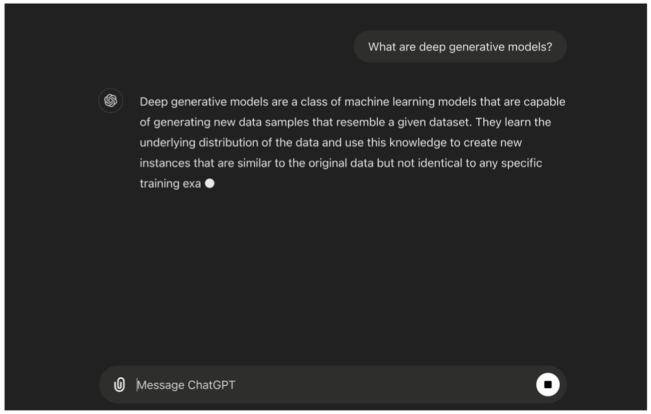
Text to Video Generation

Generated by Sora

y : *Input text prompt*

x: *Generated Visual* Content

Chatbot and natural language conversation



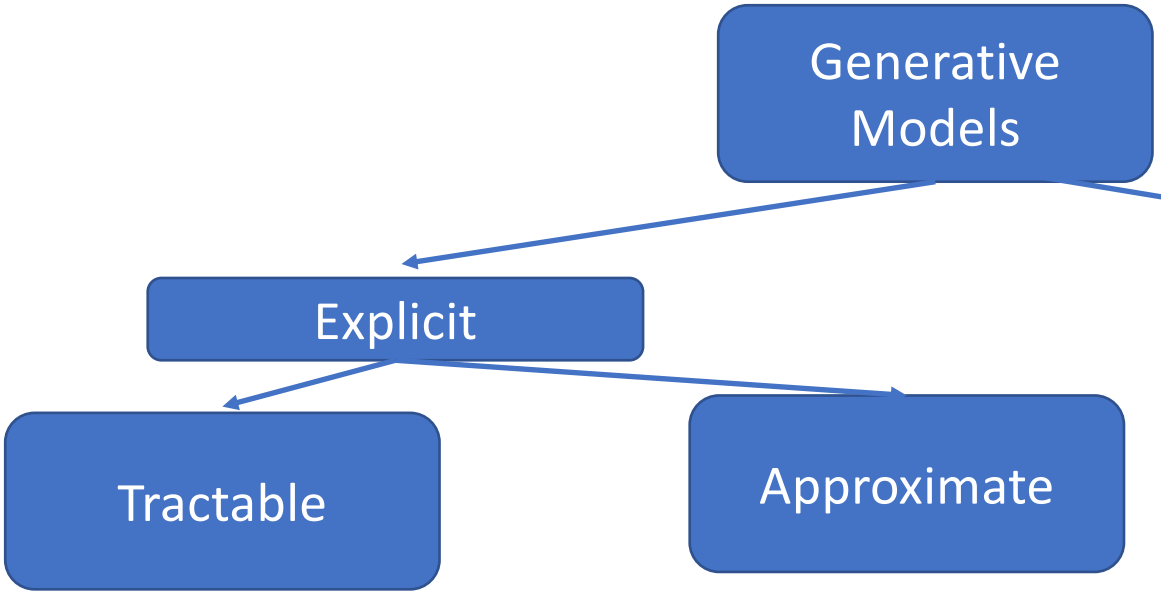
y : *Input prompt*

x: *Response*

Text to 3D Model Generation

Taxonomy Generative Models

HSLU Hochschule Luzern



GANs

2014 - "Generative Adversarial Networks" by Ian Goodfellow et al..

VAEs

2013 - ”Auto-Encoding Variational Bayes" by Diederik P. Kingma and Max Welling.

Autoregressive Models

Normalizing Flows

Diffusion Models

Denoising Diffusion Probabilistic Models (DDPM): Introduced by Jonathan Ho et al. in 2020,

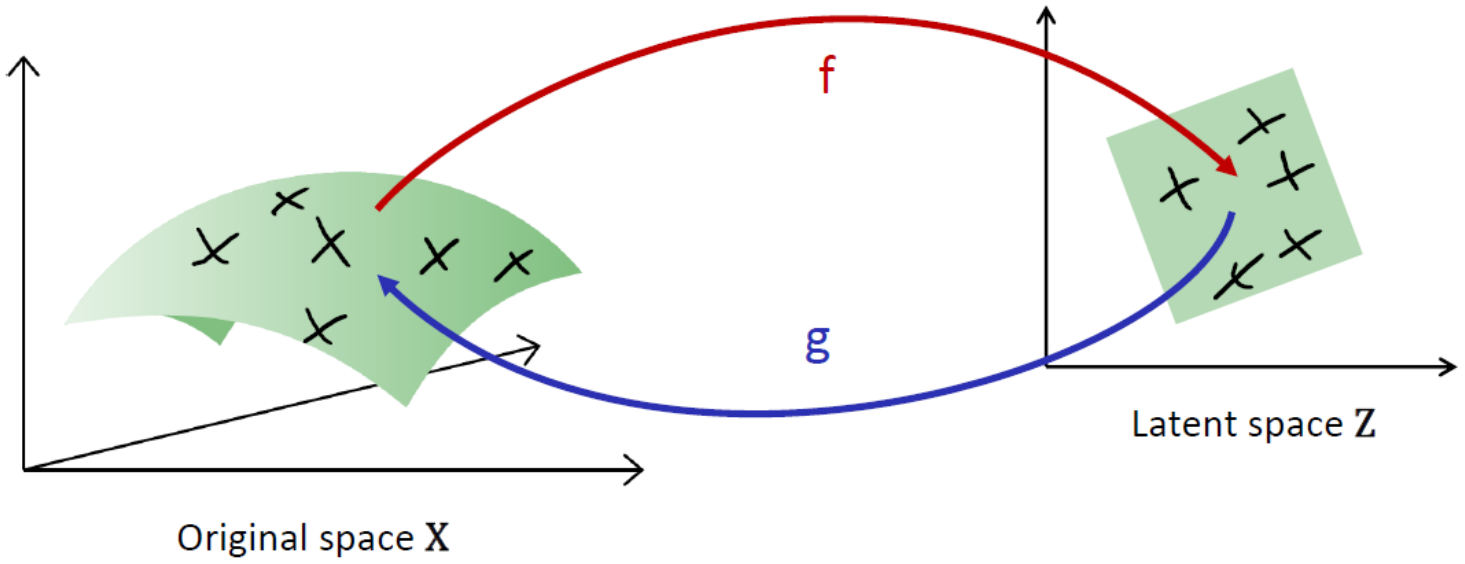
HSLU Hochschule Luzern

Variational Autoencoders

Autoencoders

HSLU Hochschule Luzern

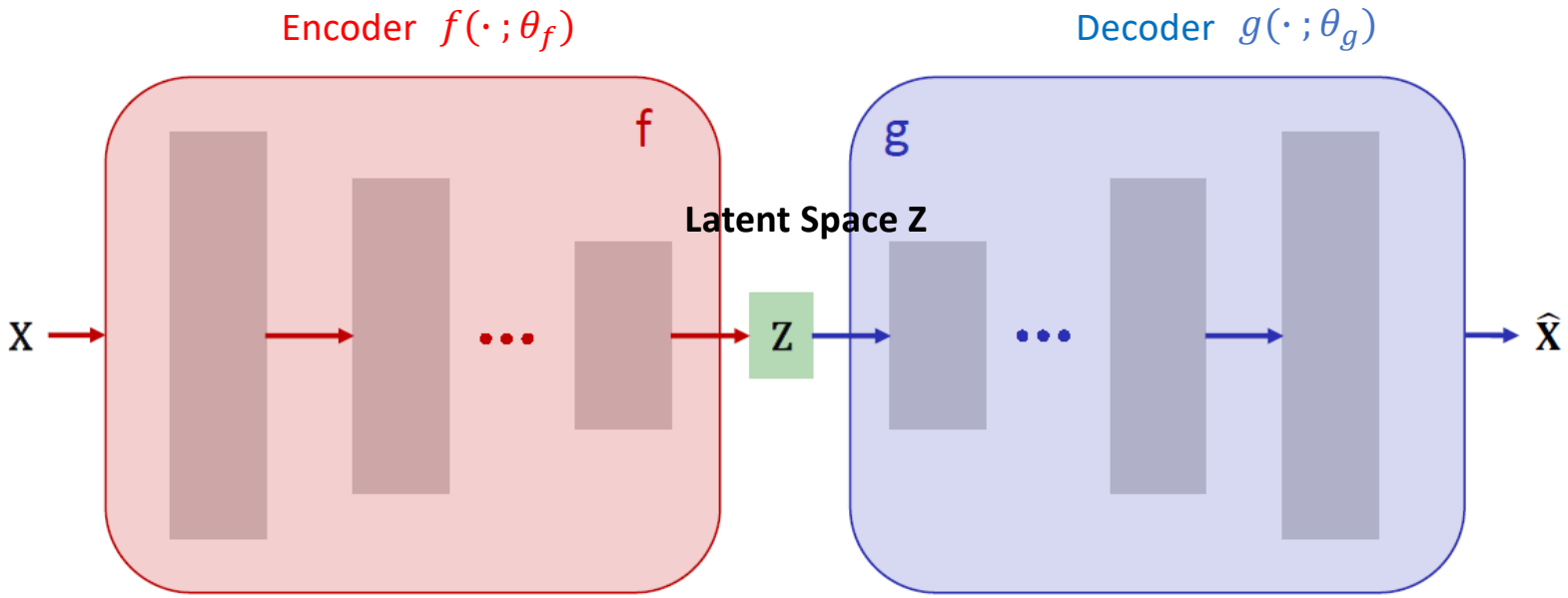
- Encoder f: Projects input space X into a latent space Z.
- **Latent Space Z**: A compressed representation. Lower dimensionality than the input space.
- Decoder g: Maps samples from latent space Z to input space X.
- The [g *o* f] approximates the identity function.



Autoencoders

HSLU Hochschule Luzern

Parametrize the mappings with neural networks.



Training: Solve for the best parameters of the networks that minimizes the error.

$$\hat{\theta}_f, \hat{\theta}_g = \underset{\theta_f,\theta_g}{\operatorname{argmin}} \sum_n^N \|x_n - \hat{x}_n\|^2$$

$$\hat{\theta}_f, \hat{\theta}_g = \underset{\theta_f,\theta_g}{\operatorname{argmin}} \sum_n^N \|x_n - g(f(x_n))\|^2$$

Autoencoders

HSLU Hochschule Luzern

Key Points:

- The encoder f and decoder g are deterministic mappings.
- The encoder f outputs fixed latent vector **z** for each input.
- The decoder g takes as input the deterministic output vector **z.** of latent space
- Latent Space **Z** does not follow any specific distribution. There is not notion of structure.

Advantages:

- Easy to train.
- Learn efficient representation of data Z without imposing any prior/specific structure.

Limitations:

- Not uncertainty modeling.
- Sampling from latent space **Z** does not guarantee meaningful output.
- Not generative capability.

Main Application: Feature extraction and data compression.

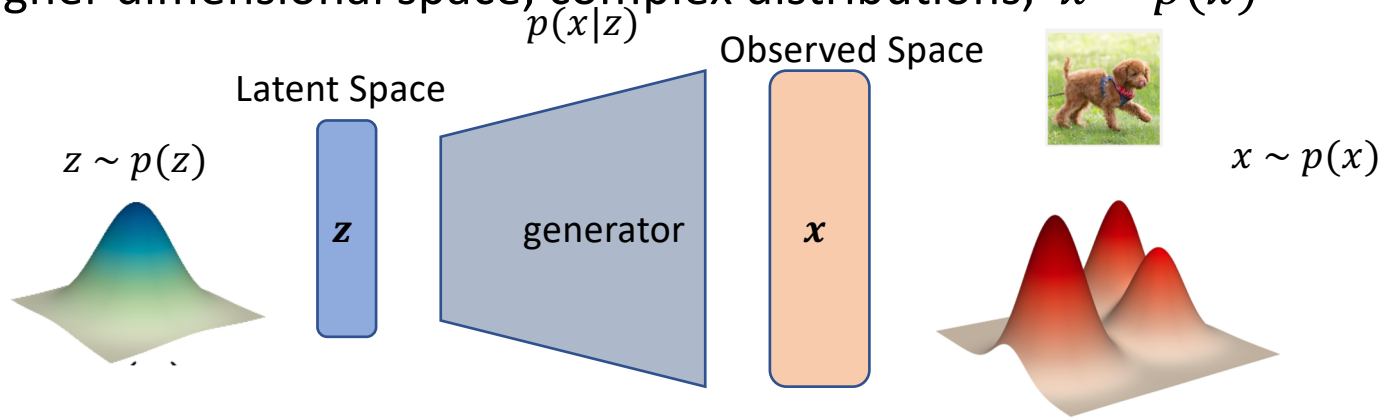
Variational Autoencoders (VAEs)

- Add the generative process.
- Extend autoencoders by incorporating probabilistic elements and by imposing prior knowledge over the latent space **Z.**

Objective: Learn the underlying probability distribution of the data to generate new data samples.

Generation Process:

- Latent Space – z
  - Lower dimensional space, simple distribution, $z \sim p(z)$
- Observed Space - x
  - Higher dimensional space, complex distributions, $x \sim p(x)$

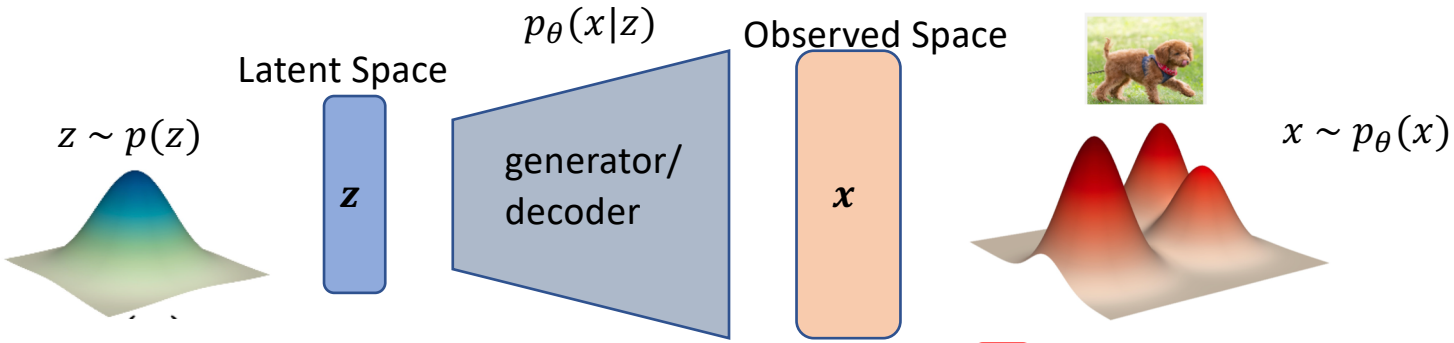


Variational Autoencoders (VAEs)

HSLU Hochschule Luzern

Represent the conditional distribution by a neural network

- Learn the parameters of the network θ **that maximizes the marginal likelihood** $\boldsymbol{p_\theta(x)}$



- Maximize the marginal likelihood of data: $p_\theta(x) = \int_z p_\theta(x|z)p(z)dz$
- $p_\theta(x|z)$: The likelihood of data x given latent variables z, modeled by the deType equation here.coder in a VAE.
- $p(z)$: The prior distribution over latent variables z, typically a standard normal distribution.

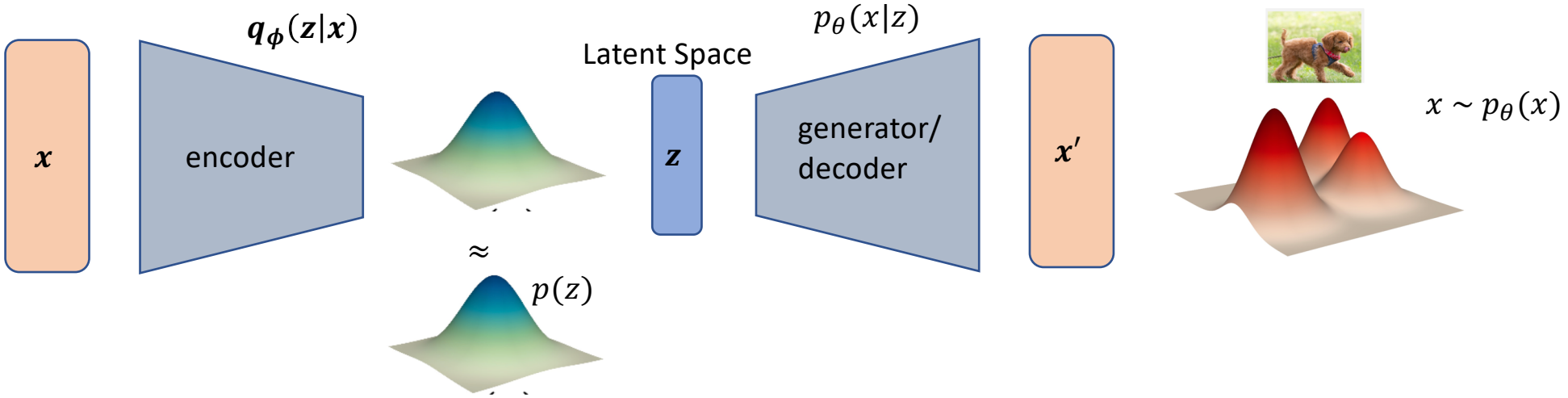
Intractability:

- High dimensional integral. Integrating across all possible values is computational expensive.
- Complex likelihood $p_\theta(x|z)$, modeled by deep neural network.
- Posterior probability distribution is also intractable:$p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}$

Variational Autoencoders (VAEs)

HSLU Hochschule Luzern

Variational Inference: Approximate the intractable posterior :$p_\theta(z|x)$ with a tractable $q_\phi(z|x)$



Maximize the log likelihood $\log p_\theta(x) - D_{KL}(q(z)||p_\theta(z|x))) = \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p_\theta(z))$

Evidence Lower Bound

➔ Minimize $\mathcal{L}_{\theta,\phi}(x) = -\mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] + D_{KL}(q_\phi(z|x)||p_\theta(z))$

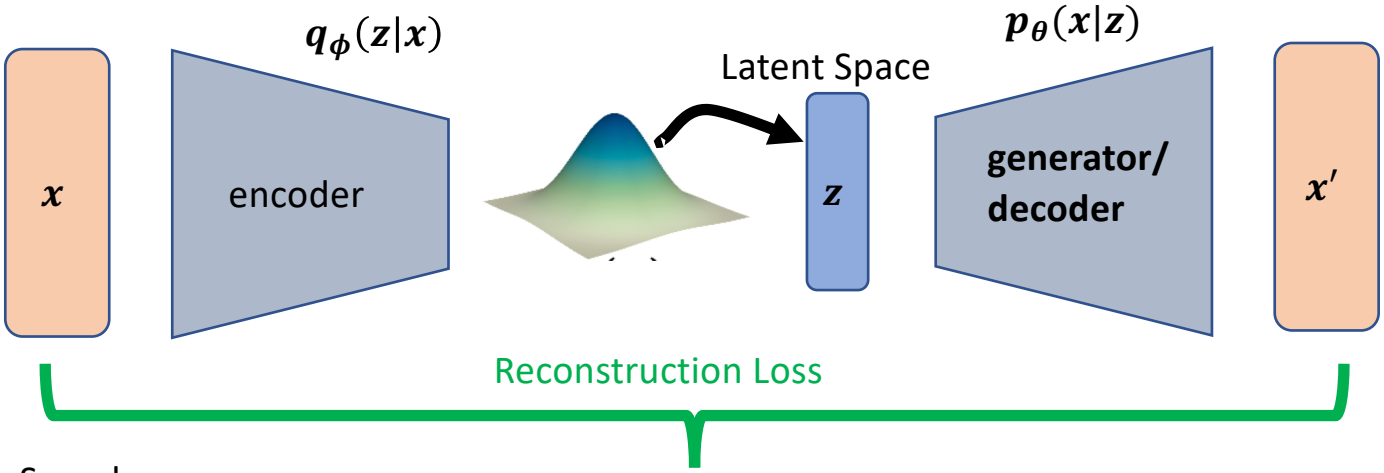
Reconstruction Loss

Regularization Loss

Variational Autoencoders (VAEs)

HSLU Hochschule Luzern

➔ Minimize $\mathcal{L}_{\theta,\phi}(x) = -\mathbb{E}_{z\sim q_\phi(z|x)}\ [\log p_\theta(x|z)] + D_{KL}(q_\phi(z|x)\ ||p_\theta(z))$

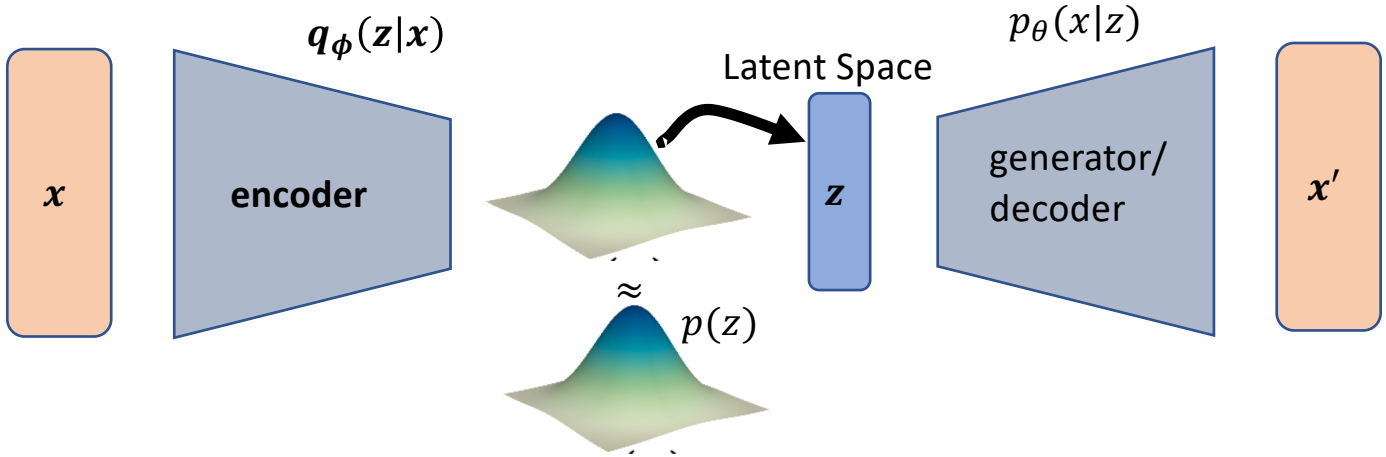


- Sample z.
- Map z through decoder network to x'.
- Model $p_\theta(x|z)$ by Gaussian N(x|x',σ^2)

Variational Autoencoders (VAEs)

HSLU Hochschule Luzern

➔ Minimize $\mathcal{L}_{\theta,\phi}(x) = -\mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] + D_{KL}(q_\phi(z|x)\,||p_\theta(z))$ Regularization Loss

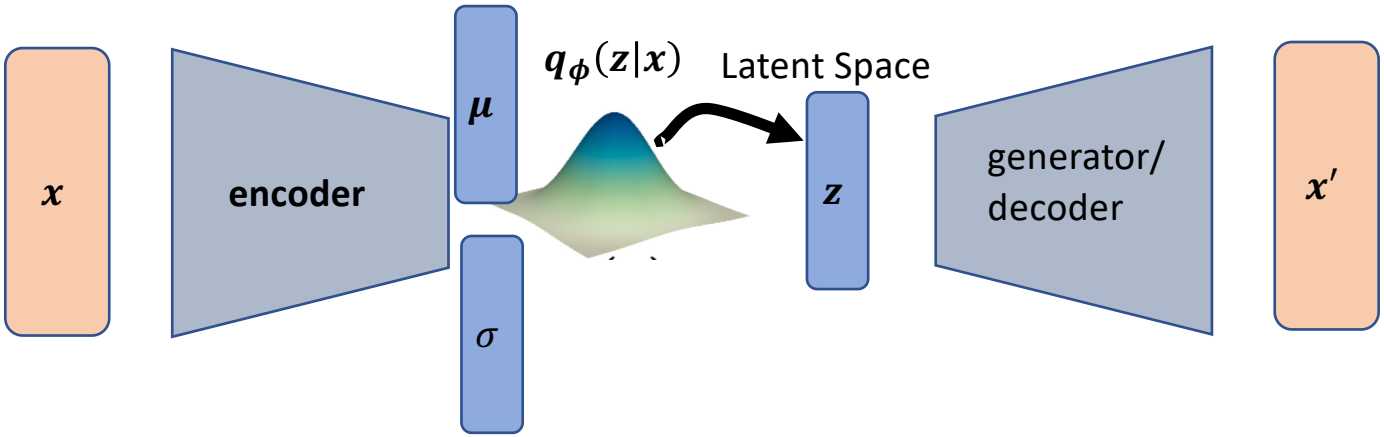


- Let p(z) = N(z|0,I)
- Model $q_\phi(z|x)$ by Gaussian: N(z|μ,σ)
- Map x through encoder network to μ,σ

Variational Autoencoders (VAEs): Training

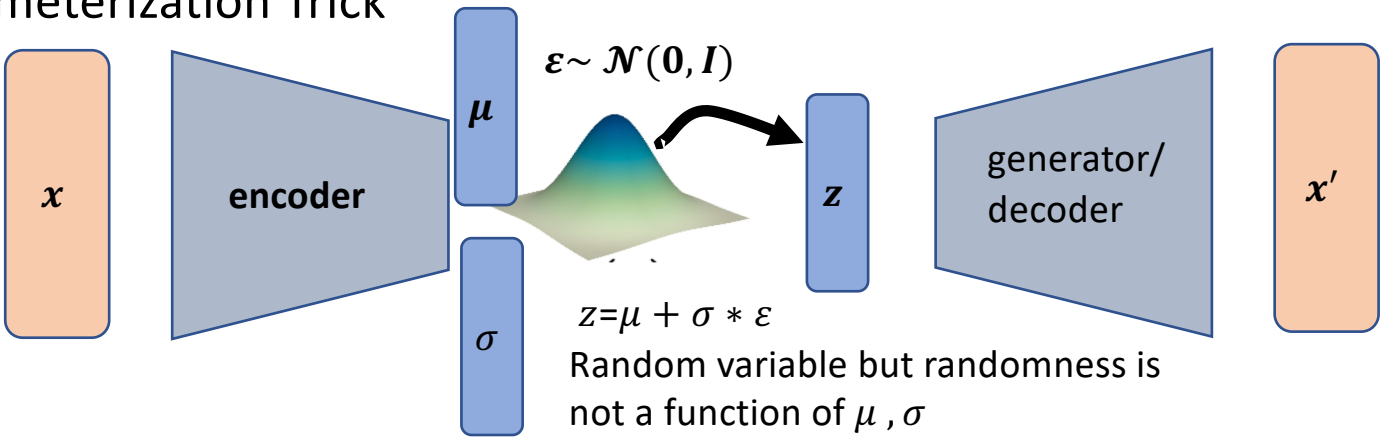
HSLU Hochschule Luzern

- Minimize $\mathcal{L}_{\theta,\phi}(x) = -\mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] + D_{KL}(q_\phi(z|x)\,||p_\theta(z))$



Training is problematic since gradients cannot be back-propagated.

- Reparameterization Trick



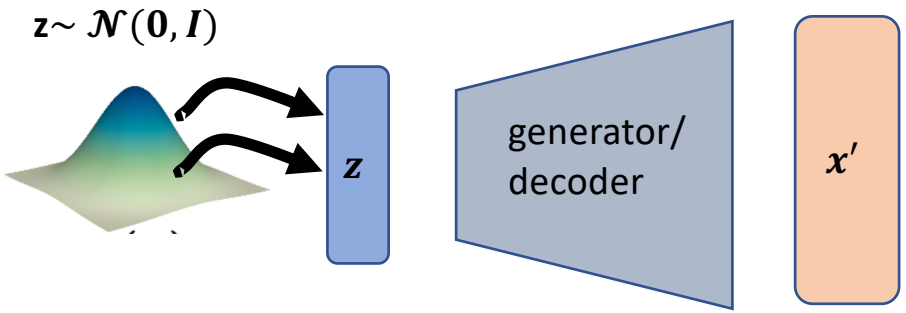
$z=\mu+\sigma*\varepsilon$

Random variable but randomness is not a function of μ , σ

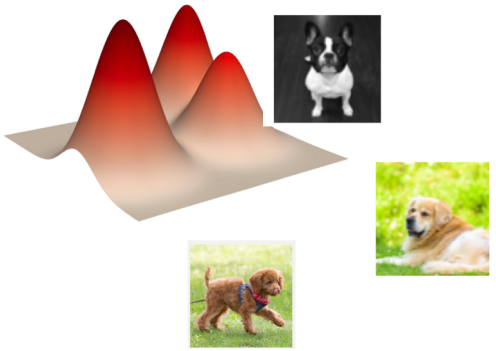
Variational Autoencoders (VAEs): Generation

HSLU Hochschule Luzern

To generate new data we use only decoder:



- Sample latent variable z from Normal distribution
- Map z through decoder/generator net.



Variational Autoencoders (VAEs): Limitations

HSLU Hochschule Luzern

- **Blurry Outputs:**
  - Low-Quality Reconstructions:

  Reason:

  The use of a Gaussian distribution in the decoder's output encourages the model to average over possible outputs
  - Difficulty Capturing Sharp Details

  Reason:

  The probabilistic nature of VAEs, where they model pixel values as probabilities, can lead to smoothing out variations that are important for sharp details.

- **Limited Expressiveness:**

  The latent space in a VAE is designed to be simple (often a standard normal distribution). This simplicity can prevent the model from capturing all the complex features of the data.
- **Mode Collapse:**

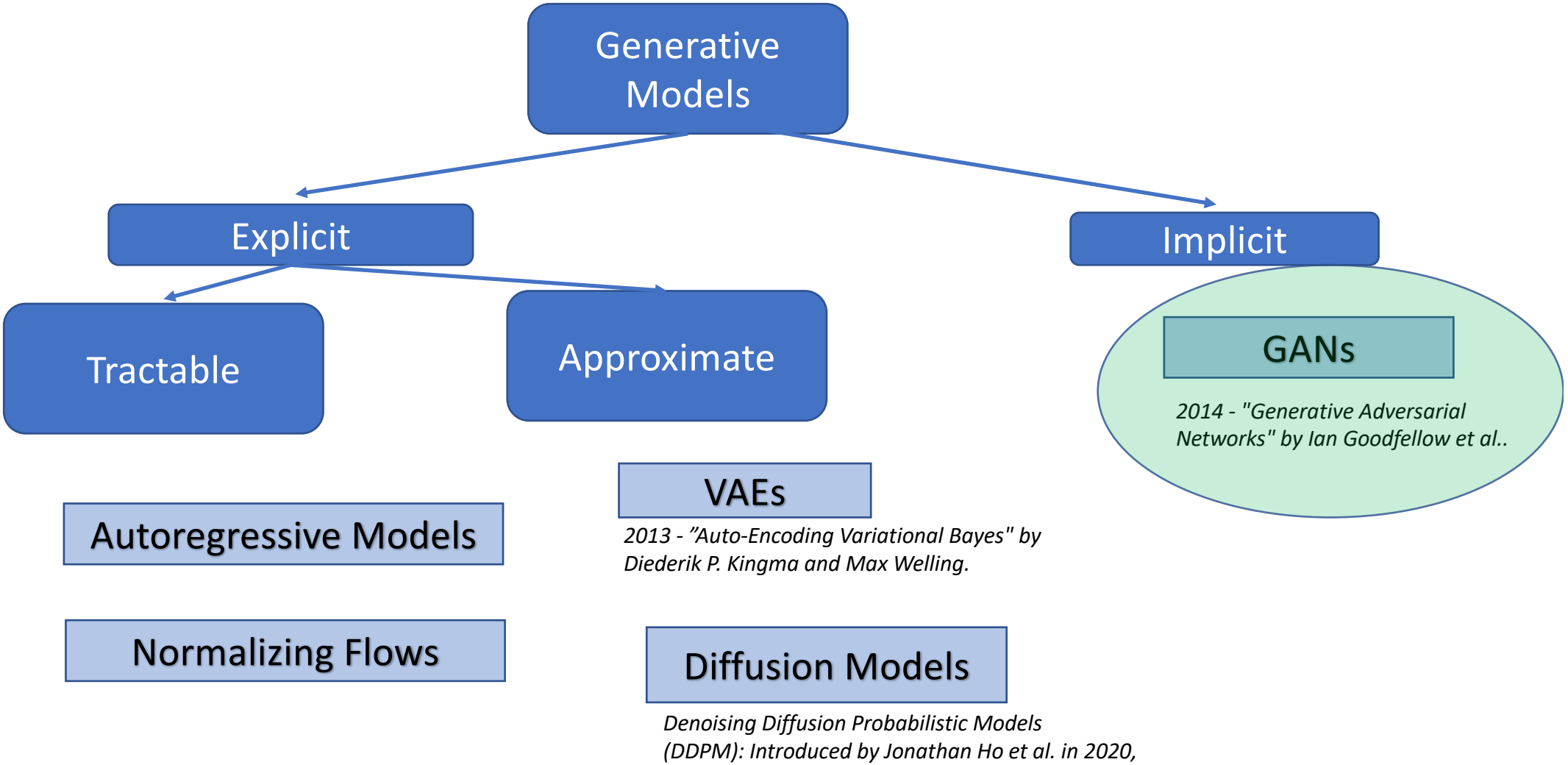
  May fail to capture the full diversity of the data distribution
- **……**

HSLU Hochschule Luzern

Code Discussion

Taxonomy Generative Models

HSLU Hochschule Luzern



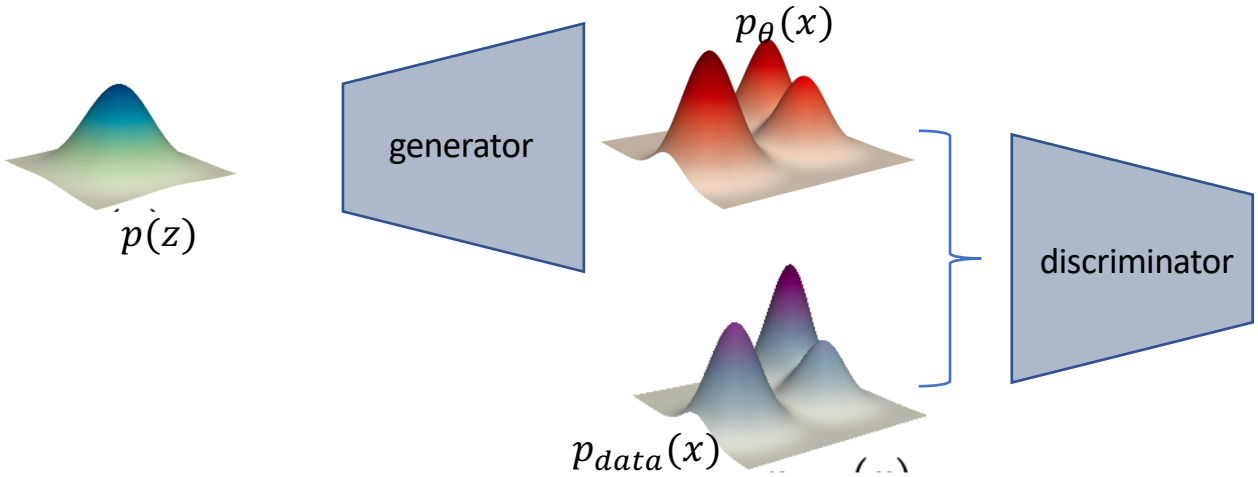
HSLU Hochschule Luzern

Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs)

HSLU Hochschule Luzern

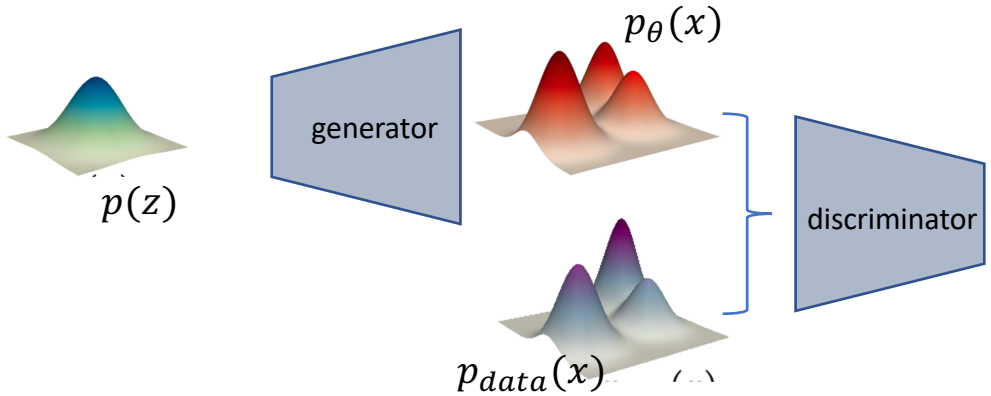
Learn the underlying probability distribution of complex, high-dimensional data **without explicitly** calculating the probability density function.



Idea: Introduce another network discriminator to represent the distribution difference between estimated and real data.

Generative Adversarial Networks (GANs)

HSLU Hochschule Luzern



Key Components:

- **Generator:**
  Try to full the discriminator by generating real-looking data.

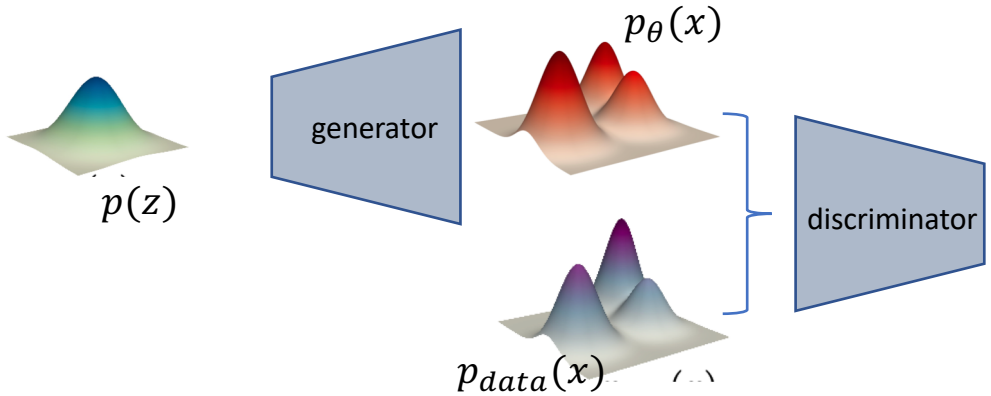
- **Discriminator:**
  Try to distinguish between real and fake data.

- **Adversarial Process:**
  The generator aims to fool the discriminator, while the discriminator aims to correctly identify real vs. fake data.

Step 1: The generator creates fake data from random noise $p(z)$
Step 2: The discriminator evaluates both real and fake data.
Step 3: D-step, The discriminator updates its weights to better classify real vs. fake data.
Step 4: G-step, The generator updates its weights to produce more convincing fake data.

Generative Adversarial Networks (GANs)

HSLU Hochschule Luzern



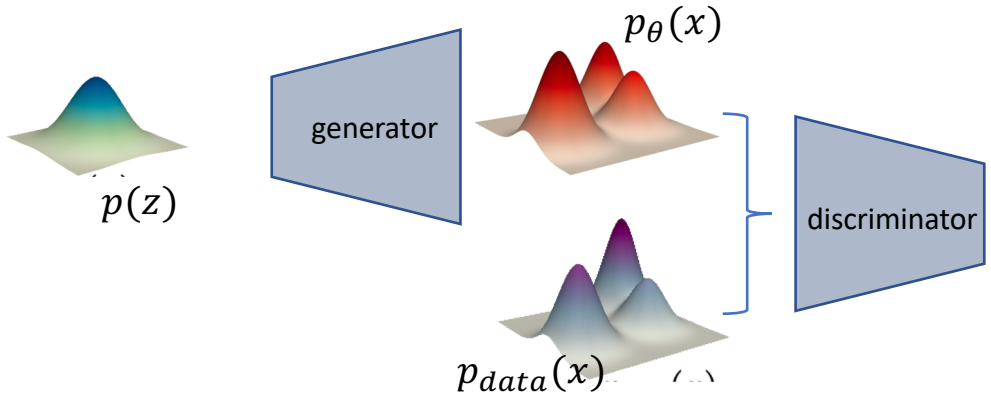
Adversarial Objective:

GANs training process is formulated as a minmax game

$$\min_G \max_D \mathcal{L}(D,G) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{x\sim p_z}[\log(1 - D(G(z))]$$

Generative Adversarial Networks (GANs)

HSLU Hochschule Luzern



Adversarial Objective: D-step

$$\min_G \max_D \mathcal{L}(D,G) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{x\sim p_z}[\log(1 - D(G(z))]$$

➔ Fix G, Optimize D

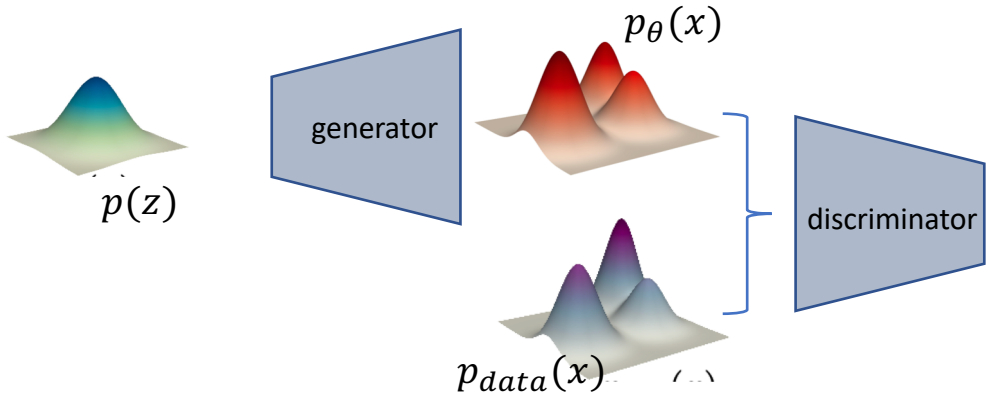
$$\max_D \mathcal{L}(D) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{x\sim p_z}[\log(1 - D(G(z))]$$

Increase for real data x
push to 1

Decrease for generated data
push to 0

Generative Adversarial Networks (GANs)

HSLU Hochschule Luzern



Adversarial Objective: G-step

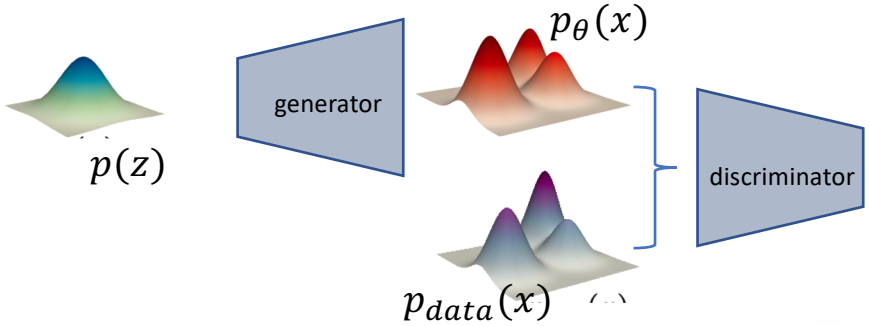
$$\min_G \max_D \mathcal{L}(D,G) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{x\sim p_z}[\log(1-D(G(z))]$$

➔ Fix D, Optimize G

$$\min_G \mathcal{L}(G) = \mathbb{E}_{x\sim p_z}[\log(1-D(G(z))]$$

Increase for generated data
push to 1

Generative Adversarial Networks (GANs)



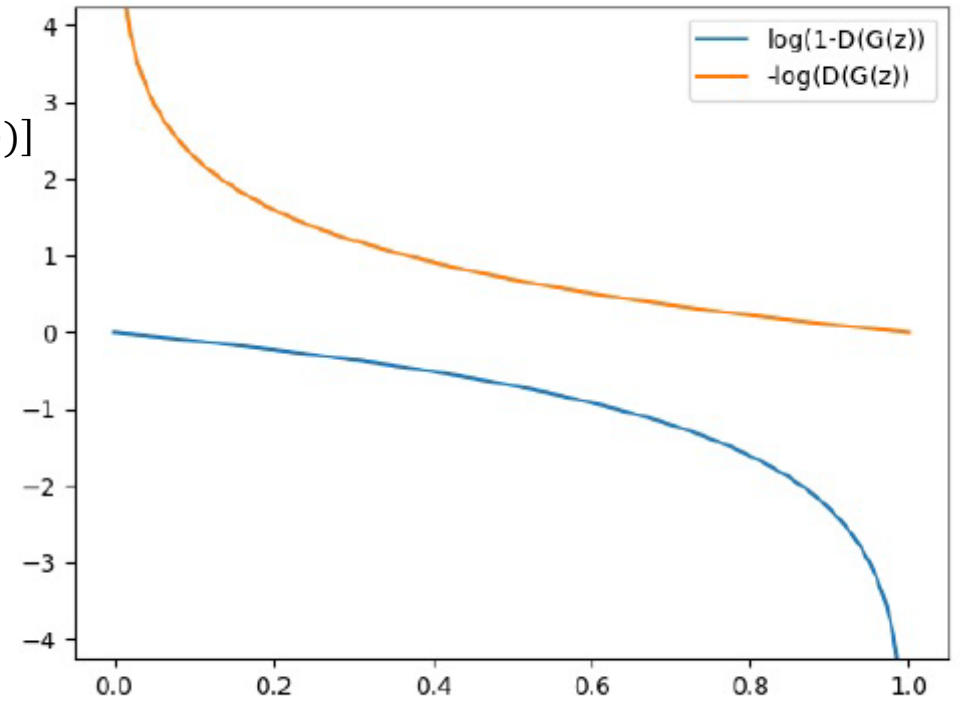
Adversarial Objective: G-step

$$\min_G \max_D \mathcal{L}(D,G) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{x\sim p_z}[\log(1-D(G(z))]$$

➔ Fix D, Optimize G

$$\min_G \mathcal{L}(G) = \mathbb{E}_{x\sim p_z}[\log(1-D(G(z))]$$

Can lead to vanishing gradients



Alternative is :

$$\max_G \mathbb{E}_{x\sim p_z}[\log(D(G(z))]$$

Generative Adversarial Networks (GANs): Training

HSLU Hochschule Luzern

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

[Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014]

Generative Adversarial Networks (GANs): Keras for GANs

HSLU Hochschule Luzern

The most important aspect of using Keras for GANs is the necessity of building a custom training loop (and not using **compile()/fit()** approach.

```python
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      # Calculation of X_{fake}
      generated_images = generator(noise, training=True)

      # Calculation of \hat Y_{real}
      real_output = discriminator(images, training=True)

      # Calculation of \hat Y_{fake}
      fake_output = discriminator(generated_images, training=True)

      # Calculation of L_G
      gen_loss = generator_loss(fake_output)

      # Calculation of L_D
      disc_loss = discriminator_loss(real_output, fake_output)

    #
    # Gradients Calculation
    #
    # Calculation of the gradients of L_G for backpropagation
    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)

    # Calculation of the gradients of L_D for backpropagation
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    #
    # Training Steps A and B
    #
    # Step A
    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))

    # Step B
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))
```

X_{fake}

X_{real}

$L_G = \text{CE}(\hat{Y}_{\text{fake}}, 1)$

$L_D = \text{CE}(\hat{Y}_{\text{fake}}, 0) + \text{CE}(\hat{Y}_{\text{real}}, 1)$

Generative Adversarial Networks (GANs)

HSLU Hochschule Luzern

Advantages:

- High-Quality Data Generation, better than VAEs.
- Unsupervised Learning
- Implicit Density Modeling

Limitations:

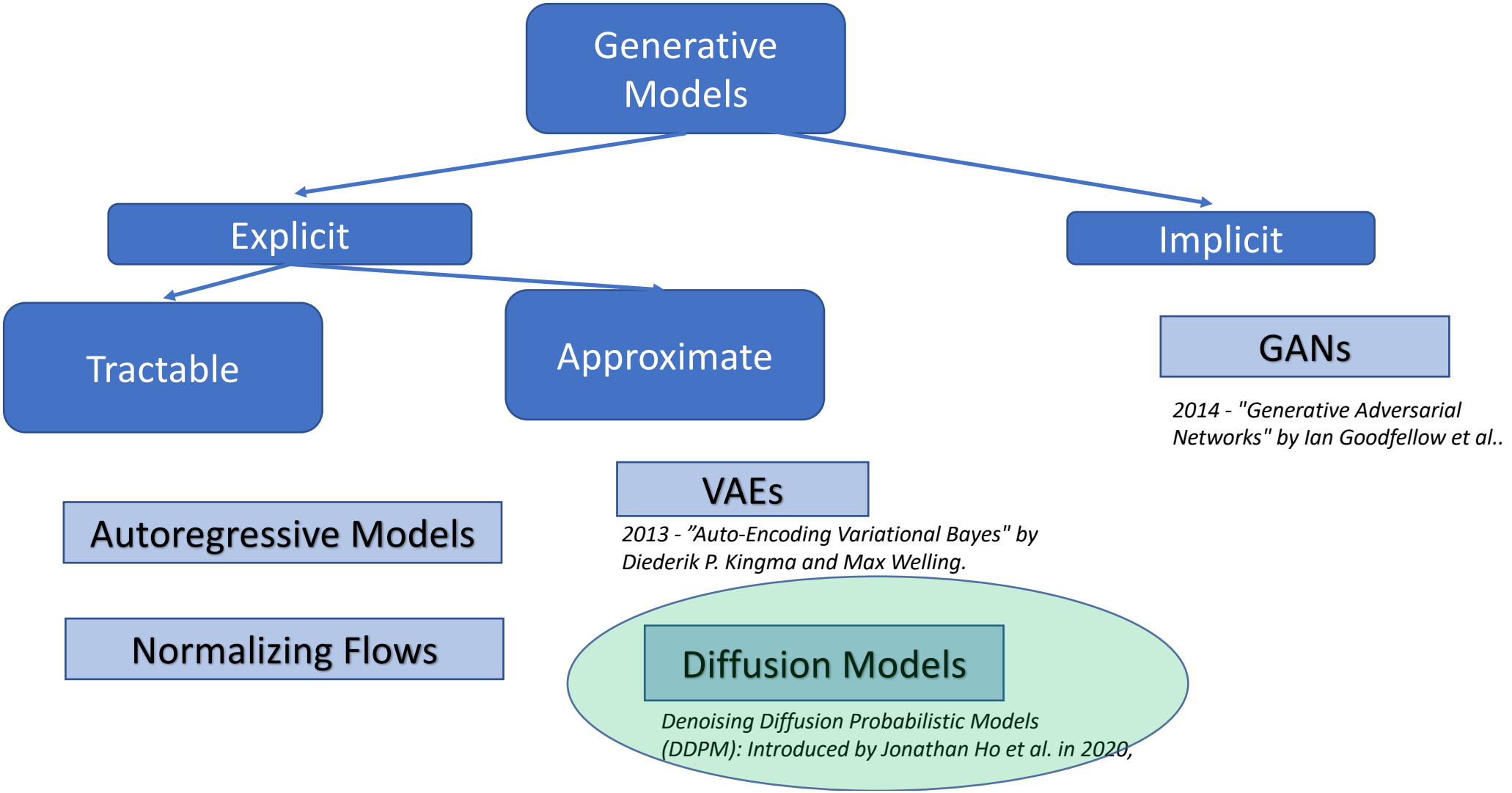
- Training Instability
  - **Mode Collapse**: The generator may produce a limited variety of outputs, failing to capture the full diversity of the data distribution.
  - **Non-Convergence**: Training may not converge due to the adversarial nature, with GG and DD failing to reach equilibrium.
- Training Complexity
  - **Vanishing Gradients**: The generator can receive weak gradients if the discriminator becomes too strong, hindering learning.

HSLU Hochschule Luzern

Code Discussion

Taxonomy Generative Models

HSLU Hochschule Luzern



HSLU Hochschule Luzern

Diffusion Models

Diffusion Models (DMs)

HSLU Hochschule Luzern

Key Components:

- **Forward Process:**
Add noise to data

- **Reverse Process:**
Learn to denoise the data from a noisy step to the previous one. Therough this reeverse diffusion process models can generate new data samples. By startin from appoint in the simple distribution and diffusiong it step by step to the deried more complex distribution.

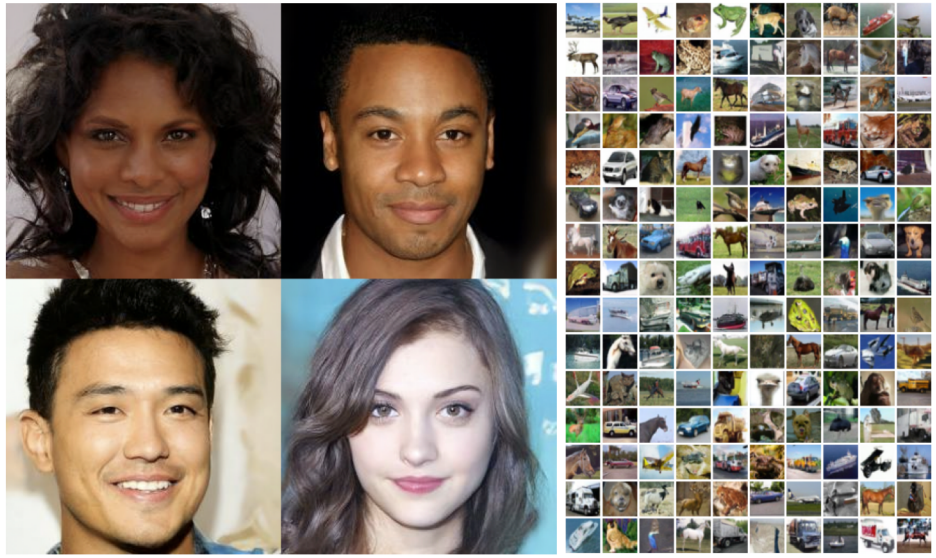


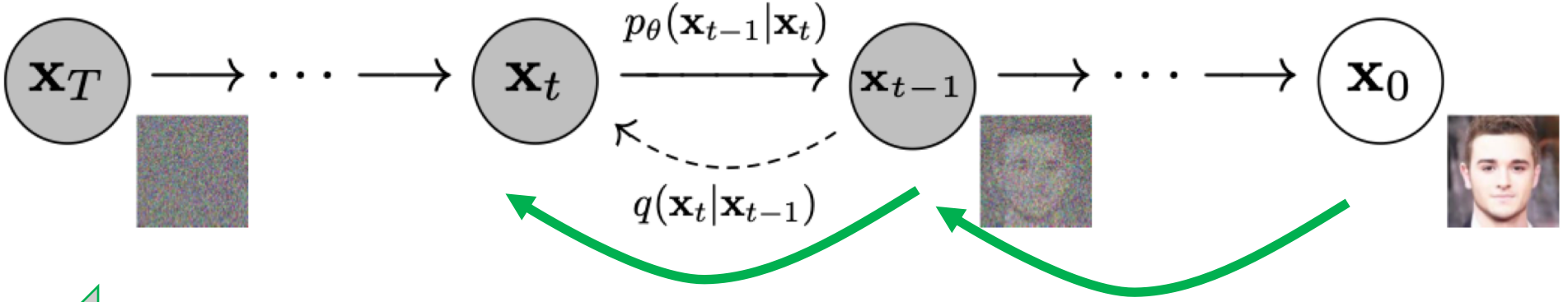
Figure 1: Generated samples on CelebA-HQ 256 × 256 (left) and unconditional CIFAR10 (right)

Diffusion Models (DMs) in a nutshell

HSLU Hochschule Luzern

Noise

Input Images



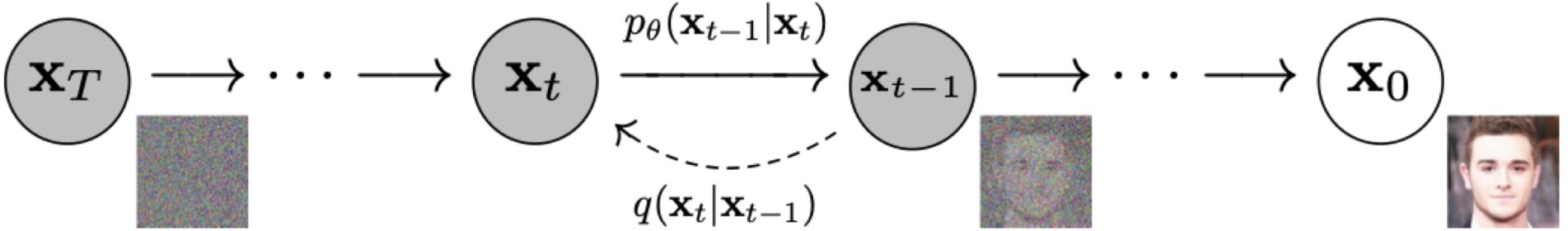
Forward Process: Add noise

Diffusion Models (DMs) in a nutshell

HSLU Hochschule Luzern

Noise

Input Images

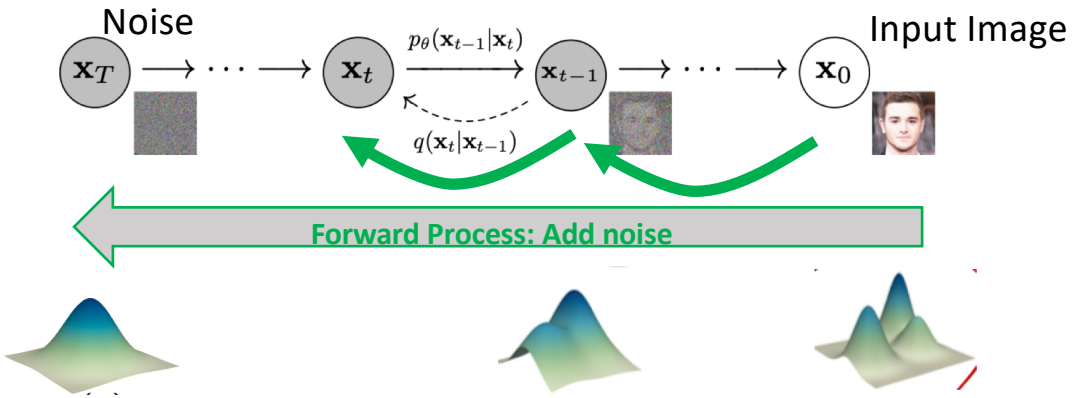


Reverse Process: Remove noise

Diffusion Models (DMs): Forward Process

HSLU Hochschule Luzern

- **Time Steps**: The process is defined over T discrete time steps t=1,2,...,T
- **Markov Chain**: Each step depends only on the previous state, forming a Markov chain.
- **Noise Addition**: At each step, a small amount of Gaussian noise is added to the data.



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

- **Direct Sampling from input data:** Due to Gaussian properties, we can samples directly form initial data.

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

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

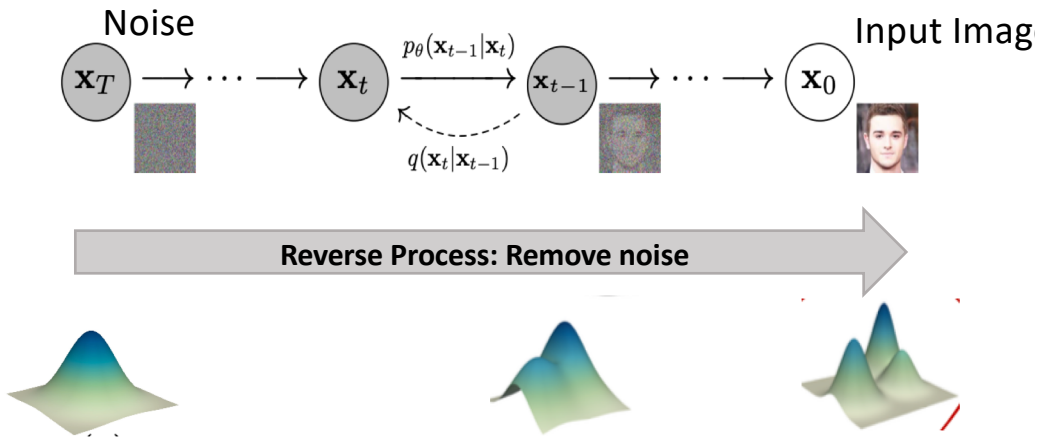
Diffusion Models (DMs): Reverse Process

- **Starting Point**: Begins with pure noise x~N(0,I)
- **Learned Approximation**: Since the true reverse distributions are intractable, we train a neural network to approximate them.

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$$

Reverse Process: Network learns the mean.

- **Iterative Denoising:** At each step, the model predicts the noise component and subtracts it.



Diffusion Models (DMs): Training & Sampling

HSLU Hochschule Luzern

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

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** $\mathbf{x}_0$

Fig. 4. The training and sampling algorithms in DDPM (Image source: Ho et al. 2020)

Diffusion Models (DMs)

HSLU Hochschule Luzern

Advantages:

- **High-Quality Outputs**: Capable of generating images with fine details and high fidelity.
- **Stable Training**: Avoid issues like mode collapse that are common in other generative models like GANs (Generative Adversarial Networks).

Limitations:

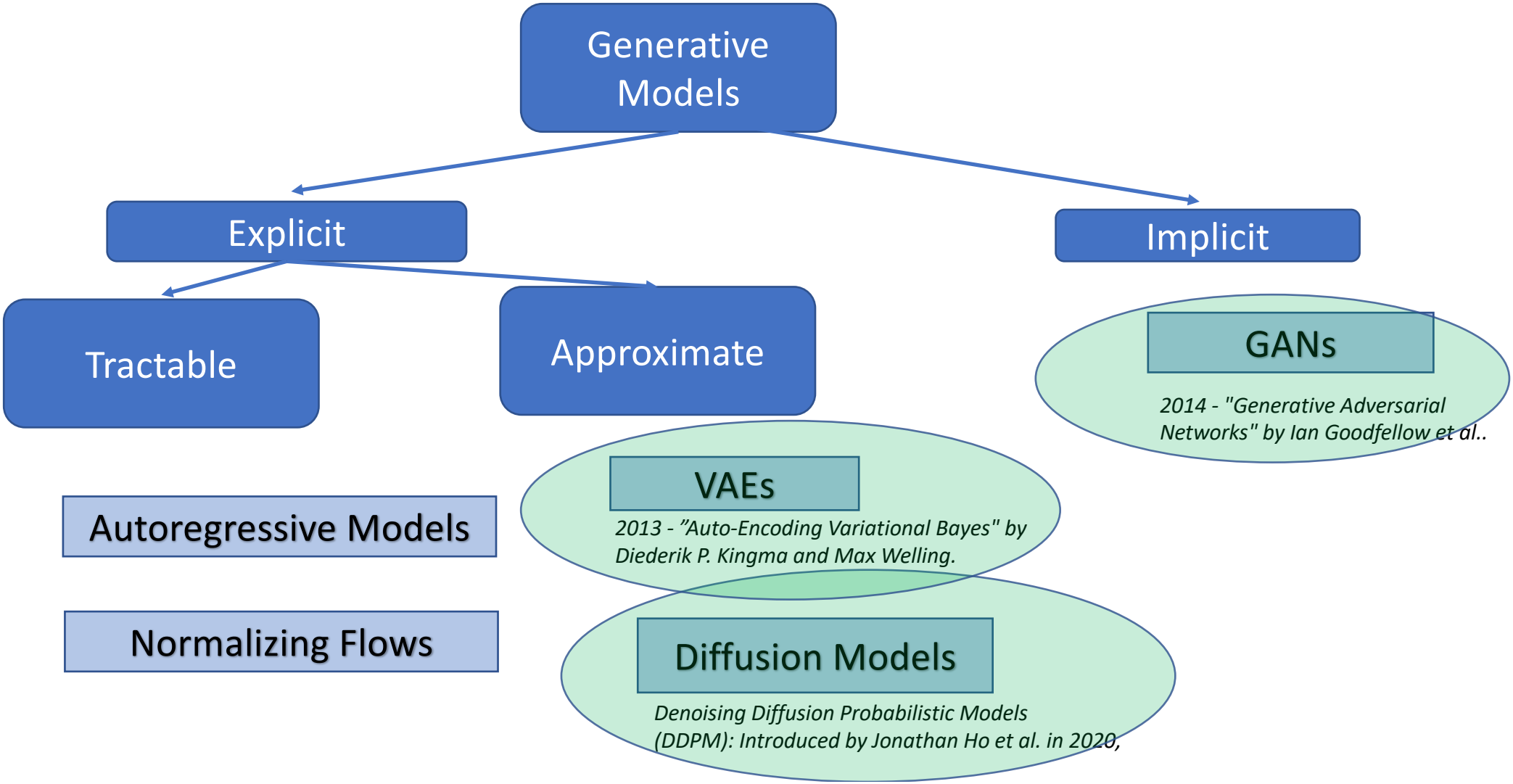
- Computational Complexity
  - **High Computational Cost**: Diffusion models require a large number of sequential steps to generate a single sample. This makes both training and inference computationally intensive.
  - **Slow Sampling Speed**: The iterative nature of the reverse diffusion process means that generating new data can be significantly slower compared to models like GANs or VAEs (Variational Autoencoders).
- Resource Requirements
  - **Memory Consumption**: The need to store intermediate states during the diffusion process can lead to high memory usage.
  - **Hardware Dependency**: Effective training often requires access to high-performance computing resources, such as GPUs or TPUs, which may not be accessible to everyone.

HSLU Hochschule Luzern

Code Discussion

Taxonomy Generative Models

HSLU Hochschule Luzern



Summary

HSLU Hochschule Luzern

- **Generative Adversarial Networks**
- **Diffusion Models**
- **Variational Autoencoders**

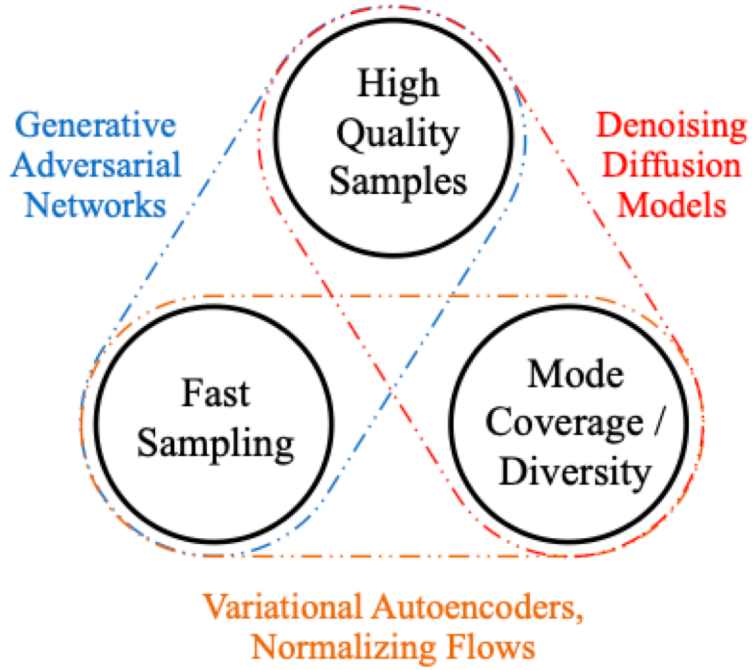


Figure 1: Generative learning trilemma.

HSLU Hochschule Luzern

Thank you