Deep Learning

A course about theory & practice

Generative Networks

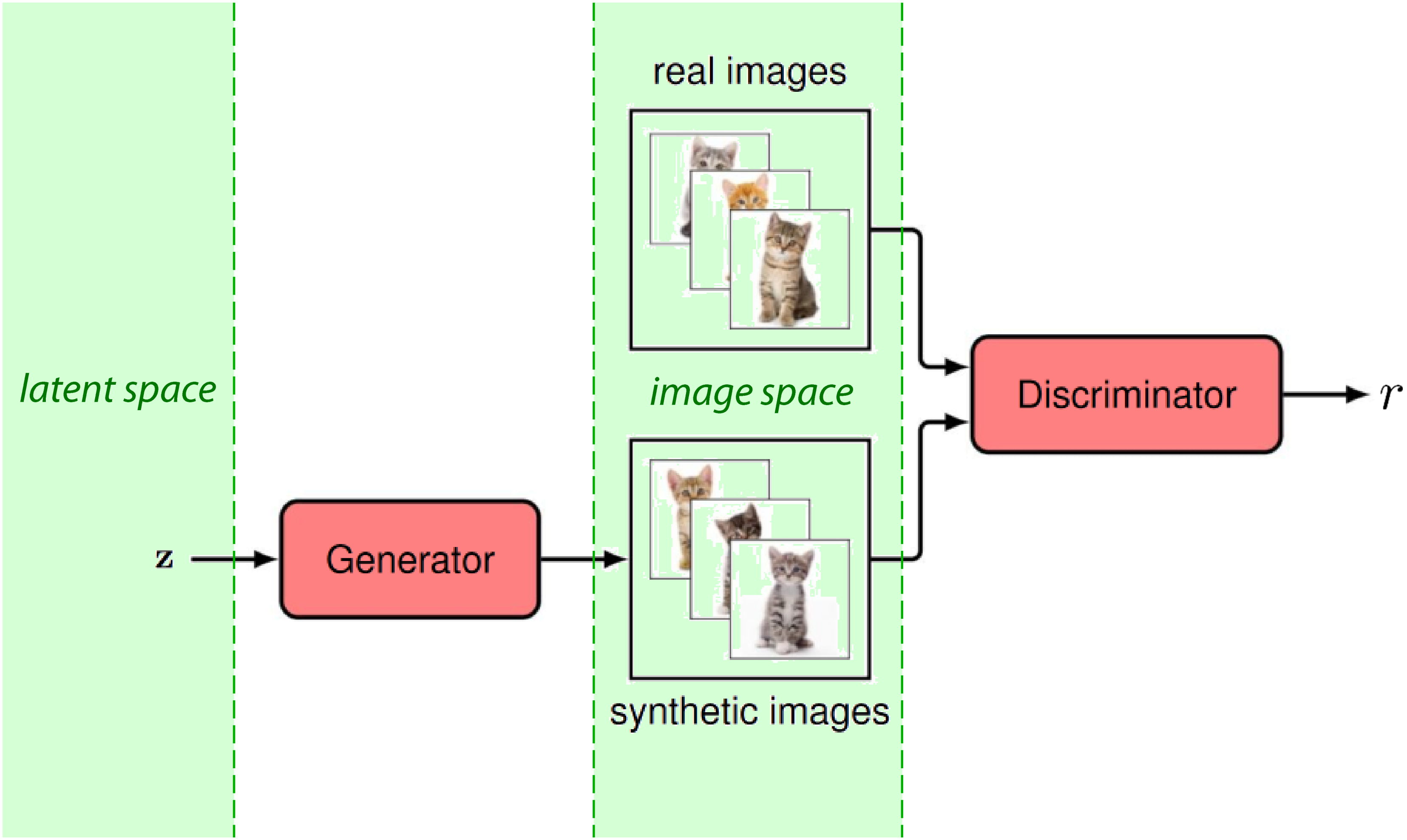
Marco Piastra

Generative Adversarial Networks

Basic idea

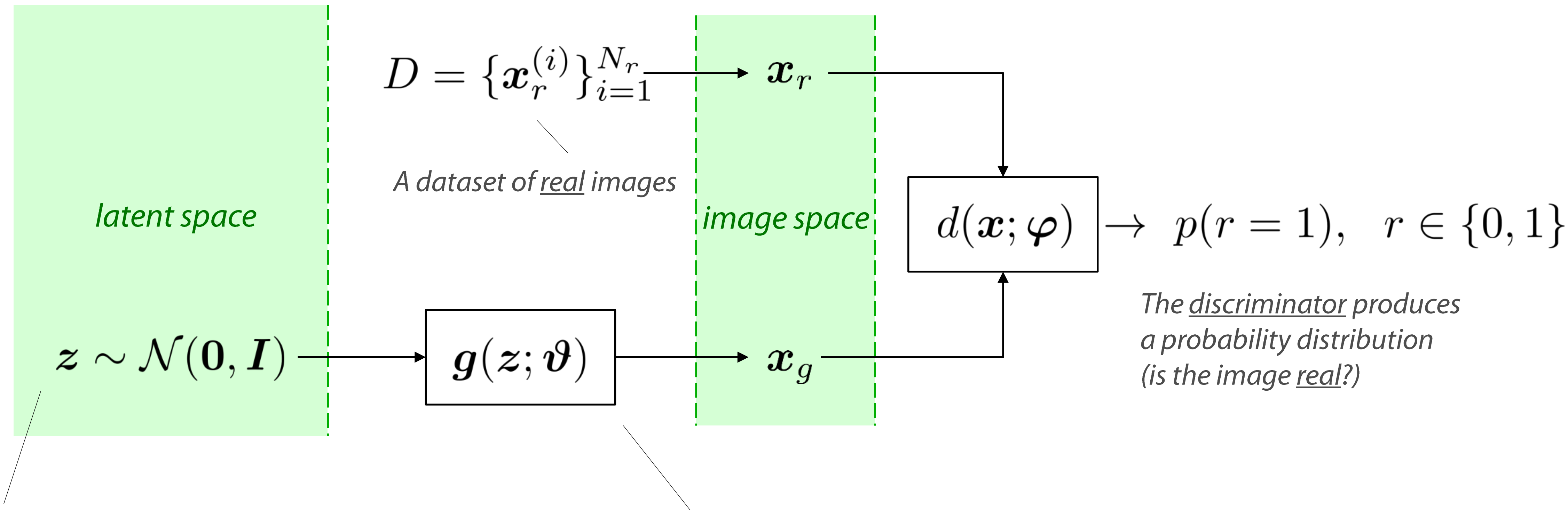
Objective:
creating a non-linear transformation from a latent space to a data space

Method:
training together a generator and a discriminator using a real dataset



[Image from https://www.bishopbook.com/]

Generative Adversarial Network (GAN)



latent space

$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$

$D = \{\boldsymbol{x}_r^{(i)}\}_{i=1}^{N_r}$

A dataset of real images

$\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\vartheta})$

image space

$\boldsymbol{x}_r$

$\boldsymbol{x}_g$

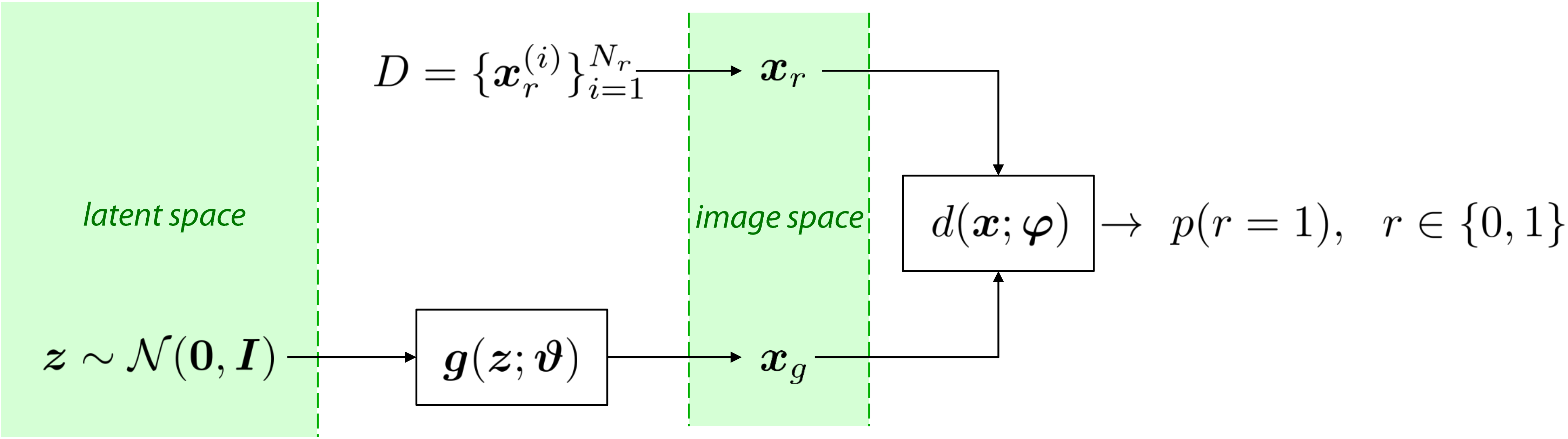
$d(\boldsymbol{x}; \boldsymbol{\varphi}) \to p(r = 1), \quad r \in \{0, 1\}$

The discriminator produces a probability distribution (is the image real?)

Samples in latent space are distributed as Gaussian

The transformation (generator) is deterministic

Generative Adversarial Network (GAN)



latent space
image space
$D = \{\boldsymbol{x}_r^{(i)}\}_{i=1}^{N_r}$
$\boldsymbol{x}_r$
$d(\boldsymbol{x};\boldsymbol{\varphi})$
$\to\ p(r=1),\quad r\in\{0,1\}$
$\boldsymbol{z}\sim\mathcal{N}(\boldsymbol{0},\boldsymbol{I})$
$\boldsymbol{g}(\boldsymbol{z};\boldsymbol{\vartheta})$
$\boldsymbol{x}_g$

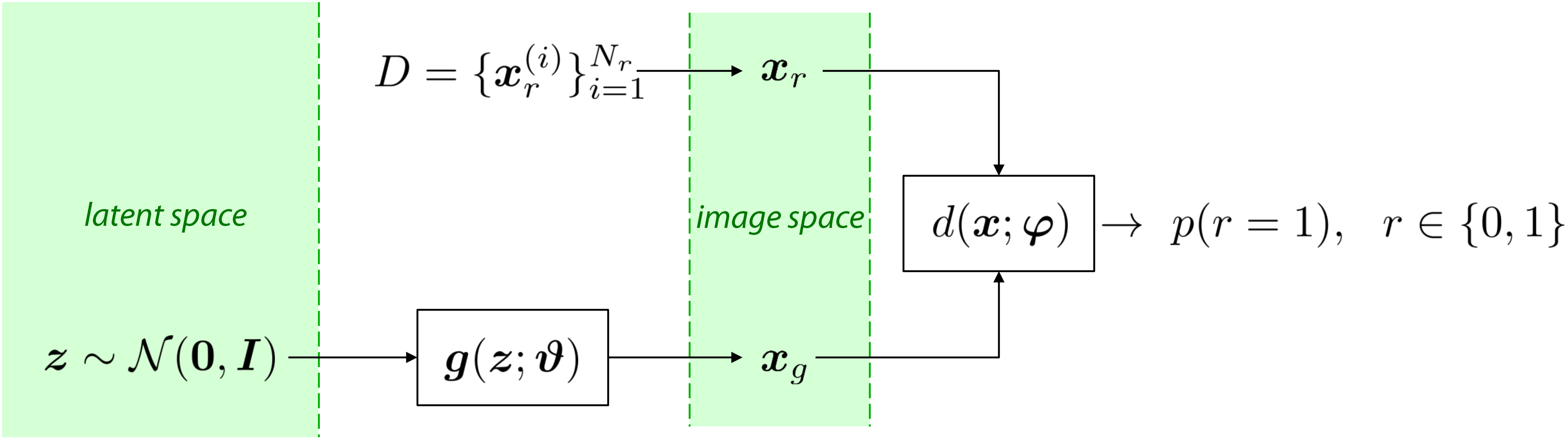
Loss function

$$L(\boldsymbol{\vartheta},\boldsymbol{\varphi}) := -\frac{1}{N_r}\sum_{i\in\mathcal{R}}\ln(d(\boldsymbol{x}_r^{(i)};\boldsymbol{\varphi})) - \frac{1}{N_g}\sum_{j\in\mathcal{G}}\ln(1-d(\boldsymbol{g}(\boldsymbol{z}^{(j)};\boldsymbol{\vartheta});\boldsymbol{\varphi}))$$

Cross-entropy
(d should recognize real images)

Cross-entropy
(d should recognize 'false' images)

Generative Adversarial Network (GAN)



$D = \{\boldsymbol{x}_r^{(i)}\}_{i=1}^{N_r}$

$\boldsymbol{x}_r$

latent space

image space

$d(\boldsymbol{x};\boldsymbol{\varphi}) \to \; p(r=1), \quad r \in \{0,1\}$

$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0},\boldsymbol{I})$

$\boldsymbol{g}(\boldsymbol{z};\boldsymbol{\vartheta})$

$\boldsymbol{x}_g$

Loss function

$$L(\boldsymbol{\vartheta},\boldsymbol{\varphi}) := -\frac{1}{N_r}\sum_{i\in\mathcal{R}} \ln(d(\boldsymbol{x}_r^{(i)};\boldsymbol{\varphi})) - \frac{1}{N_g}\sum_{j\in\mathcal{G}} \ln(1 - d(\boldsymbol{g}(\boldsymbol{z}^{(j)};\boldsymbol{\vartheta});\boldsymbol{\varphi}))$$

Gradients

$$\Delta\boldsymbol{\varphi} = -\,\eta\frac{\partial}{\partial\boldsymbol{\varphi}}L(\boldsymbol{\vartheta},\boldsymbol{\varphi}) \qquad \Delta\boldsymbol{\vartheta} = +\,\eta\frac{\partial}{\partial\boldsymbol{\vartheta}}L(\boldsymbol{\vartheta},\boldsymbol{\varphi})$$

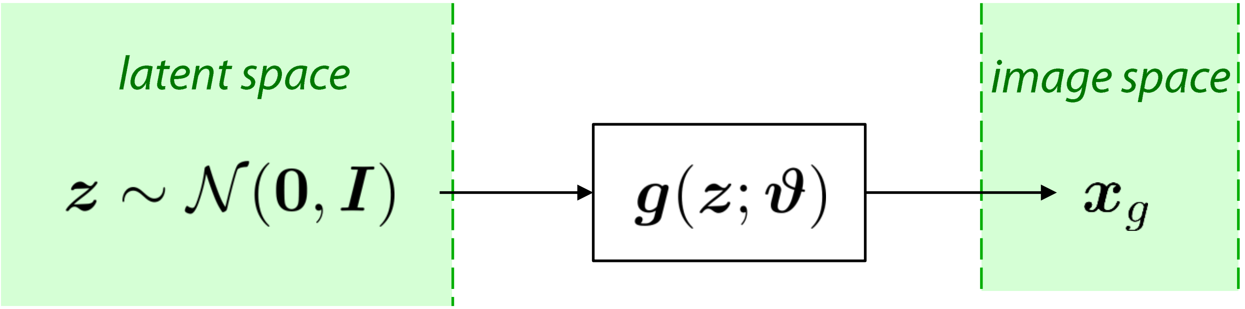
Make the discriminator smarter

Make the generator smarter: the discriminator should be fooled

Generative Adversarial Network (GAN)

After training

The generator can be used to transform random samples in latent space into realistic data items



latent space

$\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

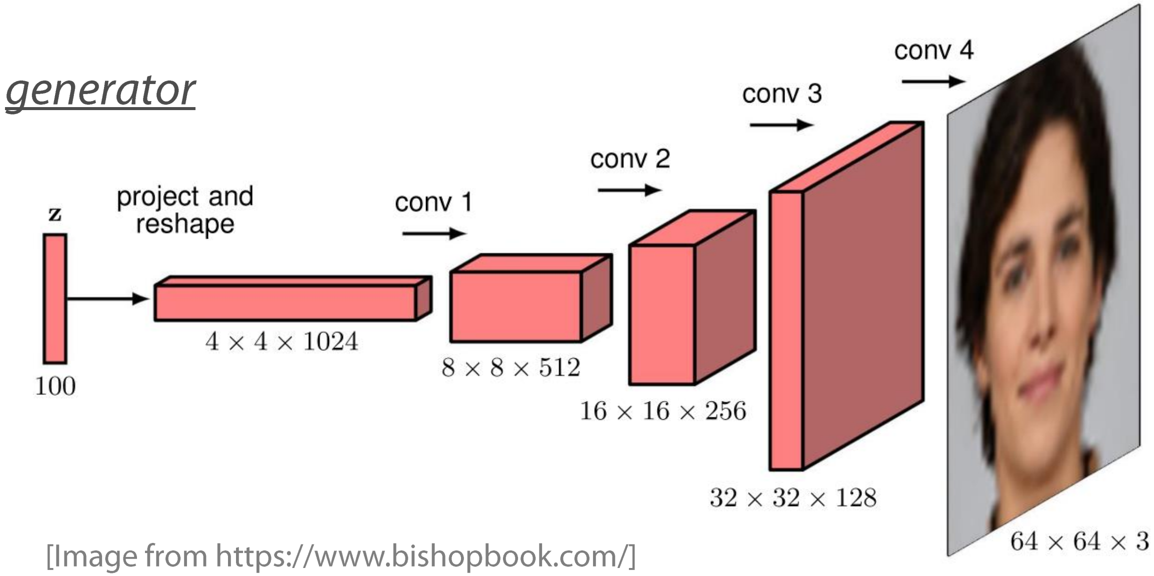
$\boldsymbol{g}(\boldsymbol{z}; \boldsymbol{\vartheta})$

image space

$\boldsymbol{x}_g$

ImageGAN

Typically, a (de)convolutional network is used for the *generator*

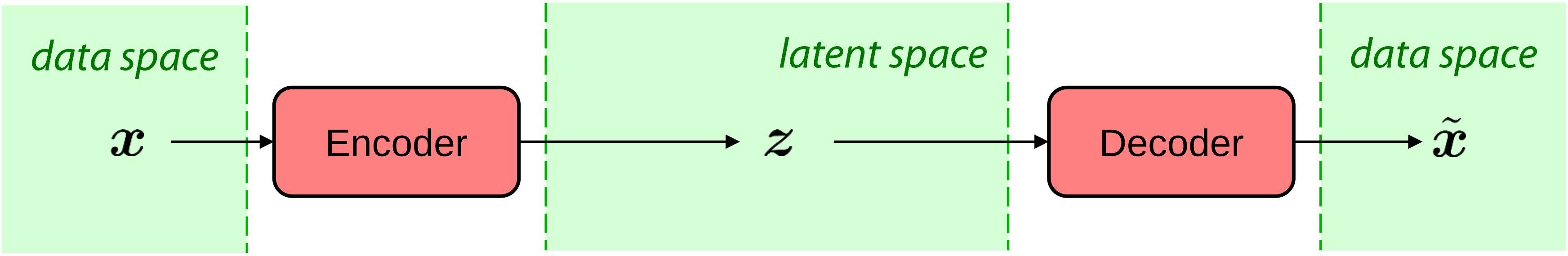


[Image from https://www.bishopbook.com/]

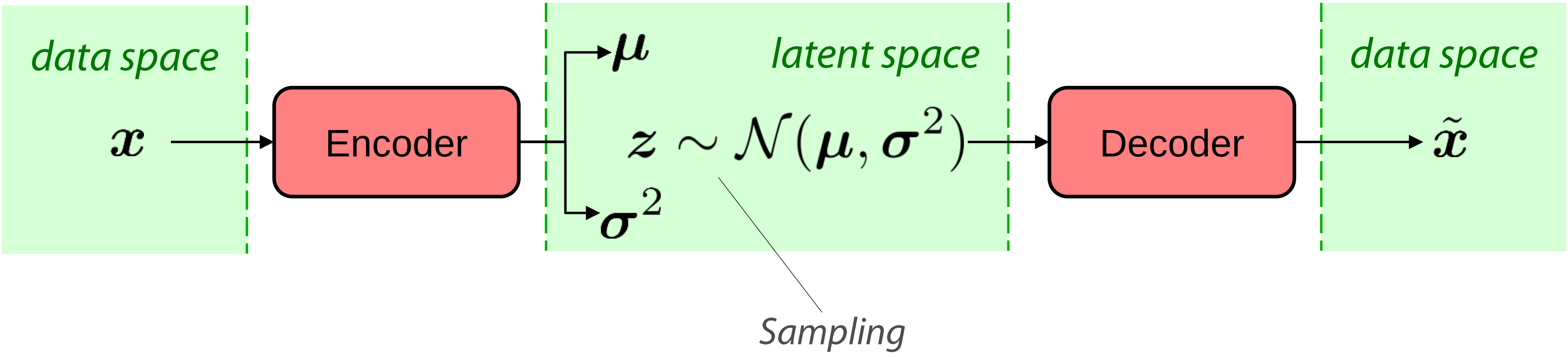
Variational Auto-Encoders

Basic idea

Auto-Encoder:
from data space
into latent space
then back

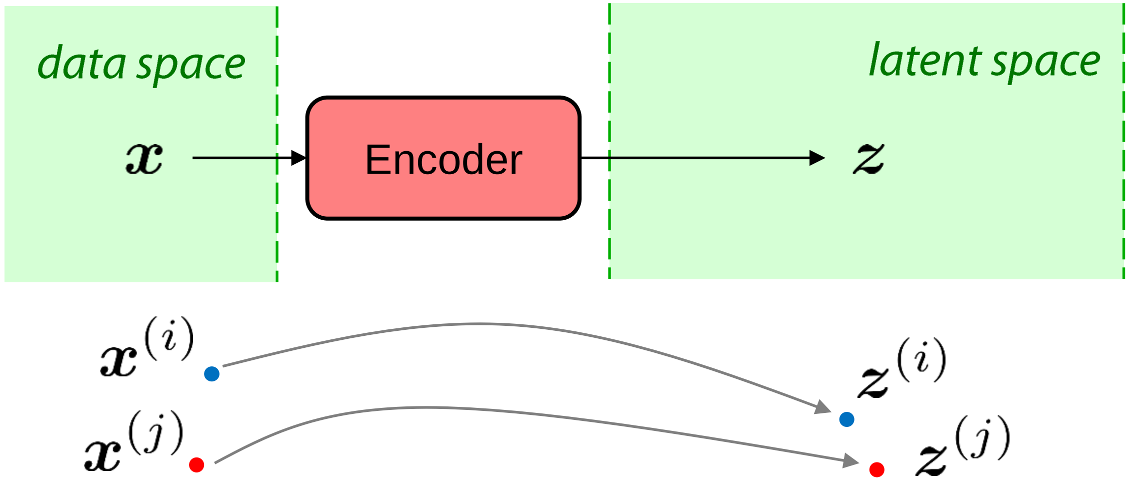


Variational
Auto-Encoder :
use a Gaussian spread
function to <u>organize</u>
the latent space

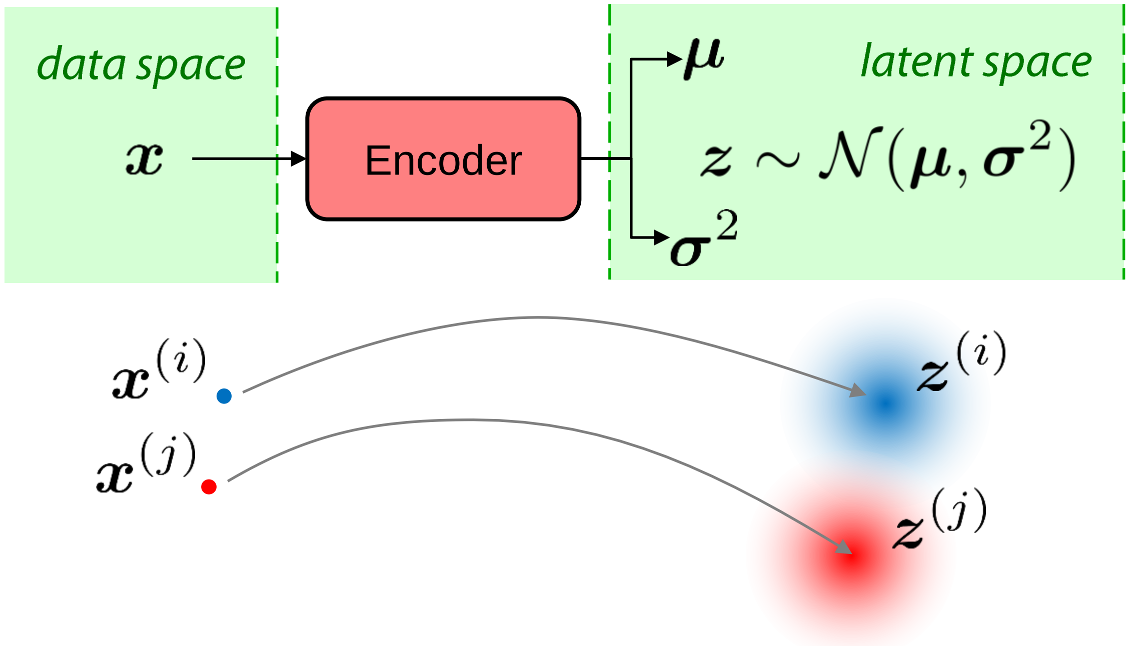


Basic idea

Auto-Encoder:
the correspondence between data space and latent space is one to one

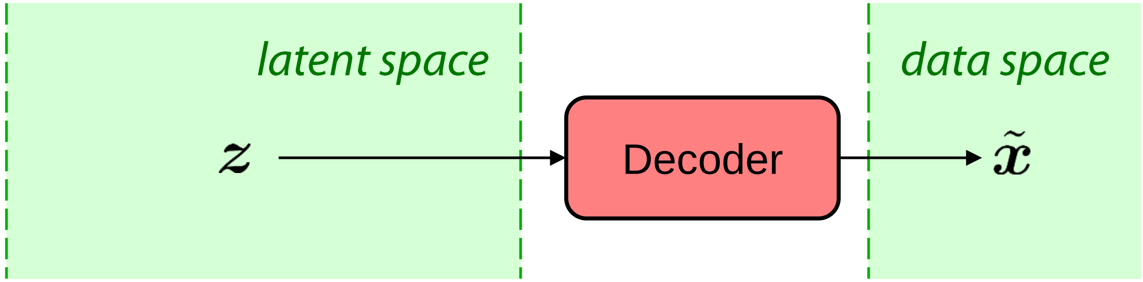


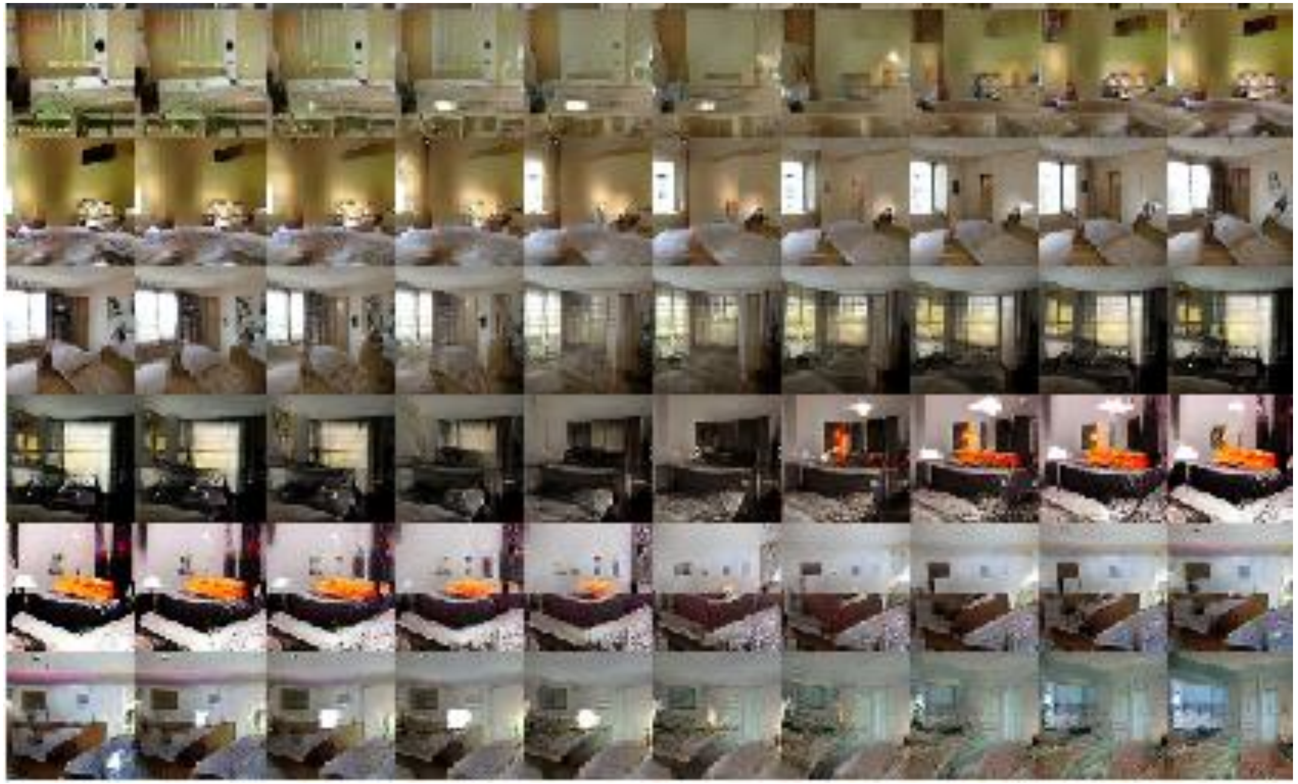
Variational Auto-Encoder :
the correspondence is one to many

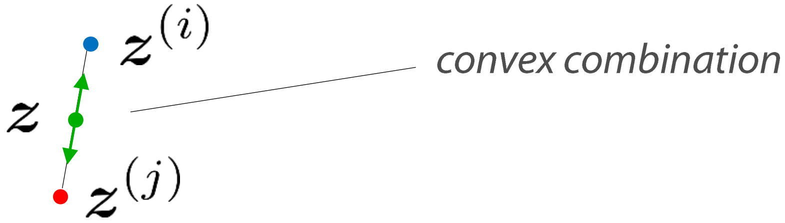


Smooth generation

Variational Auto-Encoder :
after training, any convex combination of two points in latent space will generate a data item that changes smoothly from one extreme to the other





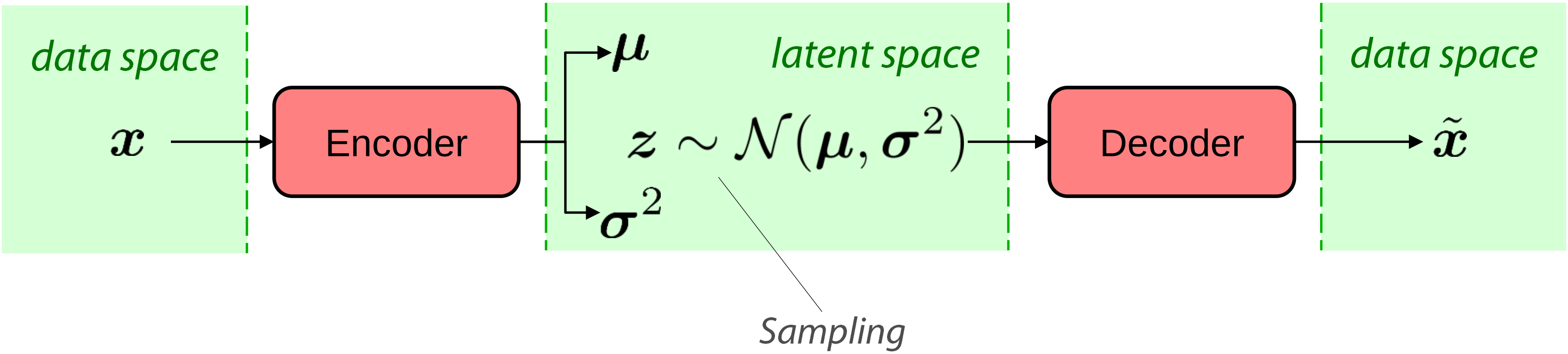


$$\boldsymbol{z} = \alpha_i \boldsymbol{z}^{(i)} + \alpha_j \boldsymbol{z}^{(j)}$$

$$\alpha_i, \alpha_j \geq 0, \quad \alpha_i + \alpha_j = 1$$

A problem: which loss function?

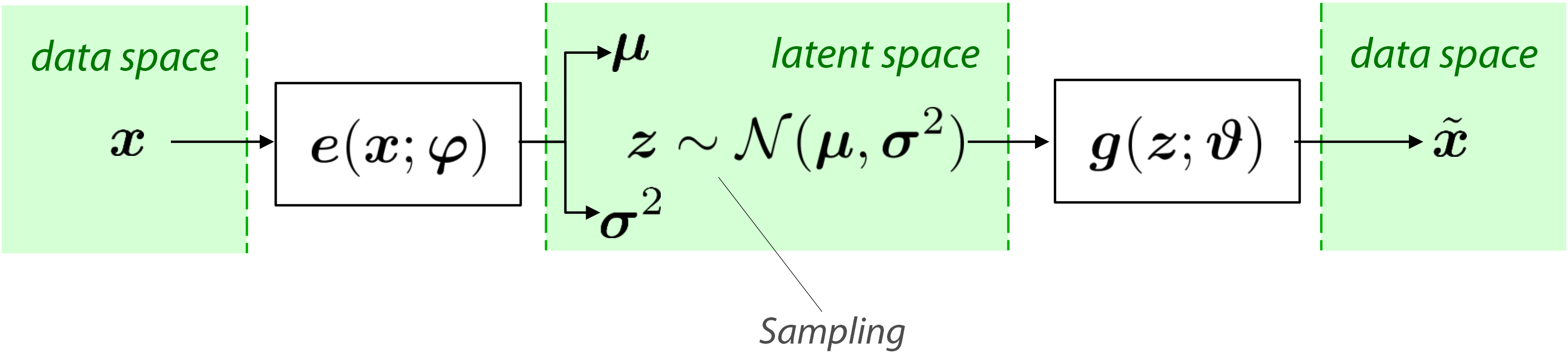
Variational Auto-Encoder :
use a Gaussian spread function to <u>organize</u> the latent space



This is what we want to train from a real dataset $D = \{\boldsymbol{x}_r^{(i)}\}_{i=1}^{N_r}$

A problem: which loss function?

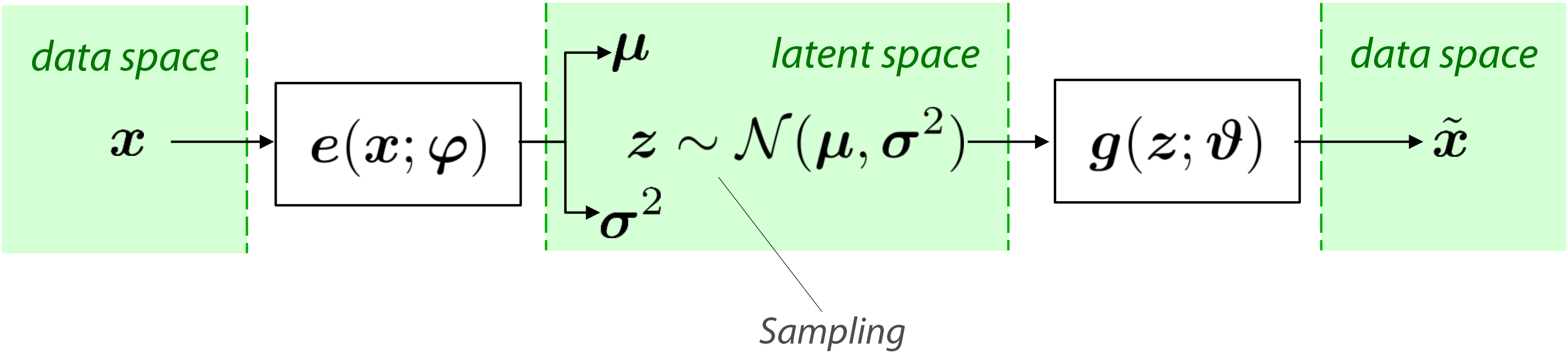
Variational Auto-Encoder :
use a Gaussian spread function to <u>organize</u> the latent space


data space
$\boldsymbol{x}$
$\boldsymbol{e}(\boldsymbol{x};\boldsymbol{\varphi})$
$\boldsymbol{\mu}$
$\boldsymbol{\sigma}^2$
latent space
$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$
Sampling
$\boldsymbol{g}(\boldsymbol{z};\boldsymbol{\vartheta})$
data space
$\tilde{\boldsymbol{x}}$

This is what we want to train from a real dataset $D = \{\boldsymbol{x}_r^{(i)}\}_{i=1}^{N_r}$

A problem: which loss function?

Variational Auto-Encoder :
use a Gaussian spread function to <u>organize</u> the latent space



This is what we want to train from a real dataset $D = \{x_r^{(i)}\}_{i=1}^{N_r}$

This is similar to the loss of a standard autoencoder

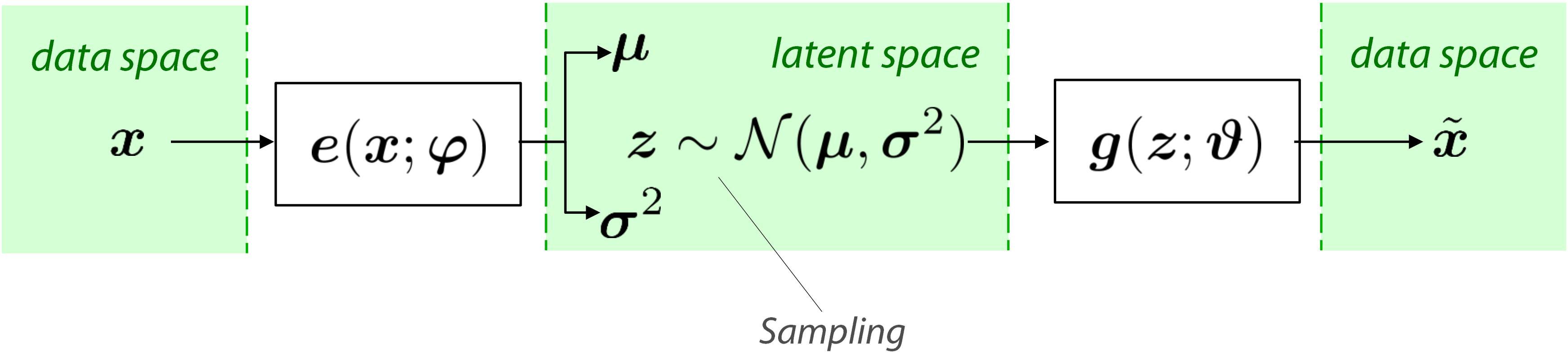
$$L(x;\vartheta,\varphi) := \mathrm{KL}(q(z \mid x,\varphi) \parallel p(z)) \; + \; \frac{1}{2}\frac{\|x-\tilde{x}\|^2}{c}$$

This is an hyperparameter (see later)

Kullback-Leibler divergence

A problem: which loss function?

Variational Auto-Encoder : *use a Gaussian spread function to <u>organize</u> the latent space*


data space
$\boldsymbol{x}$
$\boldsymbol{e}(\boldsymbol{x};\boldsymbol{\varphi})$
$\boldsymbol{\mu}$
latent space
$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu},\boldsymbol{\sigma}^2)$
$\boldsymbol{\sigma}^2$
$\boldsymbol{g}(\boldsymbol{z};\boldsymbol{\vartheta})$
data space
$\tilde{\boldsymbol{x}}$
Sampling

This is what we want to train from a real dataset $D = \{\boldsymbol{x}_r^{(i)}\}_{i=1}^{N_r}$

$$L(\boldsymbol{x};\boldsymbol{\vartheta},\boldsymbol{\varphi}) := \mathrm{KL}(q(\boldsymbol{z} \mid \boldsymbol{x},\boldsymbol{\varphi}) \parallel p(\boldsymbol{z})) \; + \; \frac{1}{2}\frac{\|\boldsymbol{x}-\tilde{\boldsymbol{x}}\|^2}{c}$$

Design choices

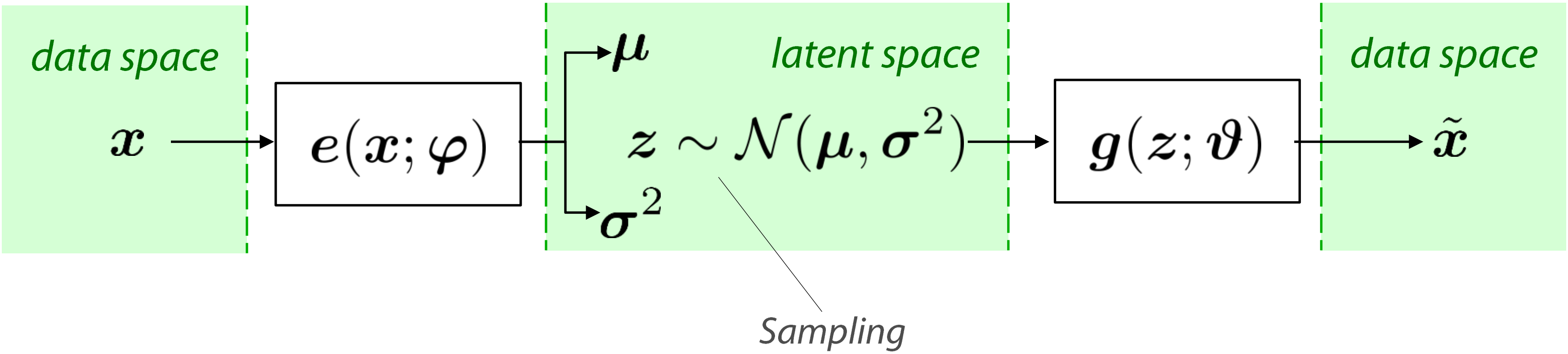
$$q(\boldsymbol{z} \mid \boldsymbol{x},\boldsymbol{\varphi}) := \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x};\boldsymbol{\varphi}),\boldsymbol{\sigma}^2(\boldsymbol{x};\boldsymbol{\varphi})\boldsymbol{I})$$

$$p(\boldsymbol{z}) := \mathcal{N}(\boldsymbol{0},\boldsymbol{I})$$

Normalization constraint: a soft limit against overspreading latent values

A problem: which loss function?

Variational Auto-Encoder : *use a Gaussian spread function to* ***<u>organize</u>*** *the latent space*



data space

$\boldsymbol{x}$

$\boldsymbol{e}(\boldsymbol{x};\boldsymbol{\varphi})$

$\boldsymbol{\mu}$

$\boldsymbol{\sigma}^2$

latent space

$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$

Sampling

$\boldsymbol{g}(\boldsymbol{z};\boldsymbol{\vartheta})$

data space

$\tilde{\boldsymbol{x}}$

In general

$$\mathrm{KL}(q(\boldsymbol{z}) \parallel p(\boldsymbol{z})) := \int q(\boldsymbol{z}) \ln \frac{q(\boldsymbol{z})}{p(\boldsymbol{z})}\, \mathrm{d}\boldsymbol{z}$$

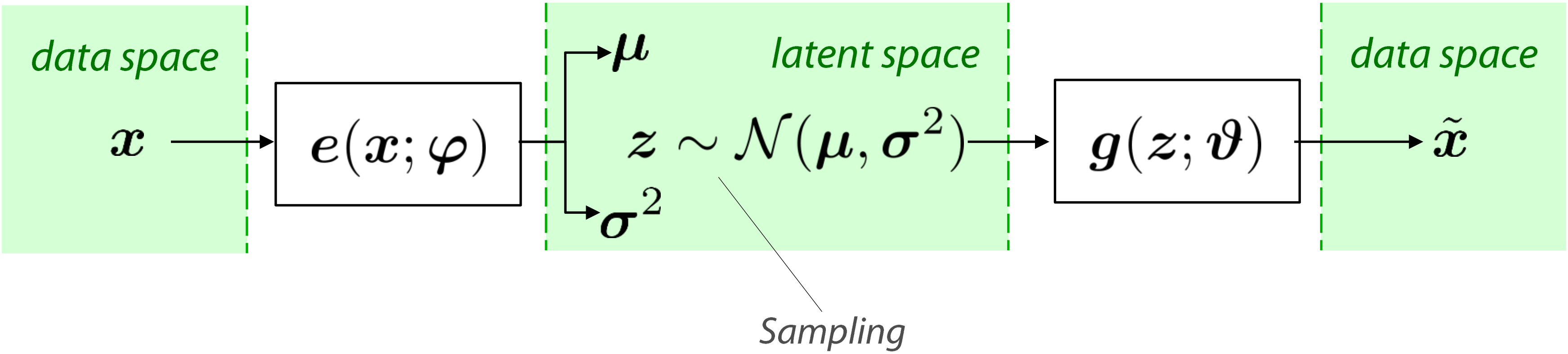
Kullback-Leibler divergence; always positive, zero when the two distributions are identical

Under the conditions adopted

$$\mathrm{KL}(q(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{\varphi}) \parallel p(\boldsymbol{z})) = -\frac{1}{2} \sum_{j=1}^{\dim(\boldsymbol{z})} \left(1 + \ln \sigma_j^2(\boldsymbol{x};\boldsymbol{\varphi}) - \mu_j^2(\boldsymbol{x};\boldsymbol{\varphi}) - \sigma_j^2(\boldsymbol{x};\boldsymbol{\varphi})\right)$$

A problem: which loss function?

Variational Auto-Encoder :
use a Gaussian spread function to <u>organize</u> the latent space



data space
$\boldsymbol{x}$
$\boldsymbol{e}(\boldsymbol{x};\boldsymbol{\varphi})$
$\boldsymbol{\mu}$
latent space
$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$
$\boldsymbol{\sigma}^2$
Sampling
$\boldsymbol{g}(\boldsymbol{z};\boldsymbol{\vartheta})$
data space
$\tilde{\boldsymbol{x}}$

With a bit more of mathematics (omitted :)) it can be shown that the second term in the loss function

$$\frac{1}{2}\frac{\|\boldsymbol{x}-\tilde{\boldsymbol{x}}\|^2}{c}$$

relates to an assumption of:

$$p(\tilde{\boldsymbol{x}}) := \mathcal{N}(\boldsymbol{x}, c\boldsymbol{I}),\ \ c > 0$$

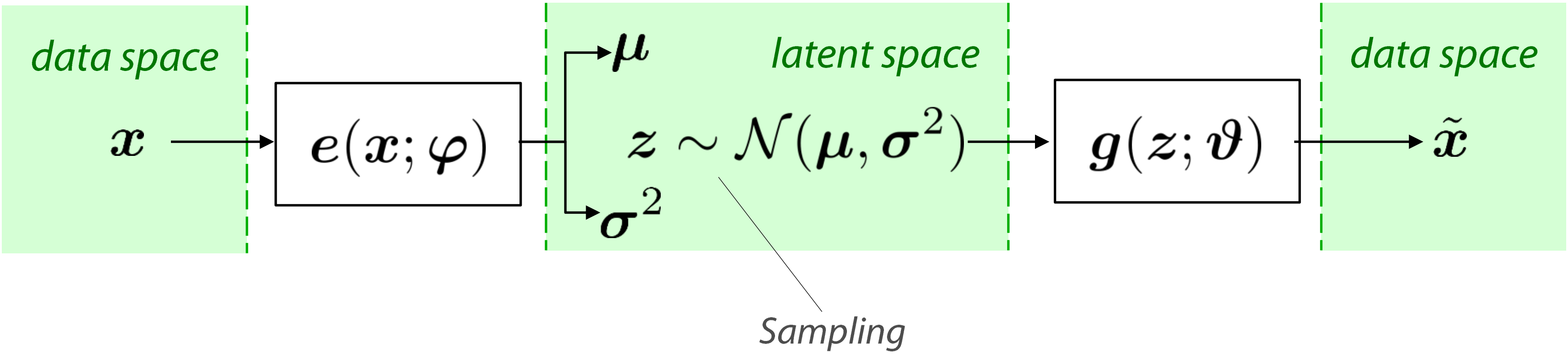
hyperparameter

Design choice

(hyper) spherical normal

Reparametrization Trick

Variational Auto-Encoder :
use a Gaussian spread function to organize the latent space


data space
$\boldsymbol{x}$
$\boldsymbol{e}(\boldsymbol{x};\boldsymbol{\varphi})$
$\boldsymbol{\mu}$
$\boldsymbol{\sigma}^2$
latent space
$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu},\boldsymbol{\sigma}^2)$
Sampling
$\boldsymbol{g}(\boldsymbol{z};\boldsymbol{\vartheta})$
data space
$\tilde{\boldsymbol{x}}$

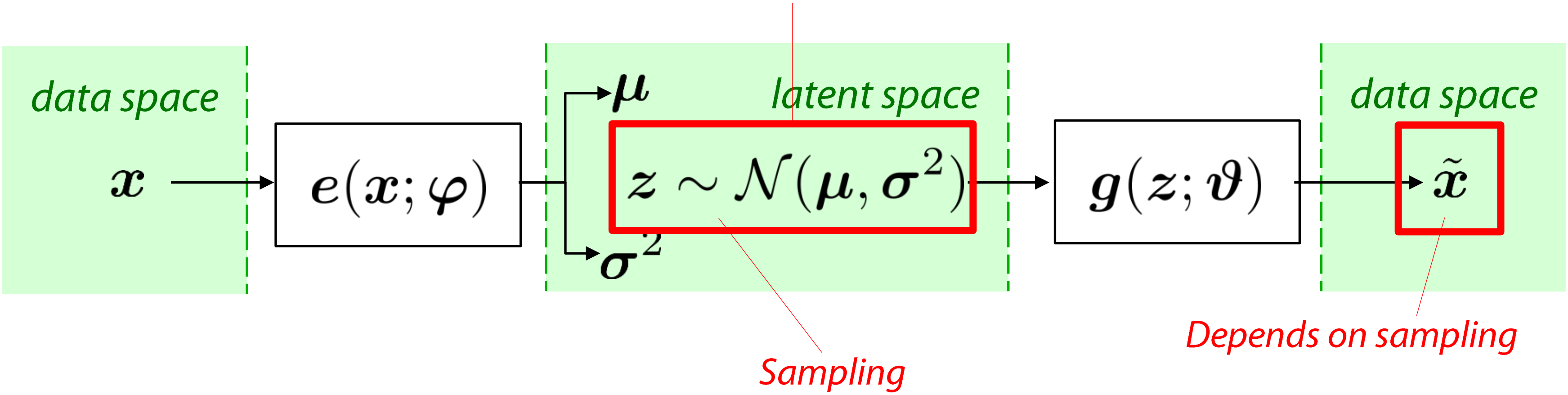
$$L(\boldsymbol{x};\boldsymbol{\vartheta},\boldsymbol{\varphi}) := -\frac{1}{2}\sum_{j=1}^{\dim(\boldsymbol{z})}\big(1+\ln\sigma_j^2(\boldsymbol{x};\boldsymbol{\varphi})-\mu_j^2(\boldsymbol{x};\boldsymbol{\varphi})-\sigma_j^2(\boldsymbol{x};\boldsymbol{\varphi})\big) \; + \; \frac{1}{2}\frac{\|\boldsymbol{x}-\tilde{\boldsymbol{x}}\|^2}{c}$$

$$\Delta\boldsymbol{\varphi} = -\,\eta\frac{\partial}{\partial\boldsymbol{\varphi}}L(\boldsymbol{x};\boldsymbol{\vartheta},\boldsymbol{\varphi}) \qquad\qquad \Delta\boldsymbol{\vartheta} = -\,\eta\frac{\partial}{\partial\boldsymbol{\vartheta}}L(\boldsymbol{x};\boldsymbol{\vartheta},\boldsymbol{\varphi})$$

Reparametrization Trick

$\tilde{\boldsymbol{x}}$ depends on both $\boldsymbol{\vartheta}$ and $\boldsymbol{\varphi}$ via $\boldsymbol{z}$
yet, when $\boldsymbol{z}$ is sampled, the derivative in $\boldsymbol{\varphi}$ is blocked

Variational Auto-Encoder :
use a Gaussian spread function to organize the latent space


data space
$\boldsymbol{x}$
$\boldsymbol{e}(\boldsymbol{x};\boldsymbol{\varphi})$
$\boldsymbol{\mu}$
latent space
$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$
$\boldsymbol{\sigma}^2$
$\boldsymbol{g}(\boldsymbol{z};\boldsymbol{\vartheta})$
data space
$\tilde{\boldsymbol{x}}$
Sampling
Depends on sampling

$$L(\boldsymbol{x};\boldsymbol{\vartheta},\boldsymbol{\varphi}) := -\frac{1}{2}\sum_{j=1}^{\dim(\boldsymbol{z})}\left(1+\ln\sigma_j^2(\boldsymbol{x};\boldsymbol{\varphi}) - \mu_j^2(\boldsymbol{x};\boldsymbol{\varphi}) - \sigma_j^2(\boldsymbol{x};\boldsymbol{\varphi})\right) + \frac{1}{2}\frac{\|\boldsymbol{x}-\tilde{\boldsymbol{x}}\|^2}{c}$$

The trick is assuming:

$$\boldsymbol{z} = \boldsymbol{\mu}(\boldsymbol{x};\boldsymbol{\varphi}) + \varepsilon\,\boldsymbol{\sigma}^2(\boldsymbol{x};\boldsymbol{\varphi})$$

where:

$$\varepsilon \sim \mathcal{N}(0,1)$$

is a parameter, therefore it is constant to the derivative.
In plain words, during training and per each data item $\boldsymbol{x}^{(i)}$ *the system draws one random value* ε *and computes the derivatives*

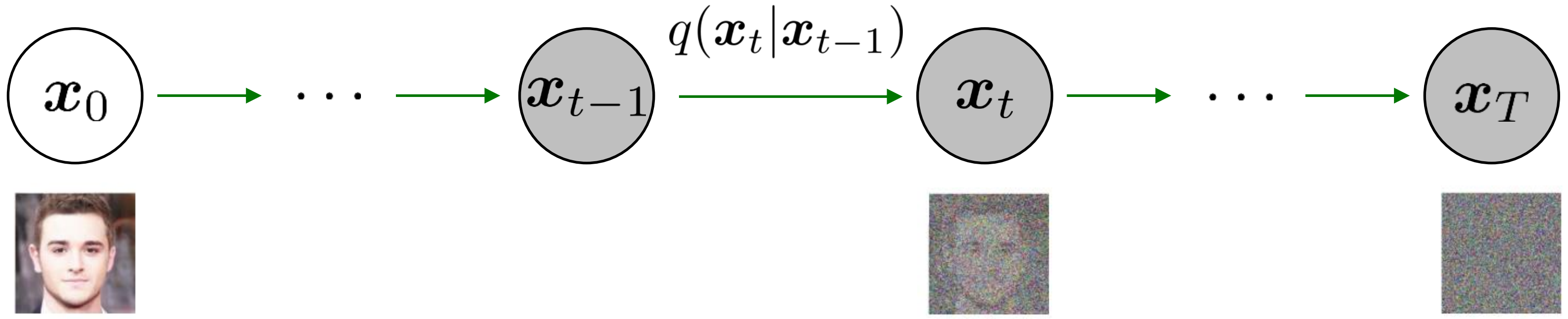
Links

https://johfischer.com/2022/09/18/denoising-score-matching/
https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73
https://en.wikipedia.org/wiki/Variational_autoencoder
https://mbernste.github.io/posts/vae/

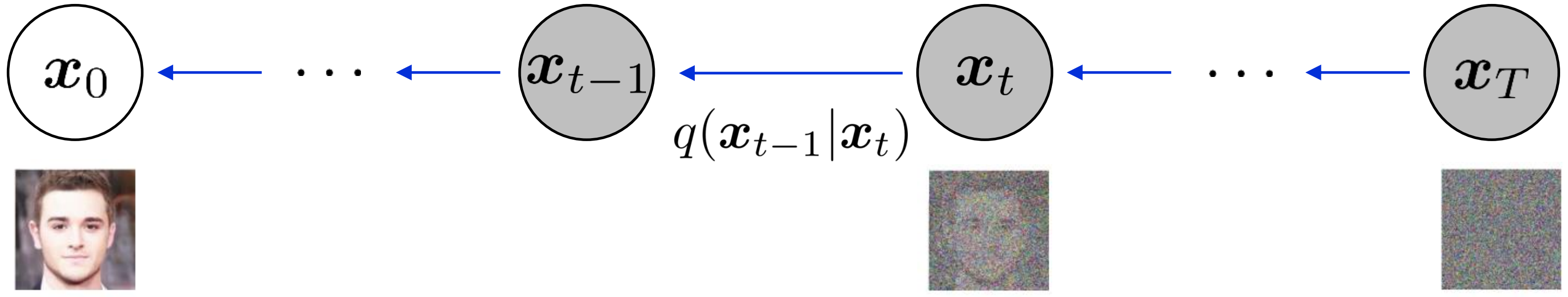
Diffusion Models

Basic idea

Forward Diffusion
Assume that images
are corrupted by
Gaussian noise
with known parameters



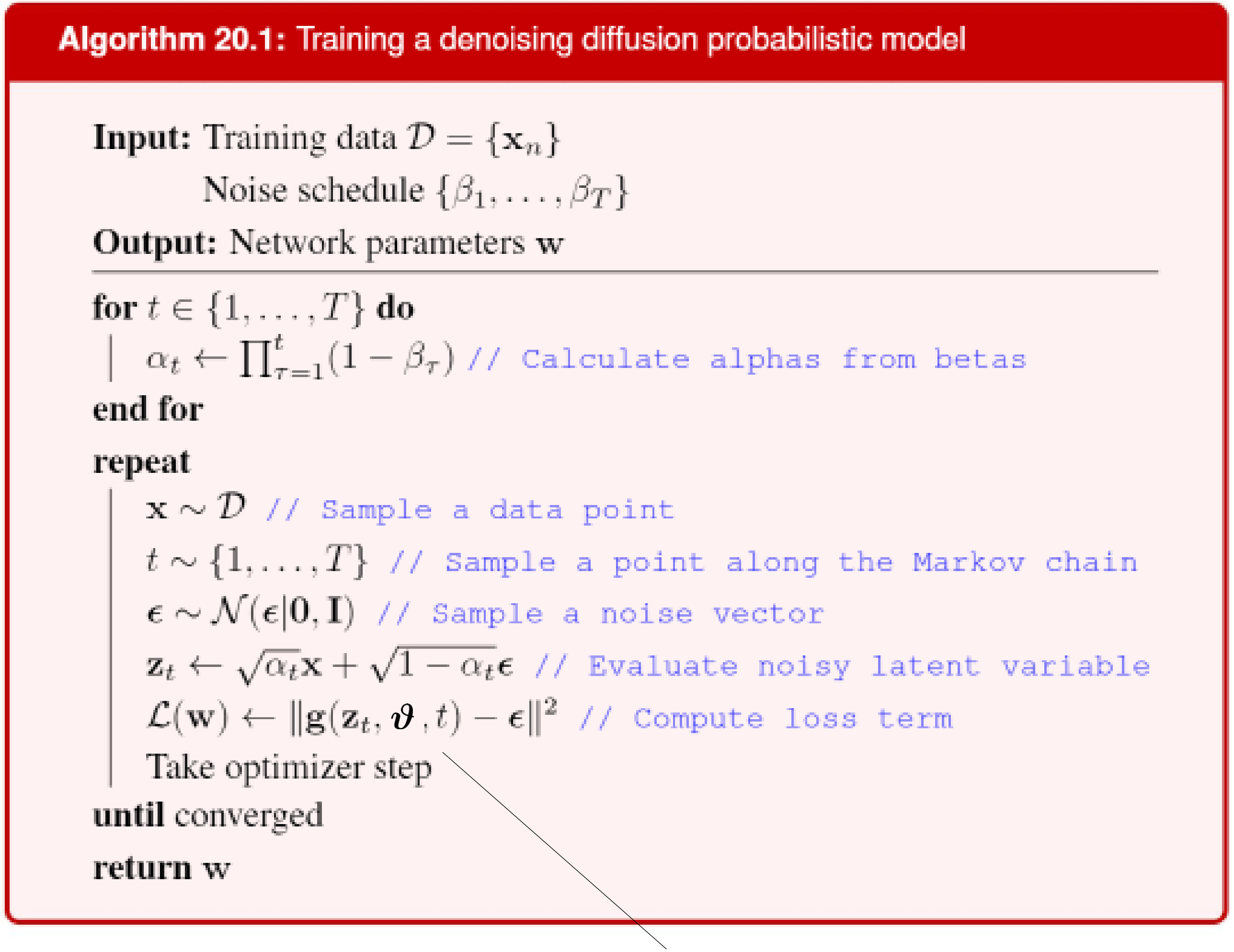
The idea behind
Denoising Diffusion Probabilistic Models
is learning how to reverse the process



Starting from the end: training algorithm

Forward Diffusion
Assume that images
are corrupted by
Gaussian noise
with known parameters

The idea behind
Denoising Diffusion Probabilistic Models
is learning how to reverse the process



Algorithm 20.1: Training a denoising diffusion probabilistic model

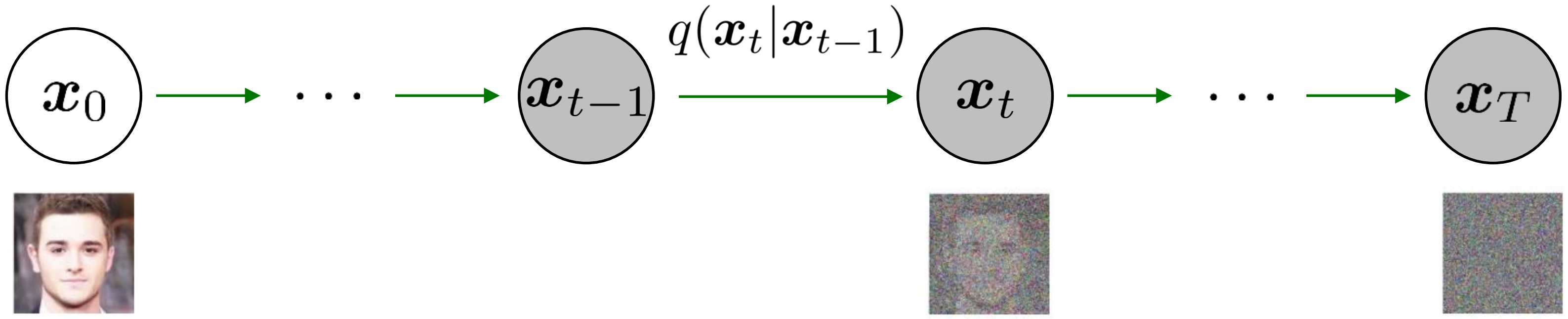
```
Input: Training data $\mathcal{D} = \{\mathbf{x}_n\}$
       Noise schedule $\{\beta_1, \ldots, \beta_T\}$
Output: Network parameters $\mathbf{w}$
for $t \in \{1, \ldots, T\}$ do
    $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1 - \beta_\tau)$ // Calculate alphas from betas
end for
repeat
    $\mathbf{x} \sim \mathcal{D}$ // Sample a data point
    $t \sim \{1, \ldots, T\}$ // Sample a point along the Markov chain
    $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ // Sample a noise vector
    $\mathbf{z}_t \leftarrow \sqrt{\alpha_t}\mathbf{x} + \sqrt{1 - \alpha_t}\boldsymbol{\epsilon}$ // Evaluate noisy latent variable
    $\mathcal{L}(\mathbf{w}) \leftarrow \|\mathbf{g}(\mathbf{z}_t, \boldsymbol{\vartheta}, t) - \boldsymbol{\epsilon}\|^2$ // Compute loss term
    Take optimizer step
until converged
return $\mathbf{w}$
```

Neural network
with suitable architecture

[Image from https://www.bishopbook.com/]

Forward diffusion

Forward Diffusion
Assume that images
are corrupted by
Gaussian noise
with known parameters



$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$

$$\boldsymbol{x}_t \sim \mathcal{N}\left(\sqrt{1-\beta_t}\,\boldsymbol{x}_{t-1}, \beta_t \boldsymbol{I}\right)$$

$$\beta_t \in (0,1),\ \forall t$$

$$\beta_1 < \beta_2 < \cdots < \beta_T$$

Hyperparameters

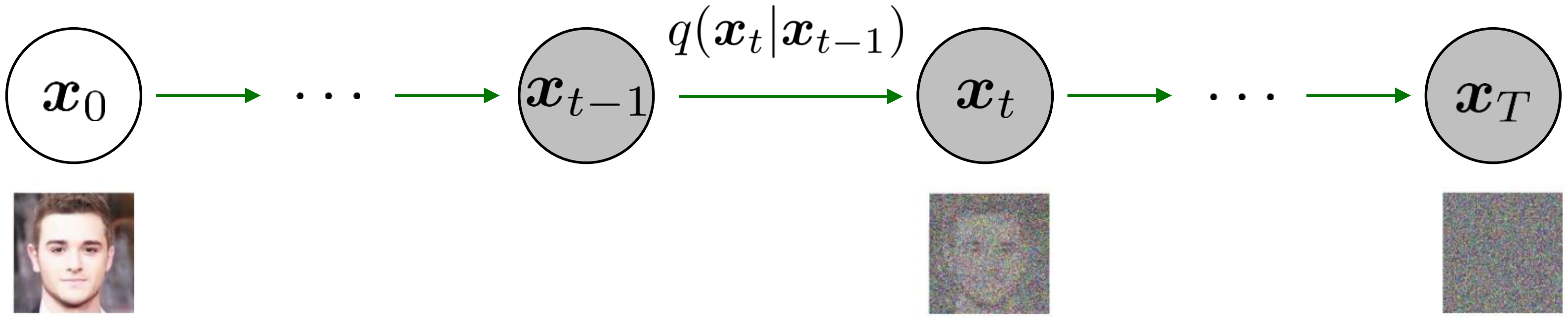
Could be rewritten as

$$\boldsymbol{x}_t = \sqrt{1-\beta_t}\,\boldsymbol{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\varepsilon}$$

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

Forward diffusion

Forward Diffusion
Assume that images
are corrupted by
Gaussian noise
with known parameters

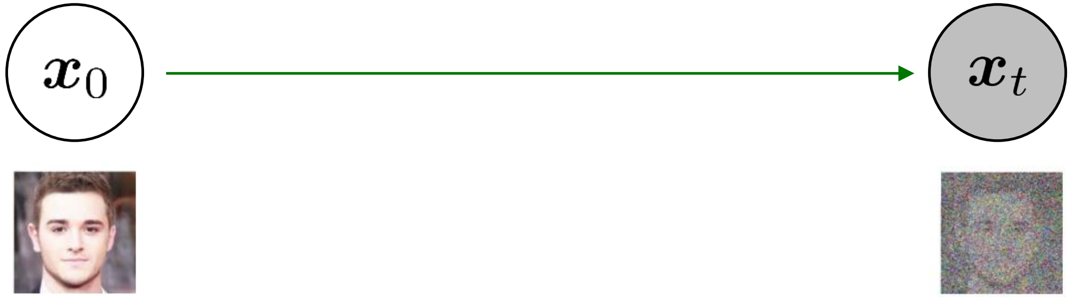


At any forward step t, the diffusion sequence can be compacted as

$$\boldsymbol{x}_t \sim \mathcal{N}\left(\sqrt{\alpha_t}\,\boldsymbol{x}_0, (1-\alpha_t)\boldsymbol{I}\right)$$

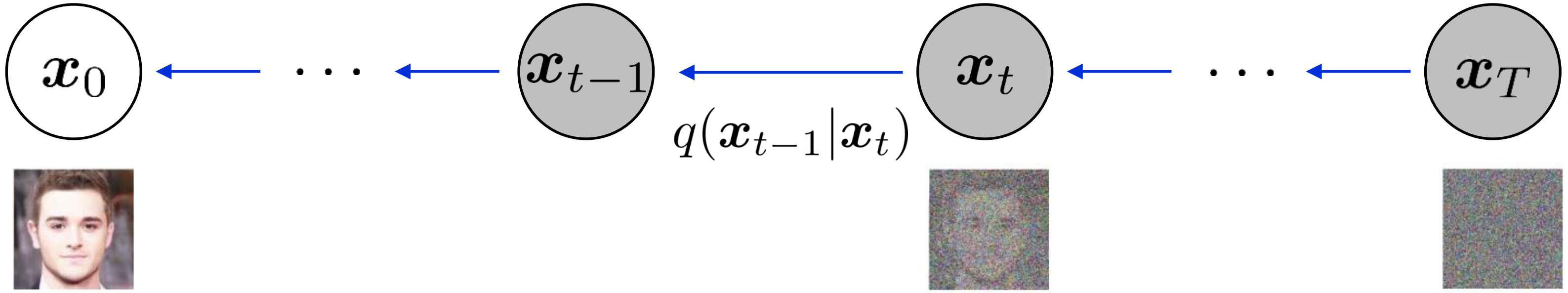
where:

$$\alpha_t = \prod_{\tau=1}^{t}(1-\beta_\tau)$$



Going backward: denoising

Backward Denoising
A neural network
is at the core
of the backward process



We assume that:

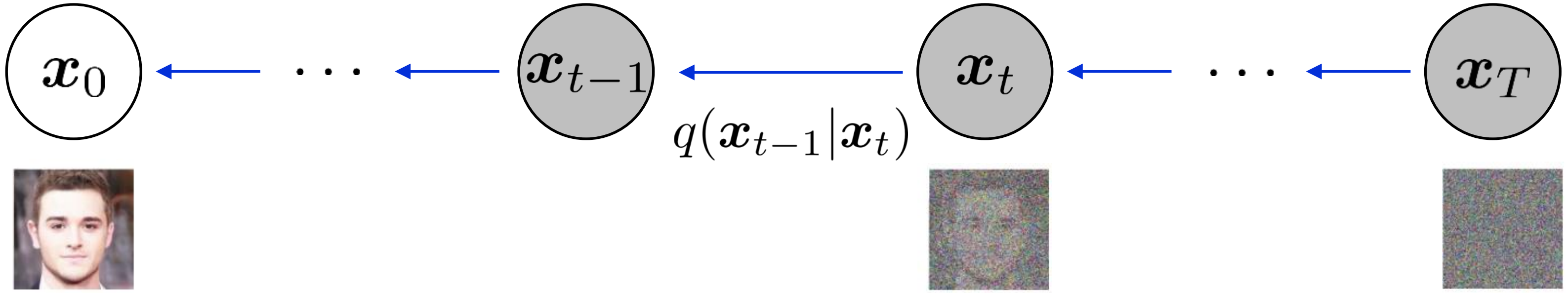
$$\boldsymbol{x}_{t-1} = \boldsymbol{\mu}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta}) + \sqrt{\beta_t}\,\boldsymbol{\varepsilon}$$

$$\boldsymbol{\mu}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta}) = \frac{1}{\sqrt{1-\beta_t}}\left\{\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\overbrace{\boldsymbol{g}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta})}^{\textit{Neural Network}}\right\}$$

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

Going backward: denoising

Backward Denoising
A neural network
is at the core
of the backward process



We assume that:

$$\boldsymbol{x}_{t-1} = \boldsymbol{\mu}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta}) + \sqrt{\beta_t}\,\boldsymbol{\varepsilon}$$

$$\boldsymbol{\mu}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta}) = \frac{1}{\sqrt{1-\beta_t}}\left\{\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\overbrace{\boldsymbol{g}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta})}^{\textit{Neural Network}}\right\}$$

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

How can the neural network be trained?
(A suitable loss function is needed)

Going backward: denoising

$\tilde{q}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$

An approximation to $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$

We assume that:

$$\boldsymbol{x}_{t-1} \sim \mathcal{N}\left(\boldsymbol{\mu}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta}), \beta_t \boldsymbol{I}\right)$$

During <u>training</u>, $\boldsymbol{x}_0$ is known. Then we can sample $\boldsymbol{\varepsilon}_t$

$$\boldsymbol{x}_t = \sqrt{\alpha_t}\, \boldsymbol{x}_0 + \sqrt{1-\alpha_t}\, \boldsymbol{\varepsilon}_t$$

Noise added at step t

Therefore, it can be proven that:

$$\boldsymbol{m}(\boldsymbol{x}_{t-1}) = \frac{1}{\sqrt{1-\beta_t}} \left(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\, \boldsymbol{\varepsilon}_t \right)$$

is the true mean of:

$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$$

Then the Kullback-Leibler divergence is:

$$\begin{aligned}&\mathrm{KL}\left(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) \parallel \tilde{q}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)\right) \\ &\quad = \frac{1}{2\beta_t} \|\boldsymbol{m}(\boldsymbol{x}_{t-1}) - \boldsymbol{\mu}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta})\|^2 + \mathrm{const}\end{aligned}$$

Going backward: denoising

$$\begin{aligned}\mathrm{KL}\left(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)\;\|\;\tilde{q}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)\right)\\ = \frac{1}{2\beta_t}\|\boldsymbol{\mu}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta}) - \boldsymbol{m}(\boldsymbol{x}_{t-1})\|^2 + \mathrm{const}\end{aligned}$$

Therefore, given (see before):

$$\boldsymbol{m}(\boldsymbol{x}_{t-1}) = \frac{1}{\sqrt{1-\beta_t}}\left(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{\varepsilon}_t\right)$$

$$\boldsymbol{\mu}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta}) = \frac{1}{\sqrt{1-\beta_t}}\left\{\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{g}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta})\right\}$$

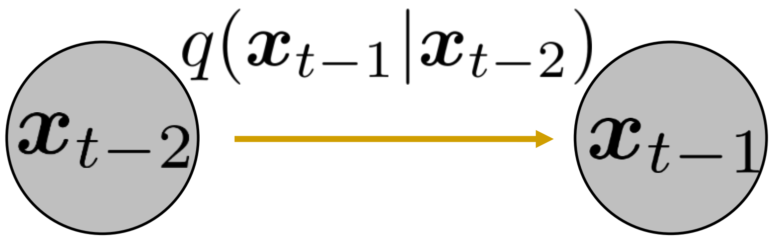
$$\mathrm{KL}\left(\tilde{q}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)\;\|\;q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)\right) \propto \|\;\boldsymbol{g}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta}) - \boldsymbol{\varepsilon}_t\|^2$$

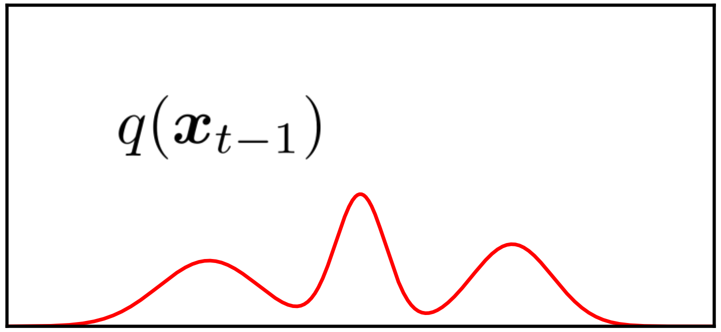
$$L(\boldsymbol{\vartheta}) := \|\;\boldsymbol{g}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta}) - \boldsymbol{\varepsilon}_t\|^2$$

To be minimized

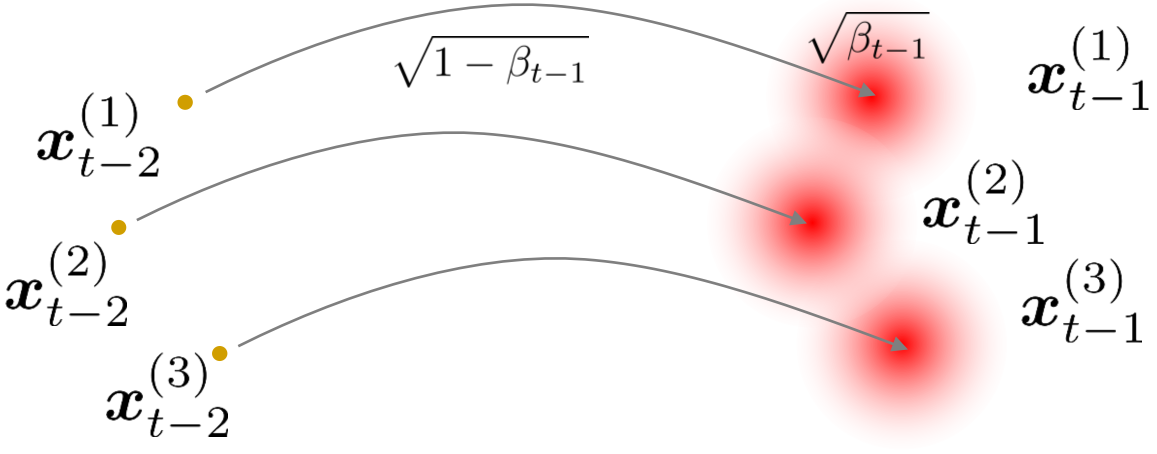
Diffusion Models:
Why so many steps?

Going forward: adding noise



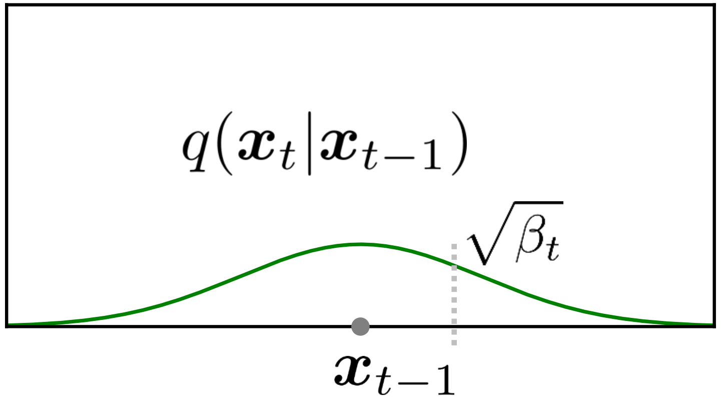


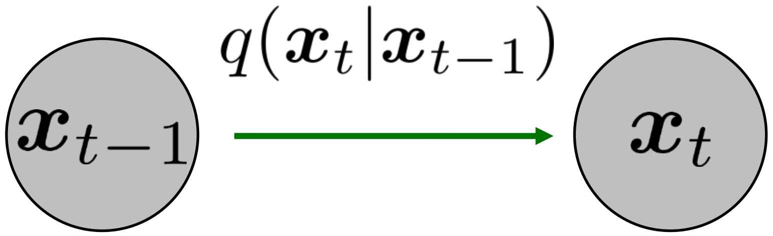
Marginal distribution at step $t-1$



$\sqrt{1-\beta_{t-1}}$
$\sqrt{\beta_{t-1}}$
$\boldsymbol{x}_{t-2}^{(1)}$
$\boldsymbol{x}_{t-2}^{(2)}$
$\boldsymbol{x}_{t-2}^{(3)}$
$\boldsymbol{x}_{t-1}^{(1)}$
$\boldsymbol{x}_{t-1}^{(2)}$
$\boldsymbol{x}_{t-1}^{(3)}$

Forward probability distribution (Gaussian)

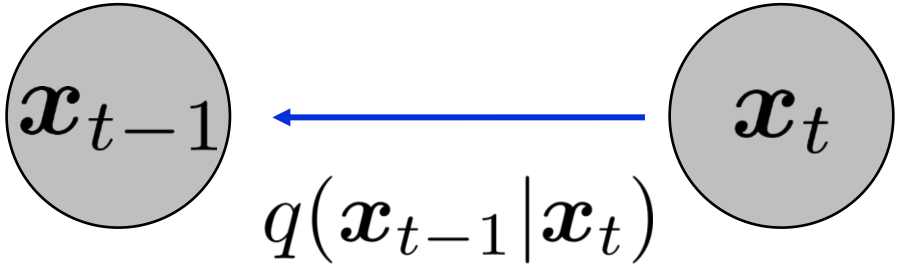


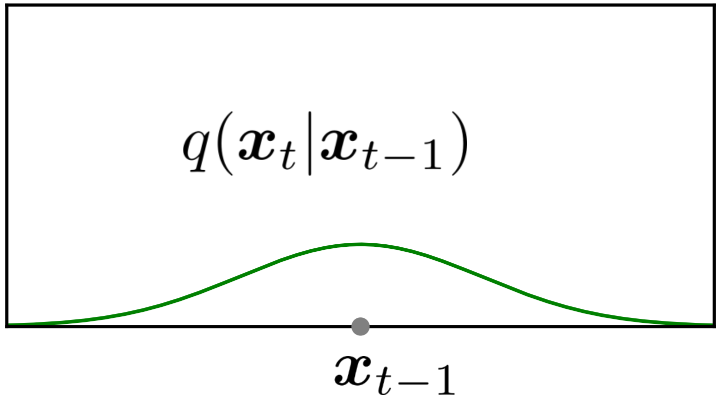


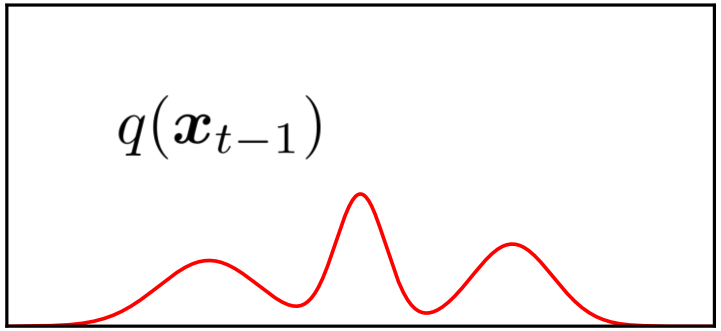
$$\boldsymbol{x}_t = \sqrt{1-\beta_t}\,\boldsymbol{x}_{t-1} + \sqrt{\beta_t}\,\varepsilon$$

Going backward: denoising

The backward probability distribution can be computed from forward and marginals using Bayes' theorem





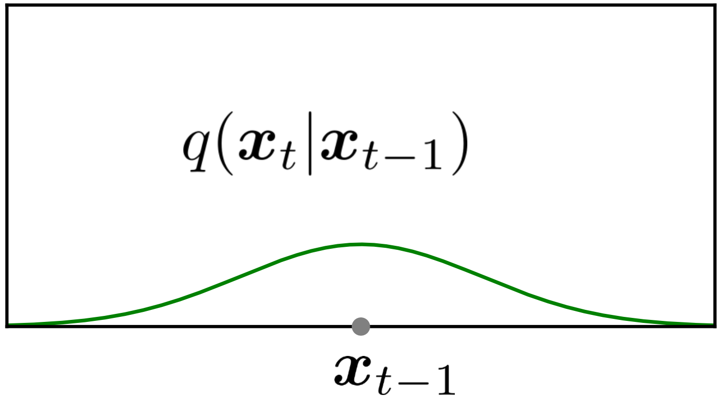


This is what we want to learn

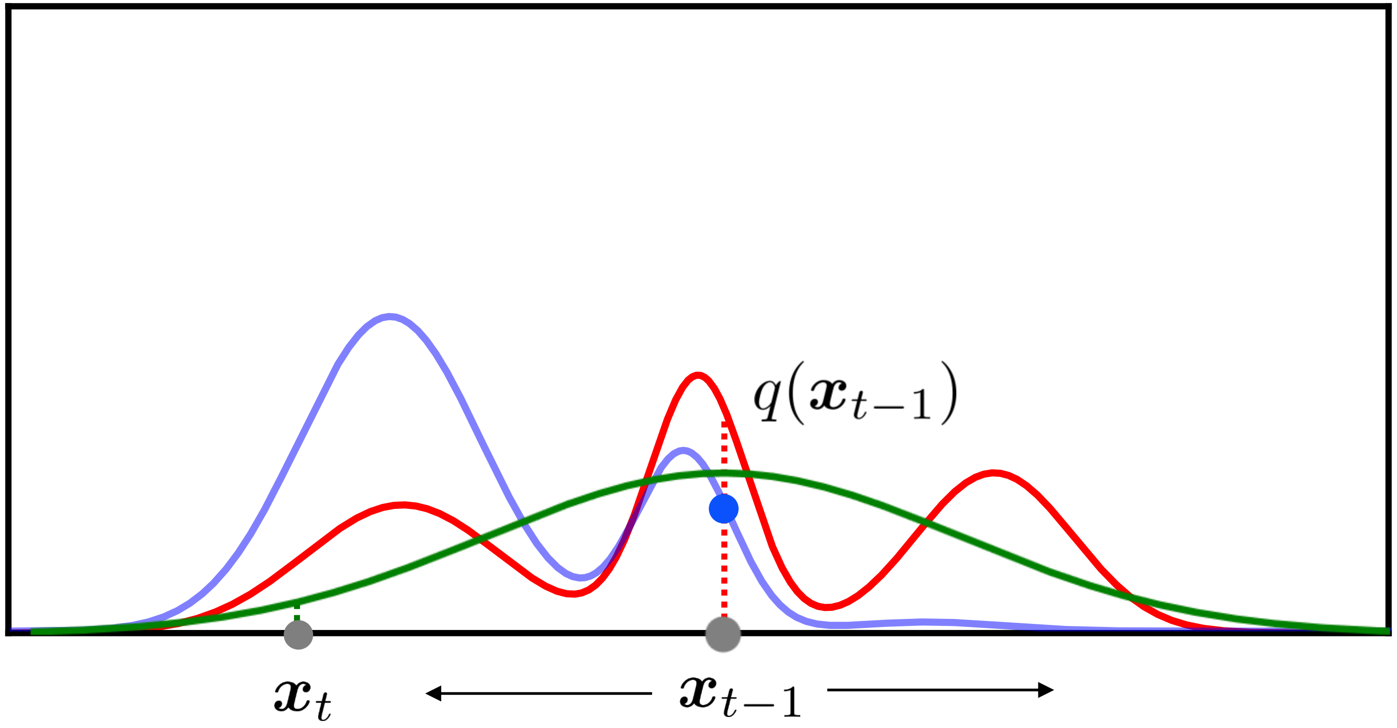
$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \frac{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})q(\boldsymbol{x}_{t-1})}{q(\boldsymbol{x}_t)}$$

Bayes' Theorem

Going backward: denoising



At training time, $\boldsymbol{x}_t$ is known (dataset + forward diffusion)

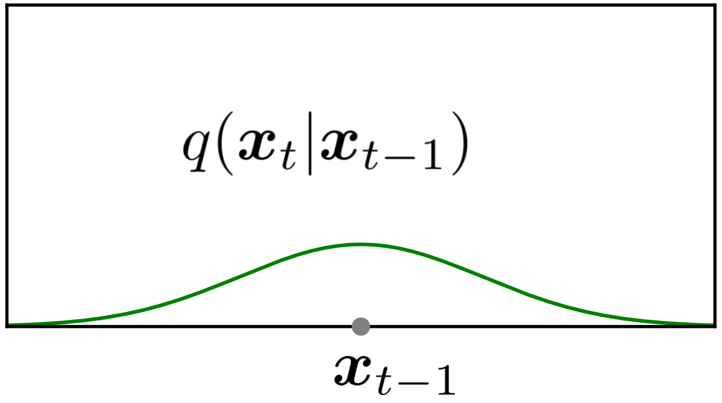


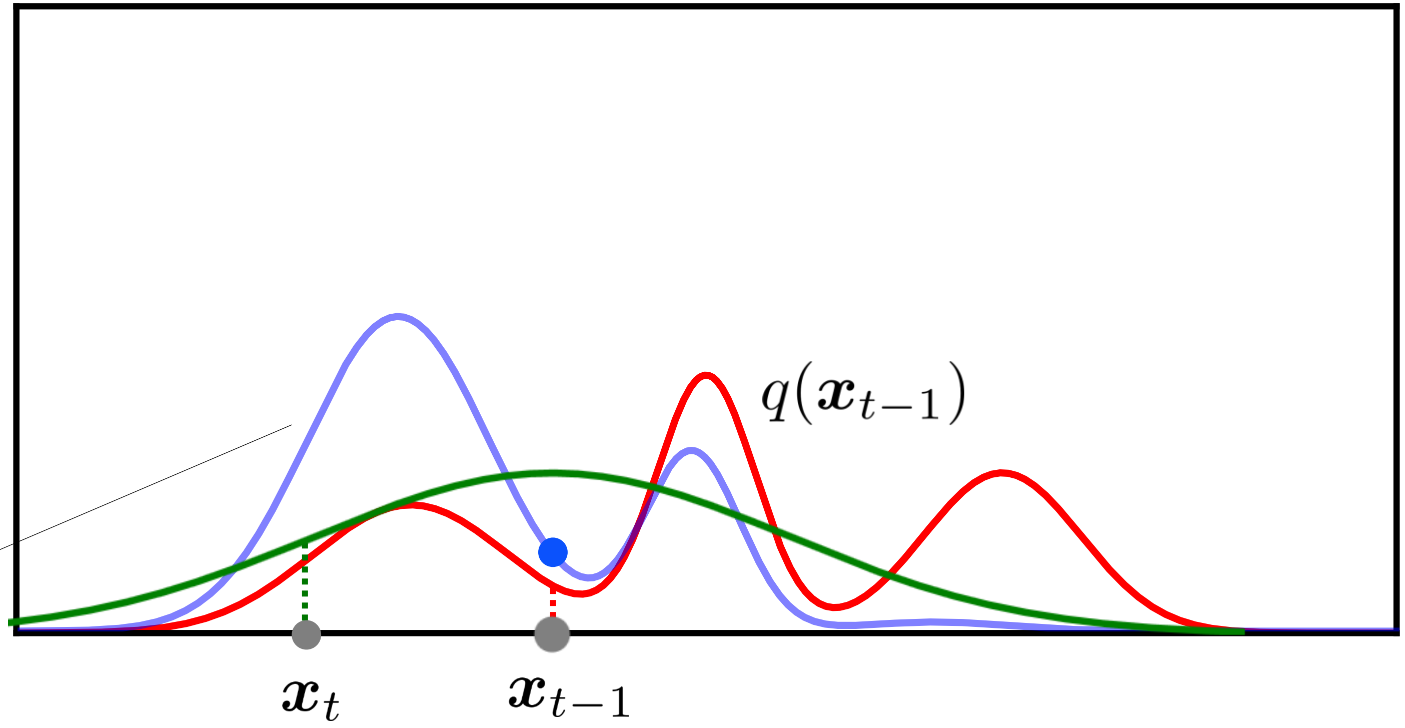
$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \frac{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})q(\boldsymbol{x}_{t-1})}{q(\boldsymbol{x}_t)}$$

Bayes' Theorem

[Image from https://www.bishopbook.com/]

Going backward: denoising





At training time, $\boldsymbol{x}_t$ is known (dataset + forward diffusion)

The reverse probability is the blue curve

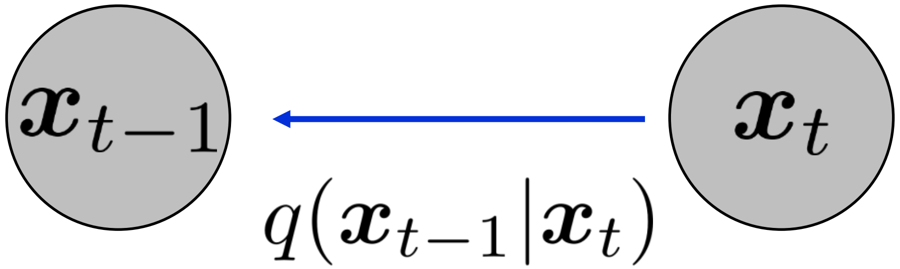
$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \frac{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})q(\boldsymbol{x}_{t-1})}{q(\boldsymbol{x}_t)}$$

Bayes' Theorem

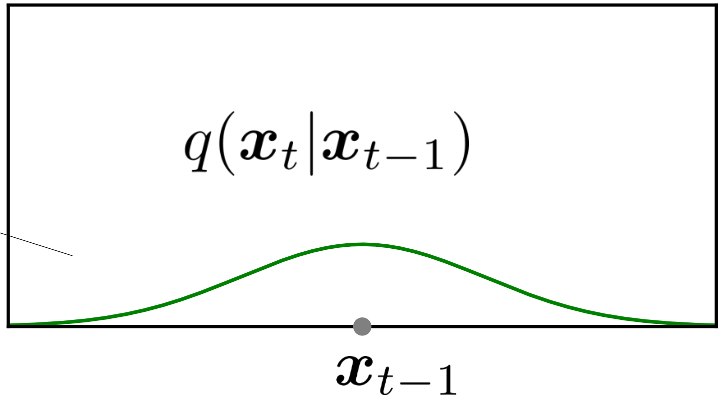
[Image from https://www.bishopbook.com/]

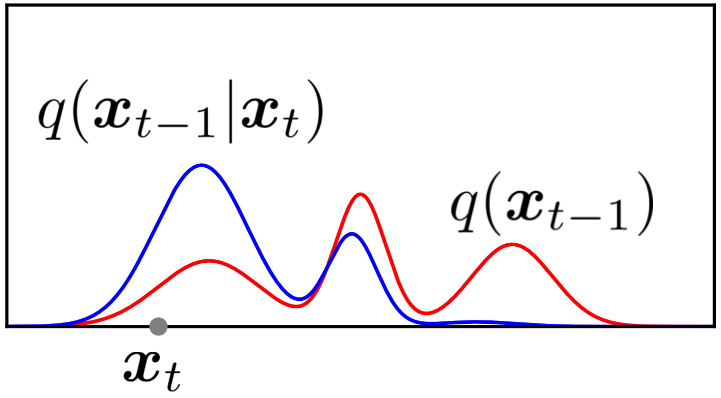
Going backward: denoising

When β_t is large
$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$
becomes very different from a Gaussian, hence unsuitable for training



β_t large





$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \frac{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})q(\boldsymbol{x}_{t-1})}{q(\boldsymbol{x}_t)}$$

Bayes' Theorem

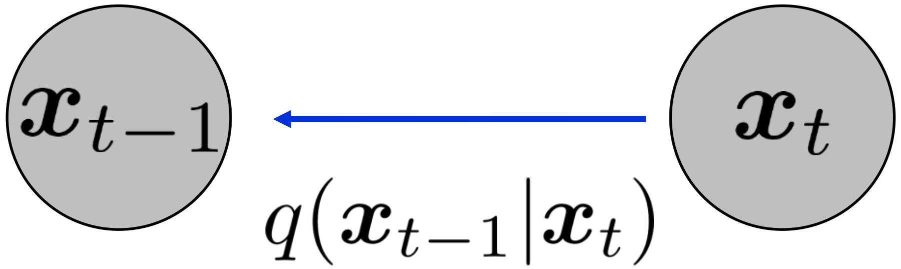
[Image from https://www.bishopbook.com/]

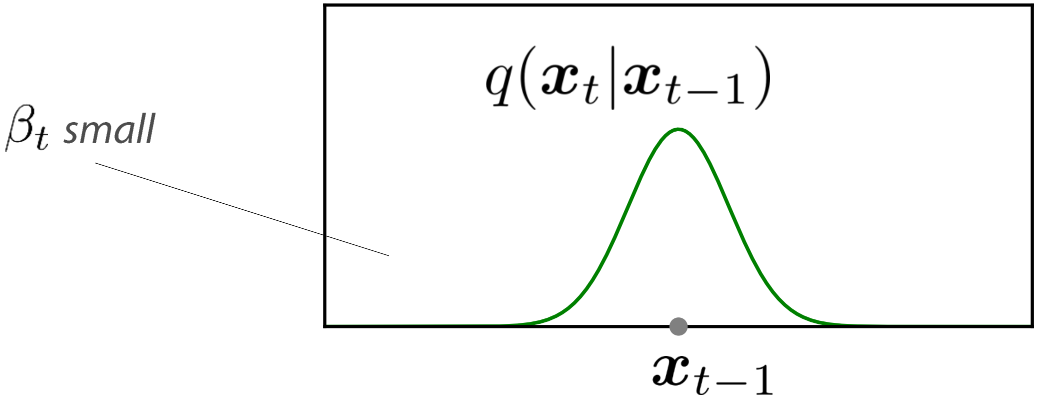
Going backward: denoising

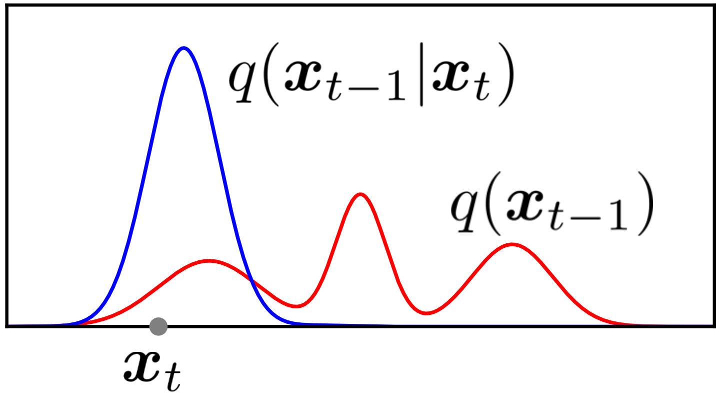
When β_t is small

$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$

is approximately Gaussian




β_t small



$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \frac{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})q(\boldsymbol{x}_{t-1})}{q(\boldsymbol{x}_t)}$$

Bayes' Theorem

Links

https://www.assemblyai.com/blog/diffusion-models-for-machine-learning-introduction/

https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

https://www.superannotate.com/blog/diffusion-models

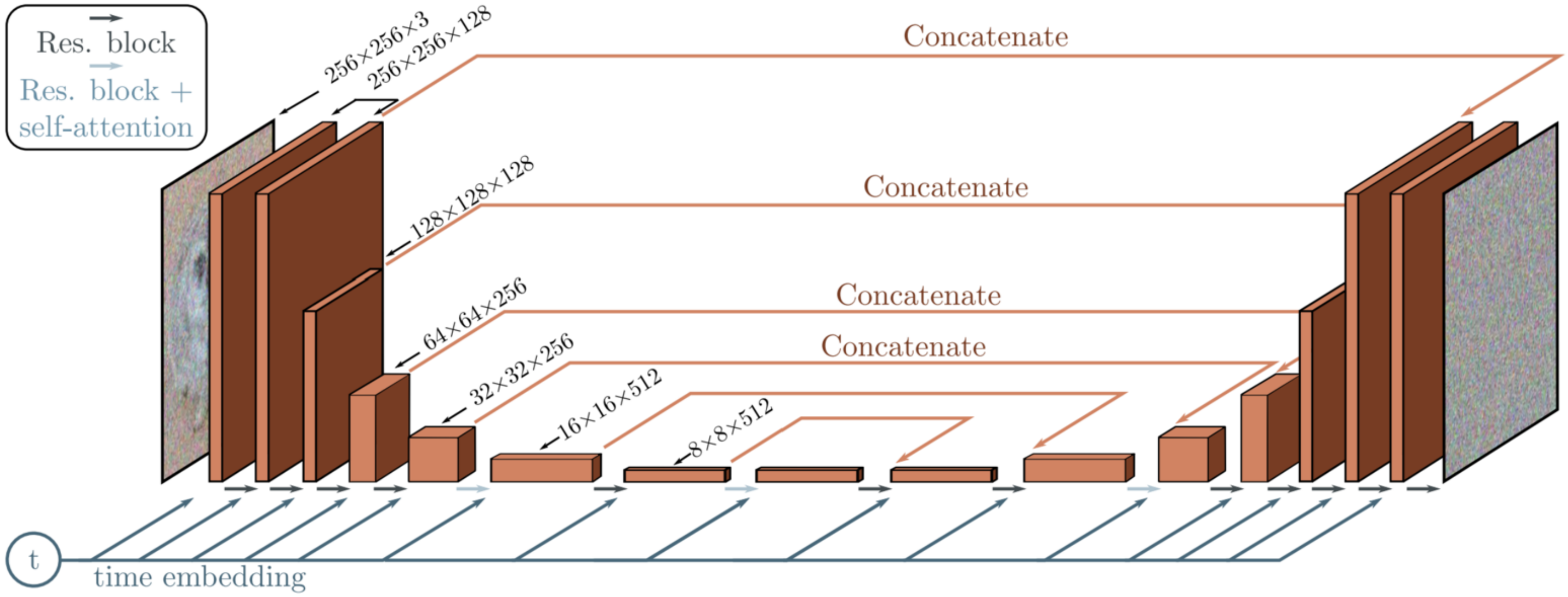
https://encord.com/blog/diffusion-models/

Practical Implementation

Conditional U-Net as basic denoising block

Loss function: $L(\boldsymbol{\vartheta}) := \| \boldsymbol{g}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta}) - \boldsymbol{\varepsilon}_t \|^2$

The network architecture for $\boldsymbol{g}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta})$ *is a U-Net*



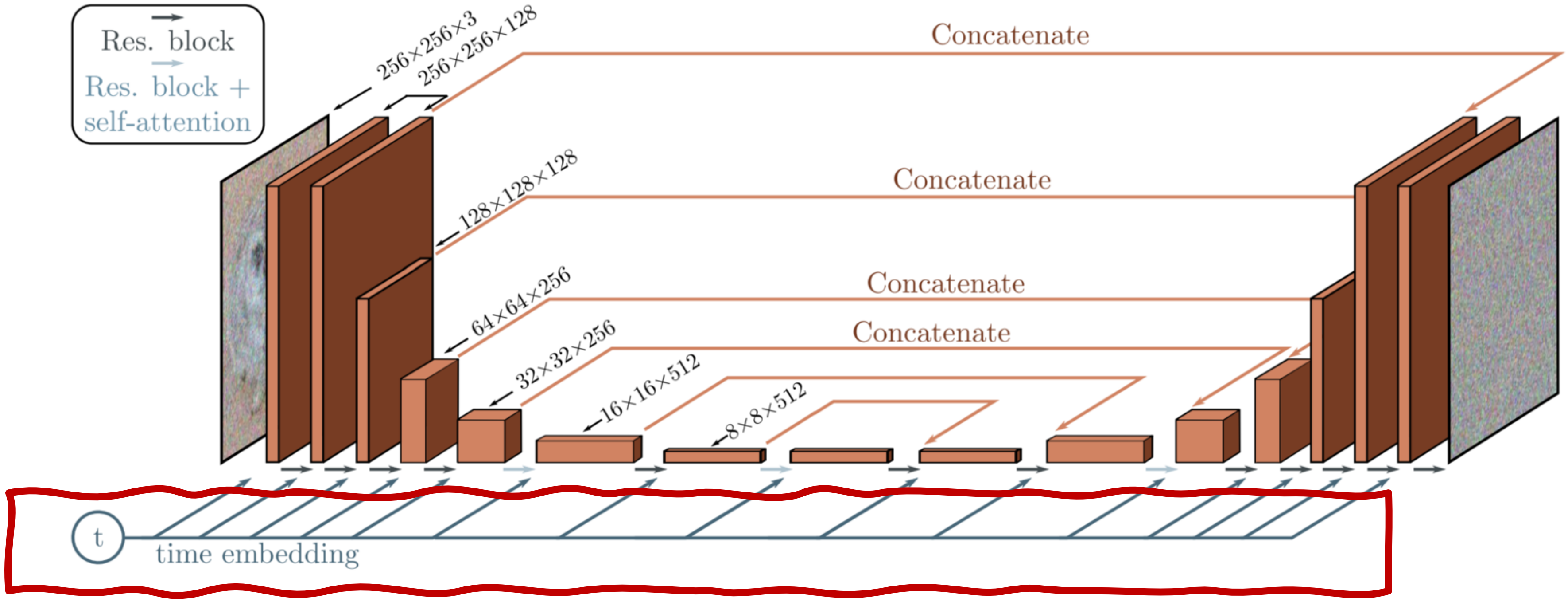
[Image from https://learnopencv.com/denoising-diffusion-probabilistic-models/]

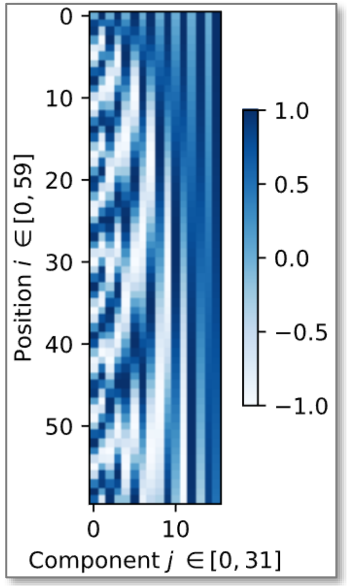
[Ho, Jain & Abbeel, 2020 - https://arxiv.org/pdf/2006.11239]

Conditional U-Net as basic denoising block

Loss function: $L(\boldsymbol{\vartheta}) := \| \boldsymbol{g}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta}) - \boldsymbol{\varepsilon}_t \|^2$

The network architecture for $\boldsymbol{g}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta})$ *is a U-Net*





[Image from https://learnopencv.com/denoising-diffusion-probabilistic-models/]

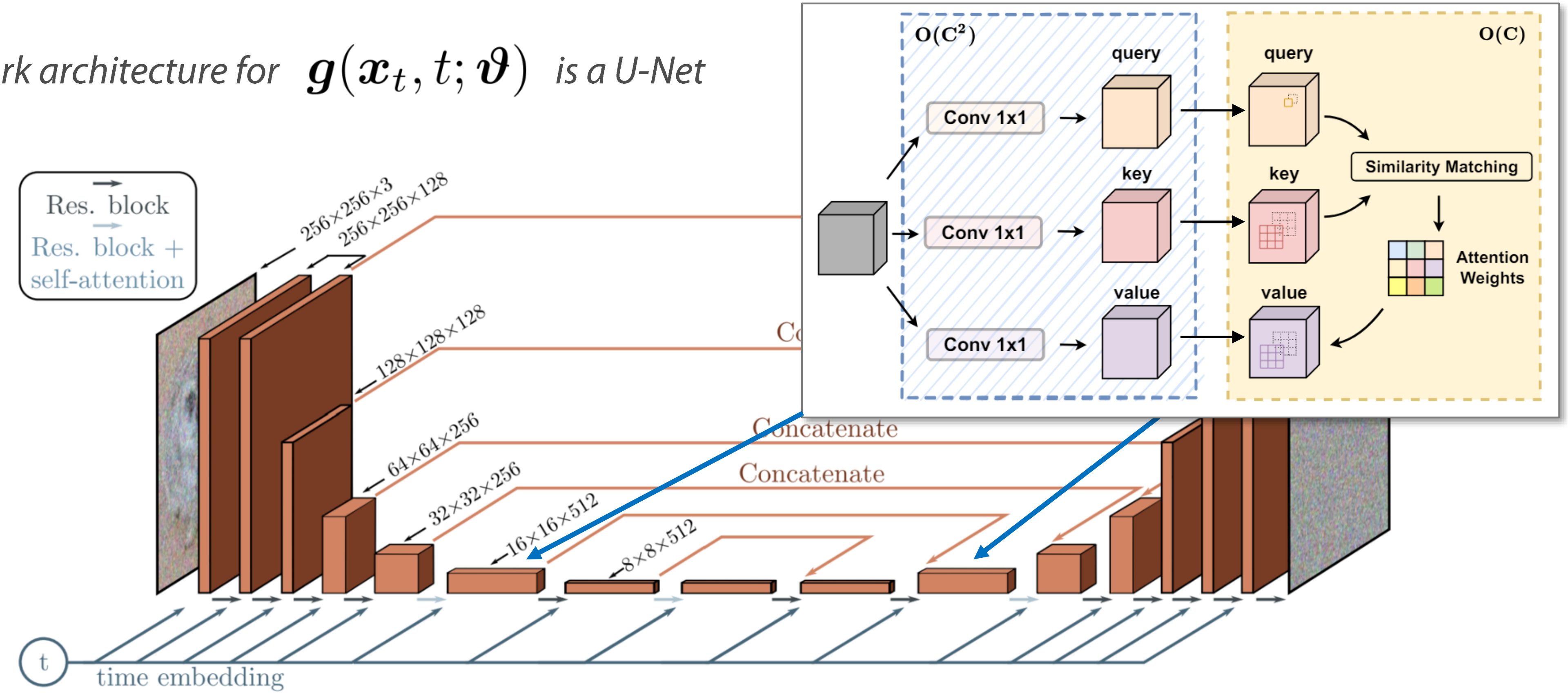
The U-Net is conditioned by the time parameter which is embedded with sinusoidal positioning and added to each residual block

[Ho, Jain & Abbeel, 2020 - https://arxiv.org/pdf/2006.11239]

Conditional U-Net as basic denoising block

Loss function: $L(\boldsymbol{\vartheta}) := \|\, \boldsymbol{g}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta}) - \boldsymbol{\varepsilon}_t \|^2$

The network architecture for $\boldsymbol{g}(\boldsymbol{x}_t, t; \boldsymbol{\vartheta})$ is a U-Net



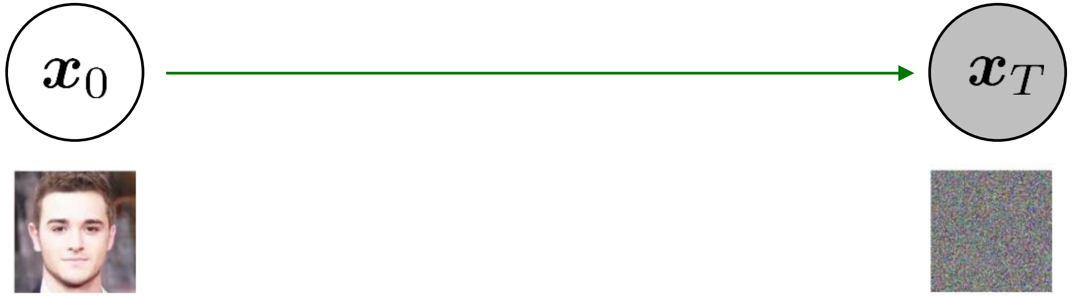
[Image from https://learnopencv.com/denoising-diffusion-probabilistic-models/]

Self- Attention modules
are interspersed with convolutional blocks in the pipeline

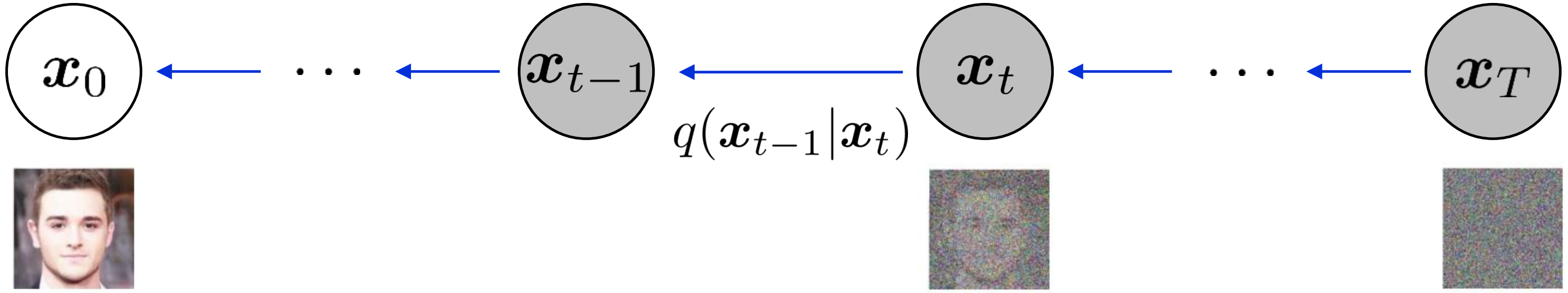
[Ho, Jain & Abbeel, 2020 - https://arxiv.org/pdf/2006.11239]

Latent Diffusion Models

Forward Diffusion
It is relatively easy and inexpensive
(It can be performed in one step)

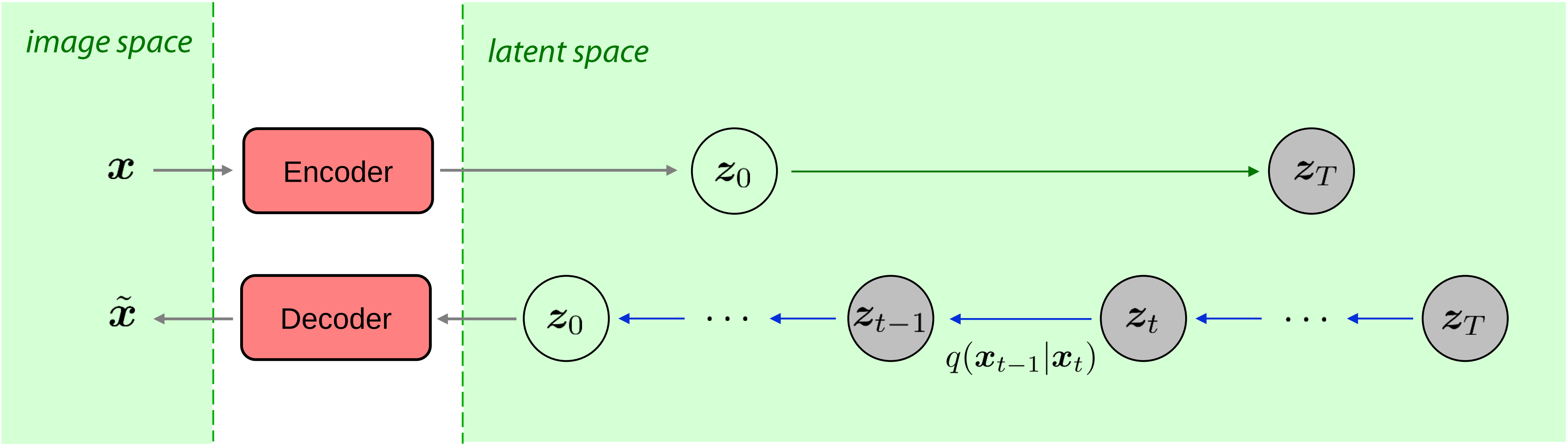


Backward Denoising
Must be performed in small steps
and is quite expensive,
in particular with high-resolution images



Latent Diffusion Models

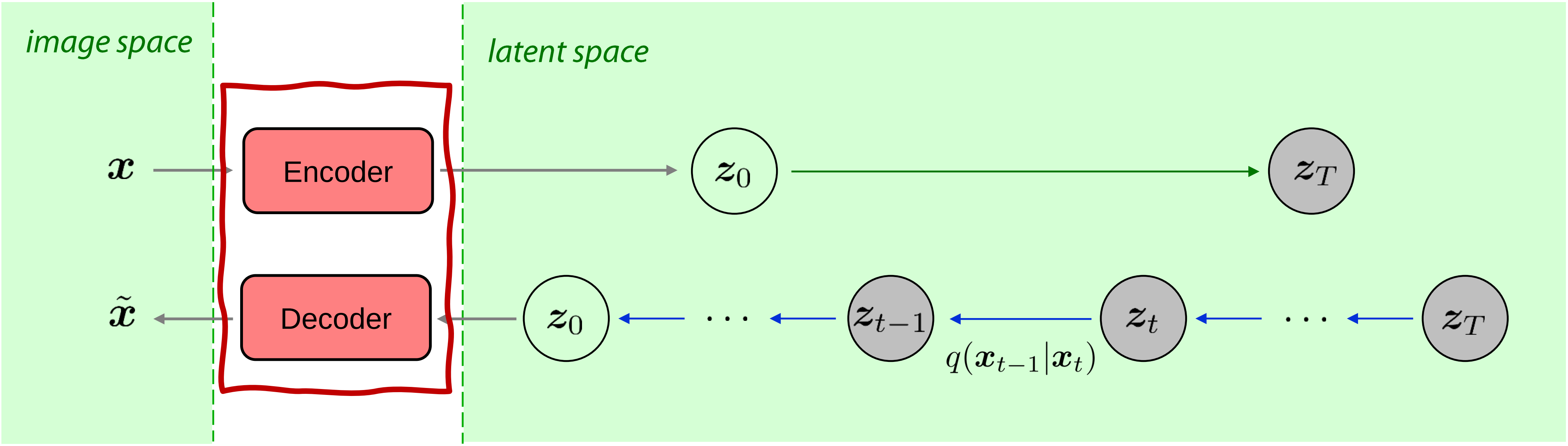
Latent Diffusion Model
The intuitive idea is to perform diffusion in the *latent space*



[Rombach et al., 2022 - https://arxiv.org/pdf/2112.10752]

Latent Diffusion Models

Latent Diffusion Model
The intuitive idea is to perform diffusion in the *latent space*

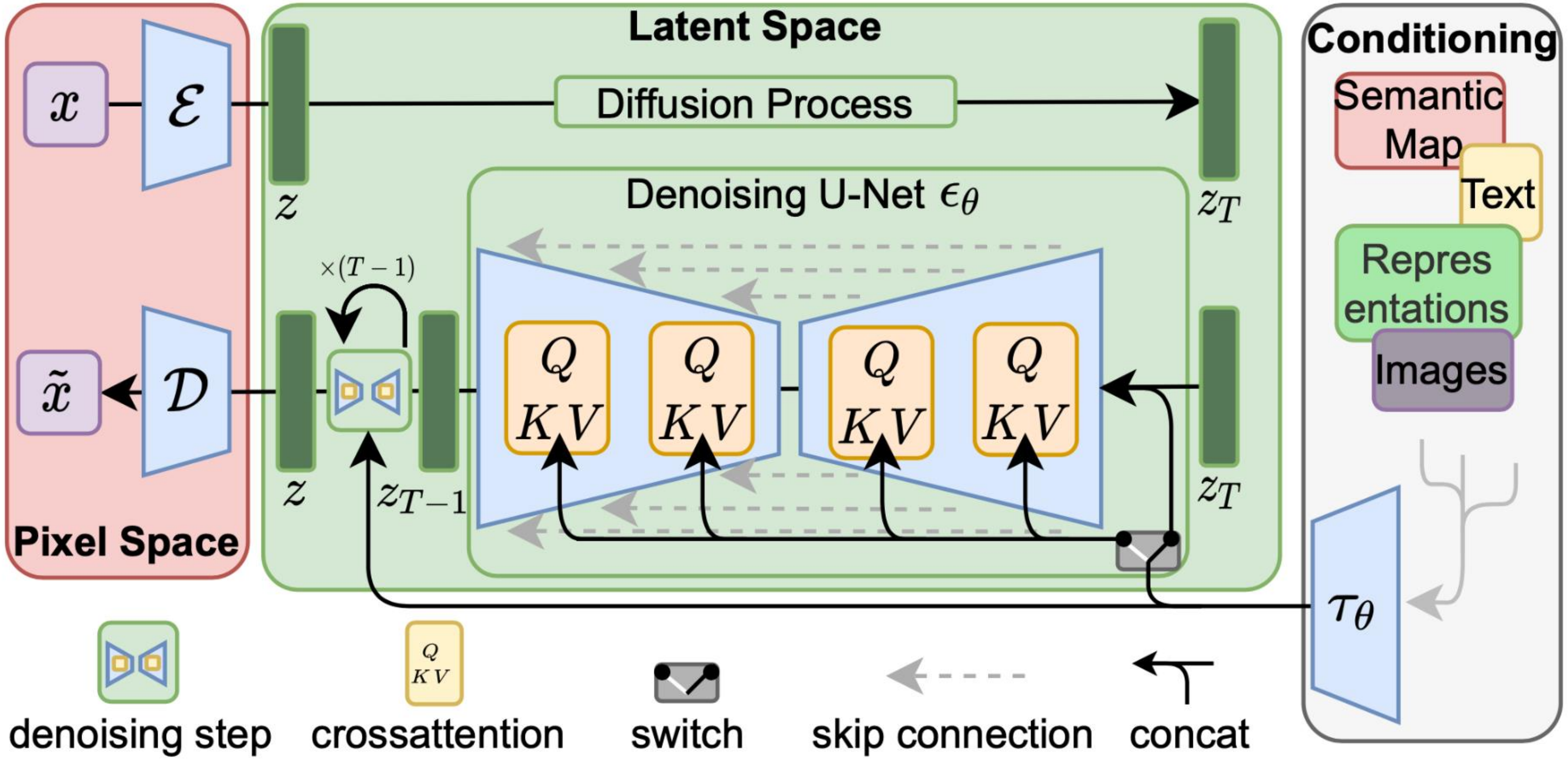


A pre-trained VAE is used to encode and decode high-resolution images into a suitable (reduced) latent format

[Rombach et al., 2022 - https://arxiv.org/pdf/2112.10752]

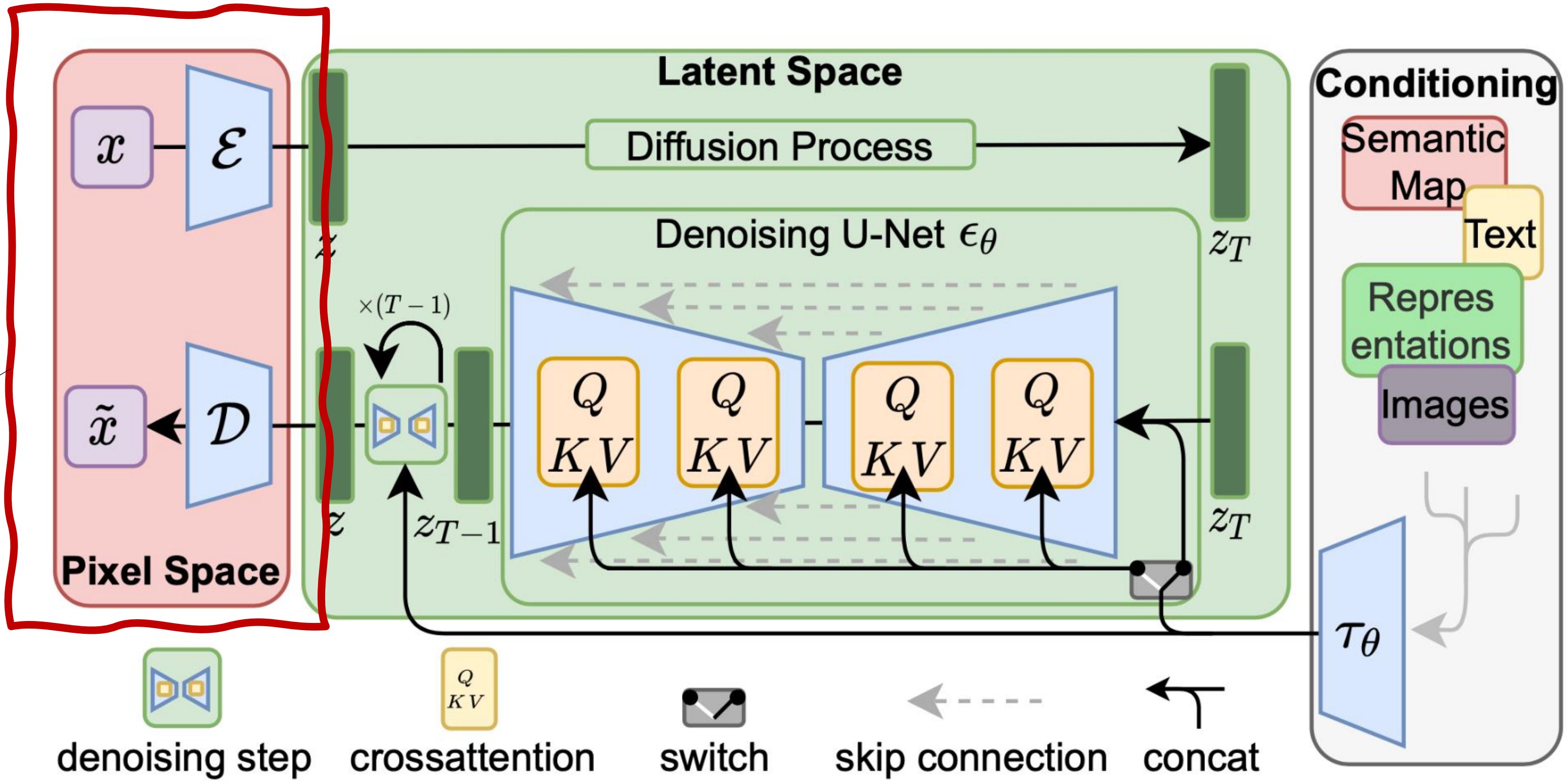
Conditioning on Labels

Latent Diffusion Models with Conditioning



[Image from Rombach et al., 2022 - https://arxiv.org/pdf/2112.10752]

Latent Diffusion Models with Conditioning

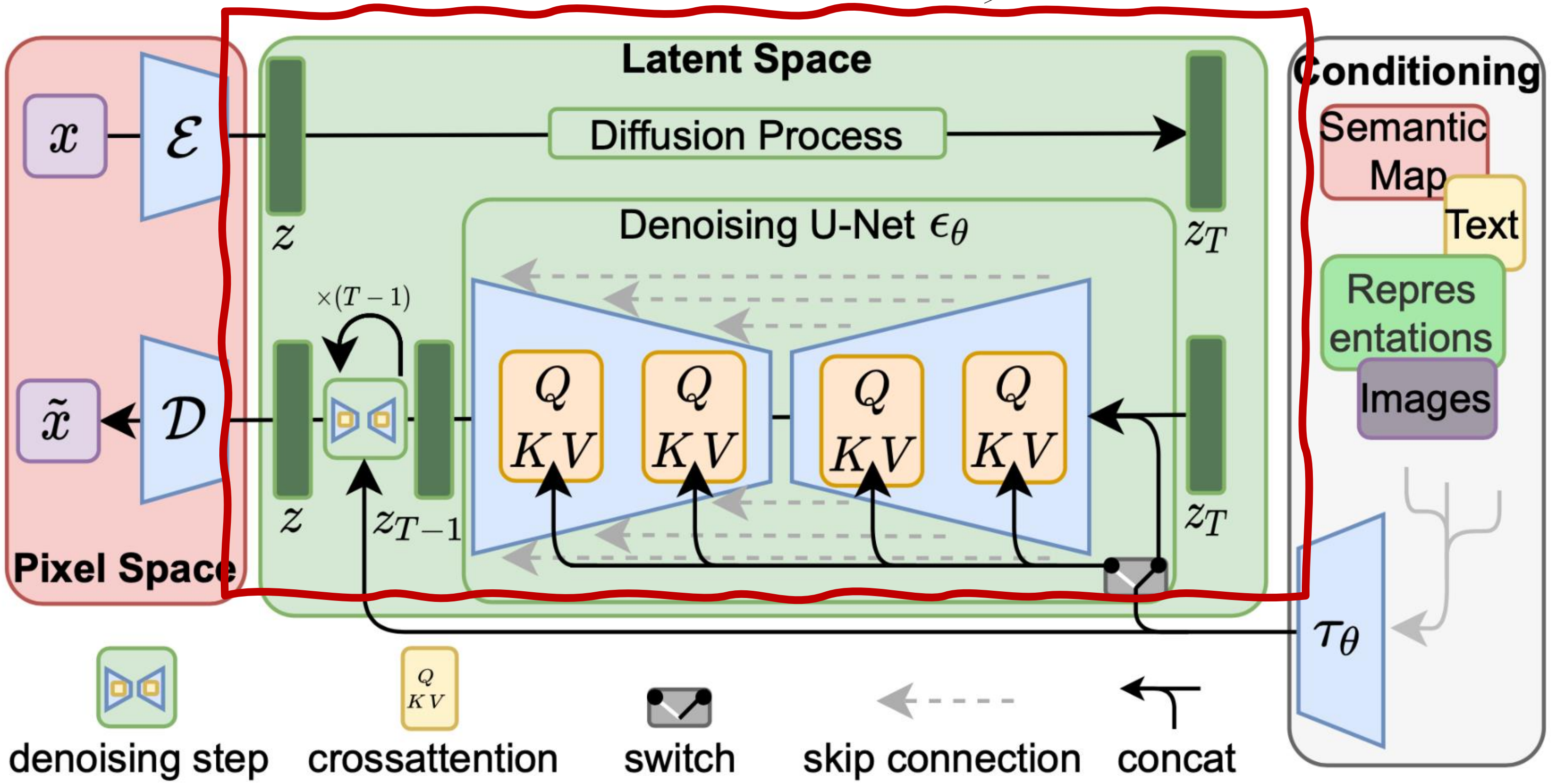


A suitable, VAE-style Encoder-Decoder is *<u>pre-trained</u>*

[Image from Rombach et al., 2022 - https://arxiv.org/pdf/2112.10752]

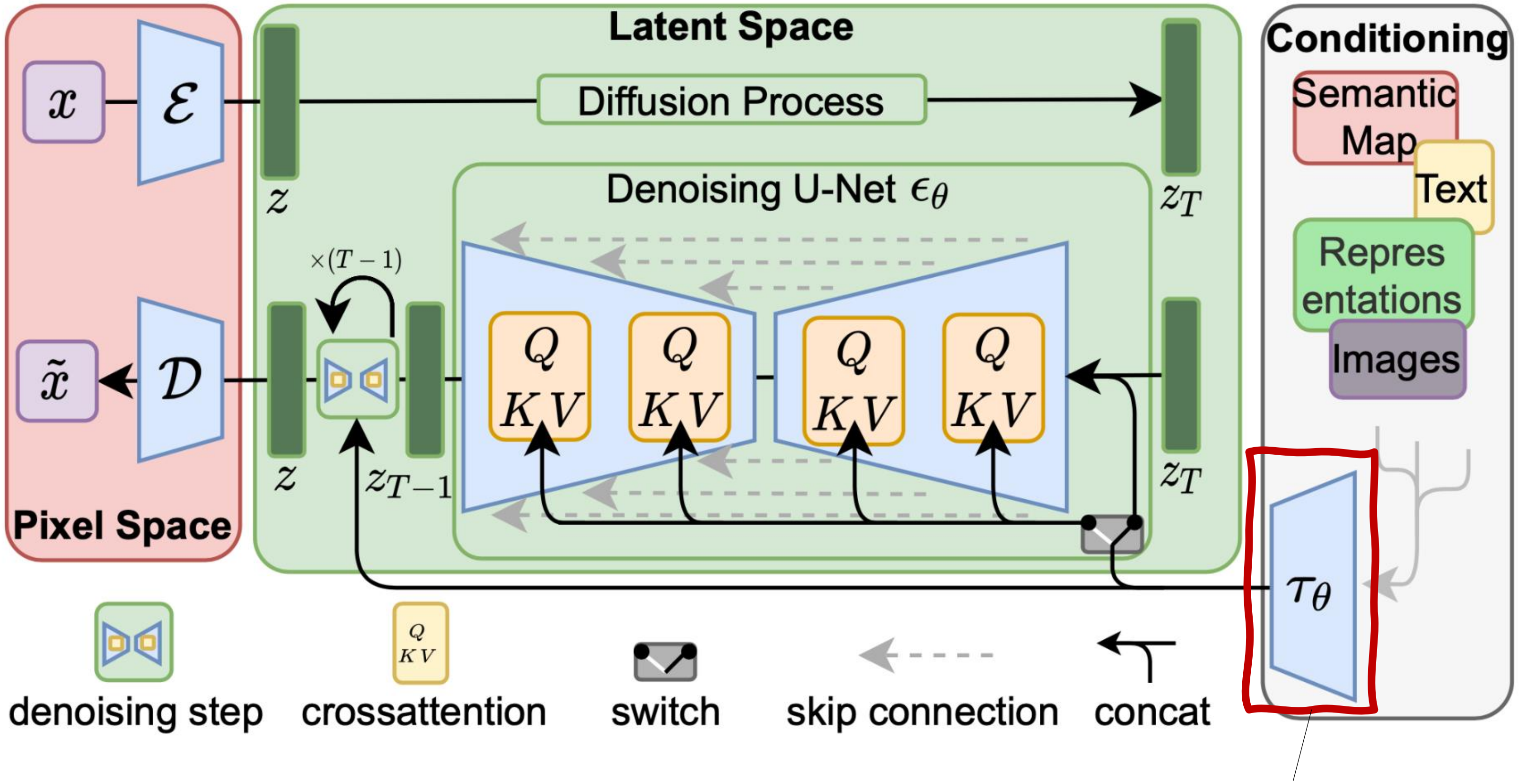
Latent Diffusion Models with Conditioning

The latent diffusion model is then *<u>pre-trained</u>* *(without conditioning)*



[Image from Rombach et al., 2022 - https://arxiv.org/pdf/2112.10752]

Latent Diffusion Models with Conditioning

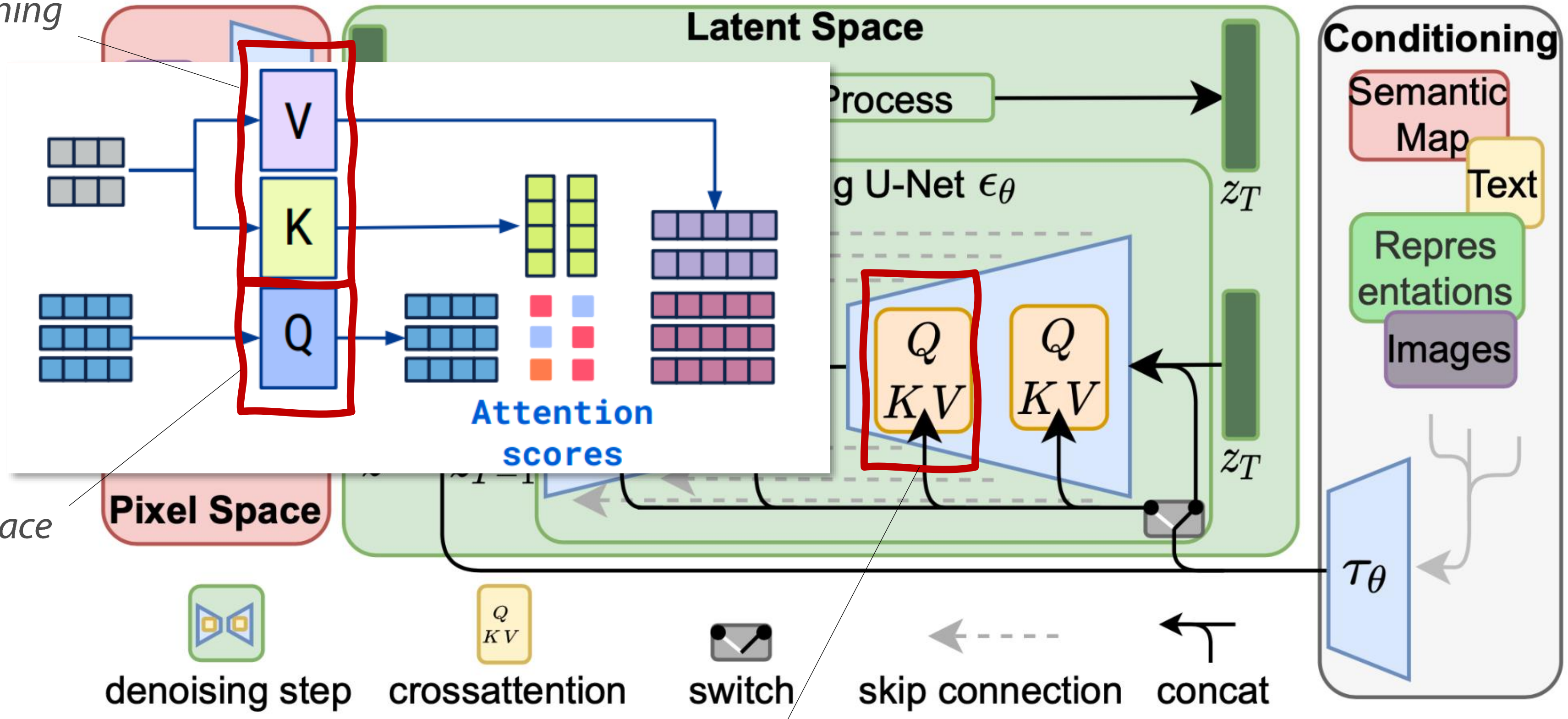


A suitable encoder of the conditioning elements is <u>pre-trained</u> separately

[Image from Rombach et al., 2022 - https://arxiv.org/pdf/2112.10752]

Latent Diffusion Models with Conditioning

Conditioning

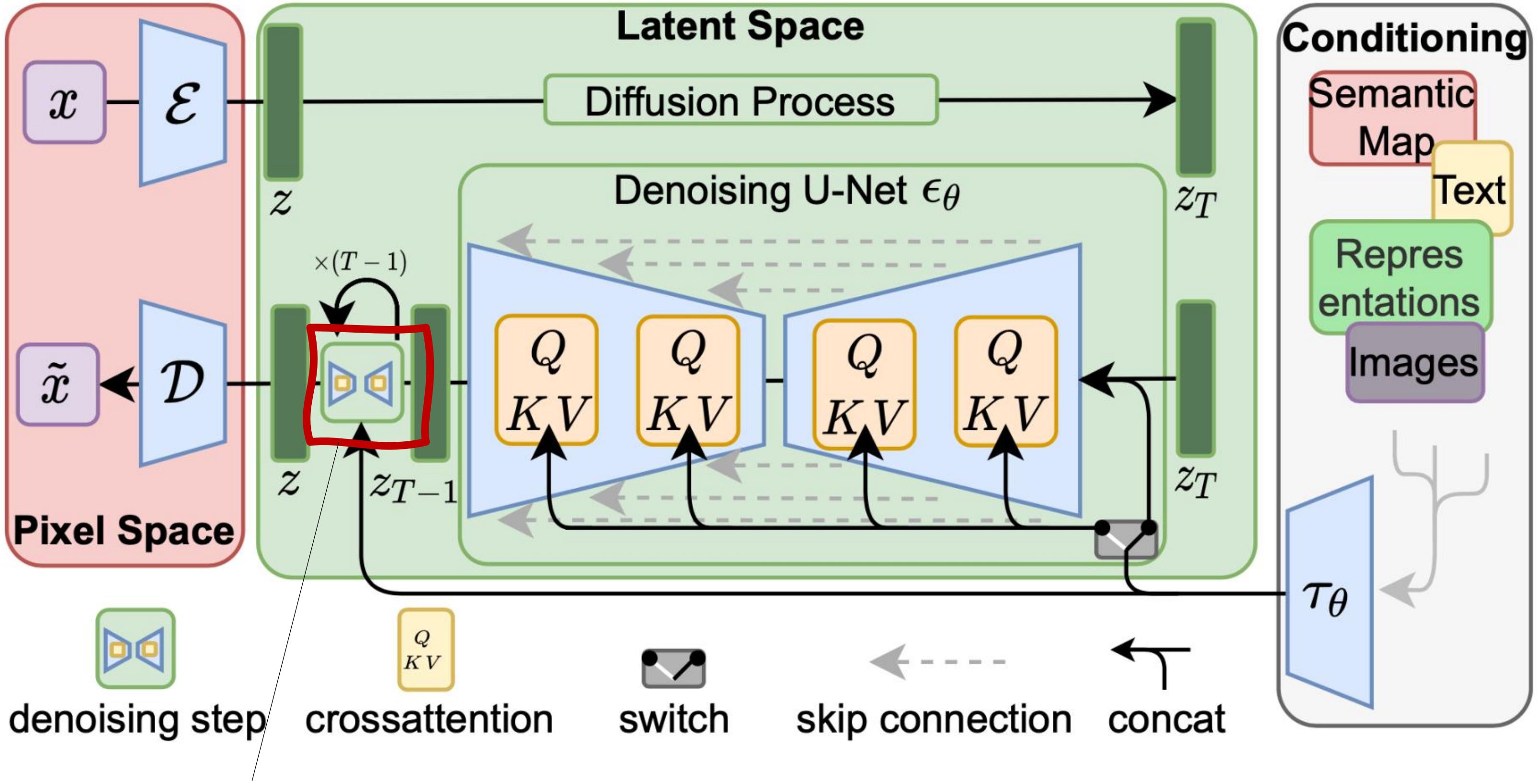


Latent-space

Latent-space representations and embedded condition elements are combined via <u>cross-attention</u>

[Image from Rombach et al., 2022 - https://arxiv.org/pdf/2112.10752]

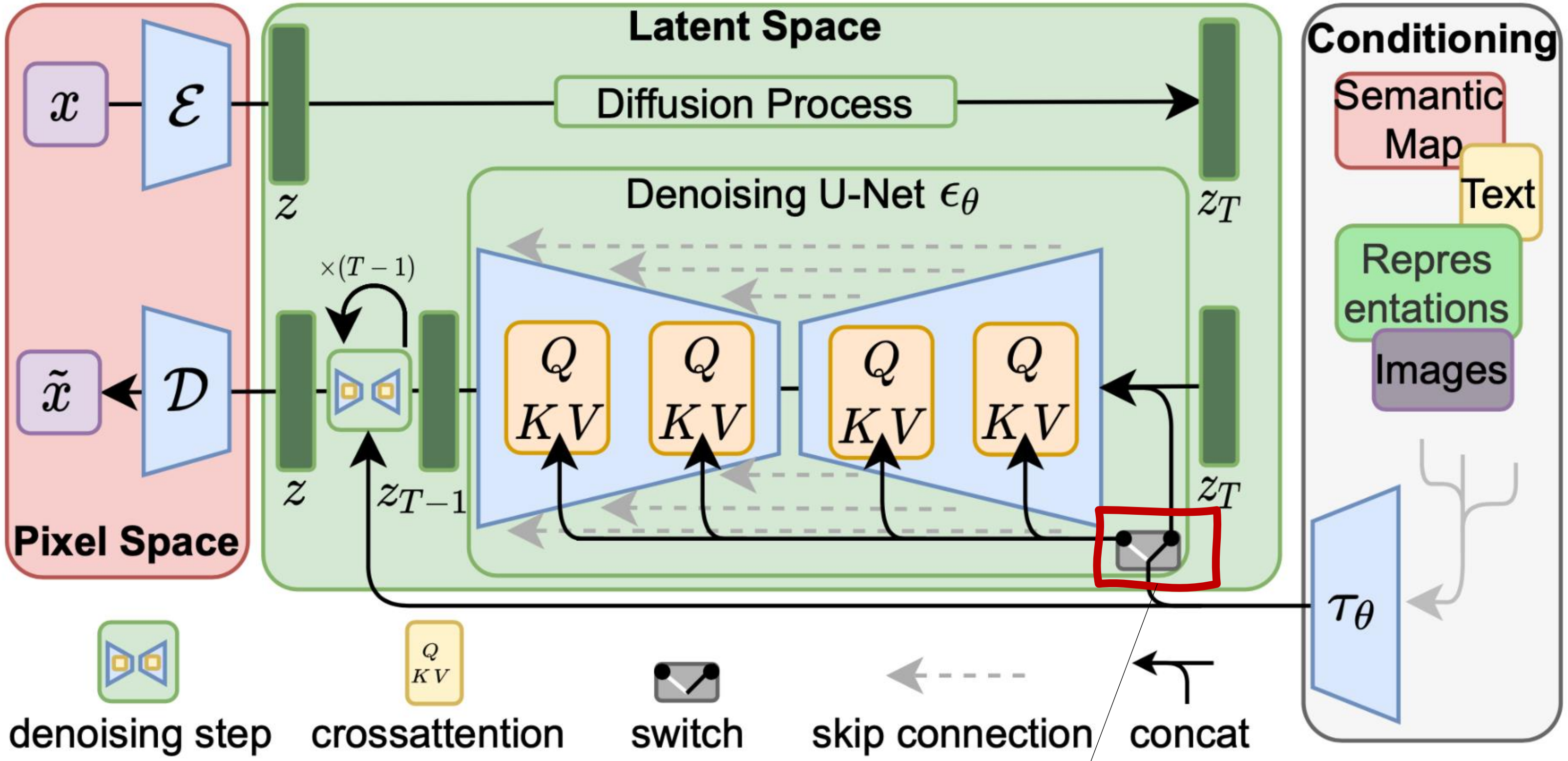
Latent Diffusion Models with Conditioning



The same step is iterated
$T-1$ more times

[Image from Rombach et al., 2022 - https://arxiv.org/pdf/2112.10752]

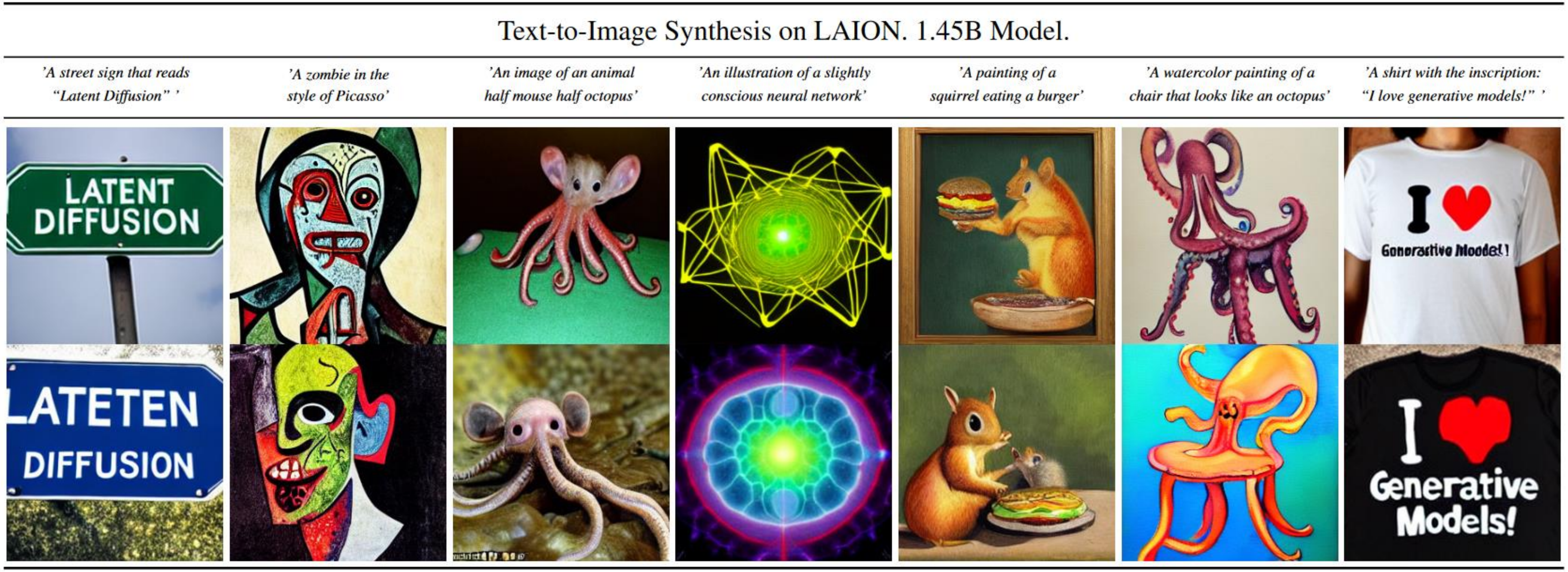
Latent Diffusion Models with Conditioning



The switch is for multi-modality:
if the conditioning element is a class or text, use cross-attention,
if the input is an image, use concatenation

[Image from Rombach et al., 2022 - https://arxiv.org/pdf/2112.10752]

Latent Diffusion Models with Conditioning



Text-to-Image Synthesis on LAION. 1.45B Model.

'A street sign that reads “Latent Diffusion” '	'A zombie in the style of Picasso'	'An image of an animal half mouse half octopus'	'An illustration of a slightly conscious neural network'	'A painting of a squirrel eating a burger'	'A watercolor painting of a chair that looks like an octopus'	'A shirt with the inscription: “I love generative models!” '

[Image from Rombach et al., 2022 - https://arxiv.org/pdf/2112.10752]

Links

https://poloclub.github.io/diffusion-explainer/

https://blog.marvik.ai/2023/11/28/an-introduction-to-diffusion-models-and-stable-diffusion/

https://theaisummer.com/diffusion-models/

https://learnopencv.com/denoising-diffusion-probabilistic-models/

https://www.assemblyai.com/blog/diffusion-models-for-machine-learning-introduction/

https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

https://www.superannotate.com/blog/diffusion-models

https://encord.com/blog/diffusion-models/