# Diffusion Model

Diffusion models are **probabilistic models** that define a distribution $p(z)$ over a latent variable $z$ , i.e. a nonlinear mapping from latent variables $z$ to input data $x$.

**IDEA:**

1. Corrupt a training image with a multi-step noise process
2. Get a corrupted image which has a standard Gaussian distribution (independently from the starting image $x$)
3. Use a Neural Network to invert this process (for each noise step)
4. After training we can generate new images starting from a sample from a standard Gaussian

Main similarities with other generative models:

- like Normalizing Flow they define nonlinear mapping between variables of the same dimensions $\mathbf{z}$ and $\mathbf{x}$ (differently from variational autoencoders and GANs)
- like Variational Autoencoder they approximate data likelihood using an ELBO (likelihood lower-bound) based on an encoder

Main differences with other generative models:

- the mapping in the encoding part is $\mathbf{x} \rightarrow \mathbf{z}$, while in normalizing flows is $\mathbf{z} \rightarrow \mathbf{x}$
- the *encoder* is predetermined (is the noise process), while in VAEs the encoder is parametrized and trainable
- the goal is to only learn the decoder (inverse of the noise process)

# Forward Encoder

**Setup:**

- $x \rightarrow$ training image
- $z_t \rightarrow t$-th noise-corrupted/latent variable
- $\epsilon_t \sim \mathcal{N}(\epsilon_t|\mathbf{0}, \mathbf{I}) \rightarrow$ additive gaussian noise for the $t$-th step
- $\beta_t < 1 \rightarrow$ variance of noise distribution
- the **Markov Chain** (forward) noise process:

$$\begin{cases} \mathbf{z}_1 = \sqrt{1-\beta_1}\mathbf{x} + \sqrt{\beta_1}\boldsymbol{\epsilon}_1 & \iff q(\mathbf{z}_1|\mathbf{x}) = \mathcal{N}(\mathbf{z}_1|\sqrt{1-\beta_1}\mathbf{x}, \beta_1\mathbf{I}) \\ . \\ . \\ . \\ \mathbf{z}_t = \sqrt{1-\beta_t}\mathbf{x} + \sqrt{\beta_t}\boldsymbol{\epsilon}_t & \iff q(\mathbf{z}_t|\mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t|\sqrt{1-\beta_t}\mathbf{x}, \beta_t\mathbf{I}) \\ . \\ . \\ . \\ \mathbf{z}_t = \sqrt{1-\beta_T}\mathbf{x} + \sqrt{\beta_T}\boldsymbol{\epsilon}_T & \iff q(\mathbf{z}_T|\mathbf{x}) = \mathcal{N}(\mathbf{z}_T|\mathbf{0}, \mathbf{I}) \end{cases}$$

**⚠ Obs 1**

Convergence to the standard Gaussian noise $\iff$ correct choice $\sqrt{1-\beta_t}$ and $\sqrt{\beta_t}$ that ensure:

- the mean of $q(\mathbf{z}_t|\mathbf{z}_{t-1})$ is closer to zero than the mean of $q(\mathbf{z}_{t-1}|\mathbf{z}_{t-2})$
- the variance of $q(\mathbf{z}_t|\mathbf{z}_{t-1})$ is closer to $I$ than the variance of $q(\mathbf{z}_{t-1}|\mathbf{z}_{t-2})$

**⚠ Obs 2:**

Final distribution $q(\mathbf{z}_T|\mathbf{x}) = \mathcal{N}(\mathbf{z}_T|\mathbf{0}, \mathbf{I})$ that is a standard Gaussian and doesn't depend neither on $\mathbf{z}_{T-1}$, nor on $x$ ... for the proof see the section on *diffusion kernel*

**⚠ Obs 3:**

We set $\beta_1 < \beta_2 < \ldots < \beta_T$

## Diffusion Kernel

1. Start from the joint distribution of latent vars.:

$$q(\mathbf{z}_1, \ldots, \mathbf{z}_t|\mathbf{x}) = q(\mathbf{z}_1|\mathbf{x}) \prod_{\tau=2}^{t} q(\mathbf{z}_\tau|\mathbf{z}_{\tau-1}).$$

2. Obtain the **diffusion kernel** by marginalizing (integrating) over intermediate vars. $\mathbf{z}_1, \ldots, \mathbf{z}_{t-1}$:

$$q(\mathbf{z}_t|\mathbf{x}) = \mathcal{N}(\mathbf{z}_t|\sqrt{\alpha_t}\mathbf{x}, (1-\alpha_t)\mathbf{I}) \quad \text{, where } \alpha_t = \prod_{\tau=1}^{t}(1-\beta_\tau)$$

3. This way we get dependence only on $x$:

$$\mathbf{z}_t = \sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\epsilon_t$$

4. After $T \to \infty$ steps the image becomes Gaussian noise, no more depending on $\mathbf{x}$:

$$q(\mathbf{z}_T) = \mathcal{N}(\mathbf{z}_T|\mathbf{0}, \mathbf{I})$$

## Conditional reverse distributions before introducing decoding

**Goal:** learn to undo the noise process $\implies$ find the reverse process and apply it

1. Consider the reverse of $q(\mathbf{z}_t|\mathbf{z}_{t-1})$ and use the **Bayes' theorem**:

$$q(\mathbf{z}_{t-1}|\mathbf{z}_t) = \frac{q(\mathbf{z}_t|\mathbf{z}_{t-1})q(\mathbf{z}_{t-1})}{q(\mathbf{z}_t)}$$

2. Try to **marginalize** (but it is intractable since we don't know $p(x)$ i.e., distribution of noiseless/starting image):

$$q(\mathbf{z}_{t-1}) = \int q(\mathbf{z}_{t-1}|\mathbf{x})p(\mathbf{x})\mathrm{d}\mathbf{x}$$

3. Consider instead the conditional (on $x$) version of the reverse distribution:

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

4. Consider that:

   **a)** by the **Markov property** of the forward process:

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

   **b)** by definition of diffusion kernel

$$\to q(\mathbf{z}_{t-1}|\mathbf{x}) = \mathcal{N}(\mathbf{z}_{t-1}|\sqrt{\alpha_{t-1}}\mathbf{x}, (1-\alpha_{t-1})\mathbf{I})$$

   **c)** ignore $q(\mathbf{z}_t|\mathbf{x})$ since not dependent on $\mathbf{z}_{t-1}$

   **d)** set:

$$\begin{cases} \mathbf{m}_t(\mathbf{x}, \mathbf{z}_t) = \dfrac{(1-\alpha_{t-1})\sqrt{1-\beta_t}\mathbf{z}_t + \sqrt{\alpha_{t-1}}\beta_t\mathbf{x}}{1-\alpha_t} \\ \sigma_t^2 = \dfrac{\beta_t(1-\alpha_{t-1})}{1-\alpha_t} \end{cases}$$

**e)** finally use the technique of "completing the square":

$$q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) = \mathcal{N}\left(\mathbf{z}_{t-1}|\mathbf{m}_t(\mathbf{x}, \mathbf{z}_t), \sigma_t^2\mathbf{I}\right)$$

# Reverse Decoder

**Goal:**

- forward encoder $\rightarrow$ sequence of Gaussians $q(\mathbf{z}_t|\mathbf{z}_{t-1})$
- reverse decoder $\rightarrow$ invert the encoding process getting a sequence of reversed conditionals $q(\mathbf{z}_{t-1}|\mathbf{z}_t)$

**Problem:**
The above conditionals are intractable distributions

**Solution:**

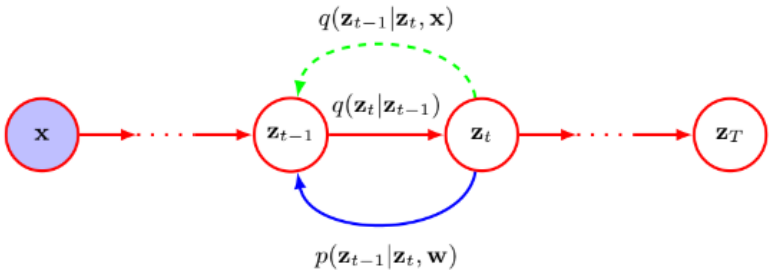
- Learn an approximation of the reverse conditionals by learning a distribution $p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})$ with a Neural Network (parameterized by weights $\mathbf{w}$)
- In practice we model the reverse process with another Gaussian (look at `info` box below to understand why the gaussian is a plausible model):

$$p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w}) = \mathcal{N}(\mathbf{z}_{t-1}|\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t), \beta_t\mathbf{I})$$

- where $\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t)$ is a NN with params. $\mathbf{w}$

**The Neural Network:**

- takes the step index $t$ as input $\rightarrow$ s.t. it accounts for the specific step variance $\beta_t$ $\rightarrow$ s.t. we use a single NN to invert all the steps of the Markov chain (instead of a different NN for each step)
- NN output should have same dimensionality of the input $\rightarrow$ **U-Net** is a good choice




\TODO

> ⓘ **Why the Gaussian is a good model for the reverse distribution $p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})$?**
>
> Look at the step in the section `Conditional distributions for decoding` of this notes and consider the example at pag. 586 @Bishop-Deep Learning.

## Reverse denoising process

The overall denoising process is also this time a Markov Chain:

$$p(\mathbf{x}, \mathbf{z}_1, \dots, \mathbf{z}_T|\mathbf{w}) = p(\mathbf{z}_T) \left\{ \prod_{t=2}^{T} p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w}) \right\} p(\mathbf{x}|\mathbf{z}_1, \mathbf{w})$$

> ⚠ **Assumption**
>
> We assume $p(\mathbf{z}_T) = q(\mathbf{z}_T) = \mathcal{N}(\mathbf{z}_T|0, \mathbf{I})$

> ϟ **Generation**
>
> Once the model is trained, sampling (generating) an image is easy:
>
> 1. sample from a simple Gaussian $p(\mathbf{z}_t)$
> 2. sample sequentially from $p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})$
> 3. finally sample from $p(\mathbf{x}|\mathbf{z}_1, \mathbf{w})$ to obtain $x$

## Training the Decoder

**Goal:**
We have to choose the objective function to optimize for training.

**Problem:**

We cannot use the classical data likelihood function, since the integral below would be difficult to solve (complex NN functions):

$$p(\mathbf{x}|\mathbf{w}) = \int \cdots \int p(\mathbf{x}, \mathbf{z}_1, \ldots, \mathbf{z}_T|\mathbf{w}) \mathrm{d}\mathbf{z}_1 \ldots \mathrm{d}\mathbf{z}_T$$

**Solution:**

Use a lower-bound of the log-likelihood as objective function i.e., the **evidence lower-bound (ELBO)**. Here we provide the so-called *variational lower-bound* for any choice of $q(\mathbf{z})$:

$$\mathcal{L}(\mathbf{w}) = \int q(\mathbf{z}) \ln \left\{ \frac{p(\mathbf{x}, \mathbf{z}|\mathbf{w})}{q(\mathbf{z})} \right\} \mathrm{d}\mathbf{z}$$

**Why is it a lower-bound of the log-likelihhod?**

For any distribution $q(\mathbf{z})$ the following relation always holds:

$$\ln p(\mathbf{x}|\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \mathrm{KL}\left(q(\mathbf{z}) \| p(\mathbf{z}|\mathbf{x}, \mathbf{w})\right)$$

where $\mathrm{KL}\left(\cdot\|\cdot\right) \geqslant 0 \implies \ln p(\mathbf{x}|\mathbf{w}) \geq \mathcal{L}(\mathbf{w})$.

**⚠ Obs:**

Optimizing $q(\mathbf{z})$ encourages the bound to be tight
$\implies p(\mathbf{x}, \mathbf{z}|\mathbf{w}) \approx$ to that of max likelihood

## ELBO parameterization

1. Defined the Expected value as:

$$\mathbb{E}_q\left[\cdot\right] \equiv \int \cdots \int q(\mathbf{z}_1|\mathbf{x}) \prod_{t=2}^{T} q(\mathbf{z}_t|\mathbf{z}_{t-1}) \left[\cdot\right] \mathrm{d}\mathbf{z}_1 \ldots \mathrm{d}\mathbf{z}_T$$

2. We get:

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_q \left[ \ln \frac{p(\mathbf{z}_T) \left\{ \prod_{t=2}^{T} p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w}) \right\} p(\mathbf{x}|\mathbf{z}_1, \mathbf{w})}{q(\mathbf{z}_1|\mathbf{x}) \prod_{t=2}^{T} q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x})} \right]$$

$$= \mathbb{E}_q \left[ \underbrace{\ln p(\mathbf{z}_T)}_{\text{indep. from w}} + \underbrace{\sum_{t=2}^{T} \ln \frac{p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})}{q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x})}}_{\text{consistency terms}} \underbrace{- \ln q(\mathbf{z}_1|\mathbf{x})}_{\text{indep. from w}} + \underbrace{\ln p(\mathbf{x}|\mathbf{z}_1, \mathbf{w})}_{\text{reconstruction term}} \right]$$

3. *Reconstruction term* can be evaluated by **Monte Carlo** estimate:

$$\mathbb{E}_q \left[ \ln p(\mathbf{x}|\mathbf{z}_1, \mathbf{w}) \right] \simeq \sum_{l=1}^{L} \ln p(\mathbf{x}|\mathbf{z}_1^{(l)}, \mathbf{w})$$

4. For the *consistency terms* we could sample from $q(\mathbf{z}_{t-1}|\mathbf{x})$ (Gaussian distro) and obtain the sample $\mathbf{z}_t$ by using:

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

> ⚠ **Problem:**
>
> The approach at `point 4.` → pairs of sampled values give very noisy estimates → high variance → large number of samples is required

## ELBO reparameterization

**Goal:**
Write the ELBO in terms of KL divergences → expressible in closed form

1. Staring from the single consistency term, the denominator of each term can be rewritten with Bayes' theorem:

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

2. We can rewrite the entire consistency term as:

$$\ln \frac{p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})}{q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x})} = \ln \frac{p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})}{q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x})} + \underbrace{\ln \frac{q(\mathbf{z}_{t-1}|\mathbf{x})}{q(\mathbf{z}_t|\mathbf{x})}}_{\text{indep. from w}}$$

3. Plugging-in the above considerations we get:

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_q \left[ \underbrace{\sum_{t=2}^{T} \ln \frac{p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})}{q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x})}}_{\text{consistency terms}} + \underbrace{\ln p(\mathbf{x}|\mathbf{z}_1, \mathbf{w})}_{\text{reconstruction term}} \right]$$

4. When applying the expectation as an integral we get that:
   - the reconstruction term only depends on $\mathbf{z}_1 \rightarrow$ all conditional distributions integrate to 1 leaving only the integral over $\mathbf{z}_1$.
   - for the consistency terms:
     - we pull out the sum (linearity of the integral) and we get the sum of $T-2$ integrals
     - each integral involves only two adjacent latent variables $\mathbf{z}_{t-1}$ and $\mathbf{z}_t$
5. In the integral form, we write the above expected value as:

$$\mathcal{L}(\mathbf{w}) = \underbrace{\int q(\mathbf{z}_1|\mathbf{x}) \ln p(\mathbf{x}|\mathbf{z}_1, \mathbf{w}) \, \mathrm{d}\mathbf{z}_1}_{\text{reconstruction term}} - \underbrace{\sum_{t=2}^{T} \int \mathrm{KL}(q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) \| p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})) q(\mathbf{z}_t|\mathbf{x}) \, \mathrm{d}\mathbf{z}_t}_{\text{consistency terms}}$$

**In practice**

1. reconstruction term $\implies$ can be trained by using the **MC sampling estimate**
2. consistency terms $\implies$ expressed in closed form for gaussian distributions (closed-form **KL divergence for Gaussian distributions**), i.e.:

$$q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) = \mathcal{N}\left(\mathbf{z}_{t-1}|\mathbf{m}_t(\mathbf{x}, \mathbf{z}_t), \sigma_t^2 \mathbf{I}\right)$$

$$p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w}) = \mathcal{N}(\mathbf{z}_{t-1}|\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t), \beta_t \mathbf{I})$$

$$\implies \mathrm{KL}(q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) \| p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})) = \frac{1}{2\beta_t} \|\mathbf{m}_t(\mathbf{x}, \mathbf{z}_t) - \boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t)\|^2 + \text{const}$$

**Obs**

Considering the previous box, we minimize a **squared error** (between the mean of the true distribution and the mean of the learned distribution) because there is a minus sign in front of the KL divergence terms in the ELBO.

## Predicting the noise instead of the denoised image

**Idea**:
A simple modification to the neural network used at each step of the denoising process (decoder)(usually a U-Net) seems to improve the performances of a diffusion model:

- before $\implies$ the NN was used to predict the denoised image at each step of the Markov chain
- now $\implies$ it predicts the **total noise** that was added to the original image at that step

Now, at each decoding step $t$ the neural network predicts:

$$\mathbf{x} = \frac{1}{\sqrt{\alpha_t}}\mathbf{z}_t - \frac{\sqrt{1-\alpha_t}}{\sqrt{\alpha_t}}\epsilon_t$$

**Implications**:

1. We newly define the function for the mean of posterior conditional distribution $q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x})$:

$$\mathbf{m}_t(\mathbf{x}, \mathbf{z}_t) = \frac{1}{\sqrt{1-\beta_t}}\left\{\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{\epsilon}_t\right\}$$

2. We can write the new neural network as a function $\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t)$ related to the old neural network $\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t)$ in the following way:

$$\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) = \frac{1}{\sqrt{1-\beta_t}}\left\{\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t)\right\}$$

3. Now is also possible to derive the new KL divergence:

$$\begin{aligned}
&\mathrm{KL}(q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x})\|p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})) \\
&= \frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)}\left\|\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t) - \boldsymbol{\epsilon}_t\right\|^2 + \text{const} \\
&= \frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)}\left\|\mathbf{g}(\sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\epsilon_t, \mathbf{w}, t) - \epsilon_t\right\|^2 + \text{const}
\end{aligned}$$

4. From two chapters above we know the approximation of the reconstruction term in the ELBO and, having the new NN, the single term in the sum is equal to the KL divergence above (for $t = 1$):

$$\begin{aligned}
\ln p(\mathbf{x}|\mathbf{z}_1, \mathbf{w}) &= -\frac{1}{2\beta_1}\left\|\mathbf{x} - \boldsymbol{\mu}(\mathbf{z}_1, \mathbf{w}, 1)\right\|^2 + \text{const} \\
&= -\frac{1}{2(1-\beta_t)}\left\|\mathbf{g}(\mathbf{z}_1, \mathbf{w}, 1) - \boldsymbol{\epsilon}_1\right\|^2 + \text{const}
\end{aligned}$$

5. Omitting the coefficient $\frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)}$ is now possible to get the new ELBO for this setup:

$$\mathcal{L}(\mathbf{w}) = -\sum_{t=1}^{T} \left\| \mathbf{g}(\sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t, \mathbf{w}, t) - \boldsymbol{\epsilon}_t \right\|^2$$

**Interpretation of the new loss**:
Given step $t$ in the Markov chain and training data point $x$, we:

1. sample a noise vector $\epsilon_t$
2. create the corresponding noisy latent vector $z_t$ for that step
3. the loss function is then the squared difference between the predicted noise and the actual noise.

> ⚠ **Mini-batching over time**
>
> For each data point we randomly select a step $t$ along the Markov chain, rather than evaluate the error for every term in the summation over $t$ (sort of stochastic gradient descent where we do mini-batching also over time)!

# Generation

**Idea**:
Once the network has been trained for the decoding process, we can generate new samples in the data space.

**Procedure**:

1. Sample from the gaussian distribution $p(\mathbf{z}_T)$
2. Denoise successively through each step of the Markov chain:
   1. take a denoised sample $\mathbf{z}_t$ at timestep $t$
   2. generate a sample $\mathbf{z}_{t-1}$ in three steps:
   1. evaluate the output of the neural network given by
   2. evaluate $\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t)$ using:

$$\mathrm{KL}(q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) \| p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w})) = \frac{\beta_t}{2(1-\alpha_t)(1-\beta_t)} \left\| \mathbf{g}(\sqrt{\alpha_t}\mathbf{x} + \sqrt{1-\alpha_t}\epsilon_t, \mathbf{w}, t) - \epsilon_t \right\|^2$$

$$+ \text{const}$$

   3. generate a sample $\mathbf{z}_{t-1}$ from $p(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{w}) = \mathcal{N}(\mathbf{z}_{t-1}|\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t), \beta_t \mathbf{I})$ s.t.:

$$\mathbf{z}_{t-1} = \boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) + \sqrt{\beta_t}\boldsymbol{\epsilon}$$

### Speed-up generation

Main issue when generating data with diffusion models is represented by the multiple inference sequential passes through the trained network.
Possible solutions:

- convert the denoising process to a differential equation over continuous time and then use alternative efficient discretization methods to solve the equation efficiently
- relax the Markovian assumption on the noise process while retaining the same objective function for training $\implies$ Denoising Diffusion Implicit Model.

## Training vs generation summary

**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`
  $\mathcal{L}(\mathbf{w}) \leftarrow \|\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t) - \boldsymbol{\epsilon}\|^2$ `// Compute loss term`
  Take optimizer step
**until** converged
**return** $\mathbf{w}$

**Algorithm 20.2:** Sampling from a denoising diffusion probabilistic model

**Input:** Trained denoising network $\mathbf{g}(\mathbf{z}, \mathbf{w}, t)$

Noise schedule $\{\beta_1, \ldots, \beta_T\}$

**Output:** Sample vector $\mathbf{x}$ in data space

---

$\mathbf{z}_T \sim \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ `// Sample from final latent space`

**for** $t \in T, \ldots, 2$ **do**

$\alpha_t \leftarrow \prod_{\tau=1}^{t}(1-\beta_\tau)$ `// Calculate alpha`

`// Evaluate network output`

$\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) \leftarrow \frac{1}{\sqrt{1-\beta_t}}\left\{\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t)\right\}$

$\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ `// Sample a noise vector`

$\mathbf{z}_{t-1} \leftarrow \boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) + \sqrt{\beta_t}\boldsymbol{\epsilon}$ `// Add scaled noise`

**end for**

$\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}}\left\{\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}}\mathbf{g}(\mathbf{z}_1, \mathbf{w}, t)\right\}$ `// Final denoising step`

**return** $\mathbf{x}$