

# Generative Adversarial Networks, Flow Matching, Diffusion Models

Milan Straka

 May 6, 2025

# Generative Models

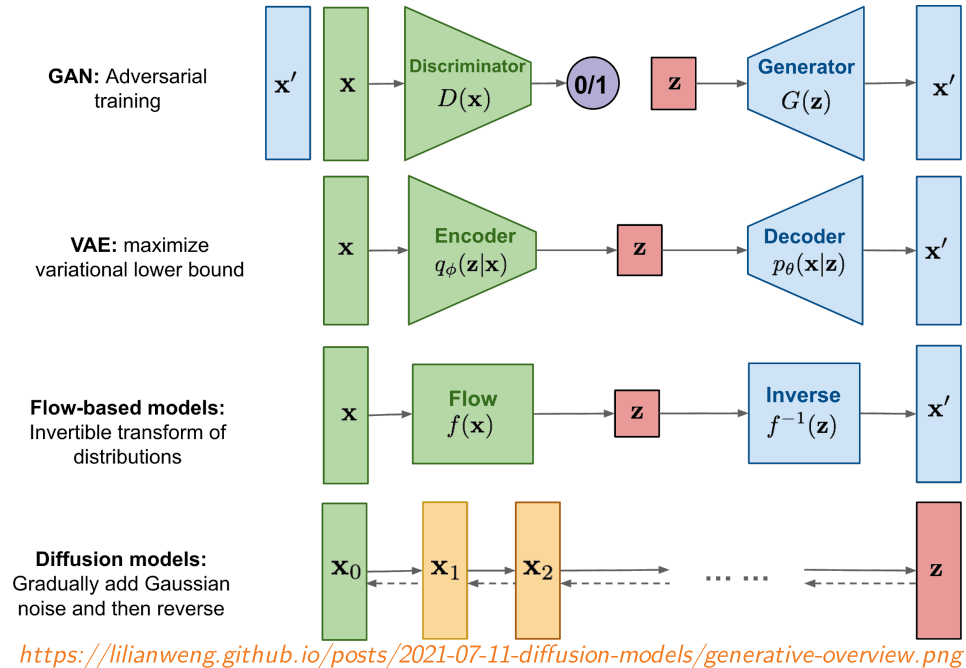
# Generative Models

There are several approaches how to represent a probability distribution  $P(\mathbf{x})$ . **Likelihood-based models** represent the probability density function directly, often using an unnormalized probabilistic model (also called energy-based model; i.e., specifying a non-zero *score* or *density* or *logits*):

$$P_{\theta}(\mathbf{x}) = \frac{e^{f_{\theta}(\mathbf{x})}}{Z_{\theta}} \cdot \text{handwritten notes: } \text{fady tech fady, m\u00fc\u00dfe b\u00e4t nehmene...}$$

However, estimating the normalization constant  $Z_{\theta} = \int e^{f_{\theta}(\mathbf{x})} d\mathbf{x}$  is often intractable.

- We can compute  $Z_{\theta}$  by restricting the model architecture (sequence modeling, invertible networks in normalizing flows);
- we can only approximate it (using for example variational inference as in VAE);
- we can use **implicit generative models**, which avoid representing likelihood (like GANs).



# Generative Adversarial Networks



# Generative Adversarial Networks

We have a **generator**  $G(\mathbf{z}; \theta_g)$ , which given  $\mathbf{z} \sim P(\mathbf{z})$  generates data  $\mathbf{x}$ .

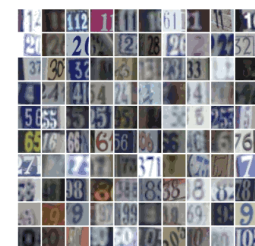
Then we have a **discriminator**  $D(\mathbf{x}; \theta_d)$ , which given data  $\mathbf{x}$  generates a probability whether  $\mathbf{x}$  comes from real data or is generated by a generator.

The discriminator and generator play the following game:

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

*maximizuj realny* *maximizuj fake*

Až jednou discriminator  
nebude umět rozlišit  
fake a real



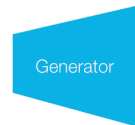
SVHN training data



sample  
(real) data



latent sample,  $\mathbf{z}$



sample  
(fake) data



hrají tuto hru,  
kdy se snaží odlišit  
mezi real a fake.

gradient posílá zpátky informaci podle čeho  
se změnil discriminator. Totiž pokud se generator  
poblížil toho, může zlepšovat.

[https://miro.medium.com/v2/1\\*-ucVYsbDnwa2NM-f5qm\\_Yg.png](https://miro.medium.com/v2/1*-ucVYsbDnwa2NM-f5qm_Yg.png)

# Generative Adversarial Networks

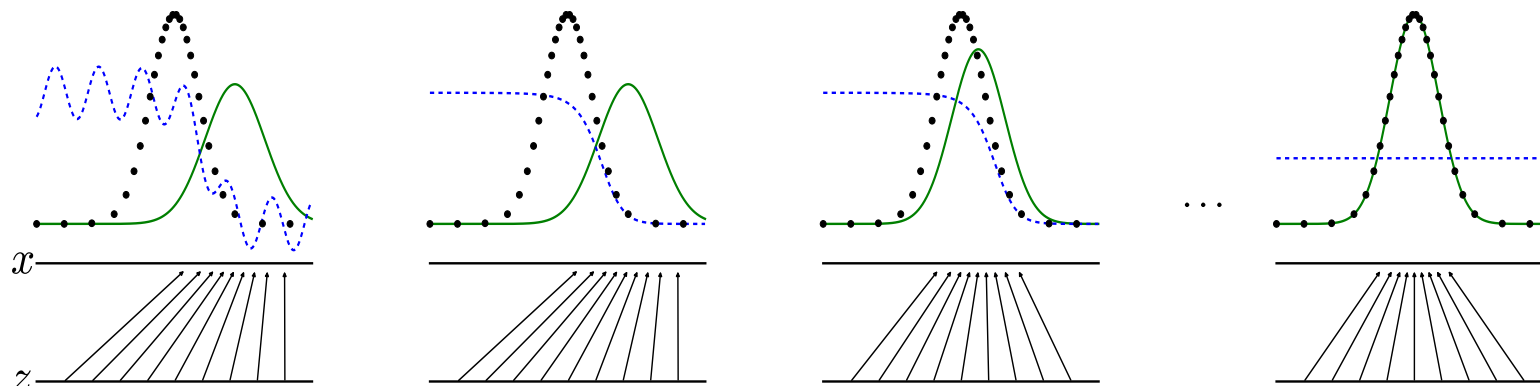


Figure 1 of "Generative Adversarial Nets", <https://arxiv.org/abs/1406.2661>

The generator and discriminator are alternately trained, the discriminator by

$$\arg \max_{\theta_d} \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

and the generator by

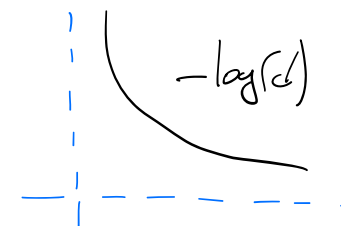
$$\arg \min_{\theta_g} \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

*→ na zničten budú hociže saturovať,  
prakticky  $\sigma(\text{false}) \rightarrow 0,0001$*

*→ ľahké mi to sigmoid nejstredí*

Basically, the discriminator acts as a trainable loss for the generator.

Because  $\log(1 - D(G(\mathbf{z})))$  can saturate at the beginning of the training, where the discriminator can easily distinguish real and generated samples, the generator can be trained by

$$\arg \min_{\theta_g} \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})} [-\log D(G(\mathbf{z}))]$$


instead, which results in the same fixed-point dynamics, but much stronger gradients early in learning.

On top of that, if you train the generator by using “real” as the gold label of the discriminator, you naturally get the above loss (which is the negative log likelihood, contrary to the original formulation).

---

**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  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
- Sample minibatch of  $m$  examples  $\{x^{(1)}, \dots, x^{(m)}\}$  from data generating distribution  $p_{\text{data}}(x)$ .
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D(x^{(i)}) + \log \left( 1 - D(G(z^{(i)})) \right) \right].$$

**end for**

- Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log \left( 1 - D(G(z^{(i)})) \right).$$

**end for**

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

---

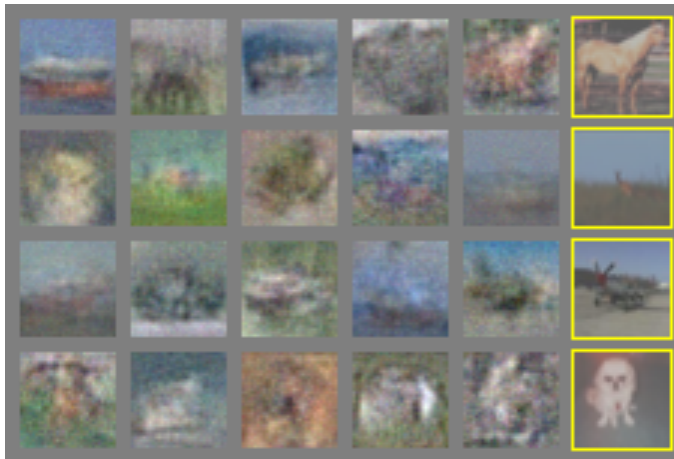
*Algorithm 1 of "Generative Adversarial Nets", <https://arxiv.org/abs/1406.2661>*



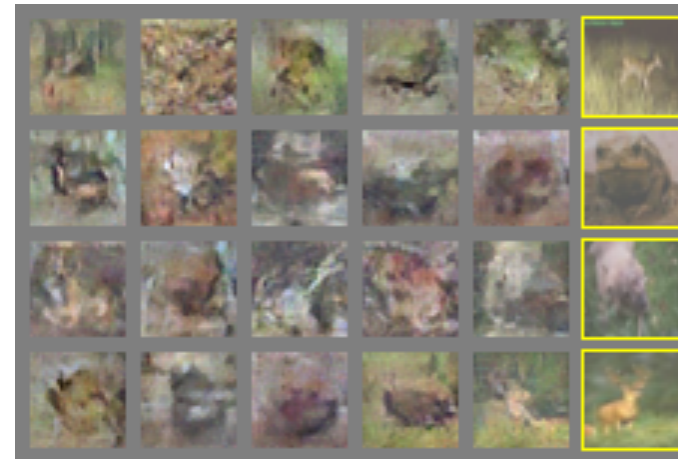
a)



b)



c)



d)

Figure 2 of "Generative Adversarial Nets", <https://arxiv.org/abs/1406.2661>

# Conditional GAN

Assuming our dataset is conditional, i.e., the individual examples are pairs  $(\mathbf{x}, y)$  with  $y$  being the image class, GANs can be easily extended to allow conditioning:

- the generator gets  $y$  as an additional input:  $G(\mathbf{z}, y)$ ,
- the discriminator also gets  $y$  as an additional input:  $D(\mathbf{x}, y)$ .

*- French generating speech from text, to condition by using text*

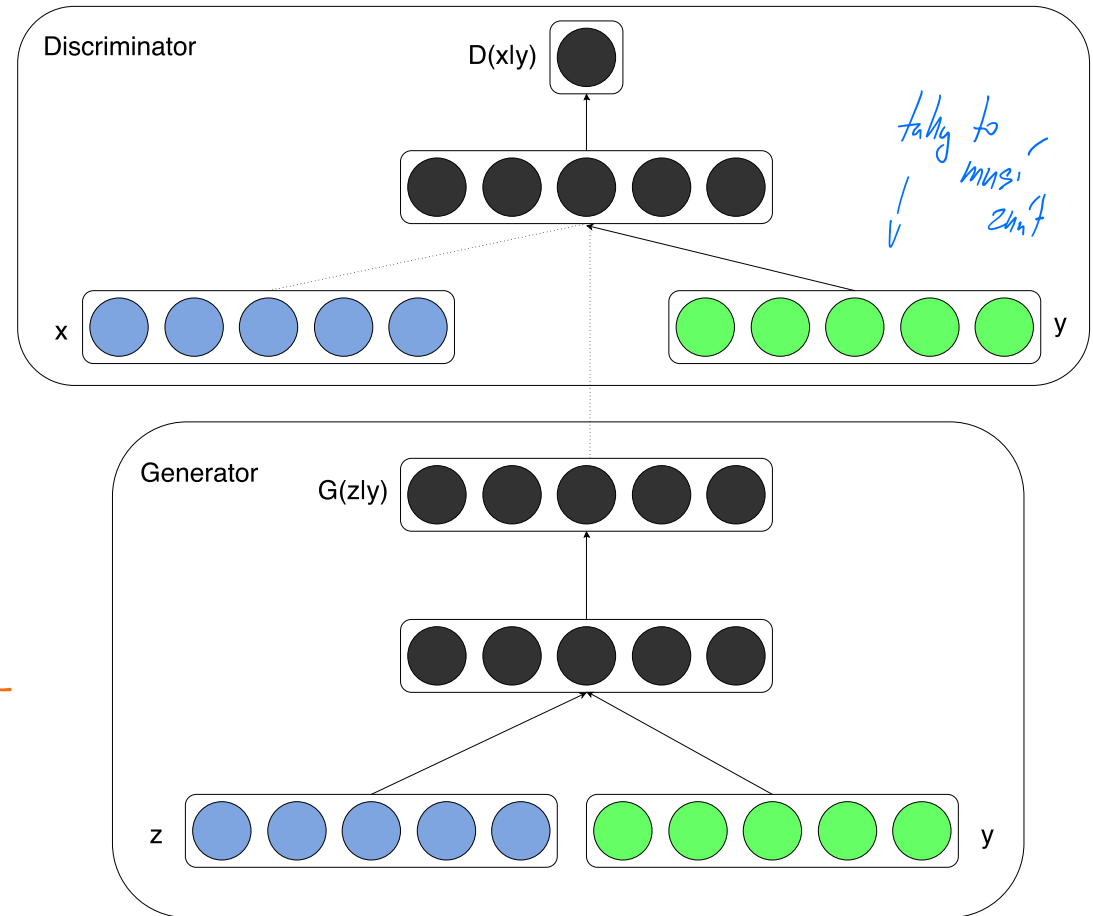
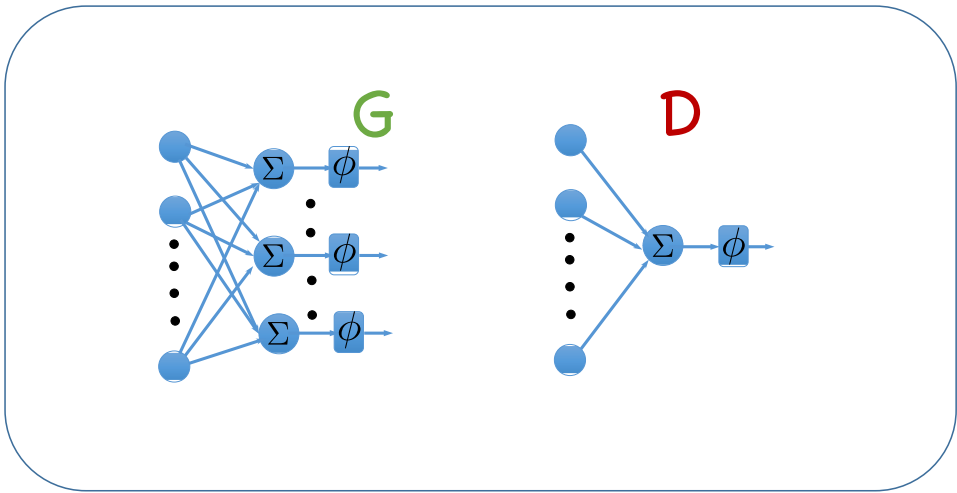


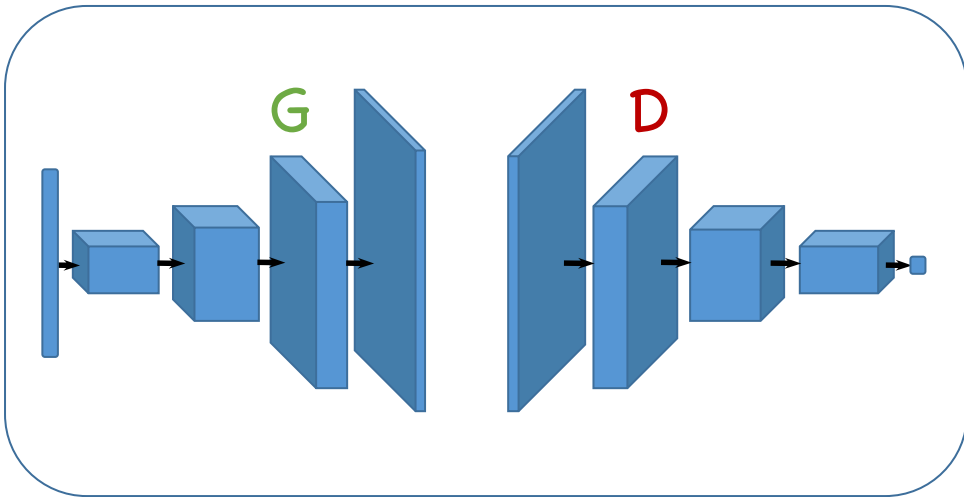
Figure 1 of "Conditional Generative Adversarial Nets", <https://arxiv.org/abs/1411.1784>

# Deep Convolutional GAN

In Deep Convolutional GAN, the discriminator is a convolutional network (with batch normalization) and the generator is also a convolutional network, utilizing transposed convolutions.



(a)



(c)

Figure 1 of "An Online Learning Approach to Generative Adversarial Networks", <https://arxiv.org/abs/1706.03269>

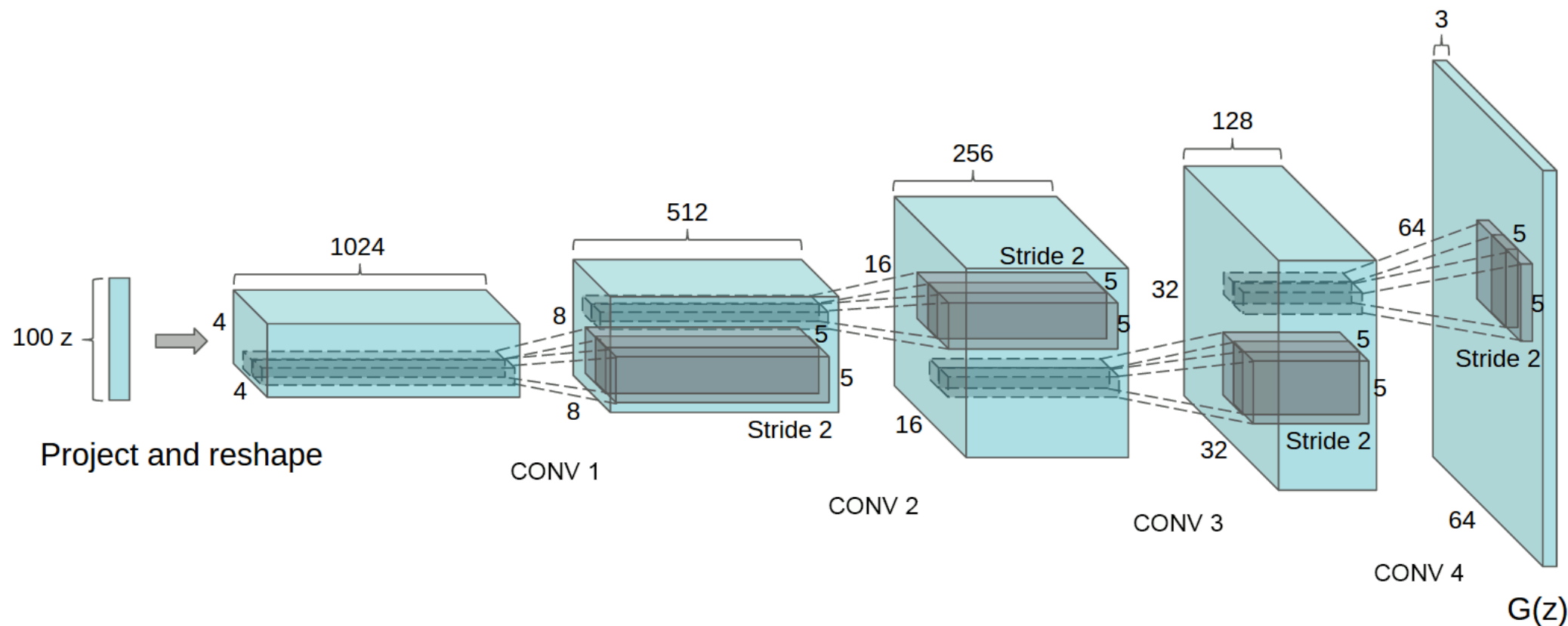


Figure 1 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", <https://arxiv.org/abs/1511.06434>





Figure 3 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", <https://arxiv.org/abs/1511.06434>



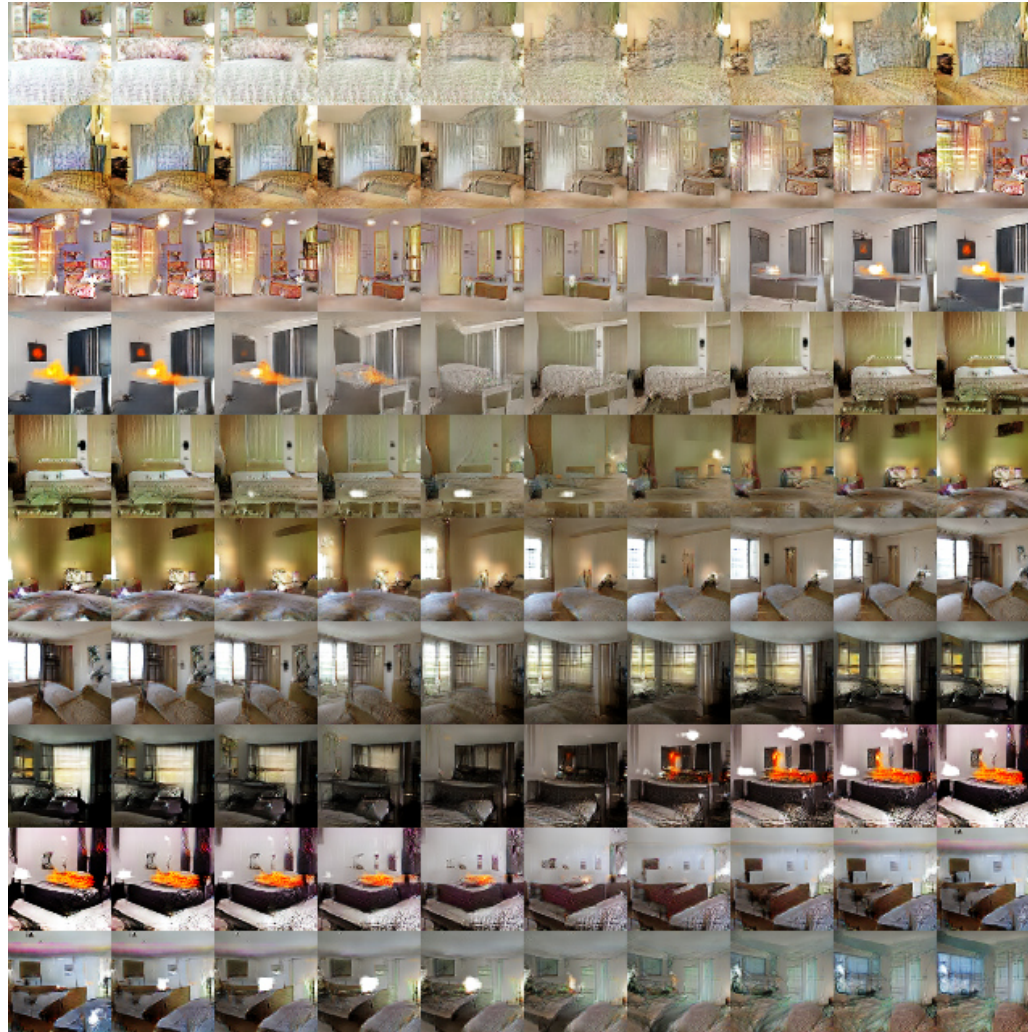


Figure 4 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", <https://arxiv.org/abs/1511.06434>

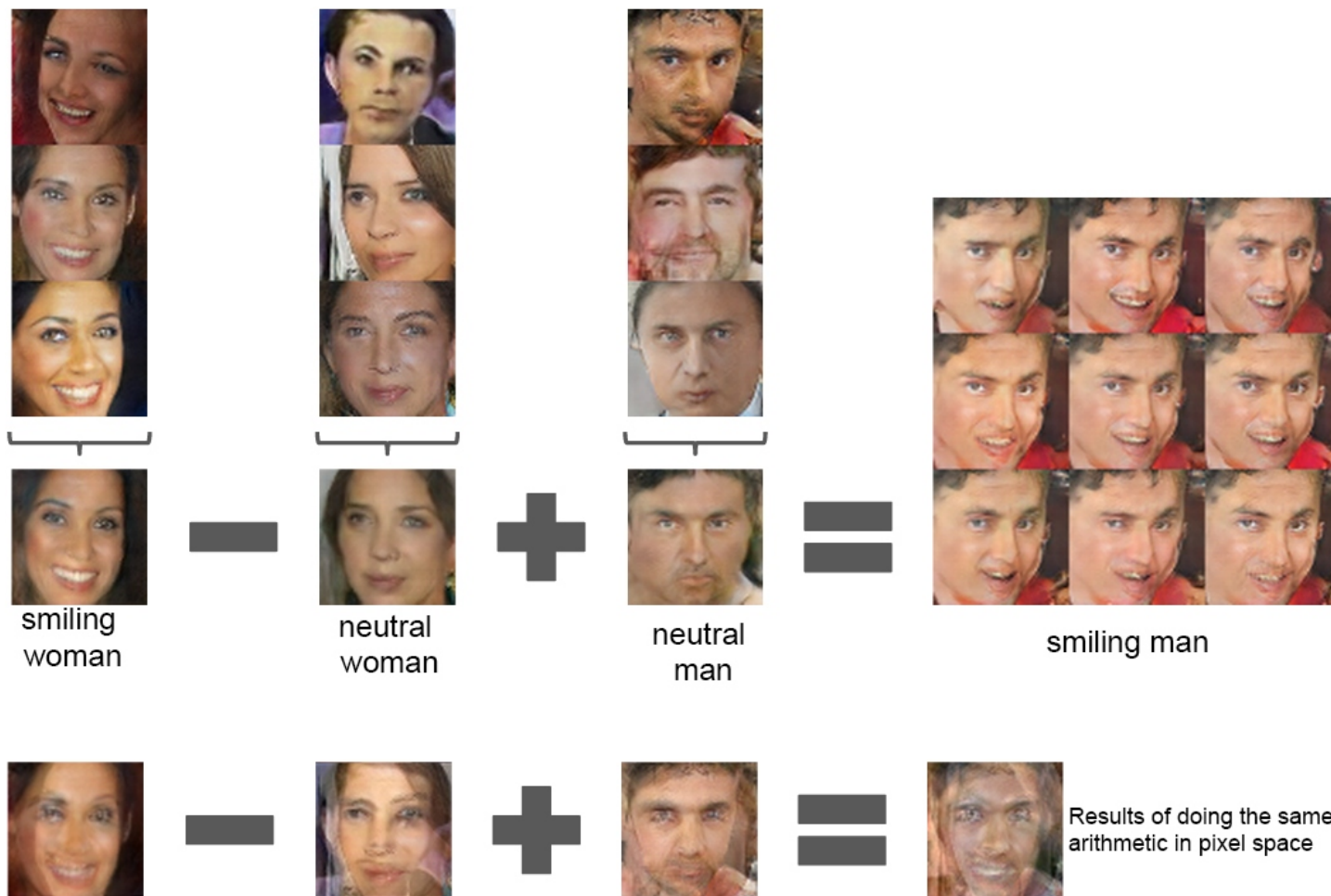


Figure 7 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", <https://arxiv.org/abs/1511.06434>

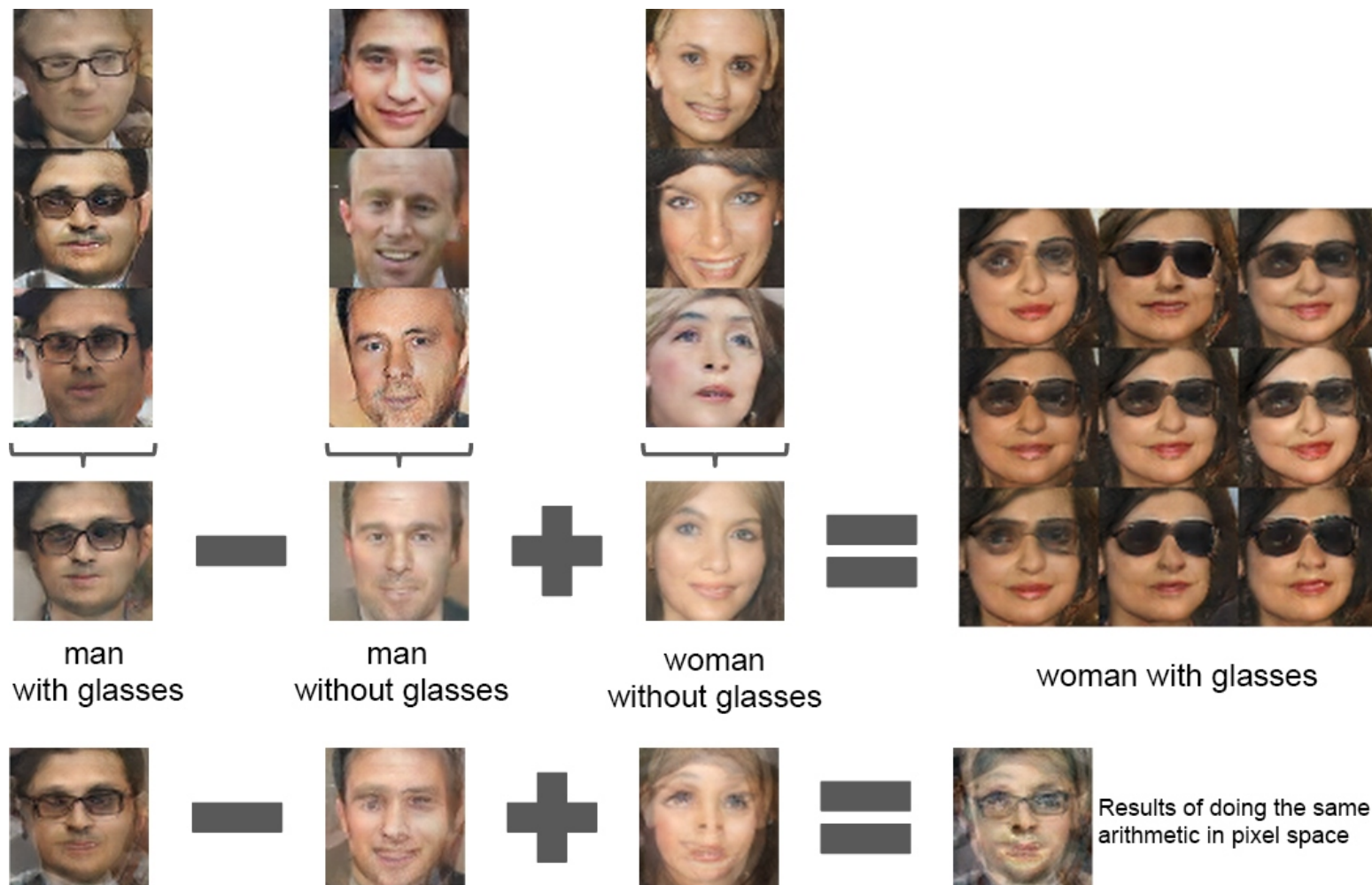


Figure 7 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", <https://arxiv.org/abs/1511.06434>





Figure 8 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", <https://arxiv.org/abs/1511.06434>

## GAN output in paper

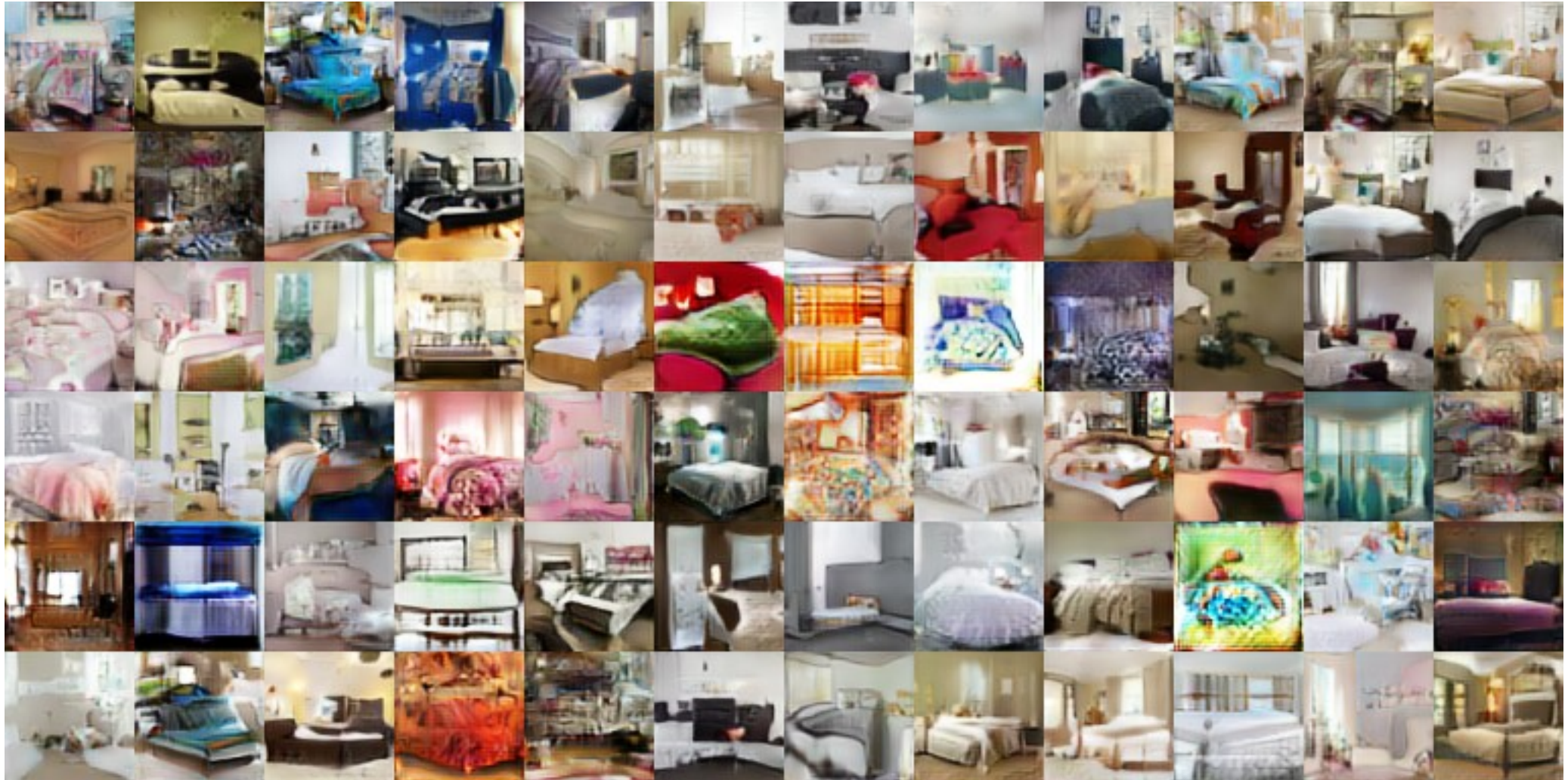


## Your GAN output



[https://miro.medium.com/max/1400/1\\*r8cuSlam5oHUERP01TCTxg.jpeg](https://miro.medium.com/max/1400/1*r8cuSlam5oHUERP01TCTxg.jpeg)





# GANs are Problematic to Train



# GANs are Problematic to Train

Unfortunately, alternating SGD steps are not guaranteed to reach even a local optimum of a minimax problem, consider the following one:

$$\min_x \max_y x \cdot y.$$

The update rules of  $x$  and  $y$  for learning rate  $\alpha$  are

$$\begin{aligned} x_{n+1} &= x - \alpha \cdot \frac{\nabla_x (x \cdot y)}{y} \\ y_{n+1} &= y + \alpha \cdot \frac{\nabla_y (x \cdot y)}{x} \end{aligned}$$

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & -\alpha \\ \alpha & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix}.$$

The update matrix is a rotation matrix multiplied by a constant  $\sqrt{1 + \alpha^2} > 1$

$$\begin{bmatrix} 1 & -\alpha \\ \alpha & 1 \end{bmatrix} = \sqrt{1 + \alpha^2} \cdot \begin{bmatrix} \cos \varphi & -\sin \varphi \\ \sin \varphi & \cos \varphi \end{bmatrix},$$

Führt to divergence!

so the SGD will not converge with arbitrarily small step size.

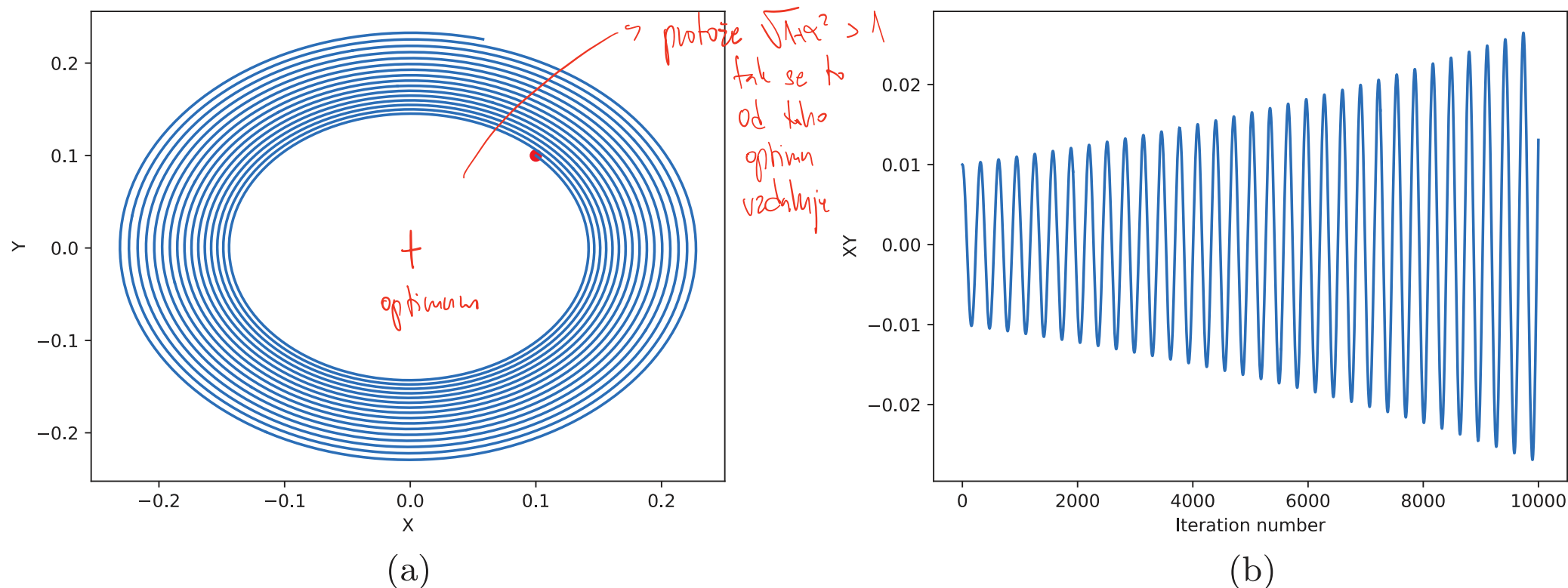


Fig. 1: Performance of gradient method with fixed step size for Example 2. (a) illustrates the choices of  $x$  and  $y$  as iteration processes, the red point  $(0.1, 0.1)$  is the initial value. (b) illustrates the value of  $xy$  as a function of iteration numbers.

Figure 1 of "Fictitious GAN: Training GANs with Historical Models", <https://arxiv.org/abs/1803.08647>

# GANs are Problematic to Train

— Zpravidla ten generátor se soustředí vždy jen na nějaký malý sektor, jakmile na to discriminator přijde, tak začne generovat jiný sektor

- GANs suffer from “mode collapse”

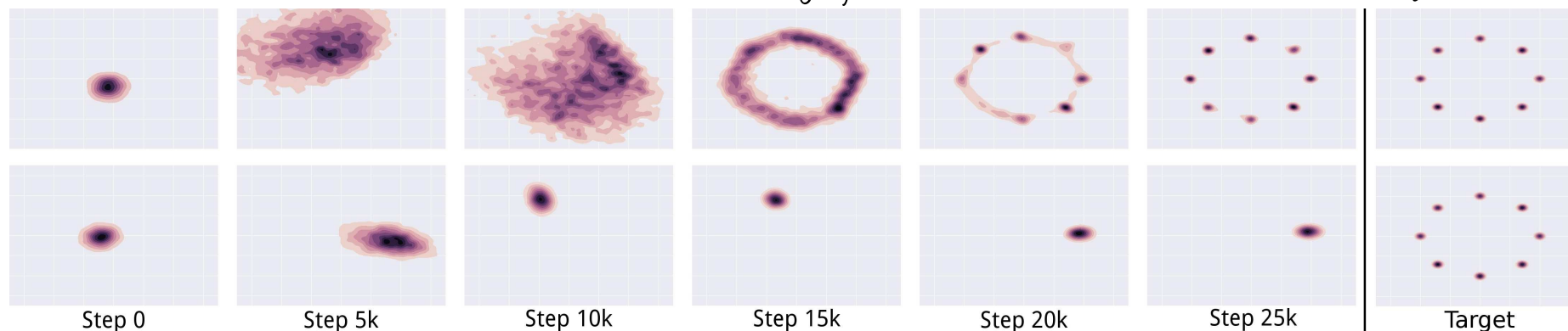


Figure 2 of "Unrolled Generative Adversarial Networks", <https://arxiv.org/abs/1611.02163>

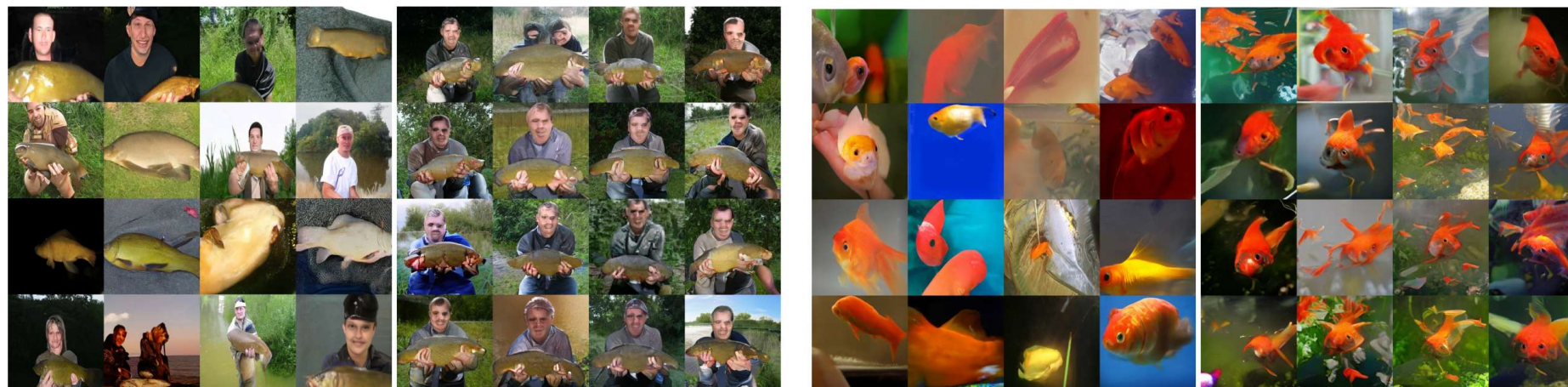


Figure 5 of "Generating Diverse High-Fidelity Images with VQ-VAE-2", <https://arxiv.org/abs/1906.00446>

# GANs are Problematic to Train

The training can be improved by various tricks:

- If the discriminator could see the whole batch, similar samples in it would be candidates for fake images.
  - Batch normalization helps a lot with this.
- Unrolling the discriminator update helps generator to consider not just the current discriminator, but also how the future versions would react to the generator outputs. (The discriminator training is unchanged.)

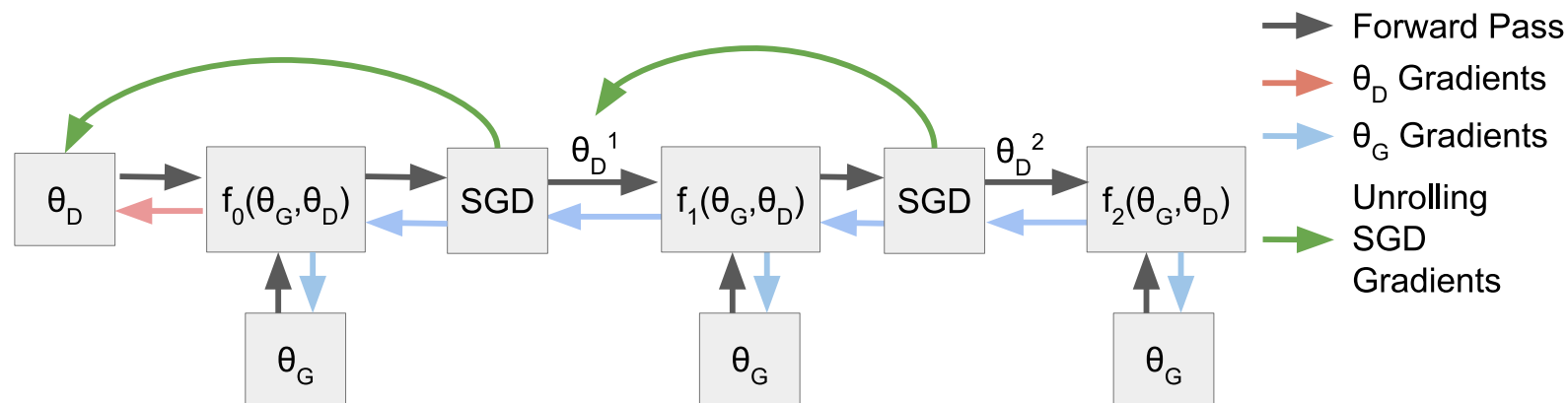


Figure 1 of "Unrolled Generative Adversarial Networks", <https://arxiv.org/abs/1611.02163>

- Many others, like Wasserstein GAN, spectral normalization, progressive growing, ...

## The Variational Autoencoders:

*to je proto, že nikde ani latentního prostoru není  
sdílení pro více ústupných obměn, takže zisková musí být rovnováha* ←

- are theoretically-pleasing;
- also provide an encoder, so apart from generation, they can be used as unsupervised feature extraction (the VAE encoder is used in various modeling architectures);
- the generated samples tend to be blurry, especially with  $L^1$  or  $L^2$  loss (because of the sampling used in the reconstruction; patch-based discriminator with perceptual loss helps).

## The Generative Adversarial Networks:

- offer high sample quality;
- are difficult to train and suffer from mode collapse.

In past few years, GANs saw a big development, improving the sample quality substantially. However, since 2019/2020, VAEs have shown remarkable progress (alleviating the blurriness issue by using perceptual loss and a 2D grid of latent variables), and are being used for generation too. Furthermore, additional approaches (normalizing flows, diffusion models) were also being explored, with diffusion models becoming the most promising approach since Q2 of 2021, surpassing both VAEs and GANs.

# Flow Matching



In the past years (since 2022), the dominant approach for generating images has been based on **diffusion models** (used by Stable Diffusion, DALL-E, ...).



[https://images.squarespace-cdn.com/content/v1/6213c340453c3f502425776e/0715034d-4044-4c55-9131-e4bfd6dd20ca/2\\_4x.png](https://images.squarespace-cdn.com/content/v1/6213c340453c3f502425776e/0715034d-4044-4c55-9131-e4bfd6dd20ca/2_4x.png)

The diffusion models are deeply connected to **score-based generative models**, which were developed independently, but are just a different perspective on the same model family.

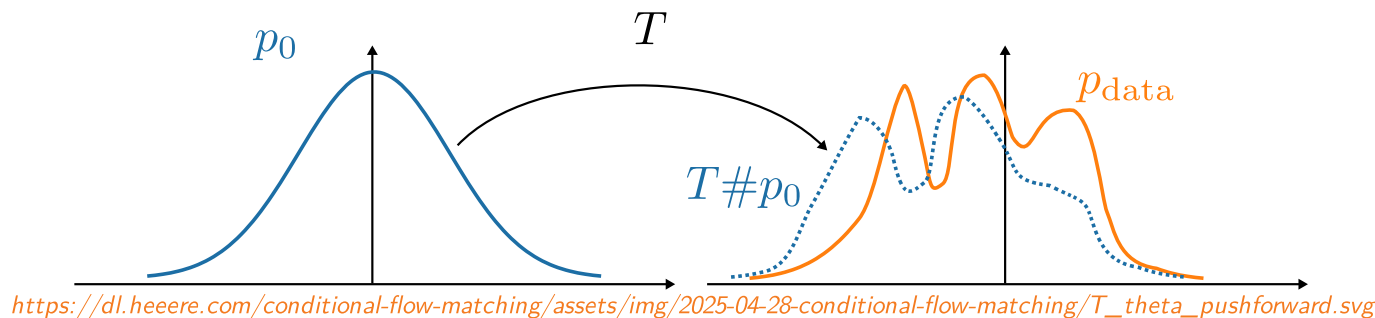
Recently, **conditional flow matching** was proposed as a generalization of both these approaches, and that is the method we will describe in most detail.

# Generative Modeling

The general framework of generative modeling assumes we have samples  $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N)}$  from the data generating distribution  $p_{\text{data}}$ , and the main challenges we should overcome are:

- provide fast sampling (diffusion models were originally not great here),
- generate high-quality samples (VAE struggles with this goal),
- properly cover the density of  $p_{\text{data}}$  (the main issue of GANs).

Modern approach to generative modeling is to start with a simple base distribution  $p_0$ , usually a standard Gaussian  $\mathcal{N}(\mathbf{0}, \mathbf{I})$ , and learn a mapping that transforms that distribution into  $p_{\text{data}}$ .



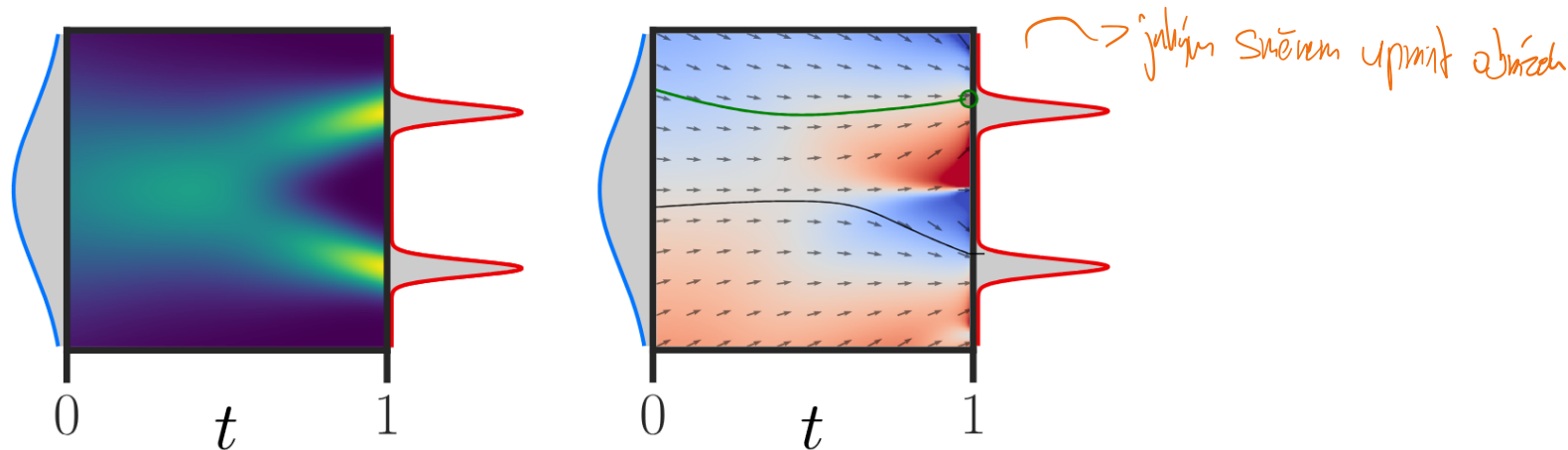
Sampling then can be performed by sampling from  $p_0$  and performing the transformation.



# Generating Images From Standard Normal Base Distribution







<https://dl.heeere.com/conditional-flow-matching/blog/conditional-flow-matching/>

The important concepts used by flow matching are:

- the **probability density path**  $p : [0, 1] \times \mathbb{R}^d \rightarrow \mathbb{R}_{>0}$ , which is a time-dependent probability density function, i.e.,  $\int p_t(\mathbf{x}) d\mathbf{x} = 1$ ;
  - this probability density path should transform the prior  $p_0$  into  $p_1 = p_{\text{data}}$
- a **time-dependent vector field**  $u : [0, 1] \times \mathbb{R}^d \rightarrow \mathbb{R}^d$ , which can be used to construct a **flow**  $\varphi : [0, 1] \times \mathbb{R}^d \rightarrow \mathbb{R}^d$  via an ordinary differential equation:

$$\frac{d}{dt}\varphi_t(\mathbf{x}) = u_t(\varphi_t(\mathbf{x})), \quad \varphi_0(\mathbf{x}) = \mathbf{x}.$$

# Flow Matching

Note that the solution of the flow ODE

$$\frac{d}{dt}\varphi_t(\mathbf{x}) = u_t(\varphi_t(\mathbf{x})), \quad \varphi_0(\mathbf{x}) = \mathbf{x}$$

is unique when  $u_t$  is Lipschitz continuous in  $\mathbf{x}$  and continuous in  $t$  (Picard–Lindelöf theorem).

Recall that in a residual network, we update the current value by adding the result of a residual block

$$\mathbf{h}_{t+1} = \mathbf{h}_t + f(\mathbf{h}_t; \boldsymbol{\theta}_t),$$

which we can also write as

$$\mathbf{h}_{t+1} - \mathbf{h}_t = f(\mathbf{h}_t; \boldsymbol{\theta}_t),$$

where we can interpret the residual block as a “discrete derivative”.

Therefore, the flow can be considered to be a continuous generalization of residual networks.

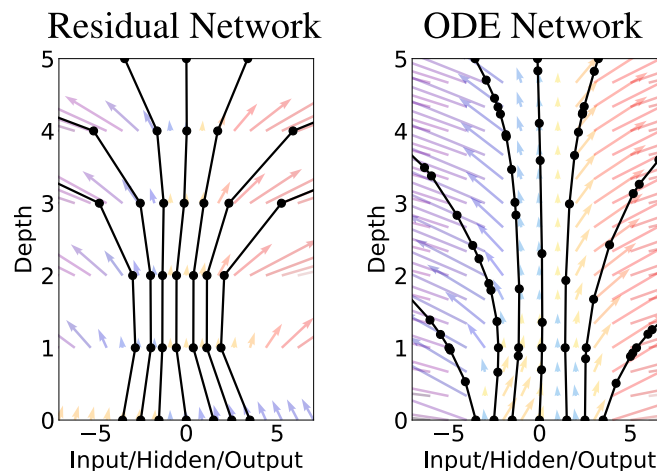


Figure 1: *Left:* A Residual network defines a discrete sequence of finite transformations. *Right:* A ODE network defines a vector field, which continuously transforms the state. *Both:* Circles represent evaluation locations.

Figure 1 of "Neural Ordinary Differential Equations",  
<https://arxiv.org/abs/1806.07366>

# Flow Matching: Transport Equation

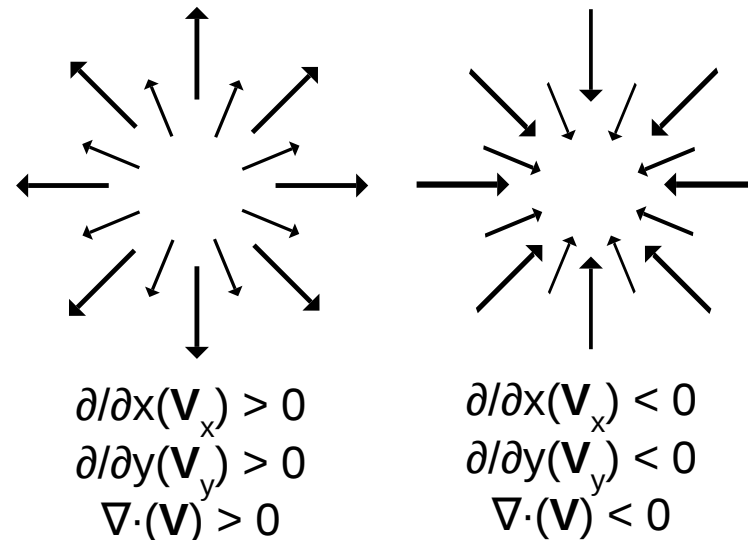
A vector field  $u_t$  is said to **generate** a probability density path  $p_t$  if the *transport equation* holds:

$$\frac{d}{dt} p_t(\mathbf{x}) = - \operatorname{div} \left( \overbrace{p_t(\mathbf{x}) u_t(\mathbf{x})}^{\text{flux, jehle}} \right),$$

*kolik jednotek*

where the **divergence**  $\operatorname{div}(\mathbf{z}) \stackrel{\text{def}}{=} \sum_i \frac{\partial z_i}{\partial x_i}$  is a vector operator that operates on a vector field, producing for every point a scalar value, the field's *source* at that point (a positive value means a point is a source; negative if it is a sink).

The  $p_t(\mathbf{x}) u_t(\mathbf{x})$  is a *flux*, the probability mass passing through every point of the space (in the direction of the vector field).



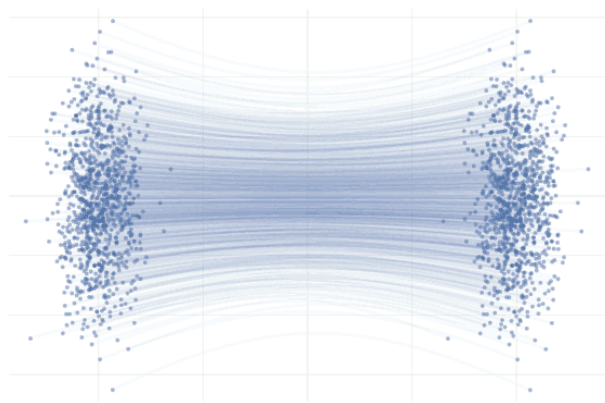
[https://upload.wikimedia.org/wikipedia/commons/e/ee/Divergence\\_\(captions\).svg](https://upload.wikimedia.org/wikipedia/commons/e/ee/Divergence_(captions).svg)

# Flow Matching: Basic Objective

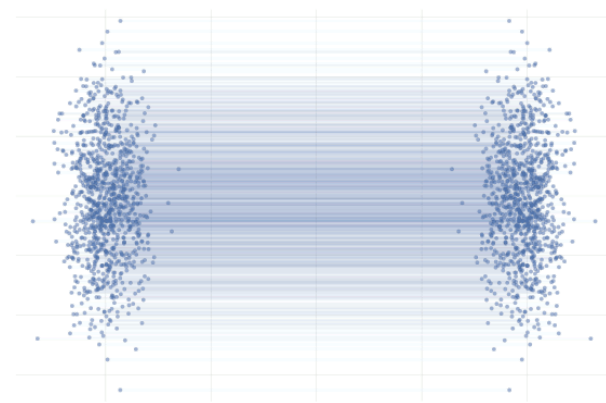
Assume we have a base distribution  $p_0$ , usually  $\mathcal{N}(\mathbf{0}, \mathbf{I})$ , and samples  $\mathbf{x}^{(i)}$  of the data generating distribution  $p_1 = p_{\text{data}}$ . Given a target probability density path  $p_t$  and a corresponding vector field  $u_t$  generating  $p_t$ , the *flow matching (FM) objective* is

$$\mathcal{L}_{\text{FM}}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \mathbb{E}_{t, p_t(\mathbf{x})} \|v_t(\mathbf{x}; \boldsymbol{\theta}) - u_t(\mathbf{x})\|^2.$$

However, we need to overcome that we have no prior knowledge on how the  $p_t$  and  $u_t$  should look like given that there are many possible probability density paths  $p_t$ , and that for an arbitrary  $p_t$ , we usually do not have access to the closed form of its generating vector field  $u_t$ .



<https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html>



<https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html>

# Flow Matching: Basic Overview

Once we solve the remaining issues with the flow matching objective, we train a NN model  $v_t(\mathbf{x}; \boldsymbol{\theta})$  by predicting (matching) the vector field  $u_t(\mathbf{x})$ .

Usually, the UNet architecture with skip connections is used to model  $v_t$ .

## Training

During training, we minimize an objective corresponding to flow matching

$$\mathcal{L}_{\text{FM}}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \mathbb{E}_{t, p_t(\mathbf{x})} \|v_t(\mathbf{x}; \boldsymbol{\theta}) - u_t(\mathbf{x})\|^2,$$

i.e., by performing a regression on the vector predicted by the model  $v_t$ .

## Sampling

In order to generate an image, we start by sampling  $\mathbf{x}_0 \sim p_0$ , and then perform numerical integration by the Euler method using  $T$  steps by

$$\mathbf{x}_{k+1/T} \leftarrow \mathbf{x}_k + \frac{1}{T} v_t(\mathbf{x}_k; \boldsymbol{\theta}).$$

*postupně produkován polem  
a tvořím výstup*

More involved methods like the midpoint or Runge-Kutta can also be used.

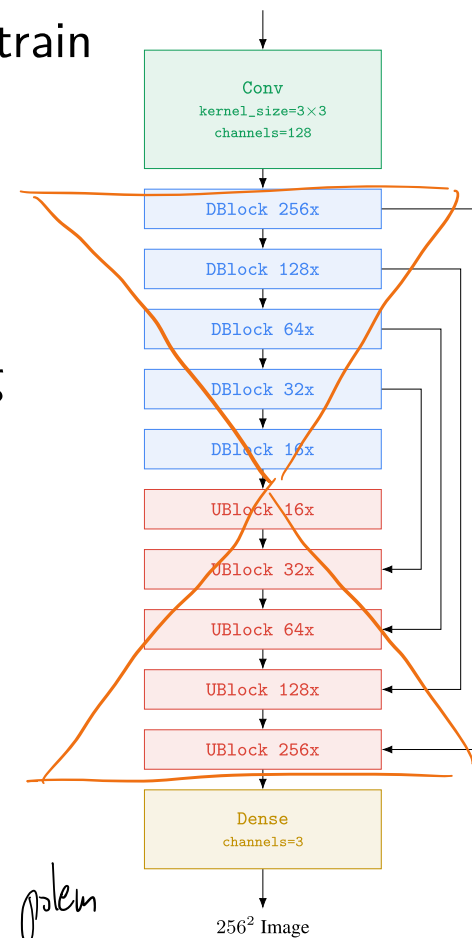


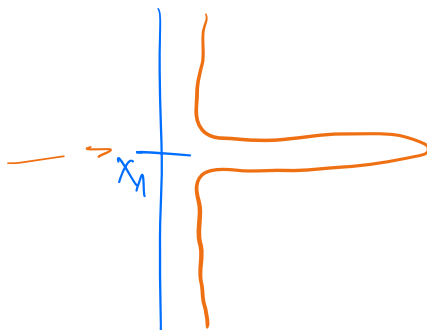
Figure A.30 of "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding".  
<https://arxiv.org/abs/2205.11487>

# Conditional Flow Matching

# Conditional and Marginal Probability Paths

Instead of directly defining the probability density path, we can construct it as a mixture of simpler probability paths. Together with the fact that we do not have direct access to the data generating distribution  $p_1 = p_{\text{data}}$  apart from its samples, we turn to **conditional probability paths**  $p_t(\mathbf{x}|\mathbf{x}_1)$ , which we design so that

- $p_0(\mathbf{x}|\mathbf{x}_t) = p_0(\mathbf{x})$ ,
- $p_1(\mathbf{x}|\mathbf{x}_1)$  is tightly concentrated around  $\mathbf{x}_1$ , for example by using a normal distribution with a small variance  $\sigma_{\min}^2 > 0$ :

$$p_1(\mathbf{x}|\mathbf{x}_1) = \mathcal{N}(\mathbf{x}|\mathbf{x}_1, \sigma_{\min}^2 \mathbf{I}).$$


We can then define the *marginal probability path* as

$$p_t(\mathbf{x}) \stackrel{\text{def}}{=} \mathbb{E}_{\mathbf{x}_1 \sim p_{\text{data}}} [p_t(\mathbf{x}|\mathbf{x}_1)] = \int p_t(\mathbf{x}|\mathbf{x}_1) p_{\text{data}}(\mathbf{x}_1) d\mathbf{x}_1.$$

Because we defined the conditional probability paths to concentrate tightly around  $\mathbf{x}_1$ , the marginal probability  $p_1(\mathbf{x}) \approx p_{\text{data}}$ .



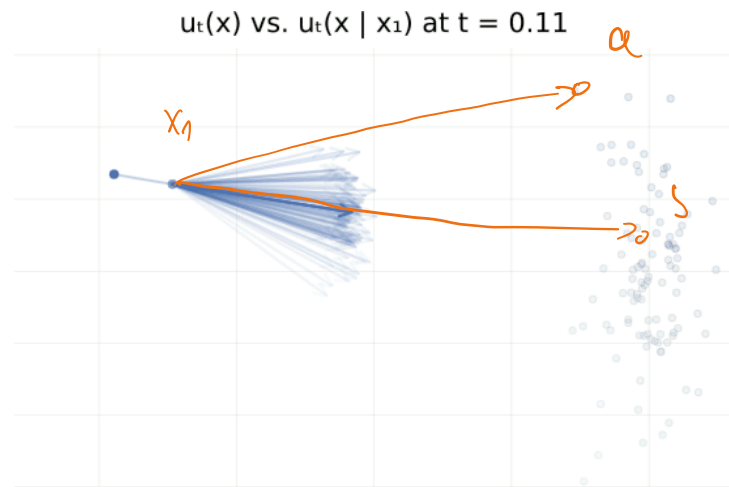
# Conditional and Marginal Vector Fields

Analogously to how we defined the marginal probability path using the conditional probability paths, it is also possible to define the *marginal vector field* (the vector field of the marginal probability path) using the *conditional vector fields*  $u_t(\mathbf{x}|\mathbf{x}_1)$  (the vector fields of the conditional probability paths):

*jak moc ještě dává smysl to generant z  $x_1$*

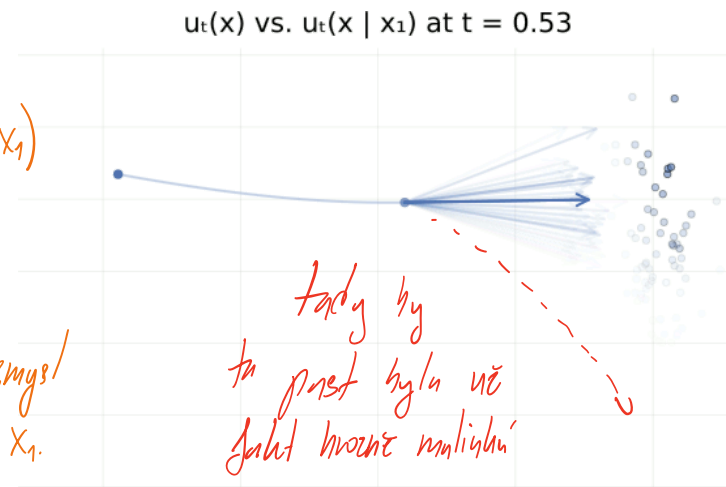
$$u_t(\mathbf{x}) \stackrel{\text{def}}{=} \mathbb{E}_{\mathbf{x}_1 \sim p_{\text{data}}} \left[ u_t(\mathbf{x}|\mathbf{x}_1) \frac{p_t(\mathbf{x}|\mathbf{x}_1)}{p_t(\mathbf{x})} \right] = \int u_t(\mathbf{x}|\mathbf{x}_1) \frac{p_t(\mathbf{x}|\mathbf{x}_1)p_{\text{data}}(\mathbf{x})}{p_t(\mathbf{x})} d\mathbf{x}_1.$$

Such a marginal vector field actually generates the marginal probability path.



<https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html>

*$p_t(a|x_1) < p_t(b|x_1)$*   
*daleko víc dává smysl generant b z  $x_1$ .*



<https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html>

*tady by to prost bylo už jako hodně nepochopitelné*

# Conditional and Marginal Vector Fields

To verify that the marginal vector field generates the marginal probability path, we need to check that  $p_t$  and  $u_t$  satisfy the transport equation:

$$\begin{aligned}
 \frac{d}{dt}p_t(\mathbf{x}) &= \mathbb{E}_{\mathbf{x}_1 \sim p_{\text{data}}} \left[ \frac{d}{dt}p_t(\mathbf{x}|\mathbf{x}_1) \right] \\
 &= \mathbb{E}_{\mathbf{x}_1 \sim p_{\text{data}}} \left[ -\operatorname{div} (u_t(\mathbf{x}|\mathbf{x}_t)p_t(\mathbf{x}|\mathbf{x}_1)) \right] \\
 &= -\operatorname{div} \left( \mathbb{E}_{\mathbf{x}_1 \sim p_{\text{data}}} [u_t(\mathbf{x}|\mathbf{x}_t)p_t(\mathbf{x}|\mathbf{x}_1)] \right) \\
 &= -\operatorname{div} \left( \underbrace{\mathbb{E}_{\mathbf{x}_1 \sim p_{\text{data}}} [u_t(\mathbf{x}|\mathbf{x}_t)p_t(\mathbf{x}|\mathbf{x}_1)]}_{\text{definition of } u_t(\mathbf{x})} p_t(\mathbf{x}) \right) \\
 &= -\operatorname{div} (u_t(\mathbf{x})p_t(\mathbf{x})).
 \end{aligned}$$

Note that swapping a derivative and an integral requires various smoothness conditions; we assume all our objects “nice enough”.

Even with the definition of the marginal vector field, obtaining an unbiased estimate of the flow matching objective is still intractable (the integrals in the definition of the conditional probability path/vector field are intractable).

However, we can use the following simplified *conditional flow matching (CFM) objective*:

$$\mathcal{L}_{\text{CFM}}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \mathbb{E}_{t, \mathbf{x}_1 \sim p_{\text{data}}, \mathbf{x} \sim p_t(\mathbf{x}|\mathbf{x}_1)} \|v_t(\mathbf{x}; \boldsymbol{\theta}) - u_t(\mathbf{x}|\mathbf{x}_1)\|^2.$$

This objective allows unbiased estimates, given that only the conditional variants of  $p_t$  and  $u_t$  are needed.

Importantly, while this objective  $\mathcal{L}_{\text{CFM}}$  is different from the original flow matching objective  $\mathcal{L}_{\text{FM}}$ , it has the same gradients with respect to  $\boldsymbol{\theta}$ .

WIP: The proof that the gradient with respect to  $\theta$  of  $\mathcal{L}_{\text{FM}}$  and  $\mathcal{L}_{\text{CFM}}$  will be added later.

We now define the conditional probability paths that we will use. We consider normal distributions with time-dependent mean  $\mu_t(\mathbf{x}_1)$  and variance  $\sigma_t^2(\mathbf{x}_1)$ :

$$p_t(\mathbf{x}|\mathbf{x}_1) \stackrel{\text{def}}{=} \mathcal{N}(\mathbf{x}|\mu_t(\mathbf{x}_1), \sigma_t^2(\mathbf{x}_1)\mathbf{I}).$$

We require

- $\mu_0(\mathbf{x}_1) = 0$  and  $\sigma_0^2(\mathbf{x}_1) = 1$  so that  $p_0(\mathbf{x}|\mathbf{x}_1) = p_0(\mathbf{x})$ , and
- $\mu_t(\mathbf{x}_1) = \mathbf{x}_1$  and  $\sigma_1^2(\mathbf{x}_1) = \sigma_{\min}^2$  so that  $p_1(\mathbf{x}|\mathbf{x}_1)$  is concentrated around  $\mathbf{x}_1$ .

While there are infinitely many vector fields generating these probability paths, we use the simplest one, corresponding to the flow (dependent on  $\mathbf{x}_1$ )

$$\varphi_t(\mathbf{x}) = \sigma_t(\mathbf{x}_1)\mathbf{x} + \mu_t(\mathbf{x}_1),$$

which is an affine transformation mapping standard normal distribution to a normal distribution with mean  $\mu_t(\mathbf{x}_1)$  and variance  $\sigma_t^2(\mathbf{x}_1)$ .

# Construction of the Conditional Vector Field

We now derive the conditional vector field  $u_t(\mathbf{x}|\mathbf{x}_1)$  so that its flow is the defined

$$\varphi_t(\mathbf{x}) = \sigma_t(\mathbf{x}_1)\mathbf{x} + \mu_t(\mathbf{x}_1).$$

Recalling the flow ODE  $\frac{d}{dt}\varphi_t(\mathbf{x}) = u_t(\varphi_t(\mathbf{x}))$ , it could be used to defined  $u_t$ , but unfortunately for  $\varphi_t(\mathbf{x})$ , not for arbitrary  $\mathbf{x}$ . *→ udeľme pole z tech trajektorii*

However, the affine map  $\varphi_t$  has an analytical inverse (assuming  $\sigma_t(\mathbf{x}_1) > 0$ )

*odtud jsem vyšel, abyoh dostal "z".*

$$\varphi_t^{-1}(\mathbf{z}) = \frac{\mathbf{z} - \mu_t(\mathbf{x}_1)}{\sigma_t(\mathbf{x}_1)}.$$

Therefore, when we consider  $\mathbf{z} = \varphi_t(\mathbf{x})$ , we get  $u_t(\mathbf{z}|\mathbf{x}_1) = \varphi'_t(\varphi_t^{-1}(\mathbf{z}))$ .

Using the derivative  $\varphi'_t(\mathbf{x}|\mathbf{x}_1) = \sigma'_t(\mathbf{x}_1)\mathbf{x} + \mu'_t(\mathbf{x}_1)$ , we obtain that

$$u_t(\mathbf{z}|\mathbf{x}_1) = \frac{\sigma'_t(\mathbf{x}_1)}{\sigma_t(\mathbf{x}_1)} (\mathbf{z} - \mu_t(\mathbf{x}_1)) + \mu'_t(\mathbf{x}_1).$$

# Construction of the Conditional Vector Field

The authors propose to use conditional paths with mean and variance changing linearly in  $t$ :

$$\mu_t(\mathbf{x}_1) \stackrel{\text{def}}{=} t\mathbf{x}_1, \quad \sigma_t(\mathbf{x}_1) \stackrel{\text{def}}{=} 1 - (1 - \sigma_{\min})t.$$

Therefore, the corresponding flow and vector field are

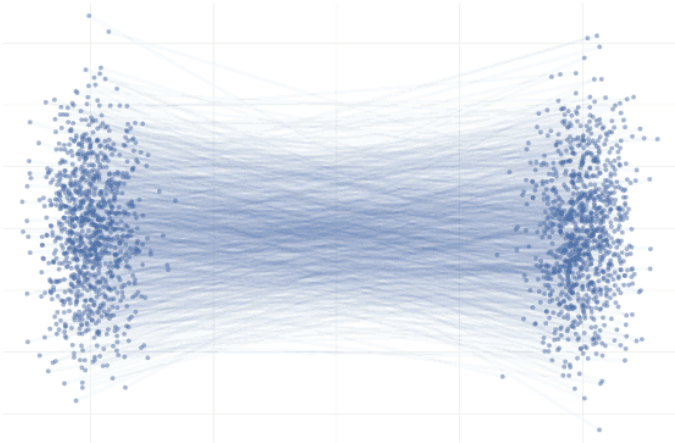
$$\varphi_t(\mathbf{x}) = (1 - (1 - \sigma_{\min})t)\mathbf{x} + t\mathbf{x}_1, \quad \text{such a flow is called Optimal Transport}$$

$$u_t(\mathbf{x}|\mathbf{x}_1) = \frac{1 - \sigma_{\min}}{1 - (1 - \sigma_{\min})t} (\mathbf{x} - t\mathbf{x}_1) + \mathbf{x}_1 = \frac{\mathbf{x}_1 - (1 - \sigma_{\min})\mathbf{x}}{1 - (1 - \sigma_{\min})t}.$$

Finally, recalling the general form  $\mathcal{L}_{\text{CFM}}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \mathbb{E}_{t, \mathbf{x}_1 \sim p_{\text{data}}, \mathbf{x} \sim p_t(\mathbf{x}|\mathbf{x}_1)} \|v_t(\mathbf{x}; \boldsymbol{\theta}) - u_t(\mathbf{x}|\mathbf{x}_1)\|^2$ , for our specific case of OT flow we obtain

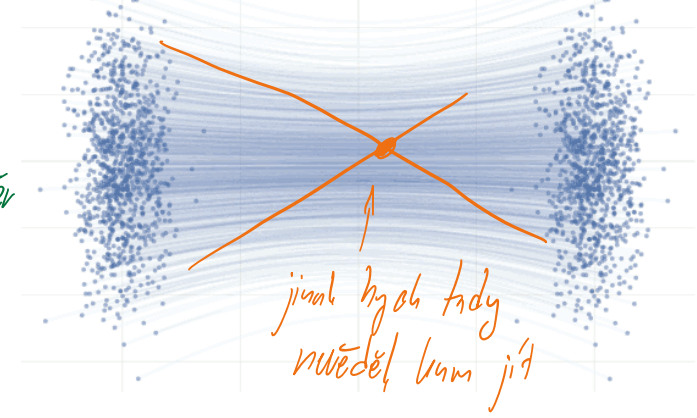
$$\mathcal{L}_{\text{CFM}}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \mathbb{E}_{t, \mathbf{x}_1 \sim p_{\text{data}}, \mathbf{x}_0 \sim p_0} \|v_t(\varphi_t(\mathbf{x}_0); \boldsymbol{\theta}) - \underbrace{\left( \mathbf{x}_1 - \underbrace{(1 - \sigma_{\min})\mathbf{x}_0}_{\text{můžeme šetřit těsně vedle}} \right)}_{\substack{\text{tohle je ten vektor, který trávíme a} \\ \text{kteřý je nezávislý na } t}} \|^2.$$





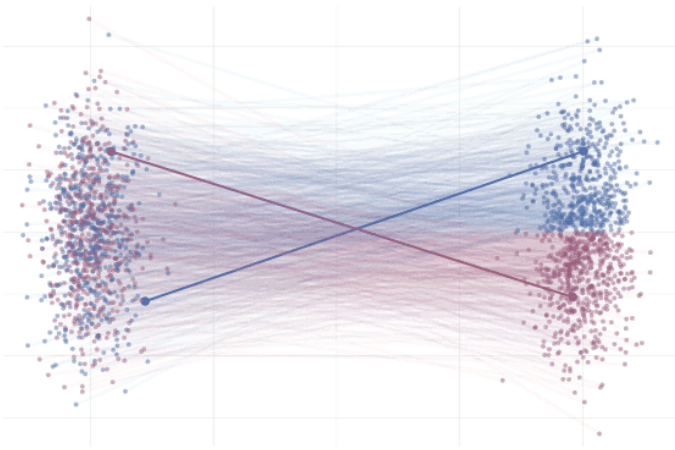
protože je řešení jednoznačné, neexistuje žádný lůžum

tohle trénuj  
→  
uprav je vlastně průměr  
zleva, proto je to  
hustější u středu

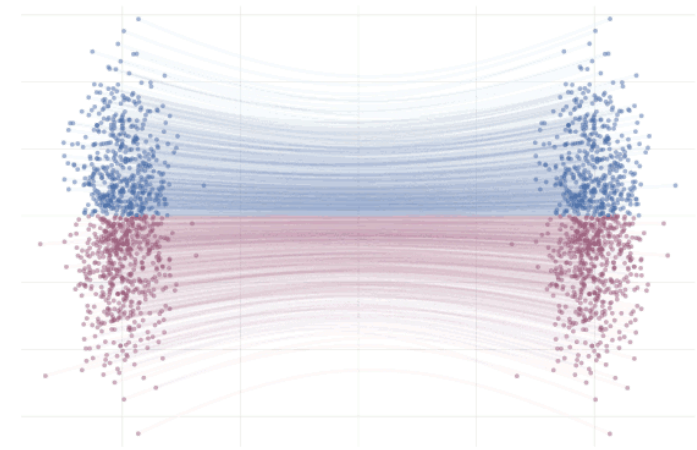


<https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html>

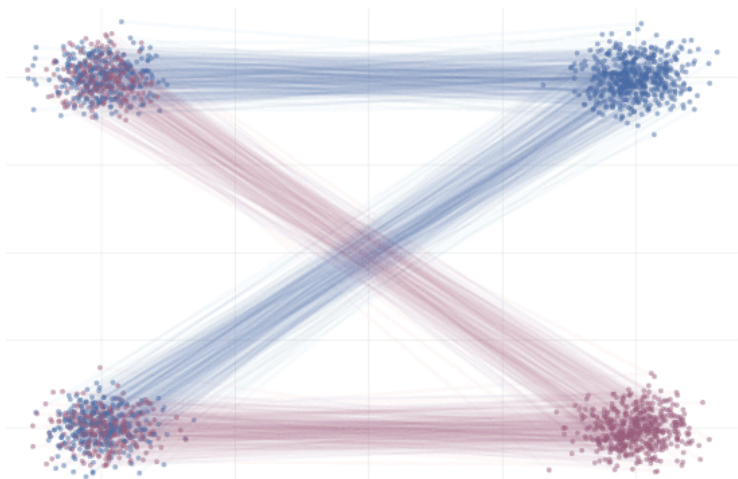
<https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html>



<https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html>

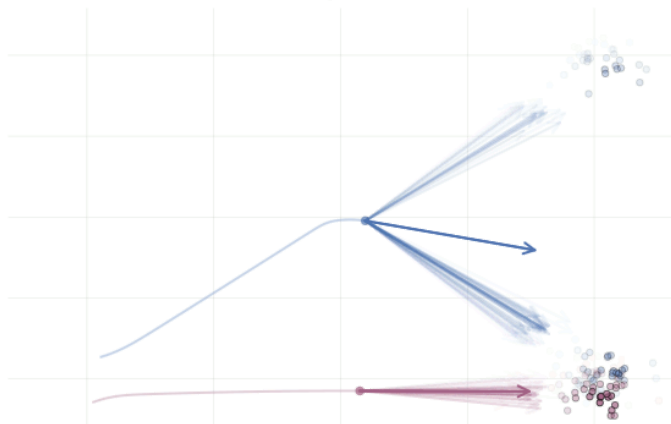


<https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html>

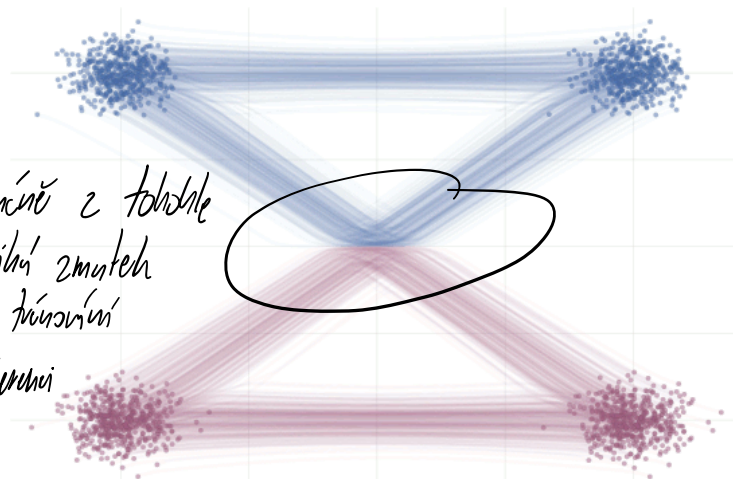


<https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html>

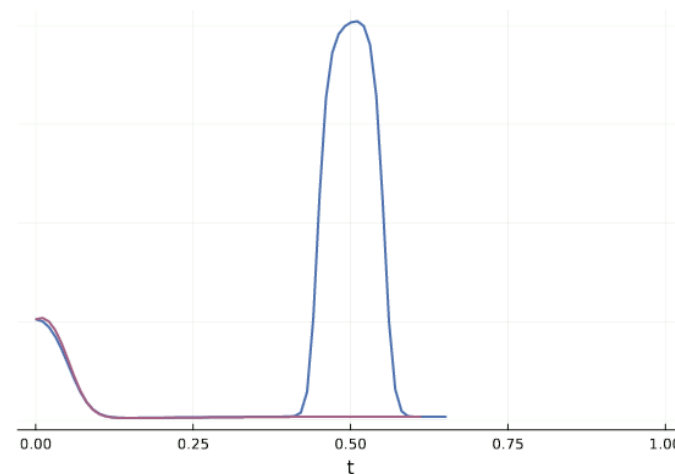
$u_t(x)$  vs.  $u_t(x | x_1)$  at  $t = 0.53$



<https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html>



<https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html>



<https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html>

# Conditional Flow Matching Algorithm

Let  $v_t(\mathbf{x}; \boldsymbol{\theta})$  be a UNet-like neural network model predicting the vector field.

## Training

During training, we minimize CFM objective by SGD/Adam:

$$\mathcal{L}_{\text{CFM}}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \mathbb{E}_{t, \mathbf{x}_1 \sim p_{\text{data}}, \mathbf{x}_0 \sim p_0} \left\| v_t(\varphi_t(\mathbf{x}_0); \boldsymbol{\theta}) - (\mathbf{x}_1 - (1 - \sigma_{\min})\mathbf{x}_0) \right\|^2.$$

Specifically, we generate batches of training data  $\mathbf{X} \sim p_{\text{data}}$  as usual, for each batch example we also generate  $t \sim U[0, 1]$  and  $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , and we minimize the mean squared error *vygenerujmy čas* *vyg. vyph. bod*

$$\left\| v_t(\varphi_t(\mathbf{x}_0); \boldsymbol{\theta}) - (\mathbf{x}_1 - (1 - \sigma_{\min})\mathbf{x}_0) \right\|^2.$$

## Sampling

In order to generate an image, we start by sampling  $\mathbf{x}_0 \sim p_0$ , and then perform numerical integration by the Euler method using  $T$  steps by

$$\mathbf{x}_{k+1/T} \leftarrow \mathbf{x}_k + \frac{1}{T} v_t(\mathbf{x}_k; \boldsymbol{\theta}).$$

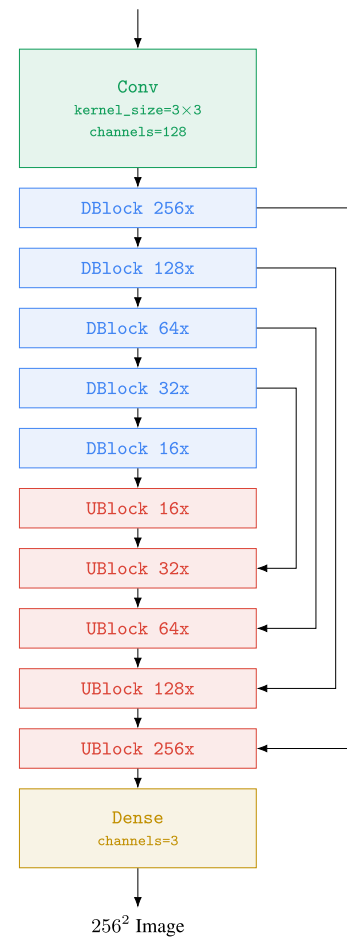


Figure A.30 of "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding".  
<https://arxiv.org/abs/2205.11487>

- An introduction to Flow Matching <https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html#fn:mini-batch-ot-deterministic-vs-stochastic>
- A Visual Dive into Conditional Flow Matching <https://dl.heeere.com/conditional-flow-matching/blog/conditional-flow-matching/>
- Diffusion Meets Flow Matching: Two Sides of the Same Coin <https://diffusionflow.github.io/>

Vector field  $v_t(\mathbf{x}; \boldsymbol{\theta})$  can be predicted by UNet architecture with pre-activated ResNet blocks.

- The current continuous time step is represented using the Transformer sinusoidal embeddings and added “in the middle” of every residual block (after the first convolution).
- Additionally, on several lower-resolution levels, a self-attention block (an adaptation of the Transformer self-attention, which considers the 2D grid of features as a sequence of feature vectors) is commonly used. Because the complexity is asymptotically the image width to the power of four, only the lower-resolution levels are used for this self-attention.

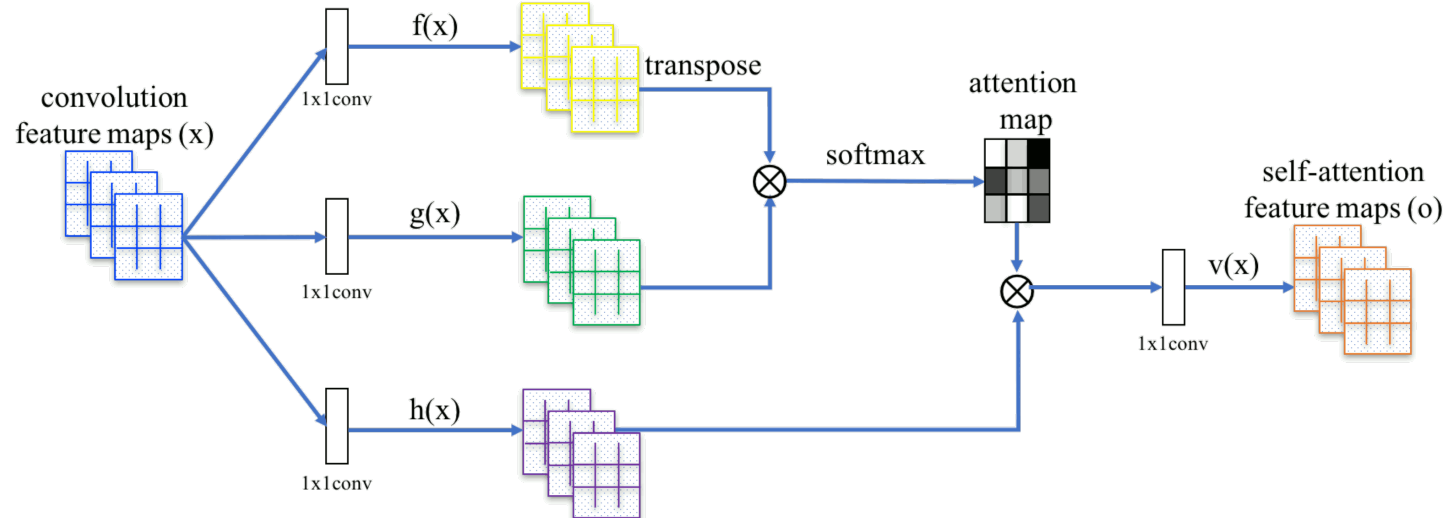


Figure 2 of "Self-Attention Generative Adversarial Networks", <https://arxiv.org/abs/1805.08318>

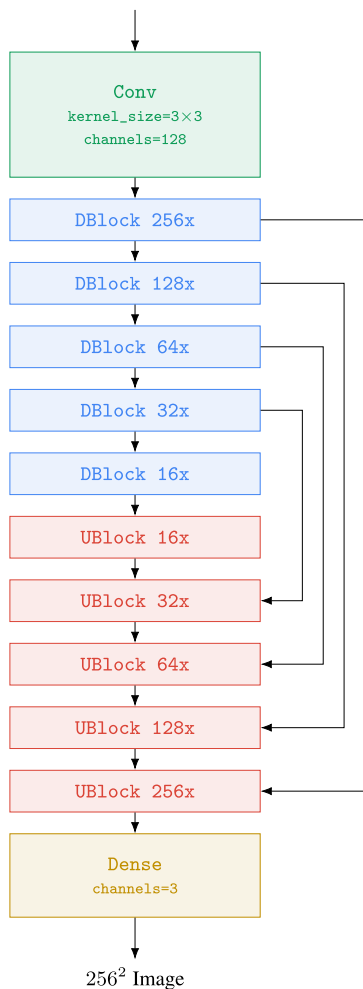


Figure A.30 of "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", <https://arxiv.org/abs/2205.11487>

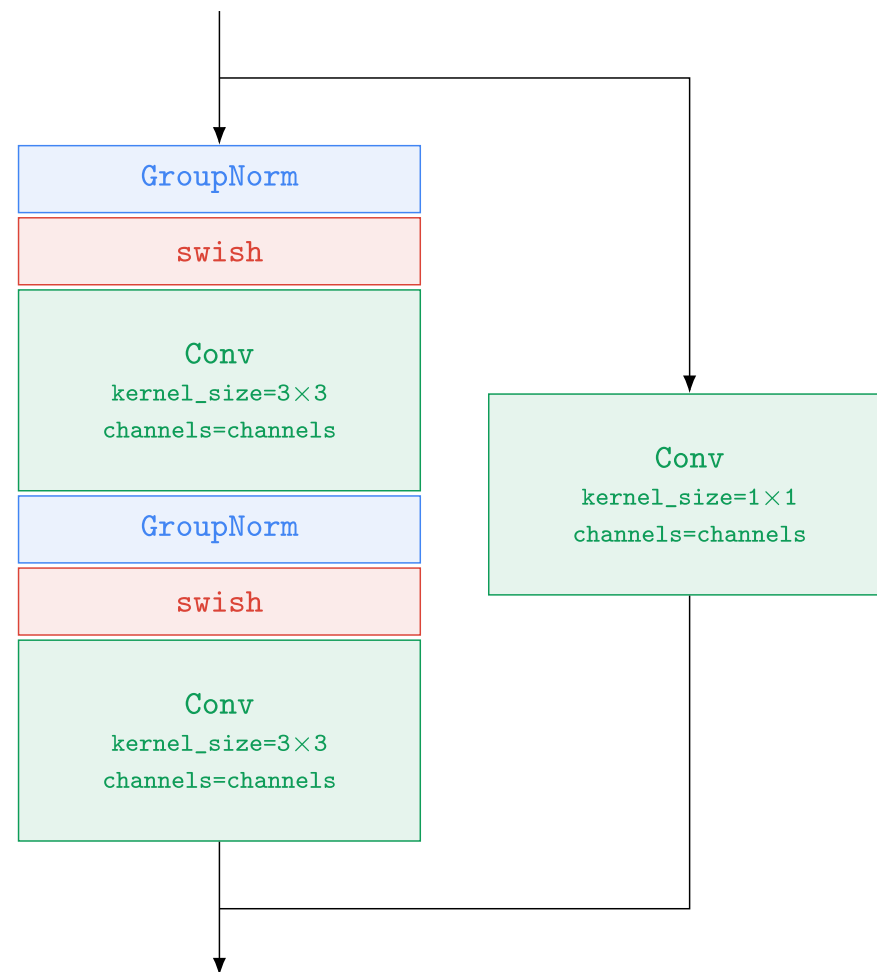


Figure A.27 of "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", <https://arxiv.org/abs/2205.11487>

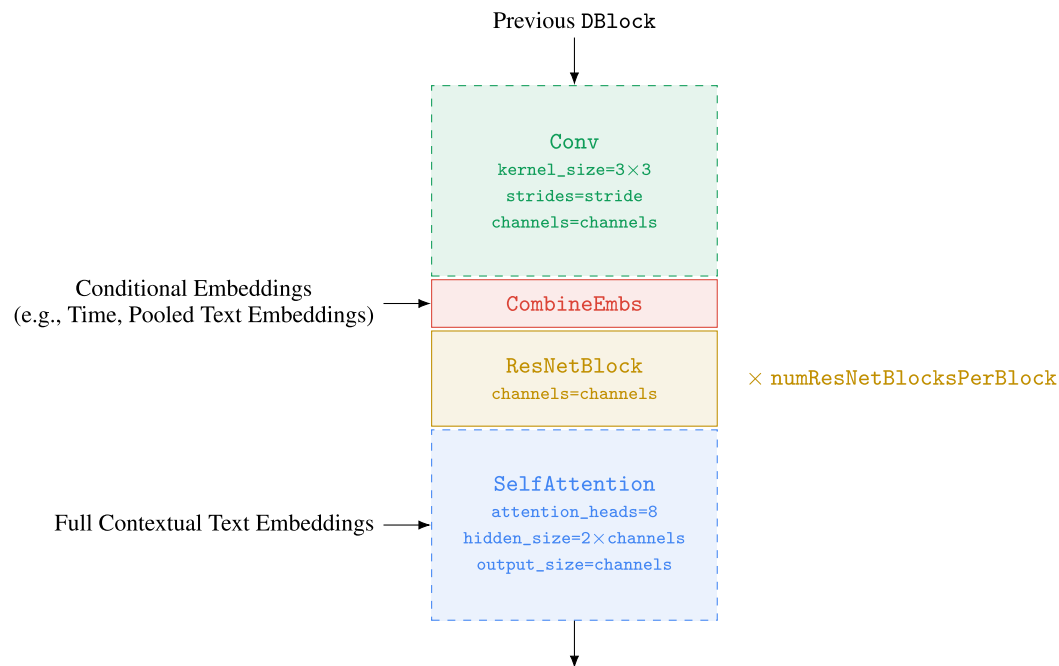


Figure A.28 of "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", <https://arxiv.org/abs/2205.11487>

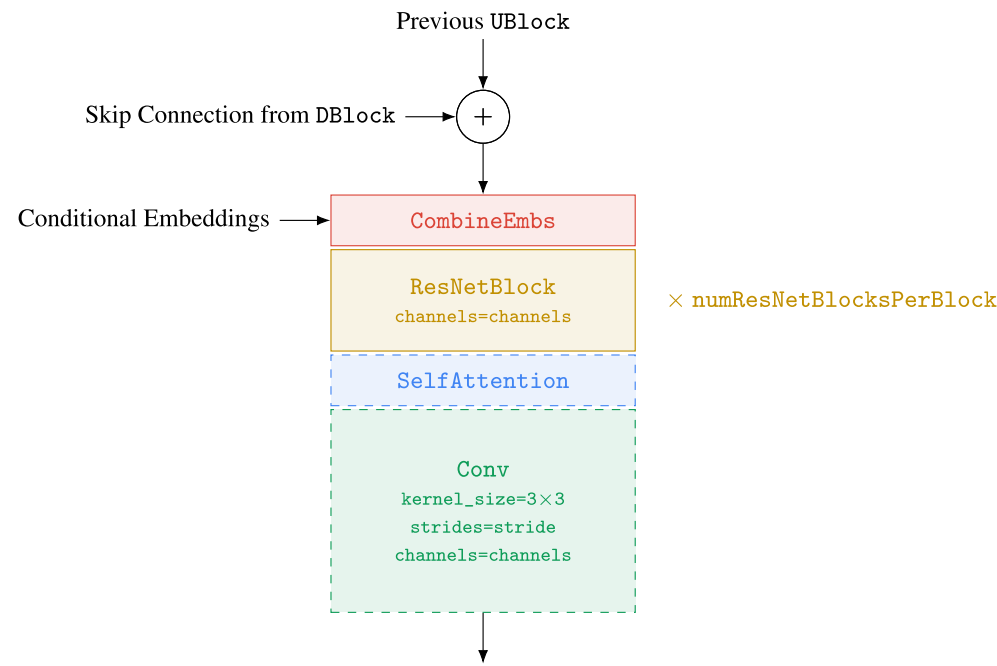


Figure A.29 of "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", <https://arxiv.org/abs/2205.11487>

There are of course many possible variants; furthermore, Visual Transformer can be used instead of the UNet architecture.



# Conditional Models, Classifier-Free Guidance

In many cases we want the generative model to be conditional. We have already seen how to condition it on the current time step. Additionally, we might consider also conditioning on

- an image (e.g., for super-resolution): the image is then resized and concatenated with the input noised image (and optionally in other places, like after every resolution change);
- a text: the usual approach is to encode the text using some pre-trained encoder, and then to introduce an “image-text” attention layer (usually after the self-attention layers).

To make the effect of conditioning stronger during sampling, we might also employ *classifier-free guidance*:

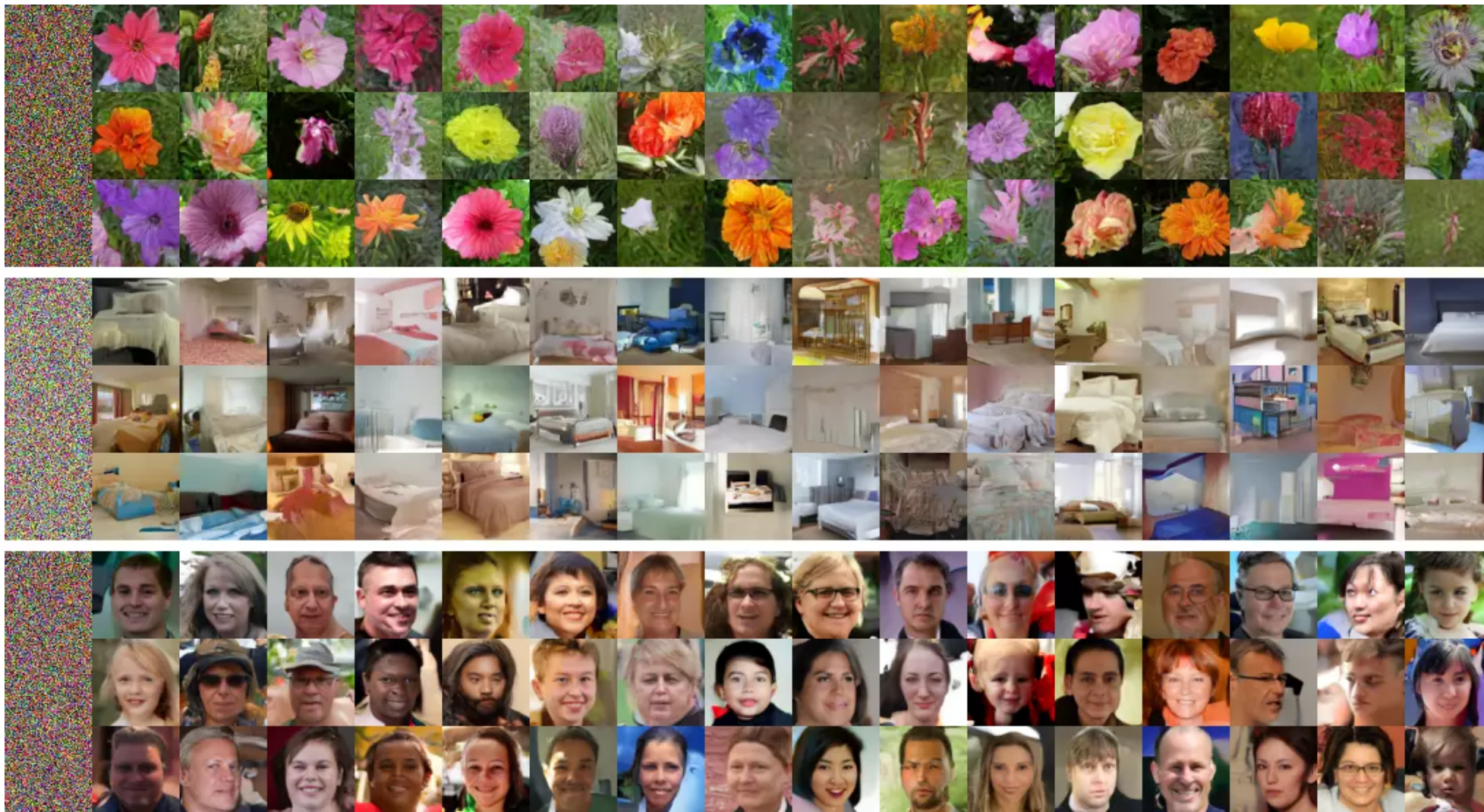
- During training, we sometimes train  $v_t(\mathbf{x}, y; \theta)$  with the conditioning  $y$ , and sometimes we train  $v_t(\mathbf{x}, \emptyset; \theta)$  without the conditioning.
- During sampling, we pronounce the effect of the conditioning by taking the unconditioned vector field and adding the difference between conditioned and unconditioned vector field *weighted by the weight  $w$*  (values like  $w = 5$  or  $w = 7.5$  are mentioned in papers):

a trik to hvacne naschlyin, aby to opredel generova

$$v_t(\mathbf{x}, \emptyset; \theta) + \left[ w(v_t(\mathbf{x}, y; \theta) - v_t(\mathbf{x}_t, \emptyset; \theta)) \right]$$

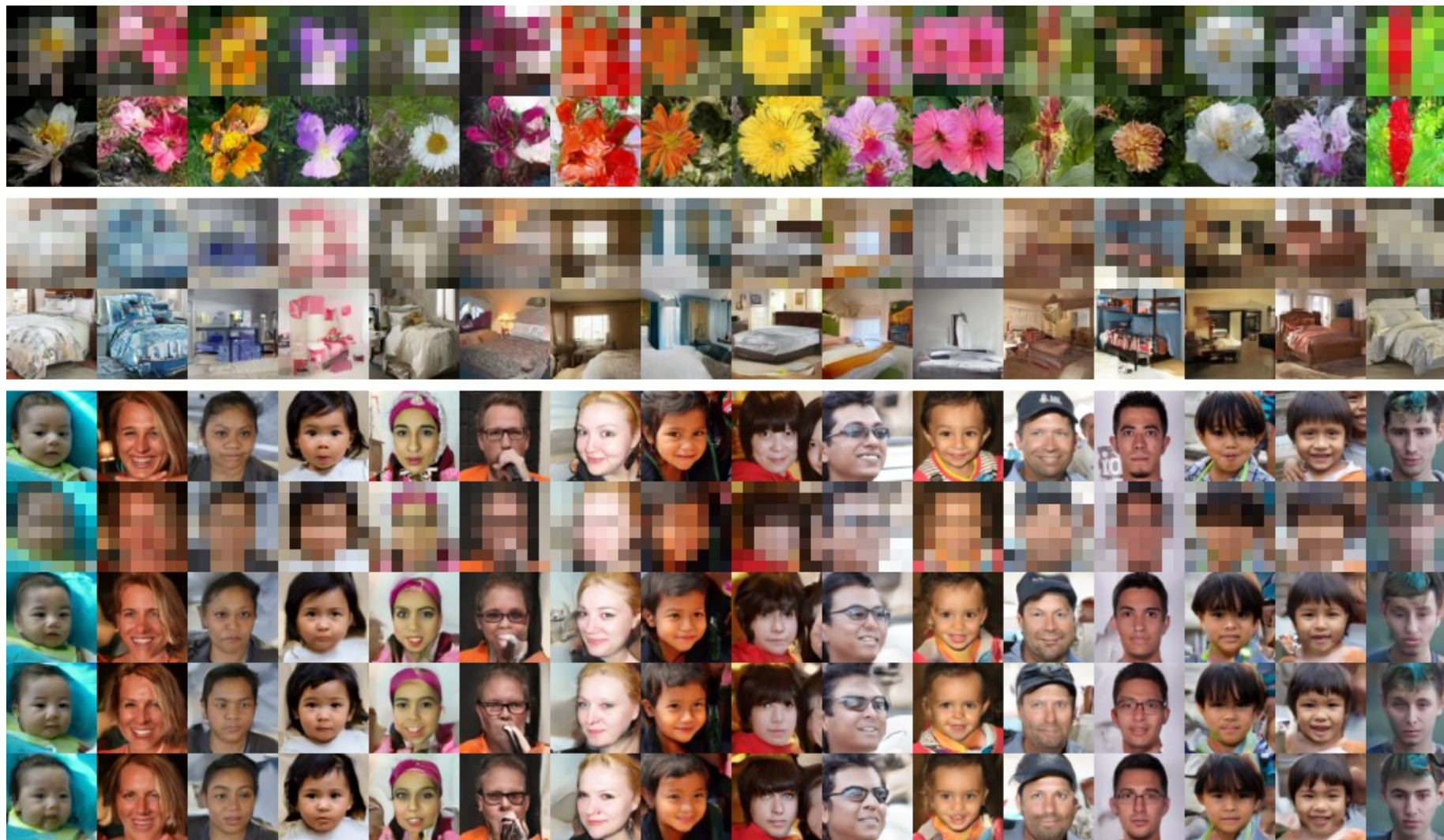
to je presne to, co chci v conditioningu

# Samples from Model Trained Last Year on Practicals





# Samples from Conditional Model Trained Last Year



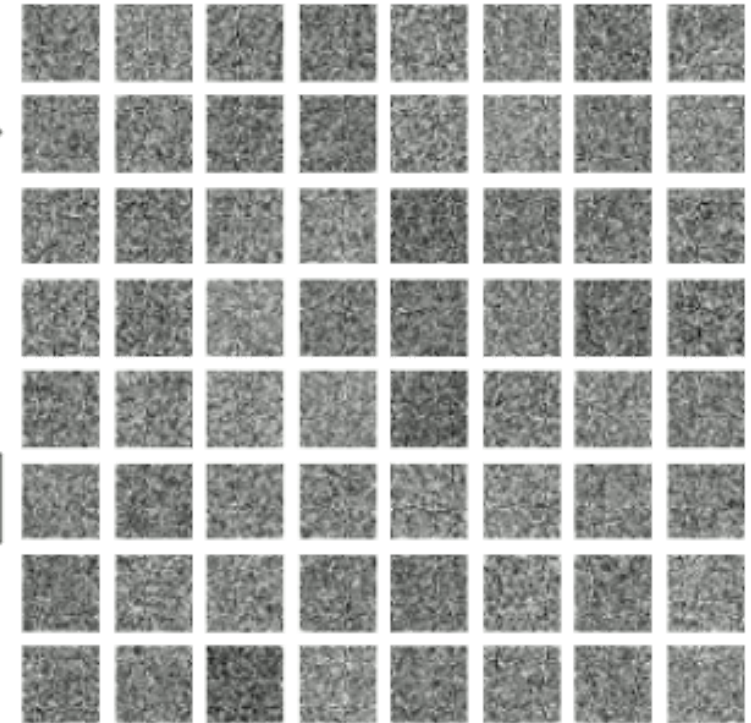
# Diffusion Models

Original MNIST digits



*Diffusion process*

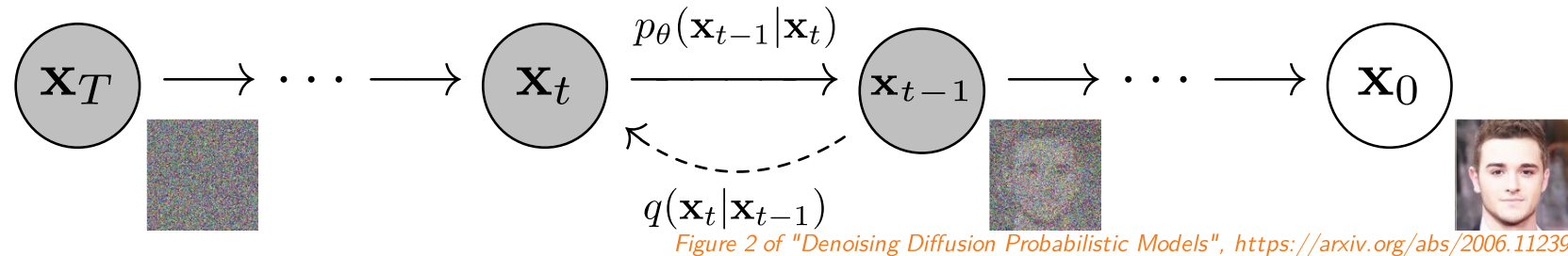
Noisy images 100%



*Denoising process*

[https://miro.medium.com/v2/1\\*jKDPZ9vo2gl0BGKpw9\\_IKw.png](https://miro.medium.com/v2/1*jKDPZ9vo2gl0BGKpw9_IKw.png)





Given a data point  $\mathbf{x}_0$  from a real data distribution  $q(\mathbf{x})$ , we define a  $T$ -step *diffusion process* (or the *forward process*) which gradually adds Gaussian noise to the input image:

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

Our goal is to reverse the forward process  $q(\mathbf{x}_t | \mathbf{x}_{t-1})$ , and generate an image by starting with  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , and then performing the forward process in reverse. We therefore learn a model  $p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)$  to approximate the reverse of  $q(\mathbf{x}_t | \mathbf{x}_{t-1})$ , and obtain a *reverse process*:

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

# Diffusion Models – Model Overview

The  $p_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)$  is commonly modelled using a UNet architecture with skip connections.

## Training

During training, we randomly sample a time step  $t$ , and perform an update of the parameters  $\theta$  in order for  $p_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)$  to better approximate the reverse of  $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ .

## Sampling

In order to sample an image, we start by sampling  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , and then perform  $T$  steps of the reverse process by sampling  $\mathbf{x}_{t-1} \sim p_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)$  for  $t$  from  $T$  down to 1.

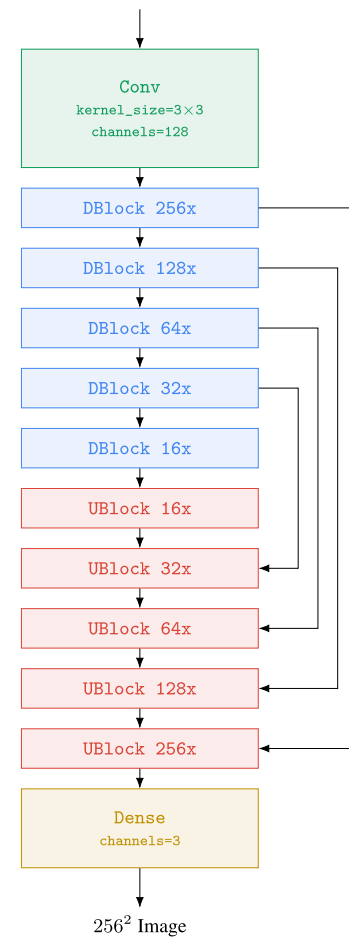


Figure A.30 of "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding".  
<https://arxiv.org/abs/2205.11487>

# Normal Distribution Reminder

Normal (or Gaussian) distribution is a continuous distribution parametrized by a mean  $\mu$  and variance  $\sigma^2$ :

$$\mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

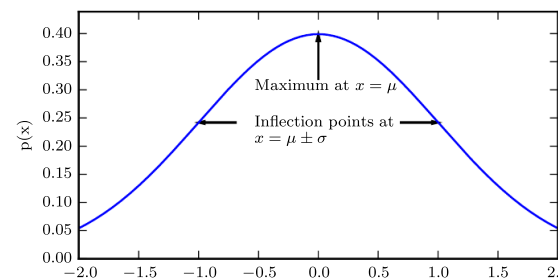


Figure 3.1 of "Deep Learning" book,  
<https://www.deeplearningbook.org>

For a  $D$ -dimensional vector  $\mathbf{x}$ , the multivariate Gaussian distribution takes the form

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \stackrel{\text{def}}{=} \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right).$$

The biggest difference compared to the single-dimensional case is the *covariance matrix*  $\boldsymbol{\Sigma}$ , which is (in the non-degenerate case, which is the only one considered here) a *symmetric positive-definite matrix* of size  $D \times D$ .

However, in this lecture we will only consider *isotropic* distribution, where  $\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}$ :

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \sigma^2 \mathbf{I}) = \prod_i \mathcal{N}(x_i; \mu_i, \sigma^2).$$

# Normal Distribution Reminder

- A normally-distributed random variable  $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2 \mathbf{I})$  can be written using the reparametrization trick also as

$$\mathbf{x} = \boldsymbol{\mu} + \sigma \mathbf{e}, \quad \text{where } \mathbf{e} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

- The sum of two independent normally-distributed random variables  $\mathbf{x}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \sigma_1^2 \mathbf{I})$  and  $\mathbf{x}_2 \sim \mathcal{N}(\boldsymbol{\mu}_2, \sigma_2^2 \mathbf{I})$  has normal distribution  $\mathcal{N}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2, (\sigma_1^2 + \sigma_2^2) \mathbf{I})$ .

Therefore, if we have two standard normal random variables  $\mathbf{e}_1, \mathbf{e}_2 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , then

$$\sigma_1 \mathbf{e}_1 + \sigma_2 \mathbf{e}_2 = \sqrt{\sigma_1^2 + \sigma_2^2} \mathbf{e}$$

for a standard normal random variable  $\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ .

# DDPM – The Forward Process

We now describe *Denoising Diffusion Probabilistic Models (DDPM)*.

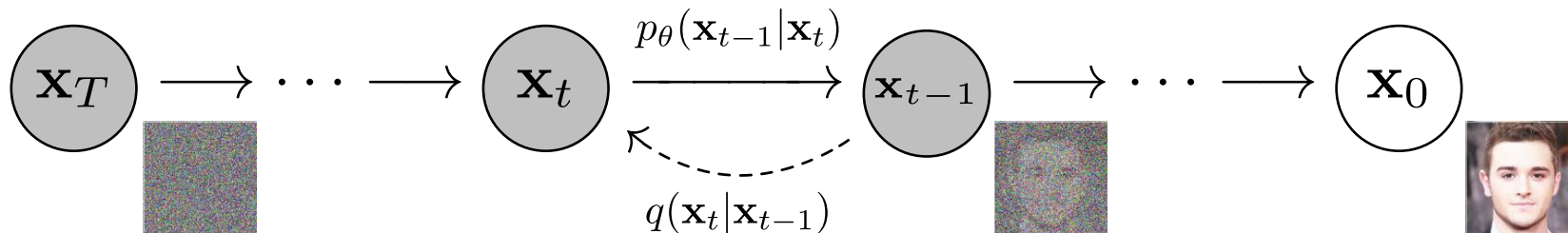


Figure 2 of "Denoising Diffusion Probabilistic Models", <https://arxiv.org/abs/2006.11239>

Given a data point  $\mathbf{x}_0$  from a real data distribution  $q(\mathbf{x})$ , we define a  $T$ -step *diffusion process* (or the *forward process*) which gradually adds Gaussian noise according to some variance schedule  $\beta_1, \dots, \beta_T$ :

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

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

$$= \sqrt{1 - \beta_t} \mathbf{x}_{t-1} + \sqrt{\beta_t} \mathbf{e} \text{ for } \mathbf{e} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

More noise gets gradually added to the original image  $\mathbf{x}_0$ , converging to pure Gaussian noise.



Let  $\alpha_t = 1 - \beta_t$  and  $\bar{\alpha}_t = \prod_{i=1}^t \alpha_i$ . Then we have

$$\begin{aligned}
 \mathbf{x}_t &= \sqrt{\alpha_t} \mathbf{x}_{t-1} + \sqrt{1 - \alpha_t} \mathbf{e}_t \\
 &= \sqrt{\alpha_t} \left( \sqrt{\alpha_{t-1}} \mathbf{x}_{t-2} + \sqrt{1 - \alpha_{t-1}} \mathbf{e}_{t-1} \right) + \sqrt{1 - \alpha_t} \mathbf{e}_t \\
 &= \sqrt{\alpha_t \alpha_{t-1}} \mathbf{x}_{t-2} + \sqrt{\alpha_t (1 - \alpha_{t-1}) + (1 - \alpha_t)} \bar{\mathbf{e}}_{t-1} \\
 &= \sqrt{\alpha_t \alpha_{t-1}} \mathbf{x}_{t-2} + \sqrt{1 - \alpha_t \alpha_{t-1}} \bar{\mathbf{e}}_{t-1} \\
 &= \sqrt{\alpha_t \alpha_{t-1} \alpha_{t-2}} \mathbf{x}_{t-3} + \sqrt{1 - \alpha_t \alpha_{t-1} \alpha_{t-2}} \bar{\mathbf{e}}_{t-2} \\
 &= \dots \\
 &= \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \bar{\mathbf{e}}_0
 \end{aligned}$$

for standard normal random variables  $\mathbf{e}_i$  and  $\bar{\mathbf{e}}_i$ .

In other words, we have shown that  $q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I})$ .

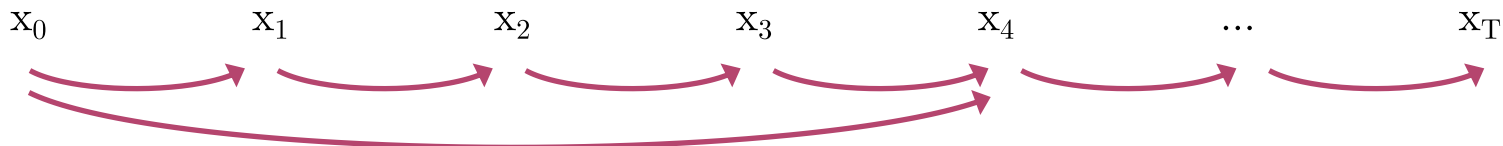
Therefore, if  $\bar{\alpha}_t \rightarrow 0$  as  $t \rightarrow \infty$ , the  $\mathbf{x}_t$  converges to  $\mathcal{N}(\mathbf{0}, \mathbf{I})$  as  $t \rightarrow \infty$ .

Forward diffusion process (fixed)

Data



Noise



CVPR 2022 tutorial <https://cvpr2022-tutorial-diffusion-models.github.io/>

# DDPM – Noise Schedule

Originally, linearly increasing sequence of noise variations  $\beta_1 = 0.0001, \dots, \beta_T = 0.04$  was used.

However, the resulting sequence  $\bar{\alpha}_t$  was not ideal (nearly the whole second half of the diffusion process was mostly just random noise), so later a cosine schedule was proposed:

$$\bar{\alpha}_t = \frac{1}{2} \left( \cos(t/T \cdot \pi) + 1 \right).$$

In practice, we want to avoid both the values of 0 and 1, and keep  $\alpha_t$  in  $[\varepsilon, 1 - \varepsilon]$  range.

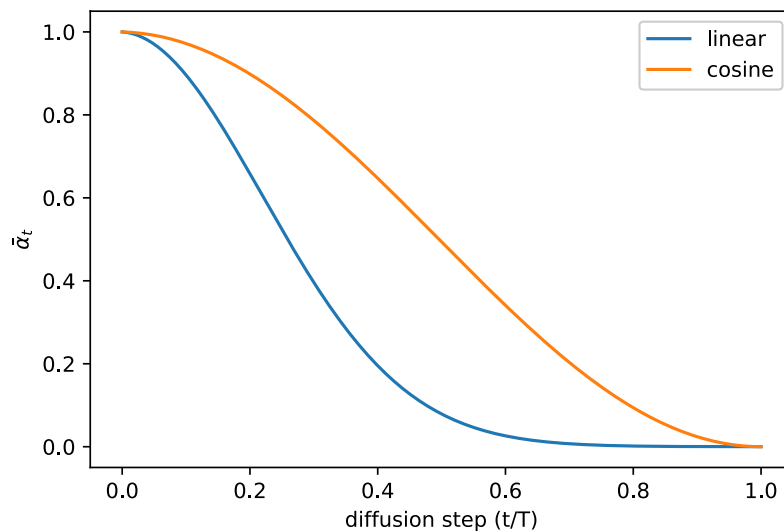


Figure 5.  $\bar{\alpha}_t$  throughout diffusion in the linear schedule and our proposed cosine schedule.

Figure 5 of "Improved Denoising Diffusion Probabilistic Models",  
<https://arxiv.org/abs/2102.09672>

# DDPM – Noise Schedule

We assume the images  $\mathbf{x}_0$  have zero mean and unit variance (we normalize them to achieve that). Then every

$$q(\mathbf{x}_t|\mathbf{x}_0) = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\mathbf{e}$$

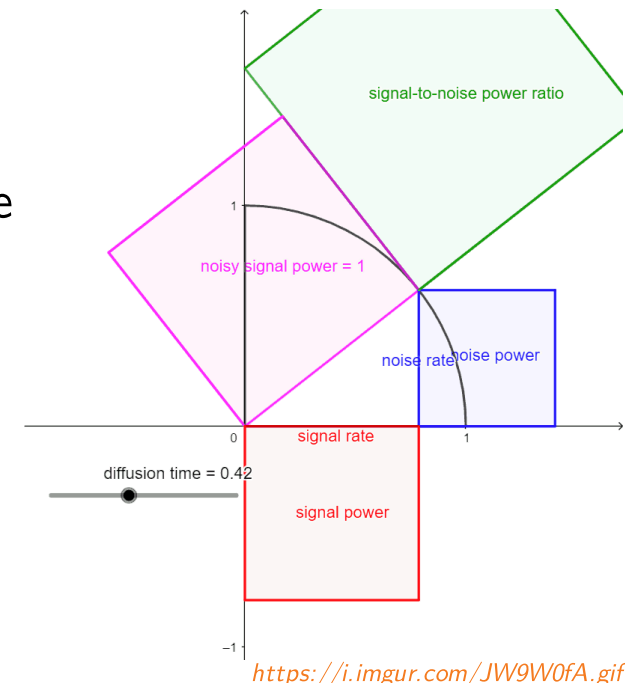
has also zero mean and unit variance.

The  $\sqrt{\bar{\alpha}_t}$  and  $\sqrt{1 - \bar{\alpha}_t}$  can be considered as the *signal rate* and the *noise rate*.

Because  $\sqrt{\bar{\alpha}_t}^2 + \sqrt{1 - \bar{\alpha}_t}^2 = 1$ , the signal rate and the noise rate form a circular arc. The proposed cosine schedule

$$\begin{aligned}\sqrt{\bar{\alpha}_t} &= \cos(t/T \cdot \pi/2), \\ \sqrt{1 - \bar{\alpha}_t} &= \sin(t/T \cdot \pi/2),\end{aligned}$$

corresponds to an uniform movement on this arc.



# DDPM – The Reverse Process

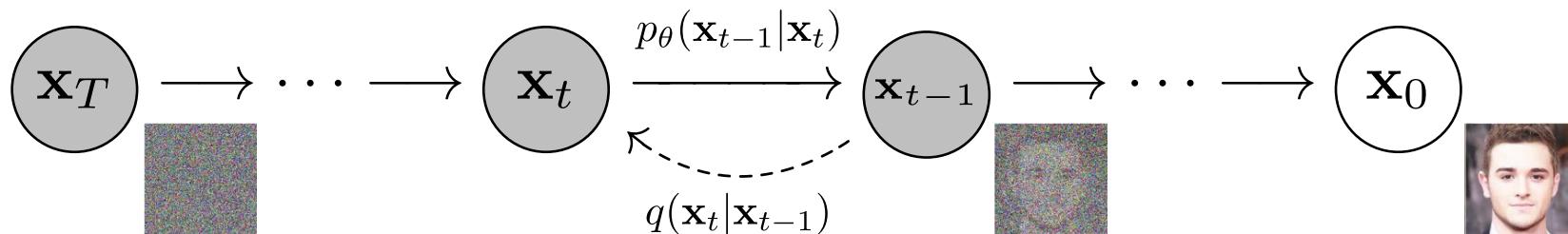


Figure 2 of "Denoising Diffusion Probabilistic Models", <https://arxiv.org/abs/2006.11239>

In order to be able to generate images, we therefore learn a model  $p_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)$  to approximate the reverse of  $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ .

When  $\beta_t$  is small, this reverse is nearly Gaussian, so we represent  $p_{\theta}$  as

$$p_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_{\theta}(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I})$$

for some fixed sequence of  $\sigma_1, \dots, \sigma_T$ .

The whole reverse process is then

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

We now want to derive the loss. First note that the reverse of  $q(\mathbf{x}_t|\mathbf{x}_{t-1})$  is actually tractable when conditioning on  $\mathbf{x}_0$ :

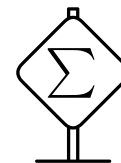
$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\boldsymbol{\beta}}_t \mathbf{I}),$$
$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\mathbf{x}_t,$$
$$\tilde{\boldsymbol{\beta}}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\beta_t.$$

We present the proof on the next slide for completeness.



# Forward Process Reverse Derivation

Starting with the Bayes' rule, we get



$$\begin{aligned}
 q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) &= q(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_0) \frac{q(\mathbf{x}_{t-1} | \mathbf{x}_0)}{q(\mathbf{x}_t | \mathbf{x}_0)} \\
 &\propto \exp \left( -\frac{1}{2} \left( \frac{(\mathbf{x}_t - \sqrt{\alpha_t} \mathbf{x}_{t-1})^2}{\beta_t} + \frac{(\mathbf{x}_{t-1} - \sqrt{\bar{\alpha}_{t-1}} \mathbf{x}_0)^2}{1 - \bar{\alpha}_{t-1}} - \frac{(\mathbf{x}_t - \sqrt{\bar{\alpha}_t} \mathbf{x}_0)^2}{1 - \bar{\alpha}_t} \right) \right) \\
 &= \exp \left( -\frac{1}{2} \left( \frac{\mathbf{x}_t^2 - 2\sqrt{\alpha_t} \mathbf{x}_t \mathbf{x}_{t-1} + \alpha_t \mathbf{x}_{t-1}^2}{\beta_t} + \frac{\mathbf{x}_{t-1}^2 - 2\sqrt{\bar{\alpha}_{t-1}} \mathbf{x}_{t-1} \mathbf{x}_0 + \bar{\alpha}_{t-1} \mathbf{x}_0^2}{1 - \bar{\alpha}_{t-1}} + \dots \right) \right) \\
 &= \exp \left( -\frac{1}{2} \left( \left( \frac{\alpha_t}{\beta_t} + \frac{1}{1 - \bar{\alpha}_{t-1}} \right) \mathbf{x}_{t-1}^2 - 2 \left( \frac{\sqrt{\alpha_t}}{\beta_t} \mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1 - \bar{\alpha}_{t-1}} \mathbf{x}_0 \right) \mathbf{x}_{t-1} + \dots \right) \right)
 \end{aligned}$$

From this formulation, we can derive that  $q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\boldsymbol{\beta}}_t \mathbf{I})$  for

$$\tilde{\boldsymbol{\beta}}_t = 1 / \left( \frac{\alpha_t}{\beta_t} + \frac{1}{1 - \bar{\alpha}_{t-1}} \right) = 1 / \left( \frac{\alpha_t(1 - \bar{\alpha}_{t-1}) + \beta_t}{\beta_t(1 - \bar{\alpha}_{t-1})} \right) = 1 / \left( \frac{\alpha_t + \beta_t - \bar{\alpha}_t}{\beta_t(1 - \bar{\alpha}_{t-1})} \right) = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t,$$

$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = \left( \frac{\sqrt{\alpha_t}}{\beta_t} \mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1 - \bar{\alpha}_{t-1}} \mathbf{x}_0 \right) \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t = \frac{\sqrt{\bar{\alpha}_{t-1}} \beta_t}{1 - \bar{\alpha}_t} \mathbf{x}_0 + \frac{\sqrt{\alpha_t} (1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \mathbf{x}_t.$$

The full derivation of the loss is available in the Bonus Content of this presentation. The resulting loss is

$$L_t = \mathbb{E} \left[ \frac{1}{2 \|\sigma_t \mathbf{I}\|^2} \left\| \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \right\|^2 \right].$$

The model is then changed to predict  $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$  instead of  $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ . The loss then becomes

$$L_t = \mathbb{E} \left[ \frac{(1 - \alpha_t)^2}{2 \alpha_t (1 - \bar{\alpha}_t) \|\sigma_t \mathbf{I}\|^2} \left\| \mathbf{e}_t - \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \mathbf{e}_t, t) \right\|^2 \right].$$

The authors found that training without the weighting term performs better, so the final loss is

$$L_t^{\text{simple}} = \mathbb{E}_{t \in \{1..T\}, \mathbf{x}_0, \mathbf{e}_t} \left[ \left\| \mathbf{e}_t - \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \mathbf{e}_t, t) \right\|^2 \right].$$

Note that both losses have the same optimum if we used independent  $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}_t}$  for every  $t$ .

# DDPM – Training and Sampling Algorithms

## Algorithm 1 Training

```

1: repeat
2:    $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ 
3:    $t \sim \text{Uniform}(\{1, \dots, T\})$ 
4:    $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
5:   Take gradient descent step on
        $\nabla_{\theta} \left\| \epsilon - \epsilon_{\theta}(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t) \right\|^2$ 
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:    $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_{\theta}(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$ 
5: end for
6: return  $\mathbf{x}_0$ 

```

Algorithms 1, 2 of "Denoising Diffusion Probabilistic Models", <https://arxiv.org/abs/2006.11239>

In practice, instead of discrete,  $t$  may be continuous in the  $[0, 1]$  range. Note that sampling using the proposed algorithm is slow because it is common to use  $T = 1000$  steps during sampling.

The value of  $\sigma_t^2$  is chosen to be either  $\beta_t$  or  $\tilde{\beta}_t$ , or any value in between (it can be proven that these values correspond to upper and lower bounds on the reverse process entropy).

Both of these issues are alleviated by using a different sampling algorithm DDIM, which runs in several tens of steps and does not use  $\sigma_t^2$ .

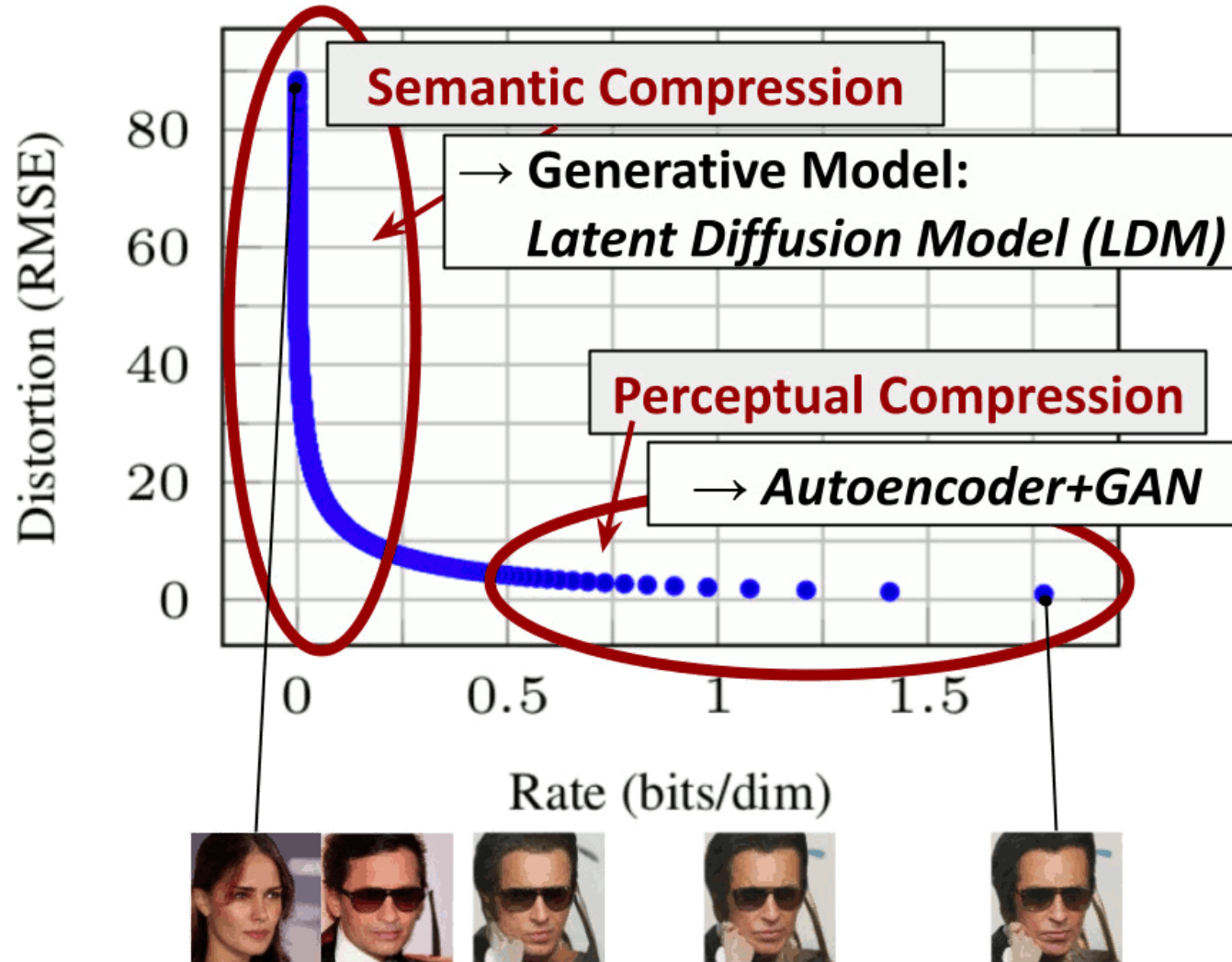


Figure 2 of "High-Resolution Image Synthesis with Latent Diffusion Models", <https://arxiv.org/abs/2112.10752>

# Stable Diffusion – Architecture

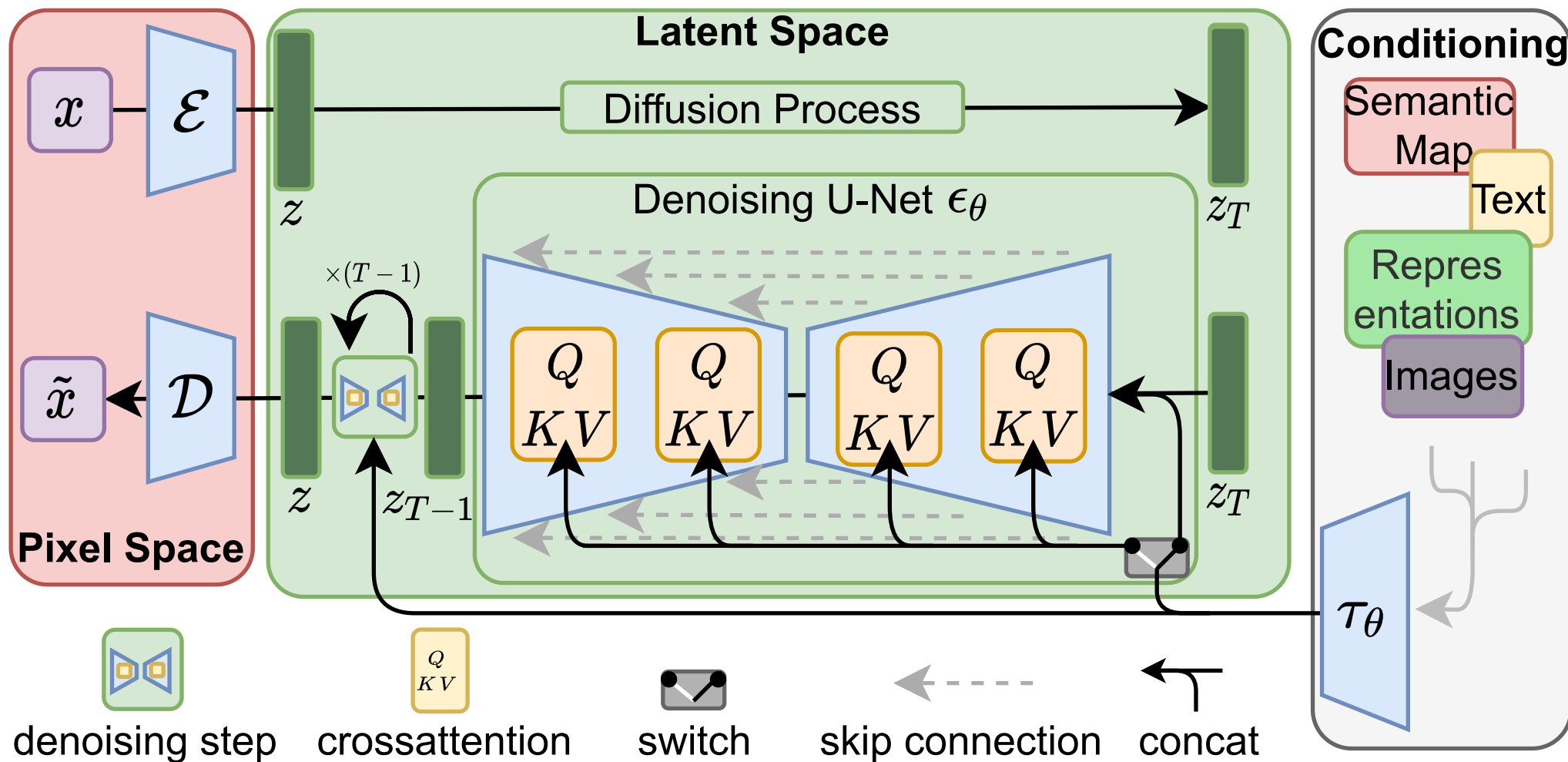


Figure 3 of "High-Resolution Image Synthesis with Latent Diffusion Models", <https://arxiv.org/abs/2112.10752>

- (SD) Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, Björn Ommer: **High-Resolution Image Synthesis with Latent Diffusion Models**  
<https://arxiv.org/abs/2112.10752>
- (SDXL) Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, Robin Rombach: **SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis** <https://arxiv.org/abs/2307.01952>
- (SD3) Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, Robin Rombach: **Scaling Rectified Flow Transformers for High-Resolution Image Synthesis**  
<https://arxiv.org/abs/2403.03206>
- (SD3-Turbo) Axel Sauer, Frederic Boesel, Tim Dockhorn, Andreas Blattmann, Patrick Esser, Robin Rombach: **Fast High-Resolution Image Synthesis with Latent Adversarial Diffusion Distillation** <https://arxiv.org/pdf/2403.12015>



# Bonus Content

## DDPM Loss

# DDPM – Deriving Loss using Jensen's Inequality

$$\begin{aligned}
 -\mathbb{E}_{q(\mathbf{x}_0)} [\log p_\theta(\mathbf{x}_0)] &= -\mathbb{E}_{q(\mathbf{x}_0)} [\log \mathbb{E}_{p_\theta(\mathbf{x}_{1:T})} [p_\theta(\mathbf{x}_0 | \mathbf{x}_{1:T})]] \\
 &= -\mathbb{E}_{q(\mathbf{x}_0)} \left[ \log \mathbb{E}_{q(\mathbf{x}_{1:T} | \mathbf{x}_0)} \left[ \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} | \mathbf{x}_0)} \right] \right] \\
 &\leq -\mathbb{E}_{q(\mathbf{x}_{0:T})} \left[ \log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} | \mathbf{x}_0)} \right] = \mathbb{E}_{q(\mathbf{x}_{0:T})} \left[ \log \frac{q(\mathbf{x}_{1:T} | \mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})} \right] \\
 &= \mathbb{E}_{q(\mathbf{x}_{0:T})} \left[ -\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{q(\mathbf{x}_t | \mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)} + \log \frac{q(\mathbf{x}_1 | \mathbf{x}_0)}{p_\theta(\mathbf{x}_0 | \mathbf{x}_1)} \right] \\
 &= \mathbb{E}_{q(\mathbf{x}_{0:T})} \left[ -\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \left( \frac{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)} \frac{q(\mathbf{x}_t | \mathbf{x}_0)}{q(\mathbf{x}_{t-1} | \mathbf{x}_0)} \right) + \log \frac{q(\mathbf{x}_1 | \mathbf{x}_0)}{p_\theta(\mathbf{x}_0 | \mathbf{x}_1)} \right] \\
 &= \mathbb{E}_{q(\mathbf{x}_{0:T})} \left[ -\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)} + \log \frac{q(\mathbf{x}_T | \mathbf{x}_0)}{q(\mathbf{x}_1 | \mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1 | \mathbf{x}_0)}{p_\theta(\mathbf{x}_0 | \mathbf{x}_1)} \right] \\
 &= \mathbb{E}_{q(\mathbf{x}_{0:T})} \left[ \log \frac{q(\mathbf{x}_T | \mathbf{x}_0)}{p_\theta(\mathbf{x}_T)} + \sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)} - \log p_\theta(\mathbf{x}_0 | \mathbf{x}_1) \right] \\
 &= \mathbb{E}_{q(\mathbf{x}_{0:T})} \left[ \underbrace{D_{\text{KL}}(q(\mathbf{x}_T | \mathbf{x}_0) \| p_\theta(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^T \underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t))}_{L_t} - \underbrace{\log p_\theta(\mathbf{x}_0 | \mathbf{x}_1)}_{L_0} \right]
 \end{aligned}$$

# DDPM – Deriving Loss using Jensen's Inequality

The whole loss is therefore composed of the following components:

- $L_T = D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)||p_{\theta}(\mathbf{x}_T))$  is constant with respect to  $\theta$  and can be ignored,
- $L_t = D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t))$  is KL divergence between two Gaussians, so it can be computed explicitly as

$$L_t = \mathbb{E} \left[ \frac{1}{2\|\sigma_t \mathbf{I}\|^2} \left\| \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_{\theta}(\mathbf{x}_t, t) \right\|^2 \right],$$

- $L_0 = -\log p_{\theta}(\mathbf{x}_0|\mathbf{x}_1)$  can be used to generate discrete  $\mathbf{x}_0$  from the continuous  $\mathbf{x}_1$ ; we will ignore it in the slides for simplicity.

# DDPM – Reparametrizing Model Prediction

Recall that  $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\boldsymbol{\beta}}_t \mathbf{I})$  for

$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\mathbf{x}_t,$$

$$\tilde{\boldsymbol{\beta}}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\beta_t.$$

Because  $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\mathbf{e}_t$ , we get  $\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t}\mathbf{e}_t)$ .

Substituting  $\mathbf{x}_0$  to  $\tilde{\boldsymbol{\mu}}_t$ , we get

$$\begin{aligned}\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) &= \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t} \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t}\mathbf{e}_t) + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\mathbf{x}_t \\ &= \left( \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t} \frac{1}{\sqrt{\bar{\alpha}_t}} + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \right) \mathbf{x}_t - \left( \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t} \frac{\sqrt{1 - \bar{\alpha}_t}}{\sqrt{\bar{\alpha}_t}} \right) \mathbf{e}_t \\ &= \frac{\beta_t + \alpha_t(1 - \bar{\alpha}_{t-1})}{(1 - \bar{\alpha}_t)\sqrt{\alpha_t}} \mathbf{x}_t - \left( \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}\sqrt{\alpha_t}} \right) \mathbf{e}_t = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \mathbf{e}_t \right).\end{aligned}$$

# DDPM – Reparametrizing Model Prediction

We change our model to predict  $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)$  instead of  $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$ . The loss  $L_t$  then becomes

$$\begin{aligned}
 L_t &= \mathbb{E} \left[ \frac{1}{2 \|\sigma_t \mathbf{I}\|^2} \left\| \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t) \right\|^2 \right] \\
 &= \mathbb{E} \left[ \frac{1}{2 \|\sigma_t \mathbf{I}\|^2} \left\| \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \mathbf{e}_t \right) - \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t) \right) \right\|^2 \right] \\
 &= \mathbb{E} \left[ \frac{(1 - \alpha_t)^2}{2 \alpha_t (1 - \bar{\alpha}_t) \|\sigma_t \mathbf{I}\|^2} \left\| \mathbf{e}_t - \boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t) \right\|^2 \right] \\
 &= \mathbb{E} \left[ \frac{(1 - \alpha_t)^2}{2 \alpha_t (1 - \bar{\alpha}_t) \|\sigma_t \mathbf{I}\|^2} \left\| \mathbf{e}_t - \boldsymbol{\varepsilon}_\theta(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \mathbf{e}_t, t) \right\|^2 \right].
 \end{aligned}$$

The authors found that training without the weighting term performs better, so the final loss is

$$L_t^{\text{simple}} = \mathbb{E}_{t \in \{1..T\}, \mathbf{x}_0, \mathbf{e}_t} \left[ \left\| \mathbf{e}_t - \boldsymbol{\varepsilon}_\theta(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \mathbf{e}_t, t) \right\|^2 \right].$$

Note that both losses have the same optimum if we used independent  $\boldsymbol{\varepsilon}_{\theta_t}$  for every  $t$ .

# Bonus Content

## DDIM Sampling



We now describe *Denoising Diffusion Implicit Models (DDIM)*, which utilize a different forward process.

This forward process is designed to:

- allow faster sampling,
- have the same “marginals”  $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I})$ .

The second condition will allow us to use the same loss as in DDPM; therefore, the training algorithm is exactly identical to DDPM, only the sampling algorithm is different.

Note that in the slides, only a special case of DDIM is described; the original paper describes a more general forward process. However, the special case presented here is almost exclusively used.

The forward process of DDIM can be described using

$$q_0(\mathbf{x}_{1:T}|\mathbf{x}_0) = q_0(\mathbf{x}_T|\mathbf{x}_0) \prod_{t=2}^T q_0(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0),$$

where

- $q_0(\mathbf{x}_T|\mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_T}\mathbf{x}_0, (1 - \bar{\alpha}_T)\mathbf{I}),$
- $q_0(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1}}\left(\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1 - \bar{\alpha}_t}}\right), 0 \cdot \mathbf{I}\right).$

With these definitions, we can prove by induction that  $q_0(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I})$ :

$$\begin{aligned} \mathbf{x}_{t-1} &= \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1}}\left(\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1 - \bar{\alpha}_t}}\right) \\ &= \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1}}\left(\frac{\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\mathbf{e}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1 - \bar{\alpha}_t}}\right) = \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1}}\mathbf{e}_t. \end{aligned}$$

The real “forward”  $q_0(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)$  can be expressed using Bayes' theorem using the above definition, but we do not actually need it.

# Denoising Diffusion Implicit Models – The Reverse Process

The definition of  $q_0(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$  provides us also with a sampling algorithm; after sampling the initial noise  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , we perform the following for  $t$  from  $T$  down to 1:

$$\begin{aligned}\mathbf{x}_{t-1} &= \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1}}\boldsymbol{\varepsilon}_{\theta}(\mathbf{x}_t, t) \\ &= \sqrt{\bar{\alpha}_{t-1}}\left(\frac{\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\varepsilon}_{\theta}(\mathbf{x}_t, t)}{\sqrt{\bar{\alpha}_t}}\right) + \sqrt{1 - \bar{\alpha}_{t-1}}\boldsymbol{\varepsilon}_{\theta}(\mathbf{x}_t, t).\end{aligned}$$

An important property of  $q_0$  is that it can also model several steps at once:

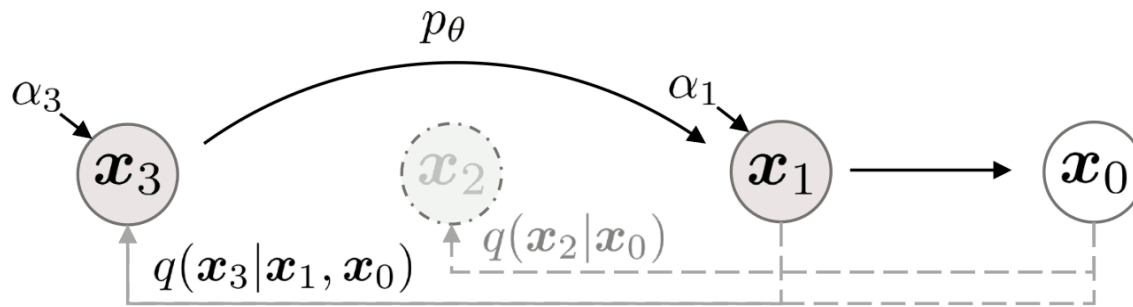


Figure 2 of "Denoising Diffusion Implicit Models", <https://arxiv.org/abs/2010.02502>

$$q_0(\mathbf{x}_{t'}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_{t'}}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{t'}}\left(\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1 - \bar{\alpha}_t}}\right), \mathbf{0}\right).$$

We base our accelerated sampling algorithm on the “multistep”  $q_0(\mathbf{x}_{t'} | \mathbf{x}_t, \mathbf{x}_0)$ .

Let  $t_S = T, t_{S-1}, \dots, t_1$  be a subsequence of the process steps (usually, a uniform subsequence of  $T, \dots, 1$  is used), and let  $t_0 = 0$ . Starting from initial noise  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , we perform  $S$  sampling steps for  $i$  from  $S$  down to 1:

$$\mathbf{x}_{t_{i-1}} \leftarrow \sqrt{\bar{\alpha}_{t_{i-1}}} \underbrace{\left( \frac{\mathbf{x}_{t_i} - \sqrt{1 - \bar{\alpha}_{t_i}} \boldsymbol{\varepsilon}_{\theta}(\mathbf{x}_{t_i}, t_i)}{\sqrt{\bar{\alpha}_{t_i}}} \right)}_{\mathbf{x}_0 \text{ estimate}} + \sqrt{1 - \bar{\alpha}_{t_{i-1}}} \boldsymbol{\varepsilon}_{\theta}(\mathbf{x}_{t_i}, t_i).$$

The sampling procedure can be described in words as follows:

- using the current time step  $t_i$ , we compute the estimated noise  $\boldsymbol{\varepsilon}_{\theta}(\mathbf{x}_{t_i}, t_i)$ ;
- by utilizing the current signal rate  $\sqrt{\bar{\alpha}_{t_i}}$  and noise rate  $\sqrt{1 - \bar{\alpha}_{t_i}}$ , we estimate  $\mathbf{x}_0$ ;
- we obtain  $\mathbf{x}_{t_{i-1}}$  by combining the estimated signal  $\mathbf{x}_0$  and noise  $\boldsymbol{\varepsilon}_{\theta}(\mathbf{x}_{t_i}, t_i)$  using the signal and noise rates of the time step  $t_{i-1}$ .

For comparison, we show both the original **DDPM** and the new **DDIM** sampling algorithms:

- sample  $\mathbf{x}_T$  from  $\mathcal{N}(\mathbf{0}, \mathbf{I})$
- let  $t_S = T, t_{S-1}, \dots, t_1 = 1$  be a subsequence of the process steps
  - **DDPM**: the original sequence  $T, \dots, 1$  is usually used
  - **DDIM**:  $S$  regularly-spaced steps  $T, \frac{S-1}{S}T, \frac{S-2}{S}T, \dots, 1$  are usually used
  - additionally, we define  $t_0 = 0$
- for  $i = S, \dots, 1$ :

**DDPM** :

$$\mathbf{x}_{t_{i-1}} \leftarrow \sqrt{\frac{1}{\alpha_{t_i}}} \left( \mathbf{x}_{t_i} - \frac{1 - \alpha_{t_i}}{\sqrt{1 - \bar{\alpha}_{t_i}}} \boldsymbol{\epsilon}_{\theta}(\mathbf{x}_{t_i}, t_i) \right) + \sigma_t \mathbf{z}_t$$

**DDIM** :

$$\mathbf{x}_{t_{i-1}} \leftarrow \underbrace{\sqrt{\bar{\alpha}_{t_{i-1}}} \left( \frac{\mathbf{x}_{t_i} - \sqrt{1 - \bar{\alpha}_{t_i}} \boldsymbol{\epsilon}_{\theta}(\mathbf{x}_{t_i}, t_i)}{\sqrt{\bar{\alpha}_{t_i}}} \right)}_{\mathbf{x}_0 \text{ estimate}} + \sqrt{1 - \bar{\alpha}_{t_{i-1}}} \boldsymbol{\epsilon}_{\theta}(\mathbf{x}_{t_i}, t_i)$$

- return  $\mathbf{x}_0$

# DDIM – Accelerated Sampling Examples

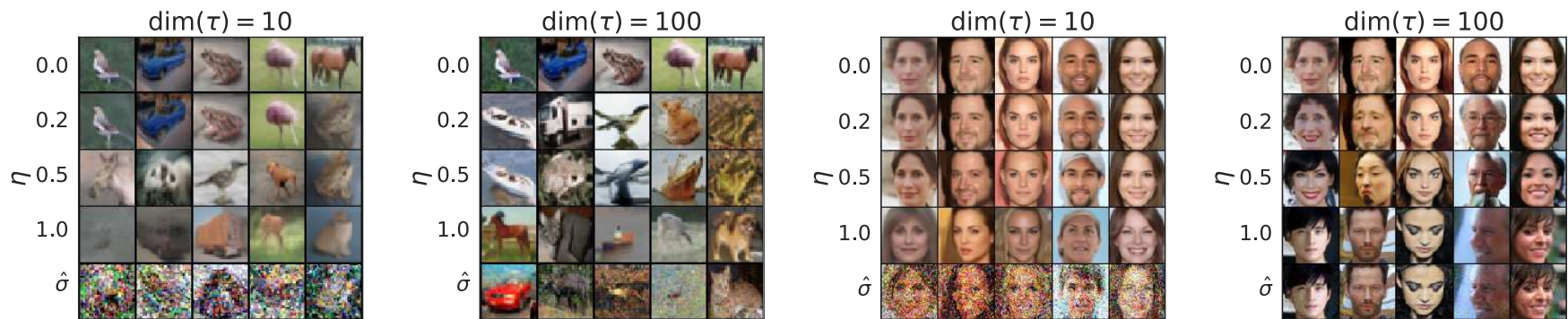


Figure 3 of "Denoising Diffusion Implicit Models", <https://arxiv.org/abs/2010.02502>



Figure 5 of "Denoising Diffusion Implicit Models", <https://arxiv.org/abs/2010.02502>



# Bonus Content Score Matching

Recall that loglikelihood-based models explicitly represent the density function, commonly using an unnormalized probabilistic model

$$p_{\theta}(\mathbf{x}) = \frac{e^{f_{\theta}(\mathbf{x})}}{Z_{\theta}},$$

and it is troublesome to ensure the tractability of the normalization constant  $Z_{\theta}$ .

One way how to avoid the normalization is to avoid the explicit density  $p_{\theta}(\mathbf{x})$ , and represent a **score function** instead, where the score function is the gradient of the log density:

$$s_{\theta}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}),$$

because

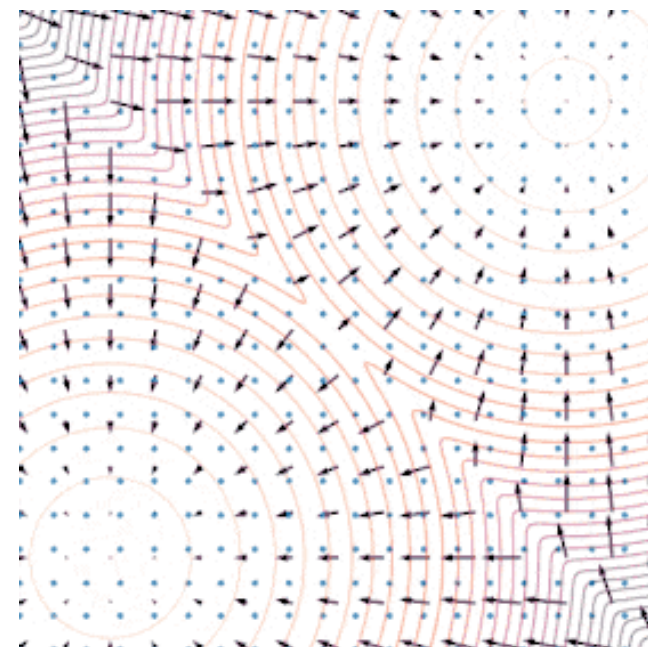
$$s_{\theta}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}) = \nabla_{\mathbf{x}} \log \frac{e^{f_{\theta}(\mathbf{x})}}{Z_{\theta}} = \nabla_{\mathbf{x}} f_{\theta}(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z_{\theta}}_0 = \nabla_{\mathbf{x}} f_{\theta}(\mathbf{x}).$$

# Langevin Dynamics

When we have a score function  $\nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x})$ , we can use it to perform sampling from the distribution  $p_{\theta}(\mathbf{x})$  by using **Langevin dynamics**, which is an algorithm akin to SGD, but performing sampling instead of optimum finding. Starting with  $\mathbf{x}_0$ , we iteratively set

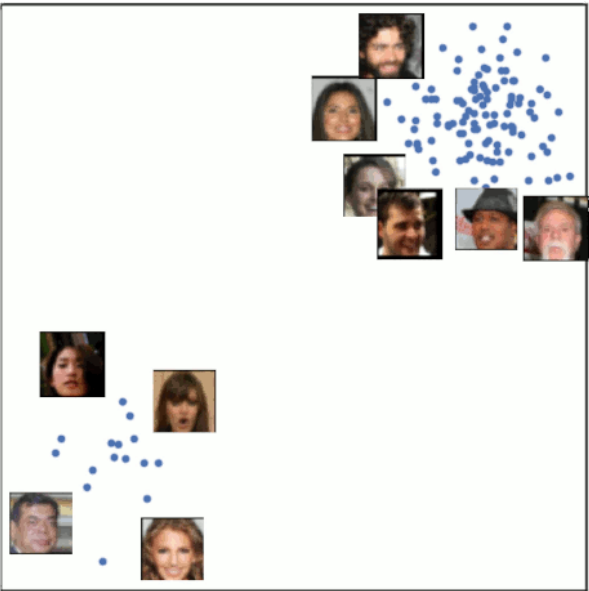
$$\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \varepsilon \nabla_{\mathbf{x}_i} \log p_{\theta}(\mathbf{x}_i) + \sqrt{2\varepsilon} \mathbf{z}_i, \text{ where } \mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

When  $\varepsilon \rightarrow 0$  and  $K \rightarrow \infty$ ,  $\mathbf{x}_K$  obtained by the Langevin dynamics converges to a sample from the distribution  $p_{\theta}(\mathbf{x})$ .



<https://yang-song.net/assets/img/score/langevin.gif>

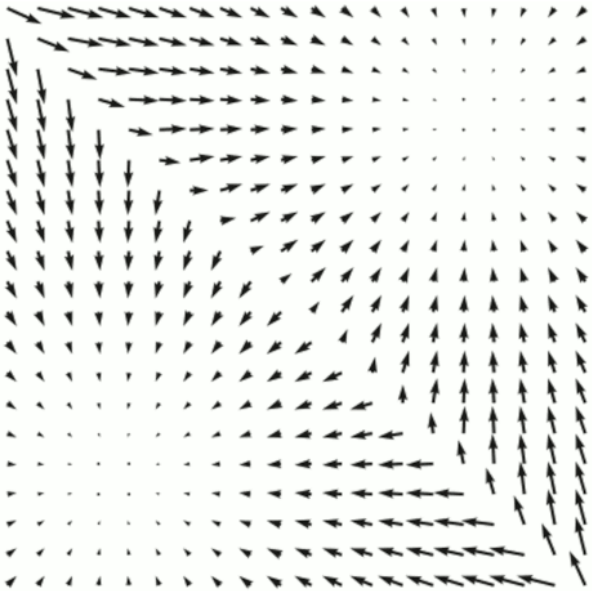
# Score-Based Generative Modeling



Data samples

$$\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\} \stackrel{\text{i.i.d.}}{\sim} p(\mathbf{x})$$

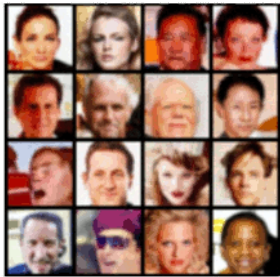
score  
matching



Scores

$$\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$$

Langevin  
dynamics

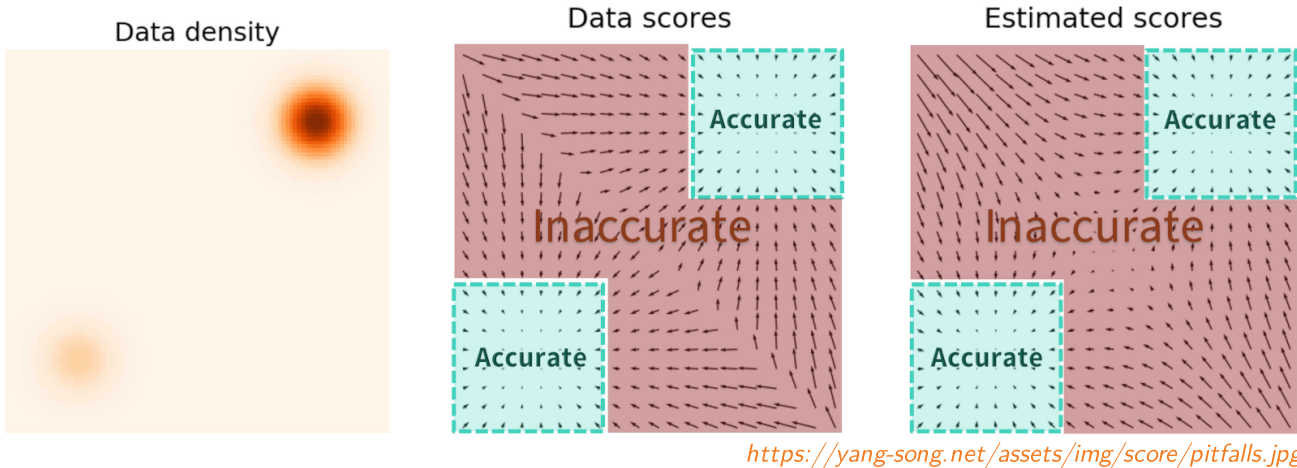


New samples

<https://yang-song.net/assets/img/score/smlld.jpg>

# Noise Conditional Score Network

However, estimating the score function from data is inaccurate in low-density regions.



In order to accurately estimate the score function in low-density regions, we perturb the data distribution by isotropic Gaussian noise with various noise rates  $\sigma_t$ :

$$q_{\sigma_t}(\tilde{\mathbf{x}}) \stackrel{\text{def}}{=} \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})} [\mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma_t^2 \mathbf{I})],$$

where the noise distribution  $q_{\sigma_t}(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma_t^2 \mathbf{I})$  as analogous to the forward process in the diffusion models.

# Noise Conditional Score Network

To train the score function  $\mathbf{s}_\theta(\mathbf{x}, \sigma_t) = \nabla_{\mathbf{x}} \log q_{\sigma_t}(\mathbf{x})$ , we need to minimize the following objective:

$$\mathbb{E}_{t, \tilde{\mathbf{x}} \sim q_{\sigma_t}} \left[ \left\| \mathbf{s}_\theta(\tilde{\mathbf{x}}, \sigma_t) - \nabla_{\tilde{\mathbf{x}}} \log q_{\sigma_t}(\tilde{\mathbf{x}}) \right\|^2 \right].$$

It can be shown (see *P. Vincent: A connection between score matching and denoising autoencoders*) that it is equivalent to minimize the *denoising score matching* objective:

$$\mathbb{E}_{t, \mathbf{x} \sim p(\mathbf{x}), \tilde{\mathbf{x}} \sim q_{\sigma_t}(\tilde{\mathbf{x}}|\mathbf{x})} \left[ \left\| \mathbf{s}_\theta(\tilde{\mathbf{x}}, \sigma_t) - \nabla_{\tilde{\mathbf{x}}} \log q_{\sigma_t}(\tilde{\mathbf{x}}|\mathbf{x}) \right\|^2 \right].$$

In our case,  $\nabla_{\tilde{\mathbf{x}}} \log q_{\sigma_t}(\tilde{\mathbf{x}}|\mathbf{x}) = \nabla_{\tilde{\mathbf{x}}} \frac{-\|\tilde{\mathbf{x}} - \mathbf{x}\|^2}{2\sigma_t^2} = -\frac{\tilde{\mathbf{x}} - \mathbf{x}}{\sigma_t^2}$ . Because  $\tilde{\mathbf{x}} = \mathbf{x} + \sigma_t \mathbf{e}$  for standard normal random variable  $\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , we can rewrite the objective to

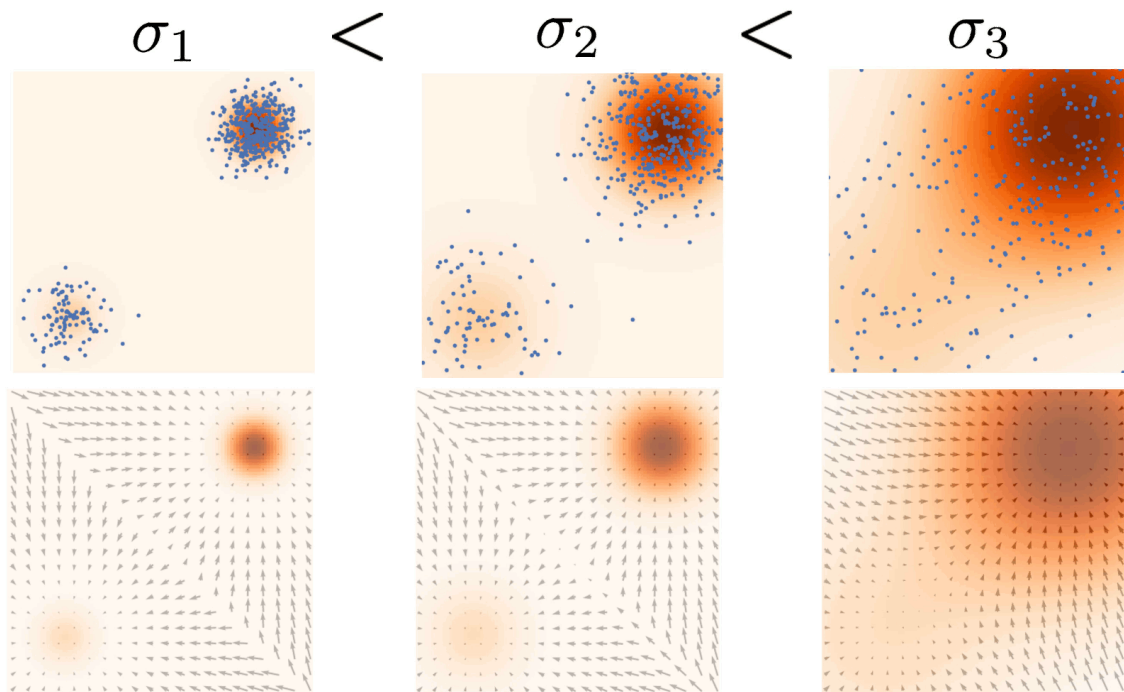
$$\mathbb{E}_{t, \mathbf{x} \sim p(\mathbf{x}), \mathbf{e} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[ \left\| \mathbf{s}_\theta(\mathbf{x} + \sigma_t \mathbf{e}, \sigma_t) - \frac{-\mathbf{e}}{\sigma_t} \right\|^2 \right],$$

so the score function basically estimates the noise given a noised image.



# Noise Conditional Score Network

Once we have trained the score function for various noise rates  $\sigma_t$ , we can sample using annealed Langevin dynamics, where we utilize using gradually smaller noise rates  $\sigma_t$ .



[https://yang-song.net/assets/img/score/multi\\_scale.jpg](https://yang-song.net/assets/img/score/multi_scale.jpg)

---

**Algorithm 1** Annealed Langevin dynamics.

---

**Require:**  $\{\sigma_i\}_{i=1}^L, \epsilon, T$ .

```

1: Initialize  $\tilde{\mathbf{x}}_0$ 
2: for  $i \leftarrow 1$  to  $L$  do
3:    $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$   $\triangleright \alpha_i$  is the step size.
4:   for  $t \leftarrow 1$  to  $T$  do
5:     Draw  $\mathbf{z}_t \sim \mathcal{N}(0, I)$ 
6:      $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2} \mathbf{s}_\theta(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i} \mathbf{z}_t$ 
7:   end for
8:    $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$ 
9: end for
return  $\tilde{\mathbf{x}}_T$ 
```

---

*Algorithm 1 of "Generative Modeling by Estimating Gradients of the Data Distribution", <https://arxiv.org/abs/1907.05600>*

Such a procedure is reminiscent to the reverse diffusion process sampling.

# Bonus Content Further Reading

- Martin Arjovsky, Soumith Chintala, Léon Bottou: **Wasserstein GAN**  
<https://arxiv.org/abs/1701.07875>
- Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville: **Improved Training of Wasserstein GANs** <https://arxiv.org/abs/1704.00028>
- Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen: **Progressive Growing of GANs for Improved Quality, Stability, and Variation** <https://arxiv.org/abs/1710.10196>
- Takeru Miyato, Toshiki Kataoka, Masanori Koyama, Yuichi Yoshida: **Spectral Normalization for Generative Adversarial Networks** <https://arxiv.org/abs/1802.05957>
- Zhiming Zhou, Yuxuan Song, Lantao Yu, Hongwei Wang, Jiadong Liang, Weinan Zhang, Zhihua Zhang, Yong Yu: **Understanding the Effectiveness of Lipschitz-Continuity in Generative Adversarial Nets** <https://arxiv.org/abs/1807.00751>
- Andrew Brock, Jeff Donahue, Karen Simonyan: **Large Scale GAN Training for High Fidelity Natural Image Synthesis** <https://arxiv.org/abs/1809.11096>
- Tero Karras, Samuli Laine, Timo Aila: **A Style-Based Generator Architecture for Generative Adversarial Networks** <https://arxiv.org/abs/1812.04948>



Figure 1 of "Large Scale GAN Training for High Fidelity Natural Image Synthesis", <https://arxiv.org/abs/1809.11096>



Figure 2 of "Large Scale GAN Training for High Fidelity Natural Image Synthesis", <https://arxiv.org/abs/1809.11096>





(a)

(b)

Figure 7 of "Large Scale GAN Training for High Fidelity Natural Image Synthesis", <https://arxiv.org/abs/1809.11096>

- Aaron van den Oord, Oriol Vinyals, Koray Kavukcuoglu: **Neural Discrete Representation Learning** <https://arxiv.org/abs/1711.00937>
- Ali Razavi, Aaron van den Oord, Oriol Vinyals: **Generating Diverse High-Fidelity Images with VQ-VAE-2** <https://arxiv.org/abs/1906.00446>
- Patrick Esser, Robin Rombach, Björn Ommer: **Taming Transformers for High-Resolution Image Synthesis** <https://arxiv.org/abs/2012.09841>
- Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, Ilya Sutskever: **Zero-Shot Text-to-Image Generation** <https://arxiv.org/abs/2102.12092>
- Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, Björn Ommer: **High-Resolution Image Synthesis with Latent Diffusion Models** <https://arxiv.org/abs/2112.10752>



- Yang Song, Stefano Ermon: **Generative Modeling by Estimating Gradients of the Data Distribution** <https://arxiv.org/abs/1907.05600>
- Jonathan Ho, Ajay Jain, Pieter Abbeel: **Denoising Diffusion Probabilistic Models** <https://arxiv.org/abs/2006.11239>
- Jiaming Song, Chenlin Meng, Stefano Ermon: **Denoising Diffusion Implicit Models** <https://arxiv.org/abs/2010.02502>
- Alex Nichol, Prafulla Dhariwal: **Improved Denoising Diffusion Probabilistic Models** <https://arxiv.org/abs/2102.09672>
- Prafulla Dhariwal, Alex Nichol: **Diffusion Models Beat GANs on Image Synthesis** <https://arxiv.org/abs/2105.05233>
- Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, Björn Ommer: **High-Resolution Image Synthesis with Latent Diffusion Models** <https://arxiv.org/abs/2112.10752>

- Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J. Fleet, M. Norouzi:  
**Image Super-Resolution via Iterative Refinement** <https://arxiv.org/abs/2104.07636>



- Alex Nichol et al.: **GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models** <https://arxiv.org/abs/2112.10741>



“a hedgehog using a calculator”



“a corgi wearing a red bowtie and a purple party hat”



“robots meditating in a vipassana retreat”



“a fall landscape with a small cottage next to a lake”



“a surrealist dream-like oil painting by salvador dalí of a cat playing checkers”



“a professional photo of a sunset behind the grand canyon”



“a high-quality oil painting of a psychedelic hamster dragon”



“an illustration of albert einstein wearing a superhero costume”

Figure 1 of “GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models”, <https://arxiv.org/abs/2112.10741>



“zebras roaming in the field”



“a girl hugging a corgi on a pedestal”



“a man with red hair”



“a vase of flowers”



“an old car in a snowy forest”



“a man wearing a white hat”

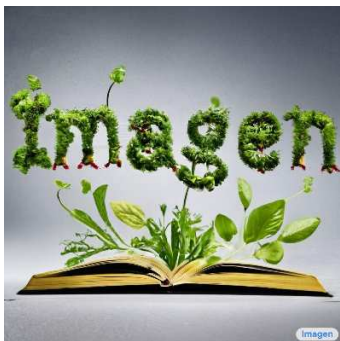
Figure 2 of "GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models", <https://arxiv.org/abs/2112.10741>



# Diffusion-Based Text-Conditional Image Generation

- Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, et al.: **Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding**

<https://arxiv.org/abs/2205.11487>



Sprouts in the shape of text 'Imagen' coming out of a fairytale book.



A photo of a Shiba Inu dog with a backpack riding a bike. It is wearing sunglasses and a beach hat.



A high contrast portrait of a very happy fuzzy panda dressed as a chef in a high end kitchen making dough. There is a painting of flowers on the wall behind him.



Teddy bears swimming at the Olympics 400m Butterfly event.



A cute corgi lives in a house made out of sushi.



A cute sloth holding a small treasure chest. A bright golden glow is coming from the chest.

Figure 1 of "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", <https://arxiv.org/abs/2205.11487>

- Laurent Dinh, David Krueger, Yoshua Bengio: **NICE: Non-linear Independent Components Estimation** <https://arxiv.org/abs/1410.8516>
- Laurent Dinh, Jascha Sohl-Dickstein, Samy Bengio: **Density estimation using Real NVP** <https://arxiv.org/abs/1605.08803>
- Diederik P. Kingma, Prafulla Dhariwal: **Glow: Generative Flow with Invertible 1x1 Convolutions** <https://arxiv.org/abs/1807.03039>



Figure 1 of "Glow: Generative Flow with Invertible 1x1 Convolutions", <https://arxiv.org/abs/1807.03039>