Generative Adversarial Networks, Diffusion Models

Milan Straka

📅 May 9, 2023

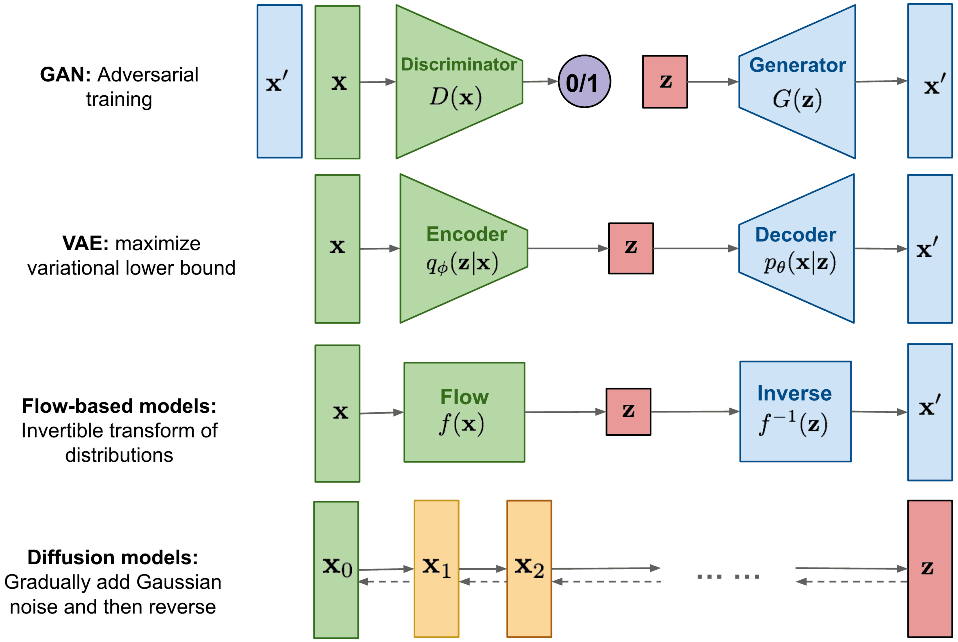
EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

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*):

$$P_{\boldsymbol\theta}(\mathbf{x}) = \frac{e^{f_{\boldsymbol\theta}(\mathbf{x})}}{Z_{\boldsymbol\theta}}.$$



GAN: Adversarial training
$\mathbf{x}'$
$\mathbf{x}$
Discriminator $D(\mathbf{x})$
0/1
$\mathbf{z}$
Generator $G(\mathbf{z})$
$\mathbf{x}'$
VAE: maximize variational lower bound
$\mathbf{x}$
Encoder $q_\phi(\mathbf{z}|\mathbf{x})$
$\mathbf{z}$
Decoder $p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$
Flow-based models: Invertible transform of distributions
$\mathbf{x}$
Flow $f(\mathbf{x})$
$\mathbf{z}$
Inverse $f^{-1}(\mathbf{z})$
$\mathbf{x}'$
Diffusion models: Gradually add Gaussian noise and then reverse
$\mathbf{x}_0$
$\mathbf{x}_1$
$\mathbf{x}_2$
... ...
$\mathbf{z}$

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

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

- We can compute $Z_{\boldsymbol\theta}$ 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).

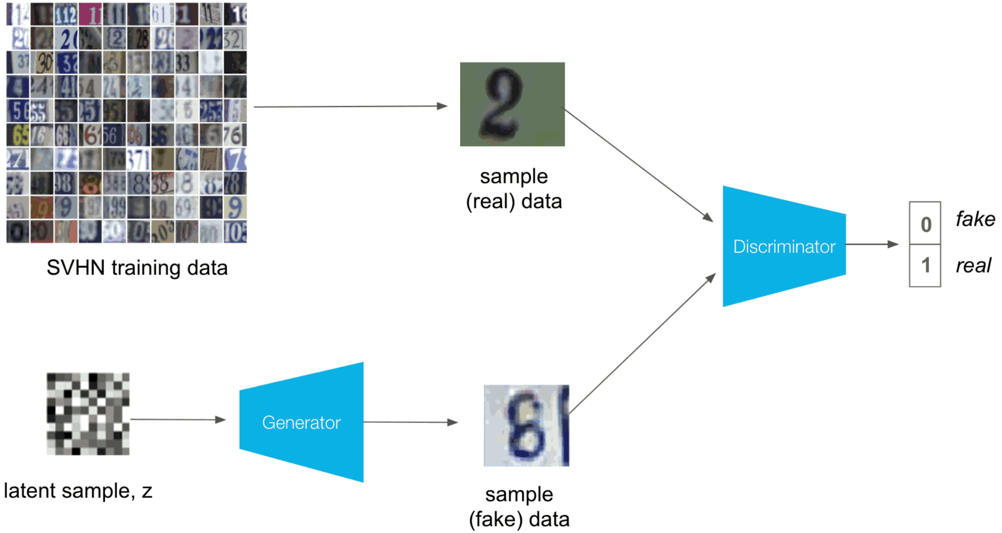
Generative Adversarial Networks

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

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

The discriminator and generator play the following game:

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



https://miro.medium.com/v2/1-ucVYsbDnwa2NM-f5qm_Yg.png*

Generative Adversarial Networks

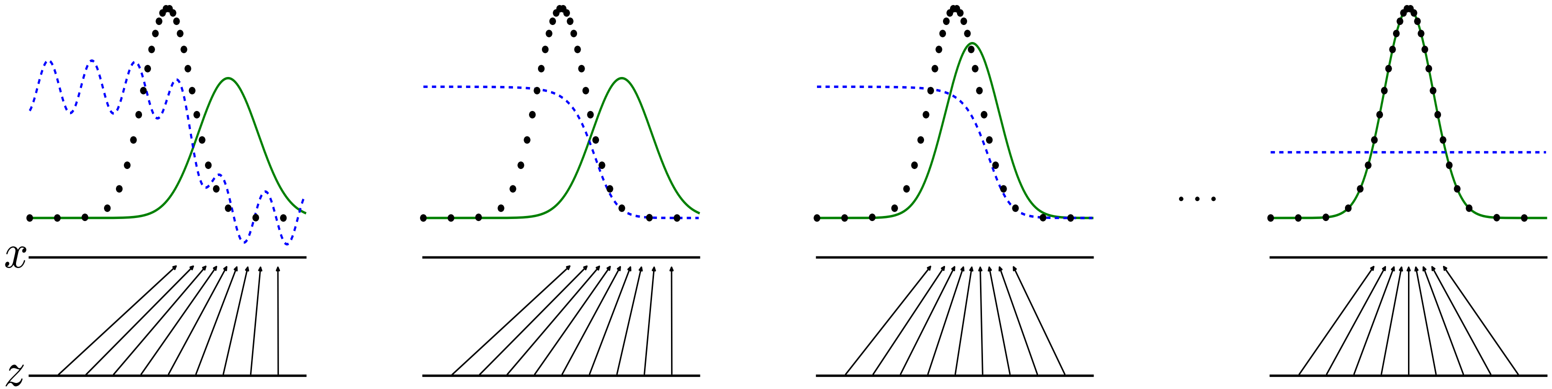


Figure 1 of "Generative Adversarial Nets", https://arxiv.org/abs/1406.2661

The generator and discriminator are alternately trained, the discriminator by

$$\underset{\boldsymbol\theta_d}{\arg\max}\, \mathbb{E}_{\boldsymbol x\sim P_\textrm{data}}\big[\log D(\boldsymbol x)\big] + \mathbb{E}_{\boldsymbol z\sim P(\mathbf z)}\big[\log(1 - D(G(\boldsymbol z)))\big]$$

and the generator by

$$\underset{\boldsymbol\theta_g}{\arg\min}\, \mathbb{E}_{\boldsymbol z\sim P(\mathbf z)}\big[\log(1 - D(G(\boldsymbol z)))\big].$$

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

Generative Adversarial Networks

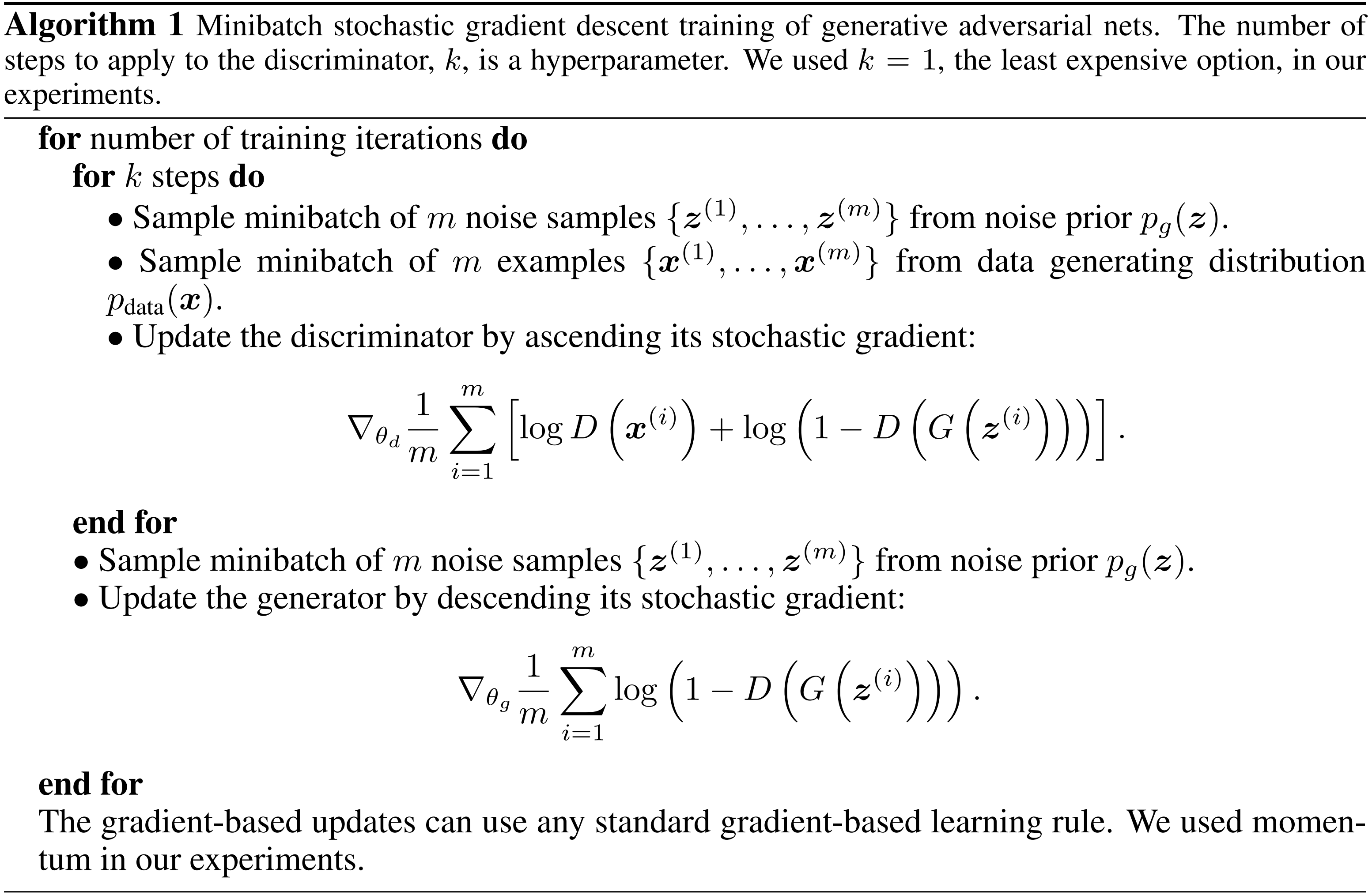
Because $\log(1 - D(G(\boldsymbol{z})))$ can saturate in the beginning of the training, where the discriminator can easily distinguish real and generated samples, the generator can be trained by

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

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).

Generative Adversarial Networks



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
        • Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
        • Update the discriminator by ascending its stochastic gradient:
            $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$
    end for
    • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Update the generator by descending its stochastic gradient:
        $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$
end for
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

Generative Adversarial Networks

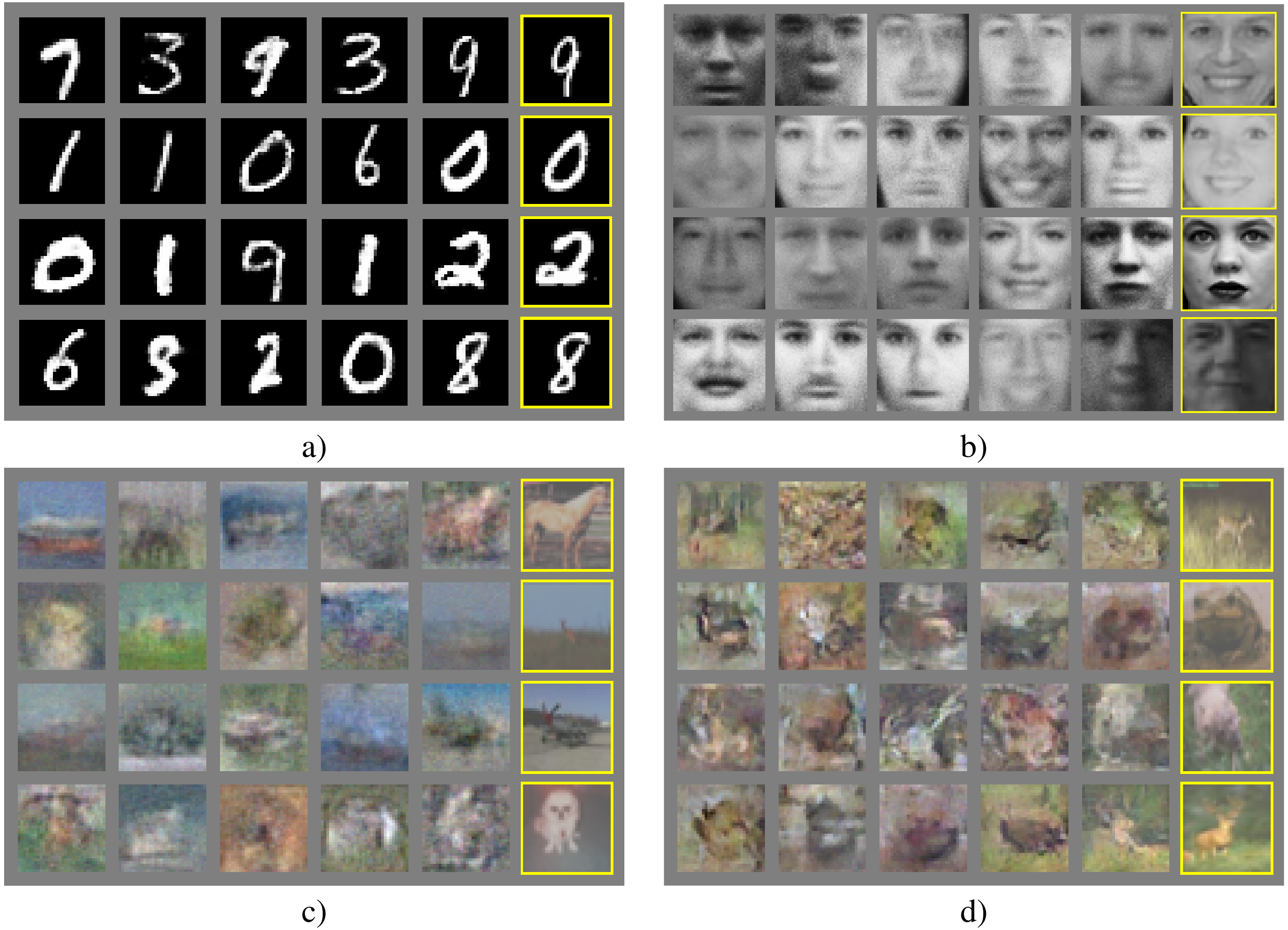


Figure 2 of "Generative Adversarial Nets", https://arxiv.org/abs/1406.2661

Conditional GAN

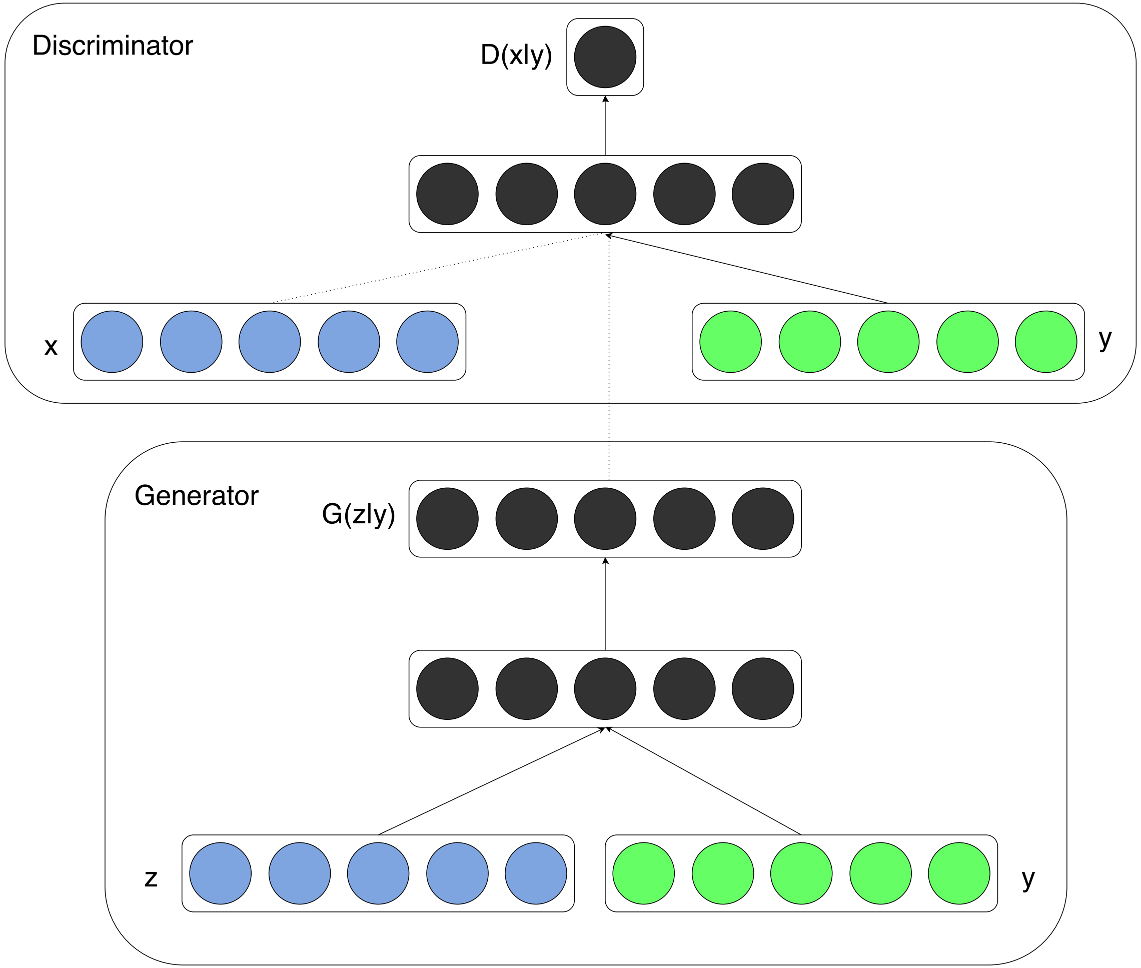


Figure 1 of "Conditional Generative Adversarial Nets", https://arxiv.org/abs/1411.1784

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.

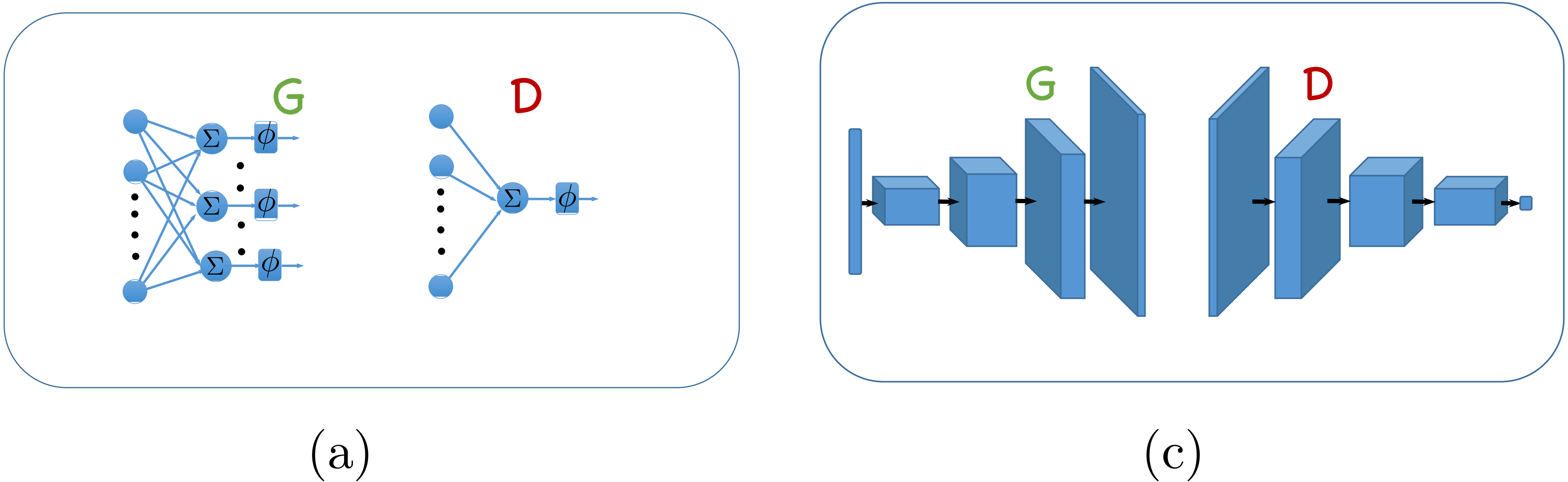


Figure 1 of "An Online Learning Approach to Generative Adversarial Networks", https://arxiv.org/abs/1706.03269

Deep Convolutional GAN

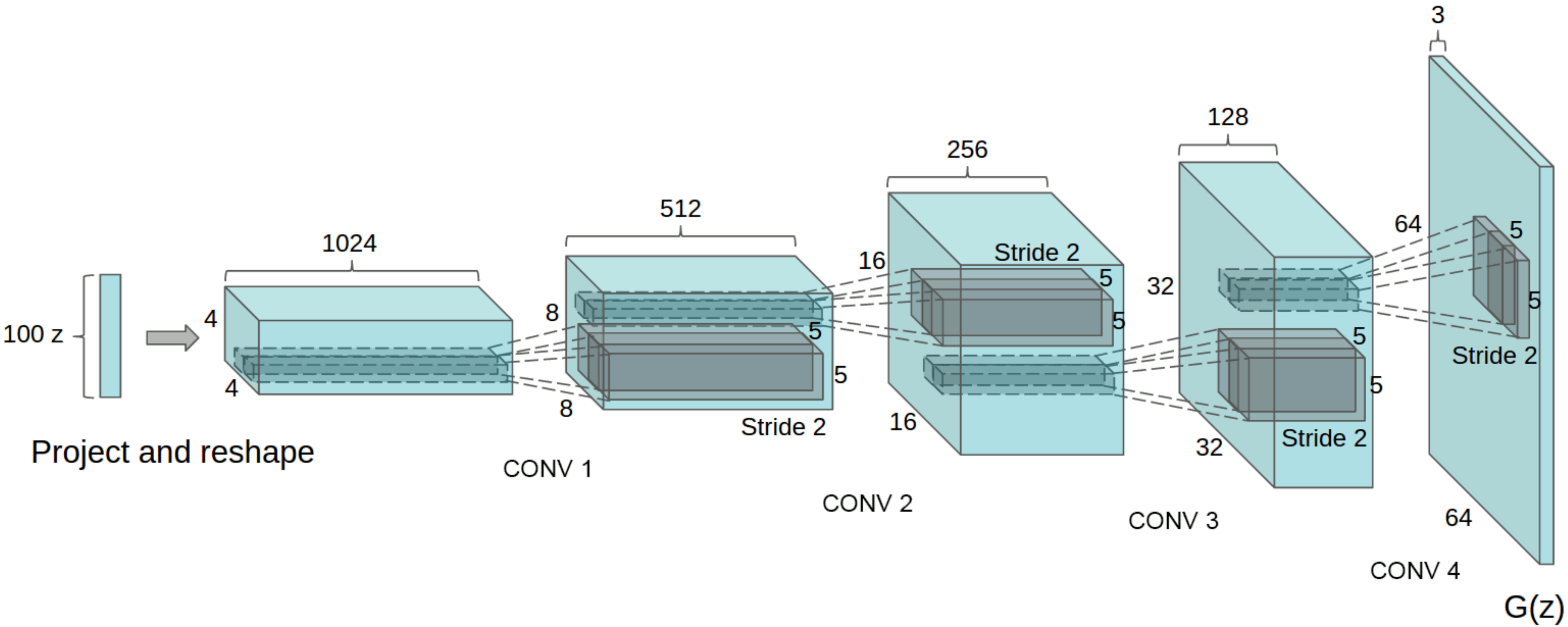


Figure 1 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434

Deep Convolutional GAN

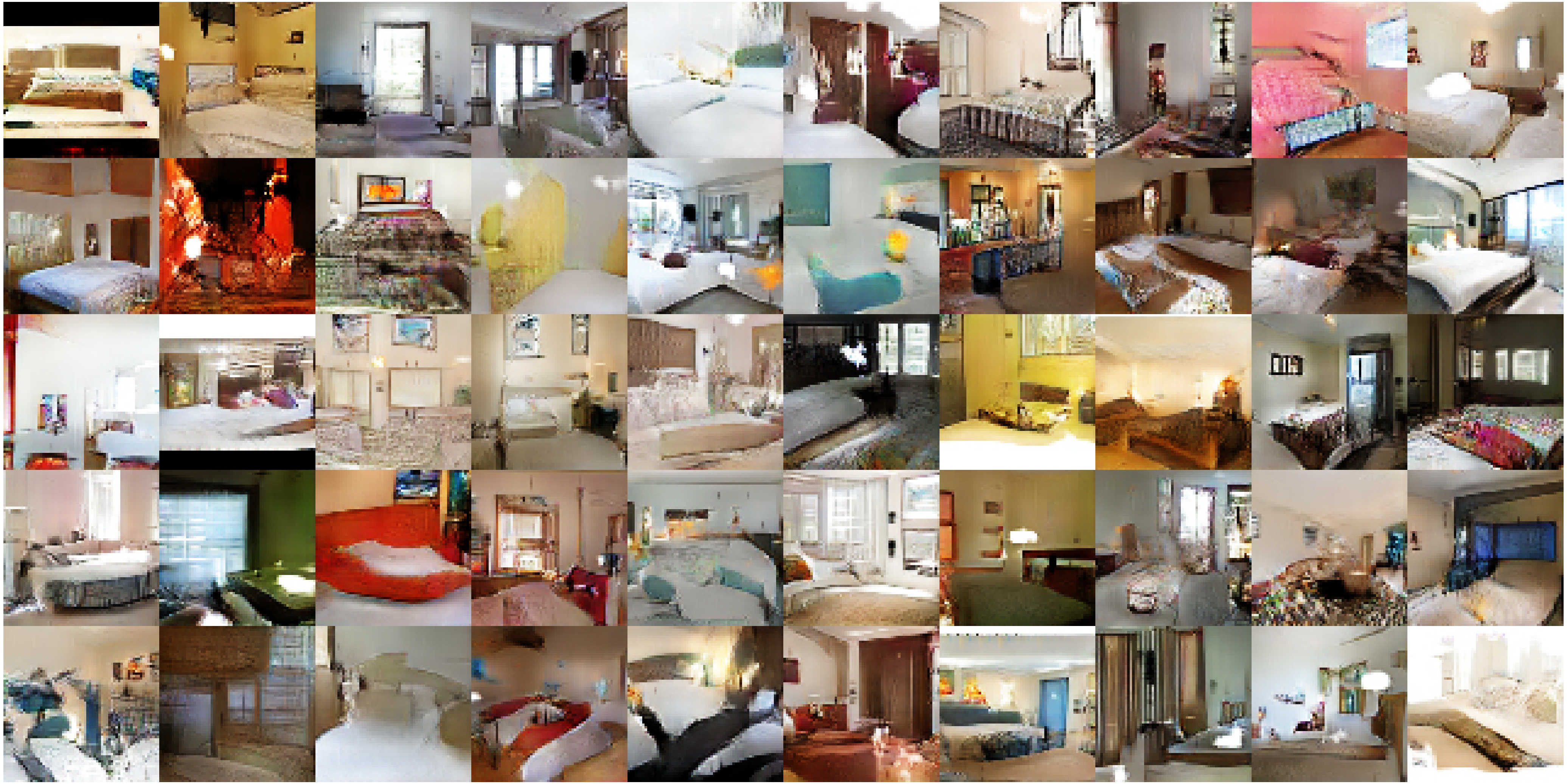


Figure 3 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434

Deep Convolutional GAN

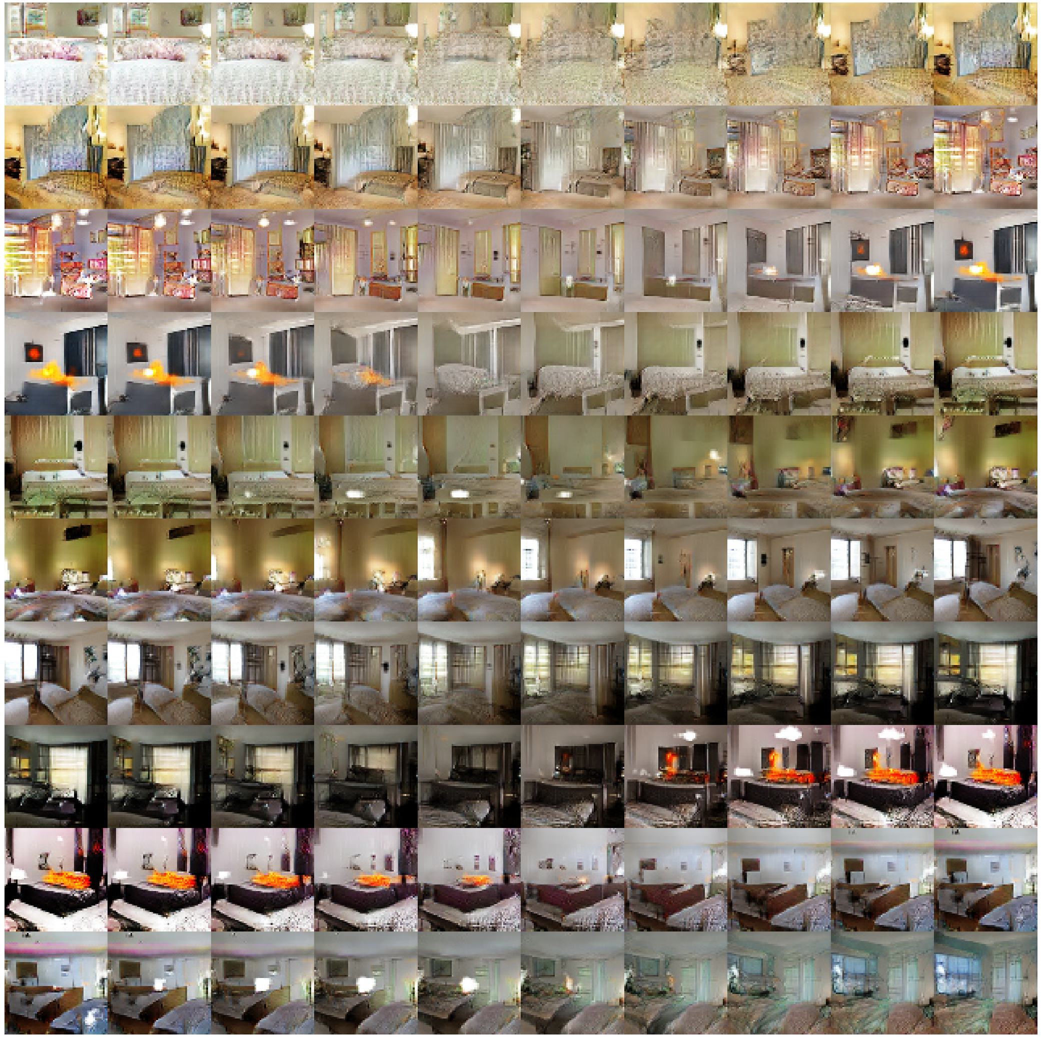


Figure 4 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434

Deep Convolutional GAN

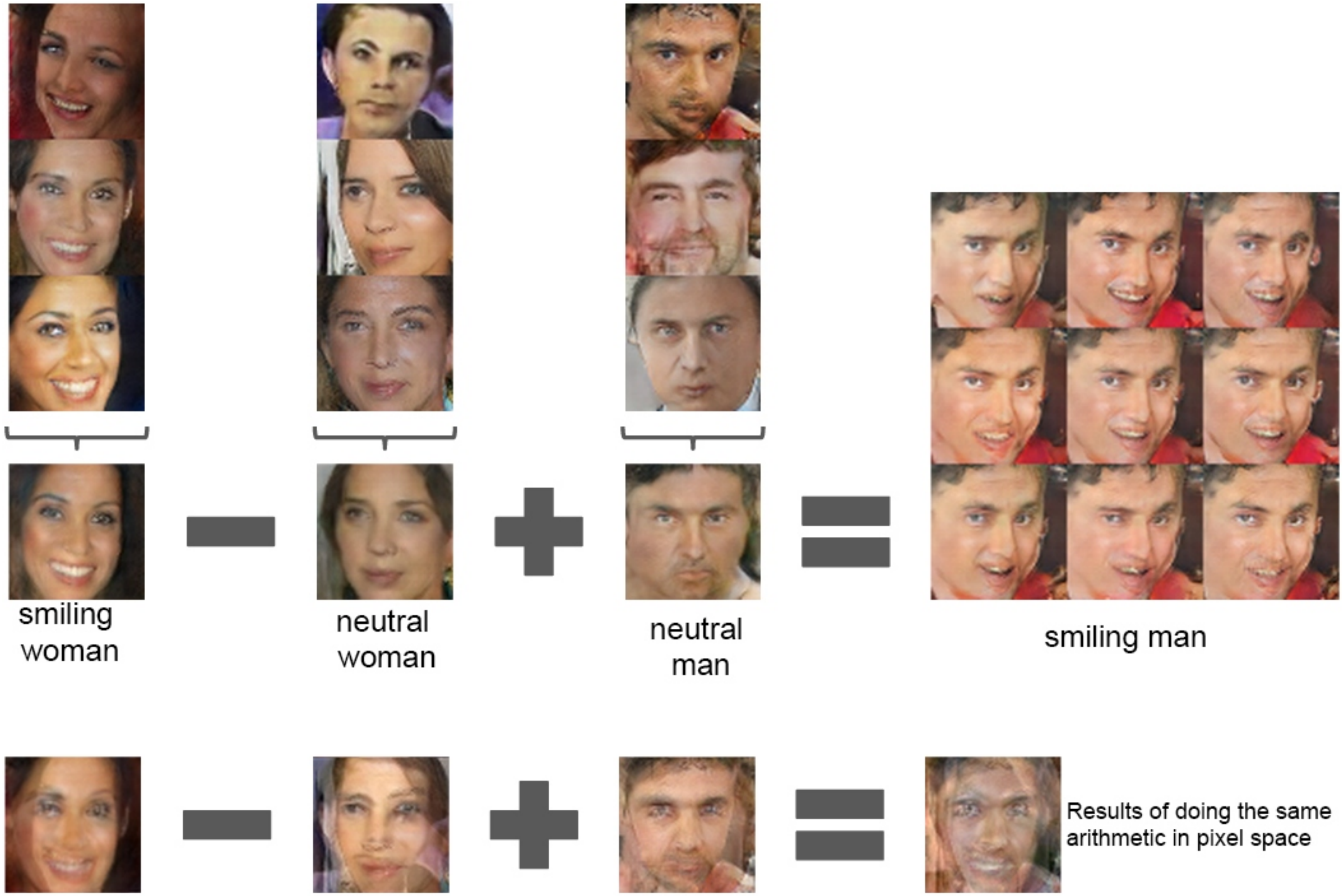


Figure 7 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434

Deep Convolutional GAN

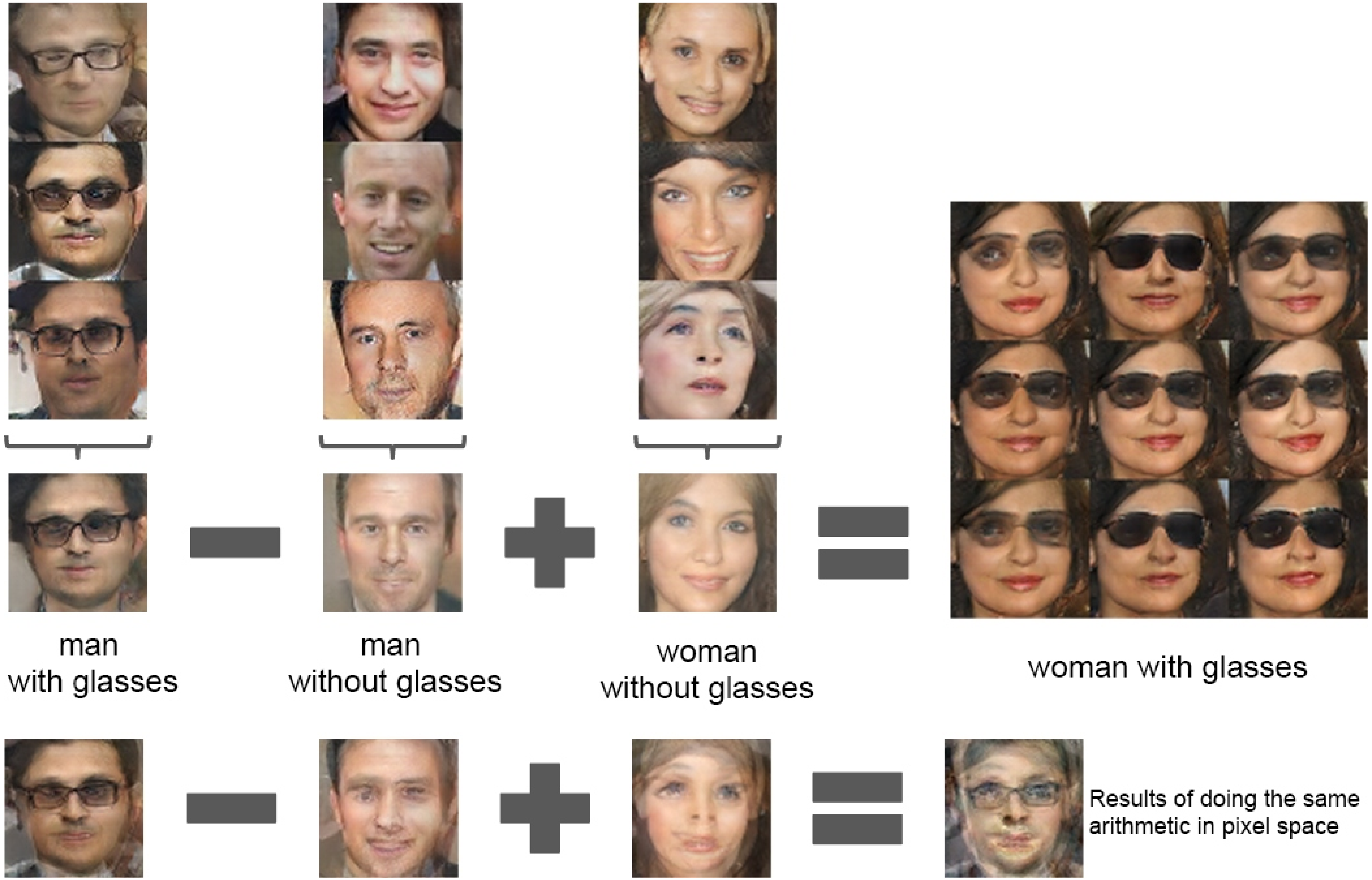


Figure 7 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434

Deep Convolutional GAN

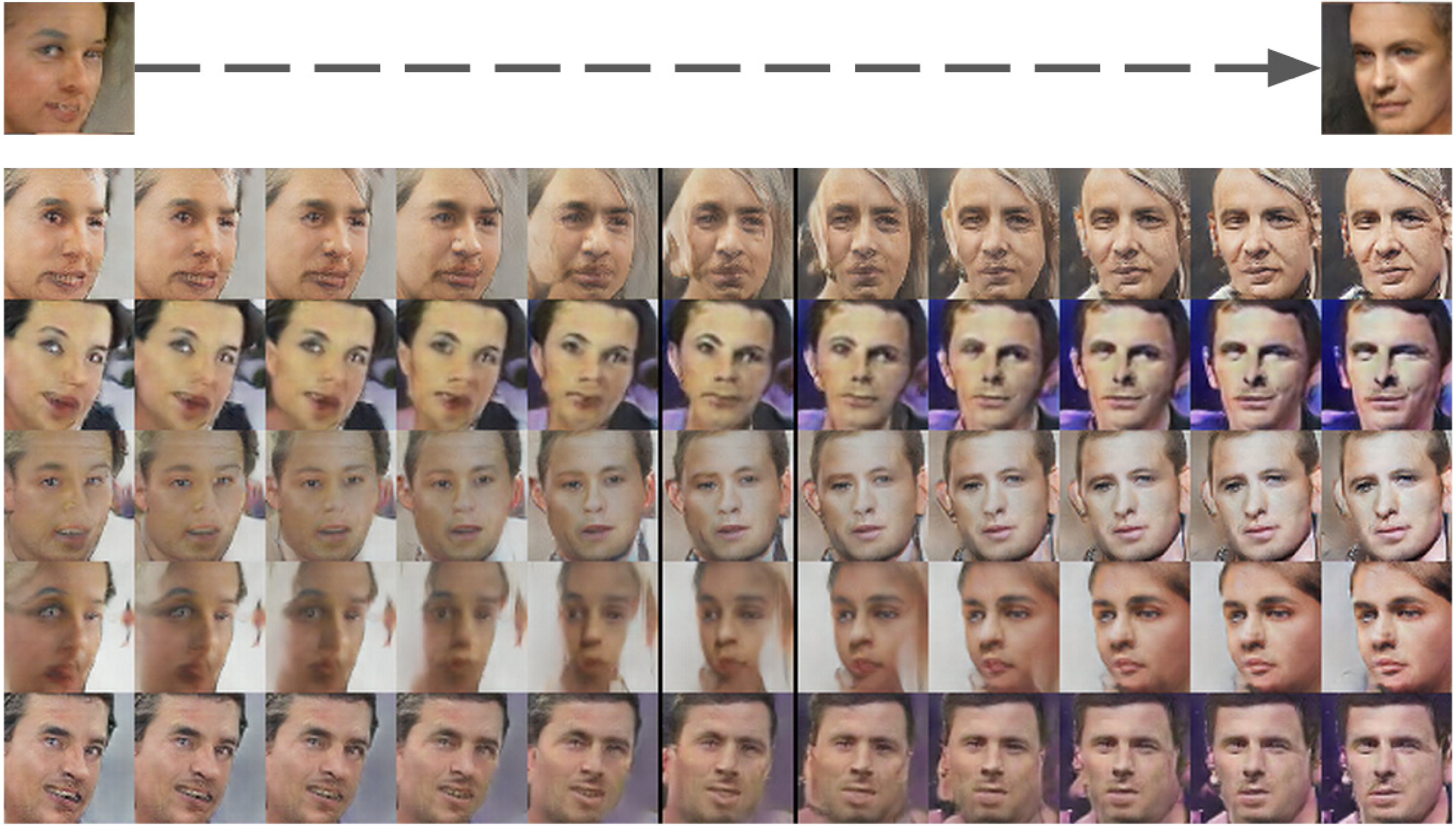


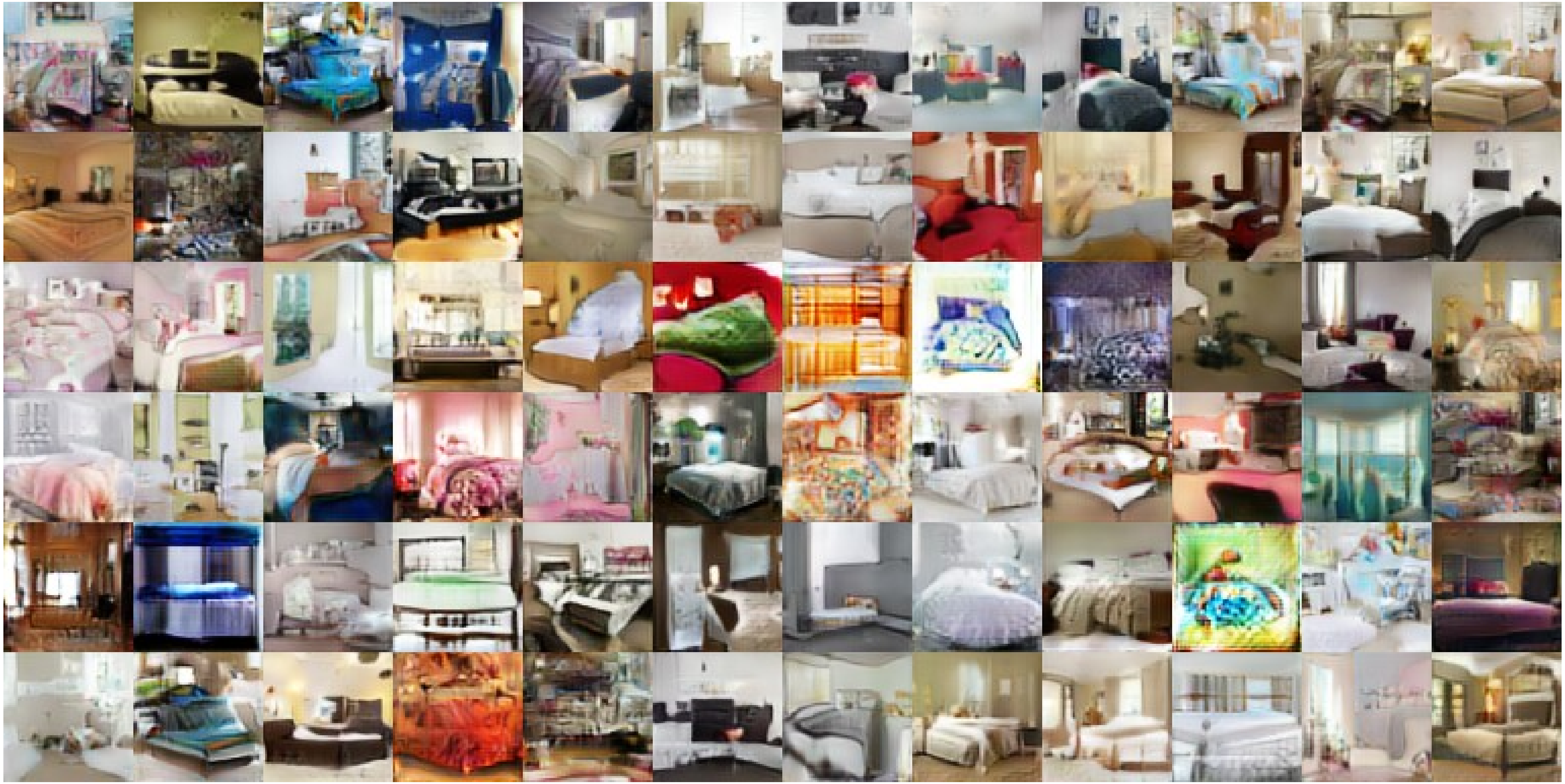
Figure 8 of "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", https://arxiv.org/abs/1511.06434

GANs Training — Training Experience

GAN output in paper
Your GAN output

*https://miro.medium.com/max/1400/1*r8cuSIaM5oHUERP01TCTxg.jpeg*

GANs Training – Results of In-House BigGAN Training



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 α are

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

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

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

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

GANs are Problematic to Train

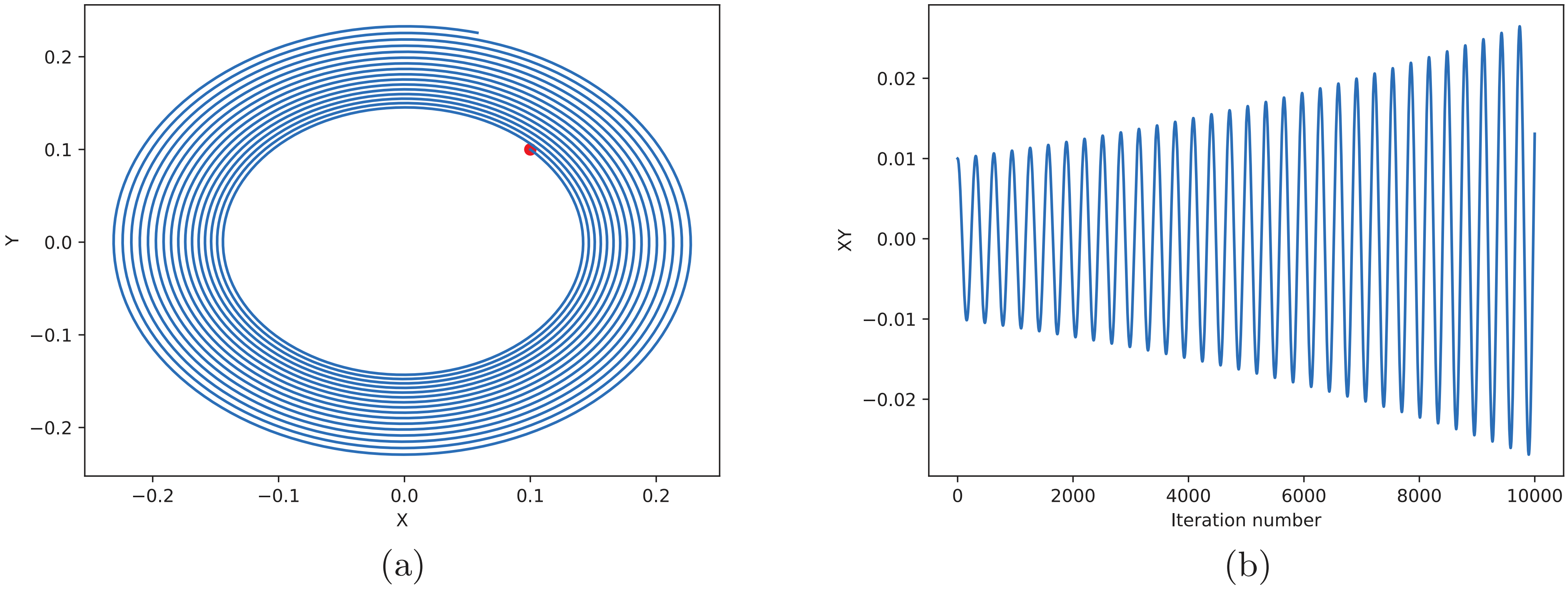


Fig. 1: Performance of gradient method with fixed step size for Example 2. (a) illustrates the choices of x and y as iteration processes, the red point $(0.1, 0.1)$ is the initial value. (b) illustrates the value of xy as a function of iteration numbers.

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

GANs are Problematic to Train

- Mode collapse

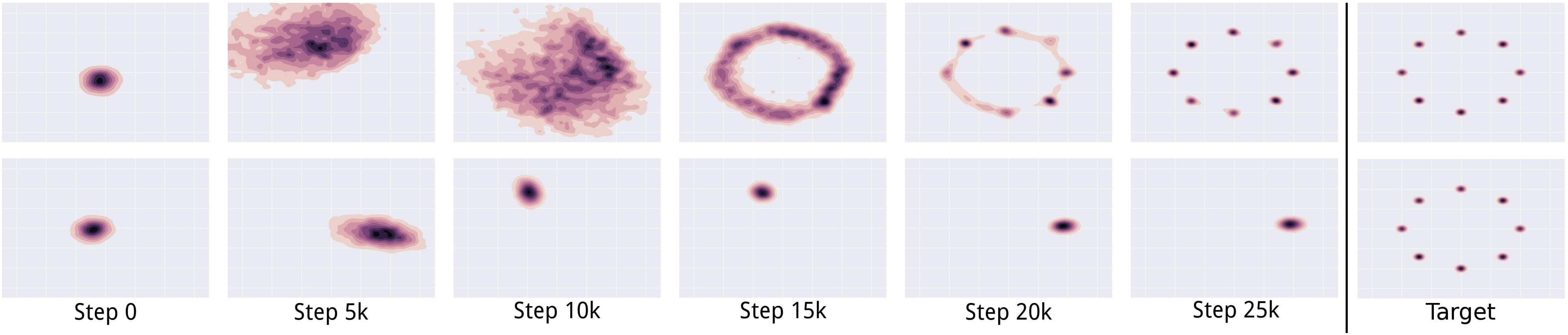


Figure 2 of "Unrolled Generative Adversarial Networks", https://arxiv.org/abs/1611.02163

  - 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.
  - Historical averaging
- Label smoothing of only positive samples helps with the gradient flow.

Comparison of VAEs and GANs

The Variational Autoencoders:

- 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.

Diffusion Models

Currently (as of May 2023), the best architecture for generating images seems to be the **diffusion models**.

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. These two approaches are in fact just different perspectives of the same model family, and many recent papers utilize both sides of these models.

Diffusion Models – Diffusion Process, Reverse Process

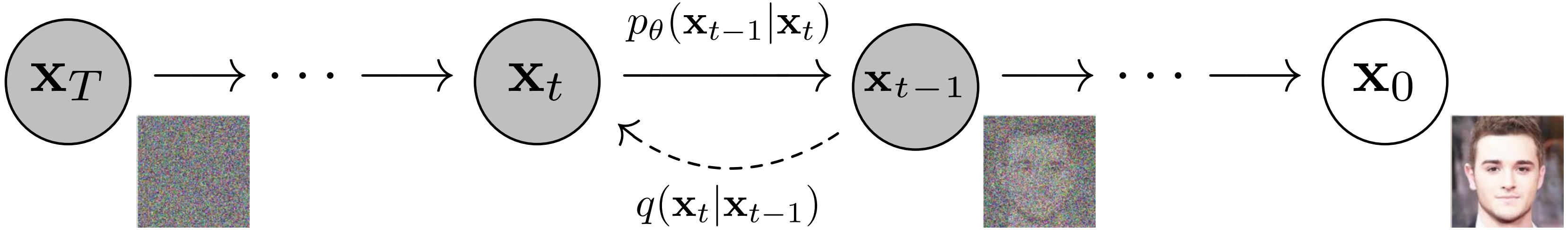

$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$\mathbf{x}_0$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$

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

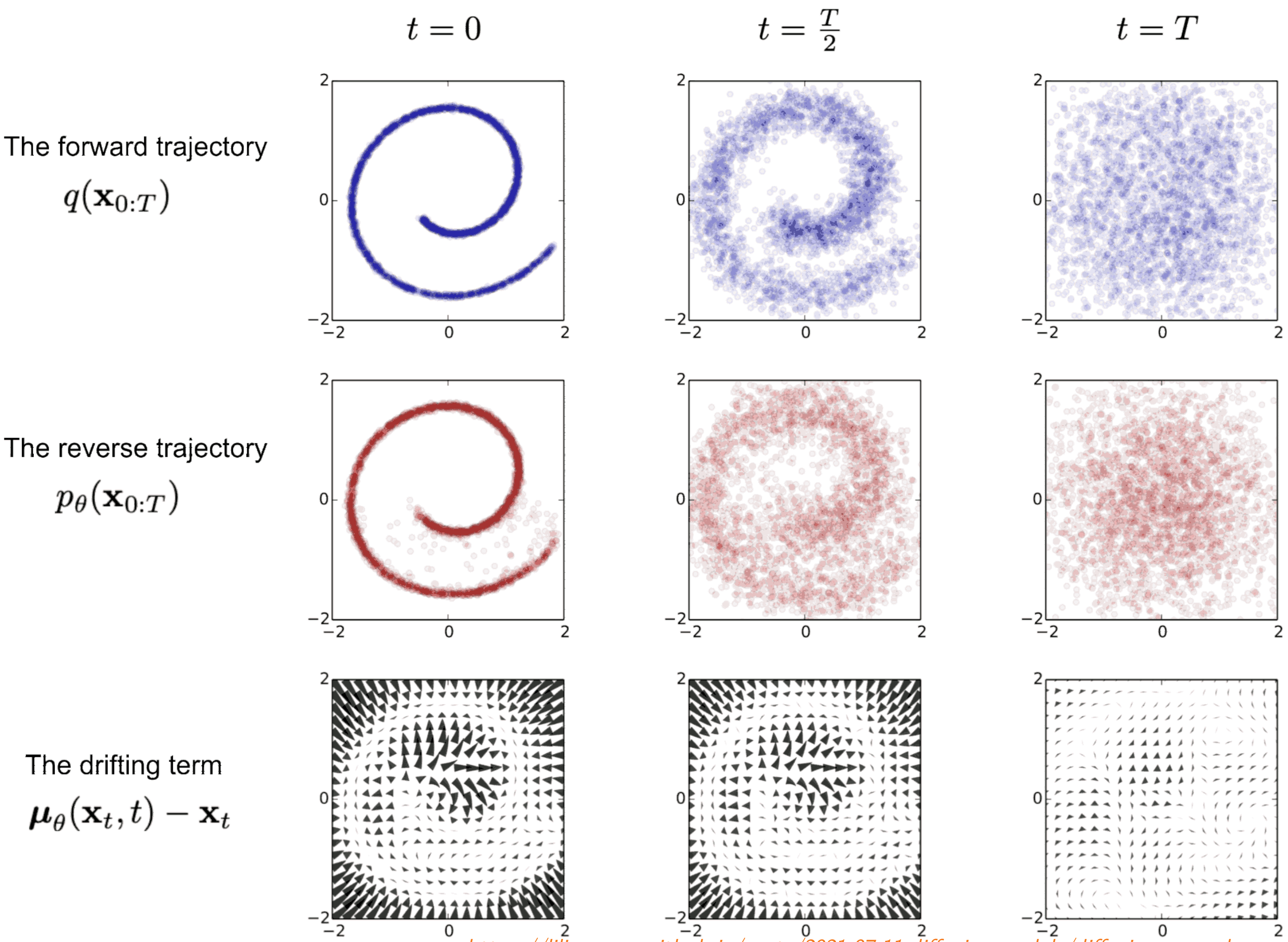
Given a data point $\mathbf{x}_0$ from a real data distribution $q(\mathbf{x})$, we define a T-step *diffusion process* (or the *forward process*) which gradually adds Gaussian noise to the input image:

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

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

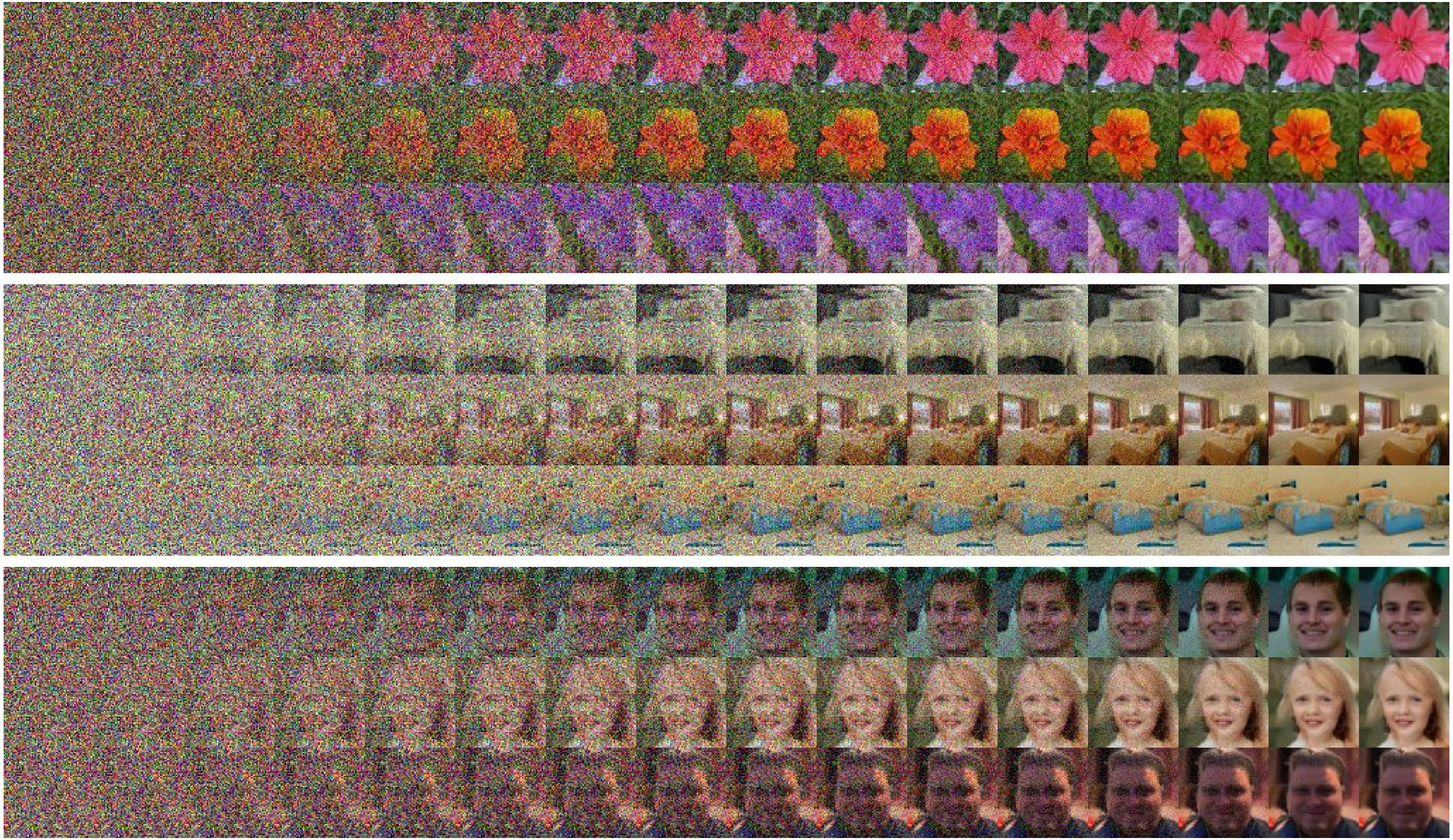
$$p_{\boldsymbol\theta}(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod\nolimits_{t=1}^T p_{\boldsymbol\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t).$$

Diffusion Models – The Diffusion Process



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

Diffusion Models – The Diffusion Process



Diffusion Models – Model Overview

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

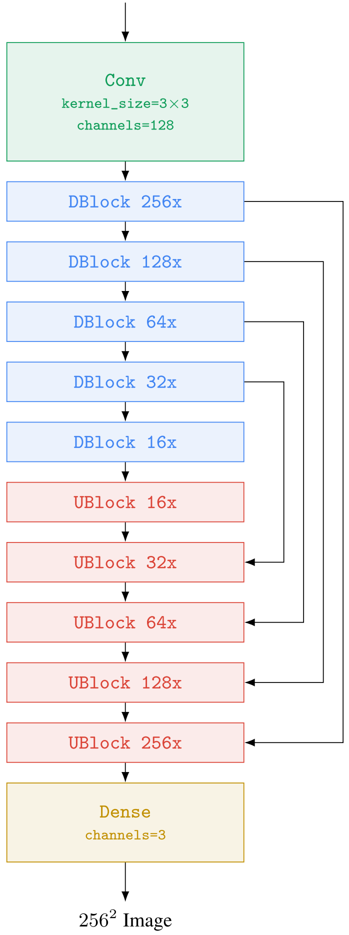


Figure A.30 of "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", https://arxiv.org/abs/2205.11487

Normal Distribution Reminder

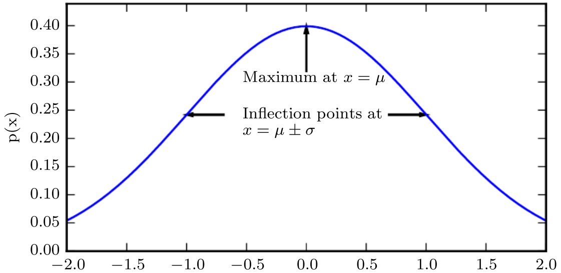


Figure 3.1 of "Deep Learning" book, https://www.deeplearningbook.org

Normal (or Gaussian) distribution is a continuous distribution parametrized by a mean μ and variance σ^2:

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

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

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

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 $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}$:

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

Normal Distribution Reminder

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

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

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

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

DDPM – The Forward Process

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

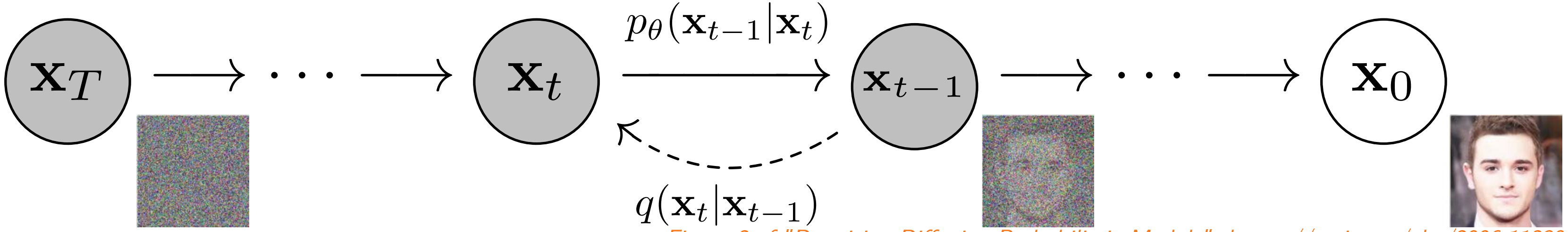

$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$\mathbf{x}_0$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$

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 $\beta_1, \ldots, \beta_T$:

$$
\begin{aligned}
q(\mathbf{x}_{1:T}|\mathbf{x}_0) &= \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}), \\
q(\mathbf{x}_t|\mathbf{x}_{t-1}) &= \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \boldsymbol{I}), \\
&= \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\mathbf{e} \ \text{ for } \ \mathbf{e} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}).
\end{aligned}
$$

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

DDPM – The Forward Process

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

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

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

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

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

DDPM – Noise Schedule

Originally, linearly increasing sequence of noise variations $\beta_1 = 0.0001, \ldots, \beta_T = 0.04$ 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:

$$\bar\alpha_t = \frac{1}{2}\Big(\cos(t/T \cdot \pi) + 1\Big),$$

and now it is dominantly used.

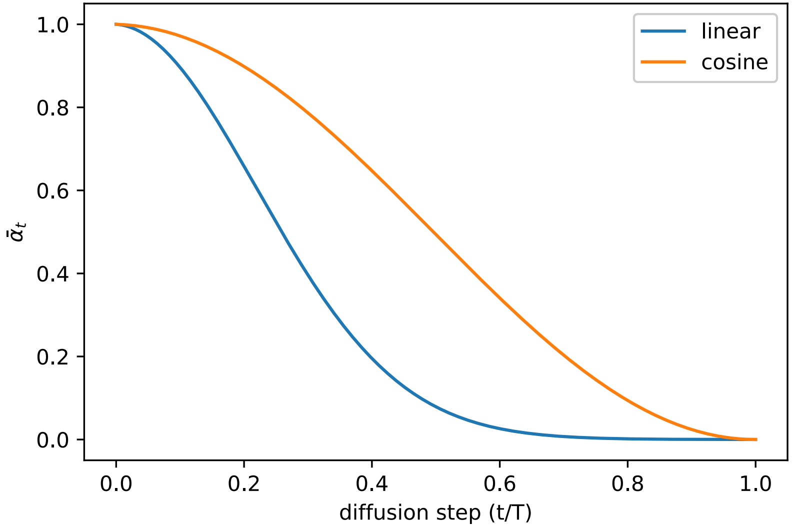


Figure 5. $\bar\alpha_t$ throughout diffusion in the linear schedule and our proposed cosine schedule.

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

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

DDPM – Noise Schedule

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

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

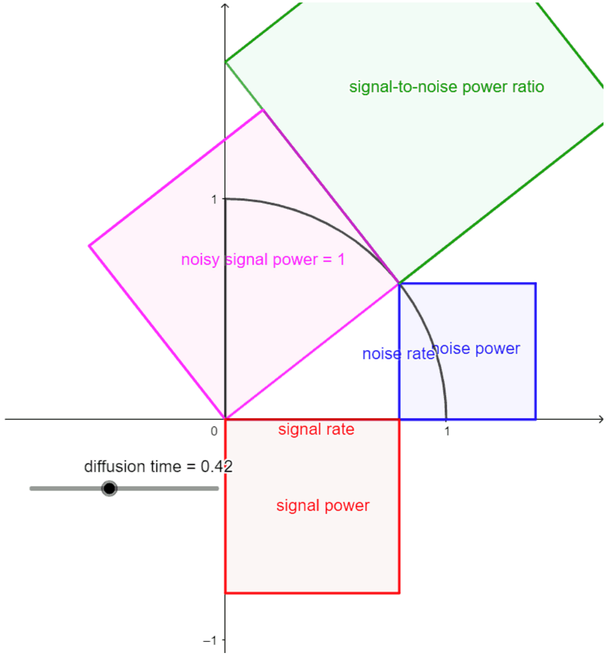
has also zero mean and unit variance.

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.



https://i.imgur.com/JW9W0fA.gif

DDPM – The Reverse Process

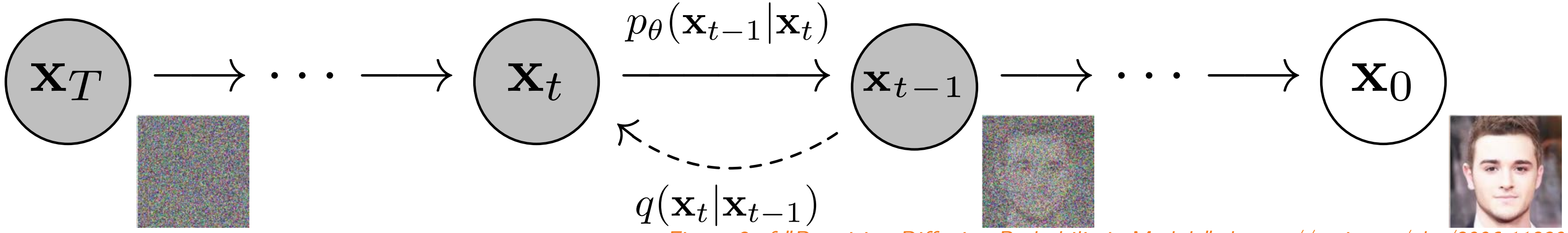

$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$\mathbf{x}_0$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$

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

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

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

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

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

The whole reverse process is then

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

DDPM – Loss

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 $\mathbf{x}_0$:

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

We present the proof on the next slide for completeness.

Forward Process Reverse Derivation

Starting with the Bayes' rule, we get

$$
\begin{aligned}
q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) &= q(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_0)\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} \\
&\propto \exp\Big(-\frac{1}{2}\Big(\frac{(\mathbf{x}_t-\sqrt{\alpha_t}\mathbf{x}_{t-1})^2}{\beta_t}+\frac{(\mathbf{x}_{t-1}-\sqrt{\bar\alpha_{t-1}}\mathbf{x}_0)^2}{1-\bar\alpha_{t-1}}-\frac{(\mathbf{x}_t-\sqrt{\bar\alpha_t}\mathbf{x}_0)^2}{1-\bar\alpha_t}\Big)\Big) \\
&= \exp\Big(-\frac{1}{2}\Big(\tfrac{\mathbf{x}_t^2-2\sqrt{\alpha_t}\mathbf{x}_t\mathbf{x}_{t-1}+\alpha_t\mathbf{x}_{t-1}^2}{\beta_t}+\tfrac{\mathbf{x}_{t-1}^2-2\sqrt{\bar\alpha_{t-1}}\mathbf{x}_{t-1}\mathbf{x}_0+\bar\alpha_{t-1}\mathbf{x}_0^2}{1-\bar\alpha_{t-1}}+\ldots\Big)\Big) \\
&= \exp\Big(-\frac{1}{2}\Big(\big(\tfrac{\alpha_t}{\beta_t}+\tfrac{1}{1-\bar\alpha_{t-1}}\big)\mathbf{x}_{t-1}^2-2\big(\tfrac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t+\tfrac{\sqrt{\bar\alpha_{t-1}}}{1-\bar\alpha_{t-1}}\mathbf{x}_0\big)\mathbf{x}_{t-1}+\ldots\Big)\Big)
\end{aligned}
$$

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

$$
\tilde\beta_t = 1/\big(\tfrac{\alpha_t}{\beta_t}+\tfrac{1}{1-\bar\alpha_{t-1}}\big) = 1/\big(\tfrac{\alpha_t(1-\bar\alpha_{t-1})+\beta_t}{\beta_t(1-\bar\alpha_{t-1})}\big) = 1/\big(\tfrac{\alpha_t+\beta_t-\bar\alpha_t}{\beta_t(1-\bar\alpha_{t-1})}\big) = \frac{1-\bar\alpha_{t-1}}{1-\bar\alpha_t}\beta_t,
$$

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

DDPM – Deriving Loss using Jensen's Inequality

$$
\begin{aligned}
-\mathbb{E}_{q(\mathbf{x}_0)}\Big[\log p_{\boldsymbol\theta}(\mathbf{x}_0)\Big] &= -\mathbb{E}_{q(\mathbf{x}_0)}\Big[\log \mathbb{E}_{p_{\boldsymbol\theta}(\mathbf{x}_{1:T})}[p_{\boldsymbol\theta}(\mathbf{x}_0)]\Big] \\
&= -\mathbb{E}_{q(\mathbf{x}_0)}\bigg[\log \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\bigg[\frac{p_{\boldsymbol\theta}(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\bigg]\bigg] \\
&\le -\mathbb{E}_{q(\mathbf{x}_{0:T})}\bigg[\log \frac{p_{\boldsymbol\theta}(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\bigg] = \mathbb{E}_{q(\mathbf{x}_{0:T})}\bigg[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_{\boldsymbol\theta}(\mathbf{x}_{0:T})}\bigg] \\
&= \mathbb{E}_{q(\mathbf{x}_{0:T})}\bigg[-\log p_{\boldsymbol\theta}(\mathbf{x}_T) + \sum\nolimits_{t=2}^{T} \log \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_{\boldsymbol\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)} + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_{\boldsymbol\theta}(\mathbf{x}_0|\mathbf{x}_1)}\bigg] \\
&= \mathbb{E}_{q(\mathbf{x}_{0:T})}\bigg[-\log p_{\boldsymbol\theta}(\mathbf{x}_T) + \sum\nolimits_{t=2}^{T} \log \bigg(\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}{p_{\boldsymbol\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)} \frac{q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}\bigg) + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_{\boldsymbol\theta}(\mathbf{x}_0|\mathbf{x}_1)}\bigg] \\
&= \mathbb{E}_{q(\mathbf{x}_{0:T})}\bigg[-\log p_{\boldsymbol\theta}(\mathbf{x}_T) + \sum\nolimits_{t=2}^{T} \log \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}{p_{\boldsymbol\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_{\boldsymbol\theta}(\mathbf{x}_0|\mathbf{x}_1)}\bigg] \\
&= \mathbb{E}_{q(\mathbf{x}_{0:T})}\bigg[\log \frac{q(\mathbf{x}_T|\mathbf{x}_0)}{p_{\boldsymbol\theta}(\mathbf{x}_T)} + \sum\nolimits_{t=2}^{T} \log \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}{p_{\boldsymbol\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)} - \log p_{\boldsymbol\theta}(\mathbf{x}_0|\mathbf{x}_1)\bigg] \\
&= \mathbb{E}_{q(\mathbf{x}_{0:T})}\Big[\underbrace{D_\mathrm{KL}(q(\mathbf{x}_T|\mathbf{x}_0)\|p_{\boldsymbol\theta}(\mathbf{x}_T))}_{L_T} + \sum\nolimits_{t=2}^{T} \underbrace{D_\mathrm{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\,\mathbf{x}_0)\|p_{\boldsymbol\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)}_{L_t}\underbrace{-\log p_{\boldsymbol\theta}(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}\Big]
\end{aligned}
$$

DDPM – Deriving Loss using Jensen's Inequality

The whole loss is therefore composed of the following components:

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

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

- $L_0 = -\log p_{\boldsymbol\theta}(\mathbf{x}_0|\mathbf{x}_1)$ 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 $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\big(\mathbf{x}_{t-1}; \tilde{\boldsymbol\mu}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde\beta_t \boldsymbol{I}\big)$ for

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

$$\tilde\beta_t = \frac{1-\bar\alpha_{t-1}}{1-\bar\alpha_t}\beta_t.$$

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

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

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

DDPM – Reparametrizing Model Prediction

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

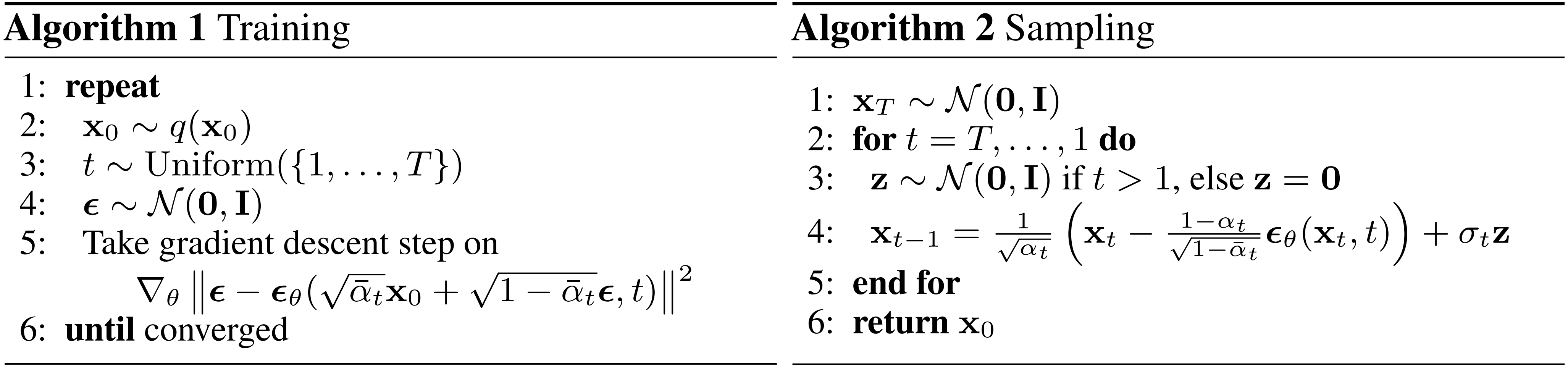
$$\begin{aligned}
L_t &= \mathbb{E}\Bigg[\frac{1}{2\|\sigma_t \boldsymbol{I}\|^2}\Big\|\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\Big\|^2\Bigg] \\
&= \mathbb{E}\Bigg[\frac{1}{2\|\sigma_t \boldsymbol{I}\|^2}\Big\|\frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t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

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

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

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

```
repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
      $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
until converged
```

Algorithm 2 Sampling

```
$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ do
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$
```

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

Sampling using the proposed algorithm is slow – it is common to use $T = 1000$ steps during sampling.

The value of σ_t^2 is chosen to be either β_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 will be alleviated later, when we present DDIM providing an updated sampling algorithm, which runs in several tens of steps and does not use σ_t^2.

DDPM – Diffusion Models Architecture

The DDPM models the noise prediction $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ using a UNet architecture with pre-activated ResNet blocks.

- The current 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.

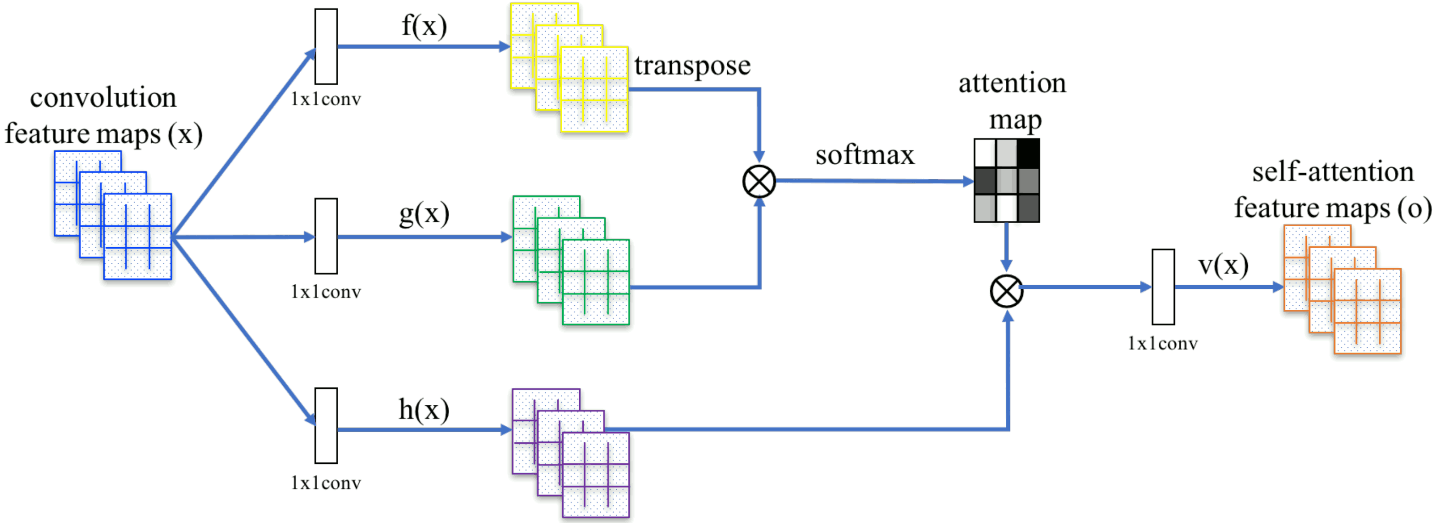


Figure 2 of "Self-Attention Generative Adversarial Networks", https://arxiv.org/abs/1805.08318

Diffusion Models Architecture – ImaGen

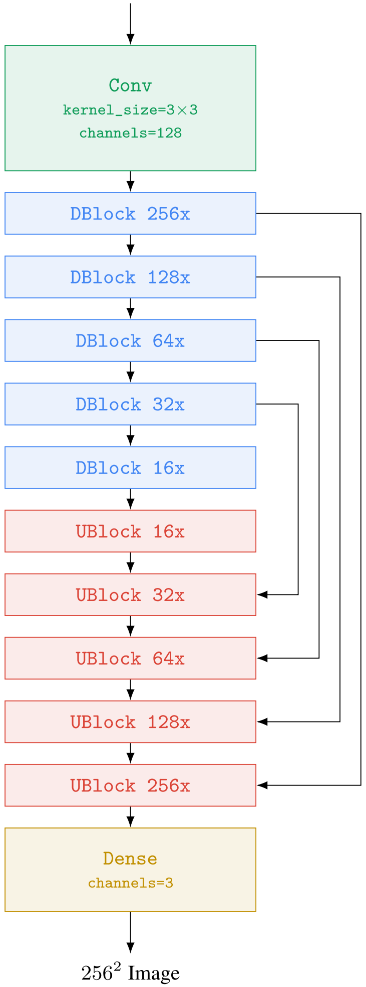


Figure A.30 of "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", https://arxiv.org/abs/2205.11487

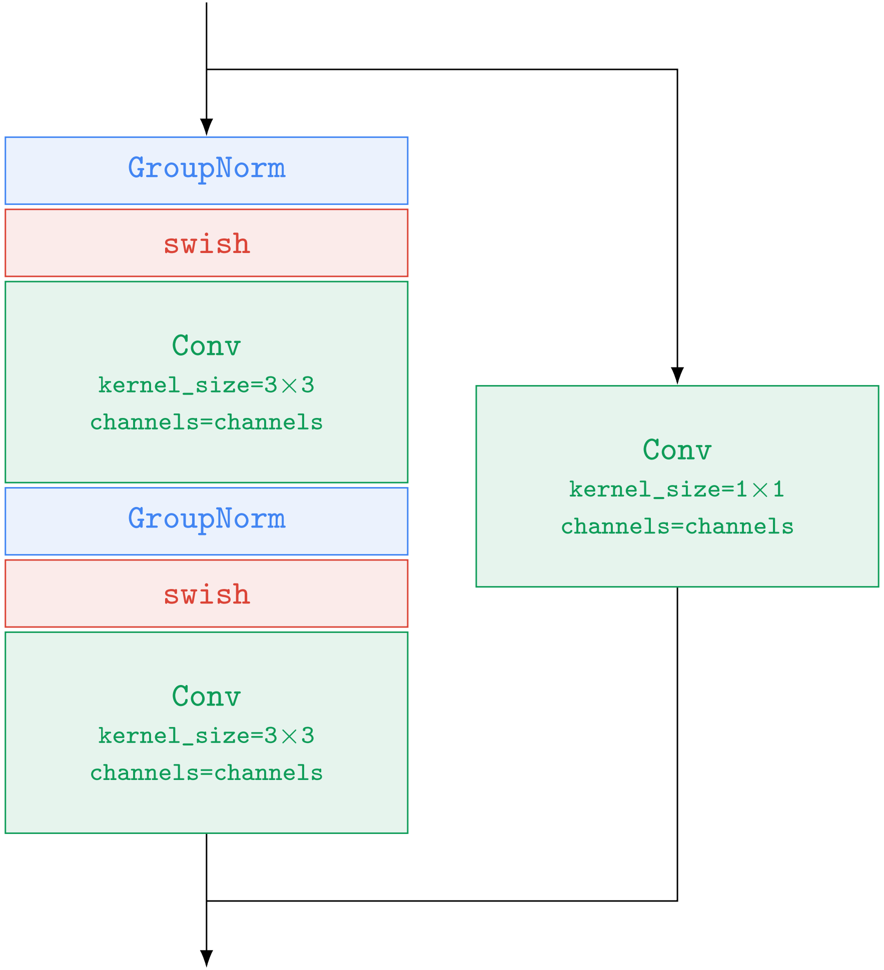


Figure A.27 of "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", https://arxiv.org/abs/2205.11487

Diffusion Models Architecture – ImaGen

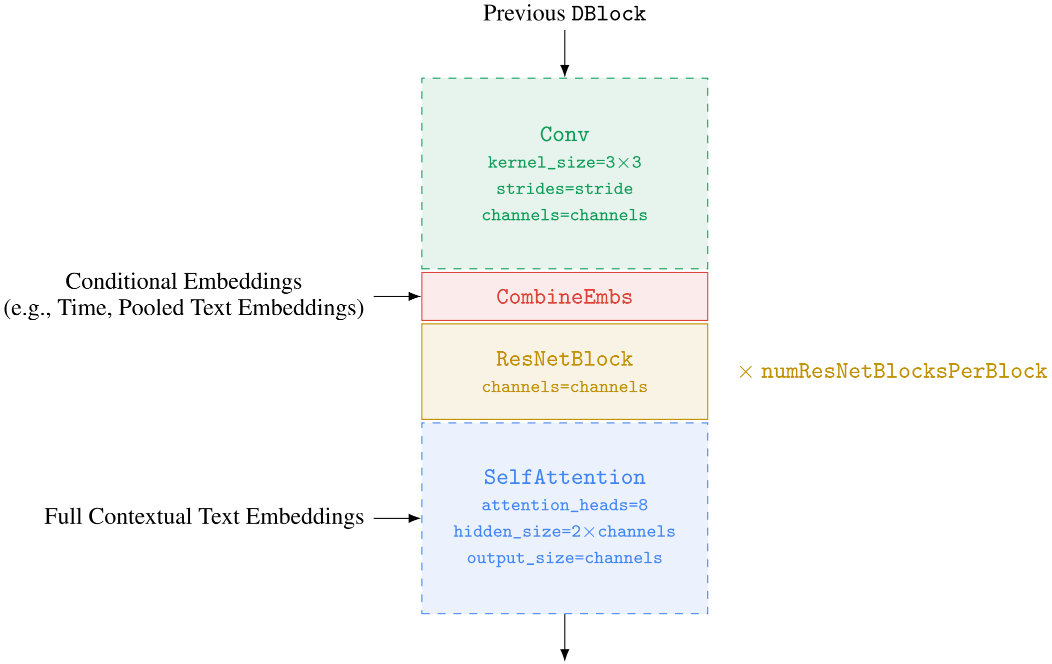


Figure A.28 of "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", https://arxiv.org/abs/2205.11487

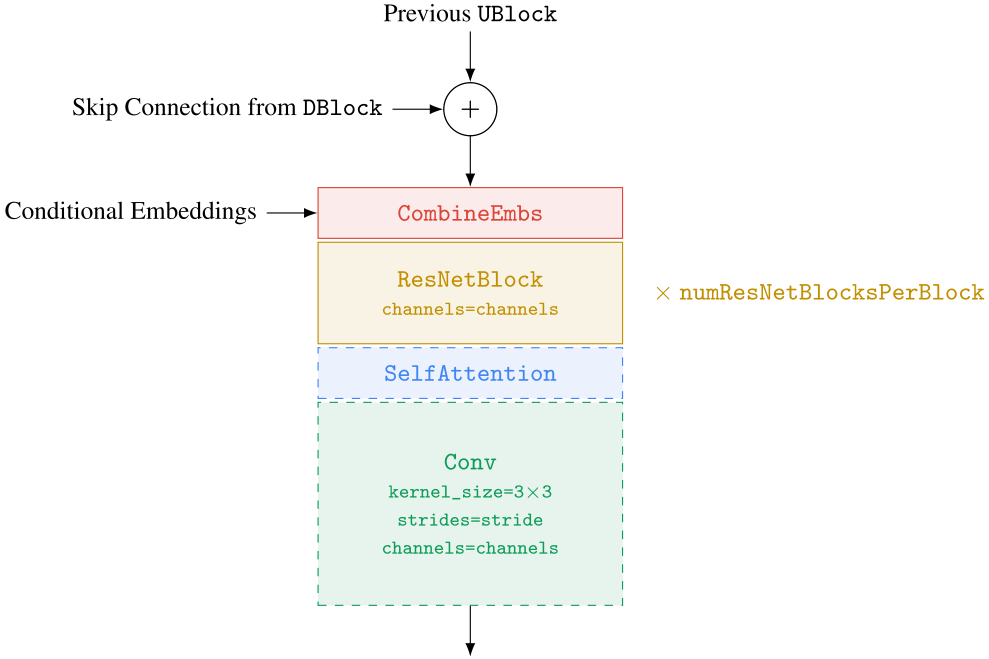


Figure A.29 of "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", https://arxiv.org/abs/2205.11487

There are just minor differences in the ImaGen architecture – for example the place where the time sinusoidal embeddings are added.

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 $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, y)$ with the conditioning y, and sometimes we train $\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, \varnothing)$ without the conditioning.
- During sampling, we pronounce the effect of the conditioning by taking the unconditioned noise and adding the difference between conditioned and unconditioned noise *weighted by the weight* w (Stable Diffusion uses $w = 7.5$):

$$\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, \varnothing) + w\big(\boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, y) - \boldsymbol{\varepsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, \varnothing)\big).$$

Denoising Diffusion Implicit Models

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" $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\big(\sqrt{\bar\alpha_t}\mathbf{x}_0, (1-\bar\alpha_t)\boldsymbol{I}\big)$.

The second condition will allow us to use the same loss as in DDPM – therefore, the training algorithm is exactly identical do 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.

Denoising Diffusion Implicit Models – The Forward Process

The forward process of DDIM can be described using

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

where

- $q_0(\mathbf{x}_T|\mathbf{x}_0) = \mathcal{N}\big(\sqrt{\bar\alpha_T}\mathbf{x}_0, (1-\bar\alpha_T)\boldsymbol{I}\big)$,
- $q_0(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\Big(\sqrt{\bar\alpha_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar\alpha_{t-1}}\big(\frac{\mathbf{x}_t - \sqrt{\bar\alpha_t}\mathbf{x}_0}{\sqrt{1-\bar\alpha_t}}\big), 0\cdot\boldsymbol{I}\Big)$.

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

$$\begin{aligned}\mathbf{x}_{t-1} &= \sqrt{\bar\alpha_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar\alpha_{t-1}}\big(\tfrac{\mathbf{x}_t - \sqrt{\bar\alpha_t}\mathbf{x}_0}{\sqrt{1-\bar\alpha_t}}\big)\\ &= \sqrt{\bar\alpha_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar\alpha_{t-1}}\big(\tfrac{\sqrt{\bar\alpha_t}\mathbf{x}_0 + \sqrt{1-\bar\alpha_t}\mathbf{e}_t - \sqrt{\bar\alpha_t}\mathbf{x}_0}{\sqrt{1-\bar\alpha_t}}\big) = \sqrt{\bar\alpha_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar\alpha_{t-1}}\mathbf{e}_t.\end{aligned}$$

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 $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, we perform the following for t from T down to 1:

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

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

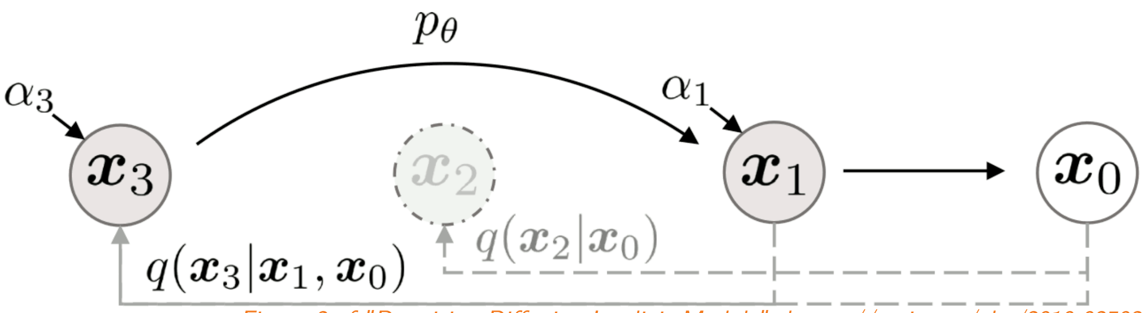


Figure 2 of "Denoising Diffusion Implicit Models", https://arxiv.org/abs/2010.02502

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

Denoising Diffusion Implicit Models – Accelerated Sampling

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

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

$$\boldsymbol{x}_{t_{i-1}} = \sqrt{\bar\alpha_{t_{i-1}}}\left(\frac{\boldsymbol{x}_{t_i} - \sqrt{1-\bar\alpha_{t_i}}\boldsymbol{\varepsilon}_{\boldsymbol\theta}(\boldsymbol{x}_{t_i}, t_i)}{\sqrt{\bar\alpha_{t_i}}}\right) + \sqrt{1-\bar\alpha_{t_{i-1}}}\boldsymbol{\varepsilon}_{\boldsymbol\theta}(\boldsymbol{x}_{t_i}, t_i).$$

The sampling procedure can be described in words as follows:

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

DDIM – Accelerated Sampling Examples

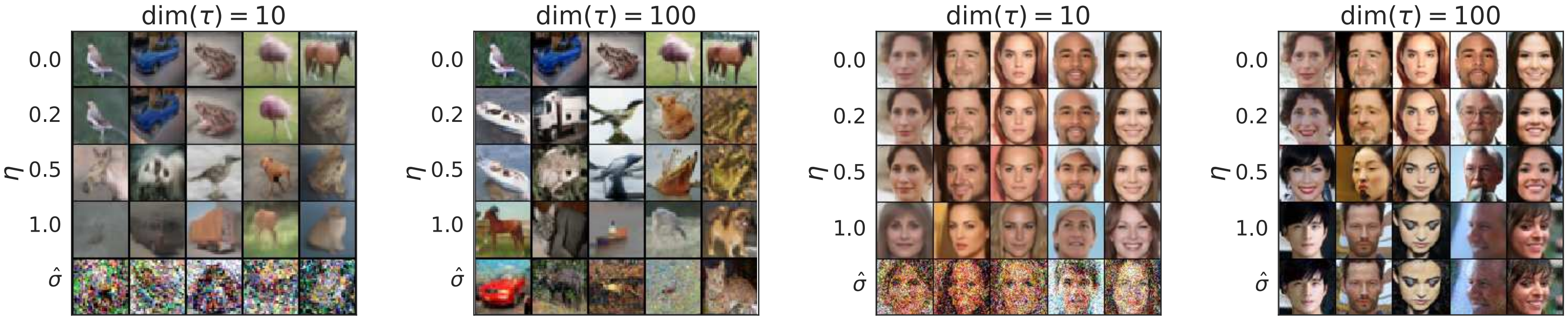


Figure 3 of "Denoising Diffusion Implicit Models", https://arxiv.org/abs/2010.02502

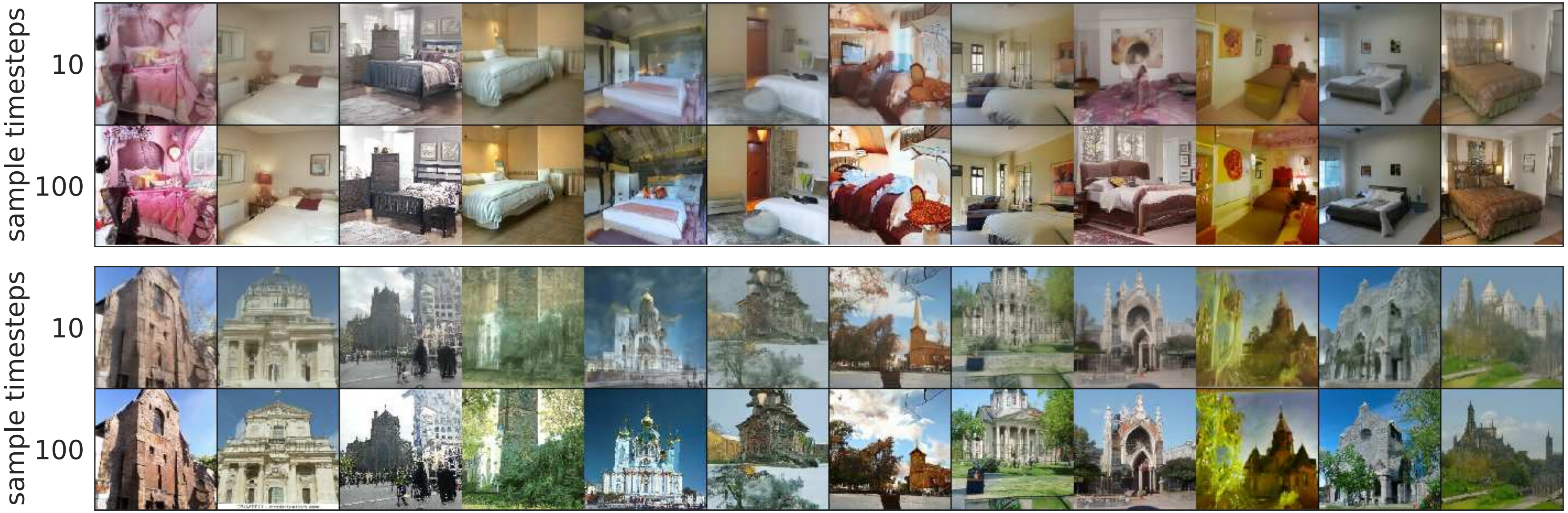
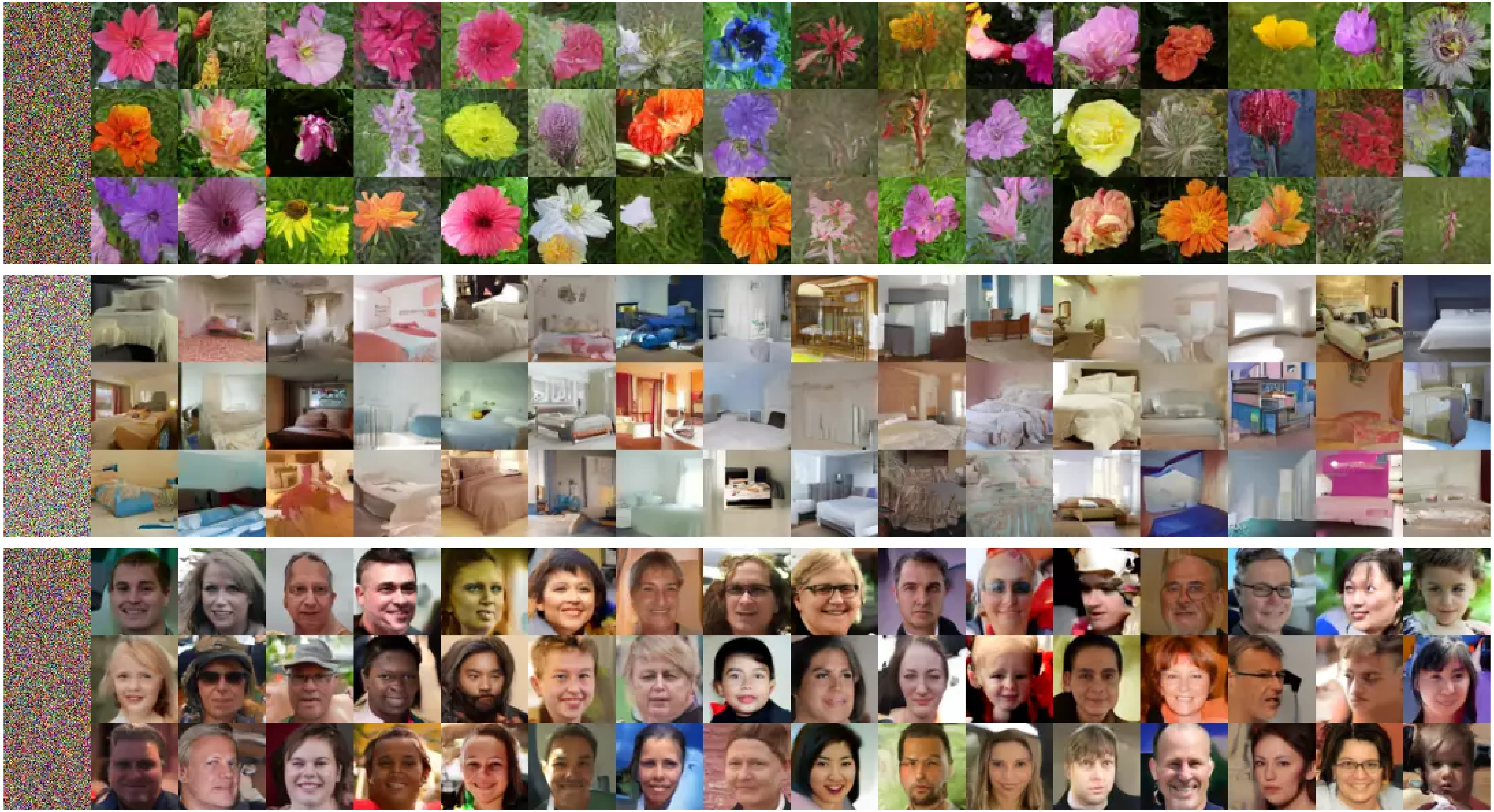
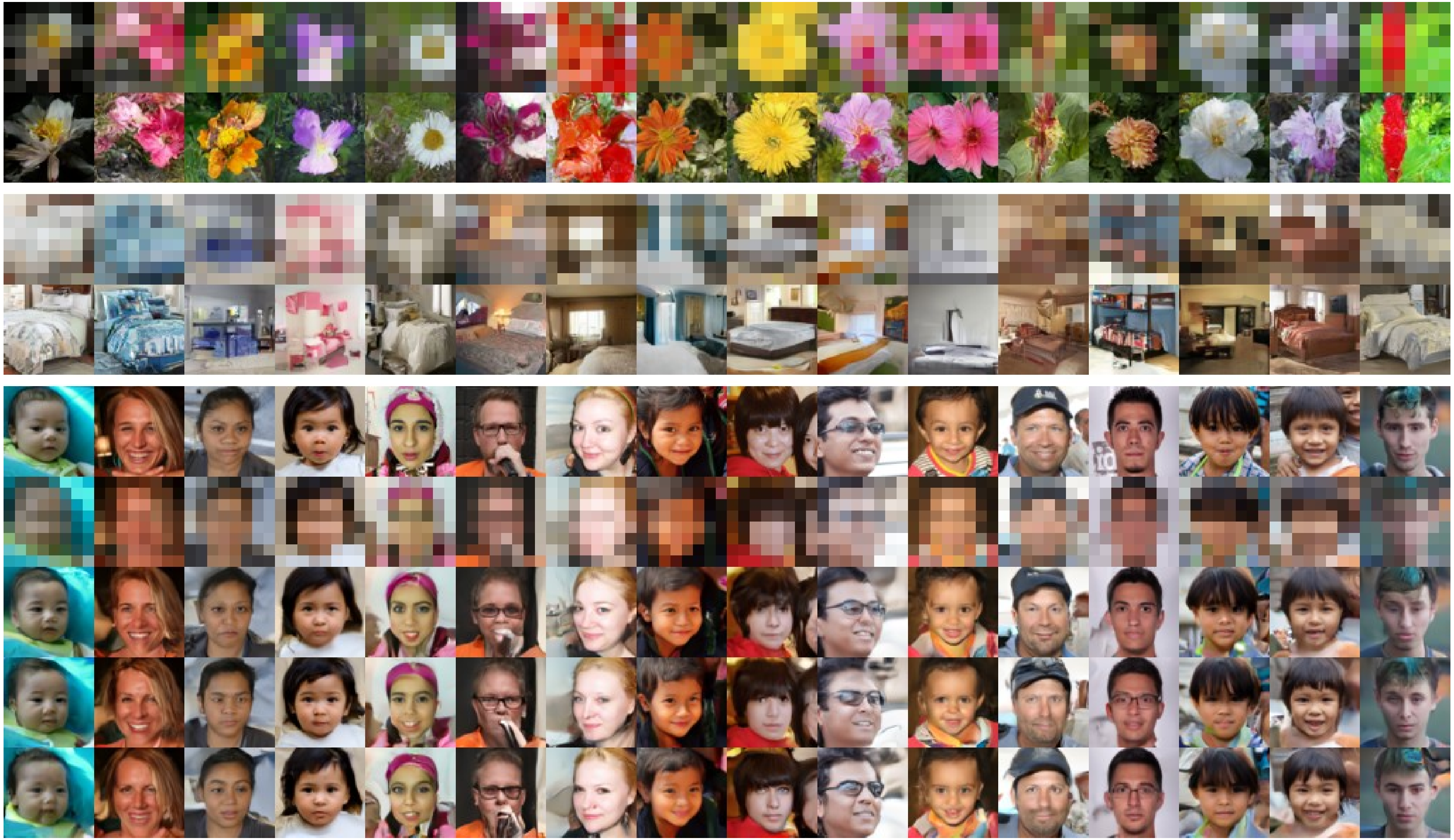


Figure 5 of "Denoising Diffusion Implicit Models", https://arxiv.org/abs/2010.02502

DDIM – Samples from Model Trained in Practicals



DDIM – Conditional Samples from Model Trained in Practicals



Stable Diffusion – Semantic and Perceptual Compression

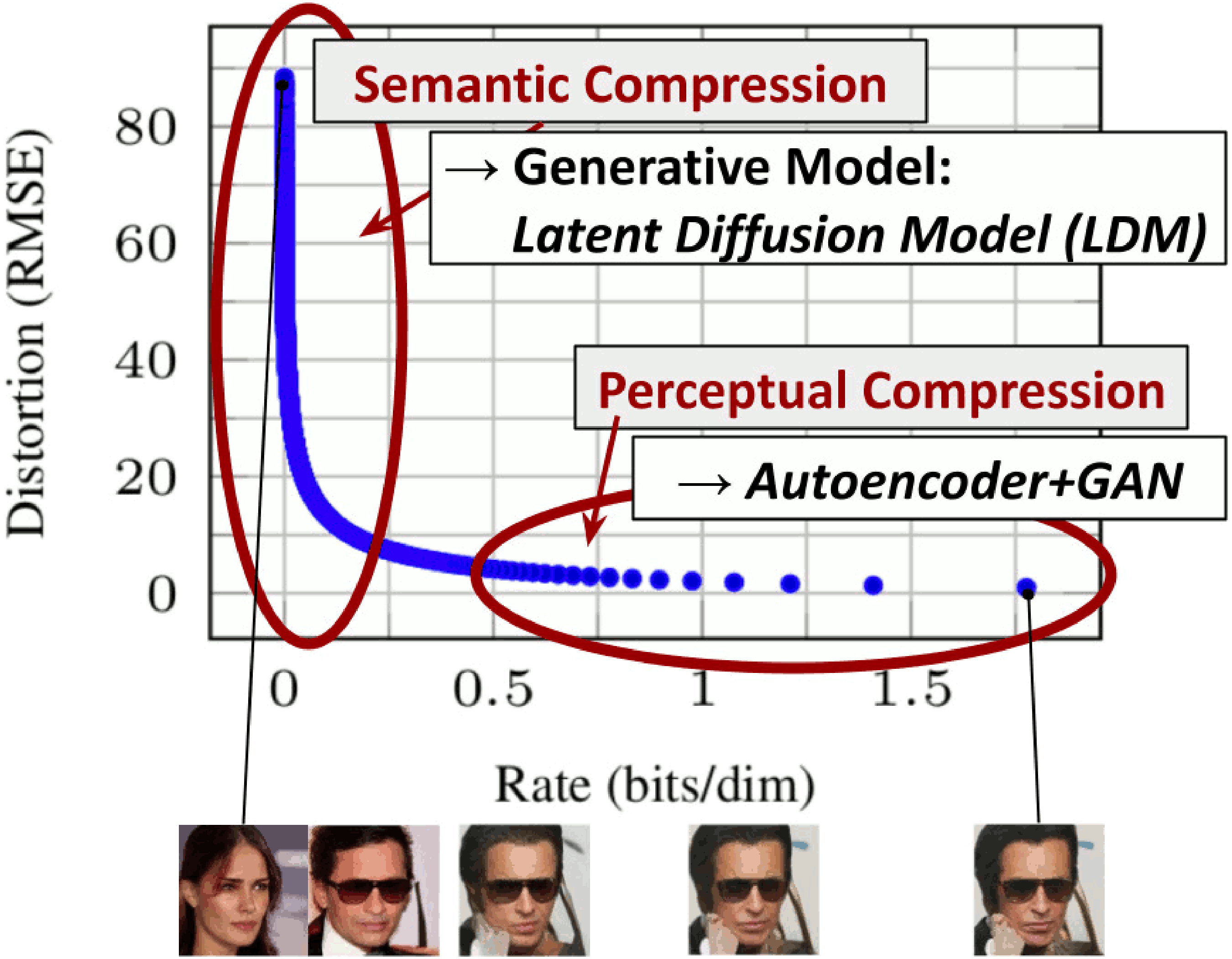


Figure 2 of "High-Resolution Image Synthesis with Latent Diffusion Models", https://arxiv.org/abs/2112.10752

Stable Diffusion – Architecture

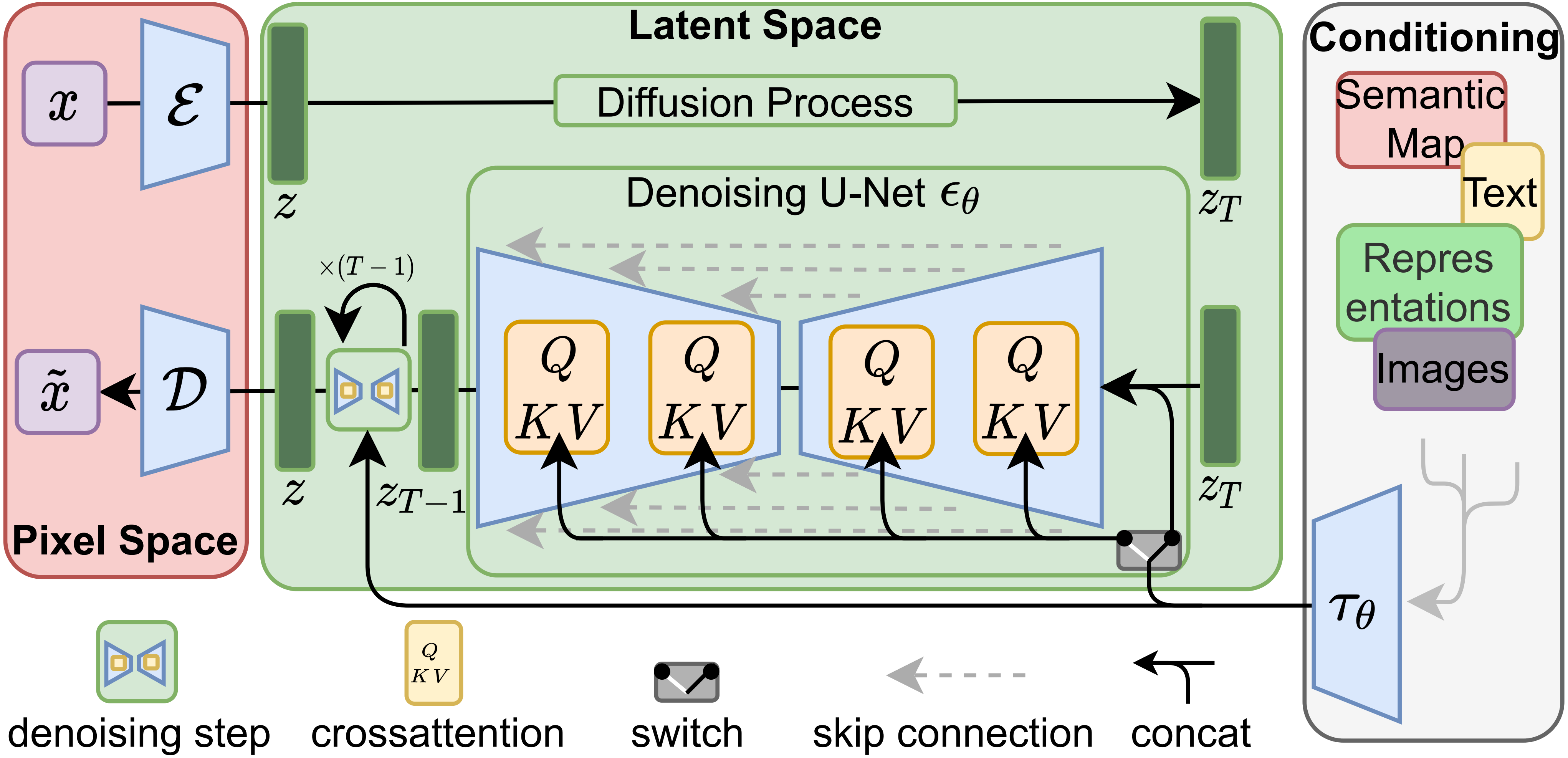


Figure 3 of "High-Resolution Image Synthesis with Latent Diffusion Models", https://arxiv.org/abs/2112.10752

Score Matching

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

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

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

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

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

because

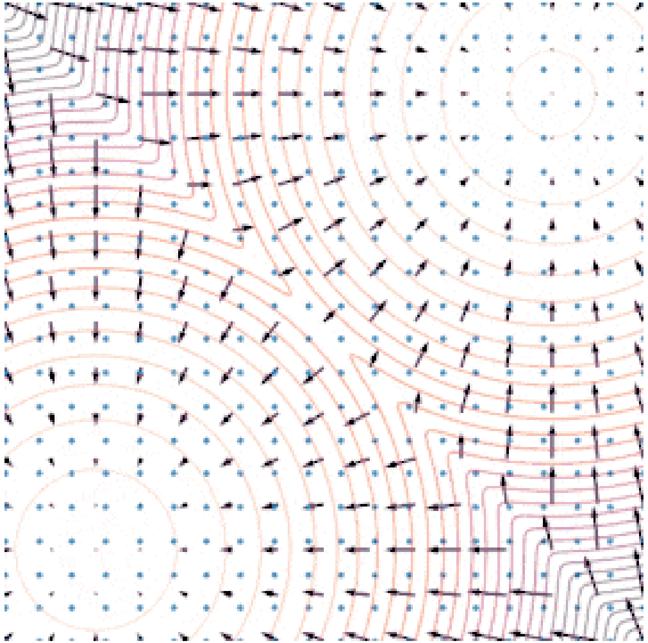
$$s_{\boldsymbol\theta}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_{\boldsymbol\theta}(\mathbf{x}) = \nabla_{\mathbf{x}} \log \frac{e^{f_{\boldsymbol\theta}(\mathbf{x})}}{Z_{\boldsymbol\theta}} = \nabla_{\mathbf{x}} f_{\boldsymbol\theta}(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z_{\boldsymbol\theta}}_{0} = \nabla_{\mathbf{x}} f_{\boldsymbol\theta}(\mathbf{x}).$$

Langevin Dynamics

When we have a score function $\nabla_{\mathbf{x}} \log p_{\boldsymbol\theta}(\mathbf{x})$, we can use it to perform sampling from the distribution $p_{\boldsymbol\theta}(\mathbf{x})$ 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

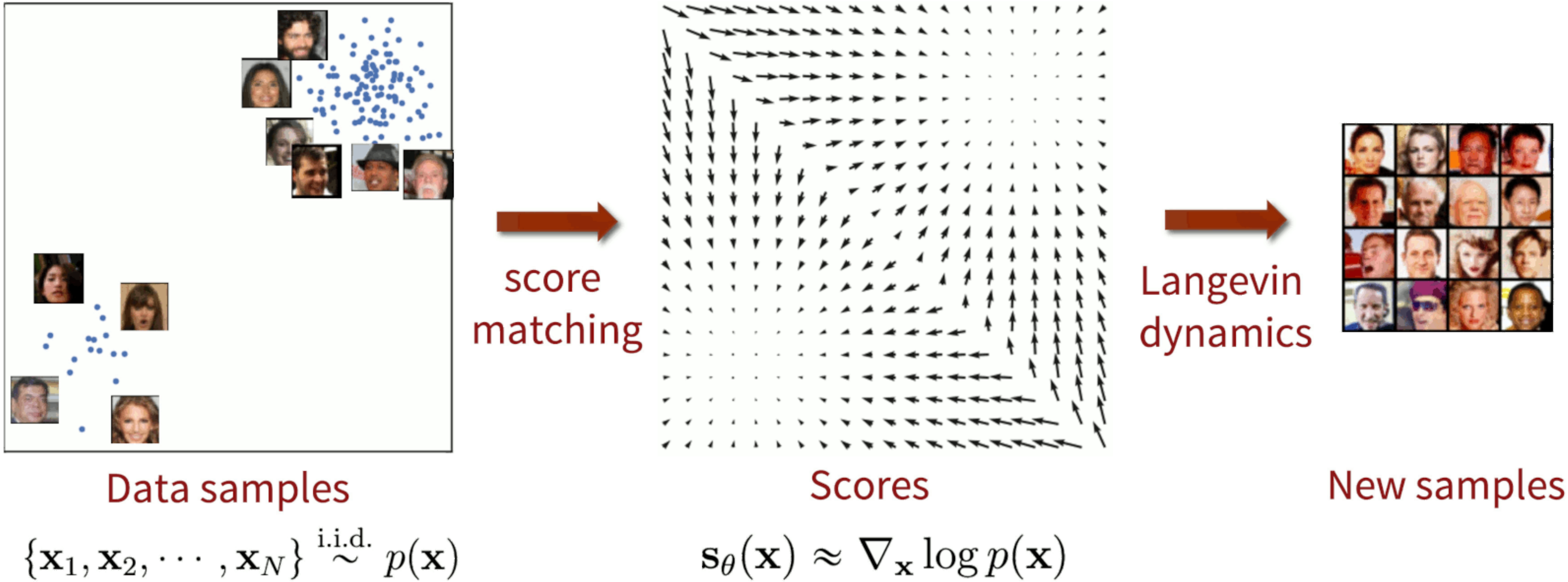
$$\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \varepsilon \nabla_{\mathbf{x}_i} \log p_{\boldsymbol\theta}(\mathbf{x}_i) + \sqrt{2\varepsilon}\, \mathbf{z}_i, \quad \text{where } \boldsymbol{z}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}).$$

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



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

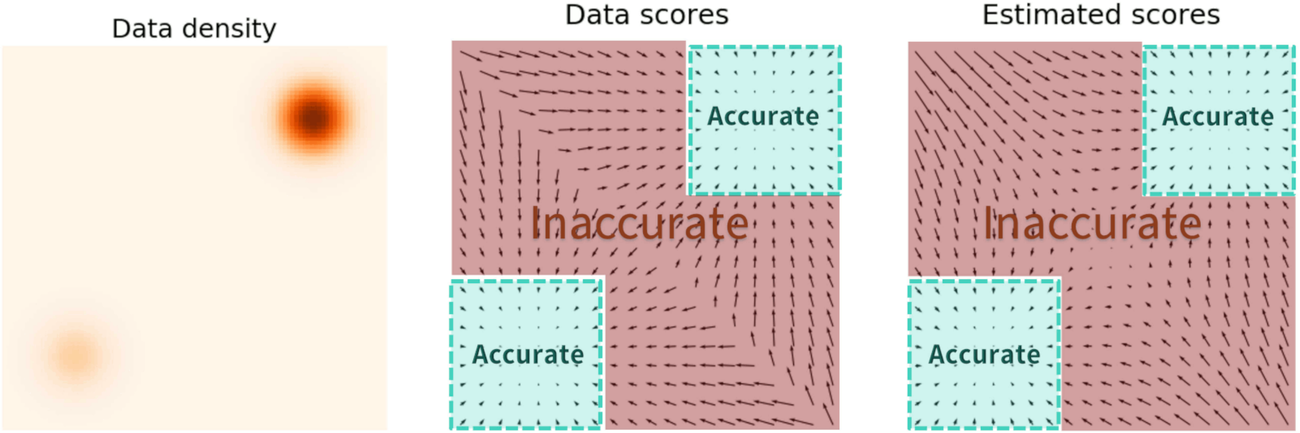
Score-Based Generative Modeling


score matching
Langevin dynamics
Data samples
$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \overset{\text{i.i.d.}}{\sim} p(\mathbf{x})$
Scores
$\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$
New samples

https://yang-song.net/assets/img/score/smld.jpg

Noise Conditional Score Network

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



https://yang-song.net/assets/img/score/pitfalls.jpg

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 σ_t:

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

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

Noise Conditional Score Network

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

$$\mathbb{E}_{t, \tilde{\mathbf{x}} \sim q_{\sigma_t}} \Big[\big\| \boldsymbol{s_\theta}(\tilde{\mathbf{x}}, \sigma_t) - \nabla_{\tilde{\mathbf{x}}} \log q_{\sigma_t}(\tilde{\mathbf{x}}) \big\|^2 \Big].$$

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:

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

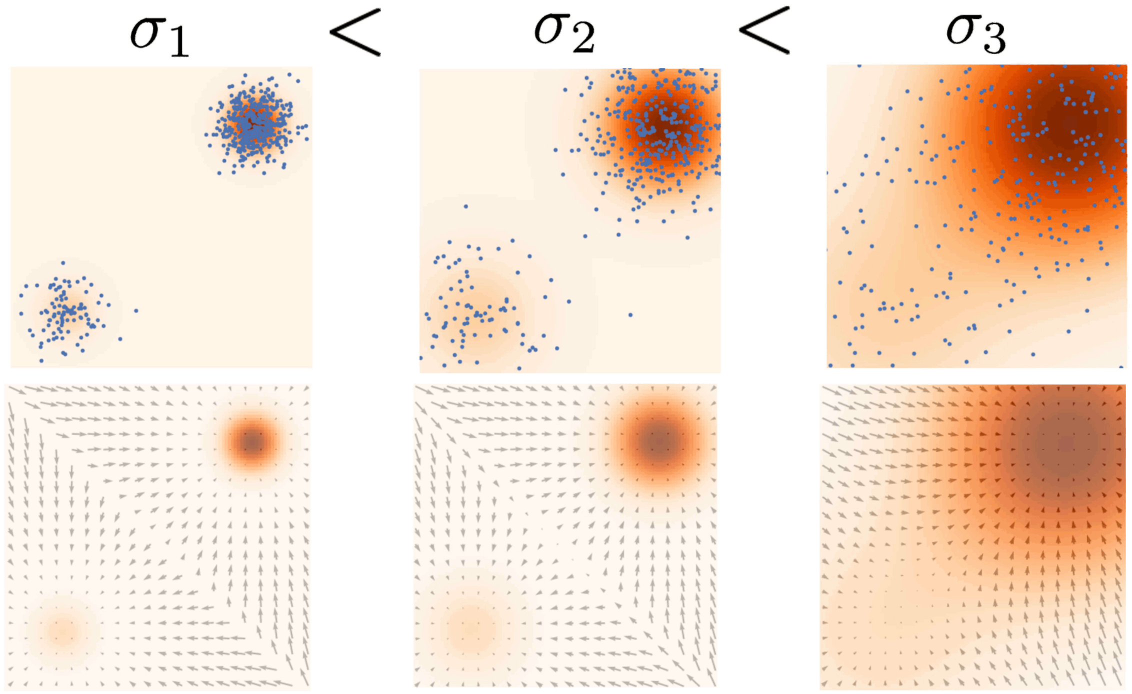
In our case, $\nabla_{\tilde{\mathbf{x}}} \log q_{\sigma_t}(\tilde{\mathbf{x}}|\mathbf{x}) = \nabla_{\tilde{\mathbf{x}}} \frac{-\|\tilde{\mathbf{x}}-\mathbf{x}\|^2}{2\sigma_t^2} = -\frac{\tilde{\mathbf{x}}-\mathbf{x}}{\sigma_t^2}$. Because $\tilde{\mathbf{x}} = \mathbf{x} + \sigma_t \mathbf{e}$ for standard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

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

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 σ_t, we can sample using annealed Langevin dynamics, where we utilize using gradually smaller noise rates σ_t.



https://yang-song.net/assets/img/score/multi_scale.jpg

Algorithm 1 Annealed Langevin dynamics.

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

Development of GANs

- 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

BigGAN

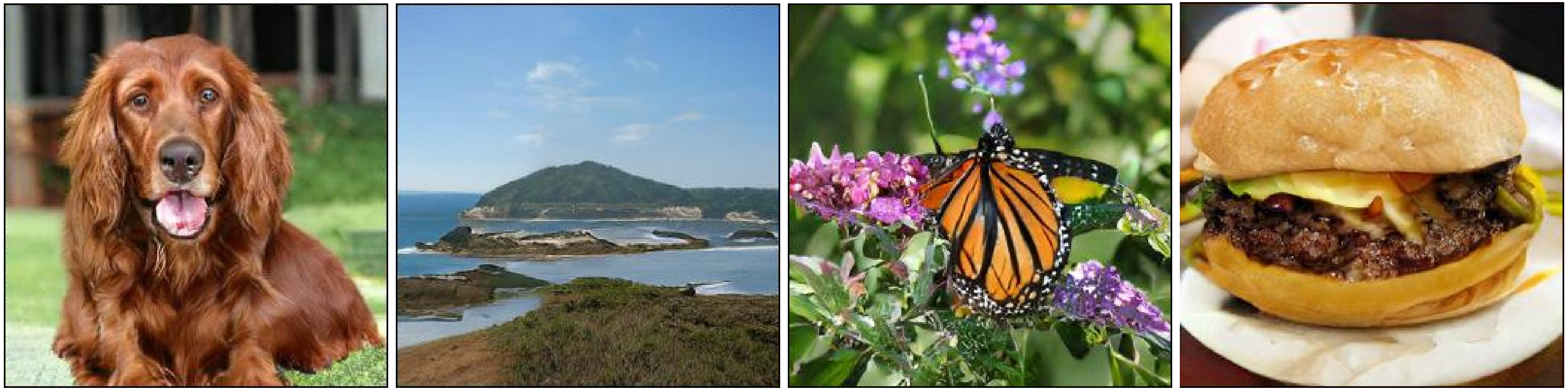


Figure 1 of "Large Scale GAN Training for High Fidelity Natural Image Synthesis", https://arxiv.org/abs/1809.11096

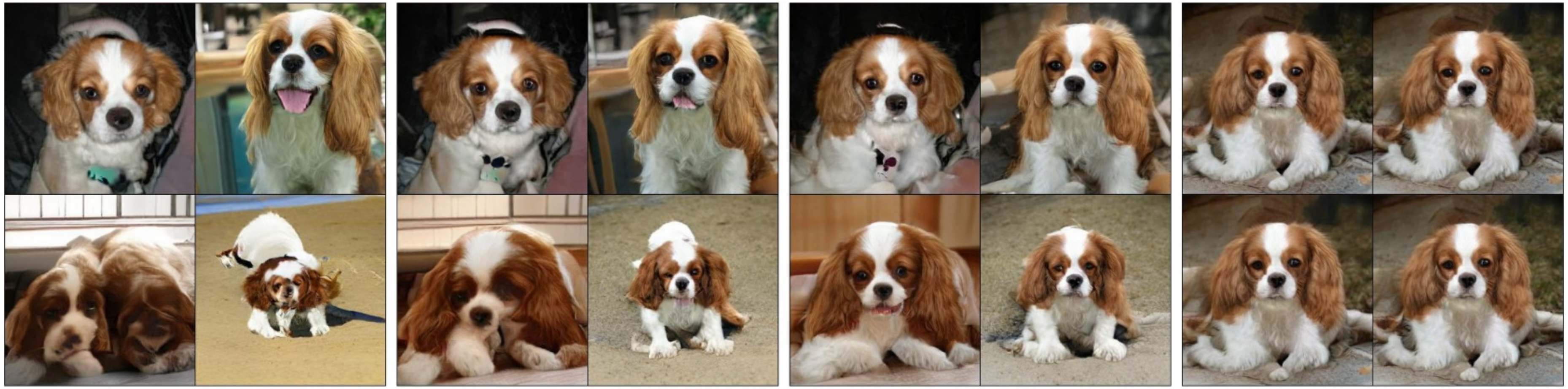


Figure 2 of "Large Scale GAN Training for High Fidelity Natural Image Synthesis", https://arxiv.org/abs/1809.11096

BigGAN

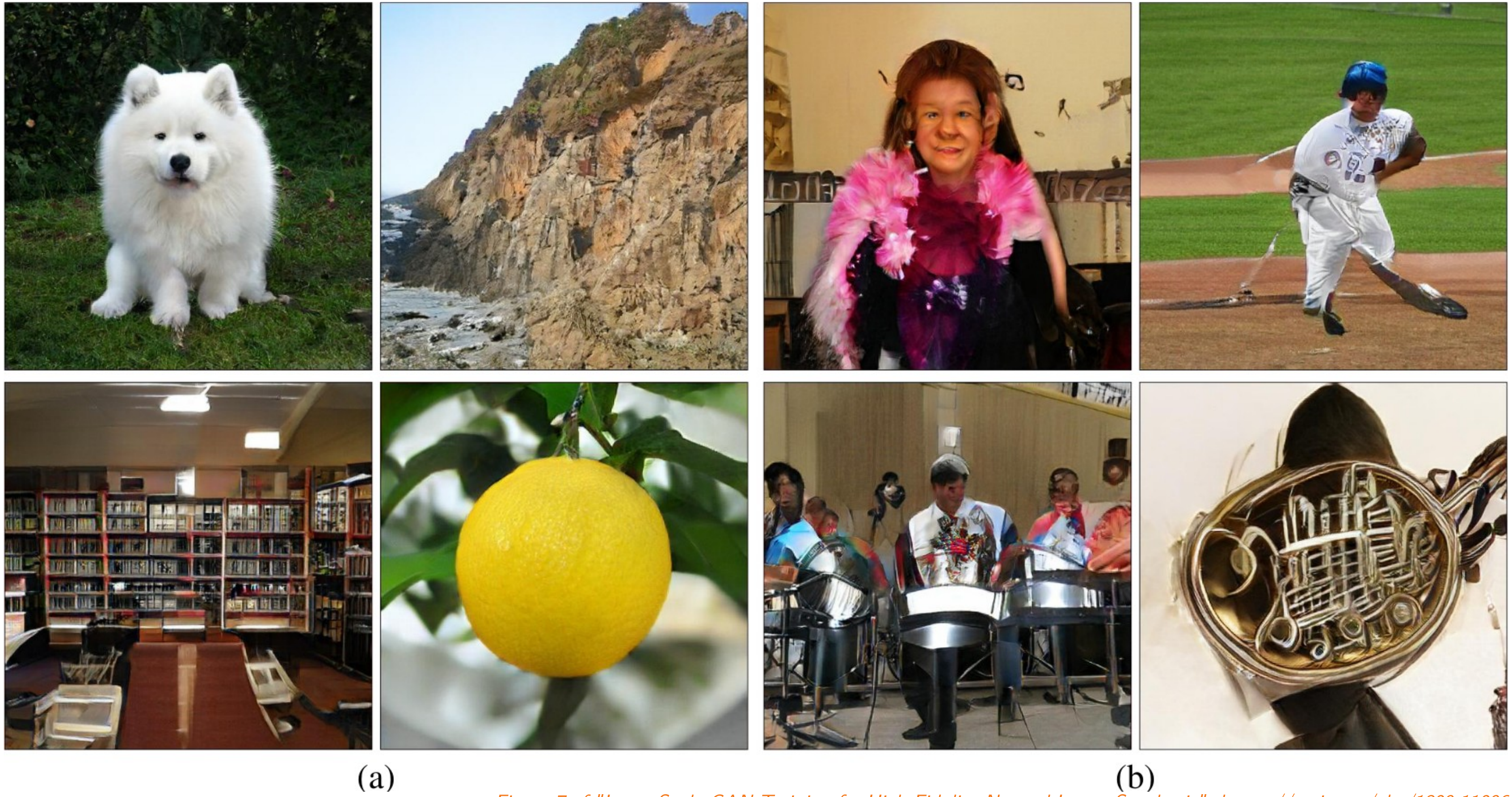


Figure 7 of "Large Scale GAN Training for High Fidelity Natural Image Synthesis", https://arxiv.org/abs/1809.11096

Development of VAEs

- 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

Development of Diffusion Models

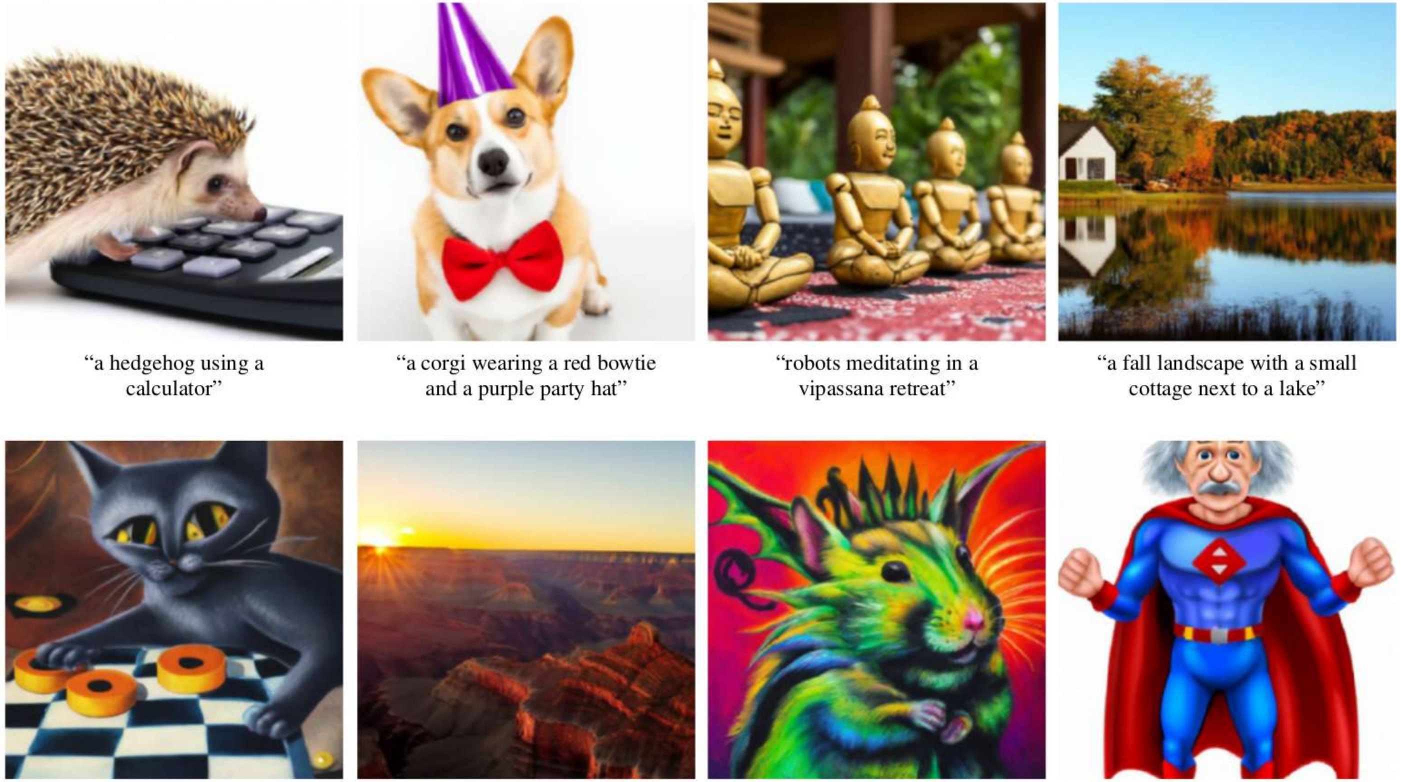
- Yang Song, Stefano Ermon: **Generative Modeling by Estimating Gradients of the Data Distribution** https://arxiv.org/abs/1907.05600
- Jonathan Ho, Ajay Jain, Pieter Abbeel: **Denoising Diffusion Probabilistic Models** https://arxiv.org/abs/2006.11239
- Jiaming Song, Chenlin Meng, Stefano Ermon: **Denoising 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

SR3 Super-Resolution via Diffusion

- 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

Diffusion-Based Text-Conditional Image Generation

- 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”

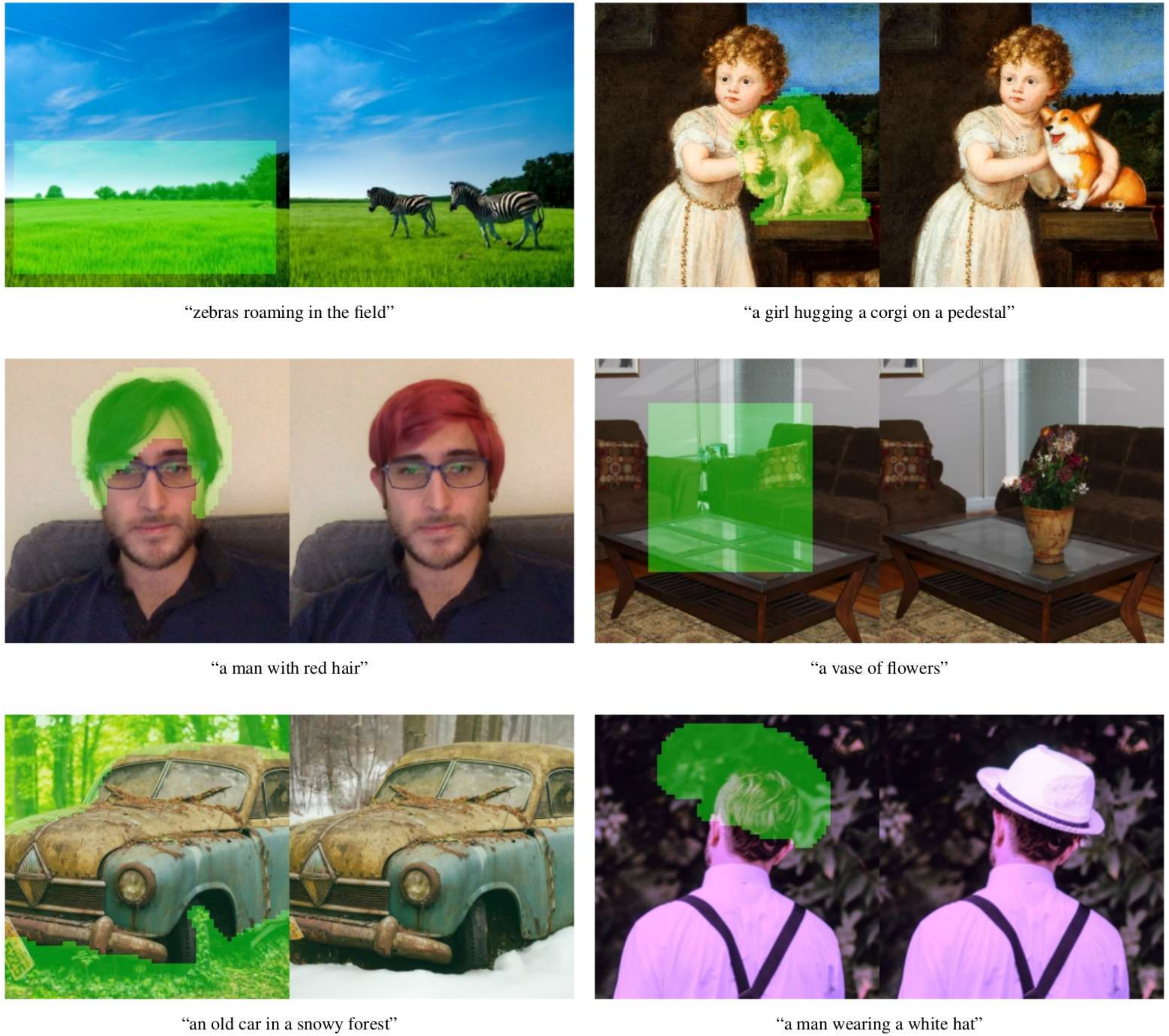
“a professional photo of a sunset behind the grand canyon”

“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

Diffusion-Based Text-Conditional Image Generation

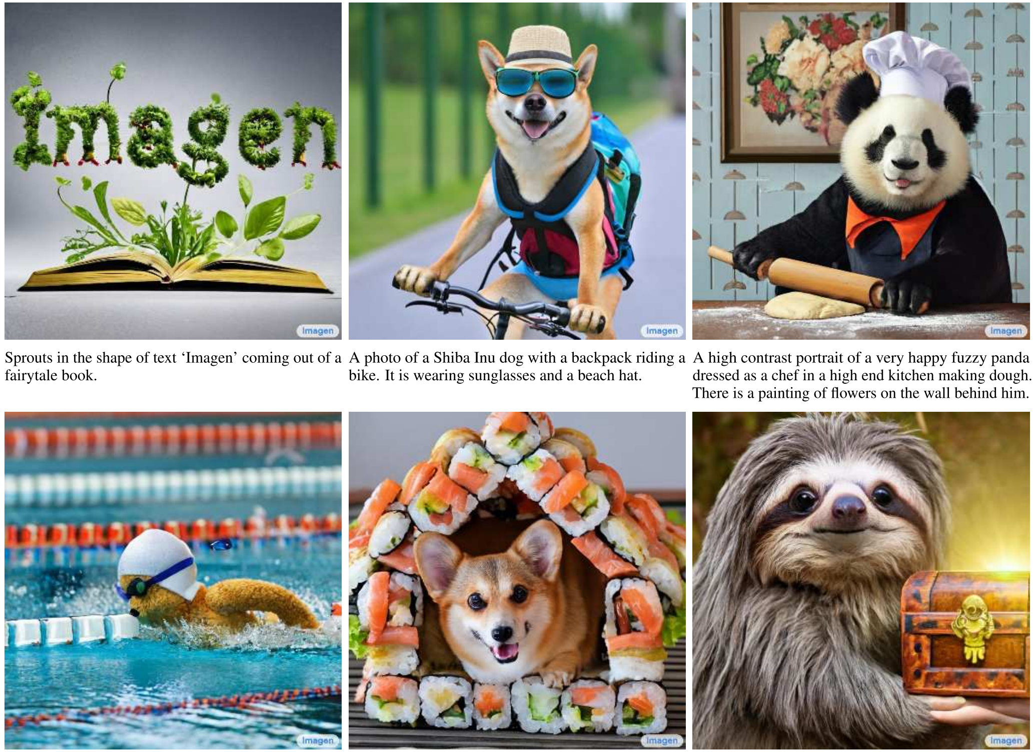


“zebras roaming in the field”
“a girl hugging a corgi on a pedestal”
“a man with red hair”
“a vase of flowers”
“an old car in a snowy forest”
“a man wearing a white hat”

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

Normalizing Flows

- 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

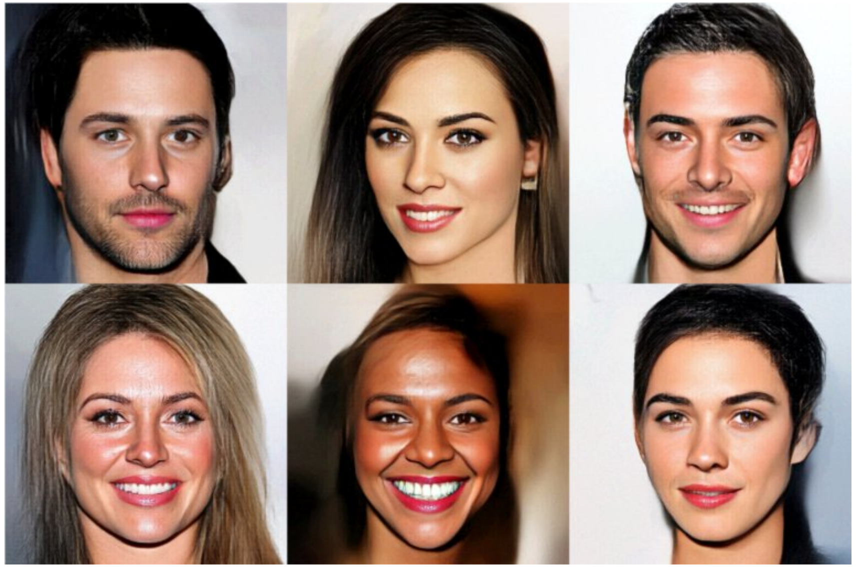


Figure 1 of "Glow: Generative Flowwith Invertible 1×1 Convolutions", https://arxiv.org/abs/1807.03039