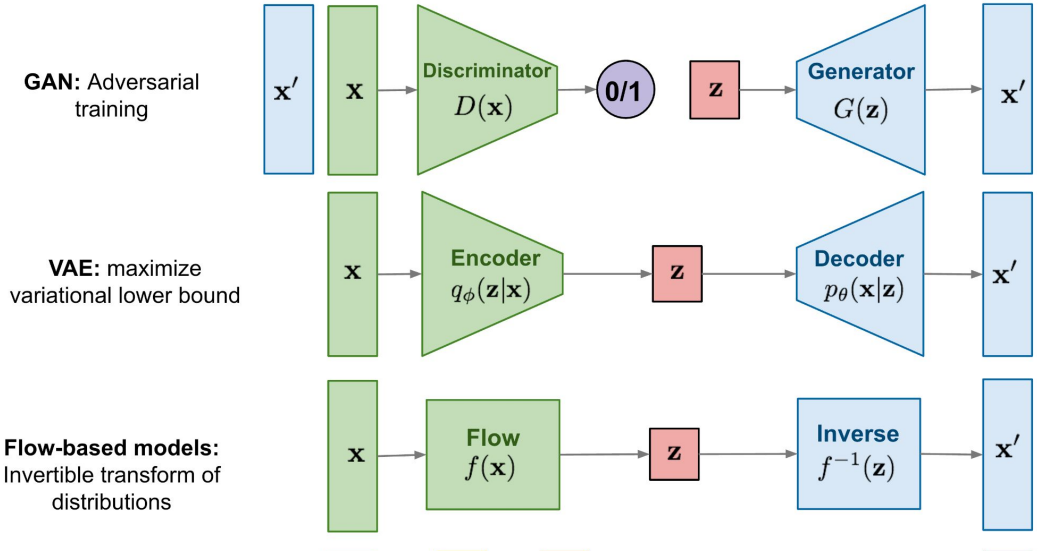
Introduction to diffusion models

Generative models

Generative models

Can you name some family of generative models you know?

Generative models

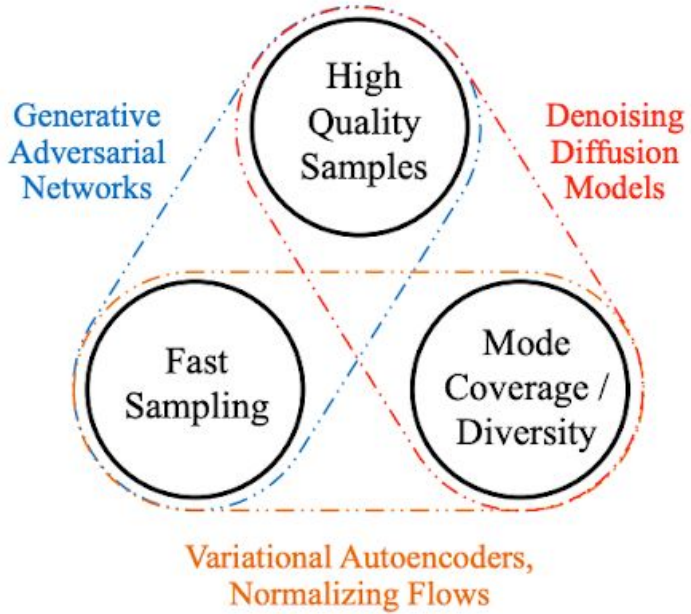


source: What are Diffusion Models?

Generative models

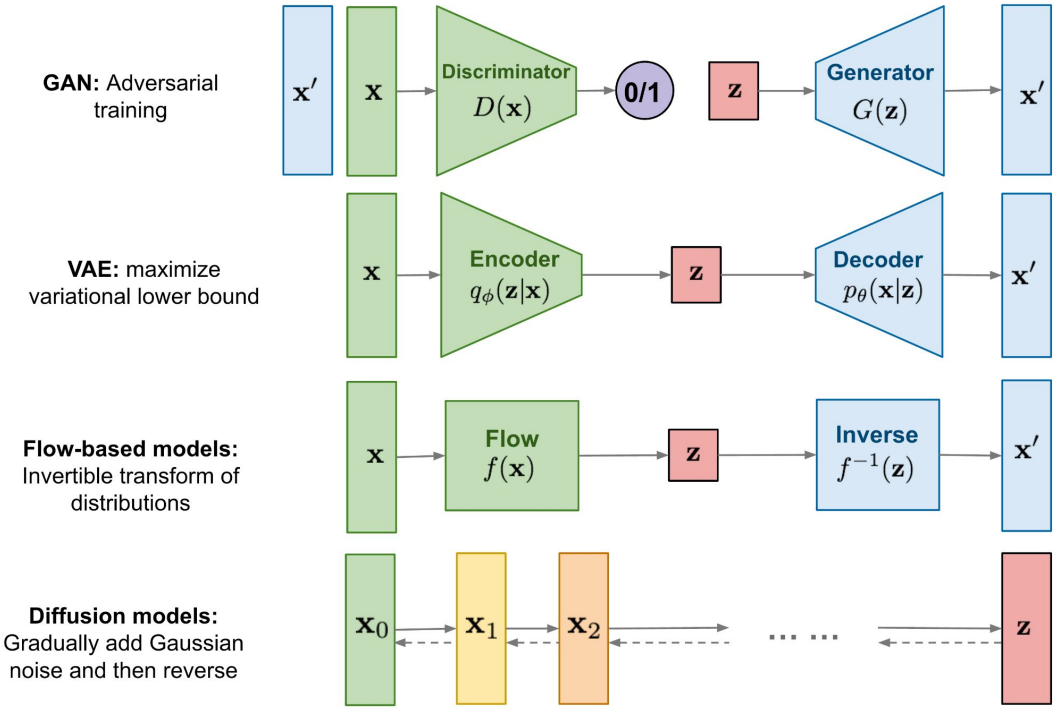
What are the advantages and drawbacks of these models?

Generative models



source: https://developer.nvidia.com/blog/improving-diffusion-models-as-an-alternative-to-gans-part-1/

Generative models



source: What are Diffusion Models?

Current best models

Flux

Stable Diffusion

Diffusion for other modalities

SORA

Stable diffusion audio

Diffusion models

Diffusion

- **slowly destroy structure in a data distribution** through an **iterative forward diffusion process**.
- learn a **reverse diffusion process** that **restores structure in data**

source: https://www.accessscience.com/highwire_display/entity_view/node/370286/focus_view

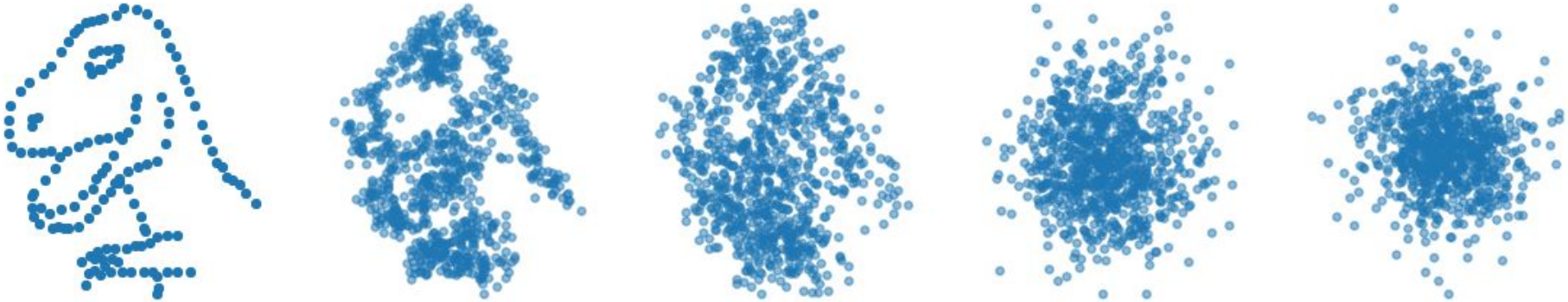
Diffusion

- Start from a data point in the original distribution
- Add a small amount of noise
- Repeat until it looks to pure noise

source: https://cvpr2022-tutorial-diffusion-models.github.io/

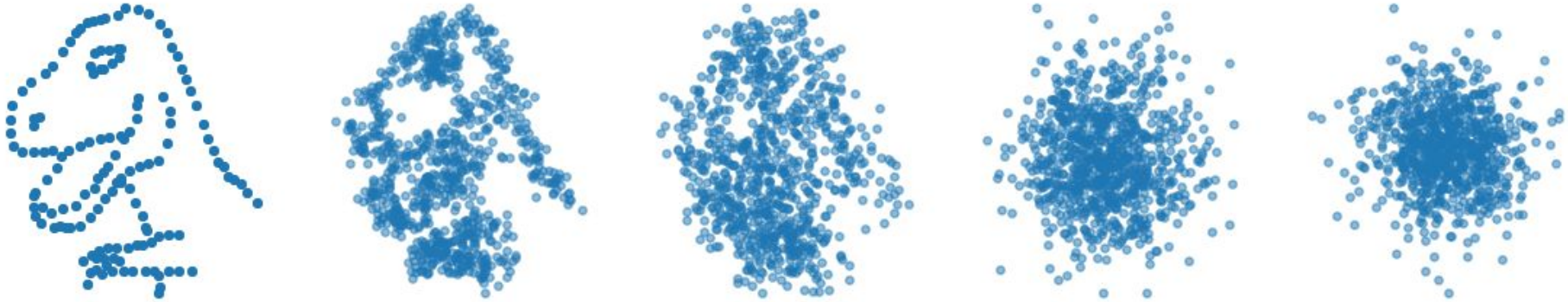
Diffusion

- At a given step we can kind of guess where was a sample at previous step.
- A sample y in the target space comes from a sample x in the original space with a probability close to $p(x)$
- If we learn how to reverse each step, we can sample from the $p(x)$



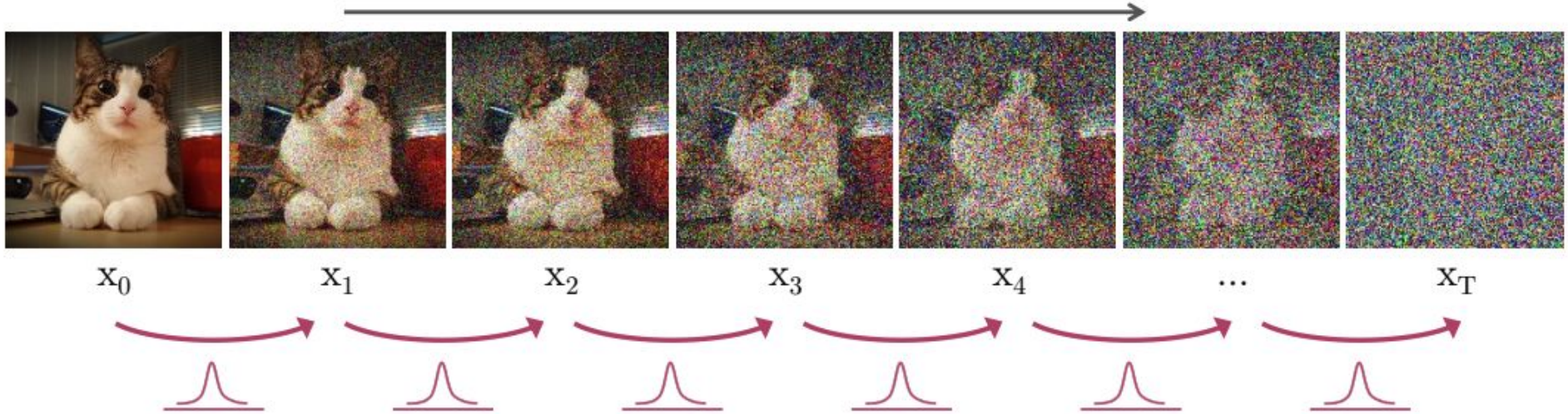
Diffusion

- **Forward pass** that gradually add noise to the data to move towards a distribution easy to sample
- **Backward** or **reverse process** that iteratively remove noise to samples to return into the original distribution



Forward pass

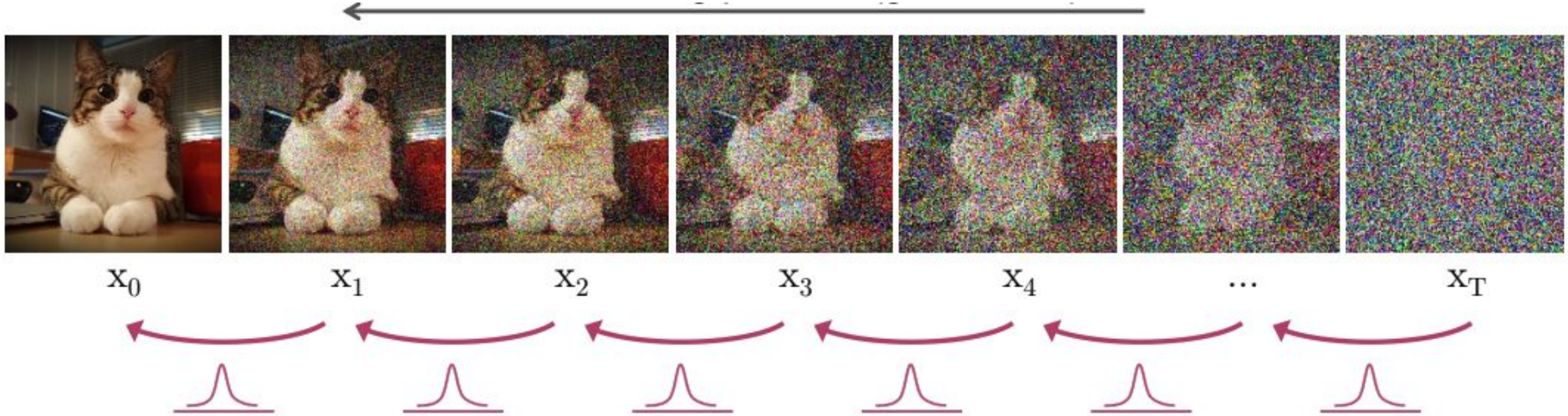
- Gradually inject noise to the data using a **fixed markov chain**
- Latent variables have **the same dimensionality** as the original ones
- Images are asymptotically transformed to **pure Gaussian noise**



source: https://cvpr2022-tutorial-diffusion-models.github.io/

Backward pass

- Going backward on this chain means that we can sample new images.
- **Problem**: we don't know this reverse process
- **Solution**: estimate it with a neural network parametrized by θ

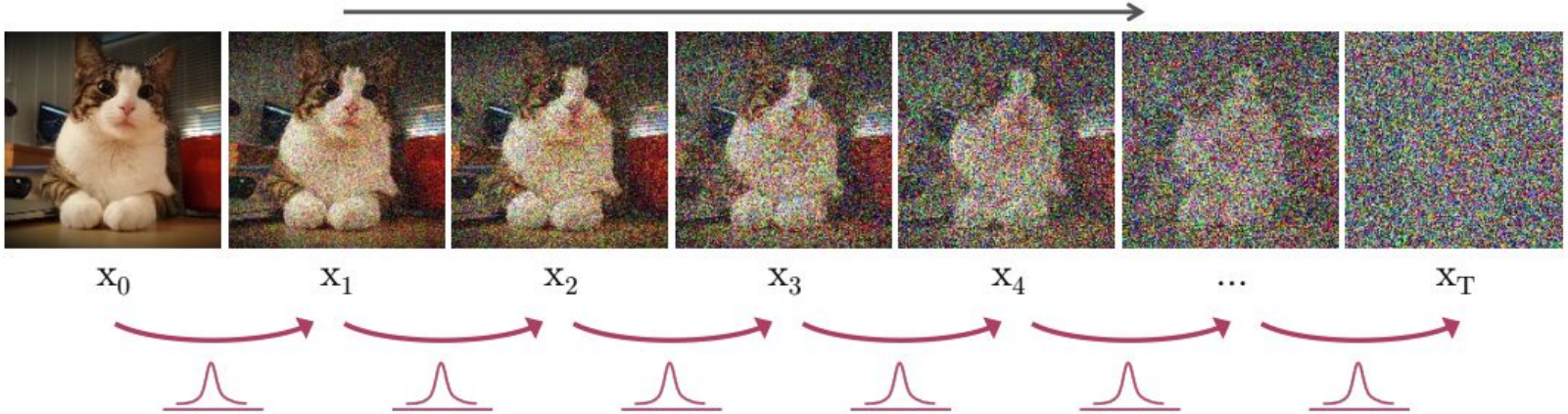


source: https://cvpr2022-tutorial-diffusion-models.github.io/

Forward pass

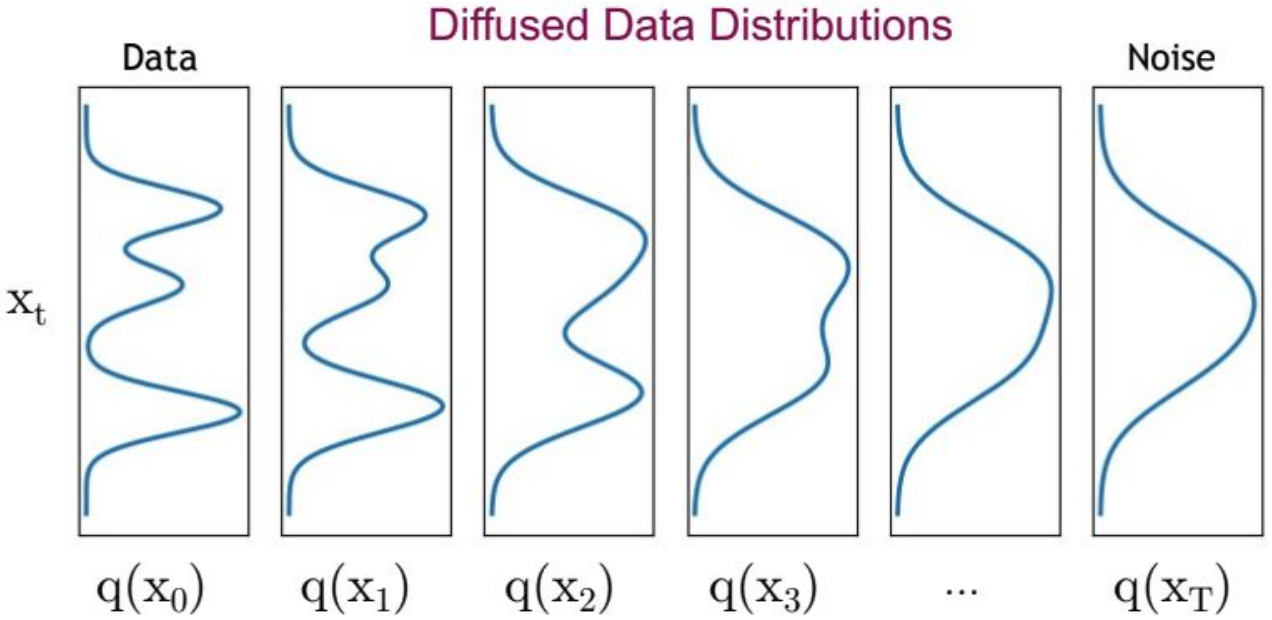
- Given a data point sampled from a source distribution $\mathbf{x}_0 \sim q(\mathbf{x})$, we define a forward diffusion process by adding a small amount of Gaussian noise in T steps

- This produces a sequence of latent variables $\mathbf{x}_1, \ldots, \mathbf{x}_T$



source: https://cvpr2022-tutorial-diffusion-models.github.io/

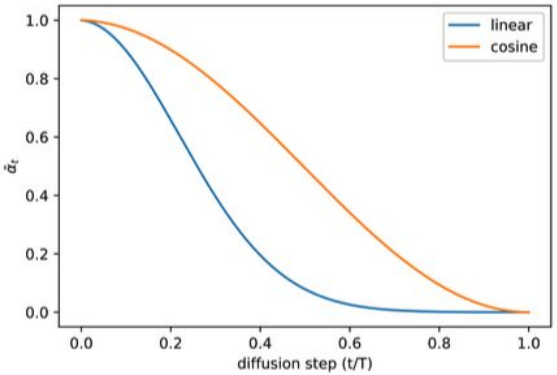
Forward pass



source: https://cvpr2022-tutorial-diffusion-models.github.io/

Forward pass

- $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$
- Where β_t is determined by a schedule (linear, cosine ...)



Linear

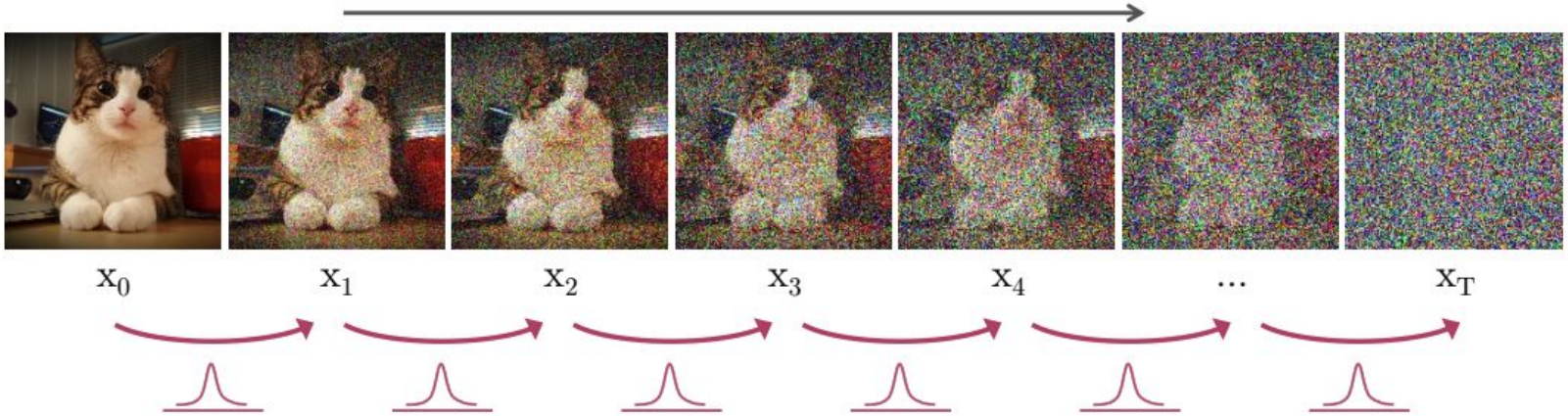
Cosine

source: Improved Denoising Diffusion Probabilistic Models, Alex Nichol and Prafulla Dhariwal (2020)

Sampling

- We want to learn the denoising process using a Neural network
- We need i.i.d data => different images at different time steps

- **Problem**: forward pass is expensive: $q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$



source: https://cvpr2022-tutorial-diffusion-models.github.io/

Sampling

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

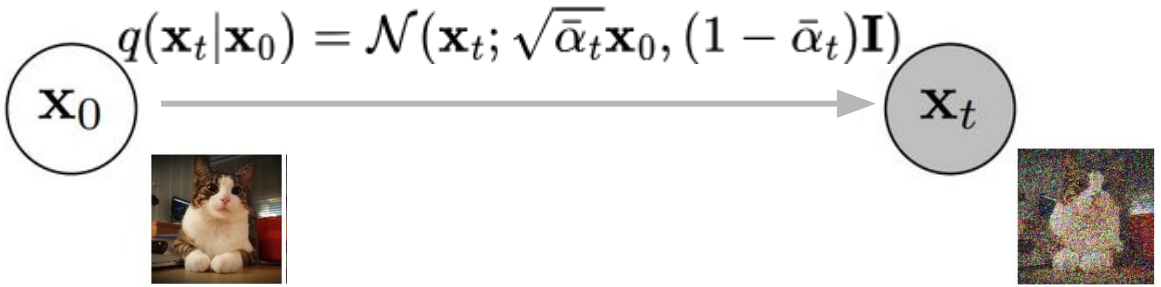
Reparametrization trick: $\mathcal{N}(\mu, \sigma^2) = \mu + \sigma * \epsilon$ with $\epsilon \sim \mathcal{N}(0, \mathbf{I})$

So we can write

$$
\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1} \\
&= \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_{t-1}}\epsilon_{t-2}) + \sqrt{1-\alpha_t}\epsilon_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\epsilon_{t-2} + \sqrt{1-\alpha_t}\epsilon_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\epsilon}_{t-2} \\
&= \cdots \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon \qquad \text{with } \bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s \\
q(\mathbf{x}_t|\mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})
\end{aligned}
$$

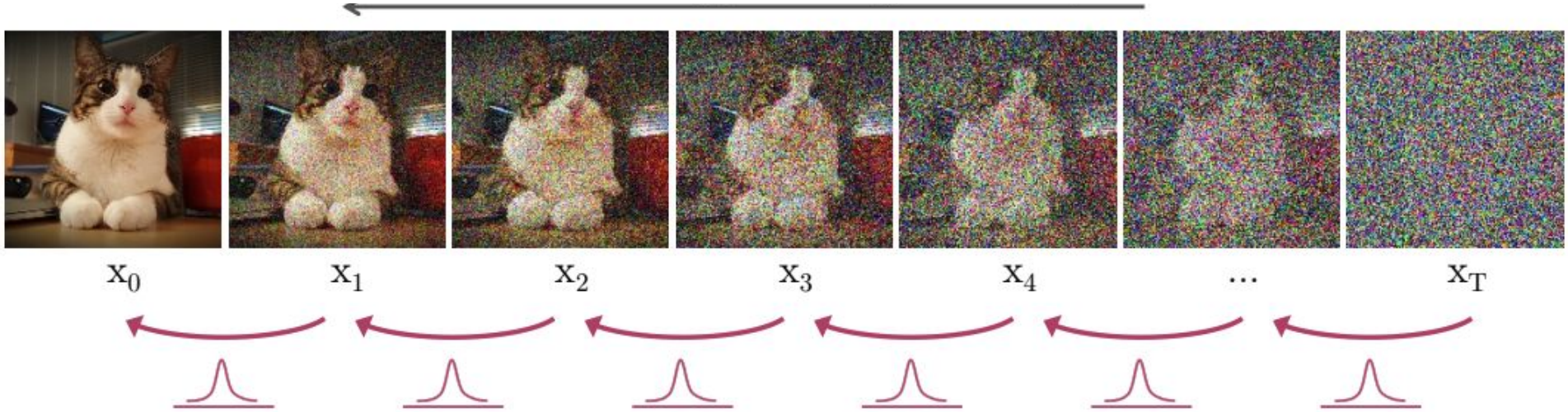
Forward pass

- We can now directly sample a noisy latent $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ for any $t \in [0, \ldots,]T$ with $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$ and $\epsilon \sim \mathcal{N}(0, \mathbf{I})$



Backward pass

- Going backward on this chain means that we can sample new images.
- **Problem**: we don't know this reverse process $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$
- **Solution**: estimate it with a model parametrized by θ: $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$



source: https://cvpr2022-tutorial-diffusion-models.github.io/

Backward pass

- Going backward on this chain means that we can sample new images.
- **Problem**: we don't know this reverse process $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$
- **Solution**: estimate it with a model parametrized by θ: $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$

We want to find θ that minimizes:

$$L = \mathbb{E}_{x_0,T\sim q}\left[\sum_{t=1}^{T} D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))\right]$$

It can be proven that minimizing L maximizes the likelihood of generating the dataset because it optimizes a lower bound for the same, through a process called variational inference (see Lilian Weng's blog for full derivations).

Full derivation (from Lilian Weng's blog)

As demonstrated in Fig. 2., such a setup is very similar to VAE and thus we can use the variational lower bound to optimize the negative log-likelihood.

$$
\begin{aligned}
-\log p_\theta(\mathbf{x}_0) &\leq -\log p_\theta(\mathbf{x}_0) + D_{\mathrm{KL}}(q(\mathbf{x}_{1:T}|\mathbf{x}_0) \| p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0)) \\
&= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T}) / p_\theta(\mathbf{x}_0)}\Big] \\
&= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_q\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})} + \log p_\theta(\mathbf{x}_0)\Big] \\
&= \mathbb{E}_q\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\Big] \\
\text{Let } L_{\mathrm{VLB}} &= \mathbb{E}_{q(\mathbf{x}_{0:T})}\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\Big] \geq -\mathbb{E}_{q(\mathbf{x}_0)} \log p_\theta(\mathbf{x}_0)
\end{aligned}
$$

It is also straightforward to get the same result using Jensen's inequality. Say we want to minimize the cross entropy as the learning objective,

$$
\begin{aligned}
L_{\mathrm{CE}} &= -\mathbb{E}_{q(\mathbf{x}_0)} \log p_\theta(\mathbf{x}_0) \\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \log \Big(\int p_\theta(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T} \Big) \\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \log \Big(\int q(\mathbf{x}_{1:T}|\mathbf{x}_0) \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} d\mathbf{x}_{1:T} \Big) \\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \log \Big(\mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \Big) \\
&\leq -\mathbb{E}_{q(\mathbf{x}_{0:T})} \log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \\
&= \mathbb{E}_{q(\mathbf{x}_{0:T})}\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\Big] = L_{\mathrm{VLB}}
\end{aligned}
$$

Full derivation (from Lilian Weng's blog)

To convert each term in the equation to be analytically computable, the objective can be further rewritten to be a combination of several KL-divergence and entropy terms (See the detailed step-by-step process in Appendix B in Sohl-Dickstein et al., 2015):

$$
\begin{aligned}
L_{\text{VLB}} &= \mathbb{E}_{q(\mathbf{x}_{0:T})}\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\Big] \\
&= \mathbb{E}_q\Big[\log \frac{\prod_{t=1}^T q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_T)\prod_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=1}^T \log \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\Big] \\
&= \mathbb{E}_q\Big[-\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)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \Big(\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\cdot\frac{q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}\Big) + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\Big] \\
&= \mathbb{E}_q\Big[-\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)} + \sum_{t=2}^T \log \frac{q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\Big] \\
&= \mathbb{E}_q\Big[-\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)}\Big] \\
&= \mathbb{E}_q\Big[\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)\Big] \\
&= \mathbb{E}_q[\underbrace{D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)\parallel 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)\parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}]
\end{aligned}
$$

Let's label each component in the variational lower bound loss separately:

$$
\begin{aligned}
L_{\text{VLB}} &= L_T + L_{T-1} + \cdots + L_0 \\
\text{where } L_T &= D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)\parallel p_\theta(\mathbf{x}_T)) \\
L_t &= D_{\text{KL}}(q(\mathbf{x}_t|\mathbf{x}_{t+1},\mathbf{x}_0)\parallel p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1})) \text{ for } 1 \le t \le T-1 \\
L_0 &= -\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)
\end{aligned}
$$

Every KL term in L_{VLB} (except for L_0) compares two Gaussian distributions and therefore they can be computed in closed form. L_T is constant and can be ignored during training because q has no learnable parameters and $\mathbf{x}_T$ is a Gaussian noise. Ho et al. 2020 models L_0 using a separate discrete decoder derived from $\mathcal{N}(\mathbf{x}_0; \boldsymbol{\mu}_\theta(\mathbf{x}_1, 1), \boldsymbol{\Sigma}_\theta(\mathbf{x}_1, 1))$.

Backward pass

$$L = \mathbb{E}_{x_0, T\sim q}\left[\sum_{t=1}^{T} D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))\right]$$

We don't know $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ but we actually know $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})q(\mathbf{x}_{t-1}|\mathbf{x}_0)q(\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)q(\mathbf{x}_0)} \quad \text{(Bayes' rule)}$$

$$= \mathcal{N}(\mu(\mathbf{x}_t, \mathbf{x}_0), \Sigma(t)\mathbf{I}) \quad \text{(By subbing in the distribution formulas and doing the algebra see next slide for details)}$$

with $\mu(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\alpha_t}(1-\alpha_{t-1})\mathbf{x}_t + \sqrt{\alpha_{t-1}}(1-\alpha_t)\mathbf{x}_0}{1-\bar{\alpha}_t}$ and $\Sigma(t) = \frac{(1-\alpha_t)(1-\alpha_{t-1})}{1-\bar{\alpha}_t}$

Full derivation (from Lilian Weng's blog)

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

Using Bayes' rule, we have:

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

where $C(\mathbf{x}_t, \mathbf{x}_0)$ is some function not involving $\mathbf{x}_{t-1}$ and details are omitted. Following the standard Gaussian density function, the mean and variance can be parameterized as follows (recall that $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{T} \alpha_i$):

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

Thanks to the nice property, we can represent $\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t)$ and plug it into the above equation and obtain:

$$\begin{aligned}
\tilde{\boldsymbol{\mu}}_t &= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t) \\
&= \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\Big)
\end{aligned}$$

Backward pass

$$L = \mathbb{E}_{x_0, T \sim q} \left[\sum_{t=1}^{T} D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) || p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) \right]$$

Hypothesis: If β_t is small enough $q(x_{t-1}|x_t)$ is gaussian.

=> We want to learn $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}, \mu_\theta(x_t, t), \sum_\theta(x_t, t))$

Denoising Diffusion Probabilistic Models
J. Ho, A. Jain, P. Abbeel. 2020.

=> For numerical stability set $\sum_\theta = \sum(t)$

Backward pass

$$L = \mathbb{E}_{x_0, T\sim q}\left[\sum_{t=1}^{T} D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))\right]$$

KL between two gaussians (with same variance $\sum(t)$)

$$\Rightarrow L = \sum_{t=1}^{T} \mathbb{E}_{x_0, T\sim q}\left[\frac{1}{2\Sigma(t)}||\mu(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t)||^2\right]$$

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

We can use $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_0 \Rightarrow x_0 = \frac{x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_0}{\sqrt{\bar{\alpha}_t}}$ to replace x_0 above

$$\Rightarrow \mu(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}\epsilon$$

Backward pass

$$L = \mathbb{E}_{x_0, T\sim q}\left[\sum_{t=1}^{T} D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))\right]$$

KL between two gaussians (with same variance $\sum(t)$)

$$\Rightarrow L = \sum_{t=1}^{T} \mathbb{E}_{x_0, T\sim q}\left[\frac{1}{2\Sigma(t)}||\mu(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t)||^2\right]$$

With $\mu(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}\epsilon$

We define then $\mu_\theta(\mathbf{x}_t)$ to match above: $\mu_\theta(\mathbf{x}_t) = \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}\epsilon_\theta(\mathbf{x}_t, t)$

$$\Rightarrow L = \sum_{t=1}^{T} \mathbb{E}_{x_0, \epsilon\sim q}\left[\frac{(1-\alpha_t)^2}{2\Sigma(t)\alpha_t(1-\bar{\alpha}_t)}||\epsilon - \epsilon_\theta(\mathbf{x}_t, t)||^2\right]$$

Backward pass

$$L = \sum_{t=1}^{T} \mathbb{E}_{x_0,\epsilon\sim q}\left[\frac{(1-\alpha_t)^2}{2\Sigma(t)\alpha_t(1-\bar{\alpha}_t)}||\epsilon - \epsilon_\theta(\mathbf{x}_t, t)||^2\right]$$

Denoising Diffusion Probabilistic Models
J. Ho, A. Jain, P. Abbeel. 2020.

=> ignoring the weighting improves the quality of results

$$L = \sum_{t=1}^{T} \mathbb{E}_{x_0,\epsilon\sim q}\left[\frac{(1-\alpha_t)^2}{2\Sigma(t)\alpha_t(1-\bar{\alpha}_t)}||\epsilon - \epsilon_\theta(\mathbf{x}_t, t)||^2\right]$$

$$L_{simple} = \sum_{t=1}^{T} \mathbb{E}_{x_0,\epsilon\sim q}\left[||\epsilon - \epsilon_\theta(\mathbf{x}_t, t)||^2\right]$$

with $x_t = \sqrt{\overline{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\overline{\alpha}_t}\epsilon$

Training:

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

Algorithm 1 Training

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

Sampling:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}, \mu_\theta(x_t, t), \sum_\theta(x_t, t))$$

With $\mu_\theta(\mathbf{x}_t) = \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}\epsilon_\theta(\mathbf{x}_t, t)$ and $\sum_\theta = \sum(t)$

Algorithm 2 Sampling

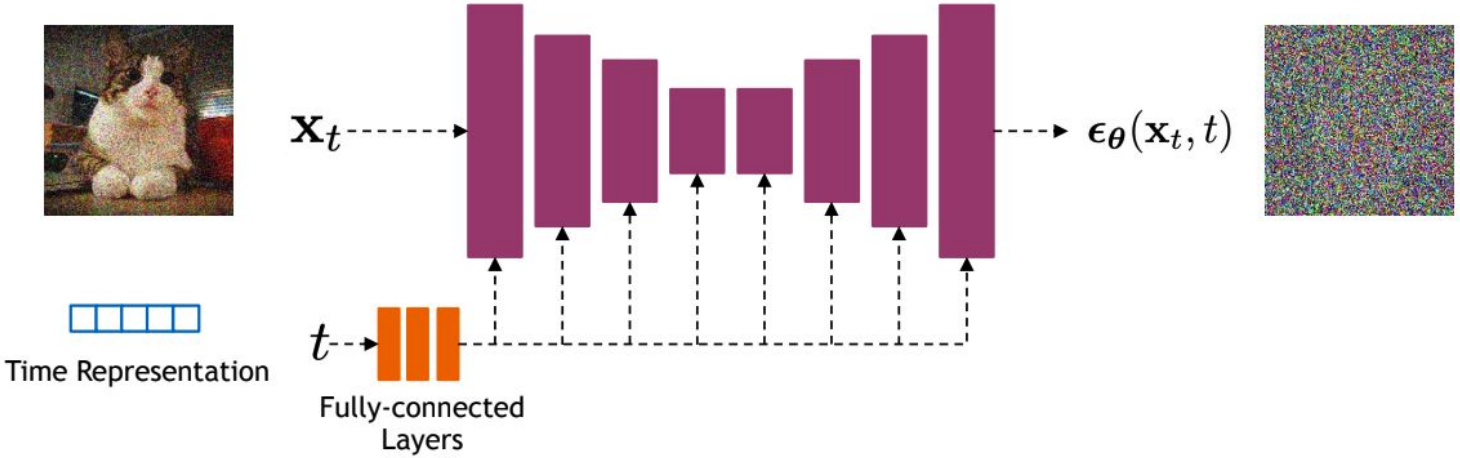
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,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

Question:

What does the network predict?

A/ Noise to remove to obtain x_{t-1}

B/ Noise to remove to obtain x_0



source: https://cvpr2022-tutorial-diffusion-models.github.io/

In practice

Training:

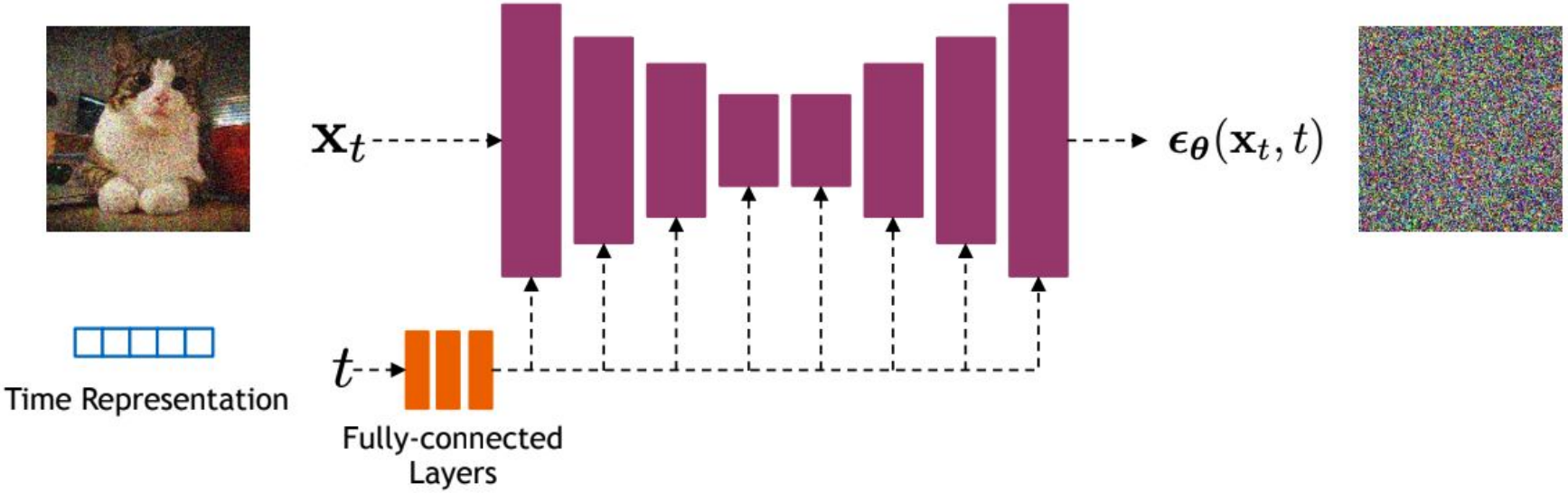
Algorithm 1 Training

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

Algorithm 2 Sampling

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

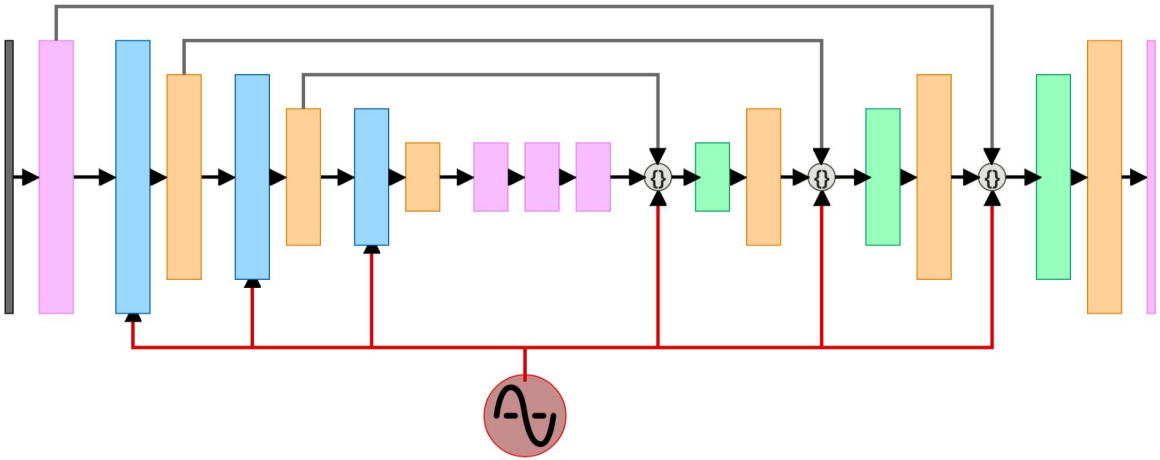
Unet denoiser



source: https://cvpr2022-tutorial-diffusion-models.github.io/

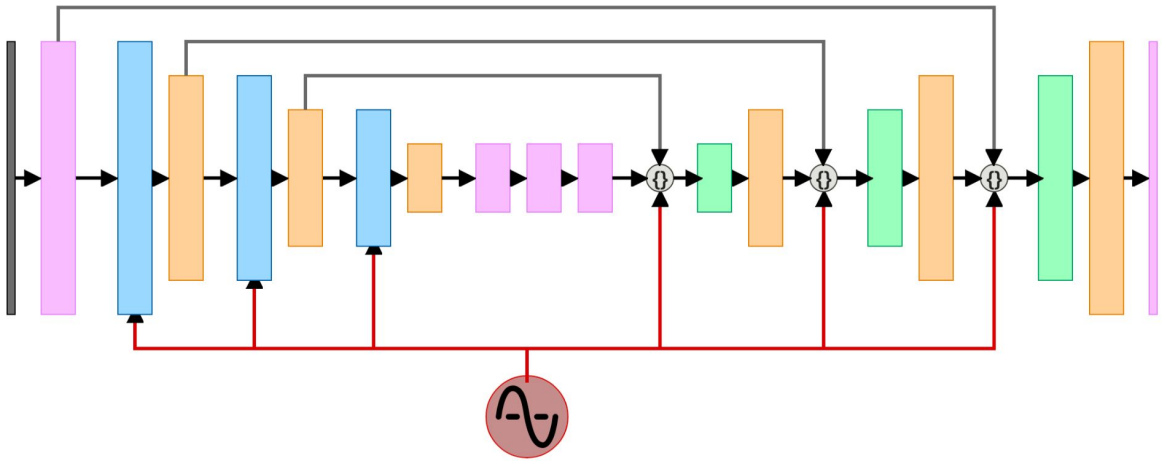
Unet denoiser

Input
Conv Block
Downsample Block
Self-Attention Block
Upsample Block
Embedding vector

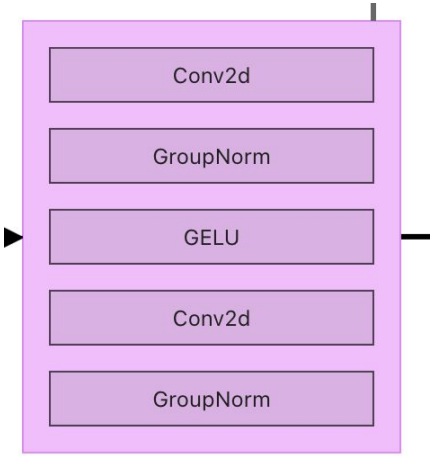


source: https://erdem.pl/2023/11/step-by-step-visual-introduction-to-diffusion-models#reverse-diffusion-diagram

Unet denoiser

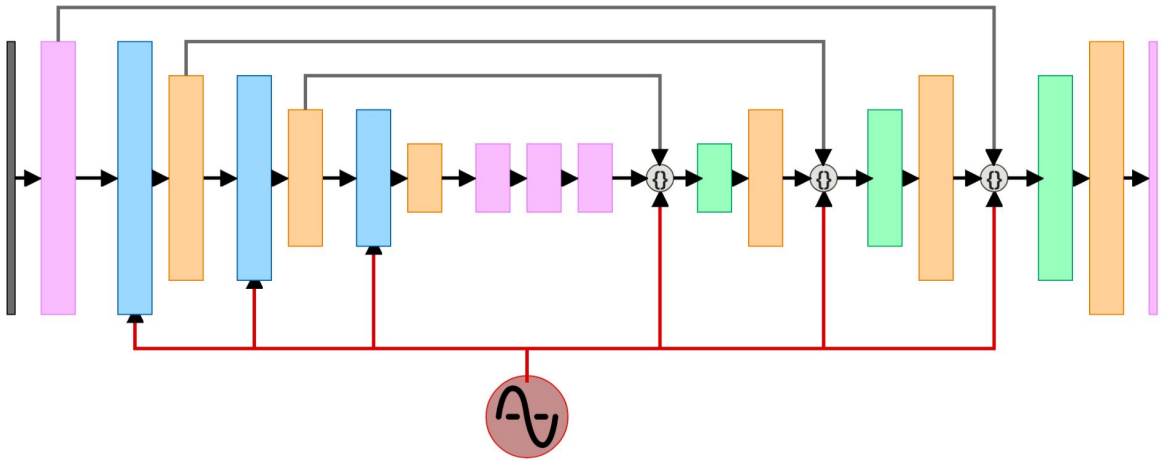


Input
Conv Block
Downsample Block
Self-Attention Block
Upsample Block
Embedding vector

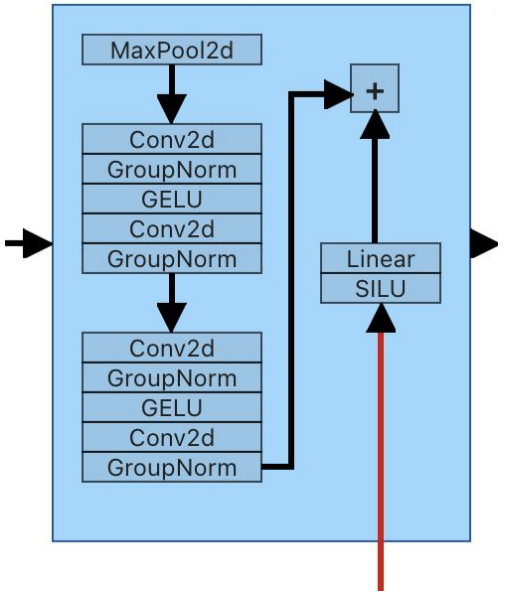


source: https://erdem.pl/2023/11/step-by-step-visual-introduction-to-diffusion-models#reverse-diffusion-diagram

Unet denoiser

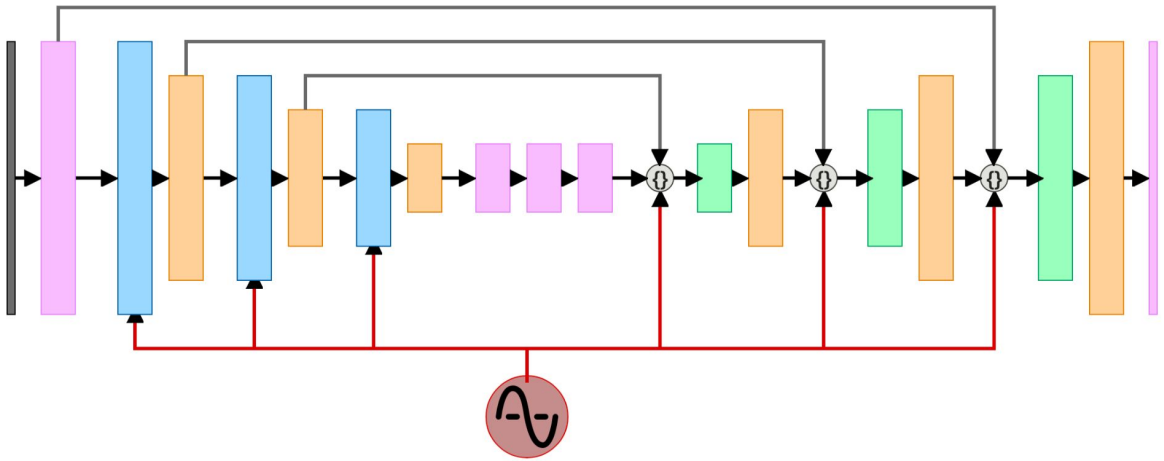


Input
Conv Block
Downsample Block
Self-Attention Block
Upsample Block
Embedding vector

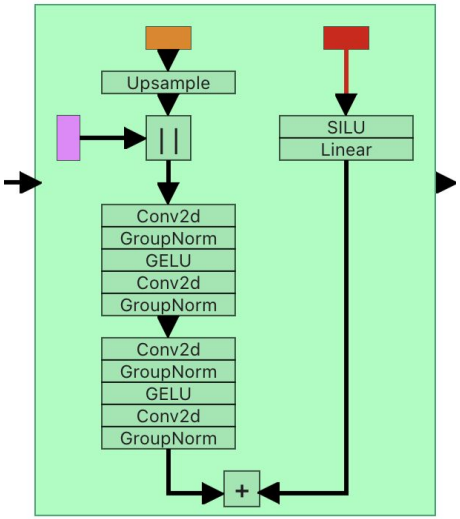


source: https://erdem.pl/2023/11/step-by-step-visual-introduction-to-diffusion-models#reverse-diffusion-diagram

Unet denoiser



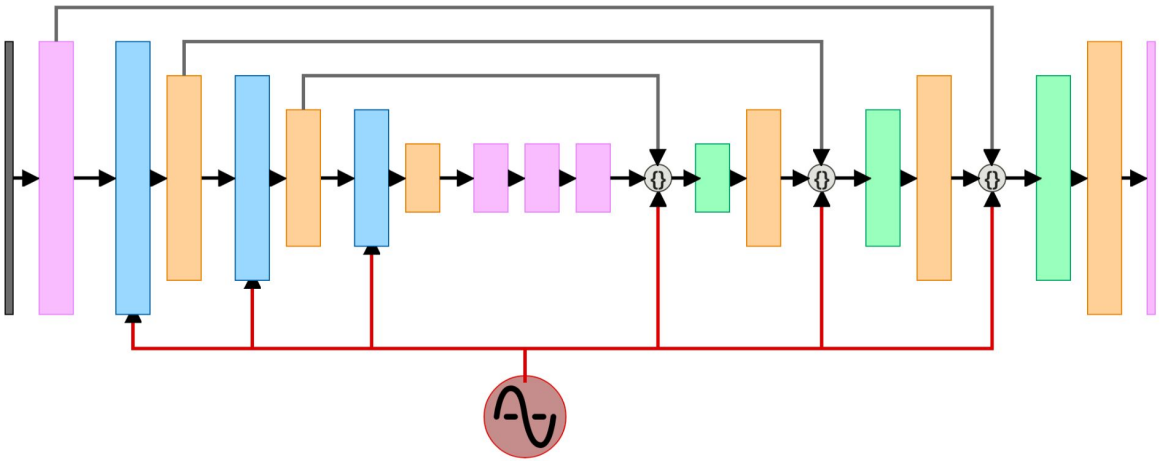
Input
Conv Block
Downsample Block
Self-Attention Block
Upsample Block
Embedding vector

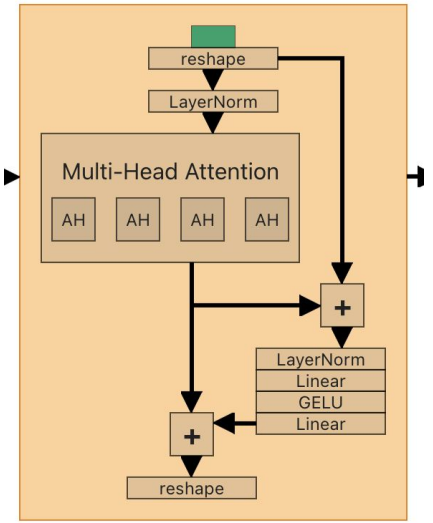


source: https://erdem.pl/2023/11/step-by-step-visual-introduction-to-diffusion-models#reverse-diffusion-diagram

Unet denoiser

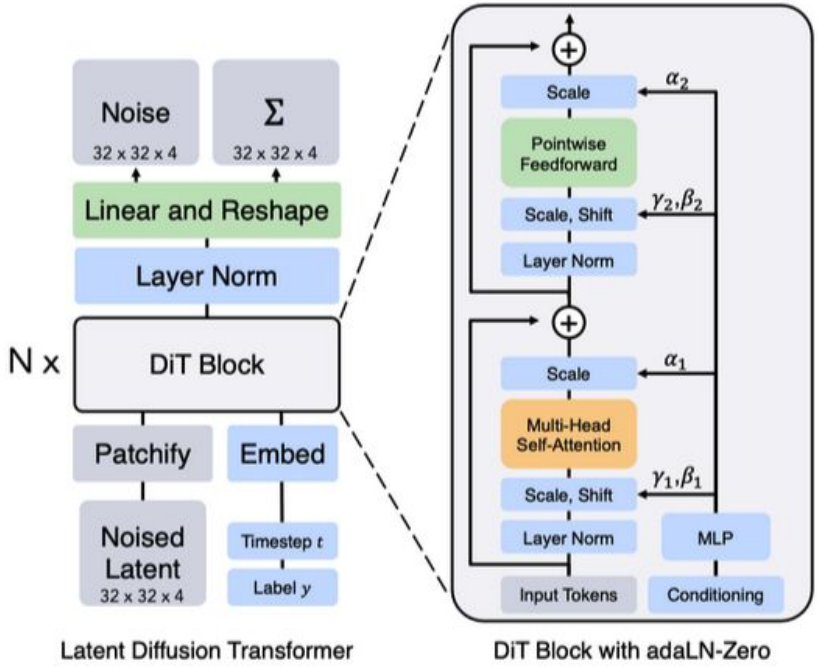
Input
Conv Block
Downsample Block
Self-Attention Block
Upsample Block
Embedding vector





source: https://erdem.pl/2023/11/step-by-step-visual-introduction-to-diffusion-models#reverse-diffusion-diagram

Diffusion transformer

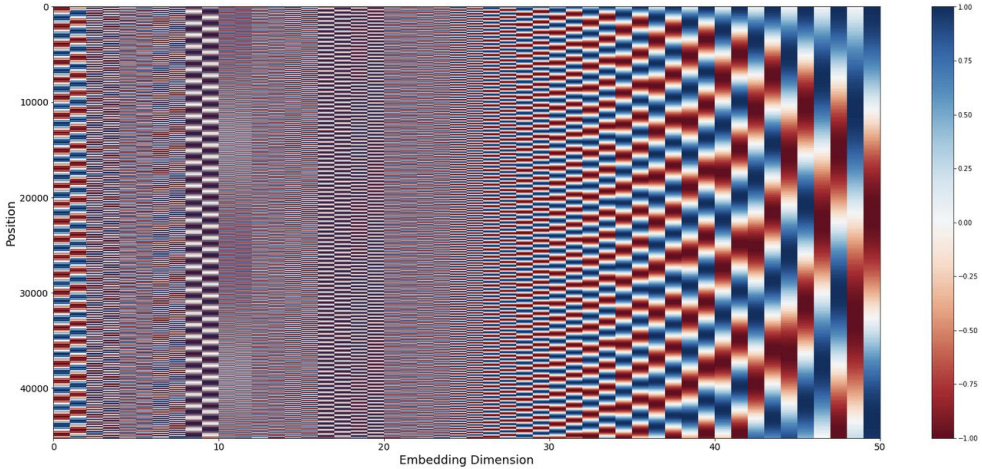


Scalable Diffusion Models with Transformers: William Peebles, Saining Xie

Timestep embedding

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$



Generated with : Tensorflow - Positional encoding code

- **Uniqueness**: allow the model to distinguish different noise levels.
- **Smoothness**: The encoding varies smoothly as the timestep changes, which helps the model interpolate between noise levels.
- **Scale invariance**: Using multiple frequency components (through different powers of 10000) allows the encoding to capture both fine and coarse-grained timestep differences.
- **Fixed bounds**: keep the encoded values bounded between -1 and 1, which helps with numerical stability.

Recap

- **Forward process** to add noise to the input sample using a noise schedule
- **Reverse process** remove small amount of noise at every step
- A **neural network** predicts the noise to be removed to move towards X0 (not from Xt to Xt-1!)
- Usually a **U-Net or a DiT**
- **Time** is projected using **sinusoidal encoding**

Conditioning

- Class (e.g. dog)

- Prompt
  (e.g. an astronaut on a horse)

- Other, depth maps, edges, other image...

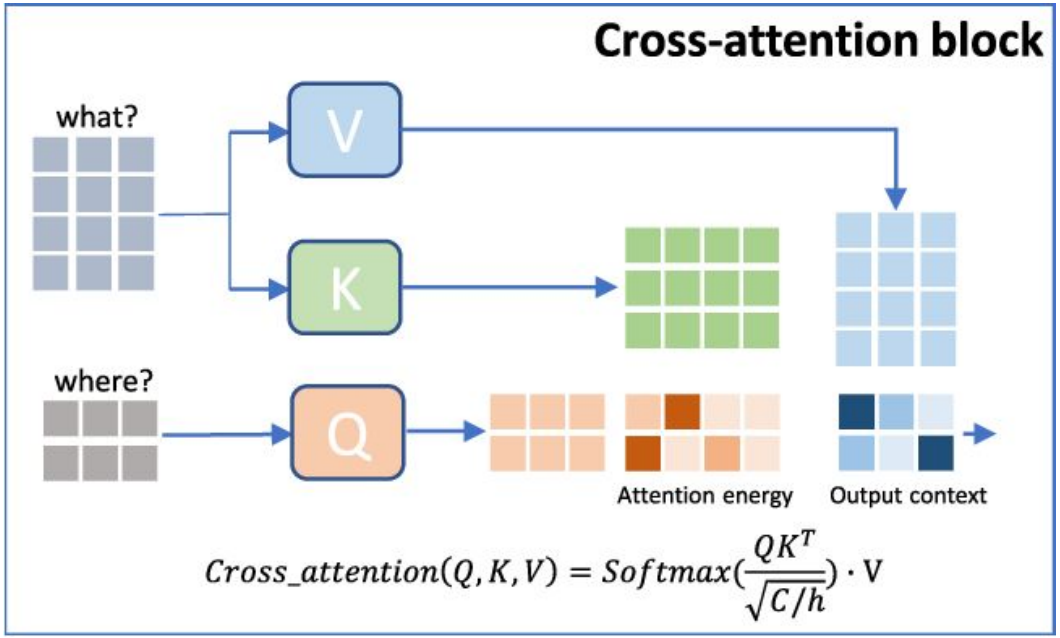
Class Conditioning

Usually just an embedding summed or concatenated to the timestep embedding or the noisy input

In practice mostly summed to the timestep embedding

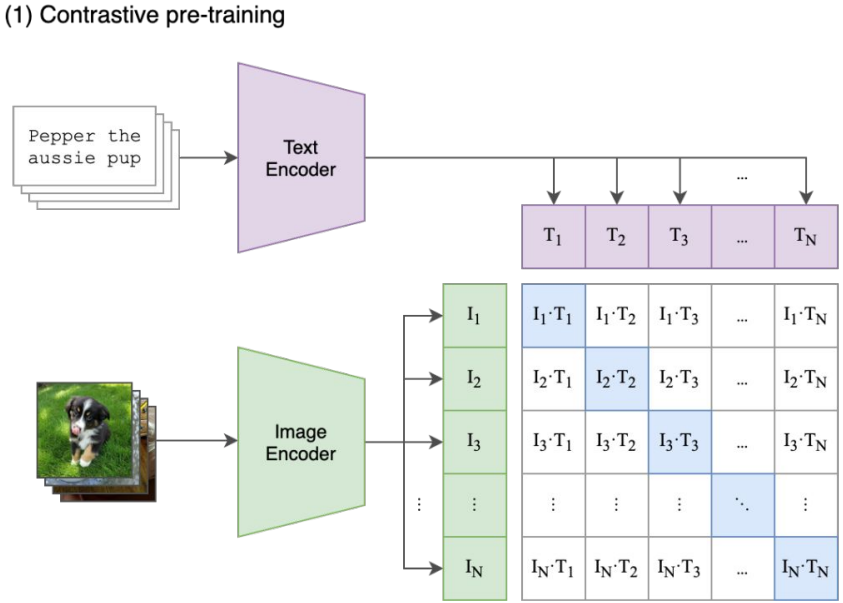
Other types of conditioning

More complex conditioning (i.e. text, images, depth maps...) usually pass through cross attention blocks



Text conditioning

- Rich text representation space: CLIP, T5, some LLMs now...



Learning Transferable Visual Models From Natural Language Supervision:
Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, Ilya Sutskever

Cascade networks

- most of the modeling capacity can be dedicated to low resolutions
- individual models can be trained independently with specific architecture choices
- Lower resolution image conditioning is provided by channelwise concatenation of the low resolution image, processed by bilinear or bicubic upsampling to the desired resolution, with the reverse process input xt
- Apply additional data augmentation on the hi-res training

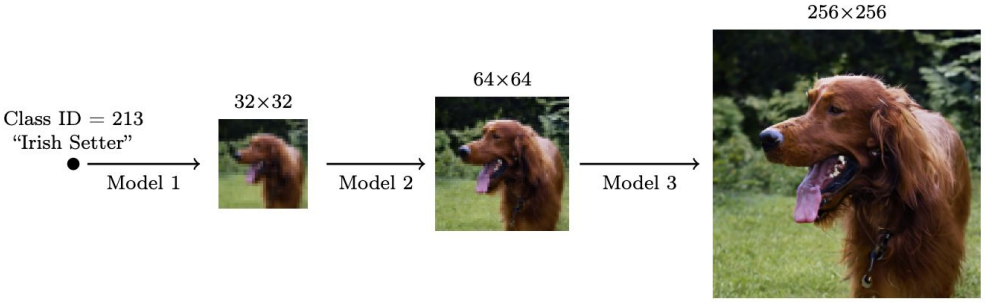
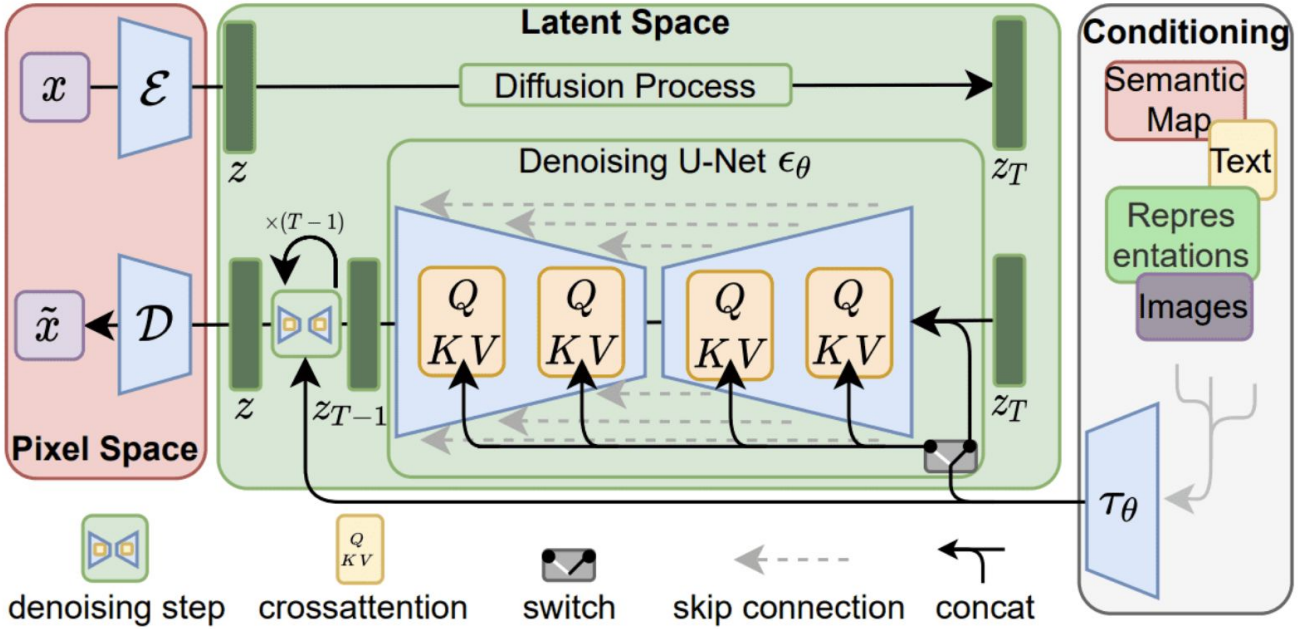


Figure 1: A cascaded diffusion model comprising a base model and two super-resolution models.

Cascaded Diffusion Models for High Fidelity Image Generation: Jonathan Ho,, Chitwan Sahari*, William Chan, David J. Fleet, Mohammad Norouzi, Tim Salimans

Latent diffusion



High-Resolution Image Synthesis with Latent Diffusion Models: Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, Björn Ommer

Score based methods

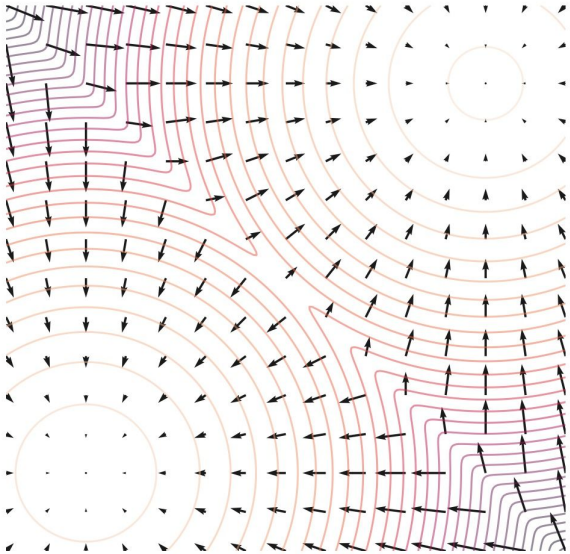
Score-based models

- **Score function**: $\nabla_x \log p(x)$

  -> the direction in which the probability density increases the fastest

- **Goal**: learn an **approximation** of the score function for **various noise levels**
  -> $s_\theta(x) = \nabla_x \log p_\theta(x)$

- **Sampling**: start with a random noise and follow the score until the data distribution



Score-Based Generative Modeling through Stochastic Differential Equations: Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, Ben Poole

Score-based models

- **Fisher divergence**: $\mathbb{E}_{p(x)}[\|\nabla_x \log p(x) - s_\theta(x)\|_2^2]$

  -> we don't know $\nabla_x \log p(x)$

- **Denoising score matching methods** that minimize the Fisher divergence without knowledge of the ground-truth data score

  -> $\mathbb{E}_{p(x),\epsilon}[\|s_\theta(\tilde{x}, \sigma) - \nabla_x \log p(\tilde{x}|x)\|^2]$

  where $\tilde{x} = x + \epsilon \, (\text{with } \epsilon \sim \mathcal{N}(0, \sigma^2))$

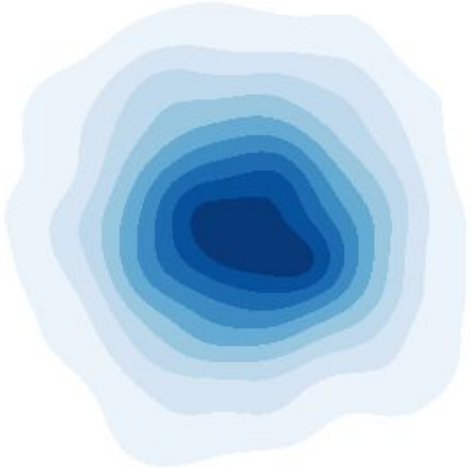
Score-Based Generative Modeling through Stochastic Differential Equations: Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, Ben Poole

Generative Modeling with Scores

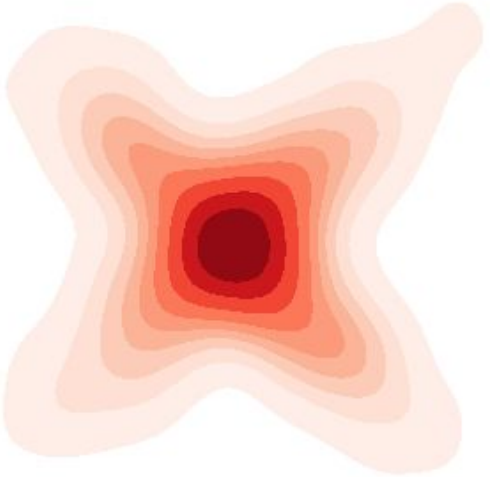
Sampling by greedy application of the score

x_t

$p_t(x)$



$q_{data}(x)$



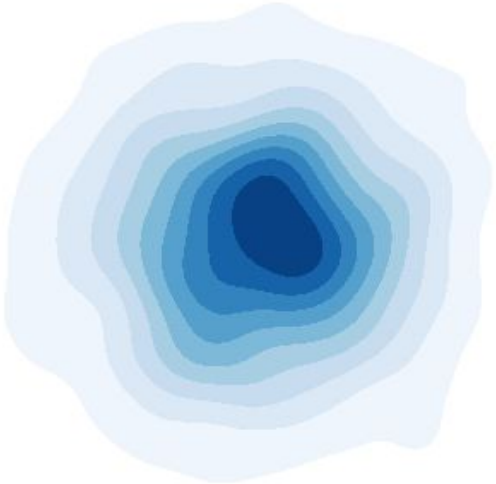
source: Ayan Das' ICLR blogpost

Langevin dynamics

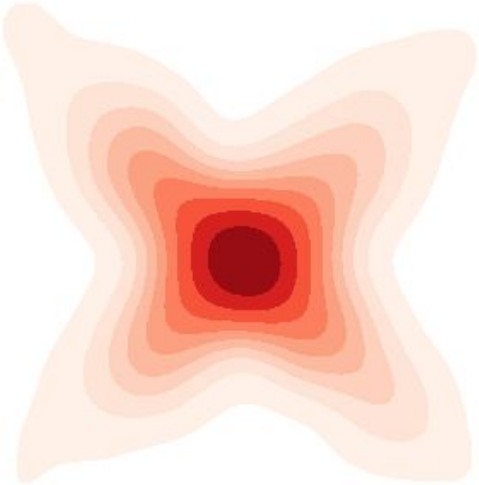
$$x_{t+1} \leftarrow x_t + \epsilon \nabla_x \log p(x) + \sqrt{2\epsilon}\, z_t \qquad \text{with} \quad z_t \sim \mathcal{N}(0, I)$$

x_t

$p_t(x)$



$q_{data}(x)$



source: Ayan Das' ICLR blogpost

Langevin dynamics proof (Ayan Das' ICLR blogpost)

Fokker-Planck Equation

The simulation is convincing; but it'd be even better if we can *theoretically verify* that the process in Eq.(6) indeed converges to $q_{data}(x)$. The key to this proof is figuring out $p_t(x)$ and making sure that it stabilizes as $t \to \infty$, i.e. $p_\infty(x) = q_{data}(x)$. It turned out that a stochastic process of the form $dx = \mu_t(x)dt + \sigma_t(x)dB_t$, acting on a random variable x, induces a time-varying distribution that can be described by this ODE

$$\frac{\partial}{\partial t}p_t(x) = -\frac{\partial}{\partial x}\Big[p_t(x)\mu_t(x)\Big] + \frac{1}{2}\frac{\partial^2}{\partial x^2}\Big[p_t(x)\sigma_t^2(x)\Big] \tag{7}$$

This is a well celebrated result know as the "Fokker-Planck equation" that even predates the Langevin Equation. So, the solution of this ODE is exactly what we are seeing in the above figure (middle). One can easily verify the convergence of Eq.(6) by first observing $\mu_t(x) = \nabla_x \log q_{data}(x), \sigma_t(x) = \sqrt{2}$ and then using $\frac{\partial}{\partial t}p_\infty(x) = \frac{\partial}{\partial t}q_{data}(x) = 0$.

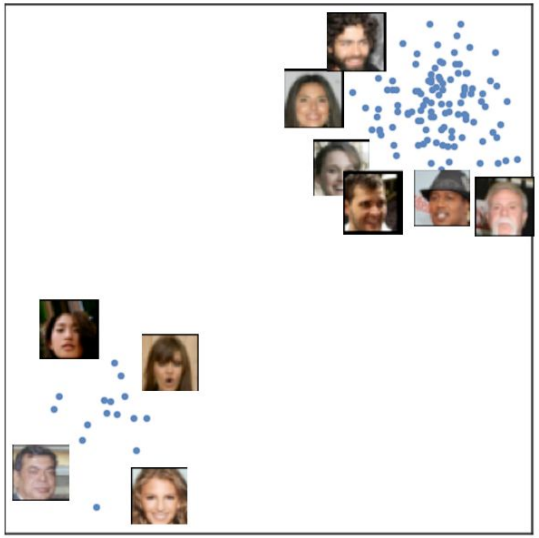
$$\begin{aligned}
\frac{\partial}{\partial t}p_\infty(x) &= -\frac{\partial}{\partial x}\Big[p_\infty(x)\nabla_x \log q_{data}(x)\Big] + \frac{(\sqrt{2})^2}{2}\frac{\partial^2}{\partial x^2}\Big[p_\infty(x)\Big] \\
\frac{\partial}{\partial t}q_{data}(x) &= -\frac{\partial}{\partial x}\Big[q_{data}(x)\nabla_x \log q_{data}(x)\Big] + \frac{(\sqrt{2})^2}{2}\frac{\partial^2}{\partial x^2}\Big[q_{data}(x)\Big] \\
0\ (\text{LHS}) &= -\frac{\partial}{\partial x}\Big[\nabla_x q_{data}(x)\Big] + \frac{\partial}{\partial x}\Big[\nabla_x q_{data}(x)\Big] = 0\ (\text{RHS})
\end{aligned}$$

The LHS holds due to the fact that after a long time (i.e. $t = \infty$) the distribution stabilizes [5]. Please also note that the proof above is for the 1 dimensional case and included for illustrative purpose only – the general case is slightly more complicated.

So, we're all good. Eq.(6) is a provable way of sampling given we have access to the true score. In fact, the very work [8] (by Song et al.) that immediately precedes DDPM, used exactly Eq.(6) in its discrete form

$$x_{t+\delta} = x_t + \delta \cdot \nabla_x \log q_{data}(x) + \sqrt{2\delta} \cdot z \tag{8}$$

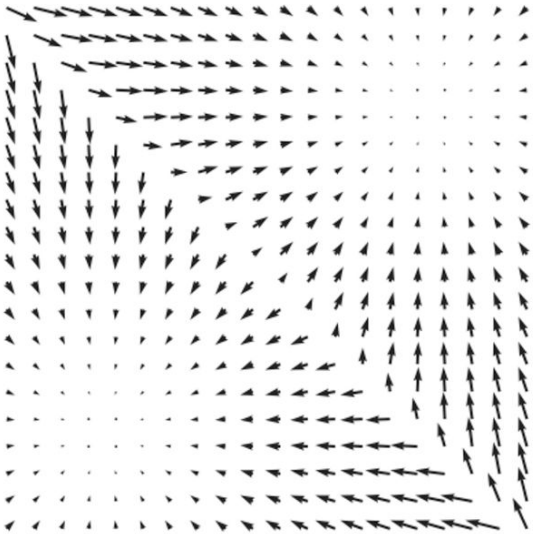
Generative Modeling with Scores



Data samples

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

score matching



Scores

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

Langevin dynamics

New samples

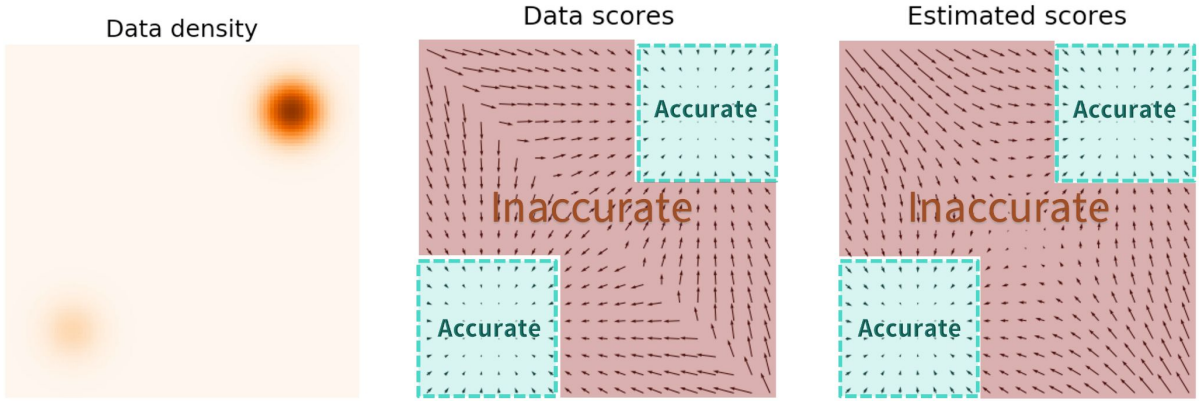
Score-Based Generative Modeling through Stochastic Differential Equations: Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, Ben Poole

Generative Modeling with Scores

- Estimated score functions are inaccurate in low density regions

$$\mathbb{E}_{p(x)}[\|\nabla_x \log p(x) - s_\theta(x)\|_2^2] = \int p(x)\|\nabla_x \log p(x) - s_\theta(x)\|_2^2 dx.$$

-> since MSE is weighted by $p(x)$ differences are ignored in low density regions



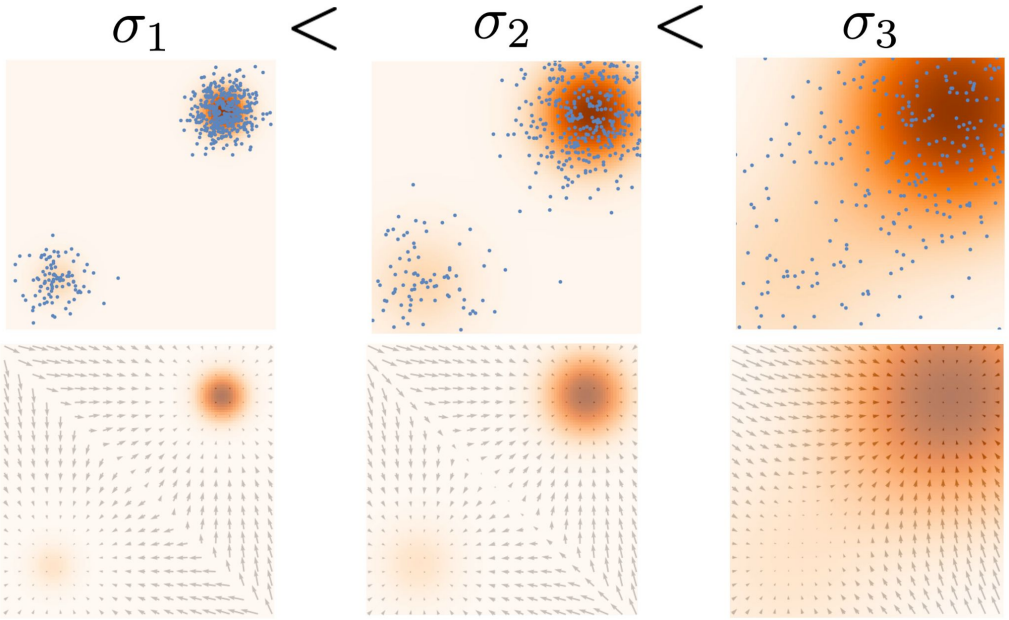
Score-Based Generative Modeling through Stochastic Differential Equations: Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, Ben Poole

Generative Modeling with Scores

Solution: use multiple noise scale

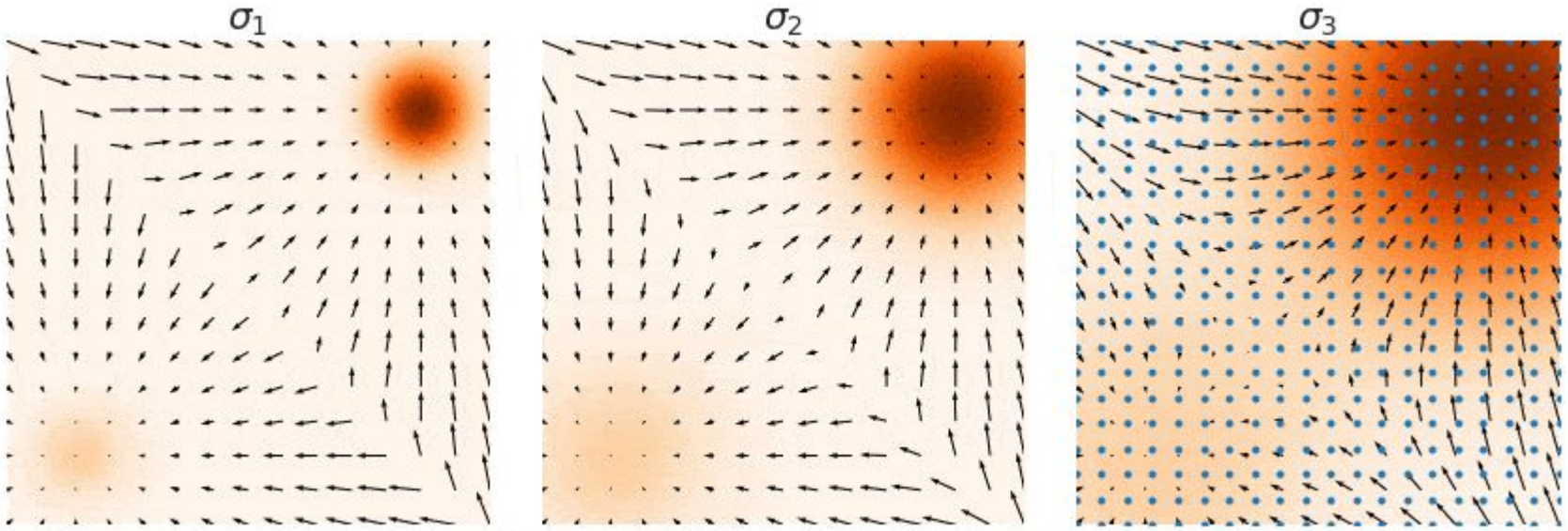
$$\sum_{i=1}^{L} \lambda(i) \mathbb{E}_{p_{\sigma_i}(x)}[\|\nabla_x \log p_{\sigma_i}(x) - s_\theta(x, i)\|_2^2]$$

with: $\lambda(i) = \sigma_i^2$



Score-Based Generative Modeling through Stochastic Differential Equations: Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, Ben Poole

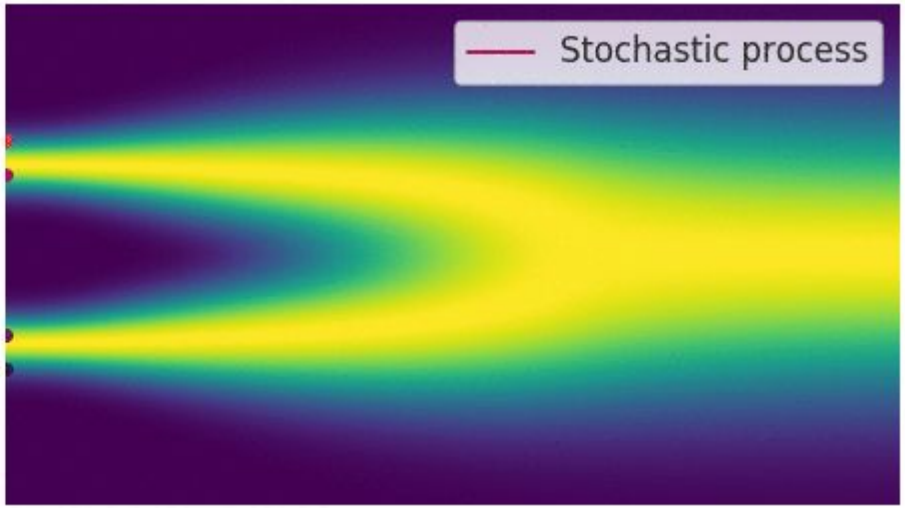
Sampling with annealed Langevin dynamics



Score-Based Generative Modeling through Stochastic Differential Equations: Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, Ben Poole

Stochastic differential equations (SDE)

- For infinite noise scale -> the noise perturbation procedure is a continuous-time stochastic process



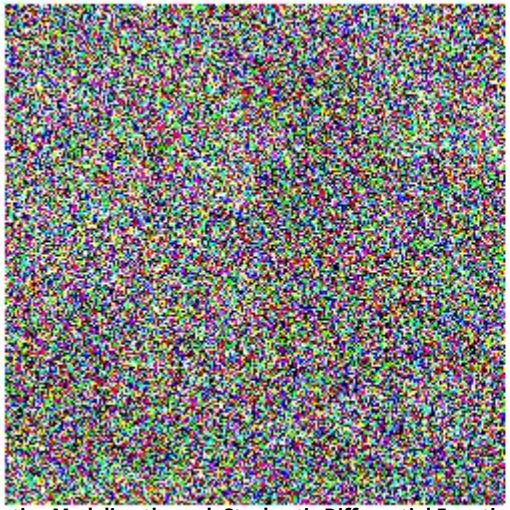
Score-Based Generative Modeling through Stochastic Differential Equations: Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, Ben Poole

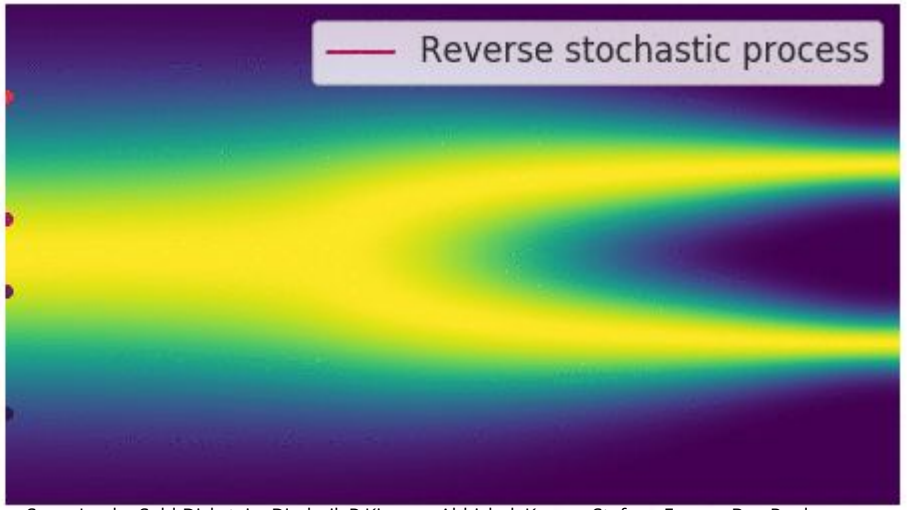
Reverse SDE

- Any SDE has a corresponding reverse SDE whose closed form is given by

$$dx = [f(x,t) - g^2(t)\nabla_x \log p_t(x)]dt + g(t)dw$$

drift diffusion coefficient brownian motion



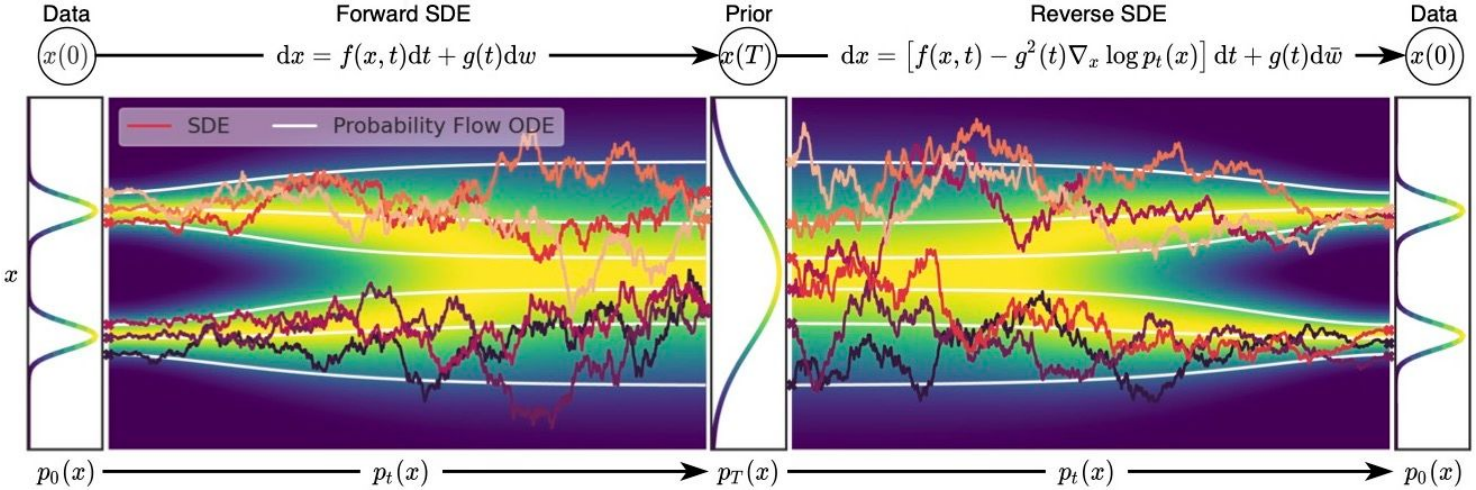


Score-Based Generative Modeling through Stochastic Differential Equations: Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, Ben Poole

Reverse SDE

Score-Based Generative Modeling through Stochastic Differential Equations [link]
Y. Song, J. Sohl-Dickstein, D.P. Kingma, A. Kumar, S. Ermon, B. Poole.
International Conference on Learning Representations. 2021.

- It is possible to convert any SDE into an ordinary differential equation (ODE) without changing its marginal distributions
  - Deterministic sampling
  - Better sample quality
  - Faster sampling



Score-Based Generative Modeling through Stochastic Differential Equations: Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, Ben Poole

Guidance

Guidance

Sometimes conditioning is just not enough to generate samples that really respect conditioning.

Or sometimes we train without conditioning but would want to condition at sampling.

Classifier Guidance

Score based perspective, we aim to learn

$$\nabla_{x_t} \log p_\theta(x_t|y) = \nabla_{x_t} \log\left(\frac{p_\theta(y|x_t)p_\theta(x_t)}{p_\theta(y)}\right) \quad \text{(Bayes rule)}$$

$$= \nabla_{x_t} \log p_\theta(x_t) + \nabla_{x_t} \log(p_\theta(y|x_t))$$

Train a classifier $f_\phi(y|x_t, t)$

Use its gradients at sampling:

$$\nabla \log p_\theta(x_t|y) = \nabla \log p_\theta(x_t) + \gamma \cdot \nabla \log(p_\theta(y|x_t))$$

Classifier guidance

Algorithm 1 Classifier guided diffusion sampling, given a diffusion model $(\mu_\theta(x_t), \Sigma_\theta(x_t))$, classifier $p_\phi(y|x_t)$, and gradient scale s.

Input: class label y, gradient scale s
$x_T \leftarrow$ sample from $\mathcal{N}(0, \mathbf{I})$
for all t from T to 1 **do**
  $\mu, \Sigma \leftarrow \mu_\theta(x_t), \Sigma_\theta(x_t)$
  $x_{t-1} \leftarrow$ sample from $\mathcal{N}(\mu + s\Sigma \nabla_{x_t} \log p_\phi(y|x_t), \Sigma)$
end for
return x_0

Diffusion Models Beat GANs on Image Synthesis: Prafulla Dhariwal, Alex Nichol

Classifier guidance

Figure 3: Samples from an unconditional diffusion model with classifier guidance to condition on the class "Pembroke Welsh corgi". Using classifier scale 1.0 (left; FID: 33.0) does not produce convincing samples in this class, whereas classifier scale 10.0 (right; FID: 12.0) produces much more class-consistent images.

Diffusion Models Beat GANs on Image Synthesis: Prafulla Dhariwal, Alex Nichol

GLIDE

Similar to classifier guidance but with text conditioning:

- Fine tune a CLIP model on noisy images
- Compute the prompt CLIP embeddings
- Compute the noisy images CLIP embeddings
- Use the gradient of the dot product between the two embeddings for guidance

GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models: Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, Mark Chen

Classifier guidance

- Allows to condition generation even if the model was not trainedwith conditioning.
- Helps steering generation for a model towards conditioning.

- Often requires training an auxiliary network on noisy inputs
- In for latent diffusion models may require decoding the latents
- Gradients of the conditioner may yield arbitrary or adversarial directions

Classifier free guidance

Bayes rule: $$p(y|x) = \frac{p(x|y) \cdot p(y)}{p(x)}$$

$$\log p(y|x) = \log p(x|y) + \log p(y) - \log p(x)$$

$$\nabla_x \log p(y|x) = \nabla_x \log p(x|y) - \nabla_x \log p(x).$$

Replace in the classifier guidance:

$$\begin{aligned}\nabla_x \log p_\gamma(x|y) &= \nabla_x \log p(x) + \gamma(\nabla_x \log p(x|y) - \nabla_x \log p(x) \\ &= (1-\gamma)\nabla_x \log p(x) + \gamma \nabla_x \log p(x|y).\end{aligned}$$

Classifier-Free Diffusion Guidance: Jonathan Ho, Tim Salimans

Classifier free guidance

- Train a conditioned diffusion model
- Drop conditioning 10-20% of the time (replace with special input)
- Model can now generate with and without conditioning
- Use CFG at sampling

Without CFG With CFG

Classifier-Free Diffusion Guidance: Jonathan Ho, Tim Salimans

Fine-tuning

Continual learning

We would like to add new concepts to our model.

-> Fine tune our general/foundation

OK if we have lots of examples of this concept and an important dataset with other concepts

If not, it leads to overfitting and catastrophic forgetting

Textual inversion

- Few samples with the new concept to learn.
- Try avoiding catastrophic forgetting
- Being able to make compositions with concepts previously learnt

Figure 1: (left) We find new pseudo-words in the embedding space of a pre-trained text-to-image model which describe specific concepts. (right) These pseudo-words can be composed into new sentences, placing our targets in new scenes, changing their style or composition, or ingraining them into new products.

An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion: Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, Daniel Cohen-Or

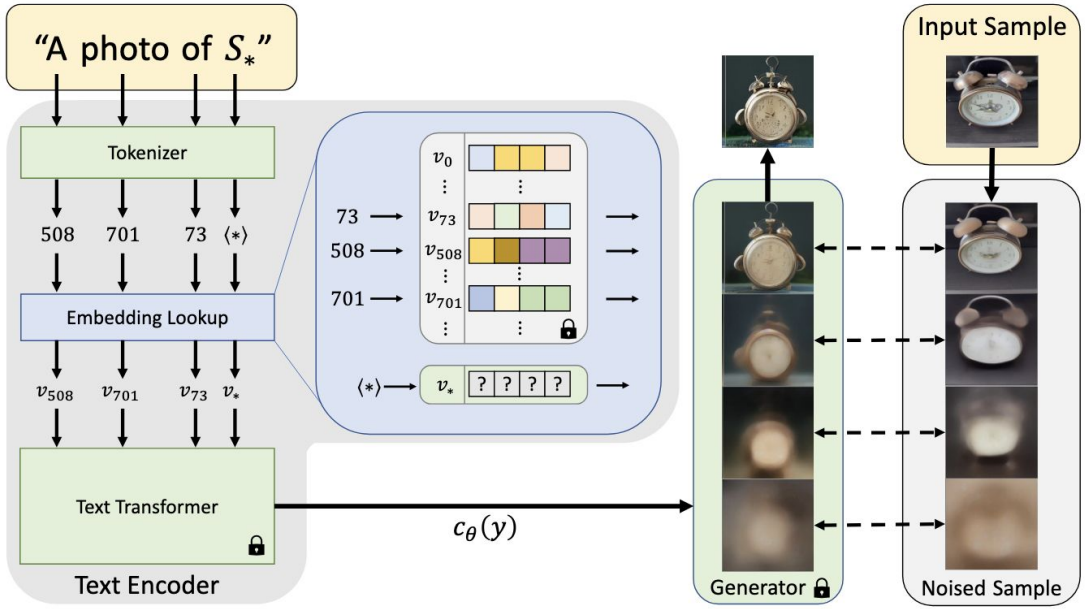
Textual inversion

Principle:

- Avoid catastrophic forgetting by freezing all networks
- Create a new token and it’s corresponding embeddings to represent the new concept
- Using the concept samples, optimize the embeddings such that when provided to the denoiser, generated images are close to the original samples.

An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion: Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, Daniel Cohen-Or

Textual inversion



Training objective:

$$v_* = \arg\min_v \mathbb{E}_{z\sim\mathcal{E}(x),y,\epsilon\sim\mathcal{N}(0,1),t}\left[\|\epsilon - \epsilon_\theta(z_t, t, c_\theta(y))\|_2^2\right]$$

An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion: Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, Daniel Cohen-Or

Textual inversion

Image variations:

An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion: Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, Daniel Cohen-Or

Textual inversion

Compositions with previous concepts:

An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion: Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, Daniel Cohen-Or

Textual inversion

Style transfer:

An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion: Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, Daniel Cohen-Or

Textual inversion

New concept composition:

An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion: Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, Daniel Cohen-Or

Textual inversion

Debiasing concept:

"A stock photo of a doctor" (Base model)

"A photo of S_*" (Ours)

An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion: Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, Daniel Cohen-Or

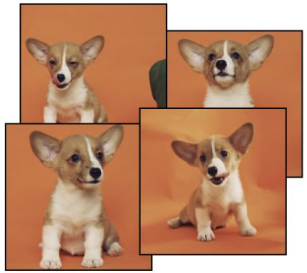
Dreambooth

Principle:

- Choose a token to describe the new concept
- Fine-tune the denoiser
- Regularize with a *“class-specific prior preservation loss”*

DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation: Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, Kfir Aberman

Dreambooth



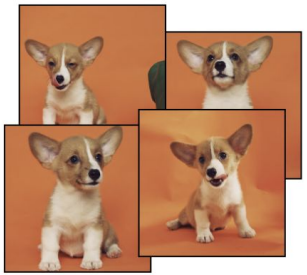
Choosing a token:

- Existing corresponding token (e.g. “dog”)
  => difficult to disentangle the original meaning of a token and reantangle it to reference the new subject
- Random sequences (e.g. “husiusl”) will be separated by the tokenizer in several tokens with a strong prior (e.g. [h, usi, us, l])
- Better to use very rare tokens

Dreambooth

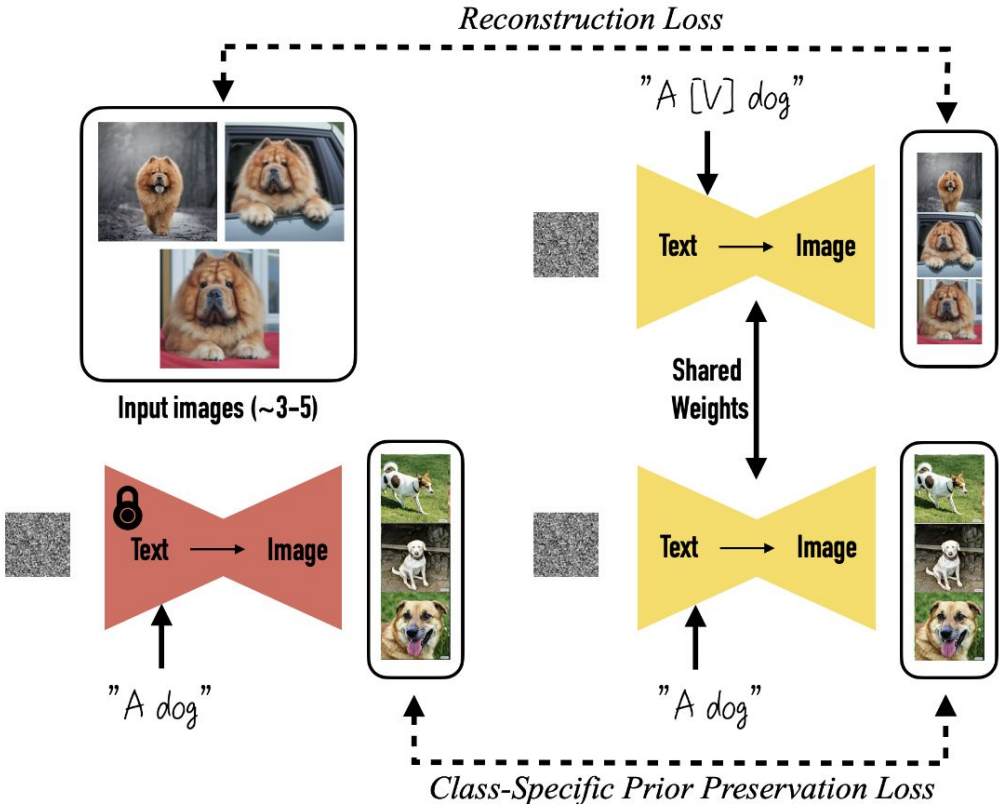
Prevent language drift:

- Fine-tune with text prompts containing a unique identifier followed by the class name of the subject (i.e. "A [V] dog")
  => class name let the model use prior knowledge on the subject class.
- Regularize to avoid the model to associate the class name witl specific concept/instance
  => ***class-specific prior preservation loss***



DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation: Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, Kfir Aberman

Dreambooth



DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation: Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, Kfir Aberman

Dreambooth

Some issues with Dreambooth-like methods:

- Methods like Dreambooth “overfit” a given concept
- Hurts your model for generating diverse images with other concepts
- Need to save several full copies of the denoiser for each new concept
- Fine-tuning the denoiser requires lot of VRAM

DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation: Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, Kfir Aberman

LOw Rank Adaptation (LORA)

Intuition:

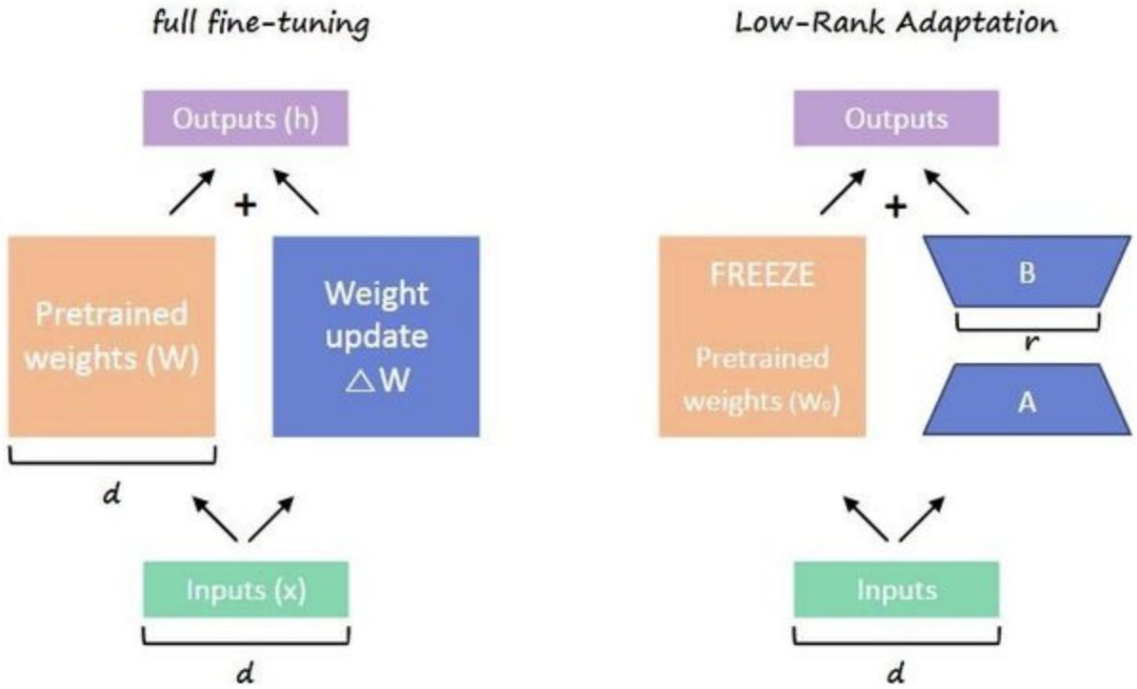
- Over-parameterized model actually reside in low intrinsic dimension
- Changes in weights during the model adaptation has low intrinsic rank.

Principle:

- Fine-tuning indirectly by optimizing rank decomposition matrices of the layer changes while freezing the original weights.

LoRA: Low-Rank Adaptation of Large Language Models: Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen

LOw Rank Adaptation (LORA)



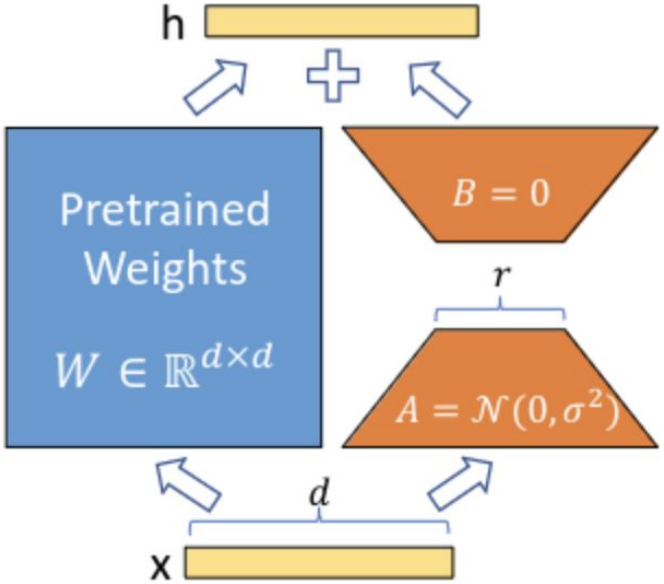
source: https://tech.hancom.com/2024-05-24-llm/

LoRA: Low-Rank Adaptation of Large Language Models: Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen

LOw Rank Adaptation (LORA)

Training tricks

- A is randomly initialized with small values drawn from a Gaussian and B is initialized with 0s to make sure that the first model inference is not perturbed by LoRA
- Provide good results when applied on Q and K matrices only



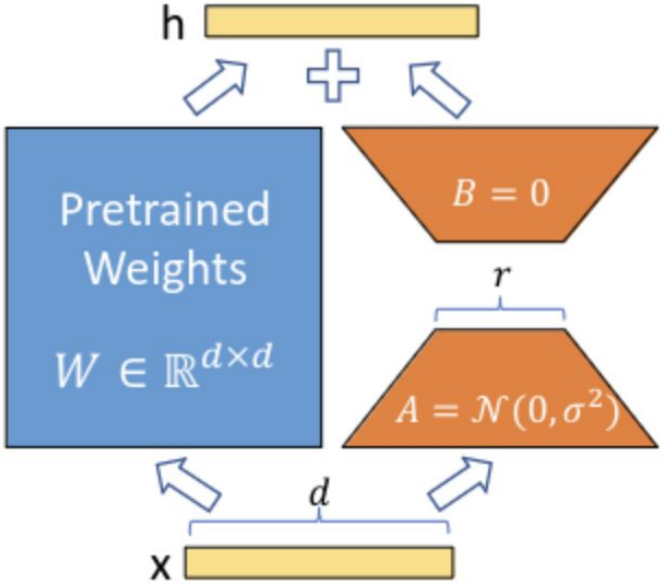
LoRA: Low-Rank Adaptation of Large Language Models: Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen

LOw Rank Adaptation (LORA)

Benefits:

- Almost as efficient as regular fine-tuning
- Lower VRAM impact
- Impact of the LORA weights can be tuned at inference
- Very small space on disk
- Several LORAs can be combined.

Examples



LoRA: Low-Rank Adaptation of Large Language Models: Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen

Faster sampling

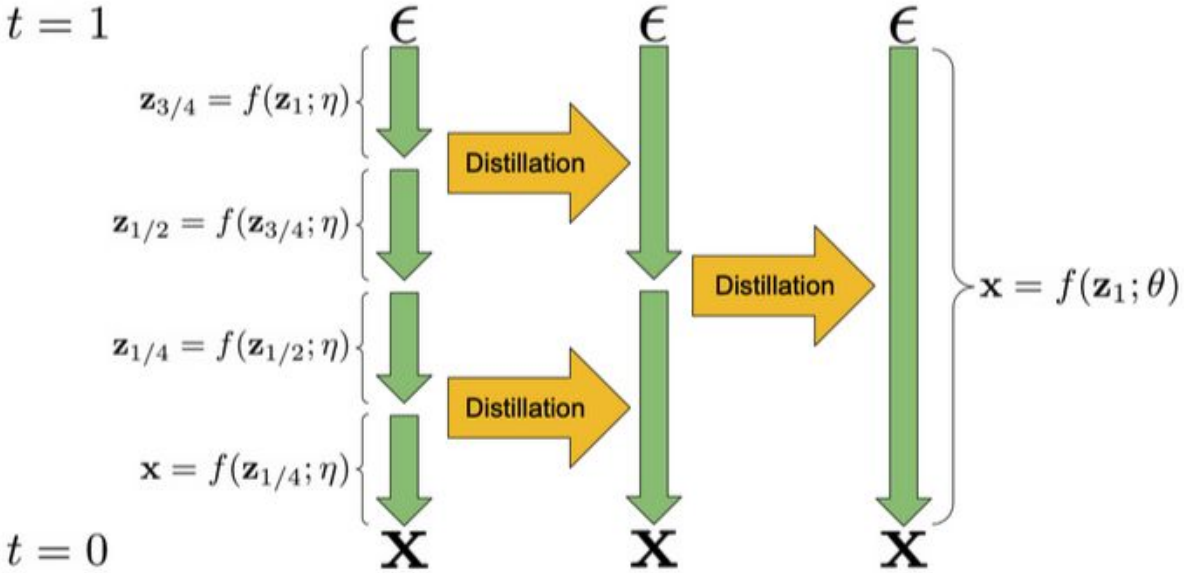
Denoising Diffusion Implicit Models (DDIM)

- **Implicit sampling:** $x_t = \sqrt{\alpha_t} x_0 + \sqrt{1 - \alpha_t}\epsilon$
- Predict x_0 from x_t -> $\hat{x}_0 = \frac{x_t - \sqrt{1-\alpha_t}\epsilon_\theta(x_t, t)}{\sqrt{\alpha_t}}$
- **DDIM sampling step**:

$$
\begin{aligned}
x_s &= \sqrt{\alpha_s}\hat{x}_0 + \sqrt{1 - \alpha_s}\epsilon_\theta(x_t, t) \\
&= \sqrt{\alpha_s}\left(\frac{x_t - \sqrt{1 - \alpha_t}\epsilon_\theta(x_t, t)}{\sqrt{\alpha_t}}\right) + \sqrt{1 - \alpha_s}\epsilon_\theta(x_t, t).
\end{aligned}
$$

Denoising Diffusion Implicit Models: Jiaming Song, Chenlin Meng, Stefano Ermon

Progressive distillation



Progressive Distillation for Fast Sampling of Diffusion Models: Tim Salimans, Jonathan Ho

Progressive distillation

Algorithm 1 Standard diffusion training

Require: Model $\hat{\mathbf{x}}_\theta(\mathbf{z}_t)$ to be trained
Require: Data set $\mathcal{D}$
Require: Loss weight function $w()$

while not converged **do**
- $\mathbf{x} \sim \mathcal{D}$ ▷ Sample data
- $t \sim U[0, 1]$ ▷ Sample time
- $\epsilon \sim N(0, I)$ ▷ Sample noise
- $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$ ▷ Add noise to data
- $\tilde{\mathbf{x}} = \mathbf{x}$ ▷ Clean data is target for $\hat{\mathbf{x}}$
- $\lambda_t = \log[\alpha_t^2 / \sigma_t^2]$ ▷ log-SNR
- $L_\theta = w(\lambda_t) \|\tilde{\mathbf{x}} - \hat{\mathbf{x}}_\theta(\mathbf{z}_t)\|_2^2$ ▷ Loss
- $\theta \leftarrow \theta - \gamma \nabla_\theta L_\theta$ ▷ Optimization

end while

Algorithm 2 Progressive distillation

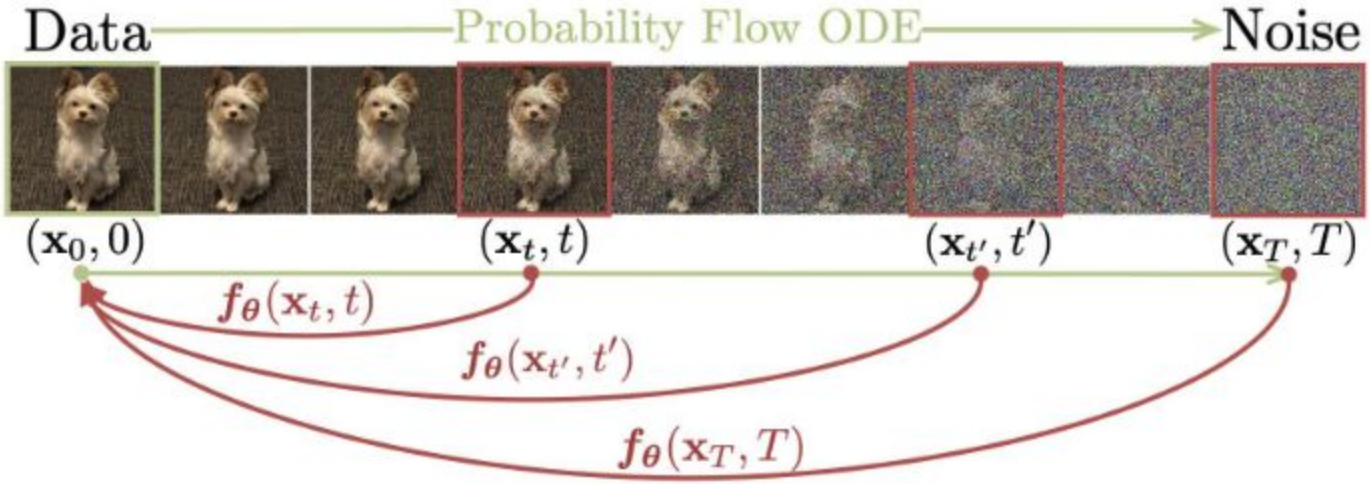
Require: Trained teacher model $\hat{\mathbf{x}}_\eta(\mathbf{z}_t)$
Require: Data set $\mathcal{D}$
Require: Loss weight function $w()$
Require: Student sampling steps N

for K iterations **do**
- $\theta \leftarrow \eta$ ▷ Init student from teacher
- **while** not converged **do**
  - $\mathbf{x} \sim \mathcal{D}$
  - $t = i/N, \;\; i \sim Cat[1, 2, \ldots, N]$
  - $\epsilon \sim N(0, I)$
  - $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$
  - `# 2 steps of DDIM with teacher`
  - $t' = t - 0.5/N, \;\; t'' = t - 1/N$
  - $\mathbf{z}_{t'} = \alpha_{t'} \hat{\mathbf{x}}_\eta(\mathbf{z}_t) + \frac{\sigma_{t'}}{\sigma_t}(\mathbf{z}_t - \alpha_t \hat{\mathbf{x}}_\eta(\mathbf{z}_t))$
  - $\mathbf{z}_{t''} = \alpha_{t''} \hat{\mathbf{x}}_\eta(\mathbf{z}_{t'}) + \frac{\sigma_{t''}}{\sigma_{t'}}(\mathbf{z}_{t'} - \alpha_{t'} \hat{\mathbf{x}}_\eta(\mathbf{z}_{t'}))$
  - $\tilde{\mathbf{x}} = \frac{\mathbf{z}_{t''} - (\sigma_{t''}/\sigma_t)\mathbf{z}_t}{\alpha_{t''} - (\sigma_{t''}/\sigma_t)\alpha_t}$ ▷ Teacher $\hat{\mathbf{x}}$ target
  - $\lambda_t = \log[\alpha_t^2 / \sigma_t^2]$
  - $L_\theta = w(\lambda_t) \|\tilde{\mathbf{x}} - \hat{\mathbf{x}}_\theta(\mathbf{z}_t)\|_2^2$
  - $\theta \leftarrow \theta - \gamma \nabla_\theta L_\theta$
- **end while**
- $\eta \leftarrow \theta$ ▷ Student becomes next teacher
- $N \leftarrow N/2$ ▷ Halve number of sampling steps

end for

Progressive Distillation for Fast Sampling of Diffusion Models: Tim Salimans, Jonathan Ho

Consistency models



Consistency Models: Yang Song, Prafulla Dhariwal, Mark Chen, Ilya Sutskever

Consistency models

Algorithm 2 Consistency Distillation (CD)

Input: dataset $\mathcal{D}$, initial model parameter $\boldsymbol{\theta}$, learning rate η, ODE solver $\Phi(\cdot, \cdot; \boldsymbol{\phi})$, $d(\cdot, \cdot)$, $\lambda(\cdot)$, and μ

$\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$

repeat

Sample $\mathbf{x} \sim \mathcal{D}$ and $n \sim \mathcal{U}[\![1, N-1]\!]$

Sample $\mathbf{x}_{t_{n+1}} \sim \mathcal{N}(\mathbf{x}; t_{n+1}^2 \boldsymbol{I})$

$\hat{\mathbf{x}}_{t_n}^{\boldsymbol{\phi}} \leftarrow \mathbf{x}_{t_{n+1}} + (t_n - t_{n+1}) \Phi(\mathbf{x}_{t_{n+1}}, t_{n+1}; \boldsymbol{\phi})$

$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^-; \boldsymbol{\phi}) \leftarrow$

$\lambda(t_n) d(\boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x}_{t_{n+1}}, t_{n+1}), \boldsymbol{f}_{\boldsymbol{\theta}^-}(\hat{\mathbf{x}}_{t_n}^{\boldsymbol{\phi}}, t_n))$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^-; \boldsymbol{\phi})$

$\boldsymbol{\theta}^- \leftarrow \operatorname{stopgrad}(\mu \boldsymbol{\theta}^- + (1 - \mu) \boldsymbol{\theta})$

until convergence

Consistency Models: Yang Song, Prafulla Dhariwal, Mark Chen, Ilya Sutskever