

CS-E4890: Deep Learning

Denoising-based generative modeling

Alexander Ilin

Representation learning by denoising

- Recall denoising autoencoders (DAE). We feed inputs corrupted with noise

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

and train a network $\mathbf{d}(\tilde{\mathbf{x}})$ to produce clean data \mathbf{x} by minimizing $\mathcal{L} = E [\|\mathbf{d}(\tilde{\mathbf{x}}) - \mathbf{x}\|^2]$.

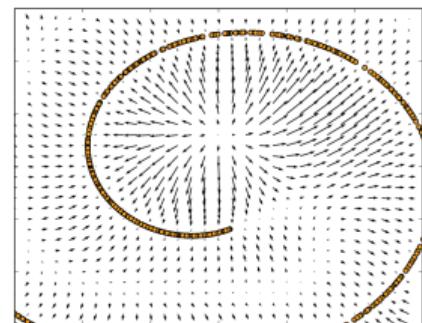
- DAE assignment: the task of denoising can encourage learning of useful representations.

- Recall that for Gaussian corruption $\epsilon \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$, the optimal denoising is given by

$$d(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}} + \sigma^2 \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}})$$

where $q_\sigma(\tilde{\mathbf{x}})$ is the perturbed data distribution.

- $\nabla_{\tilde{\mathbf{x}}} \log p(\mathbf{x})$ is often called a *score function*.



The optimal denoising function points towards areas with higher probability density ([Alain and Bengio, 2014](#))

$$\text{Optimal denoising: } d(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}} + \sigma^2 \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}})$$

- When the noise is small, $q_\sigma(\mathbf{x}) \approx p(\mathbf{x})$ and the clean data can be reconstructed almost perfectly $d(\tilde{\mathbf{x}}) \approx \mathbf{x}$, which yields

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x}) = \frac{d(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}}{\sigma^2} \approx \frac{\mathbf{x} - \tilde{\mathbf{x}}}{\sigma^2}.$$

- Thus, the score function can be estimated by training a neural network $s_\theta(\tilde{\mathbf{x}}, \sigma)$ to minimize

$$\mathcal{L}_\sigma = \frac{1}{2} E_{p_{\text{data}}(\mathbf{x})} E_{\tilde{\mathbf{x}} \sim N(\mathbf{x}, \sigma^2 I)} \left\| s_\theta(\tilde{\mathbf{x}}, \sigma) + \frac{\tilde{\mathbf{x}} - \mathbf{x}}{\sigma^2} \right\|^2. \quad (1)$$

- Training procedure: corrupt clean sample \mathbf{x} with Gaussian noise ϵ to get $\tilde{\mathbf{x}} = \mathbf{x} + \epsilon$ and train a neural network to predict the noise ϵ (scaled by $1/\sigma^2$) from $\tilde{\mathbf{x}}$.
- This type of modeling is often called *denoising score matching* (learning the score function by denoising).

Generative Modeling
via denoising score matching
(Song and Ermon, 2020)

Sampling with Langevin dynamics

- If we know the score function $\nabla_x \log p(x)$, we can sample from the corresponding distribution using Langevin dynamics, a sampling procedure which iterates the following:

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} + \alpha \nabla_{\mathbf{x}} \log p(\mathbf{x}_{t-1}) + \sqrt{2\alpha} \mathbf{z}_t, \quad 1 \leq t \leq T, \quad \mathbf{z}_t \sim N(0, \mathbf{I})$$

where $\alpha > 0$ is a step size and \mathbf{x}_0 is a sample from any prior distribution $\pi(\mathbf{x})$.

- When α is sufficiently small and T is sufficiently large, the distribution of \mathbf{x}_T will be close to $p(\mathbf{x})$ under some regularity conditions.
- Question: Why do we need to add noise \mathbf{z}_t ?
- If we have a neural network $s_{\theta}(\mathbf{x})$ which has been trained such that $s_{\theta}(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$, we can generate samples from $p(\mathbf{x})$ using $s_{\theta}(\mathbf{x}_{t-1})$ instead of $\nabla_{\mathbf{x}} \log p(\mathbf{x}_{t-1})$.

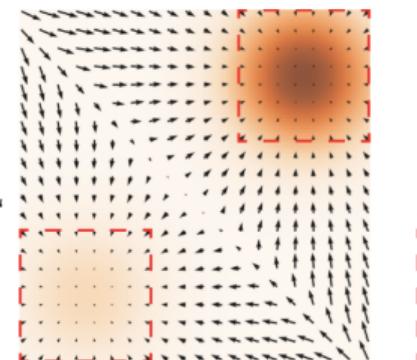
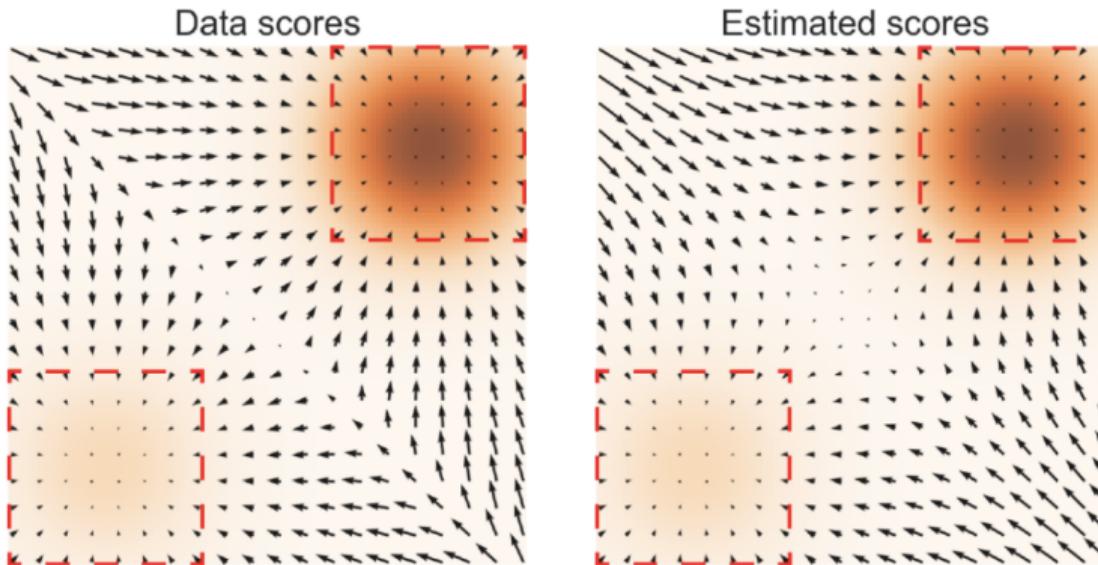


Image from [\(Song and Ermon, 2020\)](#)

Problem 1 with Langevin dynamics sampling

- Score function may be poorly estimated in regions of low data density (due to lack of data samples).



Darker color implies higher density. Red rectangles highlight regions where $\nabla_x \log p_{\text{data}}(x) \approx s_\theta(x)$.

Problem 2: Bad mixing of Langevin dynamics

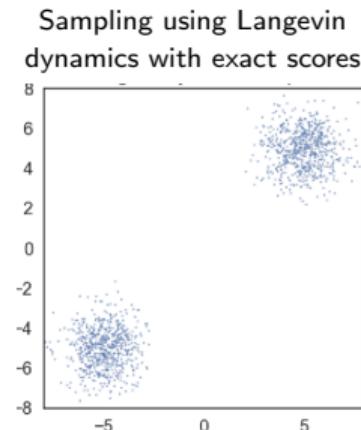
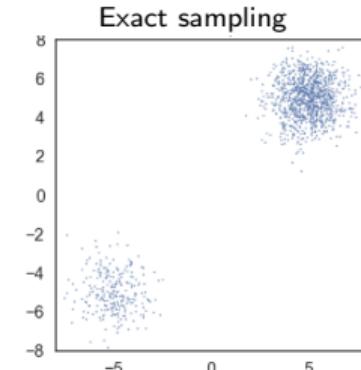
- Consider a mixture distribution $p_{\text{data}}(\mathbf{x}) = \pi p_1(\mathbf{x}) + (1 - \pi)p_2(\mathbf{x})$, where $p_1(\mathbf{x})$ and $p_2(\mathbf{x})$ are normalized distributions with disjoint supports, and $\pi \in (0, 1)$.
- In the support of $p_1(\mathbf{x})$:

$$\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) = \nabla_{\mathbf{x}}(\log \pi + \log p_1(\mathbf{x})) = \nabla_{\mathbf{x}} \log p_1(\mathbf{x})$$

- In the support of $p_2(\mathbf{x})$:

$$\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) = \nabla_{\mathbf{x}}(\log(1 - \pi) + \log p_2(\mathbf{x})) = \nabla_{\mathbf{x}} \log p_2(\mathbf{x})$$

- In either case, the score $\nabla_{\mathbf{x}} \log p_{\text{data}}$ does not depend on π .
- Langevin dynamics estimate the relative weights between the two modes incorrectly.



- Song and Ermon (2020) generate samples using Langevin dynamics with the score function learned from data.
- The problems of Langevin dynamics are addressed in the following way:
 1. Perturb the data using various levels of noise $\sigma_1 > \sigma_2 > \dots \sigma_L$ and estimate scores corresponding to all noise levels by training a single conditional score network:

$$s_\theta(\mathbf{x}, \sigma) \approx \nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x})$$

where $q_\sigma(\mathbf{x})$ is the perturbed data distribution. The loss is

$$\mathcal{L} = \frac{1}{L} \sum_{i=1}^L \lambda(\sigma_i) \mathcal{L}_i, \quad \mathcal{L}_i = \frac{1}{2} E_{p_{\text{data}}(\mathbf{x})} E_{\tilde{\mathbf{x}} \sim N(\mathbf{x}, \sigma_i^2 I)} \left[\left\| s_\theta(\tilde{\mathbf{x}}, \sigma_i) + \frac{\tilde{\mathbf{x}} - \mathbf{x}}{\sigma_i^2} \right\|^2 \right]$$

- \mathcal{L}_i is the denoising score matching objective for σ_i .
- coefficients $\lambda(\sigma) = \sigma$ (chosen empirically).

2. Generate samples using annealed Langevin dynamics.

- Initialize samples from some fixed prior distribution, e.g., uniform noise.
- Run Langevin dynamics to sample from $q_{\sigma_1}(\mathbf{x})$ with step size $\alpha_1 = \frac{\sigma_1}{\sigma_L}$.
- Run Langevin dynamics to sample from $q_{\sigma_2}(\mathbf{x})$, starting from the final samples of the previous simulation and using a reduced step size $\alpha_2 = \frac{\sigma_2}{\sigma_L}$.
- ...
- Finally, run Langevin dynamics to sample from $q_{\sigma_L}(\mathbf{x})$, which is close to $p_{\text{data}}(\mathbf{x})$ when $\sigma_L \approx 0$.

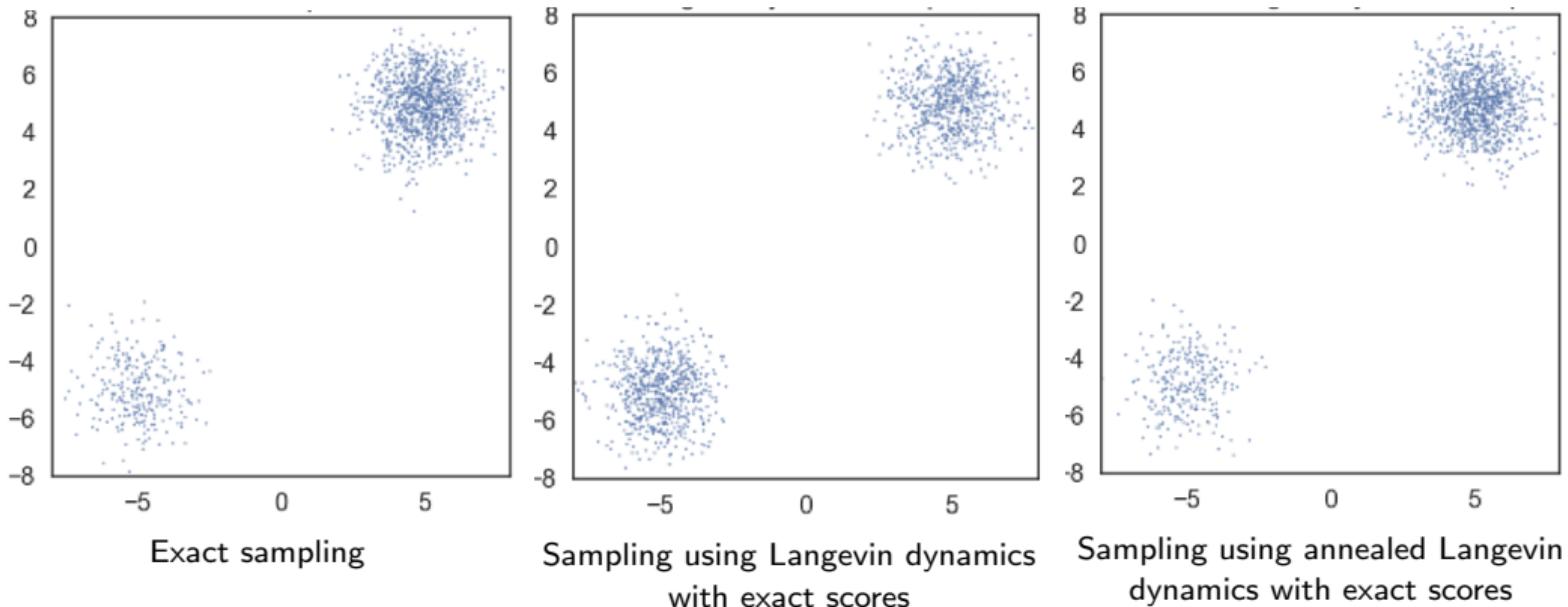
Algorithm 1 Annealed Langevin dynamics.

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

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

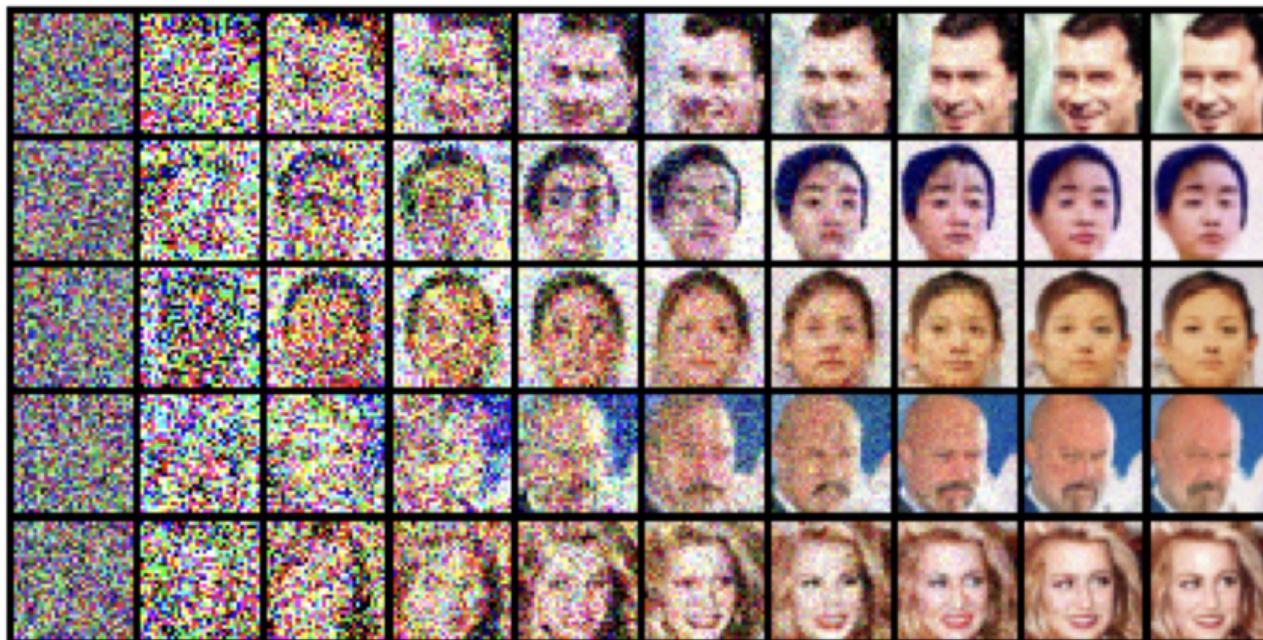
Annealed Langevin dynamics: Toy example

- Annealed Langevin dynamics recover the relative weights faithfully.



Generated samples with NCSN (Song and Ermon, 2020)

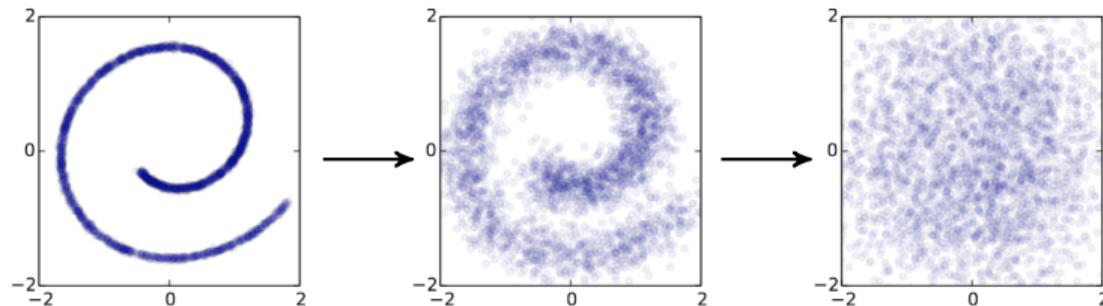
- When applied to images, the model $s_\theta(x, \sigma)$ is a U-Net with dilated convolution.
- They use a modified version of conditional instance normalization to provide conditioning on σ_i .



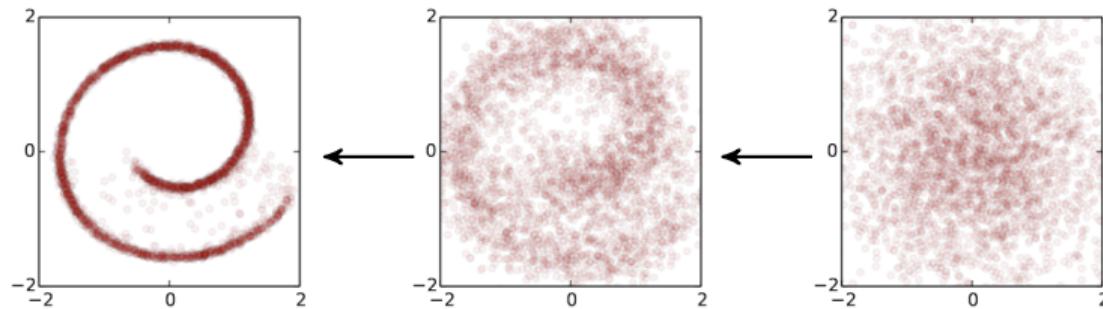
Diffusion probabilistic models
(Sohl-Dickstein et al., 2015)

Diffusion probabilistic models (Sohl-Dickstein et al., 2015)

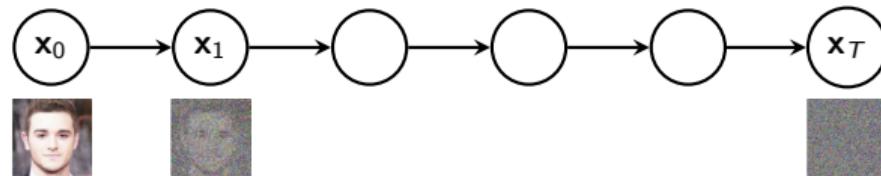
- Define a forward diffusion process which converts any complex data distribution into a simple, tractable, distribution.



- Learn the generative model which is defined by a reversal of this diffusion process



Forward diffusion process



- Most popular forward process: Given a sample from the data distribution $\mathbf{x}_0 \sim q(\mathbf{x}_0)$, produce chain $\mathbf{x}_1, \dots, \mathbf{x}_T$ by progressively adding Gaussian noise:

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = N(\mathbf{x}_t; \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I}), \quad \text{with small } \beta_t.$$

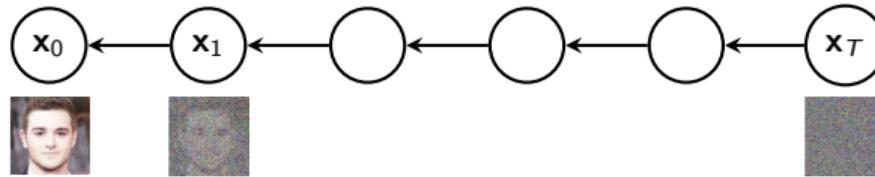
- $q(\mathbf{x}_t | \mathbf{x}_0)$ has a closed form:

$$q(\mathbf{x}_t | \mathbf{x}_0) = N(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I}), \quad \alpha_t = 1 - \beta_t, \quad \bar{\alpha}_t = \prod_{\tau=1}^t \alpha_\tau$$

and converges to a simple distribution: $q(\mathbf{x}_t | \mathbf{x}_0) \xrightarrow{t \rightarrow \infty} N(0, \mathbf{I})$.

- Selecting β_t such that $1 - \bar{\alpha}_t$ is close to 1, $q(\mathbf{x}_T)$ is well approximated by $N(0, \mathbf{I})$.

Reverse (generative) process



- Produce samples $\mathbf{x}_0 \sim p_\theta(\mathbf{x}_0)$ by starting with Gaussian noise $\mathbf{x}_T \sim N(0, \mathbf{I})$ and gradually reducing the noise in a sequence of steps $\mathbf{x}_{T-1}, \mathbf{x}_{T-2}, \dots, \mathbf{x}_0$.
- For computational convenience, we define the reverse process as

$$p(\mathbf{x}_{t-1} | \mathbf{x}_t) = N(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

and the task is to learn $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$, $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)$ to maximize the log-likelihood $E[\log p_\theta(\mathbf{x}_0)]$.

- The original paper ([Sohl-Dickstein et al., 2015](#)) proposes estimation of model parameters θ by minimizing the variational bound on negative log-likelihood:

$$E[-\log p_\theta(\mathbf{x}_0)] \leq E_q \left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right]$$

- This loss can be re-written (Appendix A, Ho et al. 2020) as

$$E_q \left[D_{\text{KL}}(q(\mathbf{x}_T \mid \mathbf{x}_0) \parallel p(\mathbf{x}_T)) + \sum_{t>1} D_{\text{KL}}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)) - \log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1) \right]$$

- Intuition: In the reverse process, the distribution $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ should be close to $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ which is obtained with the knowledge of the uncorrupted sample \mathbf{x}_0 .
- Due to the selected diffusion process, $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ has a closed form:

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

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

- The loss contains KL divergences between Gaussian distributions and therefore it can be computed analytically!

Denoising Diffusion Probabilistic Models (DDPM) (Ho et al., 2020)

- If we use fixed variances β_t and fixed diagonal covariance matrices in the reverse process

$$p(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = N(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I}),$$

where $\sigma_t^2 = \beta_t^2$ works well in practice, the loss is simplified:

$$\mathcal{L}_{t-1} = D_{\text{KL}}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)) = E_q \left[\frac{1}{2\sigma_t^2} \|\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)\|^2 \right] + C$$

- Thus, we only need to care about the means of the distributions in the reverse process!

DDPM: Simplifying the loss

Suppose that ϵ is the noise instance that was used to produce x_t from x_0 :

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon,$$

then

$$x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}} (x_t - \sqrt{1 - \bar{\alpha}_t} \epsilon)$$

and we get:

$$\begin{aligned}\hat{\mu}_t(x_t, x_0) &= \frac{\sqrt{\bar{\alpha}_{t-1}} \beta_t}{1 - \bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t \\ &= \frac{\sqrt{\bar{\alpha}_{t-1}} \beta_t}{1 - \bar{\alpha}_t} \frac{1}{\sqrt{\bar{\alpha}_t}} (x_t - \sqrt{1 - \bar{\alpha}_t} \epsilon) + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t \\ &= \frac{\beta_t}{1 - \bar{\alpha}_t} \frac{1}{\sqrt{\alpha_t}} (x_t - \sqrt{1 - \bar{\alpha}_t} \epsilon) + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t \quad \text{because } \frac{\sqrt{\bar{\alpha}_{t-1}}}{\sqrt{\bar{\alpha}_t}} = \frac{1}{\sqrt{\alpha_t}} \\ &= \frac{x_t}{(1 - \bar{\alpha}_t)\sqrt{\alpha_t}} (\beta_t + \alpha_t(1 - \bar{\alpha}_{t-1})) - \frac{\beta_t}{\sqrt{\alpha_t}\sqrt{1 - \bar{\alpha}_t}} \epsilon \\ &= \frac{x_t}{(1 - \bar{\alpha}_t)\sqrt{\alpha_t}} (1 - \bar{\alpha}_{t-1}) - \frac{\beta_t}{\sqrt{\alpha_t}\sqrt{1 - \bar{\alpha}_t}} \epsilon \quad \text{because } \beta_t = 1 - \alpha_t \\ &= \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon \right)\end{aligned}$$

- Since the target is expressed as

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

it is convenient to use a parameterization for the denoising model that has a similar form:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right),$$

- This parameterization leads to the following loss

$$\mathcal{L}_{t-1} = E_{\mathbf{x}_0, \epsilon} \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1 - \bar{\alpha}_t)} \|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2 \right]$$

DDPM: Training and sampling procedures

- The training procedure of DDPM:

1. Sample a mini-batch of samples $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
2. For each \mathbf{x}_0 , sample $t \sim \text{Uniform}(\{1, \dots, T\})$
3. Generate noise $\epsilon \sim N(0, I)$ and compute corrupted samples

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, \quad \text{with } \bar{\alpha}_t = \prod_{s=1}^t \alpha_s.$$

4. Compute the loss $\mathcal{L} = \|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2$
5. Compute the gradients and update the model parameters θ .

- Sampling procedure:

1. Sample $\mathbf{x}_T \sim N(0, I)$
2. Perform $T - 1$ steps: $\mathbf{x}_{t-1} \sim N\left(\frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t)\right), \sigma_t^2 \mathbf{I}\right)$
3. Compute generated sample $\mathbf{x}_0 = \frac{1}{\sqrt{\alpha_1}} \left(\mathbf{x}_1 - \frac{1 - \alpha_1}{\sqrt{1 - \bar{\alpha}_1}} \epsilon_\theta(\mathbf{x}_1, 1)\right)$

Connection to denoising score matching

- DDPM: Given a corrupted example \mathbf{x}_t with the noise level determined by t , the task is to find the noise instance ϵ that led to \mathbf{x}_t from \mathbf{x}_0 :

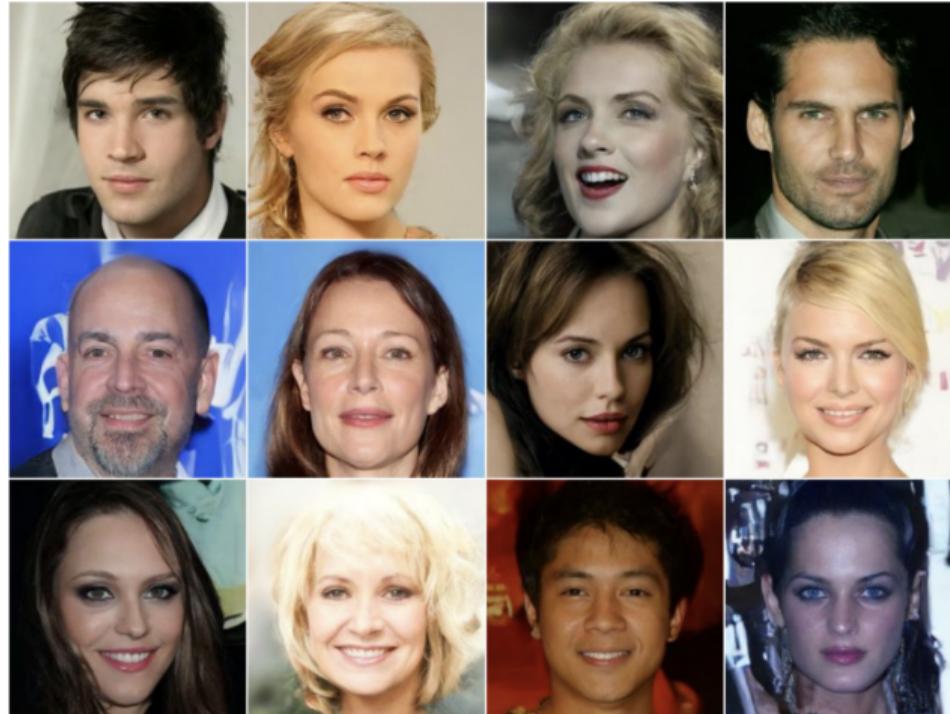
$$\mathcal{L}_{t-1} = E_{\mathbf{x}_0, \epsilon} \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1 - \bar{\alpha}_t)} \|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2 \right]$$

- Compare this to the loss used to train the Noise Conditional Score Networks ([Song and Ermon, 2020](#)) that we considered previously:

$$\mathcal{L}_i = \frac{1}{2} E_{p_{\text{data}}(\mathbf{x})} E_{\tilde{\mathbf{x}} \sim N(\mathbf{x}, \sigma_i^2 \mathbf{I})} \left[\left\| s_\theta(\tilde{\mathbf{x}}, \sigma_i) + \frac{\tilde{\mathbf{x}} - \mathbf{x}}{\sigma_i} \right\|^2 \right]$$

- The two approaches are very similar: the learning task is to denoise samples obtained with different levels of noise.

DDPM: Generated samples



Diffusion models beat GANs on image synthesis (Dhariwal and Nichol, 2020)

- Dhariwal and Nichol (2020) fine-tuned the architecture of DDPM and showed that diffusion models can outperform GANs.



Model	FID	sFID	Prec	Rec
ImageNet 128×128				
BigGAN-deep [5]	6.02	7.18	0.86	0.35
LOGAN [†] [68]	3.36			
ADM	5.91	5.09	0.70	0.65
ADM-G (25 steps)	5.98	7.04	0.78	0.51
ADM-G	2.97	5.09	0.78	0.59
ImageNet 256×256				
DCTransformer [†] [42]	36.51	8.24	0.36	0.67
VQ-VAE-2 ^{†‡} [51]	31.11	17.38	0.36	0.57
IDDPM [†] [43]	12.26	5.42	0.70	0.62
SR3 ^{†‡} [53]	11.30			
BigGAN-deep [5]	6.95	7.36	0.87	0.28
ADM	10.94	6.02	0.69	0.63
ADM-G (25 steps)	5.44	5.32	0.81	0.49
ADM-G	4.59	5.25	0.82	0.52

Denoising diffusion implicit model (DDIM)
(Song et al., 2021)

- A drawback of diffusion models is that they require many iterations to produce a high quality sample. The generative process of DDPM usually contains $T = 1000$ steps.
- For comparison, GANs only need one pass through the generator network.
- For example, it takes around 20 hours to sample 50k images of size 32×32 from a DDPM, but less than a minute to do so from a GAN on a Nvidia 2080 Ti GPU ([Song et al., 2021](#)).
- Sampling 50k images of size 256×256 from a DDPM could take nearly 1000 hours on the same GPU.

Denoising diffusion implicit model (DDIM) (Song et al., 2021)

- To speed up the generative process, Song et al. (2021) proposed to by using only a sub-sequence $\{\tau_i\}$ of steps $[1, \dots, T]$ in the reverse process *without changing the trained DDPM model*.
- One can re-write the DDPM generation steps in the following form:

$$\mathbf{x}_{t-1} = \underbrace{\sqrt{\alpha_{t-1}} \left(\frac{\mathbf{x}_t - \sqrt{1 - \alpha_t} \epsilon_\theta^{(t)}(\mathbf{x}_t)}{\sqrt{\alpha_t}} \right)}_{\text{"predicted } \mathbf{x}_0\text{"}} + \underbrace{\sqrt{1 - \alpha_{t-1} - \sigma_t^2} \cdot \epsilon_\theta^{(t)}(\mathbf{x}_t)}_{\text{"direction pointing to } \mathbf{x}_t\text{"}} + \underbrace{\sigma_t \epsilon_t}_{\text{random noise}}$$

where

$$\sigma_t = \sqrt{\frac{1 - \alpha_{t-1}}{1 - \alpha_t}} \sqrt{1 - \frac{\alpha_t}{\alpha_{t-1}}}, \quad \forall t,$$

$\alpha_0 = 1$ and $\epsilon_t \sim N(0, \mathbf{I})$ is standard Gaussian noise.

- The intuition behind is that we first modify the sample towards the direction of the “predicted” \mathbf{x}_0 , then we move to the direction pointing to \mathbf{x}_t and finally add some noise.
- Different choices of σ_t result in different generative processes for the same trained DDPM model.

- Now in order to speed up the generation process, we can assume that the forward process is defined not on all the latent variables $\mathbf{x}_{1:T}$, but on a subset $\{\mathbf{x}_{\tau_1}, \dots, \mathbf{x}_{\tau_S}\}$ with $1 \leq \tau_1 < \tau_2, \dots, \tau_S \leq T$.
- In order to use the DDPM model trained on all steps $1, \dots, T$, we require that the marginal distributions of the new forward process match the marginal distributions of the original DDPM:

$$q(\mathbf{x}_{\tau_i} \mid \mathbf{x}_0) = N(\sqrt{\alpha_{\tau_i}} \mathbf{x}_0, (1 - \alpha_{\tau_i}) \mathbf{I}).$$

- Now we can generate samples using the new forward process (with a fewer steps), for example:

$$\sigma_{\tau_i}(\eta) = \eta \sqrt{\frac{1 - \alpha_{\tau_{i-1}}}{1 - \alpha_{\tau_i}}} \sqrt{1 - \frac{\alpha_{\tau_i}}{\alpha_{\tau_{i-1}}}}.$$

- Song et al. (2021) show that using a *deterministic* generative process (with $\eta = 0$) produces samples of high quality. They refer to the deterministic approach as *denoising diffusion implicit model* (DDIM).

DDIM can generate high-quality images with fewer steps

Table 1: CIFAR10 and CelebA image generation measured in FID. $\eta = 1.0$ and $\hat{\sigma}$ are cases of DDPM (although Ho et al. (2020) only considered $T = 1000$ steps, and $S < T$ can be seen as simulating DDPMs trained with S steps), and $\eta = 0.0$ indicates DDIM.

S	CIFAR10 (32×32)					CelebA (64×64)					
	10	20	50	100	1000	10	20	50	100	1000	
η	0.0	13.36	6.84	4.67	4.16	4.04	17.33	13.73	9.17	6.53	3.51
	0.2	14.04	7.11	4.77	4.25	4.09	17.66	14.11	9.51	6.79	3.64
	0.5	16.66	8.35	5.25	4.46	4.29	19.86	16.06	11.01	8.09	4.28
	1.0	41.07	18.36	8.01	5.78	4.73	33.12	26.03	18.48	13.93	5.98
$\hat{\sigma}$	367.43	133.37	32.72	9.99	3.17	299.71	183.83	71.71	45.20	3.26	

Conditional generation with diffusion-based models

- The simplest way of conditioning the generation process on some information \mathbf{c} (e.g., class or text) is to use \mathbf{c} as extra inputs of the denoising model: $\epsilon_\theta(\mathbf{x}_t, t, \mathbf{c})$.
- Other ways of conditioning may provide better results.
- [Sohl-Dickstein et al. \(2015\)](#) proposed conditioning on label \mathbf{c} by using a classifier $\log p(\mathbf{c} | \mathbf{x}_t)$:
 1. Sample $\mathbf{x}_T \sim N(0, I)$
 2. For t from T to 2:
$$\mu_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right), \quad \Sigma_{t-1} = \sigma_t^2 \mathbf{I}$$
$$\mathbf{x}_{t-1} \sim N(\mu_{t-1} + s \Sigma_{t-1} \nabla_{\mathbf{x}_t} \log p(\mathbf{c} | \mathbf{x}_t), \Sigma_{t-1}), \quad s \text{ is a hyperparameter}$$
 3. Return \mathbf{x}_0

The classifier pulls the samples in the direction in which the probability of the desired class increases.

- The above conditional sampling is only valid for the *stochastic* diffusion sampling process, and cannot be applied to deterministic sampling methods like DDIM.
- Dhariwal and Nichol (2020) propose to address this problem in the following way. When we train a model for unconditional generation, we estimate the noise

$$\epsilon_\theta(\mathbf{x}_t) \approx \frac{\mathbf{x}_t - \bar{\alpha}_t \mathbf{x}_0}{\sqrt{1 - \bar{\alpha}_t}}$$

- $\epsilon_\theta(\mathbf{x}_t)$ can be used to approximate the score function. Assuming small noise ($\bar{\alpha}_t \approx 1$), we get:

$$\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) \approx \frac{\mathbf{x}_0 - \mathbf{x}_t}{1 - \bar{\alpha}_t} \approx -\frac{\mathbf{x}_t - \bar{\alpha}_t \mathbf{x}_0}{1 - \bar{\alpha}_t} = -\frac{\epsilon_\theta(\mathbf{x}_t)}{\sqrt{1 - \bar{\alpha}_t}}$$

or

$$\epsilon_\theta(\mathbf{x}_t) \approx -\sqrt{1 - \bar{\alpha}_t} \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t).$$

- When we perform conditional generation, we want to sample from the distribution whose score function is

$$\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) + \nabla_{\mathbf{x}_t} \log p_\phi(\mathbf{c}|\mathbf{x}_t)$$

- This means that our modified noise estimation should change from

$$\epsilon_\theta(\mathbf{x}_t) \approx -\sqrt{1 - \bar{\alpha}_t} \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t)$$

to

$$\begin{aligned}\hat{\epsilon}_\theta(\mathbf{x}_t) &\approx -\sqrt{1 - \bar{\alpha}_t} [\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) + \nabla_{\mathbf{x}_t} \log p_\phi(\mathbf{c}|\mathbf{x}_t)] \\ &= \epsilon_\theta(\mathbf{x}_t) - \sqrt{1 - \bar{\alpha}_t} \nabla_{\mathbf{x}_t} \log p_\phi(\mathbf{c}|\mathbf{x}_t)\end{aligned}$$

- This modifies the DDIM sampling in the following way:

1. Sample $\mathbf{x}_T \sim N(0, I)$

2. For t from T to 2:

$$\hat{\epsilon} = \epsilon_\theta(\mathbf{x}_t, t) - \sqrt{1 - \bar{\alpha}_t} \nabla_{\mathbf{x}_t} \log p_\phi(\mathbf{c}|\mathbf{x}_t)$$

$$\mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}} \left(\frac{\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \hat{\epsilon}}{\bar{\alpha}_t} \right) + \sqrt{1 - \bar{\alpha}_{t-1}} \hat{\epsilon}$$

3. Return \mathbf{x}_0

- Classifier guidance complicates the training pipeline: it requires an extra classifier *trained on noisy data* (not possible to plug in a pre-trained classifier).
- Furthermore, classifier-guided diffusion sampling can be interpreted as attempting to confuse an image classifier with a gradient-based adversarial attack.
- [Ho and Salimans \(2022\)](#) propose to use classifier-free guidance.
- The training procedure of the diffusion model is modified to learn two versions of the noise model: one with conditioning $\epsilon(\mathbf{x}_t, \mathbf{c})$ and one without conditioning $\epsilon(\mathbf{x}_t)$. The unconditional model is implemented by inputting a special null token to the conditional model.
- The generation procedure is modified such that the noise used to generate a new sample becomes

$$\hat{\epsilon}_t = (1 + w)\epsilon_{\theta}(\mathbf{x}_t, \mathbf{c}) - w\epsilon_{\theta}(\mathbf{x}_t)$$

where w is a parameter which determines the amount of guidance.

Palette: Image-to-Image Diffusion Models (Saharia et al., 2022)

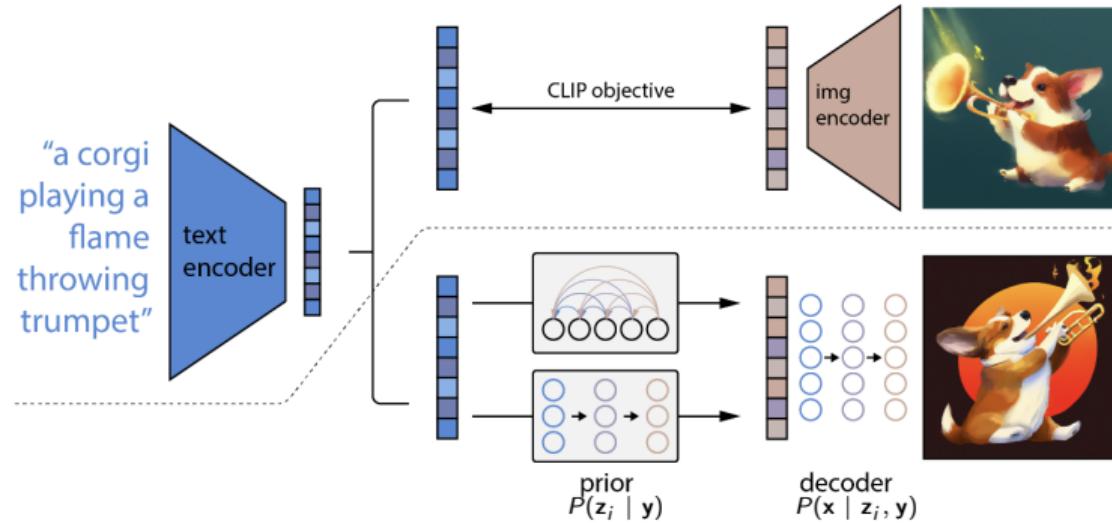
- Palette uses a U-Net architecture with additional conditioning of the source image via concatenation. The same model is used for different image-to-image translation tasks: colorization, inpainting, uncropping, and JPEG restoration.



DALL·E-2
(Ramesh et al., 2021)

DALL-E-2: Text-conditional image generation (Ramesh et al., 2021)

- DALL-E-2 considers the task of text-to-image generation: generation of image x conditioned on given textual description y .
- Text y is first processed by a text encoder which is pre-trained with a procedure called CLIP.
- The CLIP text embedding is an input of a generative model $p(z_i | y)$ of image embeddings z_i .
- Finally, the generated image embedding z_i is transformed to an image using a diffusion-based generative model $P(x | z_i, y)$ which uses image embedding z_i as conditioning information.



- Option 1. Autoregressive (AR)
 - Reduce the dimensionality of the CLIP image embeddings z_i from 1024 to 319.
 - Order the principal components and quantize each of the 319 dimensions into 1024 discrete buckets.
 - Predict the resulting sequence with the Transformer decoder.
 - The text caption y and the CLIP text embedding z_t are encoded as a prefix to the sequence.
- Option 2. Diffusion prior
 - The continuous vector z_i is modelled using a Gaussian diffusion model conditioned on the caption y .
 - Transformer decoder (with causal attention) is applied to a sequence consisting of encoded text, the CLIP text embedding, an embedding for the diffusion timestep, the noised CLIP image embedding, and a final embedding whose output from the Transformer is used to predict the unnoised CLIP image embedding
 - Simple mean-squared error loss is used:

$$\mathcal{L} = E_{t \sim [1, T], z^i(t) \sim q_t} \|f_\theta(z_i(t), t, y) - z_i\|^2$$

Decoder $P(x | z_i, y)$: A generative model of images x conditioned on CLIP image embeddings

- Images are generated using diffusion models. Conditioning on CLIP image embeddings z_i is done this way:
 - Project and add CLIP embeddings to the timestep embedding
 - Project CLIP embeddings into four extra tokens of context that are concatenated to the sequence of outputs from the text encoder.
 - The previous version called GLIDE ([Nichol et al., 2021](#)) used conditioning similar to classifier guidance:

$$\hat{\mu}_\theta(x_t | c) = \mu_\theta(x_t | c) + s \cdot \Sigma_\theta(x_t | c) \nabla_{x_t} (f(x_t) \cdot g(c))$$

where the classifier is replaced with a CLIP model: $f(x)$ and $g(c)$ are the CLIP image and caption encoders, respectively.

- To generate high resolution images, they train two diffusion upsampler models: from 64×64 to 256×256 , and from 256×256 to 1024×1024 .
 - For the upsampling model, the downsampled image 64×64 is passed as extra conditioning input to the U-Net. This is similar to VQ-VAE-2 when the codes in high-resolution are conditioned on low-resolution codes.

DALL-E-2: Selected samples, more examples here



an espresso machine that makes coffee from human souls, artstation



panda mad scientist mixing sparkling chemicals, artstation



a corgi's head depicted as an explosion of a nebula



a dolphin in an astronaut suit on saturn, artstation

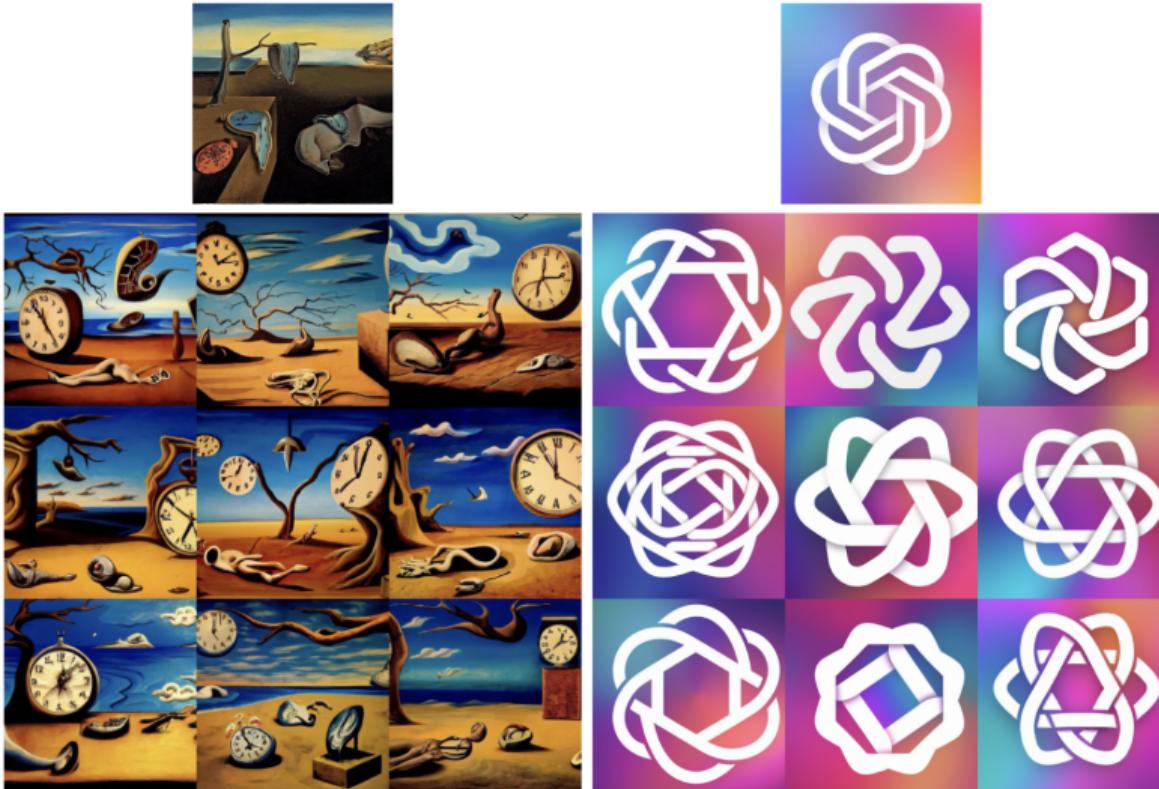


a propaganda poster depicting a cat dressed as french emperor napoleon holding a piece of cheese



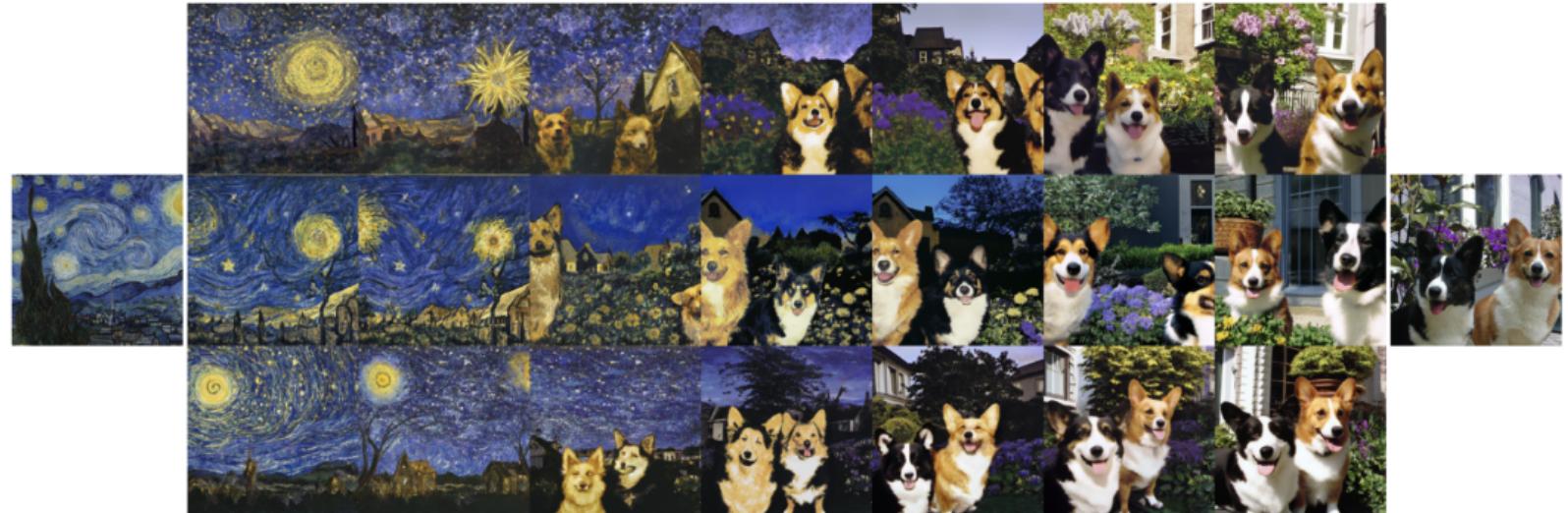
a teddybear on a skateboard in times square

DALL-E-2: Variations of one image



Variations of an input image by encoding with CLIP and then decoding with a diffusion model.

DALL-E-2: Variations between two images



Variations between two images by interpolating their CLIP image embedding and then decoding with a diffusion model.

Latent diffusion diffusion models

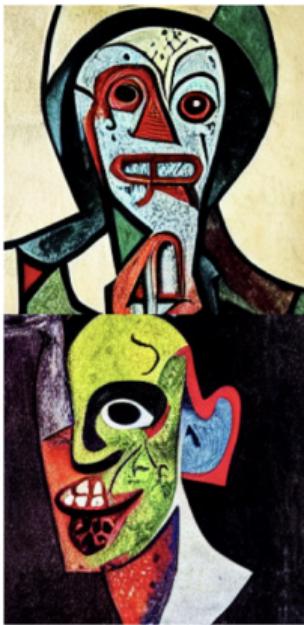
- Training of powerful diffusion models (e.g., OpenAI's Dalle·2 or Google's Imagen) often consumes hundreds of GPU days. And inference is expensive due to sequential evaluations.
- Latent diffusion models (LDMs): train a diffusion model applied to the latent space of a powerful pre-trained autoencoder.
- The autoencoder is trained by combination of a perceptual loss and a patch-based adversarial objective.
- The autoencoder reduces the resolution of the original image by factor f . $f = 4, 8, 16$ give a good balance between efficiency and perceptually faithful results.

LDM samples conditioned on language prompts

*'A street sign that reads
"Latent Diffusion"'*



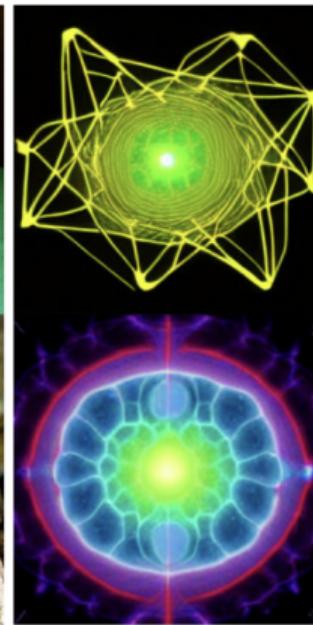
*'A zombie in the
style of Picasso'*



*'An image of an animal
half mouse half octopus'*

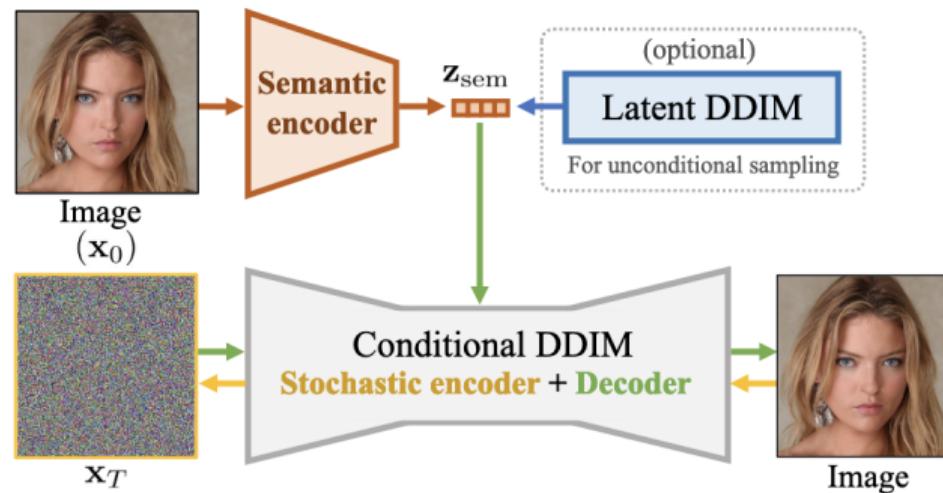


*'An illustration of a slightly
conscious neural network'*



Diffusion autoencoder (Preechakul et al., 2021)

- Standard diffusion models do not encode the input a (low-dimensional) representation. There are extensions which can do that.



Encoder path (semantic) : Image $\rightarrow z_{sem}$

Encoder path (stochastic) : Image $\rightarrow x_T$

Decoder path : $(z_{sem}, x_T) \rightarrow$ Image (reconstructed)

Image manipulation with a diffusion autoencoder

- The model allows manipulation of an existing image.

