Aalto University

CS-E4890: Deep Learning
Denoising-based generative modeling

Alexander Ilin

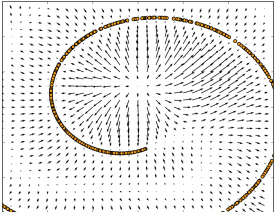
Representation learning by denoising

- Previously, we considered denoising autoencoders (DAE)
  - Corrupt data with noise: $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$
  - train a network $\mathbf{d}(\tilde{\mathbf{x}})$ to produce clean data $\mathbf{x}$ by minimizing $\mathcal{L} = E\left[\|\mathbf{d}(\tilde{\mathbf{x}}) - \mathbf{x}\|^2\right]$.
- The task of denoising encourages learning of useful representations (see the assignment).

- This lecture: we want to use the principle of denoising to build *a generative model* (which can generate new samples).
- Recall that for Gaussian corruption $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$, 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. $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ is often called a *score function*.



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

Denoising score matching

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

- 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

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

- Training procedure: corrupt clean sample $\mathbf{x}$ with Gaussian noise $\boldsymbol{\epsilon}$ to get $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}$ and train a neural network to predict the noise $\boldsymbol{\epsilon}$ (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 $\nabla_{\mathbf{x}} \log p(\mathbf{x})$, we can sample from the corresponding distribution using Langevin dynamics, a sampling procedure which iterates the following:

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

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

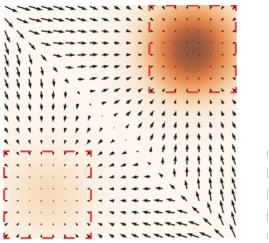
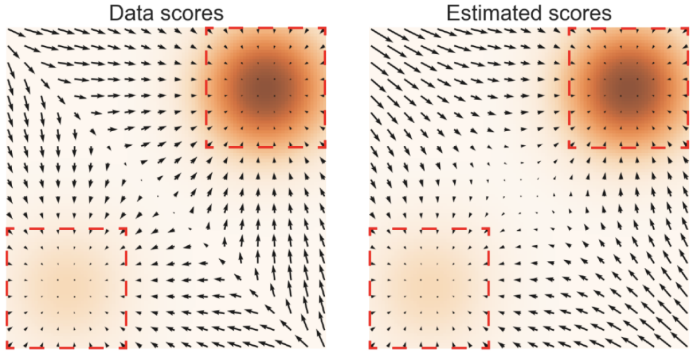


Image from (Song and Ermon, 2020)

- Question: Why do we need to add noise $\mathbf{z}_t$?
- If we have a neural network $\mathbf{s}_\theta(\mathbf{x})$ which has been trained such that $\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$, we can generate samples from $p(\mathbf{x})$ using $\mathbf{s}_\theta(\mathbf{x}_{t-1})$ instead of $\nabla_{\mathbf{x}} \log p(\mathbf{x}_{t-1})$.

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 $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) \approx \mathbf{s}_\theta(\mathbf{x})$.

Problem 2: Bad mixing of Langevin dynamics

- Consider a mixture distribution

$$p_{\text{data}}(\mathbf{x}) = \pi p_1(\mathbf{x}) + (1-\pi)p_2(\mathbf{x})$$

  - $p_1(\mathbf{x})$ and $p_2(\mathbf{x})$ are normalized distributions with disjoint supports
  - $\pi \in (0, 1)$.

- In the support of $p_1(\mathbf{x})$, the score does not depend on π:

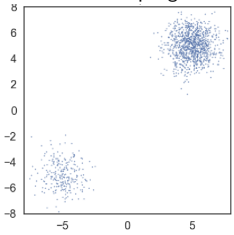
$$\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) = \nabla_{\mathbf{x}}(\log \pi + \log p_1(\mathbf{x})) = \nabla_{\mathbf{x}} \log p_1(\mathbf{x})$$

- In the support of $p_2(\mathbf{x})$, the score does not depend on π either:

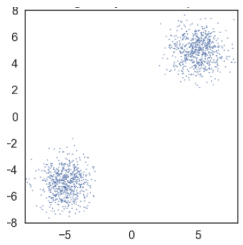
$$\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) = \nabla_{\mathbf{x}}(\log(1-\pi) + \log p_2(\mathbf{x})) = \nabla_{\mathbf{x}} \log p_2(\mathbf{x})$$

- Langevin dynamics estimate the relative weights between the two modes incorrectly.

Exact sampling



Sampling using Langevin dynamics with exact scores



Noise Conditional Score Networks (NCSN)

- 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 $\sigma_1 > \sigma_2 > \ldots \sigma_L$ and estimate scores $\nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x})$ corresponding to all noise levels by training a single neural network $s_\theta(\mathbf{x}, \sigma)$ to minimize the loss:

$$\mathcal{L} = \frac{1}{L} \sum_{i=1}^{L} \lambda(\sigma_i) \mathcal{L}_i,$$

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

  $\mathcal{L}_i$ is the denoising score matching objective for σ_i, weights $\lambda(\sigma) = \sigma$ are chosen emperically.
  2. Generate samples using annealed Langevin dynamics.

Sample generation via 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 $\sigma_L \approx 0$.

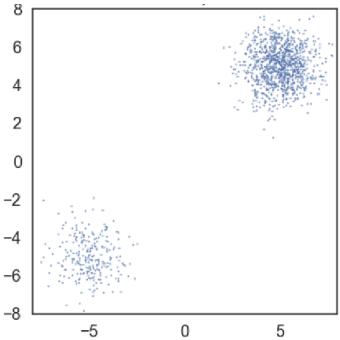
Algorithm 1 Annealed Langevin dynamics.

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

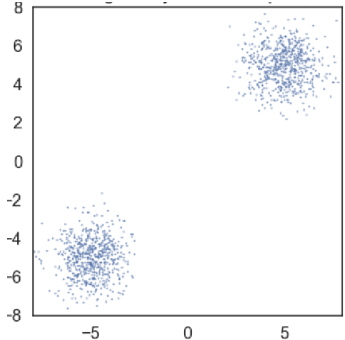
Initialize $\tilde{\mathbf{x}}_0$
for $i \leftarrow 1$ to L **do**
  $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$ ▷ α_i is the step size.
  for $t \leftarrow 1$ to T **do**
    Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
    $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}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

Annealed Langevin dynamics: Toy example

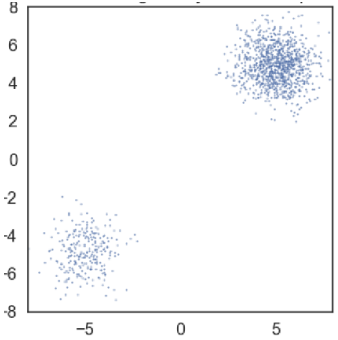
- Annealed Langevin dynamics recover the relative weights faithfully.



Exact sampling



Sampling using Langevin dynamics with exact scores



Sampling using annealed Langevin dynamics with exact scores

Generated samples with NCSN (Song and Ermon, 2020)

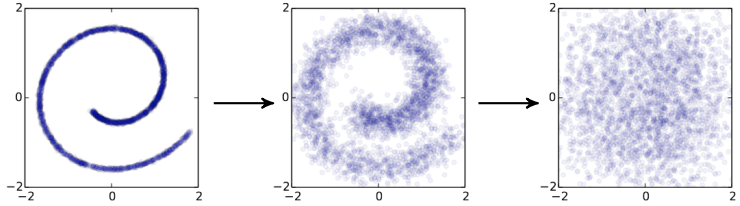
- When applied to images, the model $\mathbf{s}_\theta(\mathbf{x}, \sigma)$ is a U-Net with dilated convolution.
- They use a modified version of conditional instance normalization to provide conditioning on σ_i.

Diffusion probabilistic models

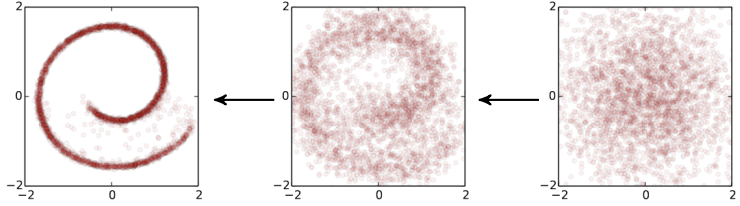
(Sohl-Dickstein et al., 2015)

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

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



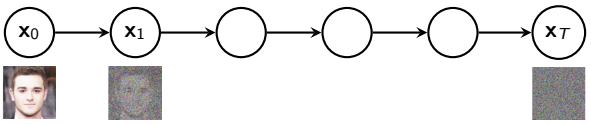
- Learn the generative model which is defined by a reversal of this diffusion process



Forward diffusion process

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

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

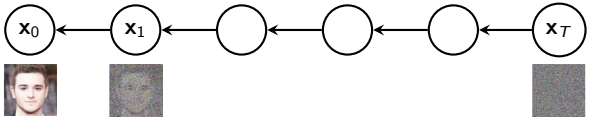


- $q(\mathbf{x}_t \mid \mathbf{x}_0)$ has a closed form and converges to $N(0, \mathbf{I})$ when $t \to \infty$:

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

- If we select 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 $\mathbf{x}_{T-1}, \mathbf{x}_{T-2}, ..., \mathbf{x}_0$.
- For computational convenience, we define the reverse process as

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

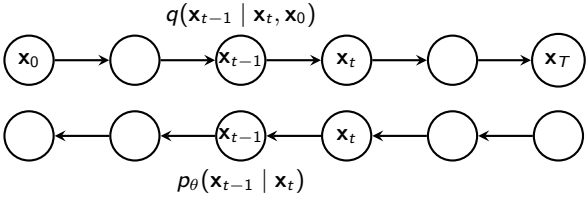
  and the task is to learn $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$, $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)$ to maximize the log-likelihood $E[\log p_\theta(\mathbf{x}_0)]$.
- The original paper (Sohl-Dickstein et al., 2015) proposes estimation of model parameters θ by minimizing the variational 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

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

- 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 $\mathbf{x}_0$.



Diffusion probabilistic models: Loss function

- Due to the selected diffusion process, $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ has a closed form:

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

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

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

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

Denoising Diffusion Probabilistic Models (DDPM)

(Ho et al., 2020)

- Denoising Diffusion Probabilistic Models (Ho et al. 2020) are diffusion probabilistic models with the following simplifications.
  - We do not train β_t used in the forward process, which makes the first term of the loss a constant:

$$L_t = E_q \left[D_{\mathrm{KL}}(q(\mathbf{x}_T \mid \mathbf{x}_0) \parallel p(\mathbf{x}_T)) \right] = \text{const}$$

  - We use fixed diagonal covariance matrices in the reverse process

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

    both $\sigma_t^2 = \beta_t$ and $\sigma_t^2 = \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$ work well in practice.
  - This simplifies the loss such that we can only care about the means of the distributions in the reverse process:

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

DDPM: Loss function

- If $\boldsymbol{\epsilon}$ is the noise instance that was used to produce $\mathbf{x}_t$ from $\mathbf{x}_0$

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}\,,$$

  the target is expressed (see the derivations on the next page) as

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

- It is then convenient to use a parameterization for the denoising model that has a similar form:

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

- This parameterization leads to the following loss

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

  In practice, the authors drop the weights, which does not compromise the performance.

DDPM: derivations

Suppose that $\boldsymbol{\epsilon}$ is the noise instance that was used to produce $\mathbf{x}_t$ from $\mathbf{x}_0$:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon},$$

then

$$\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon})$$

and we get:

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

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 $t \sim \text{Uniform}(\{1, ..., T\})$
  3. Generate noise $\epsilon \sim N(0, I)$ and compute corrupted samples

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

  4. Compute the loss $\mathcal{L} = ||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)||^2$
  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: $\mathbf{x}_{t-1} \sim N\left(\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right), \sigma_t^2\mathbf{I}\right)$
  3. Compute generated sample $\mathbf{x}_0 = \frac{1}{\sqrt{\alpha_1}}\left(\mathbf{x}_1 - \frac{\beta_1}{\sqrt{1-\bar{\alpha}_1}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_1, 1)\right)$

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 $\boldsymbol{\epsilon}$ that led to $\mathbf{x}_t$ from $\mathbf{x}_0$:

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

- 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.

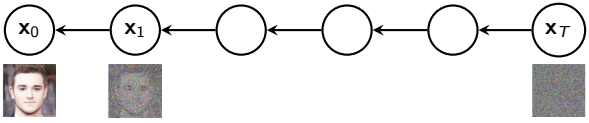
Model	FID	sFID	Prec	Rec
ImageNet 128×128				
BigGAN-deep [5]	6.02	7.18	**0.86**	0.35
LOGAN† [68]	3.36			
ADM	5.91	**5.09**	0.70	**0.65**
ADM-G (25 steps)	5.98	7.04	0.78	0.51
ADM-G	**2.97**	**5.09**	0.78	0.59
ImageNet 256×256				
DCTransformer† [42]	36.51	8.24	0.36	**0.67**
VQ-VAE-2†‡ [51]	31.11	17.38	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)

Diffusion models: Increasing the sampling speed

- 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.
- Can we speed the generation process?

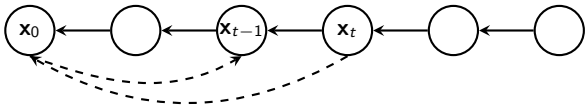
Another form for the DDPM generation step

- Song et al. (2021) re-write the DDPM generation steps in the following form:

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

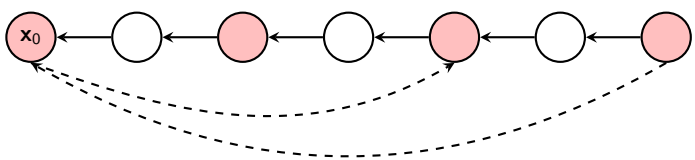
  with $\sigma_t = \sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}} \sqrt{1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}}$, $\bar{\alpha}_0 = 1$ and $\boldsymbol{\epsilon}_t \sim N(0, \mathbf{I})$ is standard Gaussian noise.

- 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.



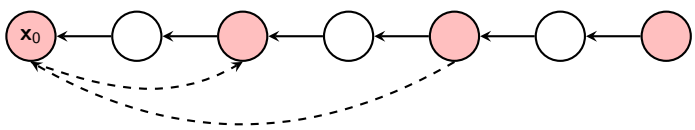
Denoising diffusion implicit model (DDIM) (Song et al., 2021)

- The authors show that different choices of σ_t can lead to different well-grounded generative processes for *the same trained model*.
- The special case of $\sigma_t = 0$ corresponds to a deterministic sampling procedure. The authors call this approach a *denoising diffusion implicit model* (DDIM).
- To speed up the generation process, one can assume that the forward process (and therefore the reverse process as well) is defined on a subset of steps $1 \leq \tau_1 < \tau_2, ..., \tau_S \leq T$. Now we can generate samples using fewer steps:



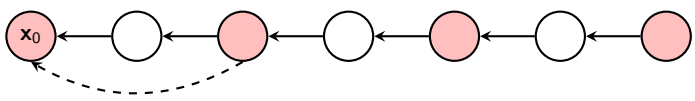
Denoising diffusion implicit model (DDIM) (Song et al., 2021)

- The authors show that different choices of σ_t can lead to different well-grounded generative processes for *the same trained model*.
- The special case of $\sigma_t = 0$ corresponds to a deterministic sampling procedure. The authors call this approach a *denoising diffusion implicit model* (DDIM).
- To speed up the generation process, one can assume that the forward process (and therefore the reverse process as well) is defined on a subset of steps $1 \leq \tau_1 < \tau_2, ..., \tau_S \leq T$. Now we can generate samples using fewer steps:



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

- The authors show that different choices of σ_t can lead to different well-grounded generative processes for *the same trained model*.
- The special case of $\sigma_t = 0$ corresponds to a deterministic sampling procedure. The authors call this approach a *denoising diffusion implicit model* (DDIM).
- To speed up the generation process, one can assume that the forward process (and therefore the reverse process as well) is defined on a subset of steps $1 \leq \tau_1 < \tau_2, ..., \tau_S \leq T$. Now we can generate samples using fewer steps:



DDIM can generate high-quality images with fewer steps

- In the experiments, the authors used $\sigma_{\tau_i}(\eta) = \eta \sqrt{\frac{1-\bar{\alpha}_{\tau_{i-1}}}{1-\bar{\alpha}_{\tau_i}}} \sqrt{1 - \frac{\bar{\alpha}_{\tau_i}}{\bar{\alpha}_{\tau_{i-1}}}}$ and experimented with a different number of steps in the sampling procedure.

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.

		CIFAR10 (32×32)					CelebA (64×64)				
	S	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

Classifier guided sampling

- 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 $\log p(\mathbf{c} \mid \mathbf{x}_t)$:

  1. Sample $\mathbf{x}_T \sim N(0, I)$
  2. For t from T to 2:

     $$\boldsymbol{\mu}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right), \quad \boldsymbol{\Sigma}_{t-1} = \sigma_t^2 \mathbf{I}$$

     $$\mathbf{x}_{t-1} \sim N\left(\boldsymbol{\mu}_{t-1} + s\, \boldsymbol{\Sigma}_{t-1} \nabla_{\mathbf{x}_t} \log p(\mathbf{c} \mid \mathbf{x}_t), \boldsymbol{\Sigma}_{t-1}\right), \qquad s \text{ is a hyperparameter}$$

  3. Return $\mathbf{x}_0$

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

Classifier guided DDIM sampling (Dhariwal and Nichol, 2020)

- 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

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

- $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t)$ can be used to approximate the score function.
  - Recall from the beginning of this lecture that $\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) \approx (\mathbf{x} - \tilde{\mathbf{x}})/\sigma^2$.
  - Assuming small noise ($\bar{\alpha}_t \approx 1$), we get:

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

  or

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

Classifier guided DDIM sampling (Dhariwal and Nichol, 2020)

- 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

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

to

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

Classifier guided DDIM sampling (Dhariwal and Nichol, 2020)

- 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-free diffusion guidance (Ho and Salimans, 2022)

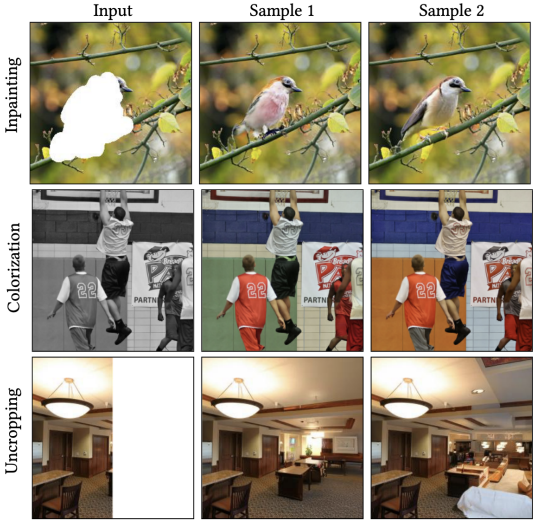
- Classifier guidance complicates the training pipeline: it requires an extra classifier *trained on noisy data* (not possible to plug in a pre-trained classifier).
- Furthermore, classifier-guided diffusion sampling can be interpreted as attempting to confuse an image classifier with a gradient-based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

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

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

Image generation conditioned on images

- Diffusion generative models can easily be used to generate images conditioned on other images.
- Palette (Saharia et al., 2022) simply concatenates the input of the denoising model with the conditioning image. The same model is used for different image-to-image translation tasks: colorization, inpainting, uncropping, and JPEG restoration.

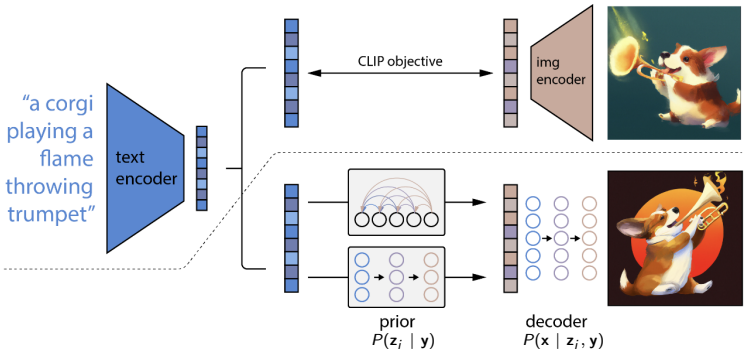


DALL·E-2

(Ramesh et al., 2021)

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

- Text-to-image generation: generation of image $\mathbf{x}$ conditioned on given textual description $\mathbf{y}$.
- Text $\mathbf{y}$ 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 $p(\mathbf{z}_i \mid \mathbf{y})$ of image embeddings $\mathbf{z}_i$.
- Finally, the generated image embedding $\mathbf{z}_i$ is transformed to an image using a diffusion-based generative model $P(\mathbf{x} \mid \mathbf{z}_i, \mathbf{y})$ which uses image embedding $\mathbf{z}_i$ as conditioning information.



Prior $P(\mathbf{z}_i \mid \mathbf{y})$: A generative model of CLIP image embeddings

- Option 1. Autoregressive (AR)
  - Reduce the dimensionality of the CLIP image embeddings $\mathbf{z}_i$ 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 $\mathbf{y}$ and the CLIP text embedding $\mathbf{z}_t$ are encoded as a prefix to the sequence.
- Option 2. Diffusion prior
  - The continuous vector $\mathbf{z}_i$ is modelled using a Gaussian diffusion model conditioned on the caption $\mathbf{y}$.
  - 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 $P(\mathbf{x} \mid \mathbf{z}_i, \mathbf{y})$: A generative model of images x conditioned on CLIP image embeddings

- Images are generated using diffusion models. Conditioning on CLIP image embeddings $\mathbf{z}_i$ 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:

    $$\hat{\boldsymbol{\mu}}_\theta(\mathbf{x}_t \mid \mathbf{c}) = \boldsymbol{\mu}_\theta(\mathbf{x}_t \mid \mathbf{c}) + s \cdot \boldsymbol{\Sigma}_\theta(\mathbf{x}_t \mid c) \nabla_{\mathbf{x}_t}(f(\mathbf{x}_t) \cdot g(\mathbf{c}))$$

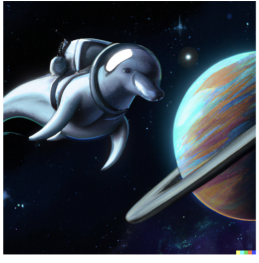
    where the classifier is replaced with a CLIP model: $f(\mathbf{x})$ and $g(\mathbf{c})$ are the CLIP image and caption encoders, respectively.
- To generate high resolution images, they train two diffusion upsampler models: from 64×64 to 256×256, and from 256×256 to 1024×1024.
  - For the upsampling model, the downsampled image 64x64 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

Stable diffusion (Rombach et al., 2021)

- 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.
- Stable diffusion demo

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.

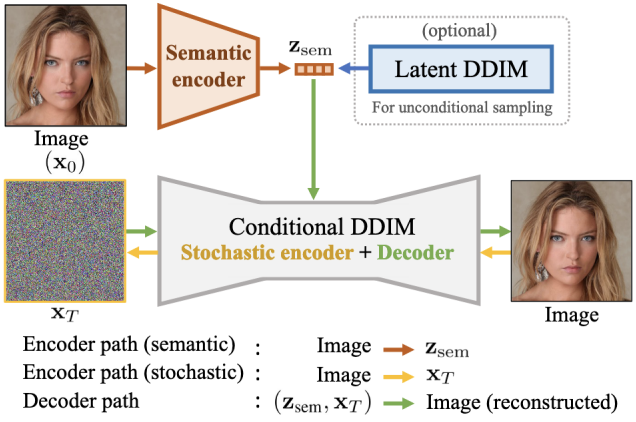


Image manipulation with a diffusion autoencoder

- The model allows manipulation of an existing image.

Home assignment

Assignment 08_diffusion

- Implement and train a diffusion-based generative model for MNIST.
- Use a trained diffusion model for in-painting.