Aalto University

CS-E4890: Deep Learning
Unsupervised learning via denoising

Alexander Ilin

Unsupervised learning via denoising

- We have previously seen that the task of denoising can encourage learning of useful representations.
- Recall denoising autoencoders:
  - Feed inputs corrupted with noise:

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

  - Train a network $\mathbf{s}(\tilde{\mathbf{x}})$ to produce clean data $\mathbf{x}$ in the output:

$$\mathcal{L} = \mathbb{E}\{\|\mathbf{s}(\tilde{\mathbf{x}}) - \mathbf{x}\|^2\}$$

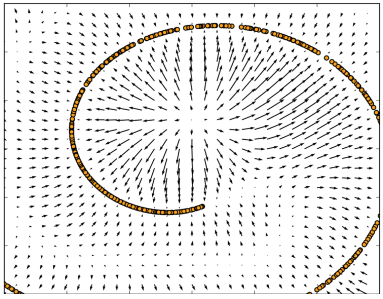
- We trained a denoising autoencoder in the assignment and it was able to learn useful features in the bottleneck layer.

Denoising score matching

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

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

  where $p(\tilde{\mathbf{x}})$ is the pdf of the distribution of corrupted data.
- For small σ, $p(\tilde{\mathbf{x}}) \approx p(\mathbf{x})$ and therefore we (implicitly) learn the properties of $p(\mathbf{x})$ by training a denoising function.



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

- $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ is often called a *score function*.
- This type of modeling is sometimes called *denoising score matching* (learning the score function by denoising).

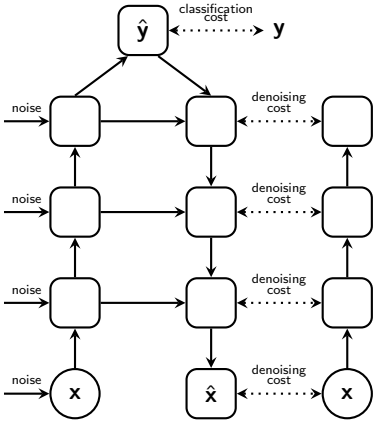
Semi-supervised learning

with Ladder networks

(Rasmus et al., 2015)

Ladder networks (Rasmus et al., 2015)

- Ladder networks used the principle of denoising to learn useful features in the semi-supervised settings (learning from both labeled and unlabeled examples).
- The architecture resembles a ladder (or a U-net).
- The bottom-up pass produces label **y** for a given input **x**.
- For labeled examples, we can compute the standard classification (e.g., cross-entropy) loss using the network output $\hat{\mathbf{y}}$ and the correct label **y**.

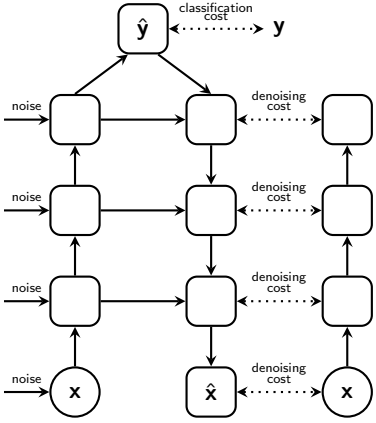


Ladder networks: Denoising

- The inputs **x** are corrupted by noise during training (e.g, we never use clean images as inputs).
- The top-down pass tries to reconstruct the clean (without noise) input **x**.
- For all examples (both labeled and unlabeled), we compute the denoising cost at the bottom:

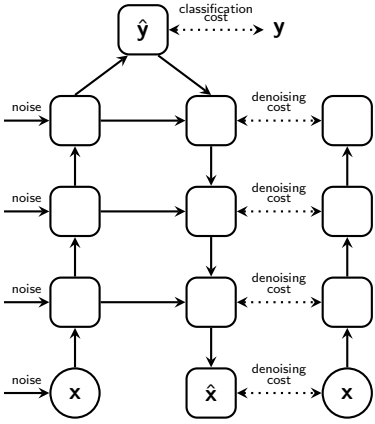
$$\text{denoising cost} = \|\mathbf{x} - \hat{\mathbf{x}}\|^2$$

- The minimized cost is the sum of the classification and denoising costs.



Ladder networks: Intuitions

- Intuition: In order to reconstruct the clean image from a noisy image, one has to understand what features are commonly present in images.
  - Note: corruption and denoising happens on multiple representation levels.
- Therefore, denoising is an auxiliary task that helps model $p(\mathbf{x})$ and hopefully develop features useful for the primary classification task.
- The label itself cannot contain enough information to reconstruct the input. We need skip connections to pass low-level details from the bottom-up pass.
- Ladder networks inspired modern models for deep semi-supervised learning.



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:

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

  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 $\mathbf{s}_\theta(\mathbf{x})$ which has been trained such that $\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$, we can generate samples from $p(x)$ using $\mathbf{s}_\theta(\mathbf{x}_{t-1})$ instead of $\nabla_{\mathbf{x}} \log p(\mathbf{x}_{t-1})$.

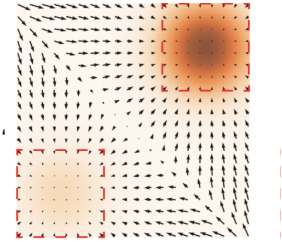
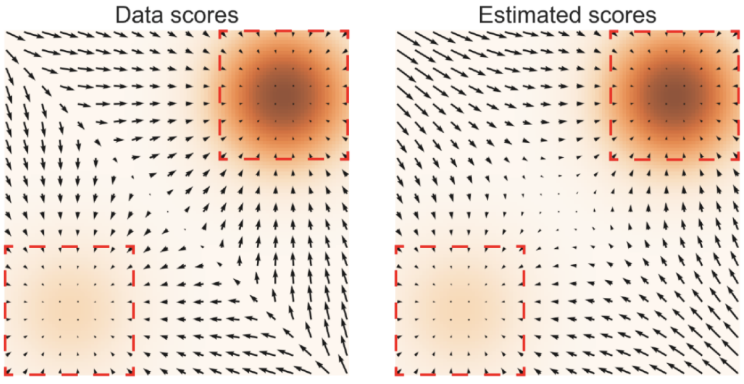


Image from (Song and Ermon, 2020)

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

Problem 2: Bad mixing of Langevin dynamics

- Consider a mixture distribution $p_{\text{data}}(\mathbf{x}) = \pi p_1(\mathbf{x}) + (1-\pi)p_2(\mathbf{x})$, 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 $p_1(\mathbf{x})$:

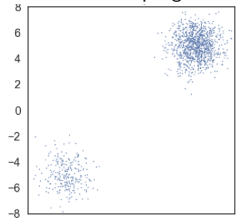
$$\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) = \nabla_{\mathbf{x}}(\log \pi + \log p_1(\mathbf{x})) = \nabla_{\mathbf{x}} \log p_1(\mathbf{x})$$

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

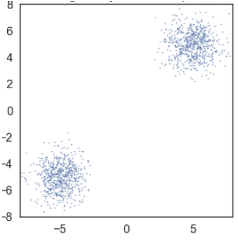
$$\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) = \nabla_{\mathbf{x}}(\log(1-\pi) + \log p_2(\mathbf{x})) = \nabla_{\mathbf{x}} \log p_2(\mathbf{x})$$

- 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.

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. Perturbe the data using various levels of noise $\sigma_1 > \sigma_2 > \ldots \sigma_L$ 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

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

  - $\mathcal{L}_i$ is the denoising score matching objective for σ_i
  - coefficients $\lambda(\sigma) = \sigma$ (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 $\sigma_L \approx 0$.

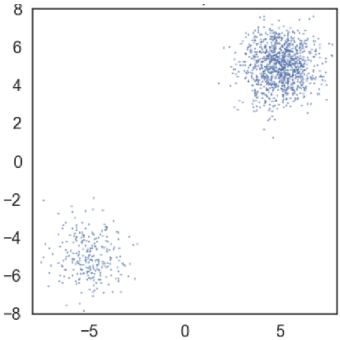
Algorithm 1 Annealed Langevin dynamics.

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

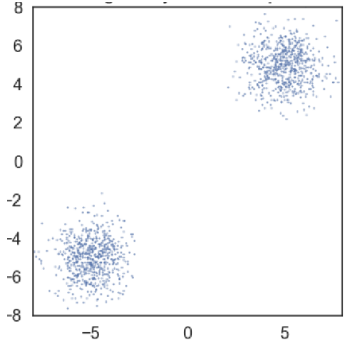
Initialize $\tilde{\mathbf{x}}_0$
for $i \leftarrow 1$ to L **do**
  $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$ ▷ α_i is the step size.
  for $t \leftarrow 1$ to T **do**
    Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
    $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}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

Annealed Langevin dynamics: Toy example

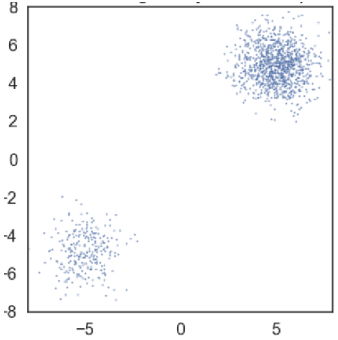
- Annealed Langevin dynamics recover the relative weights faithfully.



Exact sampling



Sampling using Langevin dynamics with exact scores



Sampling using annealed Langevin dynamics with exact scores

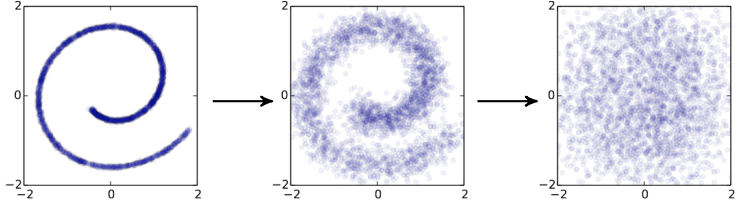
Generated samples with NCSN (Song and Ermon, 2020)

- 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.

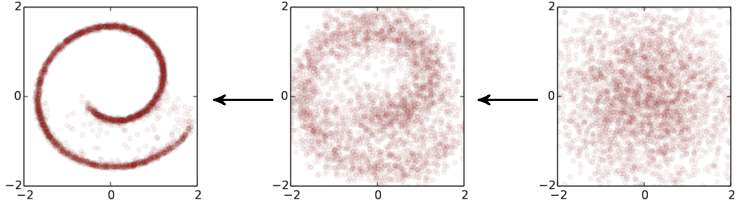
Diffusion probabilistic models

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

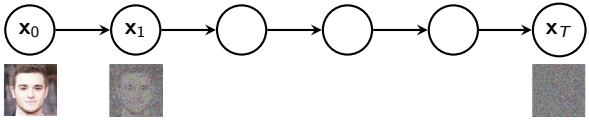
- Define a forward diffusion process which converts any complex data distribution into a simple, tractable, distribution.



- 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 $\mathbf{x}_1, ..., \mathbf{x}_T$ by progressively adding Gaussian noise:

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

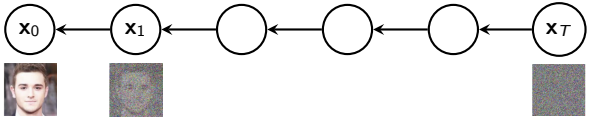
- $q(\mathbf{x}_t \mid \mathbf{x}_0)$ has a closed form:

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

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

- 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 $\mathbf{x}_{T-1}, \mathbf{x}_{T-2}, ..., \mathbf{x}_0$.
- 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:

$$E[-\log p_\theta(\mathbf{x}_0)] \leq E_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] = E_q\left[-\log p(\mathbf{x}_T) - \sum_{t\geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})}\right]$$

Reverse (generative) process (Sohl-Dickstein et al., 2015)

- This loss can be re-written 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 $\mathbf{x}_0$.
- 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!

Ho et al. (2020) proposed to simplify the diffusion model:

1. Use fixed variances β_t in the forward process.
2. Use fixed diagonal covariance matrces 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.

- This simplifies the loss terms:

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

- One can show (see next slide) that the target for the denoising model $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$ can be written as

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

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

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, \quad \text{with } \boldsymbol{\epsilon} \sim N(0, \mathbf{I})$$

DDPM: Simplifying the loss

With the following identities

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \quad \text{with } \boldsymbol{\epsilon} \sim N(0, I)$$

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

we get

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

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

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

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

$$= \frac{\mathbf{x}_t}{(1-\bar{\alpha}_t)\sqrt{\alpha_t}}(1-\bar{\alpha}_{t-1}) - \frac{\beta_t}{\sqrt{\alpha_t}\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon} \qquad \text{because } \beta_t = 1-\alpha_t$$

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

DDPM: Loss function

- Since the target is expressed 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 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

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

- We need to define the model $-\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)$ to compute $L_0 = E_q[-\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)]$.
- Ho et al. (2020) assume that image data consists of integers in $\{0, 1, ..., 255\}$ scaled linearly to $[-1, 1]$. They set $\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)$ to an independent discrete decoder derived from the Gaussian $N(\mathbf{x}_0; \boldsymbol{\mu}_\theta(\mathbf{x}_1, 1), \sigma_1^2 \mathbf{I})$:

$$p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1) = \prod_{i=1}^{D} \int_{\delta-(x_0^i)}^{\delta+(x_0^i)} N(x; \mu_\theta^i(\mathbf{x}_1, 1), \sigma_1^2) dx$$

$$\delta+(x_0^i) = \begin{cases} \infty, & \text{if } x = 1 \\ x + \frac{1}{255}, & \text{if } x < 1 \end{cases} \qquad \delta-(x_0^i) = \begin{cases} -\infty, & \text{if } x = -1 \\ x - \frac{1}{255}, & \text{if } x > -1 \end{cases}$$

where D is the data dimensionality and the i superscript indicates extraction of one coordinate.

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

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 $\mathbf{x}_0$:

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

- 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

- Sohl-Dickstein et al. (2015) showed that one can condition generation on the label y by modifying the sampling procedure in the following way:

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

$$\mathbf{x}_{t-1} \sim N\left(\boldsymbol{\mu}_{t-1} + s\boldsymbol{\Sigma}_{t-1}\nabla_{\mathbf{x}_t} \log p(y \mid \mathbf{x}_t), \boldsymbol{\Sigma}_{t-1}\right)$$

3. Return $\mathbf{x}_0$

where s is a hyperparameter. The classifier $\log p(y \mid \mathbf{x}_t)$ pulls the samples in the direction in which the probability of the desired class increases.

- Another common practice is to use the conditioning information as extra inputs of the network which models the reverse process, for example, $\epsilon_\theta(\mathbf{x}_t, t, y)$.

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.
- 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)

- Song et al. (2021) proposed to accelerate the generation process by using only a sub-sequence $\{\tau_i\}$ of steps $[1, \dots, T]$. The training procedure of DDPM is unchanged!
- Each generation step is modified as

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

  which can be seen as sampling from a generalized generative process.
- In the experiments, the use $\sigma_{\tau_i}(\eta) = \eta\sqrt{(1-\alpha_{\tau_{i-1}}/(1-\alpha_{\tau_i})}\sqrt{1-\alpha_{\tau_i}/\alpha_{\tau_{i-1}}}$
- $\eta = 0$ which corresponds to a deterministic generative process (from $\mathbf{x}_T$ to $\mathbf{x}_0$), which they call *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.

		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**

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.

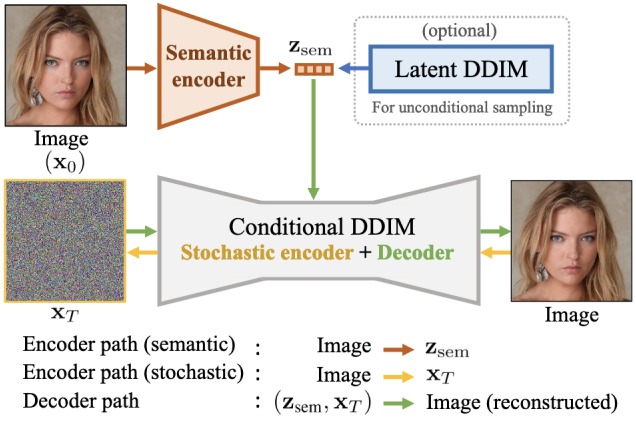


Image manipulation with a diffusion autoencoder

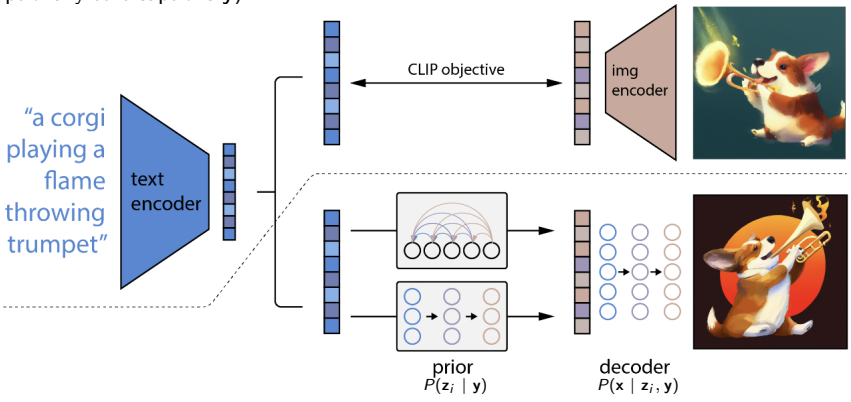
- The model allows manipulation of an existing image.

DALL·E-2

(Ramesh et al., 2021)

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

- The task is to generate an image $\mathbf{x}$ from a given textual description $\mathbf{y}$.
- DALL·E-2 consists of two components:
  1. $P(\mathbf{z}_i \mid \mathbf{y})$: A generative model of CLIP image embeddings $\mathbf{z}_i$ conditioned on captions $\mathbf{y}$.
  2. $P(\mathbf{x} \mid \mathbf{z}_i, \mathbf{y})$: A generative model of images $\mathbf{x}$ conditioned on CLIP image embeddings $\mathbf{z}_i$ (and optionally text captions $\mathbf{y}$).



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:

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

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:

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

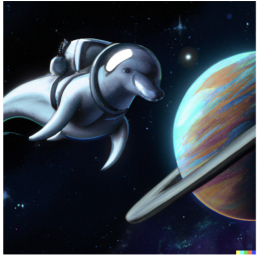
    where the classifier is replaced with a CLIP model: $f(\mathbf{x})$ and $g(\mathbf{c})$ are the CLIP image and caption encoders, respectively.
- To generate high resolution images, they train two diffusion upsampler models: from 64×64 to 256×256, and from 256×256 to 1024×1024.
  - For the upsampling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