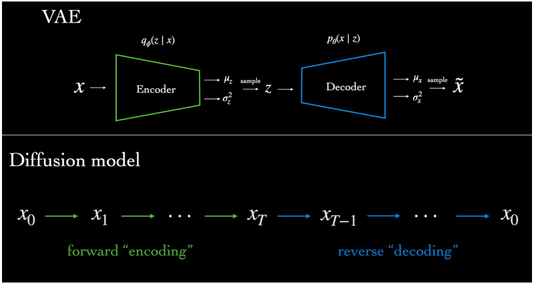
Machine Learning

Lecture 13: Introduction to Diffusion Models

Nevin L. Zhang
lzhang@cse.ust.hk

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

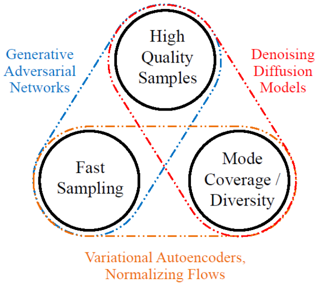
Introduction



https://www.youtube.com/watch?v=fbLgFrlTnGU

- Forward diffusion process: Adds small amount of Gaussian noise to the sample in T (1,000) steps
- Reverse diffusion process: Reverses the forward process and generates images from random noise in T steps (inference with a neural network T times).
- In contrast, VAE and Normalizing Flows generate images in one step (inference with a neural network once).

Introduction



Xiao et al, Tackling the Generative Learning Trilemma with Denoising Diffusion GANs, ICLR 2022

- GANs generate high-quality samples rapidly, but have poor mode coverage.
- VAEs and normalizing flows cover data modes faithfully, but they often suffer from low sample quality
- Diffusion models generate high-equality images (beats GAN) and have good mode coverage, but are slow in generating images (a weakness being addressed).

Outline

Sum of Two Gaussian Variables

- X and Y be two independent Gaussian random variables:

$$X \sim \mathcal{N}(\mu_X, \sigma_X^2), Y \sim \mathcal{N}(\mu_Y, \sigma_Y^2)$$

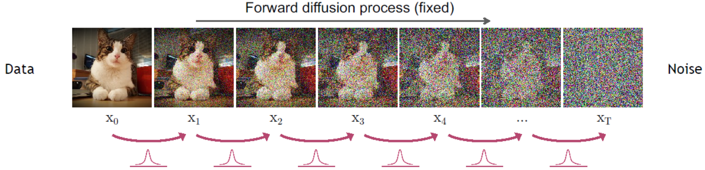
  that are normally distributed (and therefore also jointly so),

- $Z = X + Y$ is also Gaussian

$$Z \sim \mathcal{N}(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2)$$

- Side note 1: Density function of X is the convolution of those of X and Y.
- Side note 2: A mixture of gaussians is a weighted sum of gaussian densities, not a weighted sum of gaussian random variables.

DDPM: The Forward/Diffusion Process [1]



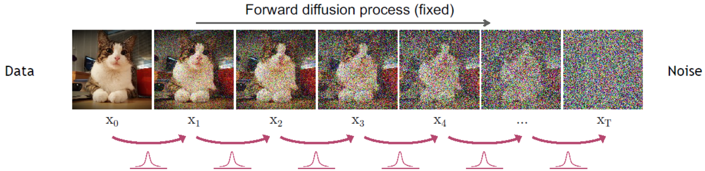
https://www.youtube.com/watch?v=cS6JQpEY9cs&t=12497s

$$
\begin{aligned}
\mathbf{x}_0 &\sim q(\mathbf{x}) && \text{(data distribution)} \\
\mathbf{x}_t &= \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon_t && \epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})
\end{aligned}
$$

- At each step, signal in input is suppressed and random noise $\sqrt{\beta_t}\epsilon_t$ is added.
- **Noise/variace schedule**: e.g., $\beta_1 = 10^{-4}$ linearly increases to $\beta_T = 10^{-2}$

[1] Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015
Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020
Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021

Denoising Diffusion Probabilistic Models (DDPM)



https://www.youtube.com/watch?v=cS6JQpEY9cs&t=12497s

$$\begin{aligned} \mathbf{x}_0 &\sim q(\mathbf{x}) && \text{(data distribution)} \\ \mathbf{x}_t &= \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon_t && \epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \end{aligned}$$

This process defines a joint distribution over $\mathbf{x}_0$ and latent variables $\mathbf{x}_{1:T}$:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}), \qquad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

$$q(\mathbf{x}_{0:T)} = q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)$$

DDPM: The Forward/Diffusion Process

$$\mathbf{x}_0 \sim q(\mathbf{x}), \qquad \mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon_t \qquad \epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\begin{aligned}
\mathbf{x}_2 &= \sqrt{1-\beta_2}\mathbf{x}_1 + \sqrt{\beta_2}\epsilon_2 \\
&= \sqrt{1-\beta_2}(\sqrt{1-\beta_1}\mathbf{x}_0 + \sqrt{\beta_1}\epsilon_1) + \sqrt{\beta_2}\epsilon_2 \\
&= \sqrt{(1-\beta_1)(1-\beta_2)}\mathbf{x}_0 + \sqrt{1-\beta_2}\sqrt{\beta_1}\epsilon_1 + \sqrt{\beta_2}\epsilon_2
\end{aligned}$$

The term in brown is a also a Gaussian variable with mean 0 and variance:

$$(1-\beta_2)\beta_1 + \beta_2 = 1 - (1-\beta_1)(1-\beta_2)$$

Hence,

$$\mathbf{x}_2 = \sqrt{(1-\beta_1)(1-\beta_2)}\mathbf{x}_0 + \sqrt{1-(1-\beta_1)(1-\beta_2)}\bar{\epsilon}_2, \qquad \bar{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Diffusion Kernel

$$\mathbf{x}_0 \sim q(\mathbf{x}), \qquad \mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon_t \qquad \epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

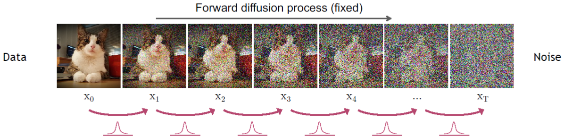
In general, define $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_t$. It can be approved that

$$\begin{aligned} \mathbf{x}_t &= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t & \bar{\epsilon}_t \sim \mathcal{N}(0, \mathbf{I}) \\ q(\mathbf{x}_t|\mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, \sqrt{1-\bar{\alpha}_t}\mathbf{I}) & \textbf{(diffusion kernel)} \end{aligned}$$

In t steps, the original signal in $\mathbf{x}_0$ is suppressed by a factor of $\sqrt{\bar{\alpha}_t}$, and the compound noise added is $\sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t \quad \bar{\epsilon}_t \sim \mathcal{N}(0, \mathbf{I})$.

- $\sqrt{1-\bar{\alpha}_t}$ is the total amount of Gaussian noise added to $\mathbf{x}_0$ to obtain $\mathbf{x}_t$.
- $\beta_t = 1 - \alpha_t$ is the amount of Gaussian noise added in step t.

Noise Schedule



The noise schedule is designed such that $\bar{\alpha}_T \approx 0$ and hence

$$q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_t; \mathbf{0}, \mathbf{I})$$

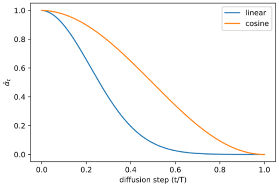


Fig. 5. Comparison of linear and cosine-based scheduling of β_t during training.
(Image source: Nichol & Dhariwal, 2021)

Noise Schedule

Cosine schedule

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \quad f(t) = \cos^2\left(\frac{\frac{t}{T}+s}{1+s}\cdot\frac{\pi}{2}\right), \quad \beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}$$

Linear schedule

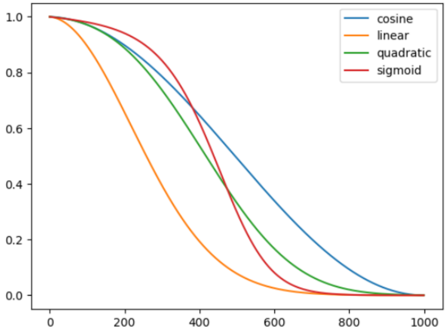
$$\beta_t = \beta_1 + \frac{t-1}{T-1}(\beta_T - \beta_1)$$

Quadratic schedule

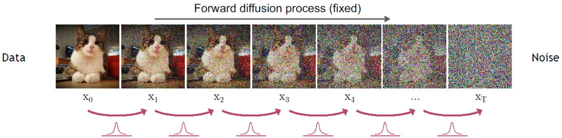
$$\beta_t = \left[\sqrt{\beta_1} + \frac{t-1}{T-1}\left(\sqrt{\beta_T} - \sqrt{\beta_1}\right)\right]^2$$

Sigmoid schedule

$$\beta_t = \beta_1 + \frac{\beta_T - \beta_1}{1 + e^{\tau\left(1-2\cdot\frac{t-1}{T-1}\right)}}$$



Ancestral Sampling



$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t \qquad \bar{\epsilon}_t \sim \mathcal{N}(0, \mathbf{I})$$

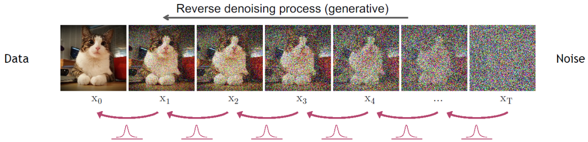
$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, \sqrt{1-\bar{\alpha}_t}\mathbf{I}) \qquad \textbf{(diffusion kernel)}$$

As $q(\mathbf{x}_t) = \int q(\mathbf{x}_t|\mathbf{x}_0)q(\mathbf{x}_0)d\mathbf{x}_0$, samples of $q(\mathbf{x}_t)$ can be obtained via ancestral sampling:

$$\mathbf{x}_0 \sim q(\mathbf{x}_0), \mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0)$$

This is important for training.

DDPM: The Reverse Process - Training and Sampling



https://www.youtube.com/watch?v=cS6JQpEY9cs&t=12497s

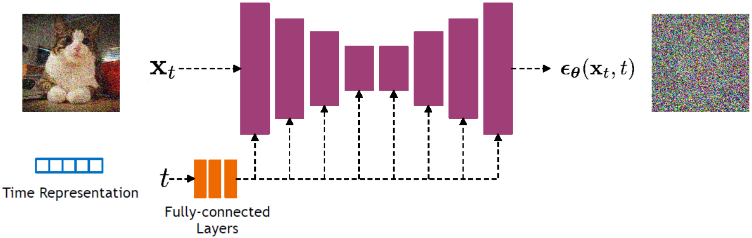
- During the reverse process, we try to undo the added noise at every time step.
- We start with the pure noise distribution (the last step of the forward process) and try to denoise the samples in the backward direction.
- Topics about the reverse process:
  1. Train a noise predictor.
  2. Use the noise predictor for denoising.

DDPM: Noise Predictor

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t \qquad \bar{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

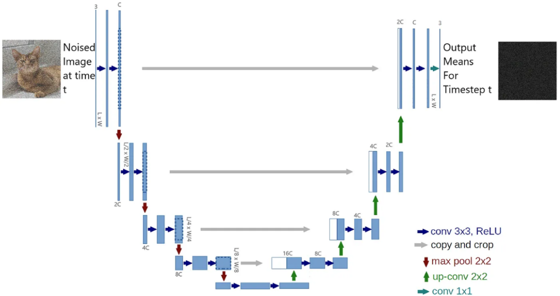
- Input: A noisy image $\mathbf{x}_t$, and embedding of time step t
- Output: An approximation $\epsilon_\theta(\mathbf{x}_t, t)$ of the compound noise $\bar{\epsilon}_t$ that was added to $\mathbf{x}_0$ to create $\mathbf{x}_t$.

Diffusion models often use U-Net architectures with ResNet blocks and self-attention layers to represent $\epsilon_\theta(\mathbf{x}_t, t)$



https://www.youtube.com/watch?v=cS6JQpEY9cs&t=12497s

DDPM: U-Net



https://betterprogramming.pub/diffusion-models-ddpms-ddims-and-classifier-free-guidance-e07b297b2869

- Input and output have the same size.
- Deeper layers extract more general features the deeper it goes.
- Skip connections reintroduce detailed features into the decoder.

DDPM: Training the Noise Predictor

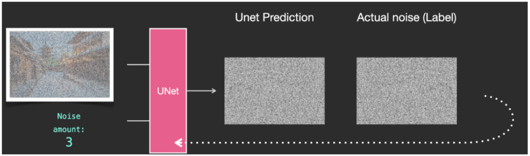
$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t \qquad \bar{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

- Input: A noisy image $\mathbf{x}_t$, and embedding of time step t
- Output: An approximation $\epsilon_\theta(\mathbf{x}_t, t)$ of the compound noise $\bar{\epsilon}_t$ that was added to $\mathbf{x}_0$ to create $\mathbf{x}_t$.

Training Objective: $\min ||\bar{\epsilon}_t - \epsilon_\theta(\mathbf{x}_t, t)||^2$

Algorithm 1 Training

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



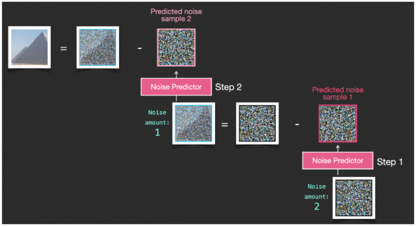
http://jalammar.github.io/illustrated-stable-diffusion/

DDPM: Sampling/Image Generation

- $\mathbf{x}_t$ is obtained from x_0 by adding compound noise $\bar{\epsilon}_t$.
- Now, we have an approximation of $\bar{\epsilon}_t$, i.e., $\epsilon_\theta(\mathbf{x}_t, t)$, we can do denoising to generate images

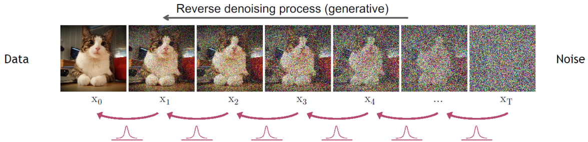
Algorithm 2 Sampling

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



http://jalammar.github.io/illustrated-stable-diffusion/

DDPM Theory: The Reverse Process



https://www.youtube.com/watch?v=cS6JQpEY9cs&t=12497s

$$
\begin{aligned}
\mathbf{x}_T &\sim p(\mathbf{x}_T) && \text{(Gaussian)} \\
\mathbf{x}_{t-1} &= \mu_\theta(\mathbf{x}_t, t) + \sigma_t \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) && \text{(often } \sigma_t^2 = \beta_t\text{)}
\end{aligned}
$$

$$
\begin{aligned}
p(\mathbf{x}_T) &= \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}) \\
p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) &= \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I})
\end{aligned}
$$

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

DDPM Theory: Training the Reverse Process

Assume the forward process $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ is fixed. (Not the data distribution $q(\mathbf{x}_0)$)
Minimizing the negative loglikelihood:

$$\mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0)}[-\log p_\theta(\mathbf{x}_0)]$$

Difficult directly. So minimizing the **variational upper bound**

$$\mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0)}[-\log p_\theta(\mathbf{x}_0)] \quad \leq \quad \mathbb{E}_{\mathbf{x}_0, \mathbf{x}_{1:T} \sim q(\mathbf{x}_{0:T}|\mathbf{x}_0)}[-\log \frac{p_\theta(\mathbf{x}_0, \mathbf{x}_{1:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}]$$

where $\mathbf{x}_{1:T}$ are the latent variables. To minimize the bound, we need to match p_θ with q.

DDPM Theory: Training the Reverse Process

It is observed that $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is Gaussian. Its mean is

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

where $\bar{\epsilon}_t$ is from $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t$ is fixed (not random only $\mathbf{x}_t$ is observed), and $\alpha_t = 1 - \beta_t$.

Making the mean $\mu_\theta(\mathbf{x}_t, t)$ of $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ close to $\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0)$ would help bring down the bound. (Proper weights for different time steps are needed, but are ignored here for simplicity)

Therefore, the mean $\mu_\theta(\mathbf{x}_t, t)$ of $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is parameterized as follows:

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

where $\epsilon_\theta(\mathbf{x}_t, t)$ is a network to predict the compound noise $\bar{\epsilon}_t$ added to $\mathbf{x}_0$ to reach $\mathbf{x}_t$

DDPM Theory: The Training Loss

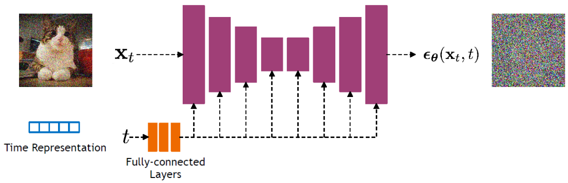
$$L = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), t \sim \mathcal{U}(1,T), \bar{\epsilon}_t \sim \mathcal{N}(\mathbf{0},\mathbf{I})} ||\bar{\epsilon}_t - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t, t)||^2$$

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

- $\bar{\epsilon}_t$ is the compound noise added to $\mathbf{x}_0$ to obtain $\mathbf{x}_t$.
- $\epsilon_\theta(\mathbf{x}_t, t)$ is a neural network that predicts the compound noise $\bar{\epsilon}_t$ added to $\mathbf{x}_0$ to obtain $\mathbf{x}_t$.

Aim to make $\epsilon_\theta(\mathbf{x}_t, t)$ close to $\bar{\epsilon}_t$

Diffusion models often use U-Net architectures with ResNet blocks and self-attention layers to represent $\epsilon_\theta(\mathbf{x}_t, t)$



DDPM Theory: Algorithm for Training and Sampling [2]

Algorithm 1 Training

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

Algorithm 2 Sampling

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

Training: Learning a network $\epsilon_\theta(\mathbf{x}_t, t)$ to predict the compound noise added to $\mathbf{x}_0$ to obtain $\mathbf{x}_t$

[2] Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020

DDPM Theory: Algorithm for Training and Sampling [3]

Algorithm 1 Training

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

Algorithm 2 Sampling

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

Image Generation/Sampling: The mean $\mu_\theta(\mathbf{x}_t, t)$ of $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is parameterized as follows:

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

Once we have $\epsilon_\theta(\mathbf{x}_t, t)$, we can sample $\mathbf{x}_{t-1}$ from $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$. It is denoising.

The data generation is slow because the error-prediction network $\epsilon_\theta(\mathbf{x}_t, t)$ is evaluated T (e.g., 1,000) times.

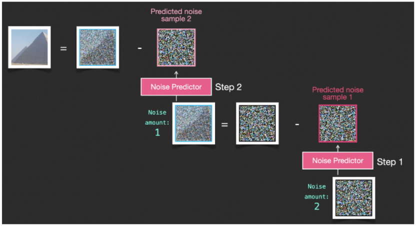
[3] Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020

Outline

DDIM Sampling

Algorithm 2 Sampling

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



http://jalammar.github.io/illustrated-stable-diffusion/

- The smampling process is slow because the error-prediction network $\epsilon_\theta(\mathbf{x}_t, t)$ is evaluated T (e.g., 1,000) times.
- Denoising Diffusion Implicit Models (DDIM) [4] does sampling at a subset of the time steps.

[4] Song et al., "Denoising Diffusion Implicit Models", ICLR 2021.

DDIM Sampling [5]

Recall $\mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\bar{\epsilon}_{t-1}$. DDIM sampling $\mathbf{x}_{t-1}$ as follows:

$$\mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\,\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\,\hat{\epsilon}_t + \sigma_t \mathbf{z} \qquad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\epsilon(x_t, t)}{\sqrt{\bar{\alpha}_t}}$$

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

$\hat{\mathbf{x}}_0$ and $\hat{\epsilon}_t$ are estimations of $\mathbf{x}_0$ and $\bar{\epsilon}_t$ from

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

$\hat{\epsilon}_t$ is used as an estimation of $\bar{\epsilon}_{t-1}$ in DDIM sampling.

[5] Song et al., "Denoising Diffusion Implicit Models", ICLR 2021.

DDIM Sampling

$$\mathbf{x}_{t-1} \quad = \quad \sqrt{\bar{\alpha}_{t-1}}\,\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\,\hat{\epsilon}_t + \sigma_t \mathbf{z} \qquad\qquad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

If $\sigma_t = 0$, DDIM is deterministic.

DDIM sampling can be accelerated by carrying it out at a subset of time points:

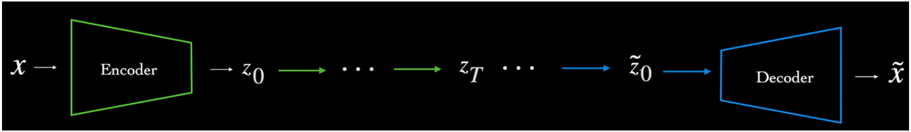
$$\tau_1 = 1 < \tau_2 < \ldots < \tau_t = T$$

$$\mathbf{x}_{\tau_{i-1}} \quad = \quad \sqrt{\bar{\alpha}_{\tau_{i-1}}}\,\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_{\tau_{i-1}}-\sigma_{\tau_i}^2}\,\hat{\epsilon}_{\tau_i} + \sigma_{\tau_i} \mathbf{z} \qquad\qquad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Please refer to Song *et al.* for theoretical justification of DDIM sampling.

Latent Diffusion Models (LDM) [6]

Latent Diffusion Models (LDM): A method to make DDPM more efficient.



Phase 1: Pre-train a VAE to map data to a latent space, e.g., from $3 \times 512 \times 512$ to $6 \times 64 \times 64$.

Phase 2: Learn a diffusion model for data in the latent space.

$$L_{LDM} = \mathbb{E}_{\mathcal{E}(\mathbf{x}),t,\epsilon}||\epsilon - \epsilon_\theta(\mathbf{z}_t, t)||^2$$

The latent space is of lower dimension than the data space.
Training and sampling with a diffusion model in the latent space is more efficient.

[6] Rombach, et al. High-resolution image synthesis with latent diffusion models, CVPR 2022.

Outline

Stable Diffusion

- Stable Diffusion is a text-to-image model released in 2022 by the start-up company Stability AI.
  - https://stability.ai/blog/stable-diffusion-v2-release
  - https://stability.ai/blog/stablediffusion2-1-release7-dec-2022
  - https://huggingface.co/spaces/stabilityai/stable-diffusion
- The following slides are based on
  - Jay Alammar, The Illustrated Stable Diffusion, http://jalammar.github.io/illustrated-stable-diffusion/.
  - Jarosław Kochanowicz et al., Diffusion models in practice, https://deepsense.ai/diffusion-models-in-practice-part-1-the-tools-of-the-trade/

Ways to Use Stable Diffusion

- Generate images from texts:

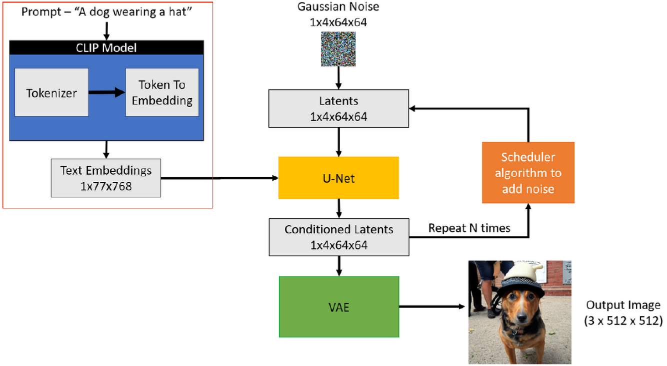
http://jalammar.github.io/illustrated-stable-diffusion/

- Alter images using text prompts:

http://jalammar.github.io/illustrated-stable-diffusion/

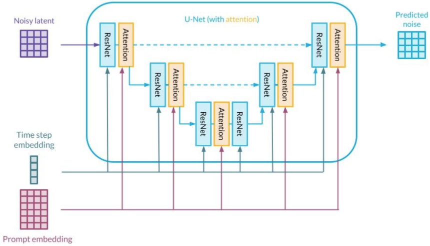
Stable Diffusion Architecture

- Left: A text encoder converts a text prompt y into a vector $\tau(y)$
- Right: The text embedding $\tau(y)$ is used to guide image generation in LDM



Text Guidance

- $\tau(y)$ is injected to the error predictor $\epsilon_\theta(\mathbf{z}_t, t, y)$ via cross-attention.



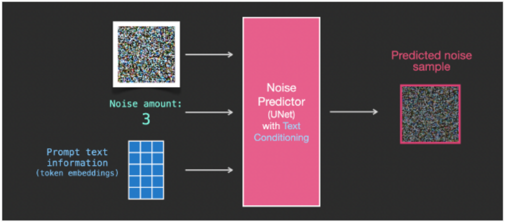
http://jalammar.github.io/illustrated-stable-diffusion/

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t \qquad \bar{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Training Objective: $\min ||\bar{\epsilon}_t - \epsilon_\theta(\mathbf{z}_t, t, y)||^2$

Text Guidance

- At each step of the generation process, the text embedding serves as a reminder of the kind of image we desire.
- For example, if you have an input prompt "cat", you can think of conditioning as telling the noise predictor:
  - "For the next denoising step, the image should look more like a cat. Now go on with the next step."



http://jalammar.github.io/illustrated-stable-diffusion/

Stable Diffusion is Based on LDM

- Stable Diffusion was trained on pairs of images and captions taken from LAION-5B, a publicly available dataset consisting of 5 billion image-text pairs.
  - For each text-image pair, the image is first mapped to the latent space, and then noise is added to it repeatedly to finally get $\mathbf{z}_T$
  - Various versions z_t of the image are used to train the noise predictor $\epsilon_\theta(\mathbf{z}_t, t, y)$ to denoise z_t by taking y into consideration.

    If z_t is a blurred version of a cat, it should be denoised in one way
    If z'_t is a blurred version of a dog it should be denoised in another

- The training was carried out on 256 Nvidia A100 GPUs for a total of 150,000 GPU-hours, at a cost of $600,000.

Stable Diffusion

Stable Diffusion 2.1 Demo:

https://huggingface.co/spaces/stabilityai/stable-diffusion