All About Diffusion

Seunghan Lee

Department of Statistics and Data Science, Yonsei University

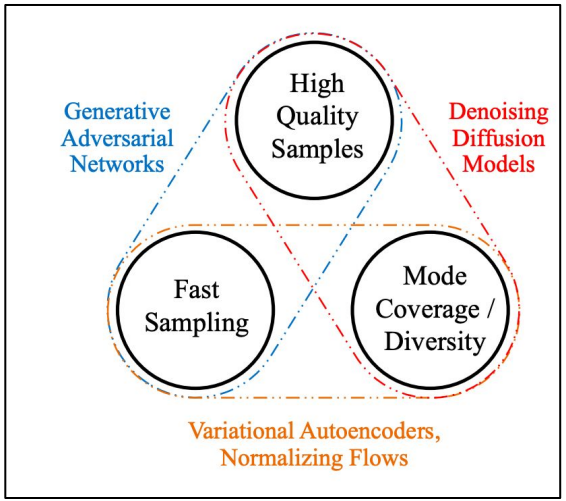
Outlines

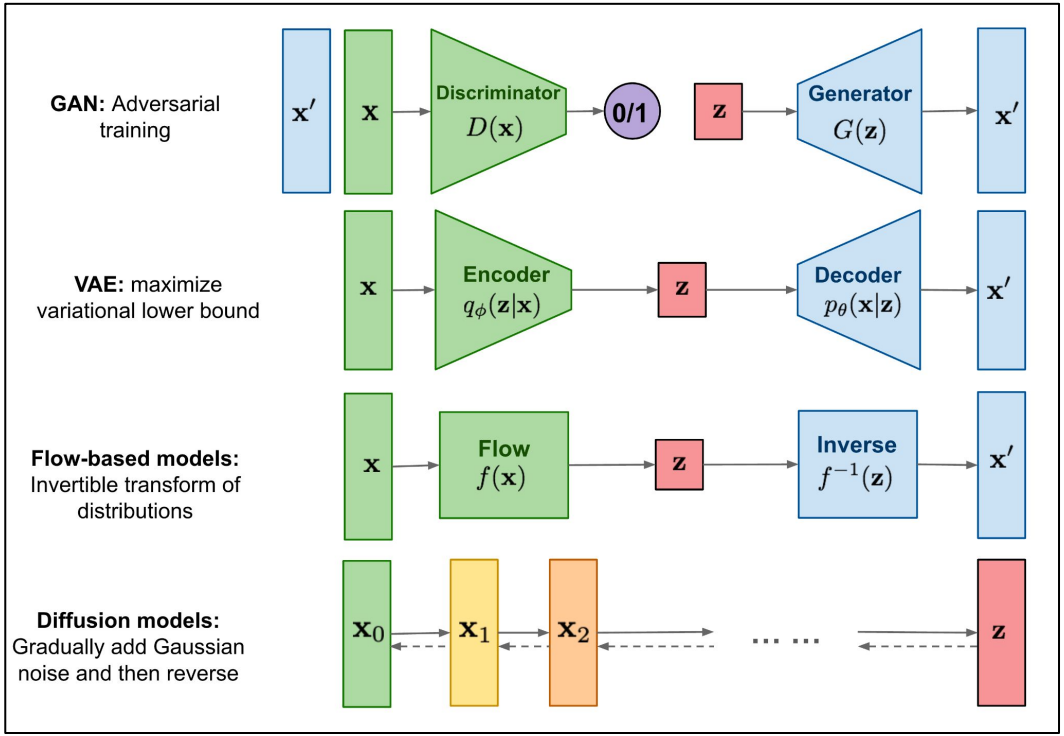
1. Generative Models
2. Diffusion Model
3. DDPM
4. Conditional Diffusion Model
5. Latent Diffusion Model
6. Application
    a. Text-to-Image
    b. Natural Language Generation
    c. Time Series Forecasting
    d. Time Series Imputation
7. Code Implementation

1. Generative Models

Generative Models

- (1) GAN
- (2) Diffusion
- (3) VAE, Normalizing Flows





Generative Models

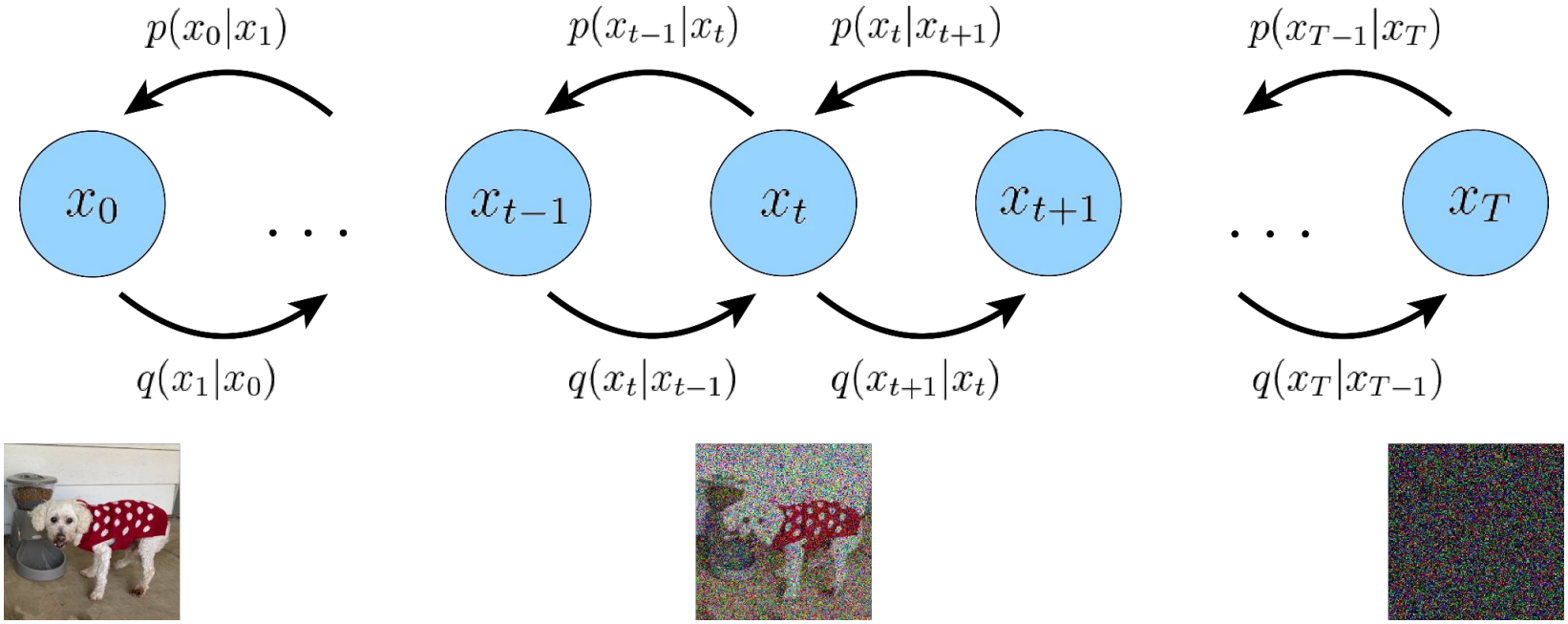
- (1) **GAN**
- (2) Diffusion
- (3) **VAE**, Normalizing Flows

$$\min_G \max_D L(D,G) = \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$$
$$= \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{x\sim p_g(x)}[\log(1-D(x)]$$
$$L(G,D^*) = 2D_{JS}(p_r \| p_g) - 2\log 2$$

$$L_{\text{VAE}}(\theta,\phi) = -\log p_\theta(\mathbf{x}) + D_{\text{KL}}(q_\phi(\mathbf{z}\mid\mathbf{x}) \| p_\theta(\mathbf{z}\mid\mathbf{x}))$$
$$= -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}\mid\mathbf{z}) + D_{\text{KL}}(q_\phi(\mathbf{z}\mid\mathbf{x}) \| p_\theta(\mathbf{z}))$$
$$\theta^*, \phi^* = \arg\min_{\theta,\phi} L_{\text{VAE}}$$
$$-L_{\text{VAE}} = \log p_\theta(\mathbf{x}) - D_{\text{KL}}(q_\phi(\mathbf{z}\mid\mathbf{x}) \| p_\theta(\mathbf{z}\mid\mathbf{x})) \le \log p_\theta(\mathbf{x})$$

2. Diffusion Model

Diffusion Model



Diffusion Model

Forward Process: add noise

$$q\left(\mathrm{x}_t \mid \mathrm{x}_{t-1}\right) = N\left(\mathrm{x}_t; \sqrt{1-\beta_t}\mathrm{x}_{t-1}, \beta_t \mathrm{I}\right)$$

Backward Process: remove noise

$$p_\theta\left(\mathrm{x}_{t-1} \mid \mathrm{x}_t\right) = N\left(\mathrm{x}_{t-1}; \mu_\theta\left(\mathrm{x}_t, t\right), \Sigma_\theta\left(\mathrm{x}_t, t\right)\right)$$

DDPM: $T = 1000, \beta_0 = 0.0001, \beta_{1000} = 0.02$

Diffusion Model

Forward Process: add noise

$$\alpha_t = 1 - \beta_t$$

$$\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$$

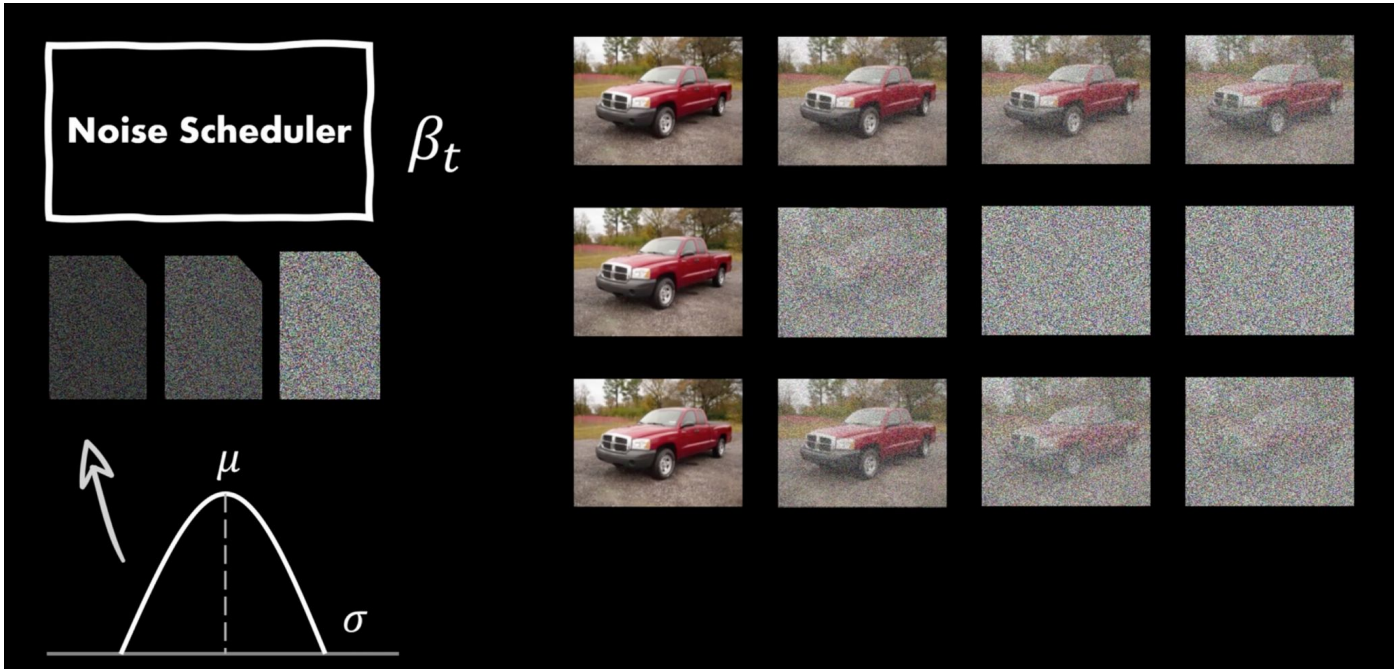
$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\epsilon}_{t-2} \\
&= \cdots \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon
\end{aligned}$$

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

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

Diffusion Model

Noise Scheduler



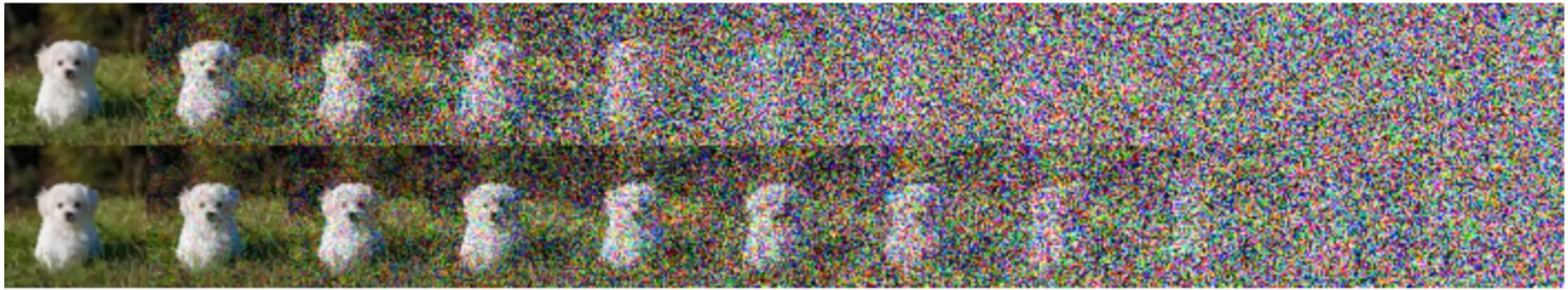
Too small

Too big

Diffusion Model

Noise Scheduler

Use “Cosine Scheduler”!



Linear (top) vs Cosine (bottom)

Diffusion Model

Loss Function

VAE $$-L_{\text{VAE}} = \log p_\theta(\mathbf{x}) - D_{\text{KL}}(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z} \mid \mathbf{x})) \leq \log p_\theta(\mathbf{x})$$

Diffusion

$$-\log p_\theta(\mathbf{x}_0) \leq \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\right] = L_{VLB}$$

$$L_{\text{VLB}} = L_T + L_{T-1} + \cdots + L_0$$

$$L_T = D_{\text{KL}}(q(\mathbf{x}_T \mid \mathbf{x}_0) \| p_\theta(\mathbf{x}_T))$$

$$L_t = D_{\text{KL}}(q(\mathbf{x}_t \mid \mathbf{x}_{t+1}, \mathbf{x}_0) \| p_\theta(\mathbf{x}_t \mid \mathbf{x}_{t+1})) \text{ for } 1 \leq t \leq T-1$$

$$L_0 = -\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)$$

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

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$$

Diffusion Model

$$L_t = D_{\mathrm{KL}}\big(q(\mathbf{x}_t \mid \mathbf{x}_{t+1}, \mathbf{x}_0) \| p_\theta(\mathbf{x}_t \mid \mathbf{x}_{t+1})\big)$$

Reverse Process (Ground truth)

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

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

$$\begin{aligned} L_{t-1} &= \mathbb{E}_{\mathbf{x}_0,\epsilon}\left[\frac{1}{2\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\tilde{\mu}(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, , t)\|_2^2\right] \\ &= \mathbb{E}_{\mathbf{x}_0,\epsilon}\left[\frac{1}{2\|\boldsymbol{\Sigma}_\theta\|_2^2}\left\|\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right) - \mu_\theta(\mathbf{x}_t, , t)\right\|_2^2\right] \end{aligned}$$

3. DDPM

DDPM (Denoising Diffusion Probabilistic Models, 2020)

Difference btw naive diffusion & DDPM

- (1) Do NOT train the **“variance term”**
- (2) Predict the **“noise”**, not the “entire mean”

DDPM (Denoising Diffusion Probabilistic Models, 2020)

Loss Function

$$\min . \quad \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p\left(\mathbf{x}_T\right)\right)}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)}_{L_{t-1}} \underbrace{-\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}_{L_0}]$$

(1) Prior Matching term

- Make it close to prior
- Treat it as a constant

($p\left(\mathbf{x}_T\right)$: Gaussian)

(2) Denoising term

(3) Reconstruction term

- Minor .. Ignore!

DDPM (Denoising Diffusion Probabilistic Models, 2020)

Loss Function

$$\min . \quad \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p\left(\mathbf{x}_T\right)\right)}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)}_{L_{t-1}} \underbrace{-\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}_{L_0}]$$

(1) Prior Matching term **(2) Denoising term** **(3) Reconstruction term**

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

$$q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) = q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right) \frac{q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_0\right)}{q\left(\mathbf{x}_t \mid \mathbf{x}_0\right)}$$

$$\propto \exp\left(-\frac{1}{2}\left(\frac{\left(\mathbf{x}_t - \sqrt{\alpha_t}\mathbf{x}_{t-1}\right)^2}{\beta_t} + \frac{\left(\mathbf{x}_{t-1} - \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0\right)^2}{1-\bar{\alpha}_{t-1}} - \frac{\left(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0\right)^2}{1-\bar{\alpha}_t}\right)\right)$$

$$= \exp\left(-\frac{1}{2}\left(\frac{\mathbf{x}_t^2 - 2\sqrt{\alpha_t}\mathbf{x}_t\mathbf{x}_{t-1} + \alpha_t\mathbf{x}_{t-1}^2}{\beta_t} + \frac{\mathbf{x}_{t-1}^2 - 2\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0\mathbf{x}_{t-1} + \bar{\alpha}_{t-1}\mathbf{x}_0^2}{1-\bar{\alpha}_{t-1}} - \frac{\left(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0\right)^2}{1-\bar{\alpha}_t}\right)\right)$$

$$= \exp\left(-\frac{1}{2}\left(\left(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}}\right)\mathbf{x}_{t-1}^2 - \left(\frac{2\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{2\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\mathbf{x}_0\right)\mathbf{x}_{t-1} + C\left(\mathbf{x}_t, \mathbf{x}_0\right)\right)\right)$$

DDPM (Denoising Diffusion Probabilistic Models, 2020)

Loss Function

$$\min . \quad \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p\left(\mathbf{x}_T\right)\right)}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)}_{L_{t-1}} \underbrace{-\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}_{L_0}]$$

(1) Prior Matching term **(2) Denoising term** **(3) Reconstruction term**

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

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

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

DDPM (Denoising Diffusion Probabilistic Models, 2020)

Loss Function

$$\min . \quad \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p\left(\mathbf{x}_T\right)\right)}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)}_{L_{t-1}} \underbrace{-\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}_{L_0}]$$

(1) Prior Matching term **(2) Denoising term** **(3) Reconstruction term**

$$\mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}}\left[\frac{\beta_t^2}{2\tilde{\beta}_t\left(1-\bar{\alpha}_t\right)}\left\|\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0+\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t\right)\right\|^2\right]$$

$$\min . \left\|\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_\theta\left(\mathbf{x}_t, t\right)\right\|^2 = \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0+\sqrt{\left(1-\bar{\alpha}_t\right)}\boldsymbol{\epsilon}, t\right\|^2\right.\right.$$

DDPM (Denoising Diffusion Probabilistic Models, 2020)

Algorithm 1 Training

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

$$q(x_t|x_0) = N(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathrm{I})$$
$$\rightarrow x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$$

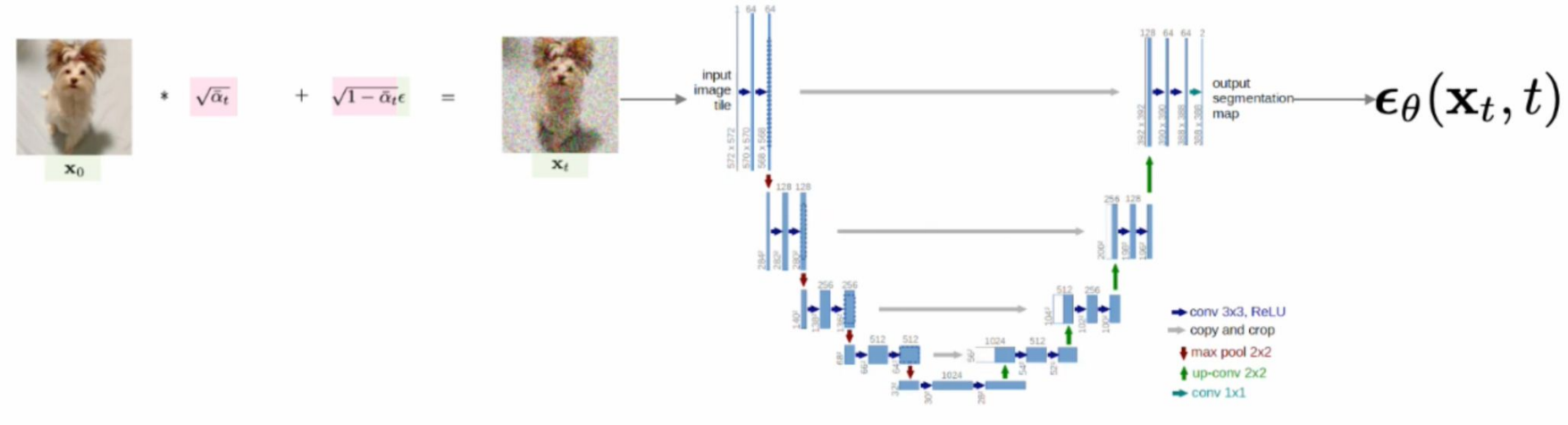
Algorithm 2 Sampling Inference

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

DDPM (Denoising Diffusion Probabilistic Models, 2020)

Model: U-Net (input dim == output dim)

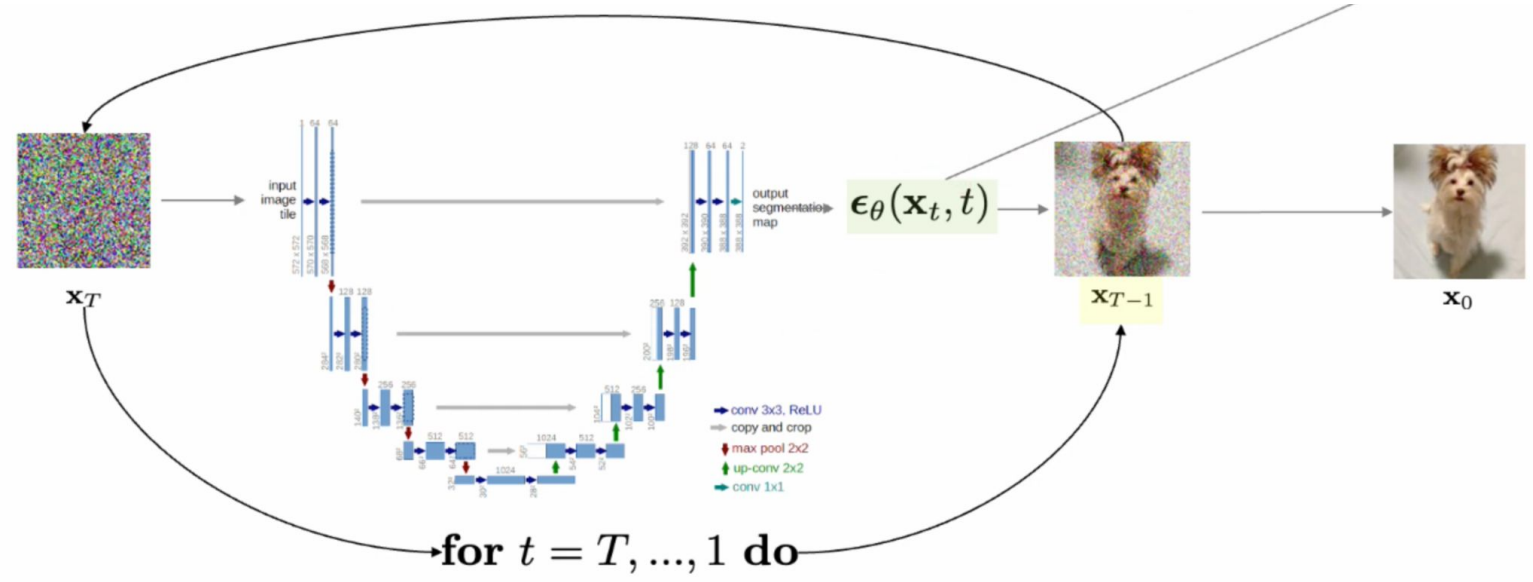
Training



DDPM (Denoising Diffusion Probabilistic Models, 2020)

Model: U-Net (input dim == output dim)

Inference



4. Conditional Diffusion Model

Conditional Diffusion Model

(1) **Classifier guidance**: with **diffusion model + classifier**

$$\hat{\epsilon}\left(x_t, t\right) = \epsilon_\theta\left(x_t, t\right) - \sqrt{1-\bar{\alpha}_t} w \nabla_{x_t} \log f_\phi\left(y \mid x_t\right)$$

(= Predicted Noise)

(2) **Classifier-free guidance (CFG)**: with only **diffusion model**

$$\begin{aligned}\hat{\epsilon}\left(x_t, t, y\right) &= (1+w)\epsilon_\theta\left(x_t, t, y\right) - w\epsilon_\theta\left(x_t, t\right) \\ &= w\left(\epsilon_\theta\left(x_t, t, y\right) - \epsilon_\theta\left(x_t, t\right)\right) + \epsilon_\theta\left(x_t, t, y\right)\end{aligned}$$

w: Guidance scale

$$\epsilon_\theta\left(x_t, t\right) = \epsilon_\theta\left(x_t, t, y = \emptyset\right)$$

Conditional Diffusion Model

(1) **Classifier guidance**: with **diffusion model + classifier**

$$\hat{\epsilon}\left(x_t, t\right) = \epsilon_\theta\left(x_t, t\right) - \sqrt{1-\bar{\alpha}_t} w \nabla_{x_t} \log f_\phi\left(y \mid x_t\right)$$

$$\nabla \log p\left(\mathbf{x}_t\right) = -\frac{1}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta\left(\mathbf{x}_t\right)$$

$$\begin{aligned}
\nabla \log p\left(\mathbf{x}_t \mid y\right) &= \nabla \log \left(\frac{p\left(\mathbf{x}_t\right) p\left(y \mid \mathbf{x}_t\right)}{p(y)} \right) \\
&= \nabla \log p\left(\mathbf{x}_t\right) + \nabla \log p\left(y \mid \mathbf{x}_t\right) - \nabla \log p(y) \\
&= \underbrace{\nabla \log p\left(\mathbf{x}_t\right)}_{\text{unconditional score}} + \gamma \underbrace{\nabla \log p\left(y \mid \mathbf{x}_t\right)}_{\text{adversarial gradient}}
\end{aligned}$$

$$\Rightarrow \quad \hat{\epsilon}\left(\mathbf{x}_t\right) = \boldsymbol{\epsilon}_\theta\left(\mathbf{x}_t\right) - \gamma \sqrt{1-\bar{\alpha}_t} \nabla \log p\left(y \mid \mathbf{x}_t\right)$$

Conditional Diffusion Model

(1) **Classifier guidance**: with **diffusion model + classifier**

$$\hat{\epsilon}\left(x_t, t\right) = \epsilon_\theta\left(x_t, t\right) - \sqrt{1-\bar{\alpha}_t} w \nabla_{x_t} \log f_\phi\left(y \mid x_t\right)$$

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

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

Inference (Sampling)

Conditional Diffusion Model

(2) **Classifier-free guidance (CFG)**: with only **diffusion model**

$$\begin{aligned}\hat{\epsilon}\left(x_t, t, y\right) &= (1+w)\epsilon_\theta\left(x_t, t, y\right) - w\epsilon_\theta\left(x_t, t\right) \\ &= w\left(\epsilon_\theta\left(x_t, t, y\right) - \epsilon_\theta\left(x_t, t\right)\right) + \epsilon_\theta\left(x_t, t, y\right)\end{aligned}$$

$$\nabla \log p\left(\mathbf{x}_t \mid y\right) = \underbrace{\nabla \log p\left(\mathbf{x}_t\right)}_{\text{unconditional score}} + \gamma \underbrace{\nabla \log p\left(y \mid \mathbf{x}_t\right)}_{\text{adversarial gradient}}$$

$$\gamma \underbrace{\nabla \log p\left(y \mid \mathbf{x}_t\right)}_{\text{adversarial gradient}} = \nabla \log p\left(\mathbf{x}_t \mid y\right) - \underbrace{\nabla \log p\left(\mathbf{x}_t\right)}_{\text{unconditional score}}$$

$$\begin{aligned}\nabla \log p\left(\mathbf{x}_t \mid y\right) &= \nabla \log p\left(\mathbf{x}_t\right) + \gamma\left(\nabla \log p\left(\mathbf{x}_t \mid y\right) - \nabla \log p\left(\mathbf{x}_t\right)\right) \\ &= \nabla \log p\left(\mathbf{x}_t\right) + \gamma \nabla \log p\left(\mathbf{x}_t \mid y\right) - \gamma \nabla \log p\left(\mathbf{x}_t\right) \\ &= \underbrace{\gamma \nabla \log p\left(\mathbf{x}_t \mid y\right)}_{\text{conditional score}} + \underbrace{(1-\gamma) \nabla \log p\left(\mathbf{x}_t\right)}_{\text{unconditional score}}\end{aligned}$$

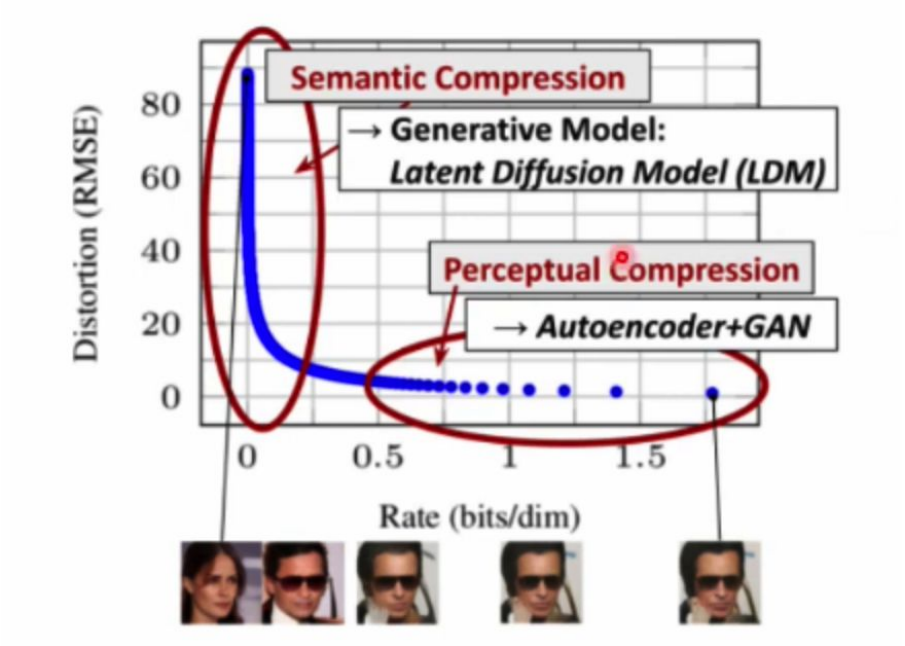
5. Latent Diffusion Model

https://arxiv.org/abs/2112.10752

Latent Diffusion Model

Perceptual & Semantic Compression

- Perceptual Compression:
  - loose HIGH-frequency details
- Semantic Compression:
  - learn semantic abstractly



https://arxiv.org/abs/2112.10752

Latent Diffusion Model

Abstract

- Paper: **High-Resolution Image Synthesis with Latent Diffusion Models**
- Limitation of diffusion: TOO LONG time in the “Perceptual Compression”
  - Naive diffusion model: Pixel Space Model
- **Stable Diffusion: Diffusion + AE**
  - Diffusion process in the LATENT space
  - Use AE for perceptual compression

    (Diffusion task for semantic compression)
- Save computational cost!

https://arxiv.org/abs/2112.10752

Latent Diffusion Model

Overview

- U-Net + Cross-attention

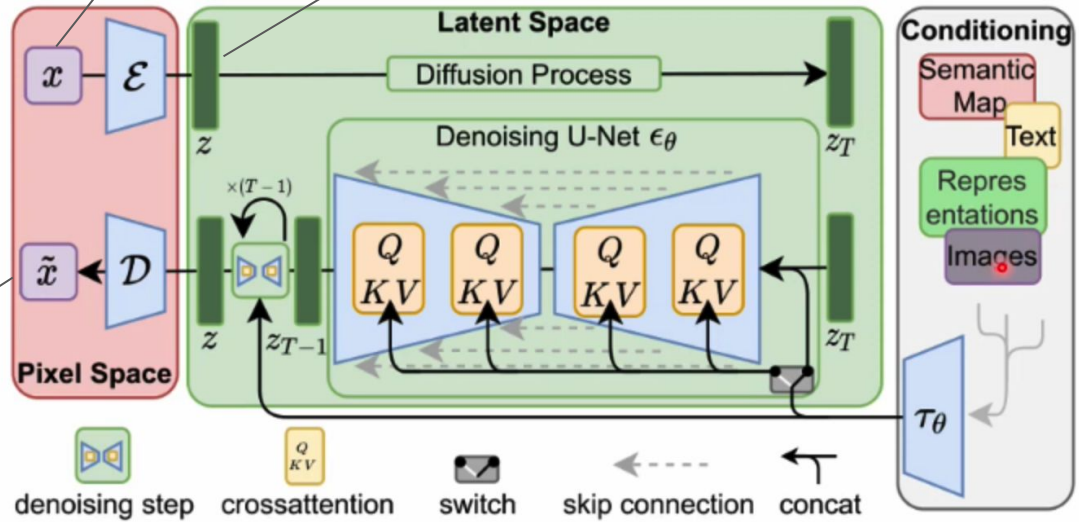
  (for conditioning)

- Text Encoder: CLIP

$x \in \mathbb{R}^{H \times W \times 3}$

$z = \mathcal{E}(x) \in \mathbb{R}^{h \times w \times c}$

Query: z_t
Key: $\tau_\theta(y)$
Value: $\tau_\theta(y)$



$\tilde{x} = \mathcal{D}(z) = \mathcal{D}(\mathcal{E}(x))$

$$L_{LDM} := \mathbb{E}_{\mathcal{E}(x),\epsilon\sim\mathcal{N}(0,1),t}\left[\|\epsilon - \epsilon_\theta(z_t, t, \tau_\theta(y))\|_2^2\right]$$

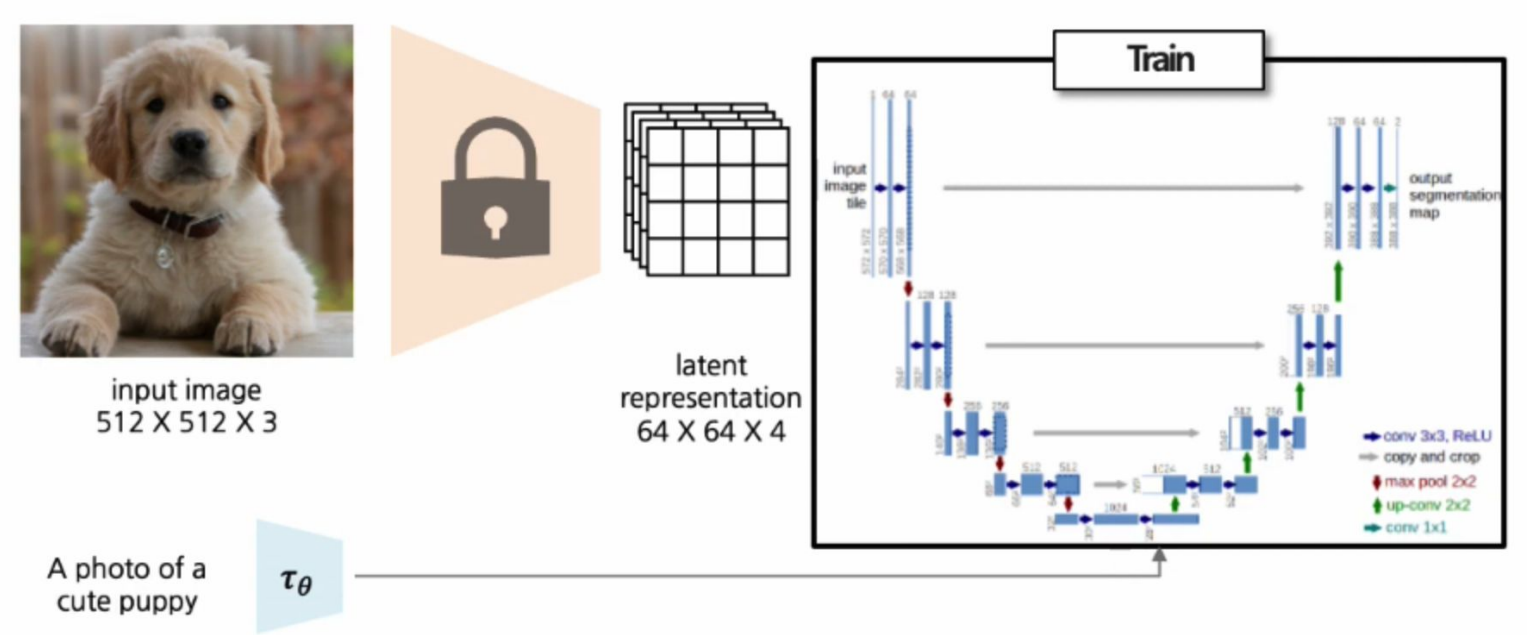
https://arxiv.org/abs/2112.10752

Latent Diffusion Model

Other conditions can be used!

(i.e. text, semantic maps, super resolution)

Training



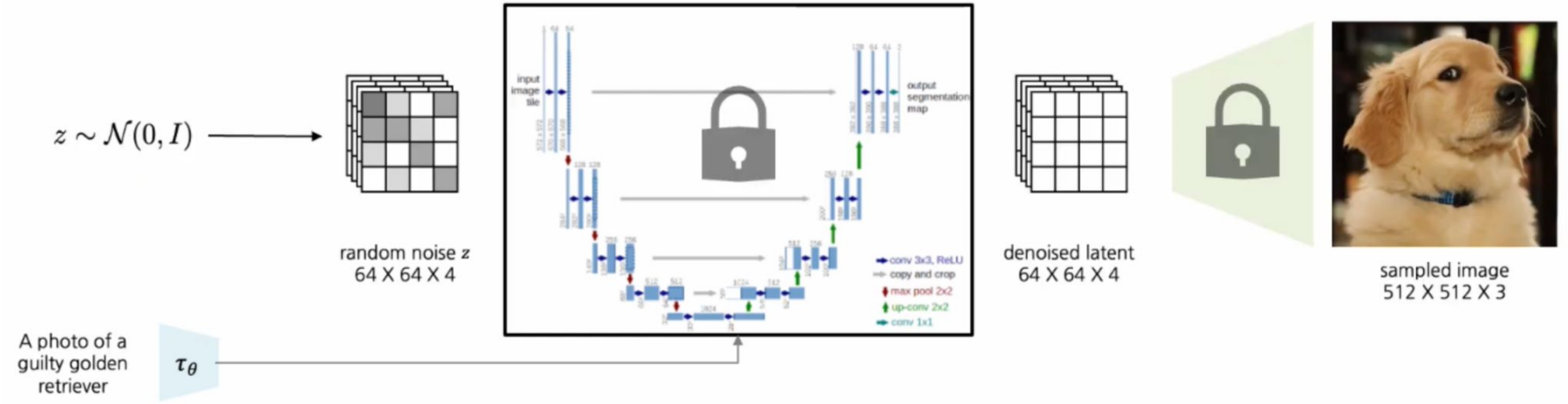
https://arxiv.org/abs/2112.10752

Latent Diffusion Model

Other conditions can be used!

(i.e. text, semantic maps, super resolution)

Inference



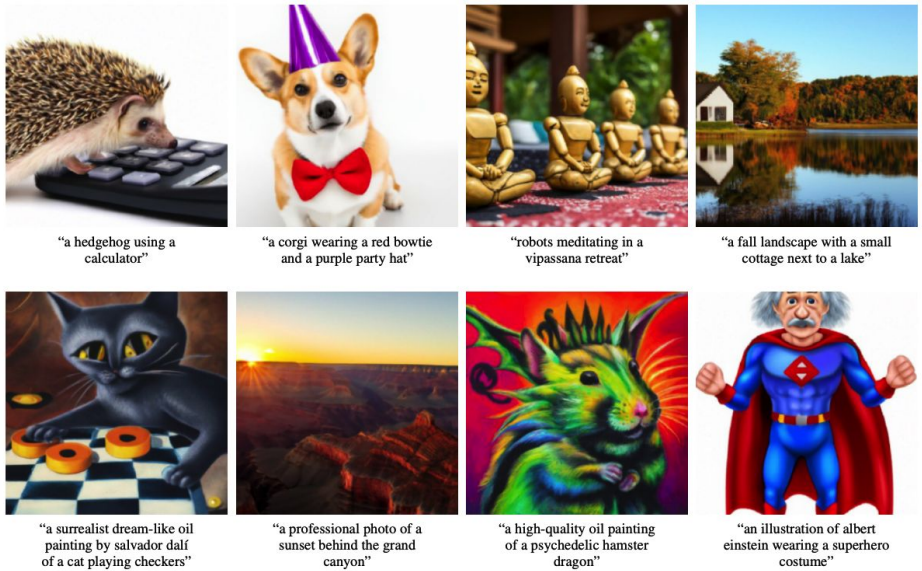
6. Application

https://arxiv.org/abs/2112.10741

Application (1): Text-to-Image

GLIDE (Guided Language to Image Diffusion for Generation and Editing)

- **GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models**
- Conditional information: **Text**



Application (1): Text-to-Image

GLIDE (Guided Language to Image Diffusion for Generation and Editing)

- Use 2 types of guidances
  - (1) Classifier-free guidance

$$\hat{\epsilon}\left(x_t, t, y\right) = (1+w)\epsilon_\theta\left(x_t, t, y\right) - w\epsilon_\theta\left(x_t, t\right)$$

  - (2) CLIP guidance

$$\hat{\epsilon}\left(x_t, t\right) = \epsilon_\theta\left(x_t, t\right) - \sqrt{1-\bar{\alpha}_t}\nabla_{x_t}\left(f\left(x_t\right) \cdot g(y)\right)$$

Image encoder Text encoder

https://arxiv.org/abs/2210.08933

Application (2): Natural Language Generation

DiffuSeq

- **DiffuSeq: Sequence to Sequence Text Generation with Diffusion Models**
- Apply diffusion on discrete text data
- Use seq2seq architecture
- Classifier-free guidance

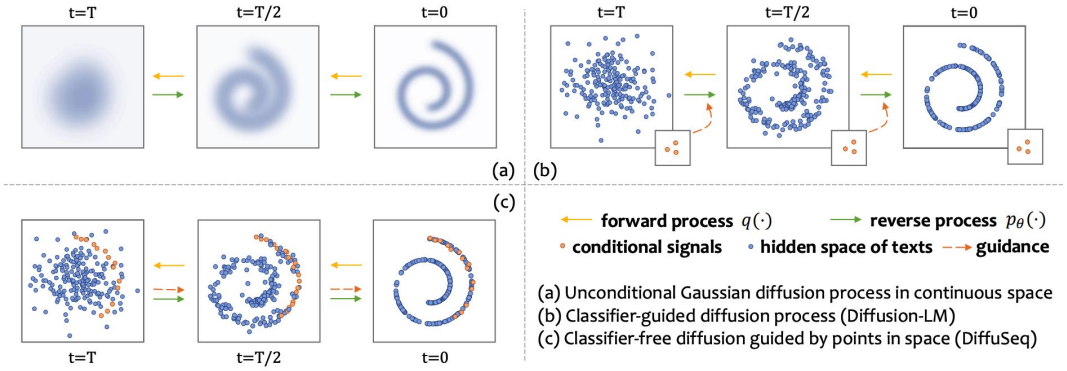


Figure 1: The demonstration of unconditional, classifier-guided, and classifier-free diffusion models.

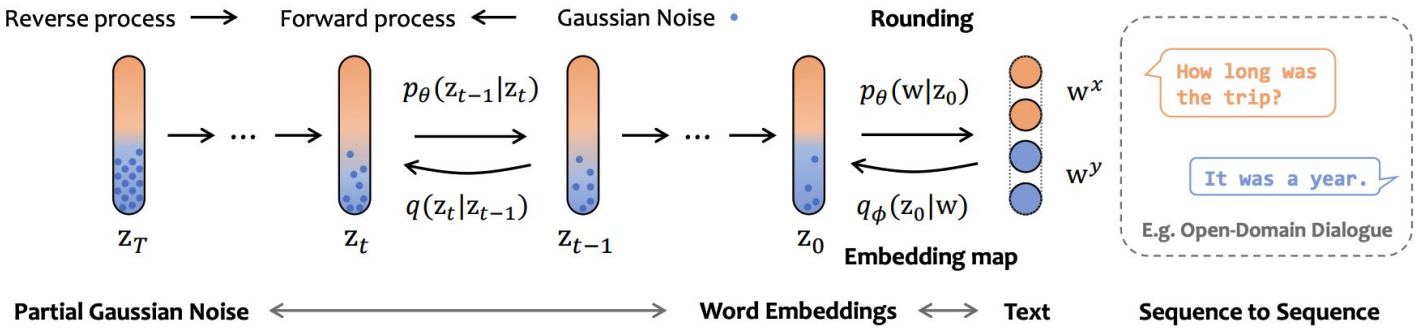
https://arxiv.org/abs/2210.08933

Application (2): Natural Language Generation

DiffuSeq

Rounding: mapping back to the discrete space (= text)

- Discrete => Continuous
  - Use embedding function (DiffusionLM (Li et al., 2022))
- Forward process: “partial” noising (noise ONLY on target w^x)
- Reverse process: “partial” denoising (use conditional signal w^y as guidance)



https://arxiv.org/abs/2210.08933

Application (2): Natural Language Generation

DiffuSeq

- Training
  - High diversity in NLP datasets & long diffusion steps => insufficient training
    - hypothesize: uniform sampling (of step t) is the problem!
  - Solution: Importance Sampling

$$\mathcal{L}_{\text{VLB}} = \mathbb{E}_{t\sim p_t}\left[\frac{\mathcal{L}_t}{p_t}\right],\ p_t \propto \sqrt{\mathbb{E}[\mathcal{L}_t^2]},\ \sum_{t=0}^{T-1} p_t = 1.$$

https://arxiv.org/abs/2210.08933

Application (2): Natural Language Generation

DiffuSeq

- Inference
  - Input: $\mathbf{z}_T$(= concatenate (a) & (b))
    - (a) Condition: $\mathrm{EMB}\left(\mathbf{w}^x\right)$
    - (b) Random Noise: $y_T \sim \mathcal{N}(0, I)$
  - Repeat reverse process: $\mathbf{z}_T \rightarrow \mathbf{z}_0$
  - Anchoring function
    - (1) rounding on $\mathbf{z}_t$
    - (2) replaces the part of recovered $\mathbf{z}_{t-1}$ that belongs to $\mathbf{w}^x$ with the original $\mathbf{x}_0$

https://arxiv.org/pdf/2101.12072.pdf

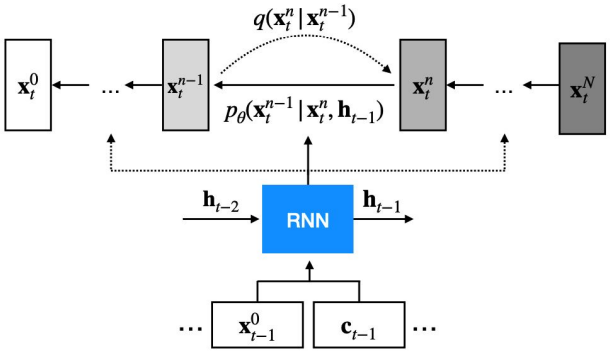
Application (3): Time Series Forecasting

TimeGrad

- **Autoregressive Denoising Diffusion Models for Multivariate Probabilistic Time Series Forecasting (2021)**

$$q_x\left(x^0_{t_0:T} \mid x^0_{1:t_0-1}, c_{1:T}\right) = \prod_{t=t_0}^{T} q_x\left(x^0_t \mid x^0_{1:t-1}, c_{1:T}\right)$$

Covariate



$$= \prod_{t=t_0}^{T} p_\theta\left(x^0_t \mid h_{t-1}\right)$$

where $\mathbf{h}_t = \mathrm{RNN}_\theta\left(\mathrm{concat}\left(\mathbf{x}^0_t, \mathbf{c}_t\right), \mathbf{h}_{t-1}\right)$

https://arxiv.org/pdf/2101.12072.pdf

Application (3): Time Series Forecasting

TimeGrad

Training

Algorithm 1 Training for each time series step $t \in [t_0, T]$

Input: data $\mathbf{x}_t^0 \sim q_{\mathcal{X}}(\mathbf{x}_t^0)$ and state $\mathbf{h}_{t-1}$
repeat
  Initialize $n \sim \text{Uniform}(1, \dots, N)$ and $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient step on

$$\nabla_\theta \|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_n}\mathbf{x}_t^0 + \sqrt{1-\bar{\alpha}_n}\epsilon, \mathbf{h}_{t-1}, n)\|^2$$

until converged

Inference

Algorithm 2 Sampling $\mathbf{x}_t^0$ via annealed Langevin dynamics

Input: noise $\mathbf{x}_t^N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and state $\mathbf{h}_{t-1}$
for $n = N$ **to** 1 **do**
  if $n > 1$ **then**
    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  else
    $\mathbf{z} = \mathbf{0}$
  end if
  $\mathbf{x}_t^{n-1} = \frac{1}{\sqrt{\alpha_n}}(\mathbf{x}_t^n - \frac{\beta_n}{\sqrt{1-\bar{\alpha}_n}}\epsilon_\theta(\mathbf{x}_t^n, \mathbf{h}_{t-1}, n)) + \sqrt{\Sigma_\theta}\mathbf{z}$
end for
Return: $\mathbf{x}_t^0$

https://arxiv.org/abs/2107.03502

Application (4): Time Series Imputation

CSDI

- **CSDI: Conditional Score-based Diffusion Models for Probabilistic Time Series Imputation (2021)**

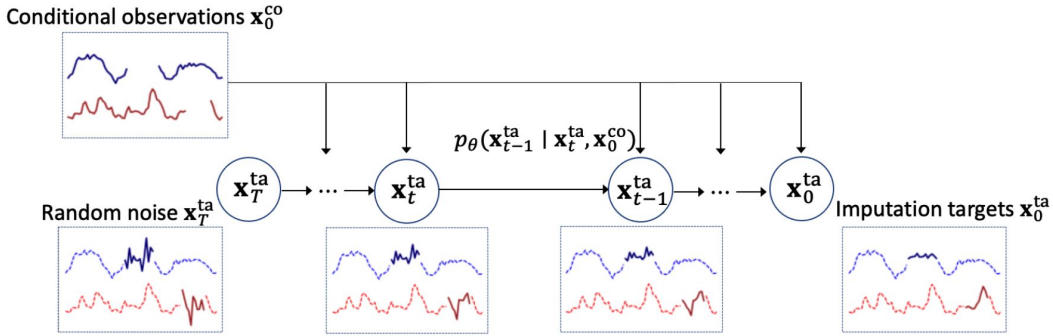


Figure 1: The procedure of time series imputation with CSDI. The reverse process p_θ gradually converts random noise into plausible time series, conditioned on observed values $\mathbf{x}_0^{co}$. Dashed lines in each box represent observed values, which are plotted in order to show the relationship with generated imputation and not included in each $\mathbf{x}_t^{ta}$.

https://arxiv.org/abs/2107.03502

Application (4): Time Series Imputation

CSDI

- Conditional diffusion model (condition = observed input)
- Conditional distribution: $p\left(\mathbf{x}_{t-1}^{\mathrm{ta}} \mid \mathbf{x}_{t}^{\mathrm{ta}}, \mathbf{x}_{0}^{\mathrm{co}}\right)$
- Imputation with diffusion model:

$$p_\theta(\mathbf{x}_{0:T}^{\mathrm{ta}} \mid \mathbf{x}_0^{\mathrm{co}}) := p(\mathbf{x}_T^{\mathrm{ta}}) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}^{\mathrm{ta}} \mid \mathbf{x}_t^{\mathrm{ta}}, \mathbf{x}_0^{\mathrm{co}}), \quad \mathbf{x}_T^{\mathrm{ta}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

$$p_\theta(\mathbf{x}_{t-1}^{\mathrm{ta}} \mid \mathbf{x}_t^{\mathrm{ta}}, \mathbf{x}_0^{\mathrm{co}}) := \mathcal{N}(\mathbf{x}_{t-1}^{\mathrm{ta}}; \boldsymbol{\mu}_\theta(\mathbf{x}_t^{\mathrm{ta}}, t \mid \mathbf{x}_0^{\mathrm{co}}), \sigma_\theta(\mathbf{x}_t^{\mathrm{ta}}, t \mid \mathbf{x}_0^{\mathrm{co}})\mathbf{I})$$

$$\boldsymbol{\mu}_\theta\left(\mathbf{x}_t^{\mathrm{ta}}, t \mid \mathbf{x}_0^{\mathrm{co}}\right) = \boldsymbol{\mu}^{\mathrm{DDPM}}\left(\mathbf{x}_t^{\mathrm{ta}}, t, \boldsymbol{\epsilon}_\theta\left(\mathbf{x}_t^{\mathrm{ta}}, t \mid \mathbf{x}_0^{\mathrm{co}}\right)\right)$$

$$\sigma_\theta\left(\mathbf{x}_t^{\mathrm{ta}}, t \mid \mathbf{x}_0^{\mathrm{co}}\right) = \sigma^{\mathrm{DDPM}}\left(\mathbf{x}_t^{\mathrm{ta}}, t\right)$$

https://arxiv.org/abs/2107.03502

Application (4): Time Series Imputation

CSDI

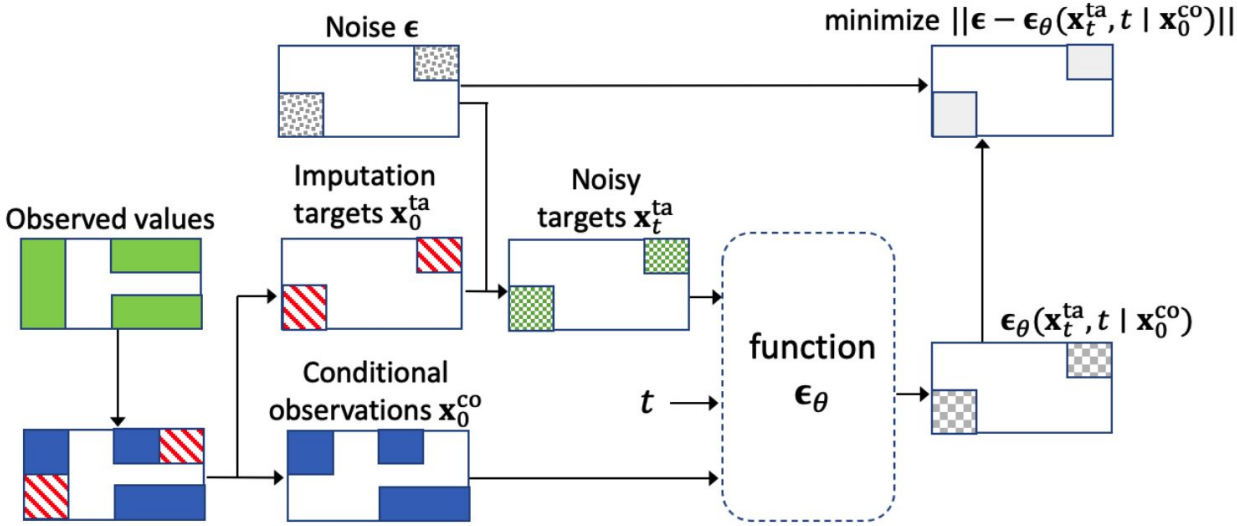


Figure 2: The self-supervised training procedure of CSDI. On the middle left rectangle, the green and white areas represent observed and missing values, respectively. The observed values are separated into red imputation targets $\mathbf{x}_0^{ta}$ and blue conditional observations $\mathbf{x}_0^{co}$, and used for training of ϵ_θ. The colored areas in each rectangle mean the existence of values.

https://arxiv.org/abs/2107.03502

Application (4): Time Series Imputation

CSDI

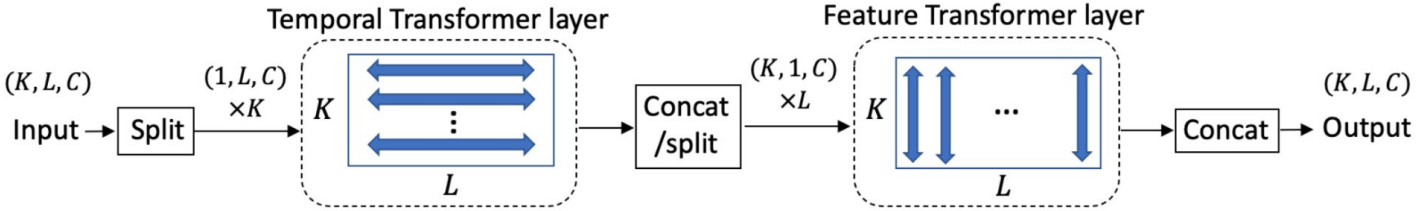


Figure 3: The architecture of 2D attention. Given a tensor with K features, L length, and C channels, the temporal Transformer layer takes tensors with $(1, L, C)$ shape as inputs and learns temporal dependency. The feature Transformer layer takes tensors with $(K, 1, C)$ shape as inputs and learns feature dependency. The output shape of each layer is the same as the input shape.

Table 1: Imputation targets $\mathbf{x}_0^{\mathrm{ta}}$ and conditional observations $\mathbf{x}_0^{\mathrm{co}}$ for CSDI at training and sampling.

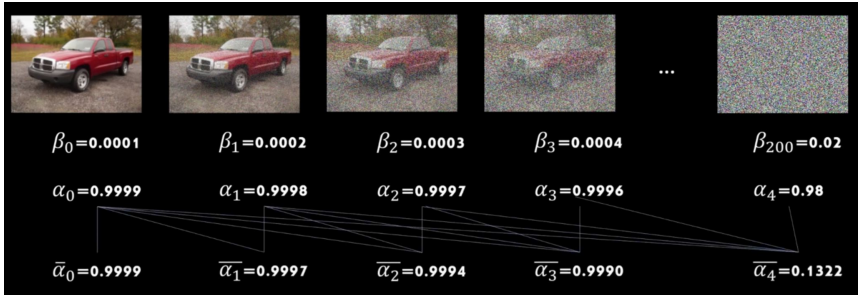
	imputation targets $\mathbf{x}_0^{\mathrm{ta}}$	conditional observations $\mathbf{x}_0^{\mathrm{co}}$
sampling (imputation)	all missing values	all observed values
training	a subset of the observed values (sampled by a target choice strategy)	the remaining observed values

7. Code Implementation

Code Implementation

Step 1) Forward Process (Noise Scheduler)

```python
def linear_beta_schedule(timesteps, start=0.0001, end=0.02):
    return torch.linspace(start, end, timesteps)

def get_index_from_list(vals, t, x_shape):
    """
    Returns a specific index t of a passed list of values vals
    while considering the batch dimension.
    """
    batch_size = t.shape[0]
    out = vals.gather(-1, t.cpu())
    return out.reshape(batch_size, *((1,) * (len(x_shape) - 1))).to(t.device)

def forward_diffusion_sample(x_0, t, device="cpu"):
    """
    Takes an image and a timestep as input and
    returns the noisy version of it
    """
    noise = torch.randn_like(x_0)
    sqrt_alphas_cumprod_t = get_index_from_list(sqrt_alphas_cumprod, t, x_0.shape)
    sqrt_one_minus_alphas_cumprod_t = get_index_from_list(
        sqrt_one_minus_alphas_cumprod, t, x_0.shape
    )
    # mean + variance
    return sqrt_alphas_cumprod_t.to(device) * x_0.to(device) \
    + sqrt_one_minus_alphas_cumprod_t.to(device) * noise.to(device), noise.to(device)
```

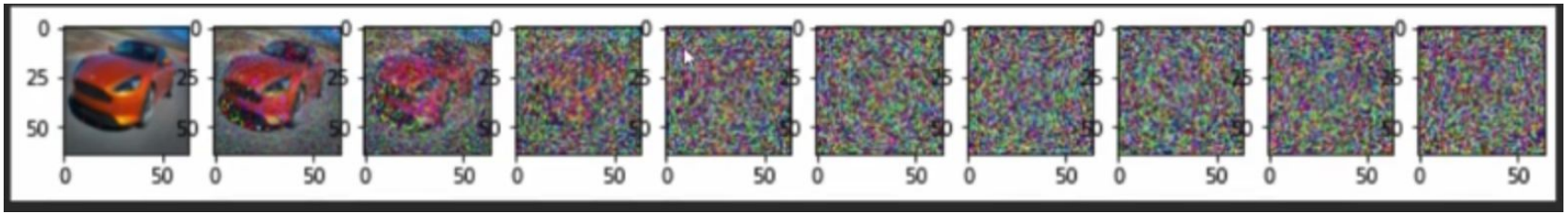


$$q(x_t \mid x_0) = N\left(x_t; \sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t) I\right)$$

Code Implementation

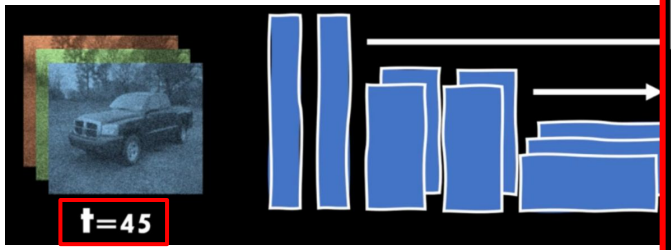
Step 1) Forward Process (Noise Scheduler)

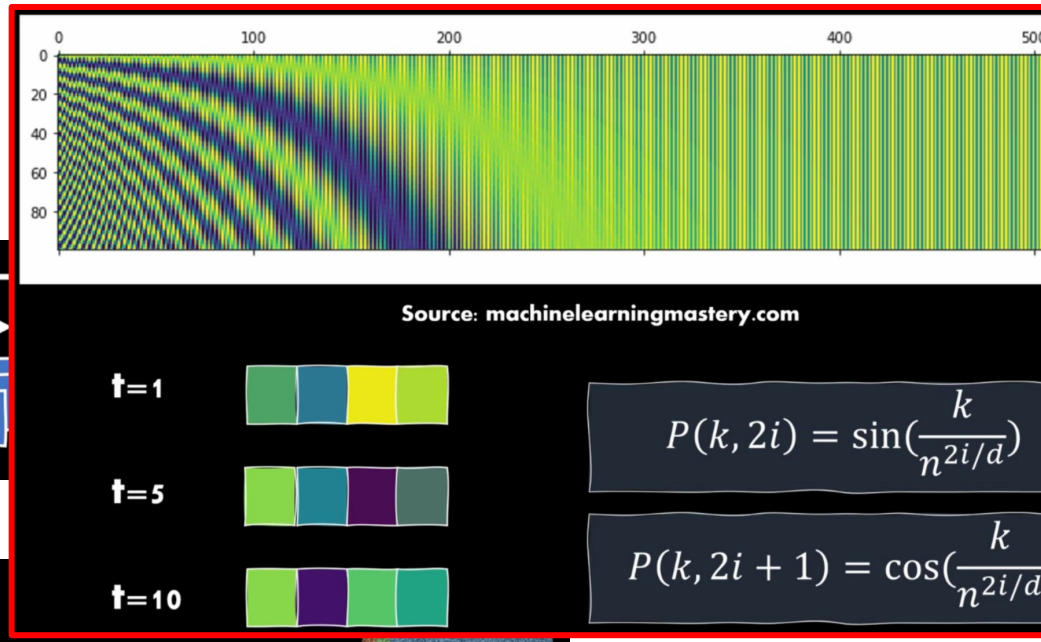
```
plt.figure(figsize=(15,15))
plt.axis('off')
num_images = 10
stepsize = int(T/num_images)

for idx in range(0, T, stepsize):
    t = torch.Tensor([idx]).type(torch.int64)
    plt.subplot(1, num_images+1, (idx/stepsize) + 1)
    image, noise = forward_diffusion_sample(image, t)
    show_tensor_image(image)
```

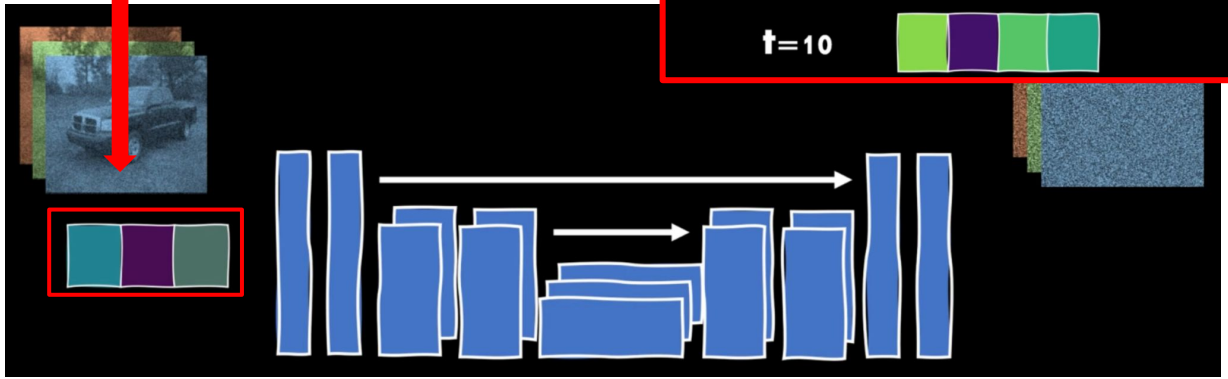


Code Implementation

Step 2) Time-step Encoding







Code Implementation

```
self.time_mlp = nn.Sequential(
        SinusoidalPositionEmbeddings(time_emb_dim)
        nn.Linear(time_emb_dim, time_emb_dim),
        nn.ReLU()
    )
```

Step 2) Time-step Encoding

```
class SinusoidalPositionEmbeddings(nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.dim = dim

    def forward(self, time):
        device = time.device
        half_dim = self.dim // 2
        embeddings = math.log(10000) / (half_dim - 1)
        embeddings = torch.exp(torch.arange(half_dim, device=device) * -embeddings)
        embeddings = time[:, None] * embeddings[None, :]
        embeddings = torch.cat((embeddings.sin(), embeddings.cos()), dim=-1)
        return embeddings
```

Code Implementation

Step 3) Model (U-Net)

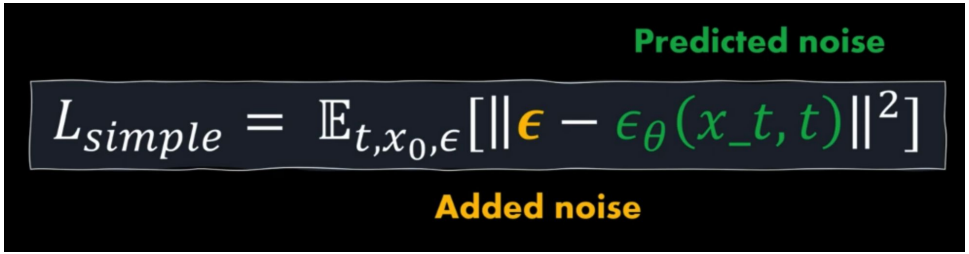
```
# Downsample
self.downs = nn.ModuleList([Block(down_channels[i], down_channels[i+1], \
                                  time_emb_dim) \
            for i in range(len(down_channels)-1)])
# Upsample
self.ups = nn.ModuleList([Block(up_channels[i], up_channels[i+1], \
                                time_emb_dim, up=True) \
            for i in range(len(up_channels)-1)])
```

```
def forward(self, x, timestep):
    # Embedd time
    t = self.time_mlp(timestep)
    # Initial conv
    x = self.conv0(x)
    # Unet
    residual_inputs = []
    for down in self.downs:
        x = down(x, t)
        residual_inputs.append(x)
    for up in self.ups:
        residual_x = residual_inputs.pop()
        # Add residual x as additional channels
        x = torch.cat((x, residual_x), dim=1)
        x = up(x, t)
    return self.output(x)
```

```
def forward(self, x, t, ):
    # First Conv
    h = self.bnorm(self.relu(self.conv1(x)))
    # Time embedding
    time_emb = self.relu(self.time_mlp(t))
    # Extend last 2 dimensions
    time_emb = time_emb[(..., ) + (None, ) * 2]
    # Add time channel
    h = h + time_emb
    # Second Conv
    h = self.bnorm(self.relu(self.conv2(h)))
    # Down or Upsample
    return self.transform(h)
```

Code Implementation

Step 4) Loss Function




```
def get_loss(model, x_0, t):
    x_noisy, noise = forward_diffusion_sample(x_0, t, device)
    noise_pred = model(x_noisy, t)
    return F.l1_loss(noise, noise_pred)
```

Code Implementation

Step 5) Sample Time-step

```python
@torch.no_grad()
def sample_timestep(x, t):
    """
    Calls the model to predict the noise in the image and returns
    the denoised image.
    Applies noise to this image, if we are not in the last step yet.
    """
    betas_t = get_index_from_list(betas, t, x.shape)
    sqrt_one_minus_alphas_cumprod_t = get_index_from_list(
        sqrt_one_minus_alphas_cumprod, t, x.shape
    )
    sqrt_recip_alphas_t = get_index_from_list(sqrt_recip_alphas, t, x.shape)

    # Call model (current image - noise prediction)
    model_mean = sqrt_recip_alphas_t * (
        x - betas_t * model(x, t) / sqrt_one_minus_alphas_cumprod_t
    )
    posterior_variance_t = get_index_from_list(posterior_variance, t, x.shape)

    if t == 0:
        return model_mean
    else:
        noise = torch.randn_like(x)
        return model_mean + torch.sqrt(posterior_variance_t) * noise
```

Algorithm 2 Sampling Inference

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

Code Implementation

Step 6) Train

```
for epoch in range(epochs):
    for step, batch in enumerate(dataloader):
      optimizer.zero_grad()

      t = torch.randint(0, T, (BATCH_SIZE,), device=device).long()
      loss = get_loss(model, batch[0], t)
      loss.backward()
      optimizer.step()
```

Select random timestep

Code Implementation

Step 7) Inference

```
@torch.no_grad()
def sample_plot_image():
    # Sample noise
    img_size = IMG_SIZE
    img = torch.randn((1, 3, img_size, img_size), device=device)
    plt.figure(figsize=(15,15))
    plt.axis('off')
    num_images = 10
    stepsize = int(T/num_images)

    for i in range(0,T)[::-1]:
        t = torch.full((1,), i, device=device, dtype=torch.long)
        img = sample_timestep(img, t)
        if i % stepsize == 0:
            plt.subplot(1, num_images, i/stepsize+1)
            show_tensor_image(img.detach().cpu())
    plt.show()
```

Recursive (reverse order)

References

- https://www.youtube.com/watch?v=BgaDxPSka8Q
- https://www.youtube.com/watch?v=Q_o0SpXv9kU
- https://www.youtube.com/watch?v=a4Yfz2FxXiY
- https://www.youtube.com/watch?v=687zEGODmHA