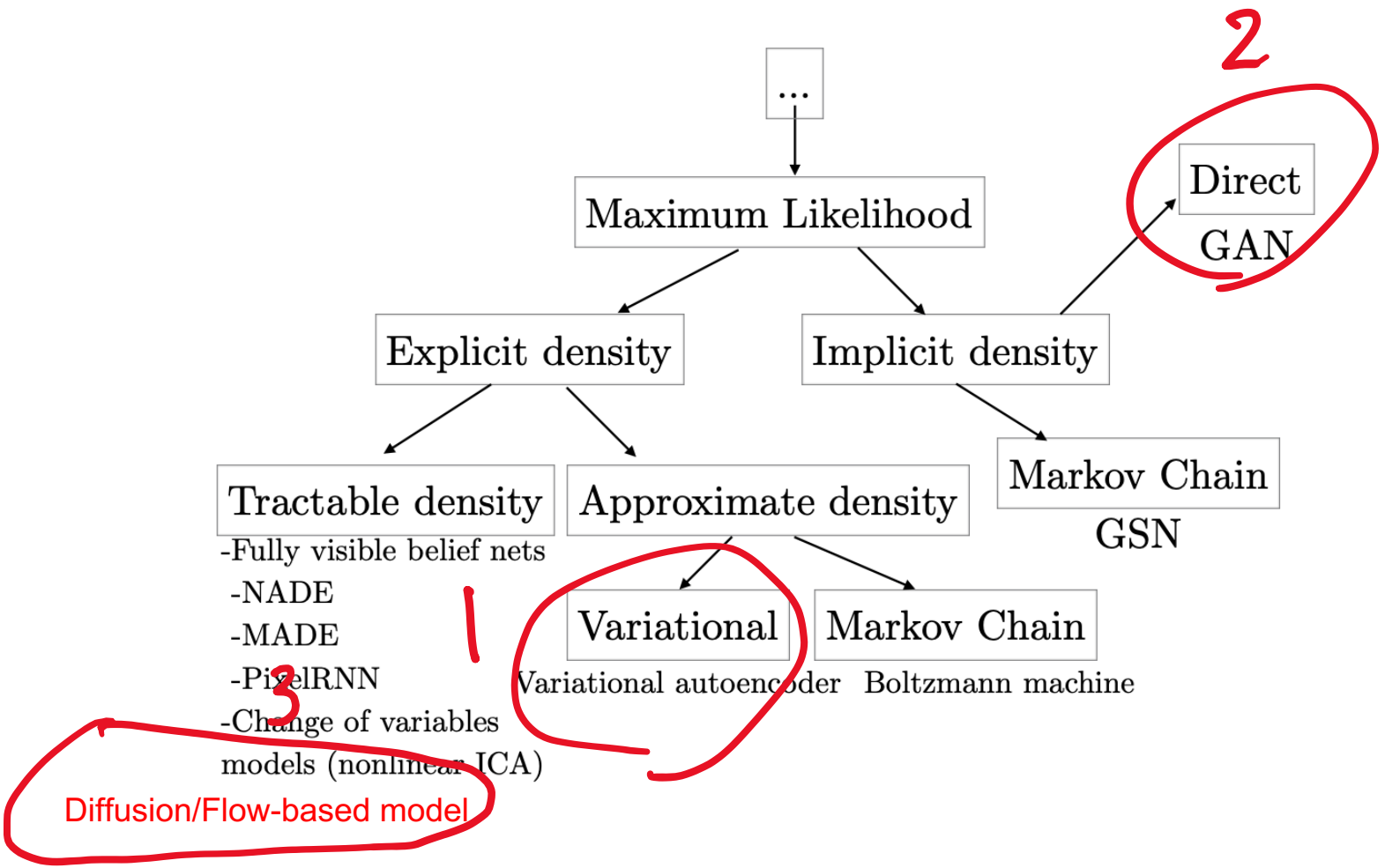
Deep Generative Models Part 2

Flow, Diffusion, and Parameter Efficient Transfer Learning


...
Maximum Likelihood
Direct
GAN
Explicit density
Implicit density
Tractable density
Approximate density
Markov Chain
GSN
-Fully visible belief nets
-NADE
-MADE
-PixelRNN
-Change of variables
models (nonlinear ICA)
Variational
Markov Chain
Variational autoencoder
Boltzmann machine
Diffusion/Flow-based model
1
2
3

VAE

Encoder

Means (2, 9)

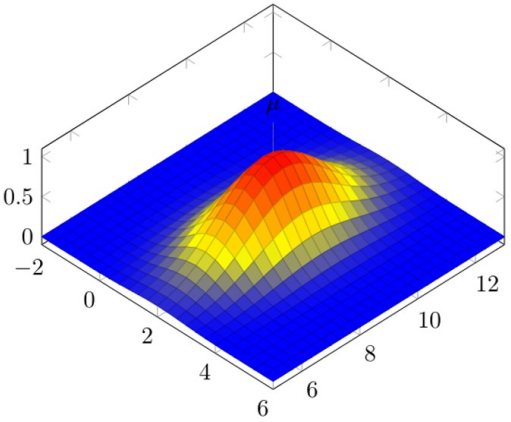
Std.

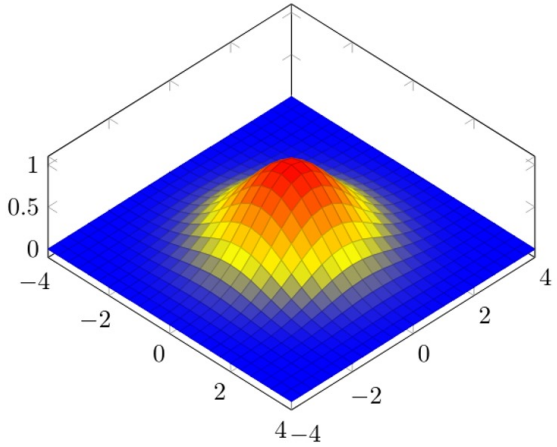
Sample

(2.1, 9.2)

$q(\mathbf{z} \mid \mathbf{x})$

Decoder





$$q^*(\mathbf{z} \mid \mathbf{x}) = \underset{q}{\operatorname{argmin}} \, \mathrm{KL}\left[q(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z} \mid \mathbf{x})\right]$$

ELBO

$q(\mathbf{z} \mid \mathbf{x})$ is forced to be close to $p(\mathbf{z}) = \mathcal{N}(0, I)$

$$\min_q \mathrm{KL}[q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})] - \mathbb{E}_q[\log p(\mathbf{x} \mid \mathbf{z})]$$

Force to be Gaussian ~L2

Effect of framing

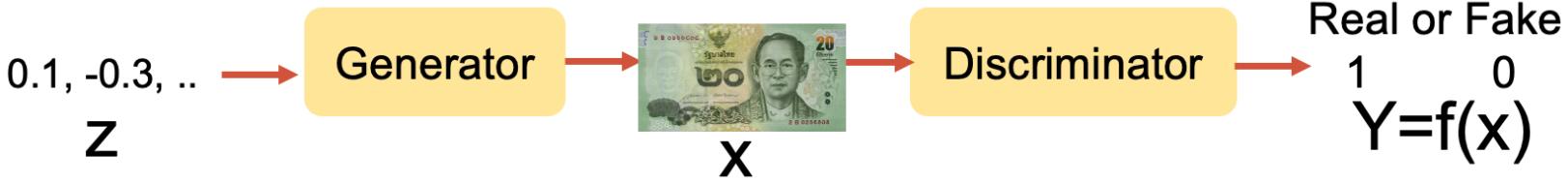
$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

1. Say we have a data distribution **p** that is a mixture of two 2D gaussians as shown below in red. We want to approximate this with one gaussian estimate **q** using KL-divergence. Which of the following three will result from optimizing $\mathbf{D}_{\mathrm{KL}}\ (p\ ||\ q)$
2. and which from $\mathbf{D}_{\mathrm{KL}}\ (q\ ||\ p)$?

$\mathbf{D}_{\mathrm{KL}}\ (q\ ||\ p)$ $\mathbf{D}_{\mathrm{KL}}\ (q\ ||\ p)$ $\mathbf{D}_{\mathrm{KL}}\ (p\ ||\ q)$

https://deeplearning.cs.cmu.edu/F23/document/slides/lec23.diffusion.updated.pdf

GAN



- Generator (Money Faker):
  - Maximize Y

$$\min_G \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z}))]$$

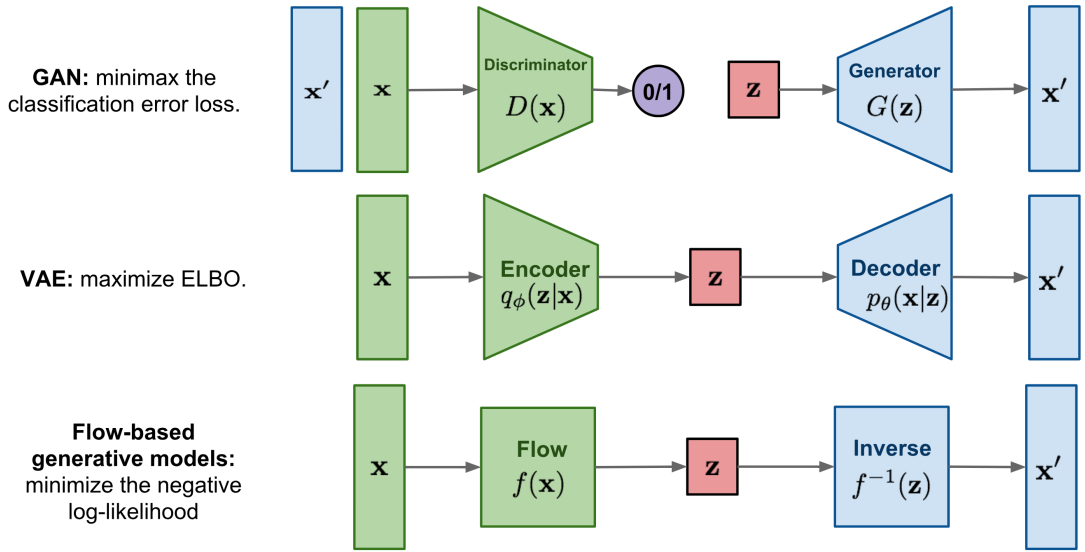
- Discriminator (Police):
  - For real images => Maximize Y
  - For generated images from the faker => Minimize Y

$$\max_D \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z}))] + \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log(D(\mathbf{x}))]$$

This comes from binary cross entropy loss

Flow-based model

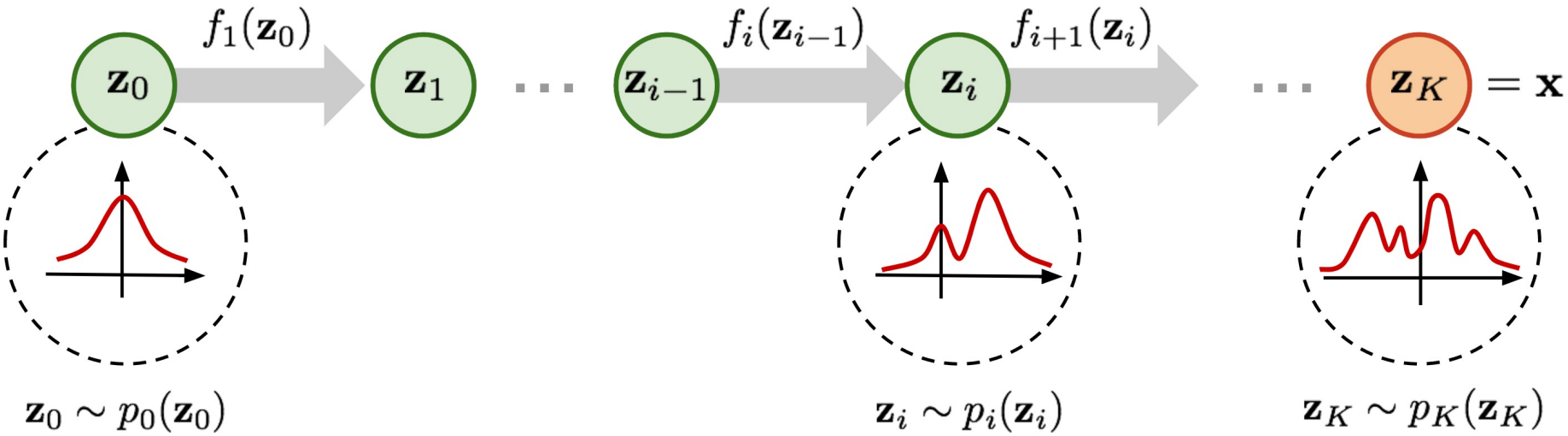
- Tweak distribution until you get the target distribution
- Deterministic reversible/invertable tweaks



https://lilianweng.github.io/posts/2018-10-13-flow-models/

https://lilianweng.github.io/posts/2018-10-13-flow-models/

Flow-based model



Noise

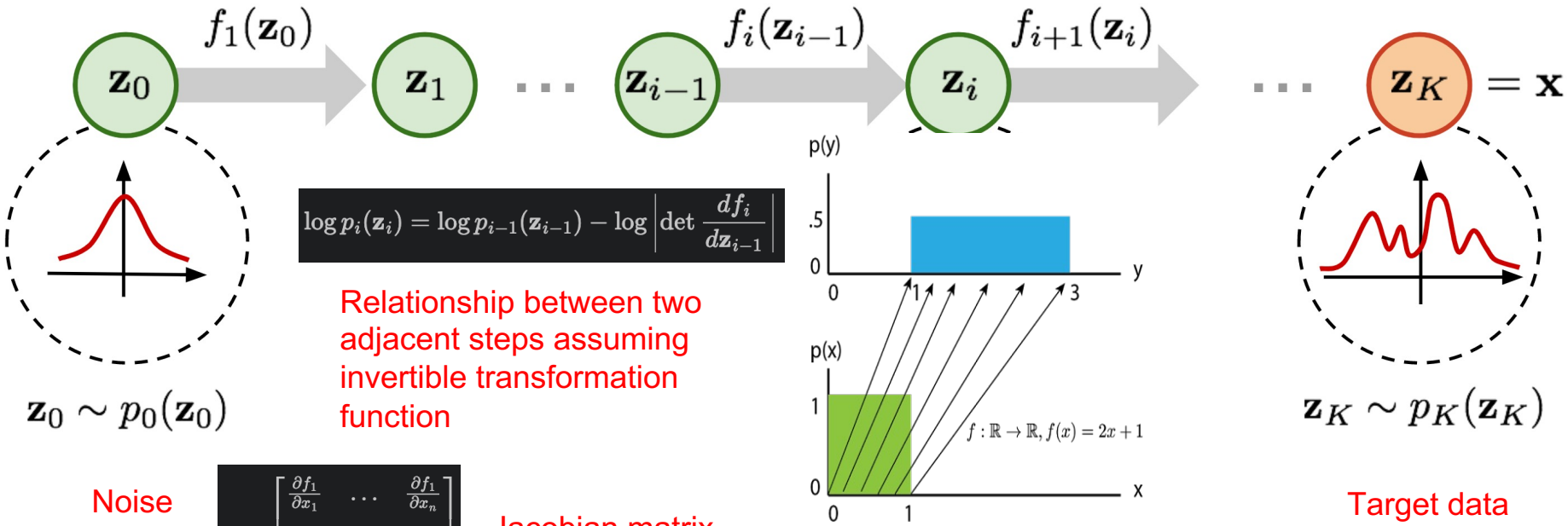
Target data

A flow-based model is a method that transforms a simple distribution to a complex one by successively applying a transformation function.

Here we are going to assume the transformation function is invertible.

https://lilianweng.github.io/posts/2018-10-13-flow-models/

Flow-based model



$$\log p_i(\mathbf{z}_i) = \log p_{i-1}(\mathbf{z}_{i-1}) - \log \left| \det \frac{d f_i}{d \mathbf{z}_{i-1}} \right|$$

Relationship between two adjacent steps assuming invertible transformation function

Noise

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Jacobian matrix (matrix of first order partial derivatives)

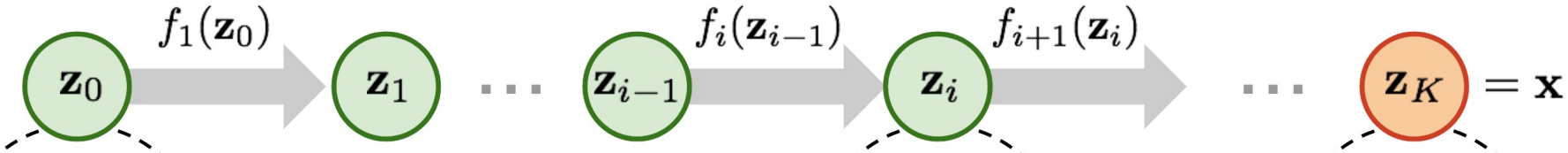
Target data

https://lilianweng.github.io/posts/2018-10-13-flow-models/

Flow-based model

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$\log p_i(\mathbf{z}_i) = \log p_{i-1}(\mathbf{z}_{i-1}) - \log \left| \det \frac{d f_i}{d \mathbf{z}_{i-1}} \right|$$

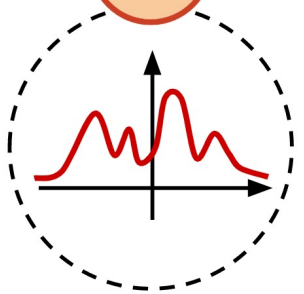


$$\mathbf{x} = \mathbf{z}_K = f_K \circ f_{K-1} \circ \cdots \circ f_1(\mathbf{z}_0)$$

$$\begin{aligned} \log p(\mathbf{x}) = \log \pi_K(\mathbf{z}_K) &= \log \pi_{K-1}(\mathbf{z}_{K-1}) - \log \left| \det \frac{d f_K}{d \mathbf{z}_{K-1}} \right| \\ &= \log \pi_{K-2}(\mathbf{z}_{K-2}) - \log \left| \det \frac{d f_{K-1}}{d \mathbf{z}_{K-2}} \right| - \log \left| \det \frac{d f_K}{d \mathbf{z}_{K-1}} \right| \\ &= \cdots \\ &= \log \pi_0(\mathbf{z}_0) - \sum_{i=1}^{K} \log \left| \det \frac{d f_i}{d \mathbf{z}_{i-1}} \right| \end{aligned}$$

Normalizing flow

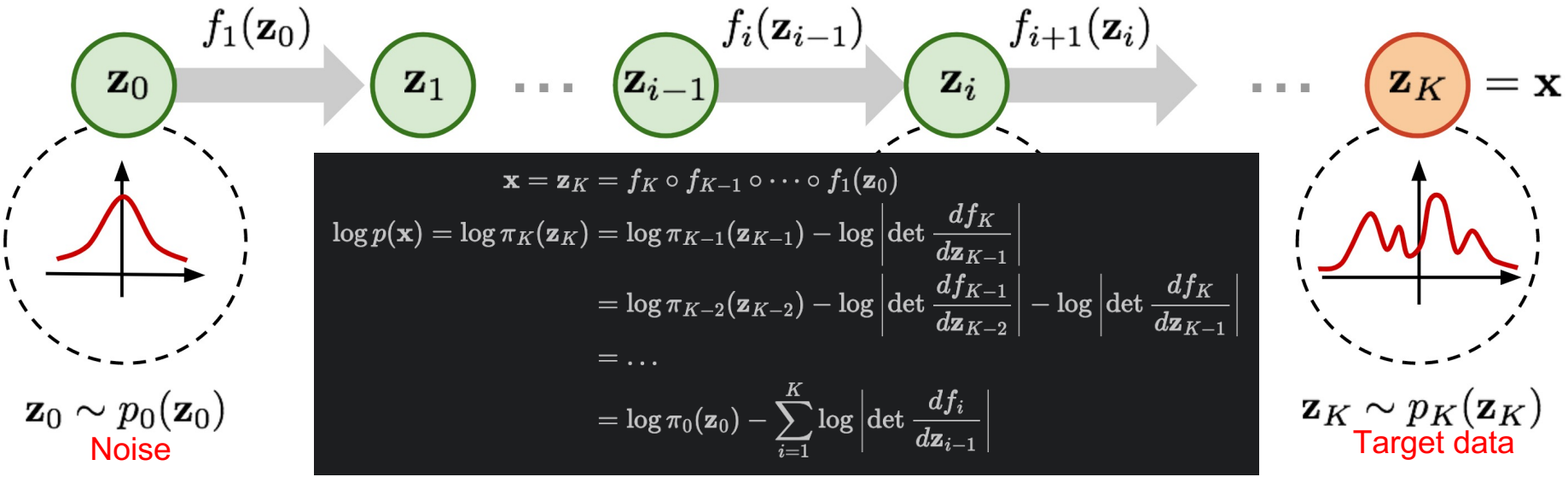
Now we can directly maximize log p(x)



$$\mathbf{z}_K \sim p_K(\mathbf{z}_K)$$

https://lilianweng.github.io/posts/2018-10-13-flow-models/

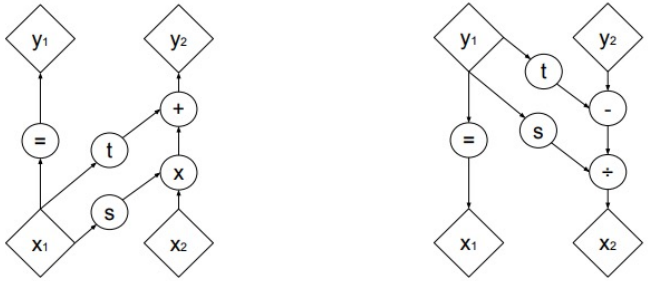
Flow-based model



Find f that maximizes the likelihood of the data
Constrain f so that it is easily invertable and det easy to compute
f is a neural network (needs networks that's easily invertable by design)

Example of an invertible layer

- RealNVP



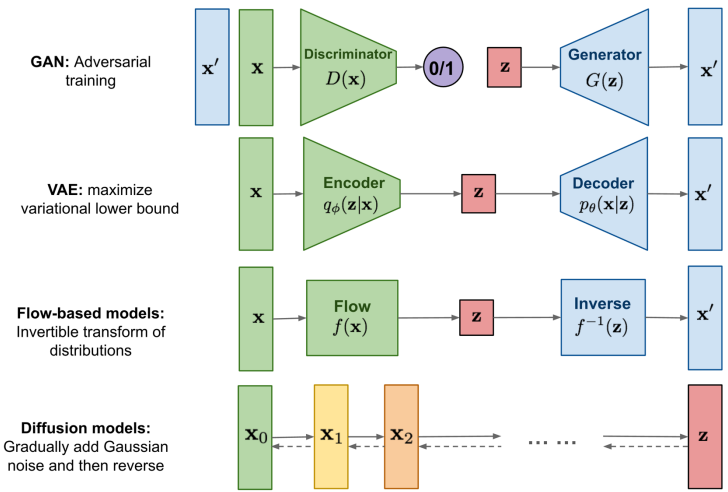
(a) Forward propagation (b) Inverse propagation

Figure 2: Computational graphs for forward and inverse propagation. A coupling layer applies a simple invertible transformation consisting of scaling followed by addition of a constant offset to one part $\mathbf{x}_2$ of the input vector conditioned on the remaining part of the input vector $\mathbf{x}_1$. Because of its simple nature, this transformation is both easily invertible and possesses a tractable determinant. However, the conditional nature of this transformation, captured by the functions s and t, significantly increase the flexibility of this otherwise weak function. The forward and inverse propagation operations have identical computational cost.

https://arxiv.org/abs/1605.08803

Diffusion Model

- Similar to flow
  - No longer constrained to revertible operations
  - Add noise to image until you get just noise



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Lecture Plan

1. Why Diffusion? (Motivation)
2. What is Diffusion?
3. Types of Diffusion
    a. Probabilistic Model (Focus on this one)
    b. Stochastic Differential Equations
4. What is the problem of Diffusion? (Trends)
    a. Stable Diffusion
    b. Knowledge Distillation
5. Parameter Efficient Transfer Learning
    a. LORA
    b. Adapters

Benefits of Diffusion

- A promising result, high-quality image with fine details and realistic textures.
- Stable training, compared to GAN

Application

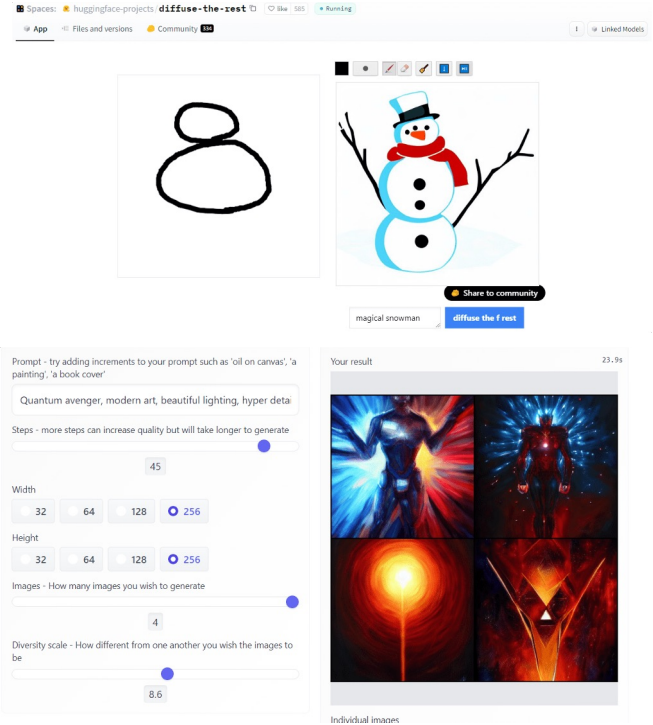
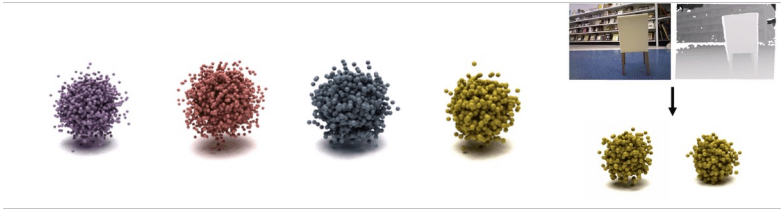


Image Generation

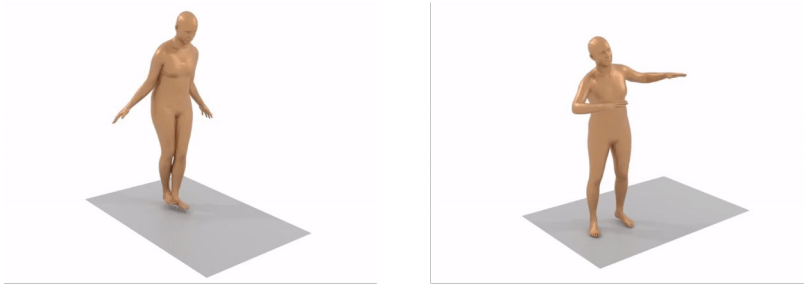
Image Restoration

Image Super Resolution

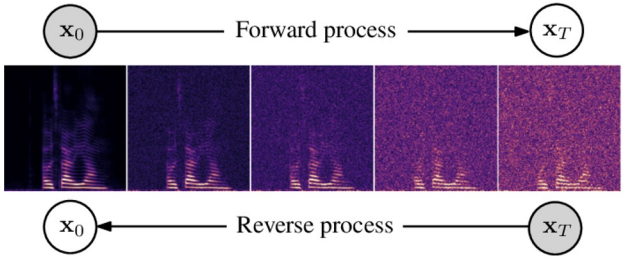
Beyond Image



3D Generation



Motion Generation



Speech Enhancement

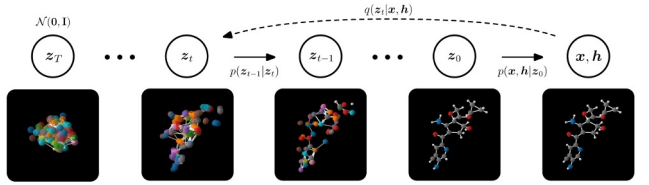


Figure 2. Overview of the Equivariant Diffusion Model. To generate molecules, coordinates $\boldsymbol{x}$ and features $\boldsymbol{h}$ are generated by denoising variables $\boldsymbol{z}_t$ starting from standard normal noise $\boldsymbol{z}_T$. This is achieved by sampling from the distributions $p(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$ iteratively. To train the model, noise is added to a datapoint $\boldsymbol{x}, \boldsymbol{h}$ using $q(\boldsymbol{z}_t|\boldsymbol{x}, \boldsymbol{h})$ for the step t of interest, which the network then learns to denoise.

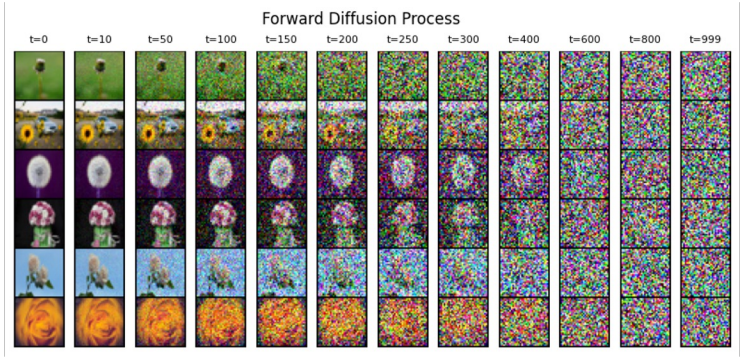
Graph Generation

What is Diffusion?

The concept of diffusion was adopted from non-equilibrium thermodynamics.

Diffusion model comprises of 2 process:

1. Forward Process
    a. Gradually insert gaussian noise into a clean image, until it becomes gaussian noise.
    b. Objective: Transform a complex distribution (a clean image) into a simple distribution (such as Gaussian distribution)
2. Reversed Process
    a. Gradually remove gaussian noise, until it becomes a clean image.
    b. Objective: Generate a new image from sampled noise.



How to model diffusion process?

Diffusion process can be classify into two categories. (Ultimately they are equivalent)

1. Probabilistic Model
    a. Exploit Markov Chain process to diffusion process (repeatedly insert noise)
    b. Example: DDPM, DDIM

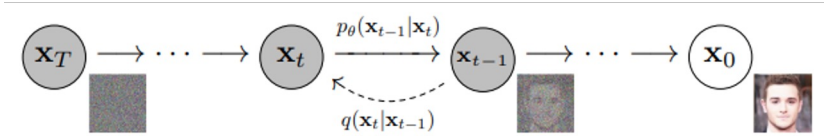
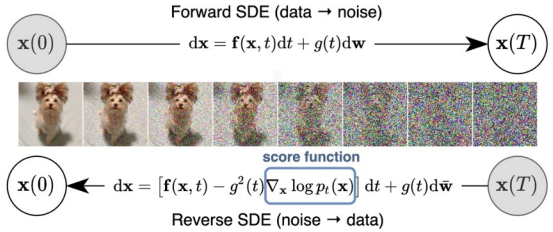


Figure 2: The directed graphical model considered in this work.

1. Stochastic Differential Equation
    a. Employ Stochastic Differential Equation to further extend the discrete process into continuous process
    b. Example: Score-based Diffusion model



Denoising Diffusion Probabilistic Model

Denoising Diffusion Probabilistic Models (DDPM)

Formulate the forward process with Markov Chain process

- Forward Process
  - First, we define the probability of $\mathbf{x}_t$ given $\mathbf{x}_{t-1}$ such that we can sample forward process with a mathematical process.

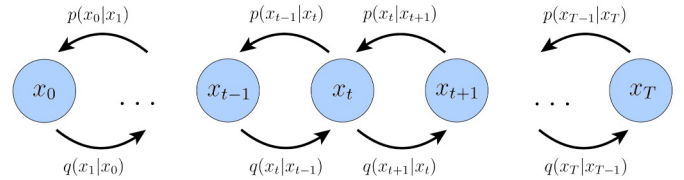
$$x_i = \sqrt{1-\beta_i} x_{i-1} + \sqrt{\beta_i}\epsilon_{i-1} \qquad i = 1, \cdots, N$$
$$x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1-\alpha_t}\epsilon_t; \qquad \epsilon_t \sim \mathcal{N}(0, \mathbb{I})$$

  - Given α_t is a decreasing sequence.

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t} x_{t-1}, (1-\alpha_t)\mathbb{I})$$

  - From markov chain property, we can further extend the probability of $\{\mathbf{x}_1, \ldots \mathbf{x}_T\}$ given $\mathbf{x}_0$

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$





https://arxiv.org/abs/2006.11239 **Denoising Diffusion Probabilistic Models 2020**

Denoising Diffusion Probabilistic Models (DDPM)

What's the benefits of defining the forward process like this? $q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)\mathbb{I})$

- It allow us to sample $\mathbf{x_t}$ from $\mathbf{x_0}$, using the following equation

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_{t-1}, (1-\bar{\alpha}_t)\mathbb{I}) \qquad \bar{\alpha}_t = \prod_{t'=1}^{t'=t} \alpha'_t$$

- Proof

$$x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon_t; \qquad \epsilon_t \sim \mathcal{N}(0, \mathbb{I})$$
$$x_t = \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_{t-1}}\epsilon_{t-1}) + \sqrt{1-\alpha_t}\epsilon_t$$
$$x_t = \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\epsilon_{t-1} + \sqrt{1-\alpha_t}\epsilon_t$$
$$x_t = \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\epsilon'_t; \qquad \epsilon'_t \sim \mathcal{N}(0, \mathbb{I})$$

We combine ϵ_{t-1} and ϵ_t into ϵ'_t by using the following theorem.

1. If a is a constant and $x \sim \mathcal{N}(\mu, \sigma^2)$, then $ax \sim \mathcal{N}(a\mu, (a\sigma)^2)$
2. If $x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$, then $x_1 + x_2 \sim \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$

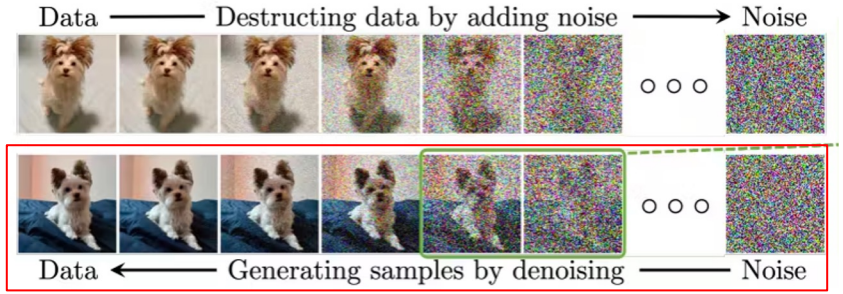
Denoising Diffusion Probabilistic Models (DDPM)

Reverse process is a generative process which we want to learn using deep learning model.

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

To sample image from reverse process:

1. Sample a noise from Gaussian distribution, $\mathbf{x_T}$
2. Deploy our deep learning model to repeatedly remove the noise



Denoising Diffusion Probabilistic Models (DDPM)

In a typical generative model, we want to learn the distribution of the data (clean images).

However, it is intractable to compute due to

- Infeasible to compute $p(x_0)$, due to the enormous space of $x_{1:T}$
- Unknown data distribution, $q(x_0)$

$$p(x_0) = \int p(x_0|x_{1:T})p(x_{1:T})dx_{1:T}$$

Fortunately, we have a technique from VAE, Evidence Lower-Bound (ELBO)

Consider $D_{\mathrm{KL}}(q(x_{1:T}|x_0)||p(x_{1:T}|x_0))$

$$\begin{aligned}
&= \mathbb{E}_{x_{1:T}\sim q}[\log q(x_{1:T}|x_0) - \log p(x_{1:T}|x_0)] \\
&= \mathbb{E}_{x_{1:T}\sim q}[\log q(x_{1:T}|x_0) - \log \frac{p(x_{0:T})}{p(x_0)}] \\
&= \mathbb{E}_{x_{1:T}\sim q}[\log q(x_{1:T}|x_0) - \log p(x_{0:T})] + \mathbb{E}_{x_{1:T}\sim q}[\log p(x_0)]
\end{aligned}$$

KL always >= 0

Therefore $\mathbb{E}_{x_0\sim q(x_0)}[-\log p(x_0)] < \mathbb{E}_{x_{0:T}\sim q(x_{0:T})}[\log q(x_{1:T}|x_0) - \log p(x_{0:T})]$

Tractable to compute

Data Distribution

Denoising Diffusion Probabilistic Models (DDPM)

For explainability, loss function is splitted into 3 terms. $\mathbb{E}_{x_{0:T}\sim q(x_{0:T})}[\log q(x_{1:T}|x_0) - \log p(x_{0:T})]$

To decompose this equation, we use three equations.

- Forward process is a markov process

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}) \qquad q(x_t|x_{t-1}) = q(x_t|x_{t-1}, x_0)$$

- Reverse process is also a markov process

$$p(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p(x_{t-1}|x_t)$$

See appendix

We learn this term

$$\mathbb{E}_{x_{0:T}\sim q(x_{0:T})}[\underbrace{D_{\mathrm{KL}}(q(x_T|x_0)||p(x_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{D_{\mathrm{KL}}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))}_{L_{t-1}} - \underbrace{\log p(x_0|x_1)}_{L_0}]$$

Prior Matching of starting point

Denoising

Reconstruction at first step

Denoising Diffusion Probabilistic Models (DDPM)

Our deep learning model is used to predict mean of the reverse process and for simplicity we set variance equal to untrained time dependent constant.

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

L_T Loss $D_{\mathrm{KL}}(q(x_T|x_0)||p(x_T))$

- This term is a constant and can be ignored because the scheduler (α_t) has no learnable parameters and prior sampling is sampled from Gaussian noise.

L_{t-1} Loss $D_{\mathrm{KL}}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$

- We need **$q(x_{t-1} \mid x_t, x_0)$** to compute this

L_0 Loss $\log p(x_0|x_1)$

- Probability of **x_0** given **x_1** is Normal distribution with predicted mean. $\log(p(x_0|x_1)) = \frac{1}{2\sigma_1^2}(x_0 - \mu_\theta(x_1, 1))^2 + C$

Denoising Diffusion Probabilistic Models (DDPM)

Fortunately L_{t-1} doesn't require the actual probability of reverse process which is intractable and it only require to calculate probability of reverse process given x_0.

This term is much simpler to calculate.

$$D_{\mathrm{KL}}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$$

$$\begin{aligned} q(x_{t-1}|x_t, x_0) &= \frac{q(x_{t-1}, x_t, x_0)}{q(x_t, x_0)} \\ &= \frac{q(x_t|x_{t-1}, x_0)q(x_{t-1}, x_0)}{q(x_t, x_0)} \\ &= \frac{q(x_t|x_{t-1})q(x_{t-1}|x_0)}{q(x_t|x_0)} \end{aligned}$$

See appendix

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}_t, \tilde{\beta}_t \mathbb{I})$$

$$\tilde{\mu}_t = \frac{\sqrt{\alpha}_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t + \frac{\sqrt{\bar{\alpha}}_{t-1}\beta_t}{1-\bar{\alpha}_t}x_0$$

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

$$D_{\mathrm{KL}}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)) = \mathbb{E}_q[\log \frac{\sqrt{\Sigma_\theta}}{\sqrt{\tilde{\beta}_t}} \exp(-\frac{1}{2}(\frac{(x_{t-1}-\tilde{\mu}_t)^2}{\tilde{\beta}_t} - \frac{(x_{t-1}-\mu_\theta)^2}{\Sigma_\theta})]$$

$$D_{\mathrm{KL}}(q(x_{t-1}|x_t,x_0)||p_\theta(x_{t-1}|x_t)) = \mathbb{E}_q[\log \frac{\sqrt{\Sigma_\theta}}{\sqrt{\tilde{\beta}_t}} \exp(-\frac{1}{2}(\frac{(x_{t-1}-\tilde{\mu}_t)^2}{\tilde{\beta}_t} - \frac{(x_{t-1}-\mu_\theta)^2}{\Sigma_\theta})]$$

Denoising Diffusion Probabilistic Models (DDPM)

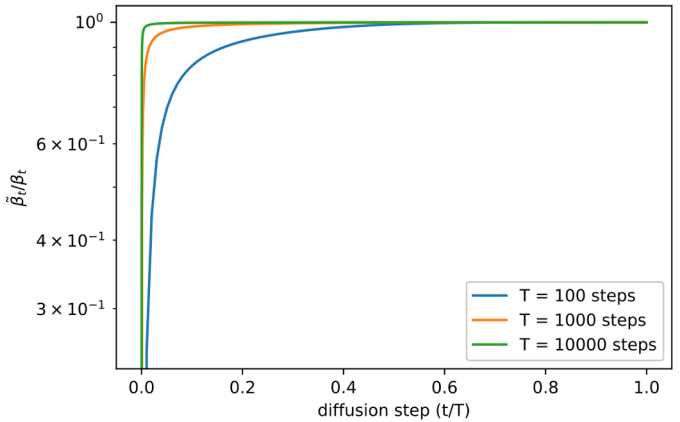
$$\tilde{\beta}_t = \frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}$$

The actual variance of reverse process. $q(x_{t-1}|x_t)$

- Upper-bound: The variance of forward process $q(x_t|x_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbb{I})$
  - Optimal for $x_0 \sim \mathcal{N}(0,\mathbb{I})$
- Lower-bound: The variance of reverse process given $\mathbf{x_0}$ $q(x_{t-1}|x_t,x_0) = \mathcal{N}(x_{t-1};\tilde{\mu}_t,\tilde{\beta}_t\mathbb{I})$
  - Optimal for $\mathbf{x_0}$ determinedly set to one point.
  - It is an extreme case for the reverse process because we don't know $\mathbf{x_0}$.

Based on the paper, Σ_θ was set to its upper bound.

Note that the range of upper-bound and lower-bound is extremely small when diffusion step is large.



Denoising Diffusion Probabilistic Models (DDPM)

In summary, our loss function $\mathbb{E}_{x_{0:T}\sim q(x_{0:T})}[\underbrace{D_{\mathrm{KL}}(q(x_T|x_0)||p(x_T))}_{L_T} + \sum_{t=2}^{T}\underbrace{D_{\mathrm{KL}}(q(x_{t-1}|x_t,x_0)||p_\theta(x_{t-1}|x_t))}_{L_{t-1}} - \underbrace{\log p(x_0|x_1)}_{L_0}]$

L_T Loss

- Constant with respect to **theta** (reverse process).

L_{t-1} Loss

$$D_{\mathrm{KL}}(q(x_{t-1}|x_t,x_0)||p_\theta(x_{t-1}|x_t)) = \mathbb{E}_q[\frac{1}{2\tilde{\beta}_t}(\tilde{\mu}_t - \mu_\theta)^2] + C$$

L_0 Loss

$$\mathbb{E}_q[-\log p_\theta(x_0|x_1)] = \mathbb{E}_q[\frac{1}{2\tilde{\beta}_1}(x_0 - \mu_\theta)^2] + C$$

Denoising Diffusion Probabilistic Models (DDPM)

Besides modeling the mean of the reverse process, we can further parameterize the model to predict the noise instead.

$$
\begin{aligned}
\tilde{\mu}_t &= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} x_0 \\
&= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_t) \\
&= \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t)
\end{aligned}
$$

$$
\begin{aligned}
q(x_t|x_0) &= \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathbf{I}) \\
x_t &= \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_t \\
x_0 &= \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_t)
\end{aligned}
$$

From the above equation, we parameterize model to predict noise.

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

$$
\begin{aligned}
\mu_\theta &= \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta) \\
\Sigma_\theta &= \tilde{\beta}_t
\end{aligned}
$$

Modeling the noise leads to easier learning*. It also has nice theoretical interpretations (see paper).

* Not always, according to my experience

Denoising Diffusion Probabilistic Models (DDPM)

Substitute the equations from the previous page into Loss function

- L_{t-1} Loss: $\mathbf{E}_q[\frac{1}{2\tilde{\beta}_t}(\tilde{\mu}_t - \mu_\theta)^2] = \mathbf{E}_q[\frac{\beta_t^2}{2\tilde{\beta}_t\alpha_t(1-\bar{\alpha}_t)}(\epsilon_t - \epsilon_\theta)^2]$

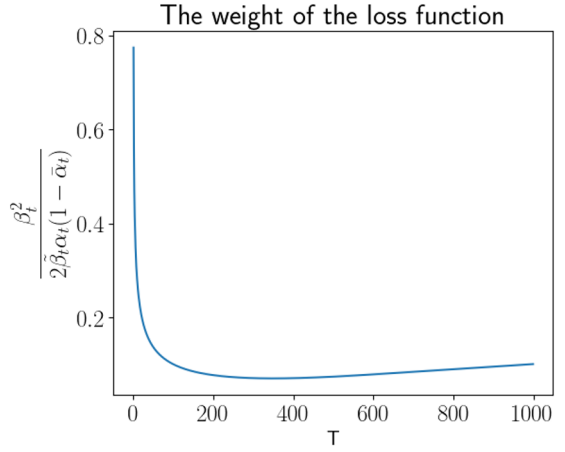
- L_0 Loss: $\mathbf{E}_q[\frac{1}{2\tilde{\beta}_1}(x_0 - \mu_\theta)^2] = \mathbf{E}_q[\frac{\beta_1^2}{2\tilde{\beta}_1\alpha_1(1-\bar{\alpha}_1)}(\epsilon_1 - \epsilon_\theta)^2]$

From our parameterization, it shows that both L_{t-1} and L_0 have the same form and can be merged.

$$\mathbf{E}_{t\sim\text{Uniform}(0,T),x_0\sim q,\epsilon_t\sim\mathcal{N}(0,\mathbb{I})}\left[\frac{\beta_t^2}{2\tilde{\beta}_t\alpha_t(1-\bar{\alpha}_t)}(\epsilon_t - \epsilon_\theta)^2\right]$$

Can be ignored

The paper also remarks that the weight can be omitted to encourage the model to focus on more challenging denoising tasks at larger time steps (t terms).



Denoising Diffusion Probabilistic Models (DDPM)

Algorithm 1 Training

1: **repeat**
2: $x_0 \sim q(x_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: $x_t \leftarrow \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$
6: Take gradient descent step on $\nabla_\theta ||\epsilon - \epsilon_\theta(x_t, t)||^2$
7: **until** converged

Algorithm 2 Sampling

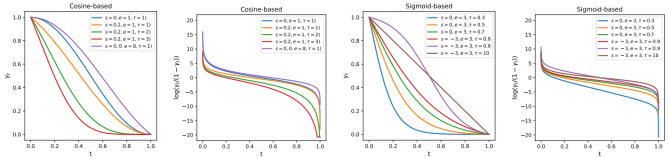
1: $x_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $z = 0$
4: $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)) + \sigma_t z$
5: **end for**
6: **return** x_0

Noise schedule α_t

- In the DDPMpaper

• We chose the β_t schedule from a set of constant, linear, and quadratic schedules, all constrained so that $L_T \approx 0$. We set $T = 1000$ without a sweep, and we chose a linear schedule from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$.

- It is found that Best noise schedule depends on the resolution of the image and the task
- Thus, it is important to find a proper noise schedule for your task



(a) Cosine (b) Cosine (logSNR) (c) Sigmoid (d) Sigmoid (logSNR)

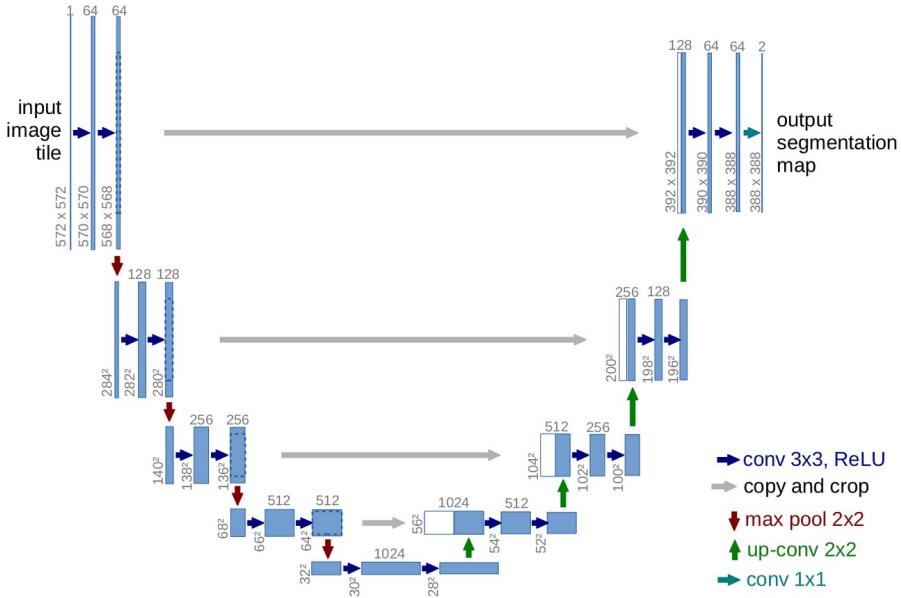
Figure 3: Instantiations of noise schedule function $\gamma(t)$ and the corresponding logSNR. Adjusting hyper-parameters of cosine and sigmoid functions leads to different noise schedules.

Frechet Inception Distance (FID) inception score (IS)

Resolution	Method	FID	IS	Params (M)
64×64	ADM [3]	–	2.07	297
	CF-guidance [6]	1.55	66.0	–
	CDM [8]	1.48	66.0	–
	RIN [10] (patch size of 4, 300K updates)	**1.23**	**66.5**	281
	RIN+our strategy (patch size of 8, 150K updates)	2.04	55.8	**214**
128×128	ADM [3]	5.91	–	386
	ADM+guidance [3]	2.97	–	> 386
	CF-guidance[6]	**2.43**	**156.0**	–
	CDM[8]	3.51	128.0	1058
	RIN [10] (patch size of 4, 700K updates)	2.75	144.1	410
	RIN+our strategy (patch size of 8, 250K updates)	3.50	120.4	**215**
256×256	ADM [3]	10.94	100.9	553
	ADM+guidance [3]	4.59	–	>553
	CDM [8]	4.88	158.7	1953
	RIN [10] (patch size of 8, 700K updates)	4.51	161.0	410
	RIN+our strategy (patch size of 8, 250K updates)	**3.52**	**186.2**	**319**
512×512	ADM [3]	23.2	58.1	559
	ADM+guidance [3]	7.72	172.7	>559
	RIN+our strategy (patch size of 8, 1M updates)	**3.95**	**216**	**320**
768×768	RIN+our strategy (patch size of 8, 1M updates)	**5.60**	**196.2**	**408**
1024×1024	RIN+our strategy (patch size of 8, 910K updates)	**8.72**	**163.9**	**412**

https://arxiv.org/pdf/2301.10972.pdf 2023 On the Importance of Noise Scheduling for Diffusion Models

Model architecture



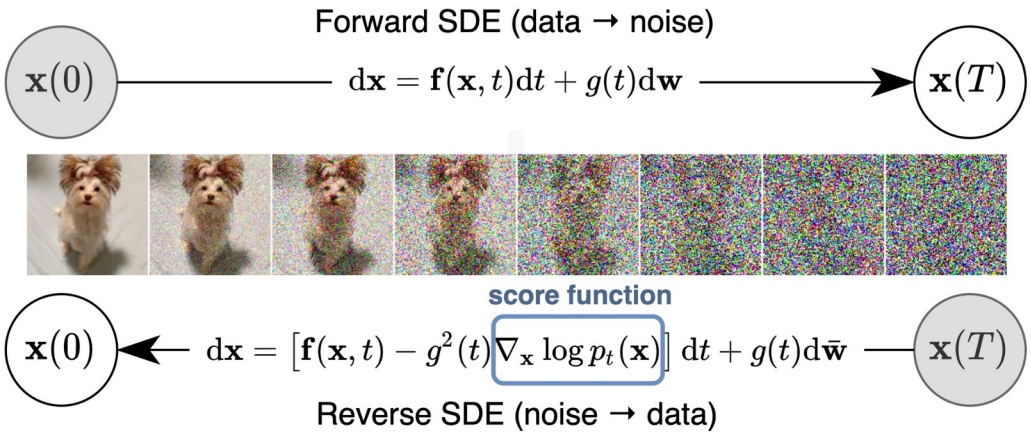
Score-based Generative Modeling through Stochastic Differential Equation

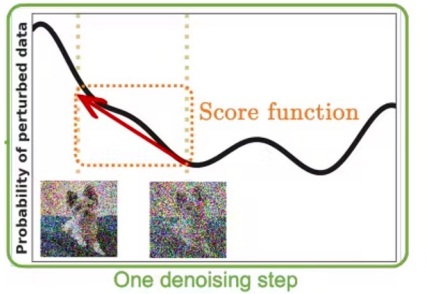
Score-Based Generative Model

Score-Based Generative Model further extends the discrete process, DDPM, to a **continuous** stochastic process.

A typical stochastic processes are solutions of stochastic differential equations (SDE) including the diffusion process.

Therefore, SDE can be used as a tool to represent diffusion process in a continuous space.





Score-Based Generative Model

Forward Process is represented as SDE

- **Drift coefficient**: Metaphor as mean of forward process
- **Diffusion coefficient**: Metaphor as standard deviation of forward process
- **Derivative of Brownian motion**: Metaphor as a Gaussian noise

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$$

Drift coefficient Diffusion coefficient Derivative of Brownian motion

Benefits of formulating through SDE

- Flexibility to design diffusion process through adjusting the drift and diffusion coefficient
- Formulate diffusion process in the continuous space

Score-Based Generative Model

The connection between DDPM and Score-Based generative model

According to DDPM, forward process: $x_i = \sqrt{1-\beta_i}x_{i-1} + \sqrt{\beta_i}\epsilon_{i-1} \qquad i = 1, \cdots, N$

As N → Infinity, the forward process will converges to the following SDE.

$$\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\mathrm{d}t + \sqrt{\beta(t)}\mathrm{d}\mathbf{w} \qquad \mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$$

This prove that DDPM and Score-Based generative model are equivalent.

The difference between these two is a tool to formulate different diffusion processes.

- **DDPM**: Design a diffusion process through ***probabilistic modeling***
- **Score-Based generative model**: Design a diffusion process through ***stochastic differential equation***

Score-Based Generative Model

Any stochastic differential equation has a corresponding reverse SDE:

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$$

Unknown

In reverse SDE, there is one unknown term, **score function**, and we will train model to predict it. $s_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$

- An advantage of using a **score function** model is that it lets us disregard the intractable normalization constants.

$$p_\theta(x) = \frac{e^{-f_\theta(x)}}{Z_\theta} \longrightarrow \int p_\theta(x)dx = 1$$

Hard to train

Score-Based Generative Model

How to train score function model?

- The objective of *score matching* task is to minimize the following equation.

$$\frac{1}{2}\mathbb{E}_{p_{\text{data}}}[||s_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})||_2^2] \Leftrightarrow \mathbb{E}_{p_{\text{data}}(\mathbf{x})}[\text{tr}(\nabla_{\mathbf{x}} s_\theta(\mathbf{x})) + \frac{1}{2}||s_\theta(\mathbf{x})||_2^2]$$

Unknown — High dimension

- To circumvent this problem, there are several techniques that can be used
  - **Denoising score matching**: Perturbs the data point **x** with a pre-specified noise

$$\frac{1}{2}\mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p_{\text{data}}(\mathbf{x})}[||s_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})||_2^2]$$

  - **Sliced score matching**: Uses random projections to approximate in $\text{tr}(\nabla_{\mathbf{x}} s_\theta(\mathbf{x}))$ score matching

$$\mathbb{E}_{p_{\mathbf{v}}}\mathbb{E}_{p_{\text{data}}}[\mathbf{v}^{\text{T}}\nabla_{\mathbf{x}} s_\theta(\mathbf{x})\mathbf{v} + \frac{1}{2}||s_\theta(\mathbf{x})||_2^2]$$

Score-Based Generative Model

Using **denoising score matching** technique, the objective is

$$\mathbf{E}_{x_t \sim p_t(x_t)}[||s_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)||_2^2]$$

$$\int p_t(\mathbf{x}_t)[||s_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)||_2^2]\mathrm{d}\mathbf{x}_t$$

$$\int \int p_t(\mathbf{x}_t|\mathbf{x}_0)p_{\text{data}}(x_0)[||s_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)||_2^2]\mathrm{d}\mathbf{x}_0\mathrm{d}\mathbf{x}_t$$

$$\mathbf{E}_{x_t \sim p_t(x_t|x_0), x_0 \sim p_{\text{data}}(x_0)}[||s_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)||_2^2]$$

The probability $\mathbf{x_t}$ given $\mathbf{x_0}$ can be computed by solving the SDE.

Score-Based Generative Model

In summary, we optimize this equation:

$$\mathbf{E}_{t\sim\text{Uniform}(0,\ \text{T}),\mathbf{x}_0\sim p_{\text{data}},\mathbf{x}_t\sim p_t(\mathbf{x}_t|\mathbf{x}_0)}[\lambda(t)||s_\theta(\mathbf{x}(t),t)-\nabla_{\mathbf{x}_t}\log p_{0t}(\mathbf{x}(t)|\mathbf{x}(0))||_2^2]$$

where lambda is a weighting function.

- In general, we design the weighting function to balance the magnitude of the loss across time.
- When $\lambda(t)=g(t)^2$, the score-based loss function is equal to DDPM loss function.

$$\begin{aligned}
p_{0t}(\mathbf{x}(t)|\mathbf{x}(0)) &= \mathcal{N}(\mu,\sigma^2\mathbf{I}) \\
\nabla_{\mathbf{x}(t)}\log p_{0t}(\mathbf{x}(t)|\mathbf{x}(0)) &= \nabla_{\mathbf{x}(t)}\log\frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{1}{2}\frac{(x(t)-\mu)^2}{\sigma^2}) \\
&= -\frac{1}{2}\nabla_{\mathbf{x}(t)}\frac{(x(t)-\mu)^2}{\sigma^2} \\
&= -\frac{(x(t)-\mu)}{\sigma^2} \\
&= -\frac{\epsilon}{\sigma}, \qquad \mathbf{x}(t)=\mu+\sigma\epsilon
\end{aligned}$$

Score-Based Generative Model

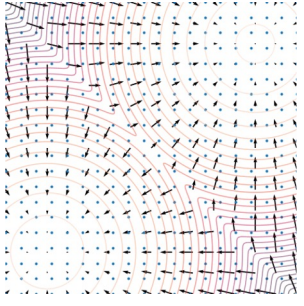
Inference consists of two steps:

1. **Predictor Step**
   a. Sampling $\mathbf{x}(t+\Delta t) \sim p_{t+\Delta t}(\mathbf{x})$ from $\mathbf{x}(t) \sim p_t(\mathbf{x})$ using any numerical SDE solver such as Euler-Maruyama.

$$\begin{aligned}\Delta\mathbf{x} &\leftarrow [\mathbf{f}(\mathbf{x}, t) - g^2(t) s_\theta(\mathbf{x}, t)]\Delta t + g(t)\sqrt{|\Delta t|}\epsilon_t \\ \mathbf{x} &\leftarrow \mathbf{x} + \Delta\mathbf{x} \\ t &\leftarrow t + \Delta t\end{aligned}$$

1. **Corrector Step**
   a. Refine $\mathbf{x}(t+\Delta t) \sim p_{t+\Delta t}(\mathbf{x})$ from predictor step using MCMC procedure such as Langevin dynamics

$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + s_\theta(\mathbf{x}, t)\Delta t + \sqrt{2\Delta t}\epsilon$$



Score-Based Generative Model

Algorithm 3 Training

repeat
  $x_0 \sim q(x_0)$
  $t \sim \text{Uniform}(\{1, \dots, T\})$
  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $x_t \leftarrow \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$
  Take gradient descent step on $\nabla_\theta \lambda(t) || s_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) ||_2^2$
until converged

Algorithm 4 Sampling

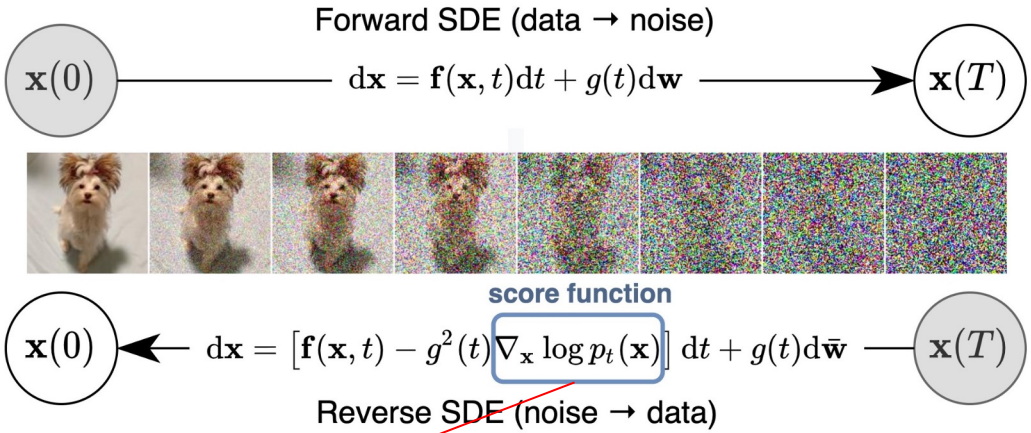
$x_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  Predictor Step (x_t, t)
  Corrector Step (x_t, t)
end for
return x_0

Further research on diffusion

Control
Speed
Architecture

How to control the generating process?

We can simply adjust score function:



$$\nabla_{\mathbf{x}} \log p_t(\mathbf{x}|\mathbf{y}) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) + w\nabla_{\mathbf{x}} \log p_t(\mathbf{y}|\mathbf{x})$$

Diffusion model

Discriminative model

log p(y) disappears because it does not depend on x

How to control the generating process?

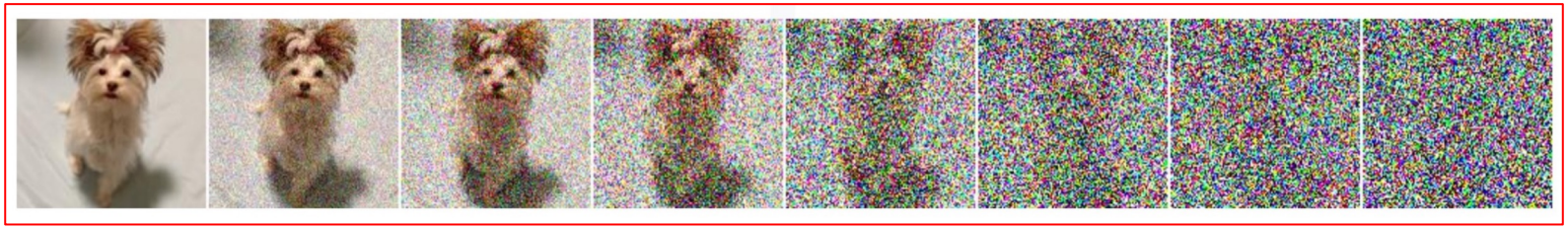
$$\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) = -\frac{\epsilon(x)}{\sigma}$$

To guide the diffusion model, we need a classifier model to acquire the updating direction, as shown in the following equation.

$$\nabla_{\mathbf{x}} \log p_t(\mathbf{x}|\mathbf{y}) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) + w \nabla_{\mathbf{x}} \log p_t(\mathbf{y}|\mathbf{x})$$

Aauxiliary classifier

However, we cannot directly employ the off-the-shelf model because the model must accommodate to predict the target given noisy image (x_t) for any t, which is a drawback of this approach.



classifier input

Classifier Free Guidance

$$\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) = -\frac{\epsilon(x)}{\sigma}$$

In classifier free paper, it propose to eliminate the classifier model from the following equation

Instead of relying on additional classifier, we can parameterize as an interpolation between predictors from diffusion model with and without labels.

$$\nabla_{\mathbf{x}} \log p_t(\mathbf{x}|\mathbf{y}) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) + w\nabla_{\mathbf{x}} \log p_t(\mathbf{y}|\mathbf{x})$$

$$\begin{aligned}
\tilde{\epsilon}(\mathbf{x}, \mathbf{y}) &= \epsilon(\mathbf{x}) - w\nabla_{\mathbf{x}} \log p_t(\mathbf{y}|\mathbf{x}) \\
&= \epsilon(\mathbf{x}) - w[\nabla_{\mathbf{x}} \log p_t(\mathbf{x}|\mathbf{y}) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x})] \\
&= \epsilon(\mathbf{x}) - w[-\epsilon(\mathbf{x}, \mathbf{y}) + \epsilon(\mathbf{x})] \\
&= (1 - w)\epsilon(\mathbf{x}) + w\epsilon(\mathbf{x}, \mathbf{y})
\end{aligned}$$

We model these

https://arxiv.org/abs/2207.12598 **Classifier-Free Diffusion Guidance 2022**

Classifier free guidance

Algorithm 1 Joint training a diffusion model with classifier-free guidance

Require: p_{uncond}: probability of unconditional training

repeat
  $(\mathbf{x}, \mathbf{c}) \sim p(\mathbf{x}, \mathbf{c})$ ▷ Sample data with conditioning from the dataset
  $\mathbf{c} \leftarrow \varnothing$ with probability p_{uncond} ▷ Randomly discard conditioning to train unconditionally
  $\lambda \sim p(\lambda)$ ▷ Sample log SNR value
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\mathbf{z}_\lambda = \alpha_\lambda \mathbf{x} + \sigma_\lambda \boldsymbol{\epsilon}$ ▷ Corrupt data to the sampled log SNR value
  Take gradient step on $\nabla_\theta \left\| \boldsymbol{\epsilon}_\theta(\mathbf{z}_\lambda, \mathbf{c}) - \boldsymbol{\epsilon} \right\|^2$ ▷ Optimization of denoising model
until converged

Algorithm 2 Conditional sampling with classifier-free guidance

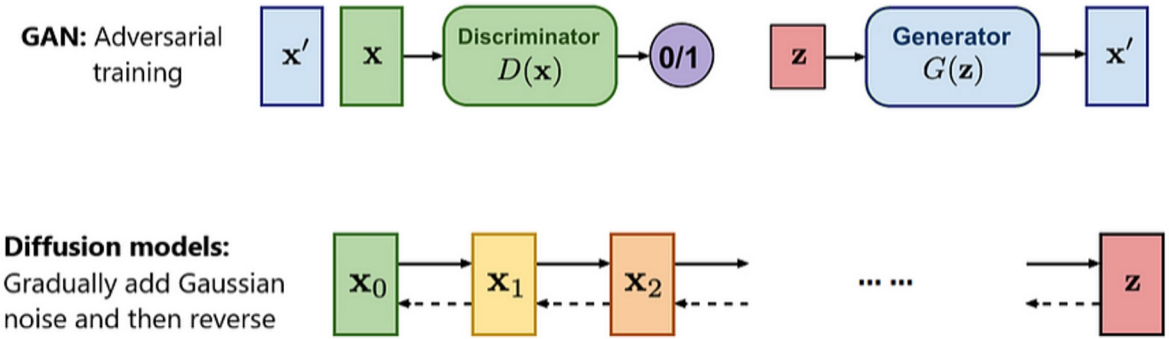
Require: w: guidance strength
Require: $\mathbf{c}$: conditioning information for conditional sampling
Require: $\lambda_1, \ldots, \lambda_T$: increasing log SNR sequence with $\lambda_1 = \lambda_{\min}, \lambda_T = \lambda_{\max}$

$\mathbf{z}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = 1, \ldots, T$ **do**
  ▷ Form the classifier-free guided score at log SNR λ_t
  $\tilde{\boldsymbol{\epsilon}}_t = (1 + w)\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, \mathbf{c}) - w\boldsymbol{\epsilon}_\theta(\mathbf{z}_t)$
  ▷ Sampling step (could be replaced by another sampler, e.g. DDIM)
  $\tilde{\mathbf{x}}_t = (\mathbf{z}_t - \sigma_{\lambda_t} \tilde{\boldsymbol{\epsilon}}_t)/\alpha_{\lambda_t}$
  $\mathbf{z}_{t+1} \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}_{\lambda_{t+1}|\lambda_t}(\mathbf{z}_t, \tilde{\mathbf{x}}_t), (\tilde{\sigma}^2_{\lambda_{t+1}|\lambda_t})^{1-v} (\sigma^2_{\lambda_t|\lambda_{t+1}})^v)$ if $t < T$ else $\mathbf{z}_{t+1} = \tilde{\mathbf{x}}_t$
end for
return $\mathbf{z}_{T+1}$

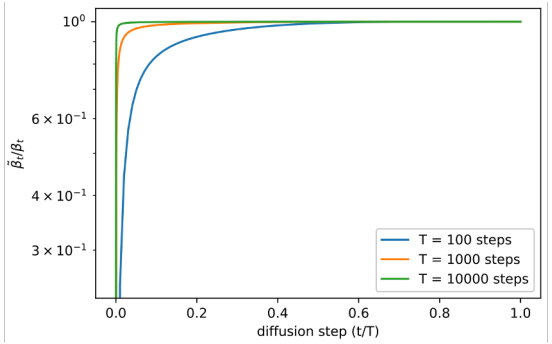
Open Research Question in Diffusion

The main problem with diffusion model is slow Inference time.

- It requires several diffusion step to obtain high-quality sample. Conversely other approach such as GAN only take one inference step.
- The diffusion model cannot be parallelized due to its Markovian nature. (Cannot utilize GPU effectively)



How can we shorten the diffusion steps?



During training process, we train diffusion model to perform N diffusion step.

Is it possible to perform only S step (S < N) when inference? (with a fixed variance)

- No, it is inappropriate to do so. As mentioned earlier, the gap between the upperbound and lowerbound becomes larger and larger when the diffusion step is small.

Therefore, in the Improved Denoising Diffusion Probabilistic Model, they proposed **learning the variance of the reverse process**. Moreover, the variance was parameterized as an interpolation between the upperbound and lowerbound in the log domain, due to the infinitesimal range of reasonable values.

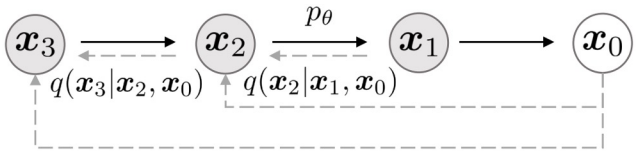
$$\Sigma_\theta(x_t, t) = \exp(v_\theta \log \beta_t + (1 - v_\theta) \log \tilde{\beta}_t)$$

Unweighted loss (vanilla version) is still a main source for mean and the weighted version is used to train variance specifically.

$$L_{\text{hybrid}} = \underbrace{L_{\text{simple}}}_{\text{unweighted loss}} + \lambda \underbrace{L_{\text{vlb}}}_{\text{weighted loss}}$$

What about the sampling method? Can we change?

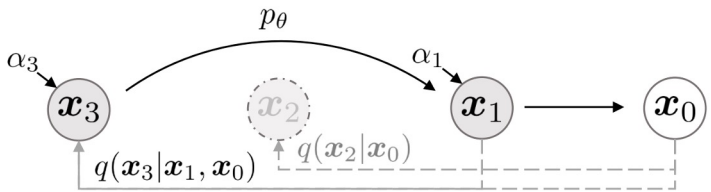
In Denoising Diffusion Implicit Model, the forward process is modified to be non-Markovian.



As a result, DDIM introduces a **new sampling method** while retaining the same loss as DDPM (use as pre-trained weights).

This sampling method enables the selection of only a subset of the sequence to expedite the inference process.

$$q_\sigma(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \sqrt{\alpha_{t-1}}x_0 + \sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \frac{x_t - \sqrt{\alpha_t}x_0}{\sqrt{1-\alpha_t}}, \sigma_t^2\mathbf{I})$$



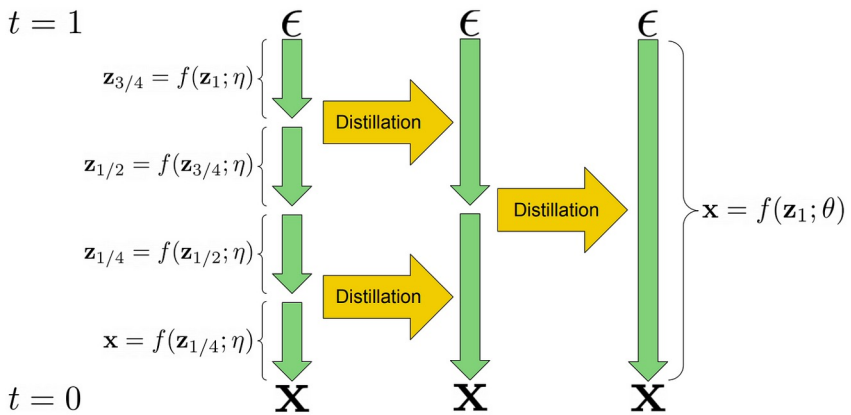
Others method like this: DPM solver

https://arxiv.org/abs/2010.02502 https://arxiv.org/abs/2206.00927

Knowledge Distillation

The paper propose a method to transfer knowledge from teacher model to student model so that student require fewer diffusion step than teacher model.

1. Training a teacher model similar to vanilla diffusion model
2. Transferring knowledge from teacher model to student model
    a. One step in student model equivalent to two steps in teacher model.
3. Set student model as teacher model and go back to step 2.



https://arxiv.org/abs/2202.00512 **Progressive Distillation for Fast Sampling of Diffusion Models** 2022

On Distillation of Guided Diffusion Models

Extend the idea of knowledge distillation from the previous slide on the guided diffusion models

1. Stage-one distillation
    a. Transfer the knowledge from the classifier-free diffusion model to a student model, enabling the student model to predict interpolated noise in a single pass.

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

$$\mathbb{E}_{w\sim U[w_{\min},w_{\max}],t\sim U[0,1],\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[w(\lambda_t)||\hat{\mathbf{x}}_{\eta_1}(\mathbf{z}_t,w) - \hat{\mathbf{x}}^{w}_{c,\theta}(\mathbf{z}_t)||_2^2\right]$$

1. Stage-two distillation
    a. Use the student model from step 1 as a teacher model and follow the same procedure as in the previous slide.

Control Diffusion model through Text

Conditioning on text provides more flexibility than using a class label. Moreover, it can perform zero-shot learning.

DALL-E 2 proposed to be conditioned on CLIP embeddings. So, what is CLIP?

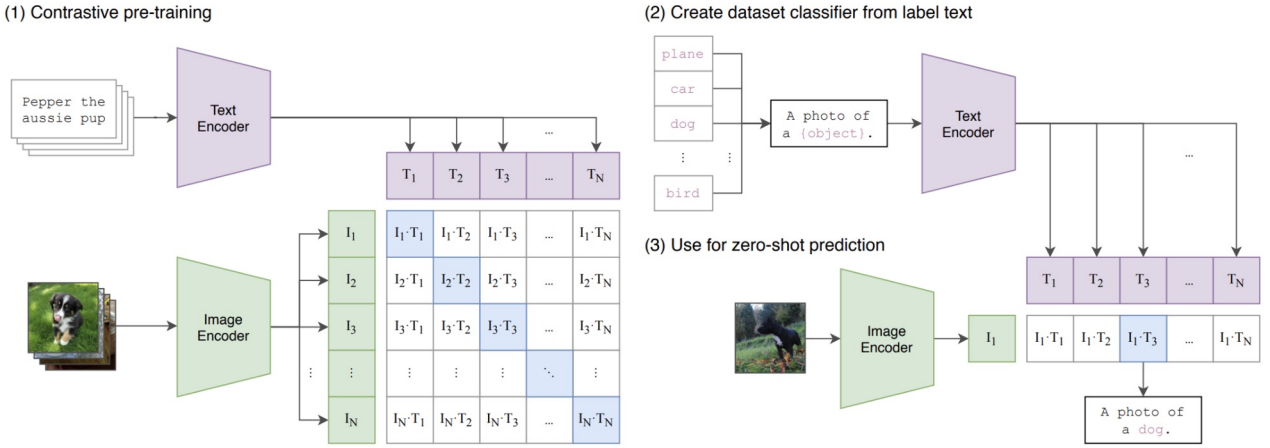
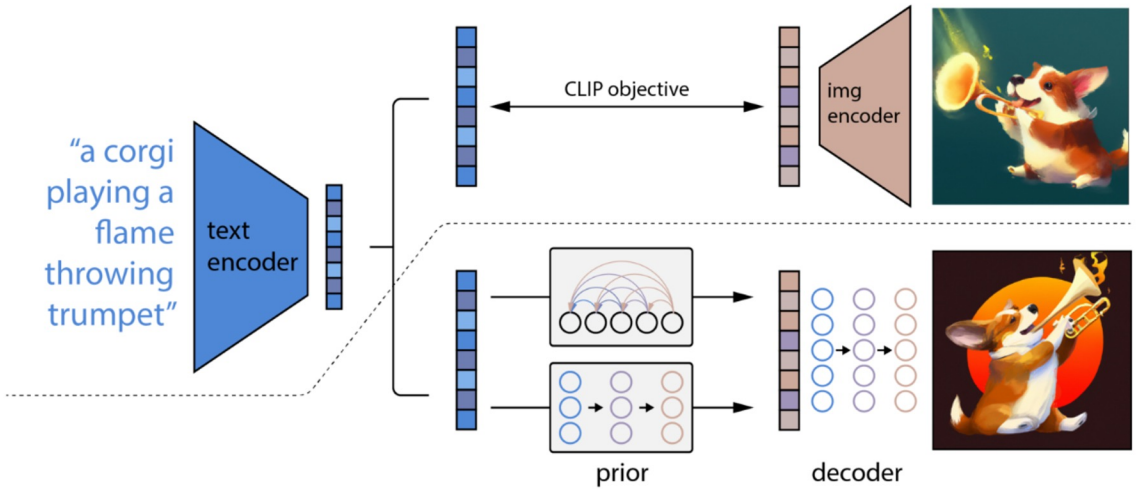


Figure 1. Summary of our approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes.

Control Diffusion model through Text

Step-by-step workflow of DALL-E 2

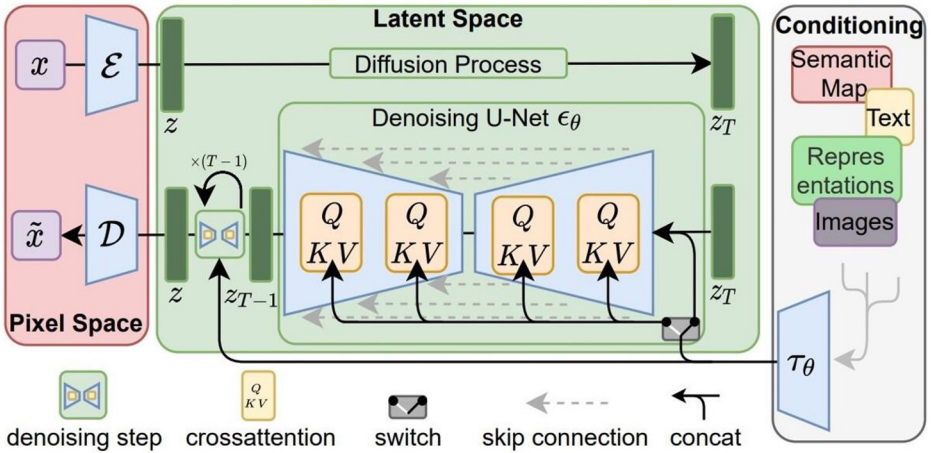
1. Text encoded using CLIP (CLIP Text Embeddings)
2. CLIP text embeddings converted to CLIP image embeddings via Prior (Autoregressive or Diffusion Model)
3. CLIP image embeddings to final image generation



Beyond Text Modality

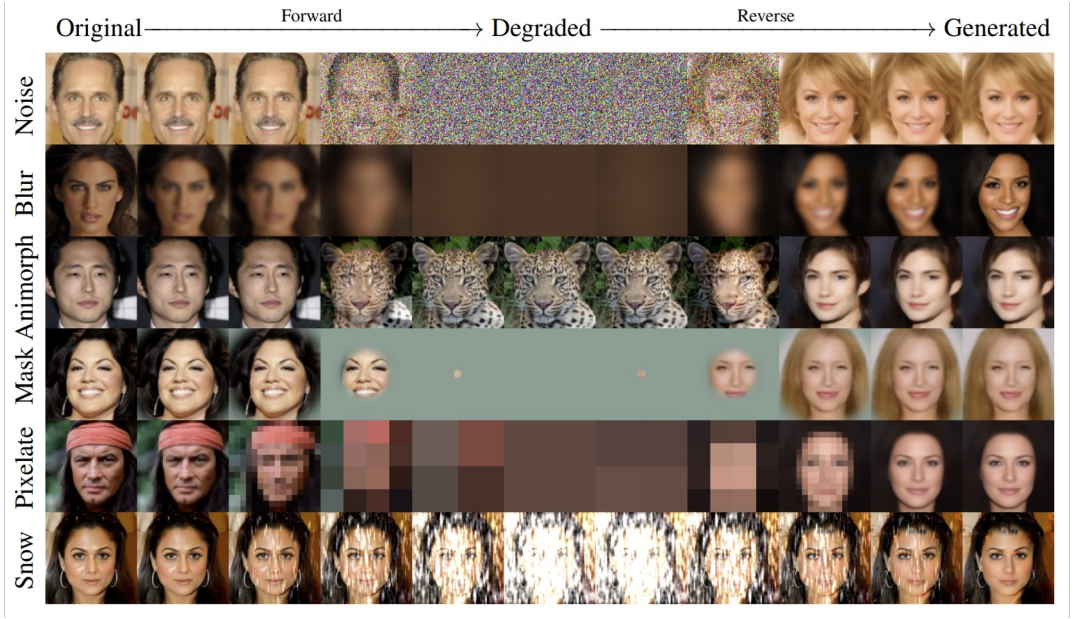
Stable diffusion proposed cross-attention, which is effective for learning attention-based models for various input modalities. It turns diffusion models into powerful and flexible generators for general conditioning inputs.

Moreover, it operates in an embedding space, significantly reducing computational requirements



Does it need to be Gaussian Noise?

In the cold diffusion paper, they showed that it is not necessary to use Gaussian noise but any image transformation. This raises questions about the limits of our theoretical understanding of diffusion models.



https://arxiv.org/abs/2208.09392 2022 **Cold Diffusion: Inverting Arbitrary Image Transforms Without Noise**

DIT (e.g. Attention is all you need ver. Diffusion)

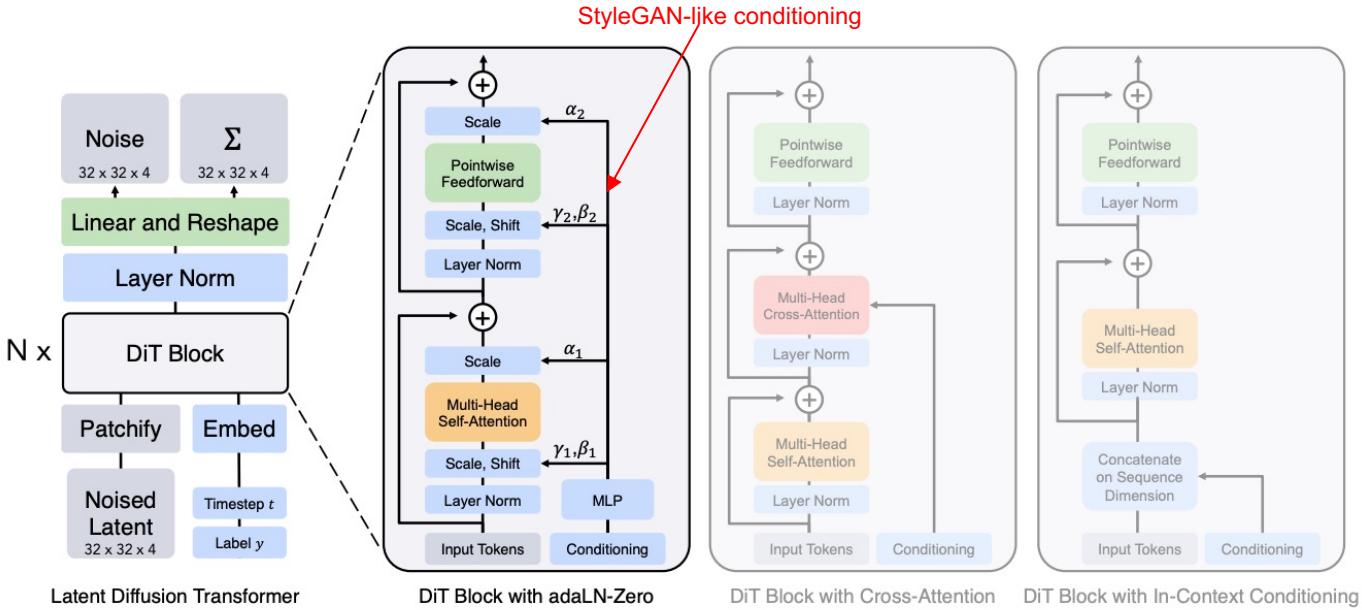


Figure 3. **The Diffusion Transformer (DiT) architecture.** *Left:* We train conditional latent DiT models. The input latent is decomposed into patches and processed by several DiT blocks. *Right:* Details of our DiT blocks. We experiment with variants of standard transformer blocks that incorporate conditioning via adaptive layer norm, cross-attention and extra input tokens. Adaptive layer norm works best.

https://arxiv.org/abs/2212.09748 Scalable Diffusion Models with Transformers (DiT) 2023

Parameter Efficient Fine-Tuning

Emerging Trends in Deep Learning Model

While scaling up deep learning models, several emerging properties occur, such as zero-shot or few-shot in LLM.

However, to finetune the model to a particular task is very cumbersome. Several researchers have been working on the model compressions:

1. Prefix tuning
   a. Append a learnable token in front of the input
2. Low-Rank Adaptation (LoRA)
   a. Represent an adaptation weight (gradient) with a low-rank matrix.
3. Adapter Module
   a. Insert a small number of layers that are relatively small compared to the entire model.

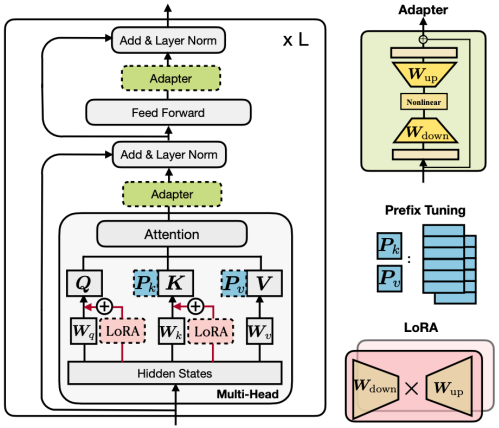
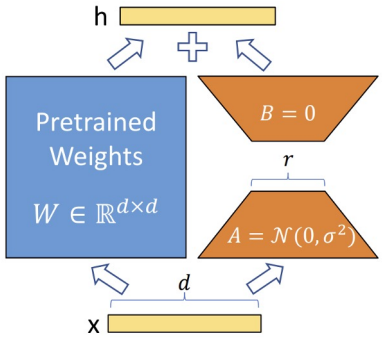
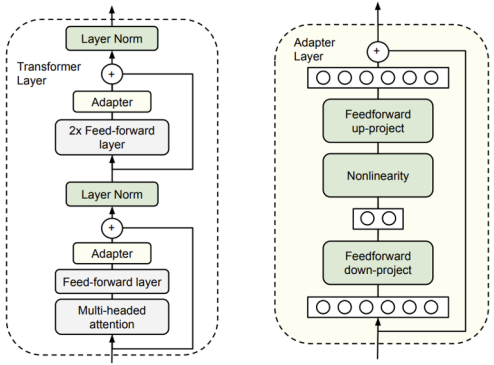


Figure 1: Illustration of the transformer architecture and several state-of-the-art parameter-efficient tuning methods. We use blocks with dashed borderlines to represent the added modules by those methods.





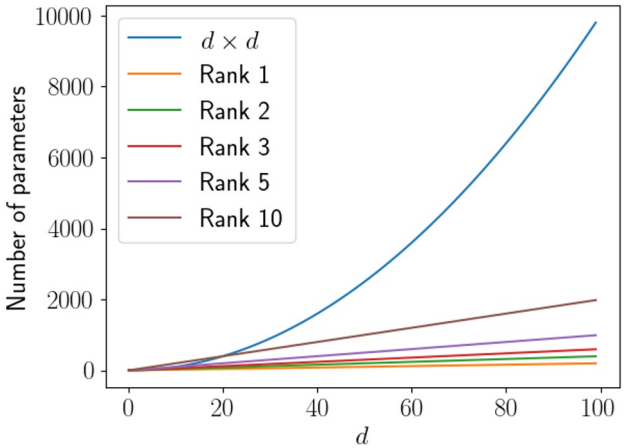
https://openreview.net/forum?id=0RDcd5Axok

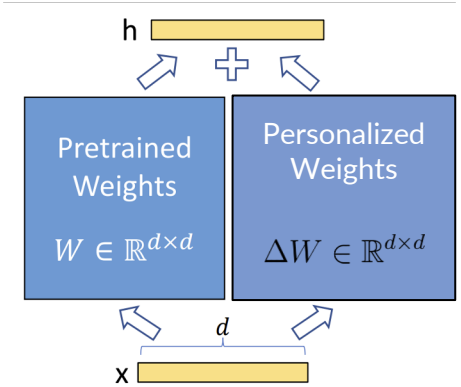
Low-Rank Adaptation

If we want to deploy a personalized large language model, we need to separately deploy each model for each user.

LoRA compress this personalized weights using low-rank decomposition

$$\Delta W = BA \qquad B \in \mathbb{R}^{d \times r} A \in \mathbb{R}^{r \times d}$$





Low-Rank Adaptation

In the initialization process, we use a random Gaussian initialization for A and zero for B.

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

0

Therefore, the model is initialized to be identical to the pretrained weights.

In addition, the paper proposes to scale low-rank matrices by alpha/r, claiming that it helps reduce the need to adjust hyperparameters when varying r.

Can be seen as Learning Rate

$$h = W_0 x + \frac{\alpha}{r} BAx$$

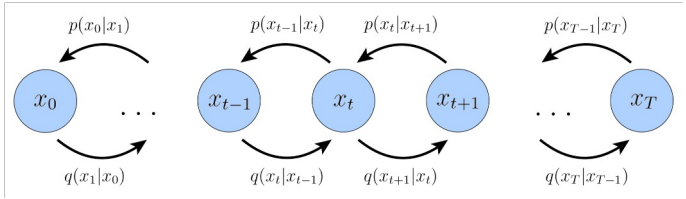
https://arxiv.org/abs/2106.09685 LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS 2021

Homework

Tutorial
Homework

Summary

- Diffusion vs Flow
  - They are quite similar (and can be shown to be the same thing under certain settings) except that flow requires invertible layers
    - Flow -> exact max likelihood
    - Diffusion -> solve via ELBO
- Serveral formulation of diffusions
  - DDPM vs Score-based



Appendix
Splitting diffusion loss into 3 terms

1.2.1 Loss Function

As a result, $\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log q(x_{1:T}|x_0) - \log p(x_{0:T})]$ is optimized instead of $\mathbf{E}_{x_0\sim q(x_0)}[-\log p(x_0)]$ which is tractable to compute. We further examine this equation and split it into 3 terms.
Consider $\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log q(x_{1:T}|x_0) - \log p(x_{0:T})]$

$$\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log q(x_{1:T}|x_0) - \log p(x_{0:T})]$$

$$\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log \prod_{t=1}^{T} q(x_t|x_{t-1}) - \log p(x_T) \prod_{t=1}^{T} p(x_{t-1}|x_t)]$$

$$\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log q(x_1|x_0) + \log \prod_{t=2}^{T} q(x_t|x_{t-1}) - \log p(x_T) \prod_{t=1}^{T} p(x_{t-1}|x_t)]$$

Forward process is Markov process, then $q(x_t|x_{t-1}) = q(x_t|x_{t-1}, x_0)$

$$\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log q(x_1|x_0) + \log \prod_{t=2}^{T} q(x_t|x_{t-1}, x_0) - \log p(x_T) \prod_{t=1}^{T} p(x_{t-1}|x_t)]$$

$$\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log q(x_1|x_0) - \log p(x_T) + \log \prod_{t=2}^{T} q(x_t|x_{t-1}, x_0) - \log \prod_{t=1}^{T} p(x_{t-1}|x_t)]$$

$$\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log \frac{q(x_1|x_0)}{p(x_T)} + \log \prod_{t=2}^{T} q(x_t|x_{t-1}, x_0) - \log \prod_{t=1}^{T} p(x_{t-1}|x_t)]$$

$$\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log \frac{q(x_1|x_0)}{p(x_T)} + \log \prod_{t=2}^{T} \frac{q(x_t, x_{t-1}, x_0)}{q(x_{t-1}, x_0)} - \log \prod_{t=1}^{T} p(x_{t-1}|x_t)]$$

$$\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log \frac{q(x_1|x_0)}{p(x_T)} + \log \prod_{t=2}^{T} \frac{q(x_{t-1}|x_t, x_0)q(x_t, x_0)}{q(x_{t-1}, x_0)} - \log \prod_{t=1}^{T} p(x_{t-1}|x_t)]$$

$$\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log \frac{q(x_1|x_0)}{p(x_T)} + \log \prod_{t=2}^{T} \frac{q(x_{t-1}|x_t, x_0)q(x_t, x_0)}{q(x_{t-1}, x_0)} - \log \prod_{t=1}^{T} p(x_{t-1}|x_t)]$$

$$\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log \frac{q(x_1|x_0)}{p(x_T)} + \log \prod_{t=2}^{T} \frac{q(x_{t-1}|x_t, x_0)q(x_t, x_0)}{q(x_{t-1}, x_0)} - \log \prod_{t=1}^{T} p(x_{t-1}|x_t)]$$

$$\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log \frac{q(x_1|x_0)}{p(x_T)} + \log \prod_{t=2}^{T} q(x_{t-1}|x_t, x_0) - \log \prod_{t=1}^{T} p(x_{t-1}|x_t) + \log \prod_{t=2}^{T} \frac{\cancel{q(x_t, x_0)}^{q(x_T,x_0)}}{\cancel{q(x_{t-1}, x_0)}^{q(x_1,x_0)}}]$$

$$\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log \frac{q(x_1|x_0)}{p(x_T)} + \log \prod_{t=2}^{T} \frac{q(x_{t-1}|x_t, x_0)}{p(x_{t-1}|x_t)} - \log p(x_0|x_1) + \log \frac{q(x_T|x_0)}{q(x_1|x_0)}]$$

$$\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log \frac{q(x_1|x_0)q(x_T|x_0)}{p(x_T)q(x_1|x_0)} + \log \prod_{t=2}^{T} \frac{q(x_{t-1}|x_t, x_0)}{p(x_{t-1}|x_t)} - \log p(x_0|x_1)]$$

$$\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\log \frac{q(x_T|x_0)}{p(x_T)} + \log \prod_{t=2}^{T} \frac{q(x_{t-1}|x_t, x_0)}{p(x_{t-1}|x_t)} - \log p(x_0|x_1)]$$

$$\mathbf{E}_{x_{0:T}\sim q(x_{0:T})}[\underbrace{D_{\mathrm{KL}}(q(x_T|x_0)||p(x_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{D_{\mathrm{KL}}(q(x_{t-1}|x_t, x_0)||p(x_{t-1}|x_t))}_{L_{t-1}} \underbrace{- \log p(x_0|x_1)}_{L_0}]$$

Appendix
Getting $q(x_{t-1}|x_t, x_0)$

From $q(x_t|x_{t-1}) = \mathcal{N}(\sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)\mathbf{I})$ and $q(x_t|x_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathbf{I})$, and let $\beta_t = 1 - \alpha_t$

$$q(x_{t-1}|x_t, x_0) = \frac{1}{\sqrt{\beta_t}\sqrt{2\pi}} \frac{\sqrt{1-\bar{\alpha}_t}}{\sqrt{1-\bar{\alpha}_{t-1}}} \exp(-\frac{1}{2}(\underbrace{\frac{(x_t - \sqrt{\alpha_t}x_{t-1})^2}{\beta_t} \cdots}_{\text{Consider this term}})$$

Consider the term inside the exponential function.

$$\frac{(x_t - \sqrt{\alpha_t}x_{t-1})^2}{\beta_t} + \frac{(x_{t-1} - \sqrt{\bar{\alpha}}_{t-1}x_0)^2}{1-\bar{\alpha}_{t-1}} - \frac{(x_t - \sqrt{\bar{\alpha}}_t x_0)^2}{1-\bar{\alpha}_t}$$

$$\underbrace{(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}})}_{a} x_{t-1}^2 - \underbrace{(\frac{2\sqrt{\alpha_t}}{\beta_t}x_t + \frac{2\sqrt{\bar{\alpha}}_{t-1}}{1-\bar{\alpha}_{t-1}})}_{b} x_{t-1} + \underbrace{\frac{x_t^2}{\beta_t} + \frac{\bar{\alpha}_{t-1}x_0^2}{1-\bar{\alpha}_{t-1}} - \frac{(x_t - \sqrt{\bar{\alpha}}_t x_0)^2}{1-\bar{\alpha}_t}}_{c}$$

From $ax^2 + bx + c = a(x + \frac{b}{2a})^2 + \frac{4ac-b^2}{4a}$

$$\frac{(x_{t-1} - (\frac{\sqrt{\alpha}_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t + \frac{\sqrt{\bar{\alpha}}_{t-1}\beta_t}{1-\bar{\alpha}_t}x_0))^2}{\frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}}$$

Therefore $q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}_t, \tilde{\beta}_t\mathbf{I})$

$$\tilde{\mu}_t = \frac{\sqrt{\alpha}_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t + \frac{\sqrt{\bar{\alpha}}_{t-1}\beta_t}{1-\bar{\alpha}_t}x_0$$

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