Carnegie Mellon University

Lecture 4: Score-based Models

Yutong (Kelly) He

10-799 Diffusion & Flow Matching, Jan 20th, 2026

Many figures derived from Yang Song's https://yang-song.net/blog/2021/score/

Modal aws

Quiz time!

10 minutes

Closed-book

Pen & Paper

Housekeeping Announcements

- Homework 1 is out! https://kellyyutonghe.github.io/10799S26/homework/
    - Q6 (Alternative Parameterization) is now an optional extra credit question!
    - Due date: 1/24 Sat, Late Due date: 1/26 Mon
    - Training models takes time! Start early!
- Office Hours are announced:
    - Kelly is hosting OH
        - In-person: Wednesdays 1:00 PM - 2:00 PM, Gates 8th Floor common area near the printer
        - Virtual: Fridays 11:00 AM - 12:00 PM, Discord
    - Krish is hosting OH Tuesdays 4:00 PM - 5:00 PM, Gates 8th Floor common area near the printer
        - Extra OH this week: Friday 3 – 4 PM same location
- We shall have our Quiz 2 next class (1/22 Thurs)

Previously on diffusion models…

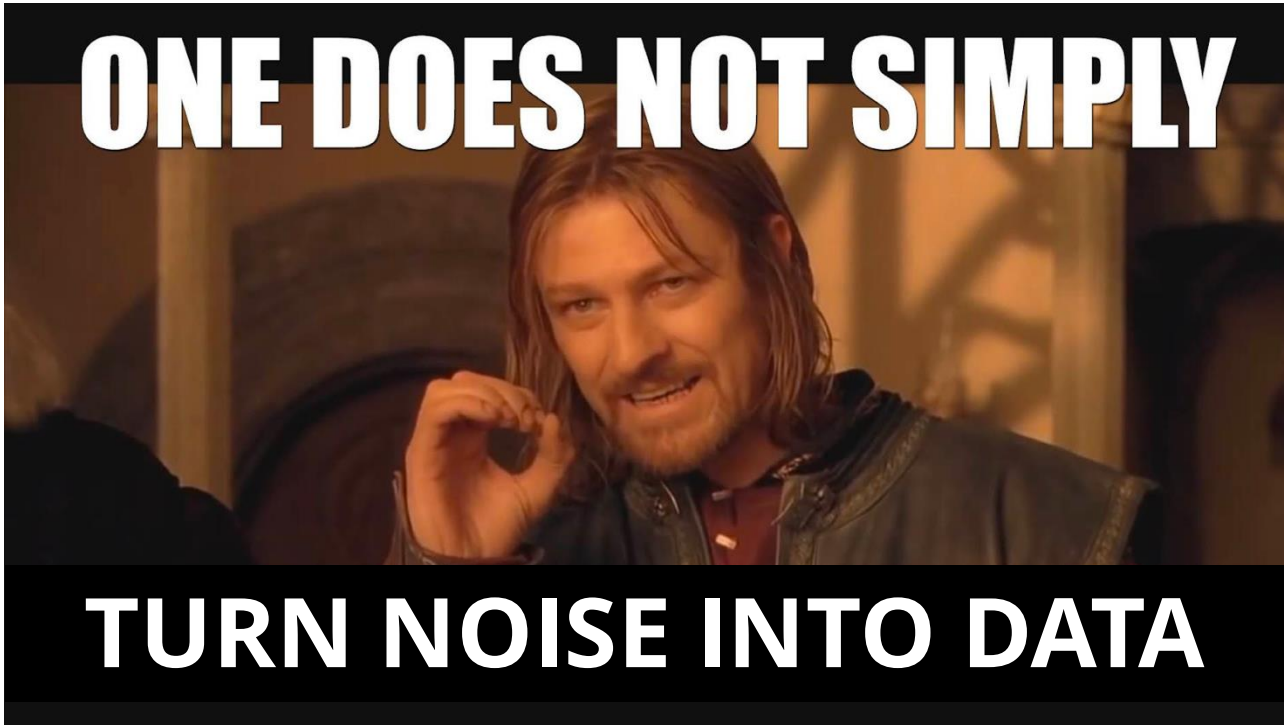


Image from The Lord of The Rings

Diffusion's way to turn noise into data

Forward process (adding noise)

Data x_0 $t = 0$ | x_1 $t = 1$ | x_2 $t = 2$ | x_3 $t = 3$ | Noise x_4 $t = 4 = T$

Reserve process (denoising)

Carnegie Mellon University

Cat stolen from Chieh-Hsin (Jesse) Lai

Diffusion's forward process

$$q(x_t|x_{t-1}) = N(\sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$
$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon, \qquad \epsilon \sim N(0, I)$$

Data x_0
$t = 0$

x_1
$t = 1$

x_2
$t = 2$

x_3
$t = 3$

Noise x_4
$t = 4 = T$

$$\alpha_t = 1 - \beta_t$$
$$\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$$

$$q(x_t|x_0) = N(\sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$$
$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, \qquad \epsilon \sim N(0, I)$$

Carnegie Mellon University

Cat stolen from Chieh-Hsin (Jesse) Lai

Diffusion's reverse process

$$p(x_{t-1}|x_t) = N(\mu_\theta(x_t,t), \Sigma_\theta(x_t,t))$$
$$x_t = \mu_\theta(x_t,t) + C_\theta(x_t,t)\epsilon$$
$$\epsilon \sim N(0,I), \quad C_\theta(x_t,t)C_\theta(x_t,t)^\top = \Sigma_\theta(x_t,t)$$

Data x_0 $t=0$ | x_1 $t=1$ | x_2 $t=2$ | x_3 $t=3$ | Noise x_4 $t=4=T$

If we fix the variance

$$p_\theta(x_{t-1}|x_t) = N(\mu_\theta(x_t,t), \sigma_t^2 I)$$

In addition, we can also write the mean w.r.t. the noise prediction

$$\mu_\theta(x_t,t) = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t,t)\right)$$

=>

Training: $||\epsilon - \epsilon_\theta(x_t,t)||^2$

Sampling: $x_{t-1} = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t,t)\right) + \sigma_t\epsilon$

Easier to train!

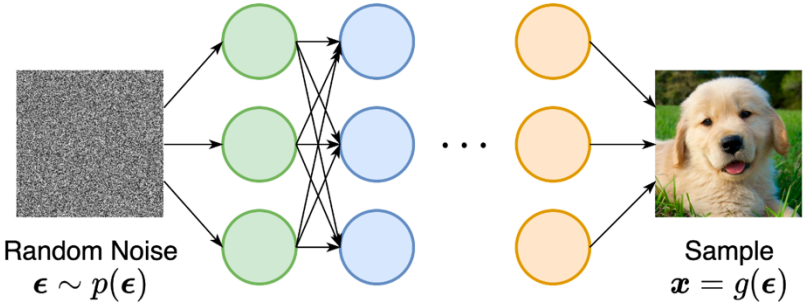
Carnegie Mellon University

Cat stolen from Chieh-Hsin (Jesse) Lai, GIF from Star Trek (1966)

So far we have seen a bunch of generative models…

In general, we can roughly categorize generative models into the following categories

- **Likelihood Based:** Autoregressive models, variational autoencoders (VAE), normalizing flow, energy-based models (EBM)
- **Likelihood Free:** Generative adversarial networks (GAN)



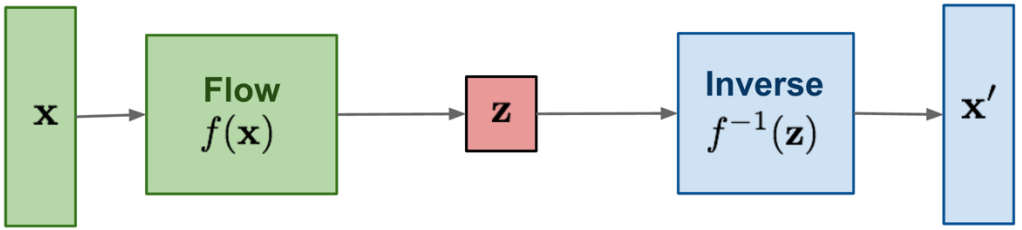
Directly sampling from P(X) is usually hard because they are usually complicated! But **sampling from a simpler distribution** (eg. a Gaussian) is easy!

Two more generative models

In general, we can roughly categorize generative models into the following categories

- **Likelihood Based:** Autoregressive models ✅, variational autoencoders (VAE) ✅, normalizing flow❓, energy-based models (EBM) ❓

Normalizing flows:



Energy-based models:

$$p_\theta(x) = \frac{1}{\int \exp(f_\theta(x))dx} \exp(f_\theta(x))$$

Partition function

$$= \frac{1}{Z(\theta)} \exp(f_\theta(x))$$

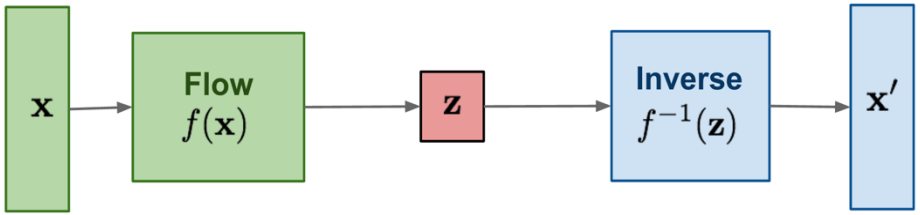
Carnegie Mellon University

Image from Lilian Weng https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Likelihood is difficult to calculate

- **Autoregressive models:** you need to break everything up by chain rule and calculate components **one by one**
- **VAEs:** you are still using a surrogated loss (ELBO) after all the hard math
- **Normalizing flows:** need weird architectures to make sure everything is invertible

Normalizing flows:



Likelihood is difficult to calculate

- **Autoregressive models:** you need to break everything up by chain rule and calculate components **one by one**
- **VAEs:** you are still using a surrogated loss (ELBO) after all the hard math
- **Normalizing flows:** need weird architectures to make sure everything is invertible
- **EBMs:** partition function is generally intractable

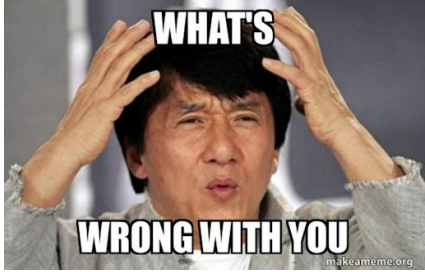
Energy-based models:

$$p_\theta(x) = \frac{1}{\int \exp(f_\theta(x))dx} \exp(f_\theta(x))$$

Partition function

$$= \frac{1}{Z(\theta)} \exp(f_\theta(x))$$

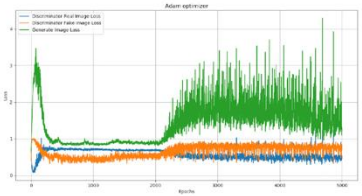
Likelihood is difficult to calculate – Let's go likelihood free?



- **Likelihood Free:** Generative adversarial networks (GAN)

=> very unstable & suffers from mode collapse

- **Theorem (informal):** If the generator updates are made in function space and discriminator is optimal at every step, then the generator is guaranteed to converge to the data distribution
- **Unrealistic assumptions**!
- In practice, the generator and discriminator loss keeps oscillating during GAN training



Source: Mirantha Jayathilaka

- No robust stopping criteria in practice (unlike MLE)

- GANs are notorious for suffering from **mode collapse**
- Intuitively, this refers to the phenomena where the generator of a GAN collapses to one or few samples (dubbed as "modes")

Arjovsky et al., 2017

Figures from Stefano Ermon Stanford CS 236

Likelihood is difficult to calculate – Let's go likelihood free?

- **Likelihood Free:** Generative adversarial networks (GAN)

  => very unstable & suffers from mode collapse

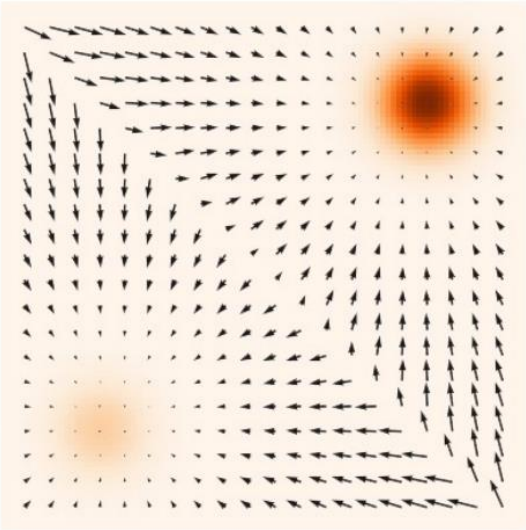
Better way to avoid directly calculating the likelihood?

If we just want to sample…

Do we really need $p_\theta(x) \approx p_{data}(x)$ to **sample** from the model?

- Idea: Stochastic **gradient** descent (or I guess ascent) to maximize **log likelihood** in the **data space**
- So we only need $\nabla_x \log p_{data}(x)$

score function



Carnegie Mellon University

Figure from Yang Song https://yang-song.net/blog/2021/score/

Score-based Models

Now we just need to train a model to estimate the score function

$$s_\theta(x) \approx \nabla_x \log p_{data}(x)$$

This can be done by minimizing an L2

$$E_{x \sim p_{data}(x)}[\left|\left|\nabla_x \log p_{data}(x) - s_\theta(x)\right|\right|^2]$$

Fisher divergence

This is nice because

- No more intractable partition function
- No more adversarial training
- No more weird architecture
- No breaking up with chain rule => can generate everything all at once

How to train your score model?

But how to do get this ground truth score $\nabla_x \log p_{data}(x)$?

Turns out you don't have to!

$$E_{x\sim p_{data}(x)}[\left|\left|\nabla_x \log p_{data}(x) - s_\theta(x)\right|\right|^2]$$

$$= E_{x\sim p_{data}(x)}\left[\left|\left|\nabla_x \log p_{data}(x)\right|\right|^2\right] + E_{x\sim p_{data}(x)}\left[\left|\left|s_\theta(x)\right|\right|^2\right]$$

$$- 2E_{x\sim p_{data}(x)}[\langle \nabla_x \log p_{data}(x)\,, s_\theta(x)\rangle]$$

constant w.r.t. θ

Because $\left|\left|x-y\right|\right|^2 = \langle x-y, x-y\rangle = \langle x, x\rangle + \langle y, y\rangle - 2\langle x, y\rangle$

How to train your score model?

$$E_{x\sim p_{data}(x)}[\left|\left|\nabla_x \log p_{data}(x) - s_\theta(x)\right|\right|^2]$$

$$= E_{x\sim p_{data}(x)}\left[\left|\left|s_\theta(x)\right|\right|^2\right] - 2E_{x\sim p_{data}(x)}[\langle \nabla_x \log p_{data}(x)\,, s_\theta(x)\rangle] + C$$

$$E_{x\sim p_{data}(x)}[\langle \nabla_x \log p_{data}(x)\,, s_\theta(x)\rangle] = \int p_{data}(x)\langle \nabla_x \log p_{data}(x)\,, s_\theta(x)\rangle dx$$

$$= \int p_{data}(x)\left\langle \frac{\nabla_x p_{data}(x)}{p_{data}(x)}, s_\theta(x)\right\rangle dx$$

$$= \int \langle \nabla_x p_{data}(x), s_\theta(x)\rangle dx$$

How to train your score model?

$$E_{x\sim p_{data}(x)}[\langle \nabla_x \log p_{data}(x)\,, s_\theta(x)\rangle] = \int \langle \nabla_x p_{data}(x), s_\theta(x)\rangle dx$$

Integration by parts and assume that the boundary vanishes, then we can get

$$= -\int p_{data}(x) \sum_d \frac{\partial s_\theta(x)_{(d)}}{\partial x_{(d)}} dx = -E_{x\sim p_{data}(x)}[\sum_d \frac{\partial s_\theta(x)_{(d)}}{\partial x_{(d)}}]$$

$$= -E_{x\sim p_{data}(x)}[\mathrm{tr}(J_{s_\theta}(x))]$$

trace of Jacobian

Score matching loss

$$E_{x\sim p_{data}(x)}\left[\left|\left|\nabla_x \log p_{data}(x) - s_\theta(x)\right|\right|^2\right]$$

$$= E_{x\sim p_{data}(x)}\left[\left|\left|s_\theta(x)\right|\right|^2\right] - 2E_{x\sim p_{data}(x)}\left[\left\langle \nabla_x \log p_{data}(x)\,, s_\theta(x)\right\rangle\right] + C$$

$$= E_{x\sim p_{data}(x)}\left[\left|\left|s_\theta(x)\right|\right|^2\right] + 2E_{x\sim p_{data}(x)}\left[\mathrm{tr}\left(J_{s_\theta}(x)\right)\right] + C$$

$$L(\theta) = \frac{1}{2}E_{x\sim p_{data}(x)}\left[\left|\left|s_\theta(x)\right|\right|^2\right] + E_{x\sim p_{data}(x)}\left[\mathrm{tr}\left(J_{s_\theta}(x)\right)\right]$$

No need for the ground truth score!

Aapo Hyvärinen. "Estimation of Non-Normalized Statistical Models by Score Matching". JMLR 2005. https://jmlr.org/papers/volume6/hyvarinen05a/hyvarinen05a.pdf

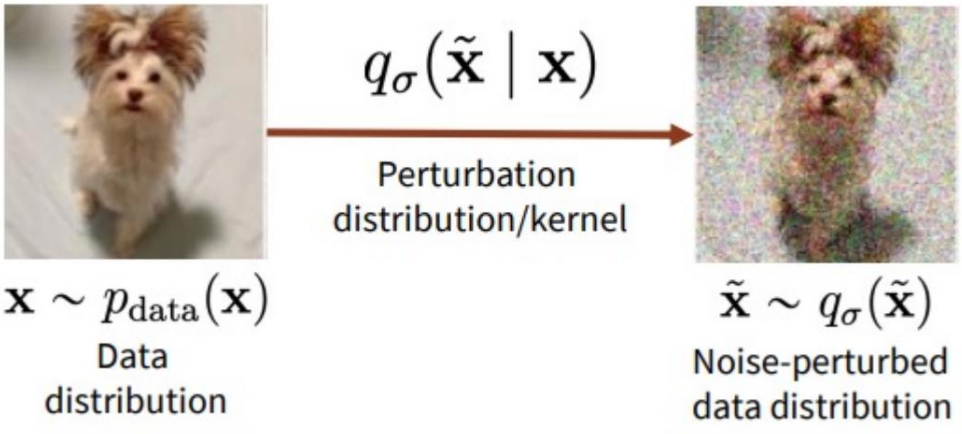
Score matching loss

$$L(\theta) = \frac{1}{2} E_{x \sim p_{data}(x)} \left[\left|\left| s_\theta(x) \right|\right|^2 \right] + E_{x \sim p_{data}(x)} \left[\mathrm{tr}\left(J_{s_\theta}(x) \right) \right]$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} \left|\left| s_\theta\left(x^{(i)}\right) \right|\right|^2 + \mathrm{tr}\left(J_{s_\theta}\left(x^{(i)}\right) \right)$$

Trace of a Jacobian is very very expensive to calculate!

Denoising score matching



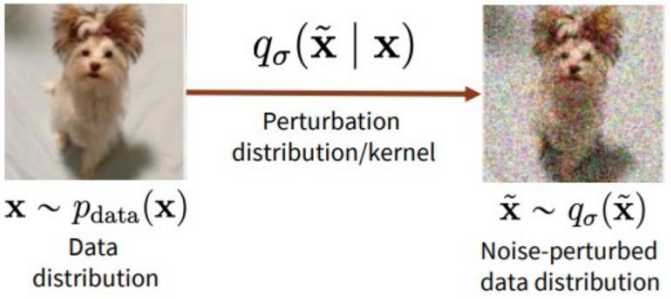
Now $E_{x\sim p_{data}(x)}[\left|\left|\nabla_x \log p_{data}(x) - s_\theta(x)\right|\right|^2]$ becomes

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

Carnegie Mellon University

Pascal Vincent. "A Connection Between Score Matching and Denoising Autoencoders". 2011. https://www.iro.umontreal.ca/~vincentp/Publications/smdae_techreport.pdf

Denoising score matching



Now $E_{x\sim p_{data}(x)}[||\nabla_x \log p_{data}(x) - s_\theta(x)||^2]$ becomes

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

If $\tilde{x} = x + \epsilon$ for $\epsilon \sim N(0, \sigma^2 I)$, then

$$q_\sigma(\tilde{x}|x) = \frac{1}{(2\pi)^{\frac{d}{2}}\sigma^d} e^{-\frac{1}{2\sigma^2}||\tilde{x}-x||^2}$$

$$\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x) = \frac{1}{\sigma^2}(x - \tilde{x})$$

Carnegie Mellon University

Figure from Stefano Ermon Stanford CS 236

Sampling with your score model – Langevin Dynamics

Once we have a trained score model, we can do "gradient ascent" in the data space to get a sample:

1. First draw from an easier-to-sample prior distribution $x_0 \sim \pi(x)$,.e.g. $N(0, I)$
2. Then with small step size η, large number of steps T and $z_t \sim N(0, I)$, iterate

$$x_{t+1} = x_t + \eta \nabla_{x_t} \log p(x_t) + \sqrt{2\eta} z_t$$

Can swap with the learned model $s_\theta(x_t)$

Exploration

Carnegie Mellon University

Langevin Dynamics

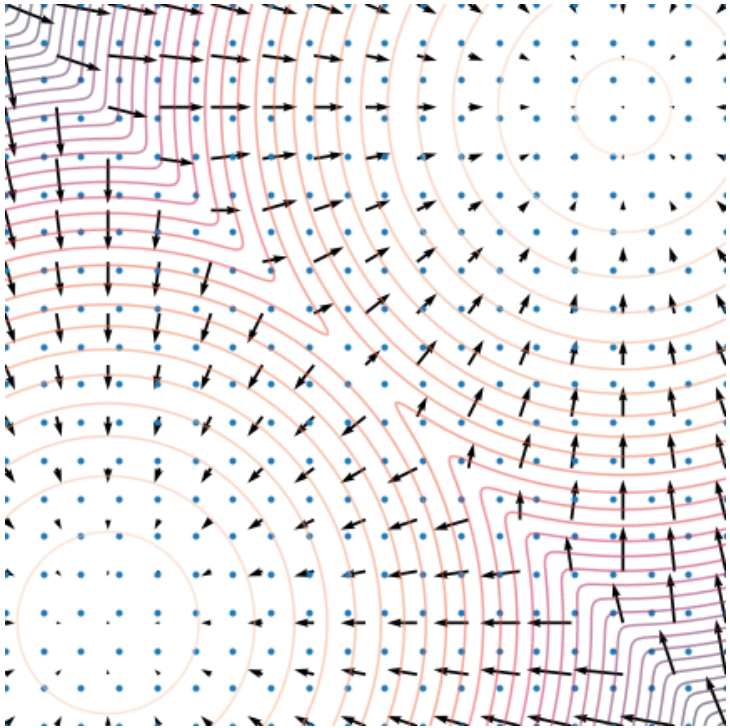
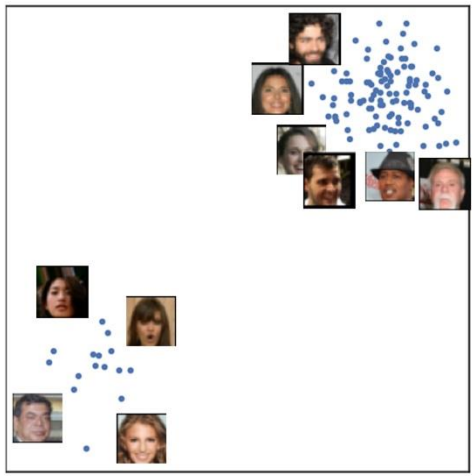


Figure from Yang Song https://yang-song.net/blog/2021/score/

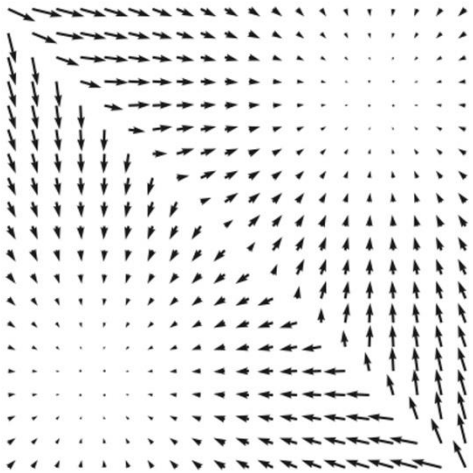
Score-based model pipeline



Data samples

$$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \overset{\text{i.i.d.}}{\sim} p(\mathbf{x})$$

score matching



Scores

$$\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$$

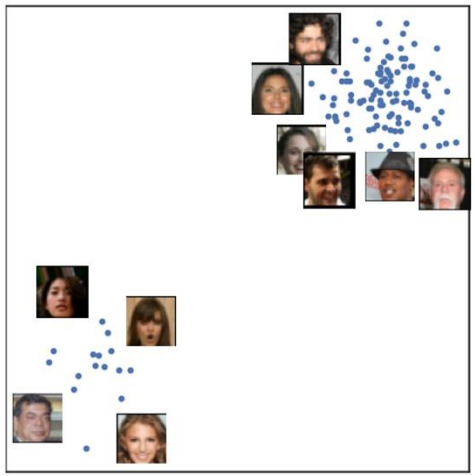
Langevin dynamics

New samples

Carnegie Mellon University

Figure from Yang Song https://yang-song.net/blog/2021/score/

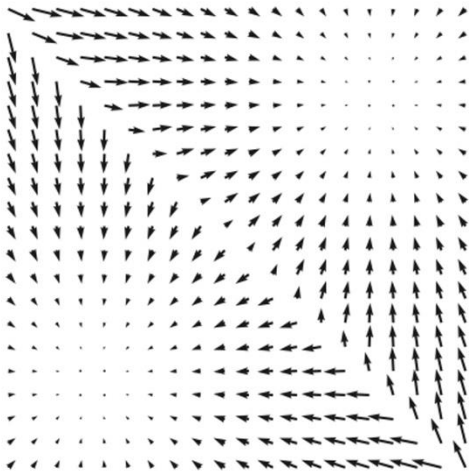
But does this actually work?



Data samples

$$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \overset{\text{i.i.d.}}{\sim} p(\mathbf{x})$$

score matching



Scores

$$\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$$

Langevin dynamics

New samples

Carnegie Mellon University

Figure from Yang Song https://yang-song.net/blog/2021/score/

Pitfall 1: The manifold hypothesis

The score is only properly defined in the whole data space, if data resides in a lower dimensional manifold (i.e. some area of the data space will have no support), then the score can blow up ($\pm\infty$ gradient) and the score estimation is not consistent any more

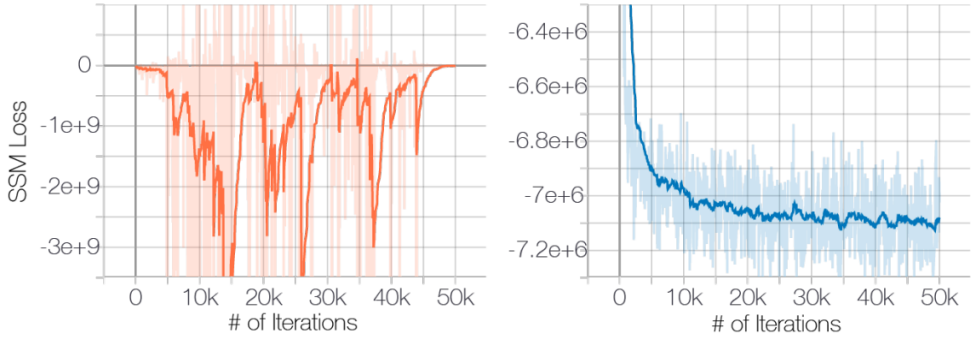


Figure 1: **Left**: Sliced score matching (SSM) loss w.r.t. iterations. No noise is added to data. **Right**: Same but data are perturbed with $\mathcal{N}(0, 0.0001)$.

Carnegie Mellon University

Yang Song and Stefano Ermon. "Generative Modeling by Estimating Gradients of the Data Distribution". NeurIPS 2019. https://arxiv.org/pdf/1907.05600.pdf

Pitfall 2: Score in low density region is inaccurate

Even when we do have full support data space, the score estimation is inaccurate in the low density regions

And our initial sample is very likely to be in those low density regions!!!

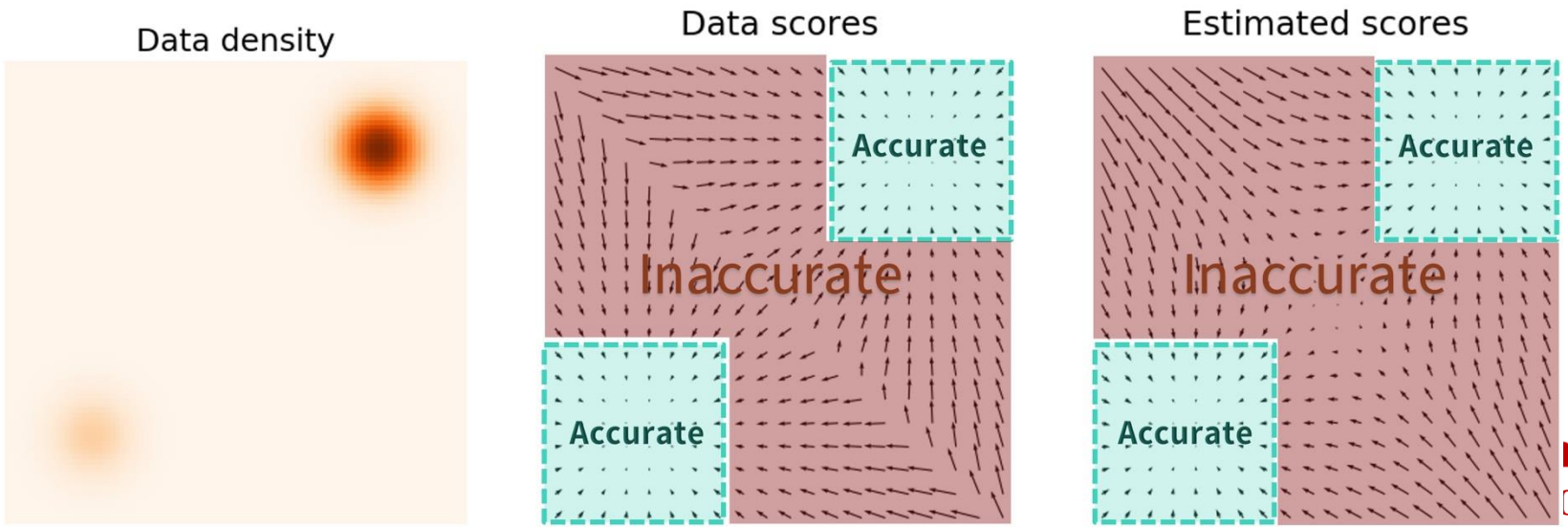


Figure from Yang Song https://yang-song.net/blog/2021/score/

Pitfall 3: Gradient cannot distinguish high density from higher density

Even in high density region, we can still get inaccurate sample distributions

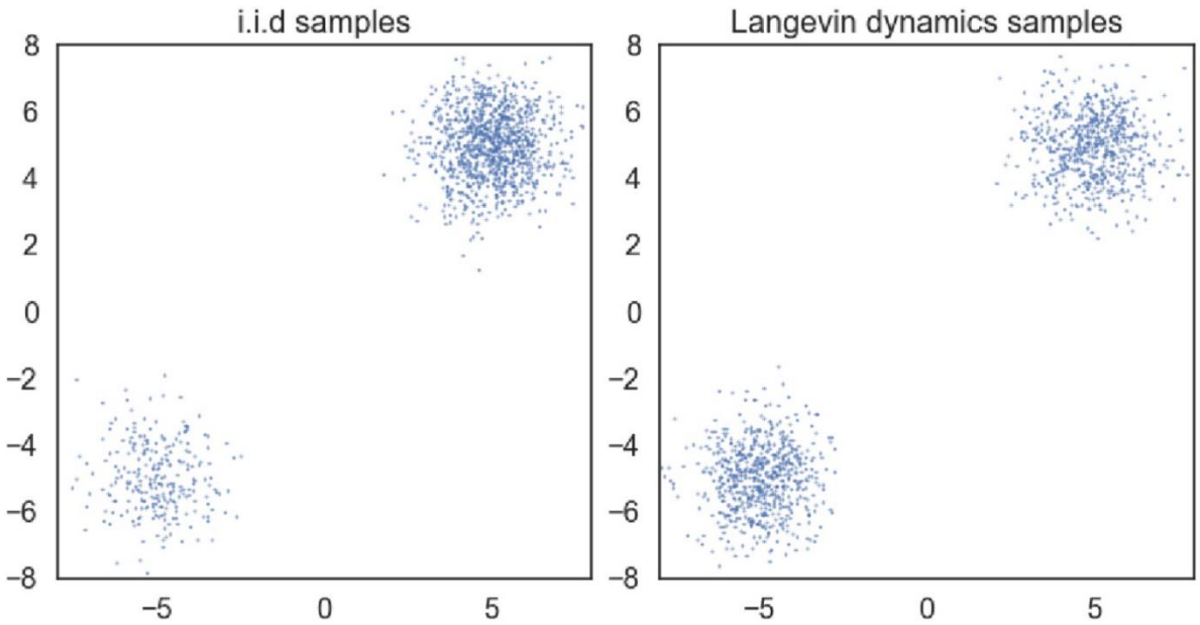
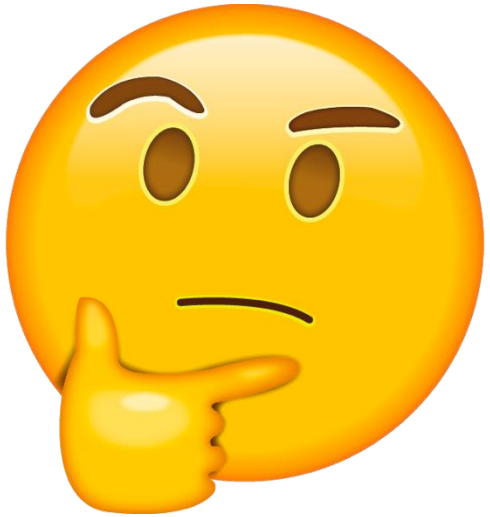


Figure from Yang Song https://yang-song.net/blog/2021/score/

Carnegie Mellon University

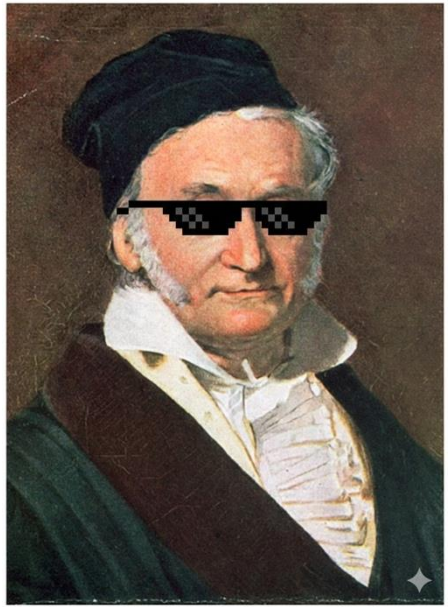
Is it possible to solve these problem all at once?

- Because the data often lies in **a lower dimensional manifold**, the score can blow up and the score estimation can be inconsistent in the ambient space
- The score estimation in **low density** region is often **inaccurate**
- Even in high density region, we can still get inaccurate sample distribution because sometimes **gradient** cannot distinguish between **high density v.s. higher density**



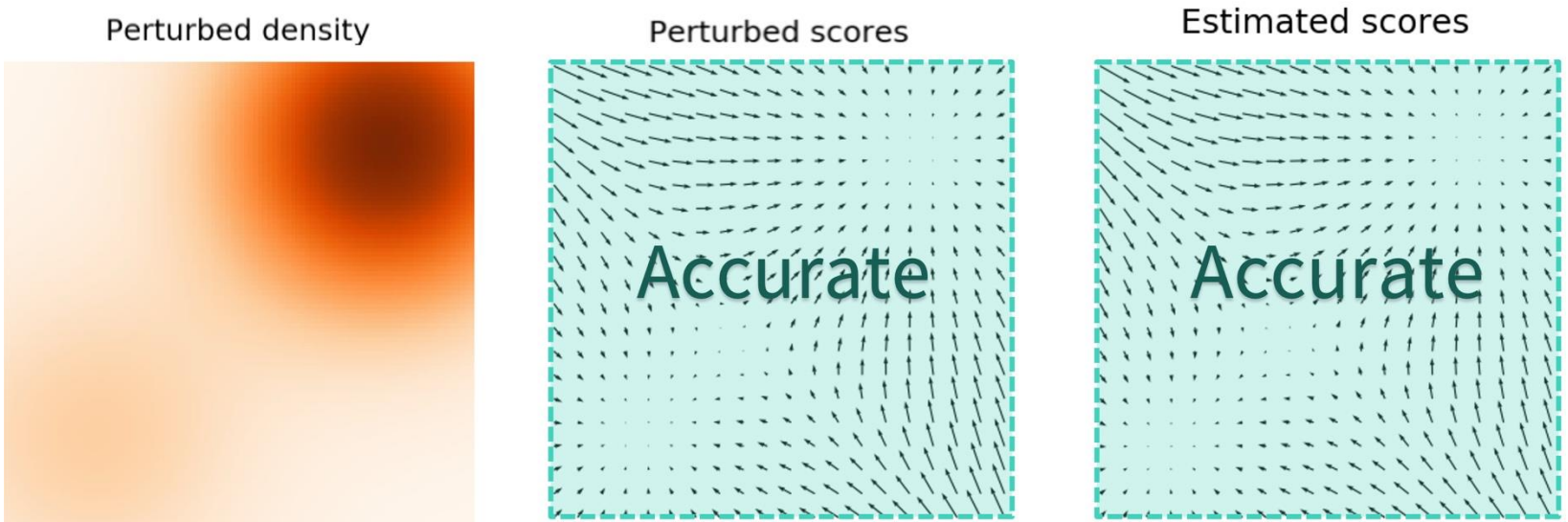
Is it possible to solve these problem all at once?

- Because the data often lies in **a lower dimensional manifold**, the score can blow up and the score estimation can be inconsistent in the ambient space
- The score estimation in **low density** region is often **inaccurate**
- Even in high density region, we can still get inaccurate sample distribution because sometimes **gradient** cannot distinguish between **high density v.s. higher density**



To solve all these problems

How about just add noise to the data?



Carnegie Mellon University

Figure from Yang Song https://yang-song.net/blog/2021/score/

Ok but, how much noise should we add

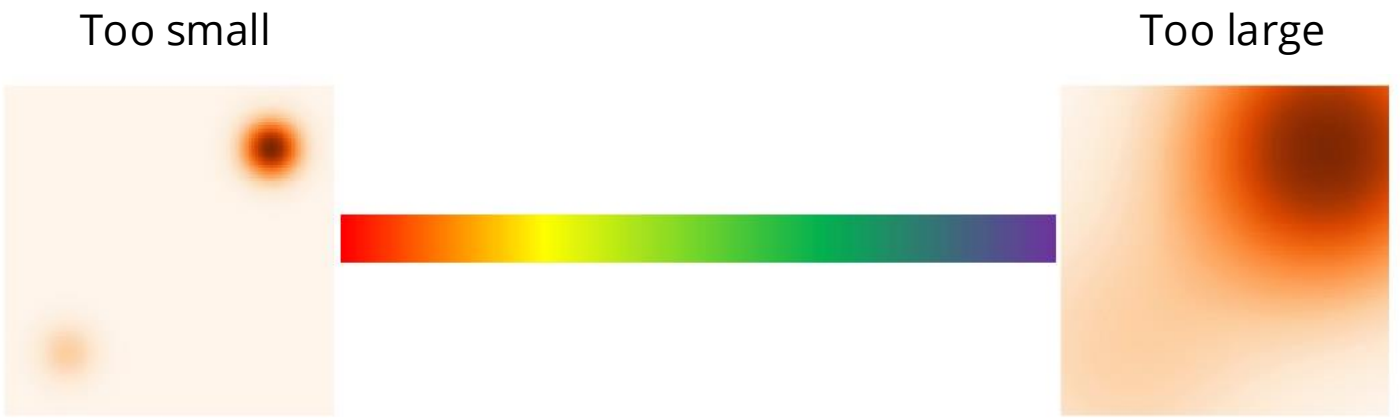


Figure from Stefano Ermon Stanford CS 236

How about we start big and gradually lower the noise level?

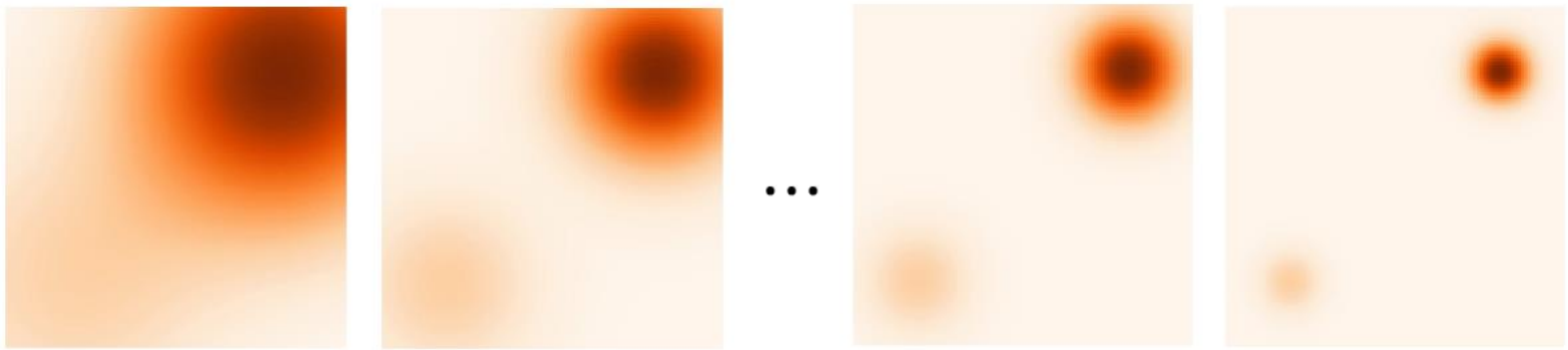
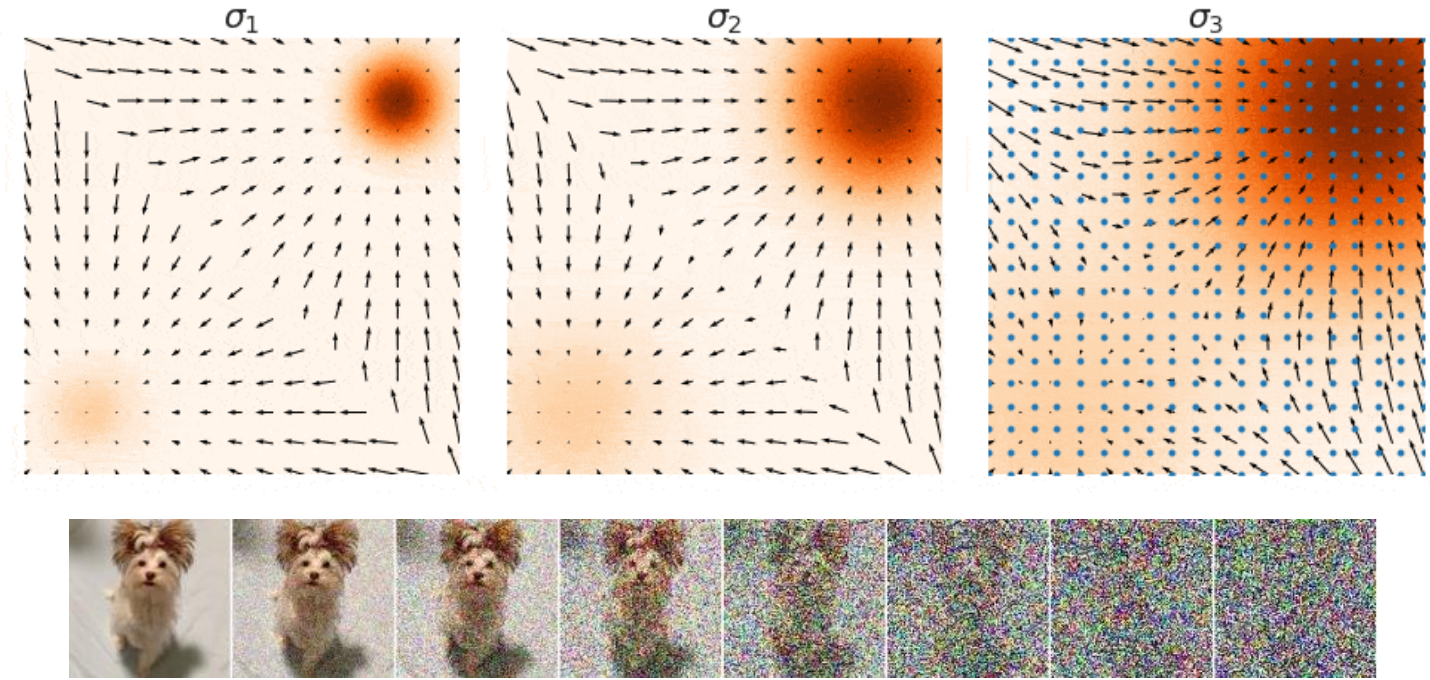


Figure from Stefano Ermon Stanford CS 236

Annealed Langevin Dynamics



Carnegie Mellon University

Figure from Yang Song https://yang-song.net/blog/2021/score/

Annealed Langevin Dynamics

Algorithm 1 Annealed Langevin dynamics.

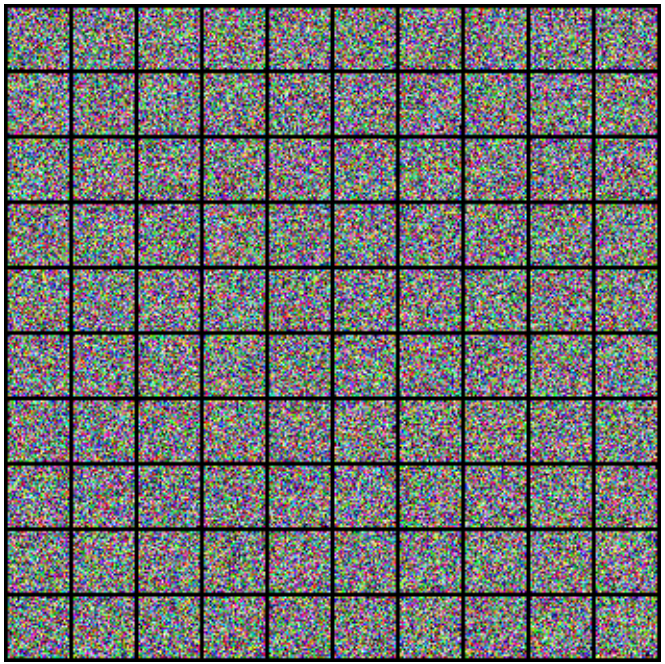
Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$.

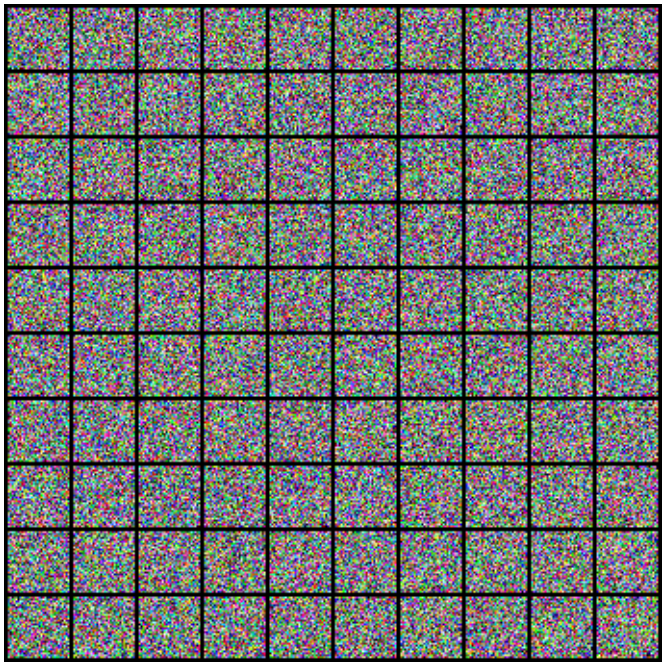
1: Initialize $\tilde{\mathbf{x}}_0$
2: **for** $i \leftarrow 1$ to L **do**
3: $\quad \alpha_i \leftarrow \epsilon \cdot \sigma_i^2/\sigma_L^2$ $\quad \triangleright$ α_i is the step size.
4: $\quad$ **for** $t \leftarrow 1$ to T **do**
5: $\quad\quad$ Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
6: $\quad\quad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

Carnegie Mellon University

Yang Song and Stefano Ermon. "Generative Modeling by Estimating Gradients of the Data Distribution". NeurIPS 2019. https://arxiv.org/pdf/1907.05600.pdf

Noise Conditional Score Network (NCSN)





Carnegie Mellon University

Yang Song and Stefano Ermon. "Generative Modeling by Estimating Gradients of the Data Distribution". NeurIPS 2019. https://arxiv.org/pdf/1907.05600.pdf

Noise Conditional Score Network (NCSN)

Network Parameterization: $s_\theta(\tilde{x}_{\sigma_t}, \sigma_t)$

Training: Multi-level denoising score matching

$$E_{\sigma_t} E_{x \sim p_{data}(x)} E_{\tilde{x}_{\sigma_t} \sim p_{\sigma_t}(\tilde{x}_{\sigma_t}|x)} \left[\left\| s_\theta(\tilde{x}_{\sigma_t}, \sigma_t) + \frac{x - \tilde{x}_{\sigma_t}}{\sigma_t^2} \right\|^2 \right]$$

Sampling: Annealed Langevin Dynamics

Carnegie Mellon University

Yang Song and Stefano Ermon. "Generative Modeling by Estimating Gradients of the Data Distribution". NeurIPS 2019. https://arxiv.org/pdf/1907.05600.pdf

Hold up, wait a minute, doesn't this look familiar?

Diffusion (DDPM)

Data x_0 $t = 0$ | x_1 $t = 1$ | x_2 $t = 2$ | x_3 $t = 3$ | Noise x_4 $t =$

Score-based model (NCSN)

Carnegie Mellon University

Spot the difference: DDPM v.s. NCSN

	NCSN	DDPM
$\mathbf{x}_{i+1}\|\mathbf{x}_i$	Derive as $\mathbf{x}_{i+1} = \mathbf{x}_i + \sqrt{\sigma_{i+1}^2 - \sigma_i^2}\boldsymbol{\epsilon}$	Define as $\mathbf{x}_{i+1} = \sqrt{1-\beta_i}\mathbf{x}_i + \sqrt{\beta_i}\boldsymbol{\epsilon}$
$\mathbf{x}_i\|\mathbf{x}$	Define as $\mathbf{x}_i = \mathbf{x} + \sigma_i\,\boldsymbol{\epsilon}$	Derive as $\mathbf{x}_i = \bar{\alpha}_i\mathbf{x} + \sqrt{1-\bar{\alpha}_i}\boldsymbol{\epsilon}$
p_{prior}	$\mathcal{N}(\mathbf{0}, \sigma_L^2\mathbf{I})$	$\mathcal{N}(\mathbf{0}, \mathbf{I})$
Loss	$\mathbb{E}_i\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\left\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_i, \sigma_i) + \frac{\boldsymbol{\epsilon}}{\sigma_i}\right\|_2^2\right]$	$\mathbb{E}_i\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\|\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_i, i) - \boldsymbol{\epsilon}\|_2^2\right]$
Sampling	Apply Langevin per layer; use output to initialize the next	Traversing the Markovian chain with $p_{\boldsymbol{\phi}^\times}(\mathbf{x}_{i-1}\|\mathbf{x}_i)$

Carnegie Mellon University

Lai et al. "The Principles of Diffusion Models". 2025. https://arxiv.org/pdf/2510.21890#page=89.12

Spot the difference: DDPM v.s. NCSN

	NCSN	DDPM
$\mathbf{x}_{i+1}\|\mathbf{x}_i$	Derive as $\mathbf{x}_{i+1} = \mathbf{x}_i + \sqrt{\sigma_{i+1}^2 - \sigma_i^2}\boldsymbol{\epsilon}$	Define as $\mathbf{x}_{i+1} = \sqrt{1-\beta_i}\mathbf{x}_i + \sqrt{\beta_i}\boldsymbol{\epsilon}$
$\mathbf{x}_i\|\mathbf{x}$	Define as $\mathbf{x}_i = \mathbf{x} + \sigma_i\boldsymbol{\epsilon}$	Derive as $\mathbf{x}_i = \bar{\alpha}_i\mathbf{x} + \sqrt{1-\bar{\alpha}_i}\boldsymbol{\epsilon}$
p_{prior}	$\mathcal{N}(\mathbf{0}, \sigma_L^2\mathbf{I})$	$\mathcal{N}(\mathbf{0}, \mathbf{I})$
Loss	$\mathbb{E}_i\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\left\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_i, \sigma_i) + \frac{\boldsymbol{\epsilon}}{\sigma_i}\right\|_2^2\right]$	$\mathbb{E}_i\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\left\|\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\mathbf{x}_i, i) - \boldsymbol{\epsilon}\right\|_2^2\right]$
Sampling	Apply Langevin per layer; use output to initialize the next	Traversing the Markovian chain with $p_{\boldsymbol{\phi}^\times}(\mathbf{x}_{i-1}\|\mathbf{x}_i)$

Carnegie Mellon University

Lai et al. "The Principles of Diffusion Models". 2025. https://arxiv.org/pdf/2510.21890#page=89.12

Noise parameterization: DDPM v.s. NCSN

First define $x_t = \lambda_t x_0 + \sigma_t \epsilon$ for $\epsilon \sim N(0, I)$

NCSN: $\lambda_t = 1$, $\sigma_t = \sigma_t$

$$\epsilon = \frac{x_t - x_0}{\sigma_t} = \frac{x_t - \lambda_t x_0}{\sigma_t}$$

DDPM: $\lambda_t = \sqrt{\overline{\alpha_t}}$, $\sigma_t = \sqrt{1 - \overline{\alpha_t}}$

$$\epsilon = \frac{x_t - \sqrt{\overline{\alpha_t}} x_0}{\sqrt{1 - \overline{\alpha_t}}} = \frac{x_t - \lambda_t x_0}{\sigma_t}$$

Loss function: DDPM v.s. NCSN

First define $x_t = \lambda_t x_0 + \sigma_t \epsilon$ for $\epsilon \sim N(0, I)$

NCSN: $\lambda_t = 1$, $\sigma_t = \sigma_t$

$$\epsilon = \frac{x_t - x_0}{\sigma_t} = \frac{x_t - \lambda_t x_0}{\sigma_t}$$

DDPM: $\lambda_t = \overline{\alpha_t}$, $\sigma_t = \sqrt{1 - \overline{\alpha_t}}$

$$\epsilon = \frac{x_t - \overline{\alpha_t} x_0}{\sqrt{1 - \overline{\alpha_t}}} = \frac{x_t - \lambda_t x_0}{\sigma_t}$$

Loss functions:

NCSN: $E_{\sigma_t} E_{x \sim p_{data}(x)} E_{\epsilon \sim N(0,I)} [\left|\left| s_\theta(x_t, \sigma_t) + \frac{\epsilon}{\sigma_t^2} \right|\right|^2]$

$$s_\theta^*(x_t, \sigma_t) = -\frac{1}{\sigma_t} E_{\epsilon \sim p_{\sigma_t}(\epsilon | x_t)}[\epsilon]$$

DDPM: $E_t E_{x \sim p_{data}(x)} E_{\epsilon \sim N(0,I)} [\left| |\epsilon_\theta(x_t, t) - \epsilon \right|^2]$

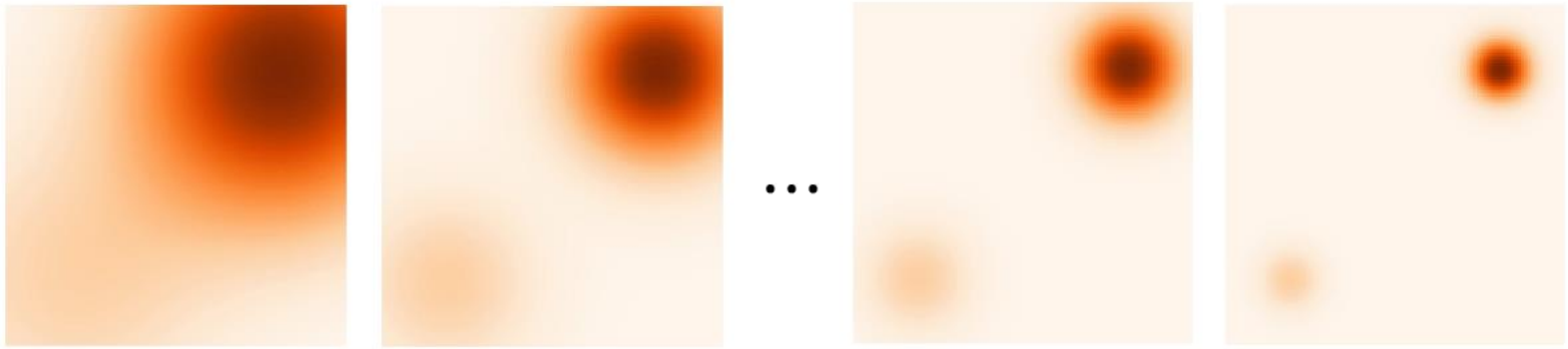
$$\epsilon_\theta^*(x_t, \sigma_t) = E_{\epsilon \sim p_t(\epsilon | x_t)}[\epsilon]$$

Basically two sides of the same coin

Diffusion (DDPM)

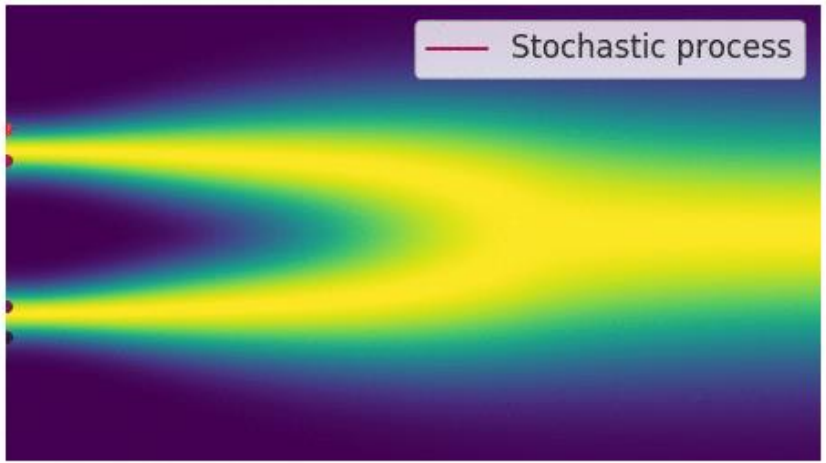
Score-based model (NCSN)

What’s gonna happen if we have an infinite number of noise levels?



When the number of noise scales goes to infinity

It becomes a **continuous-time stochastic process**, many of which can be solved by **stochastic differential equations** (SDEs)



Carnegie Mellon University

Figure from Yang Song https://yang-song.net/blog/2021/score/

Infinite noise scales: Forward Process

The forward processes are

NCSN: $x_{t+\Delta t} = x_t + \sqrt{\sigma_{t+\Delta t}^2 - \sigma_t^2}\epsilon$

$$\approx x_t + \sqrt{\frac{d\sigma^2(t)}{dt}\Delta t}\epsilon$$

DDPM: $x_{t+1} = \sqrt{1-\beta_t}x_t + \sqrt{\beta_{t+1}}\epsilon$

$$\approx x_t - \frac{1}{2}\beta(t)x_t\Delta t + \sqrt{\beta_t \Delta t}\epsilon$$

We can also write them as

NCSN: $x_{t+\Delta t} = x_t + f(x_t, t)\Delta \mathrm{t} + g(t)\sqrt{\Delta t}\epsilon$

$$f(x_t, t) = 0 \qquad g(t) = \sqrt{\frac{d\sigma^2(t)}{dt}}$$

DDPM: $x_{t+\Delta t} = x_t + f(x_t, t)\Delta \mathrm{t} + g(t)\sqrt{\Delta t}\epsilon$

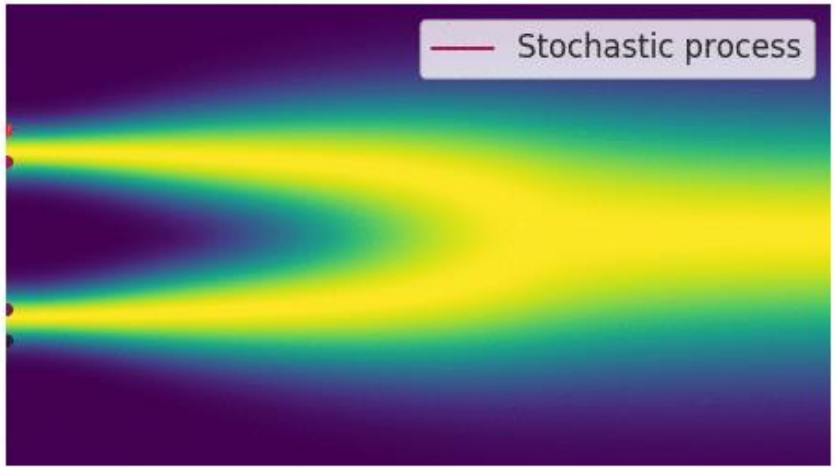
$$f(x_t, t) = -\frac{1}{2}\beta(t)x_t \qquad g(t) = \sqrt{\beta_t}$$

Carnegie Mellon University

Infinite noise scales: Forward Process

It becomes a **continuous-time stochastic process**, many of which can be solved by **stochastic differential equations** (SDEs)

$$dx = f(x,t)dt + g(t)dw$$

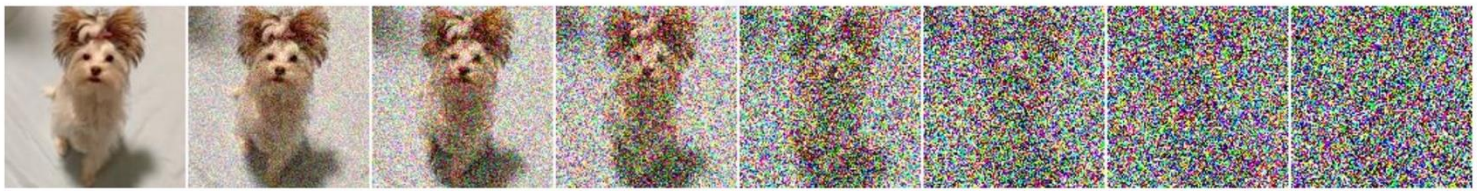


Carnegie Mellon University

Score SDE: Reverse Process w/ infinite noise scales

Forward SDE (data → noise)

$\mathbf{x}(0)$ — $\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$ → $\mathbf{x}(T)$



score function

$\mathbf{x}(0)$ ← $\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] \mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}$ — $\mathbf{x}(T)$

Reverse SDE (noise → data)

Brian D.O. Anderson. "Reverse-time diffusion equation models". Stochastic Processes and their Applications 1982.

Song et al. "Score-Based Generative Modeling through Stochastic Differential Equations". ICLR 2021. https://openreview.net/pdf?id=PxTIG12RRHS

Carnegie Mellon University

Score SDE: Training

Literally just score matching with continuous time

$$\mathcal{L}_{\mathrm{SM}}(\boldsymbol{\phi};\omega(\cdot)) := \frac{1}{2}\mathbb{E}_{t\sim p_{\mathrm{time}}}\mathbb{E}_{\mathbf{x}_t\sim p_t}\left[\omega(t)\left\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t,t)-\nabla_{\mathbf{x}}\log p_t(\mathbf{x}_t)\right\|_2^2\right]$$

$$\mathcal{L}_{\mathrm{DSM}}(\boldsymbol{\phi};\omega(\cdot)) := \frac{1}{2}\mathbb{E}_t\mathbb{E}_{\mathbf{x}_0}\mathbb{E}_{p_t(\mathbf{x}_t|\mathbf{x}_0)}\left[\omega(t)\left\|\mathbf{s}_{\boldsymbol{\phi}}(\mathbf{x}_t,t)-\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t|\mathbf{x}_0)\right\|_2^2\right], \quad (4.2.1)$$

Carnegie Mellon University

Lai et al. “The Principles of Diffusion Models”. 2025. https://arxiv.org/pdf/2510.21890#page=89.12

Score SDE: Sampling

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g^2(t)\mathbf{s}_\theta(\mathbf{x}, t)]\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$$

We can use **solvers** to solve this differential equation

e.g. Use Euler solver, we can have the sampling process as

1. Sample x_1 from source distribution
2. $\Delta\mathbf{x} \leftarrow [\mathbf{f}(\mathbf{x}, t) - g^2(t)\mathbf{s}_\theta(\mathbf{x}, t)]\Delta t + g(t)\sqrt{|\Delta t|}\mathbf{z}_t$

   $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}$

   $t \leftarrow t + \Delta t,$

   where $z_t \sim N(0, I)$
3. Iterate until $t = 0$

DDPM v.s. NCSN => VP SDE v.s. VE SDE

Discrete time:

NCSN: $x_{t+\Delta t} = x_t + f(x_t, t)\Delta t + g(t)\sqrt{\Delta t}\epsilon$

$$f(x_t, t) = 0 \qquad g(t) = \sqrt{\frac{d\sigma^2(t)}{dt}}$$

DDPM: $x_{t+\Delta t} = x_t + f(x_t, t)\Delta t + g(t)\sqrt{\Delta t}\epsilon$

$$f(x_t, t) = -\frac{1}{2}\beta(t)x_t \qquad g(t) = \sqrt{\beta_t}$$

VP = **V**ariance **P**reserving

Continuous time:

VE SDE: $dx = f(x, t)dt + g(t)dw$

$$f(x_t, t) = 0 \qquad g(t) = \sqrt{\frac{d\sigma^2(t)}{dt}}$$

VP SDE: $dx = f(x, t)dt + g(t)dw$

$$f(x_t, t) = -\frac{1}{2}\beta(t)x_t \qquad g(t) = \sqrt{\beta_t}$$

VE = **V**ariance **E**xploding

Score SDE

Carnegie Mellon University

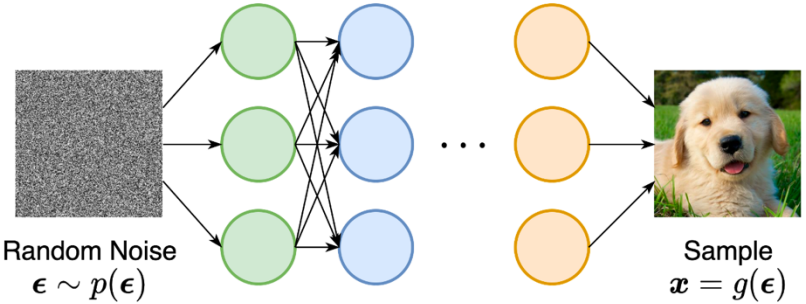
Song et al. "Score-Based Generative Modeling through Stochastic Differential Equations". ICLR 2021. https://openreview.net/pdf?id=PxTIG12RRHS

So far we have seen a bunch of generative models…

In general, we can roughly categorize generative models into the following categories

- **Likelihood Based:** Autoregressive models, variational autoencoders (VAE), normalizing flow, energy-based models (EBM), **diffusion models**
- **Likelihood Free:** Generative adversarial networks (GAN), **score-based models**

Same thing!



Directly sampling from P(X) is usually hard because they are usually complicated! But **sampling from a simpler distribution** (eg. a Gaussian) is easy!

Carnegie Mellon University

Generative modeling

Given a set of data {x} and some prior knowledge & assumptions

- **Data:** samples (e.g. images of bedrooms)
- **Prior knowledge & assumptions:** parametric form, loss function, optimization, etc

We want to learn a probability distribution $p_\theta(x)$ such that

- **Generation:** If we sample a new datapoint from $p_\theta(x)$, it'd look like a "real" sample (e.g. looks like a real image of bedroom) ✅
- **Density estimation:** Given an existing datapoint x, we should be able to assign a probability to it (probability should be high if x looks "real") ❓
- **Unsupervised learning:** We learn everything by just looking at the data

Is there an even simpler way to do the same thing?

Carnegie Mellon University

Image from Harry Potter