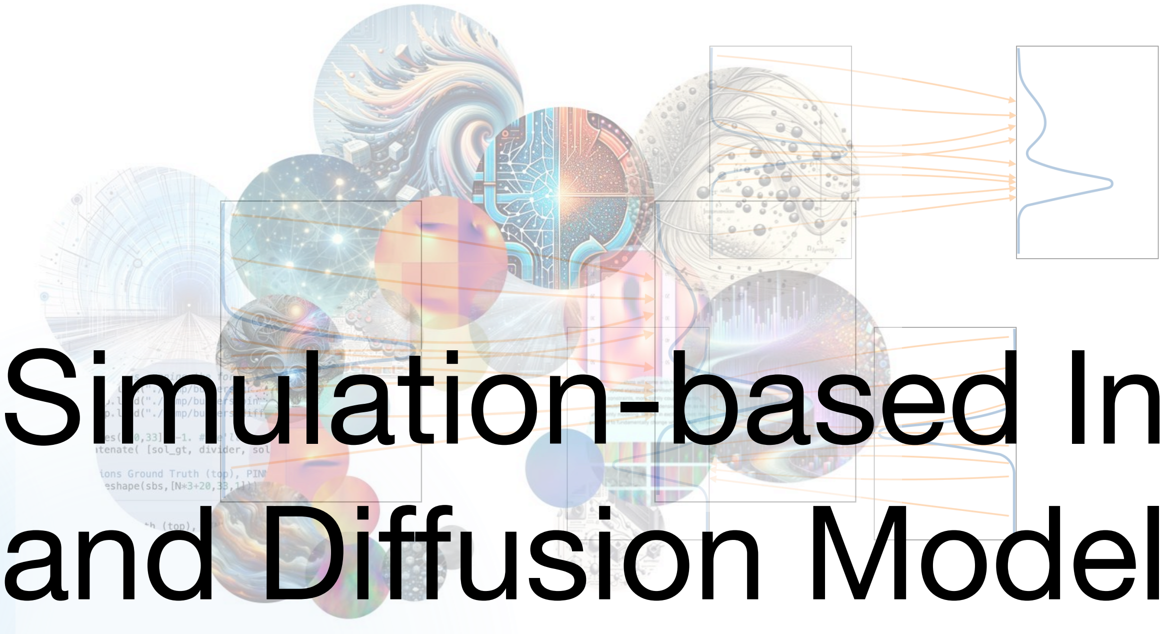
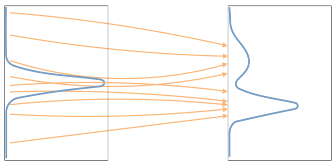


Simulation-based Inference and Diffusion Models

PROBABILISTIC INVERSE PROBLEMS

Contents

- Simulation-based inference Solving inverse problems with deep learning
- Continuous Normalizing Flows An architecture for conditional density estimation
- Denoising score matching Unnormalized density estimation
- Annealed Langevin Dynamics Turning score matching into a generative model
- Diffusion models Generating data from noise
- Physics Constraints Include PDE priors



Simulation-based Inference

Simulation-based inference

Simulations and Uncertainty

- The simulator is a (statistical) model described by a computer program
- Given a vector of parameters x, distribution of **latent variables** $z \sim p(z \mid x)$
- The simulator produces an observation or **output** $y \sim p(y \mid x, z)$

Examples for x	Examples for z
Constants of Nature, Reynolds Nr.	Unobservable / stochastic variables
(Partial) State of the system	Intermediate simulation steps
Incubation rate of a pathogen	Control flow of simulator

Reminder: Forward vs Backward

➡️ Forward Problems ➡️

- Obtain the distribution for the **outputs** $y \sim p(y \mid x, z)$ given uncertainties in z
- Typical setting for *classical numerical methods*

⬅️ Inverse Problems ⬅️

- We measure or **observe** y and want to know which $x \sim p(x \mid y, z)$ lead to it
- Typical setting for *simulation-based inference* (and this whole chapter)

Simulation-based inference

Bayesian Inference

- There is a **prior** $p(x)$ over the parameters

- The function $p(y \mid x)$ is called the **likelihood** function

- We are interested in the posterior

$$p(x \mid y) = \frac{p(y \mid x)p(x)}{\boxed{\int p(y \mid x')p(x')dx'}} \quad \textbf{Evidence}$$

- The likelihood $p(y \mid x) = \int p(y, z \mid x)dz$ is often intractable

Simulation-based inference

Challenges

- Calculating the **evidence** is expensive, typically requires Markov Chain Monte Carlo (MCMC) methods or variational inference (VI)
- The likelihood is often intractable. Solve with Approximate Bayesian Computation (ABC). Computationally expensive and expert knowledge required

Deep Learning

Train a conditional density estimator $q_\theta(x \mid y)$ for the posterior $p(x \mid y)$ that allows sampling and can be trained from simulations $y \sim p(y \mid x)$ alone

Simulation-based inference

Unconditional and Conditional Density Estimators

- Consider a model family $\{q_\theta(x)\}_\theta$ parameterized with weights θ such that for all θ the model $q_\theta(x)$ is a density

$$\int q_\theta(x)dx = 1$$

- We will discuss normalizing flows ($\rightarrow$ later) as an example how to build these models
- Train $q_\theta(x)$ to be close to a target density $p(x)$
- Theory and implementation can be extended to conditional models $q_\theta(x \mid y)$

Simulation-based inference

Classical methods

- ✅ Decades of research and rich mathematical theory
- ✅ Easy to replace simulations, likelihood functions and priors
- ❌ Computationally expensive, curse of dimensionality
- ❌ Difficult to represent arbitrary priors as mathematical models

Learning-based methods

- ✅ Fast inference once trained
- ✅ Not affected by curse of dimensionality as strongly, can represent arbitrary priors
- ❌ Lacks more rigorous theoretical guarantees, requires upfront training cost

Short Detour:
Training Objectives & Conditioning

Comparing Distributions

Kullback Leibler (KL) Divergence

- The KL divergence between two probability distributions P and Q with densities p and q is defined as

$$\mathrm{KL}(p\,||\,q) = \int p(x)\,\log\left(\frac{p(x)}{q(x)}\right)dx$$

- Always positive $\mathrm{KL}(p\,||\,q) \geq 0$, and $\mathrm{KL}(p\,||\,q) = 0$ if and only if $P = Q$

- Goal: Knowing p, optimize θ by minimizing $\mathrm{KL}(p\,||\,q_\theta)$

Unconditional Training Objective

$$\mathrm{KL}(p\,||\,q_\theta) = \int p(x)\,\log\left(\frac{p(x)}{q(x)}\right)dx$$

$$= \mathbb{E}_{x\sim p(x)}\left[\log\left(\frac{p(x)}{q_\theta(x)}\right)\right]$$

$$= \boxed{\mathbb{E}_{x\sim p(x)}[\log p(x)]} - \mathbb{E}_{x\sim p(x)}[\log q_\theta(x)]$$

constant w.r.t. θ

- When q_θ is a density, the training objective for θ is minimizing

$$\mathbb{E}_{x\sim p(x)}[-\log q_\theta(x)]$$

Conditional Training Objective

- When considering the posterior $p(x \mid y)$, the objective becomes

$$\mathbb{E}_{y \sim p(y)} \left[\mathbb{E}_{x \sim p(x|y)}[-\log q_\theta(x \mid y)] \right]$$

- We can rewrite this as

$$\begin{aligned}
\mathbb{E}_{y \sim p(y)} \left[\mathbb{E}_{x \sim p(x|y)}[-\log q_\theta(x \mid y)] \right] &= - \int \int \boxed{p(y)p(x \mid y)} \log p(x \mid y) dx dy \\
&= - \int \int \boxed{p(y, x)} \log p(x \mid y) dx dy \qquad \text{Bayes' theorem} \\
&= - \int \int \boxed{p(x)p(y \mid x)} \log p(x \mid y) dy dx \\
&= \mathbb{E}_{x \sim p(x), y \sim p(y|x)}[-\log q_\theta(x \mid y)]
\end{aligned}$$

- Great! Directly applicable to conditional distributions … strictly for densities!

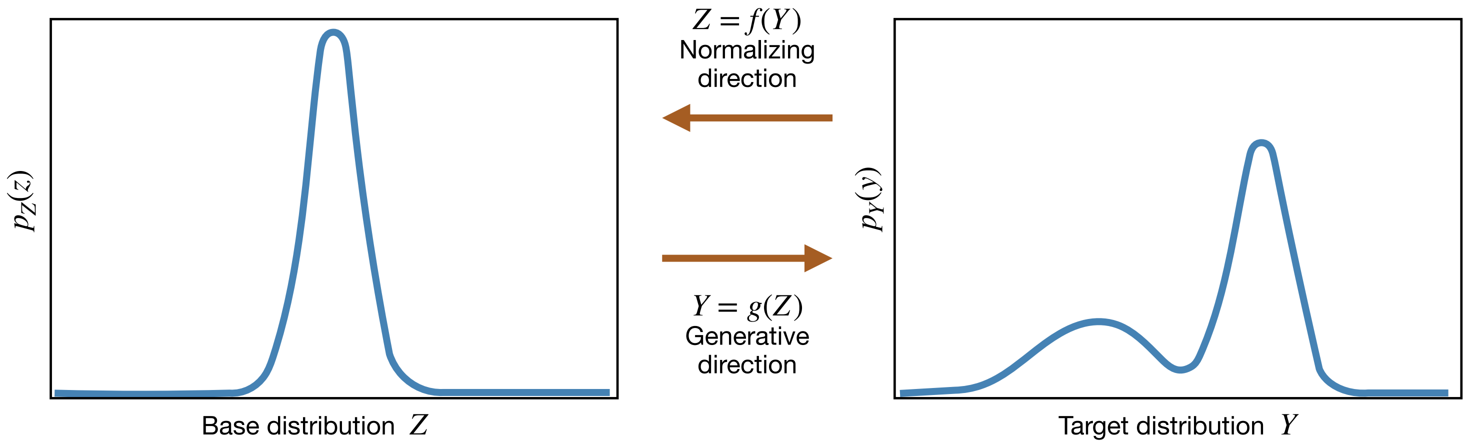
Normalizing Flows

Normalizing Flows

Normalizing Flows

Kobyzev et. al. (2020)

Normalizing Flows are transformations of a simple base distribution p_Z into a complicated target distribution p_Y via a sequence of invertible and differentiable mappings



(Similar to z from SBI, but “simple”)

Normalizing Flows

Normalizing Flows

- For a single, invertible mapping $g : \mathbb{R}^D \rightarrow \mathbb{R}^D$ and inverse function $f = g^{-1}$, we can write

$$y = g(z) \text{ and } z = f(y)$$

- The probability density $p_Y(y)$ can be computed as

$$p_Y(y) = p_Z(f(y)) \left| \det \boxed{\frac{\partial f}{\partial y}} \right| \quad \text{Jacobian of } f$$

- p_Z is usually a normal Gaussian, so evaluating $p_Z\big(f(y)\big)$ is easy

Normalizing Flows

Stacking Mappings

- We compose several mappings, so that we can write

$$g = g_1 \circ g_2 \circ \ldots \circ g_n \text{ and } f = f_n \circ f_{n-1} \circ \ldots \circ f_1$$

where g_i is the inverse of f_i

- Define $y_i = f_n \circ f_{n-1} \circ \ldots \circ f_{i+1}$

- The probability density $p_Y(y)$ can be written as $p_Y(y) = p_Z\big(f(Z)\big) \displaystyle\prod_{i=1}^{n} \left| \det \frac{\partial f_i}{\partial y_i} \right|$

Normalizing Flows

- Note: we can easily turn probability densities into (log) likelihoods. This turns the products of previous equations into sums.

- In practice, we want to parameterize the weights of g_i by θ_i. It is not trivial, to find mapping types that are invertible for all possible parameters θ_i
    - Coupling layers (Dinh et al. 2015)
    - Autoregressive flows (Kingma et al. 2016)

- **Easy sampling:** draw a random vector from $p_Z(z)$, which is usually a normal Gaussian. We obtain a sample from the target distribution via $y = g(z)$, it's probability is computed by transforming $p_Z(z)$ into p_Y

Neural Ordinary Differential Equations

Continuous-time Networks

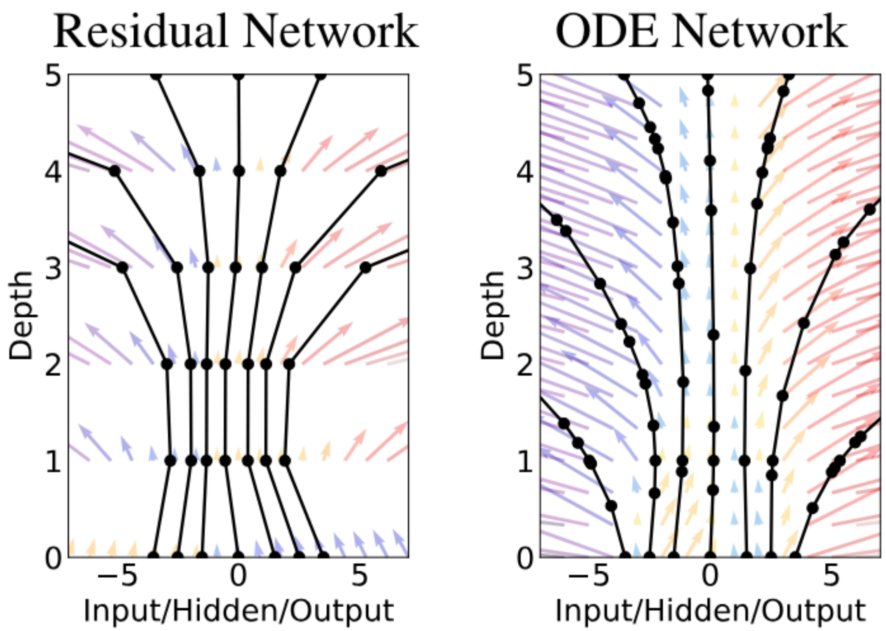
- **Before:** a sequence of skip connections

$$h_{t+1} = h_t + f(h_t, \theta_t),$$

where $f(\cdot, \theta_t)$ is a network with weights θ_t and steps $t \in \{0...T\}$

- **Neural ODE:** consider an ODE as the continuous-time limit of above

$$\frac{dh(t)}{dt} = f(h(t), t, \theta)$$



Chen et. al. (2018)

Continuous Normalizing Flows

Continuous-time Normalizing Flows

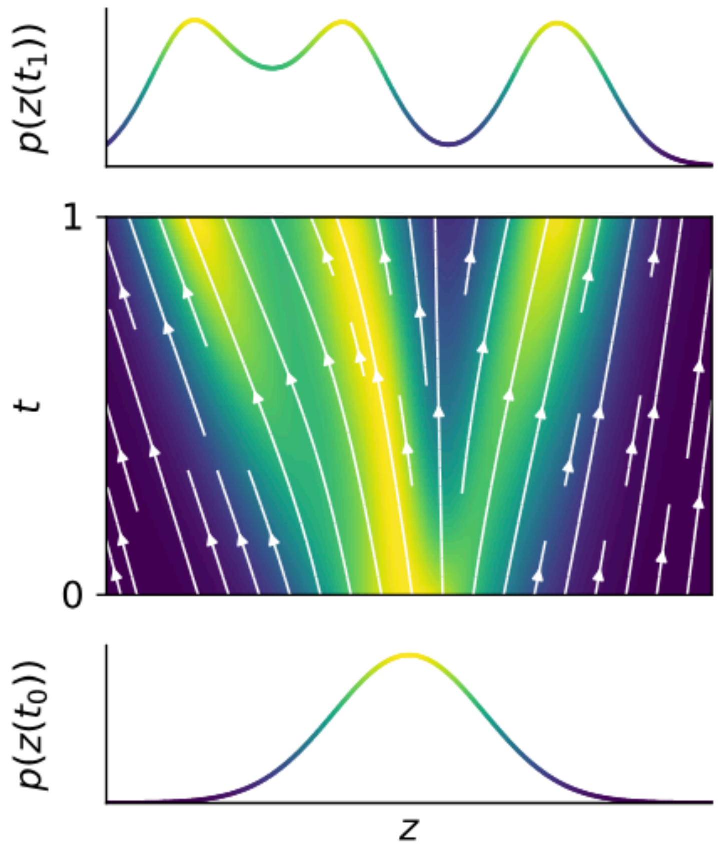
- Replace composition of layers g_θ by continuous-time network $f_\theta(\,\cdot\,, t)$
- Consider the neural ODE

$$\frac{\partial z(t)}{\partial t} = f_\theta(z(t), t)$$

with initial conditions $z(0) = z_0$ from time $t_0 = 0$ until $t_1 = 1$

- The neural ODE is invertible and differentiable w.r.t. $z(0)$ and θ
- We can track the change in probability via the instantaneous change of variables formula

$$\frac{\partial \log p(z(t))}{\partial t} = -\,\mathrm{Tr}\left(\frac{\partial f}{\partial z(t)}\right)$$



Grathwohl et. al. (2019)

Continuous Normalizing Flows

Summary: Training a CNF for Simulation-based Inference

1. Generate a dataset of pairs (x, o) where we sample $x \sim p(x)$ and simulate $o \sim p(o \mid x)$

2. The training objective is $\mathbb{E}_{(x,o) \in p(x,o)}[-\log q_\theta(x \mid o)]$

3. To compute $\log q_\theta(x \mid o)$ solve the neural ODE (Euler, RK, etc.)

$$\underbrace{\begin{bmatrix} \mathbf{z}_0 \\ \log p(\mathbf{x}) - \log p_{z_0}(\mathbf{z}_0) \end{bmatrix}}_{\text{solutions}} = \underbrace{\int_{t_1}^{t_0} \begin{bmatrix} f(\mathbf{z}(t), t; \theta) \\ -\operatorname{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) \end{bmatrix} dt,}_{\text{dynamics}} \qquad \underbrace{\begin{bmatrix} \mathbf{z}(t_1) \\ \log p(\mathbf{x}) - \log p(\mathbf{z}(t_1)) \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ 0 \end{bmatrix}}_{\text{initial values}}$$

2. Calculate the gradient $\nabla_\theta \log q_\theta(x \mid o)$ and update θ

Continuous Normalizing Flows

✅ For calculating $\nabla_\theta \log q_\theta(x \mid o)$, we can use any AD method we prefer (see previous lectures)

✅ Maximum likelihood training that directly minimizes $\mathrm{KL}(p \,||\, q_\theta)$

✅ Efficiently share parameters θ across different time steps t

❌ In practice the memory requirements and training/inference costs are large→ solving the neural ODE for training is not scalable!

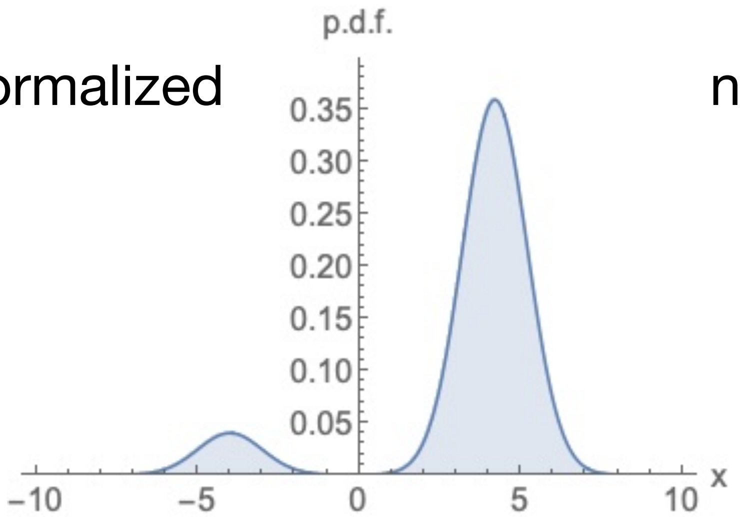
→ We need to find a way to train networks more efficiently without needing to backpropagate gradients through the entire ODE

Score Matching

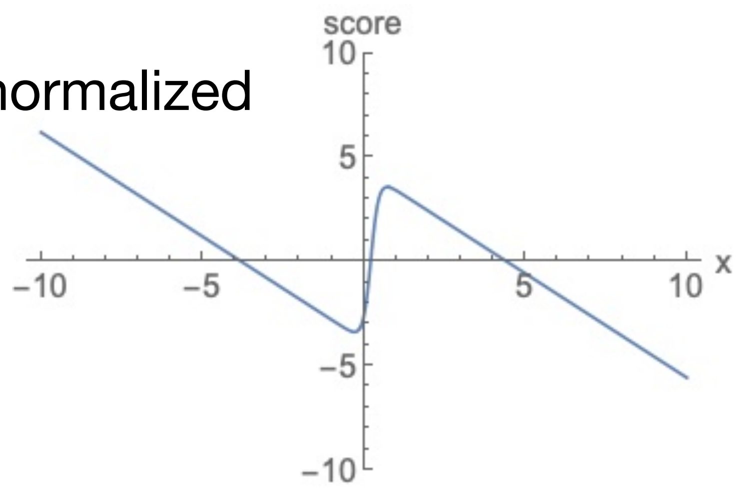
Score Matching

- A normalizing flow directly approximates the target distribution $p(x)$ by $q_\theta(x)$
- Instead, we can approximate the score $\nabla_x \log p(x)$ with a network $s_\theta(x)$

normalized



not normalized



Score Matching

For training the network $s_\theta(x)$, we want to minimize the **Fisher divergence**

$$\mathbb{E}_{x \sim p(x)}[\,||\,\nabla_x \log p(x) - s_\theta(x)\,||^2\,]$$

Unfortunately, $\nabla_x \log p(x)$ is not accessible directly

There are two alternative ways to train $s_\theta(x)$

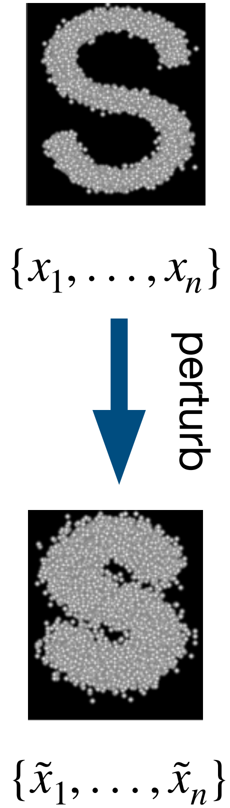
- Implicit score matching (Hyvärinen, 2005)
- **Denoising score matching** (Vincent, 2010)

Denoising Score Matching

Perturbed Dataset

- For the dataset $\{x_1, ..., x_n\}$ we consider the **perturbed** dataset $\{\tilde{x}_1, ..., \tilde{x}_n\}$
- We sample the perturbed data by adding Gaussian noise $\tilde{x} = x + \sigma z$ where $z \sim \mathcal{N}(0,I)$
- For now we keep the **noise level** $\sigma > 0$ fixed
- The smaller the noise level σ, the closer the densities p_σ and p

$$\lim_{\sigma \to 0} \mathrm{KL}(p_\sigma || p) = 0$$



Denoising Score Matching

The Score of the Perturbed Dataset

- We can write the perturbed distribution $p_\sigma(x)$ as

$$p_\sigma(\tilde{x}) = \int p_\sigma(\tilde{x} \,|\, x) p(x) dx = \mathbb{E}_{x \sim p(x)}[p_\sigma(\tilde{x} \,|\, x)]$$

- Since the **conditional density** $p_\sigma(\tilde{x} \,|\, x)$ is Gaussian, we can write

$$p_\sigma(\tilde{x} \,|\, x) = \frac{1}{\sqrt{(2\pi)^D \sigma^D}} \exp\left(-\frac{1}{2\sigma^2}(\tilde{x} - x)^T(\tilde{x} - x)\right)$$

- The score of the conditional density is

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

Denoising Score Matching

The Score of the Perturbed Dataset

We can train s_θ to approximate the score of the perturbed dataset using the identity

$$\arg\min_\theta \mathbb{E}_{\tilde{x}\sim p_\sigma(\tilde{x})}\left[||s_\theta(\tilde{x}) - \nabla_{\tilde{x}}\log p_\sigma(\tilde{x})||^2\right]$$
$$= \arg\min_\theta \mathbb{E}_{x\sim p(x),\tilde{x}\sim p_\sigma(\tilde{x}|x)}\left[||s_\theta(\tilde{x}) - \nabla_{\tilde{x}}\log p_\sigma(\tilde{x}\,|\,x)||^2\right]$$

Vincent (2010)

- All steps required in the second equation can be computed efficiently
- Assume we have trained s_θ for the perturbed dataset with noise level σ: **How can we use $\nabla_{\tilde{x}}\log p_\sigma(\tilde{x})$ to obtain a generative model for $p(x)$?**

Langevin Dynamics

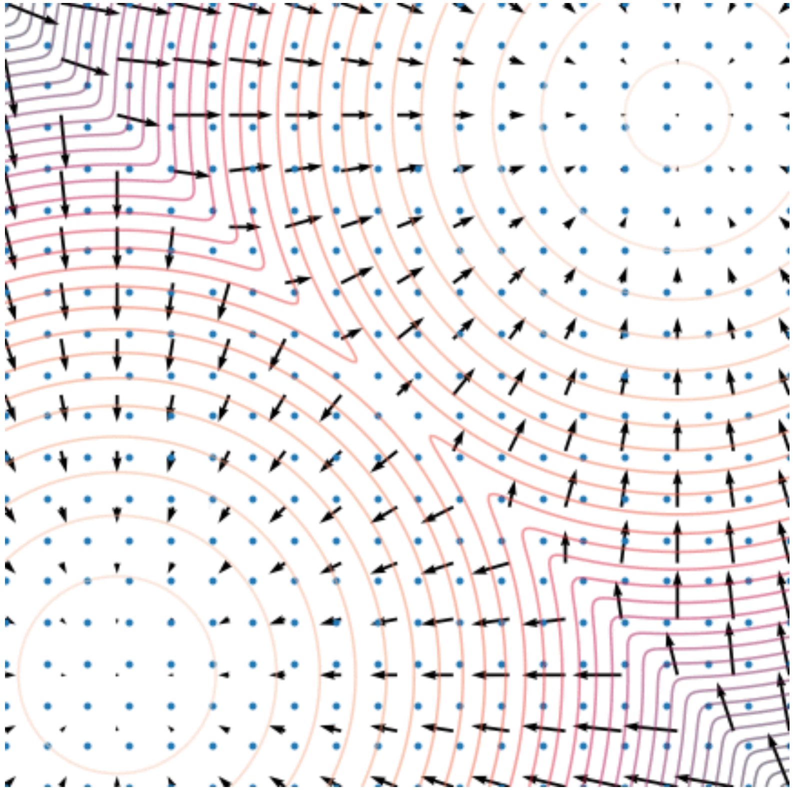
Langevin Dynamics

Langevin Dynamics: Consider a sample x_0 from an initialization distribution $\pi(x)$ and the iteration rule

$$x_{i+1} \leftarrow x_i + \epsilon \nabla_x \log p(x) + \sqrt{2\epsilon} z_i$$

for $i = 0,1,\ldots,K$ and $z_i \sim \mathcal{N}(0,I)$.

- The iterate x_K converges to a sample from $p(x)$ as $K \to \infty$ and $\epsilon \to 0$ (under regularity conditions)
- We can plug in the trained network s_θ for the score $\nabla_x \log p(x)$

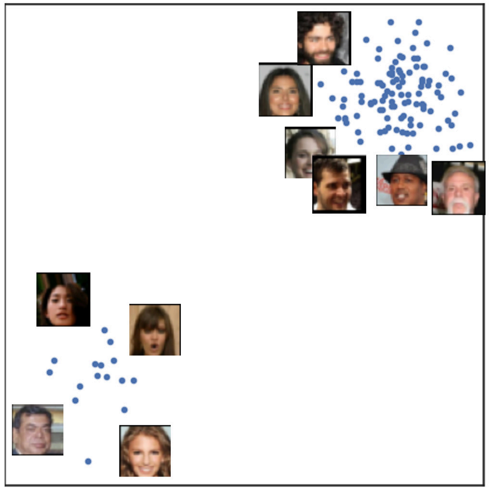


Source: yang-song.net/blog/2021/score

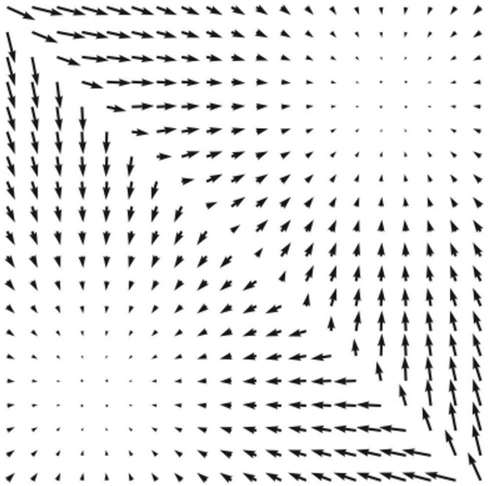
Langevin Dynamics

Score-based Generative Modeling

Source: yang-song.net/blog/2021/score



score matching



Langevin dynamics

Data samples

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

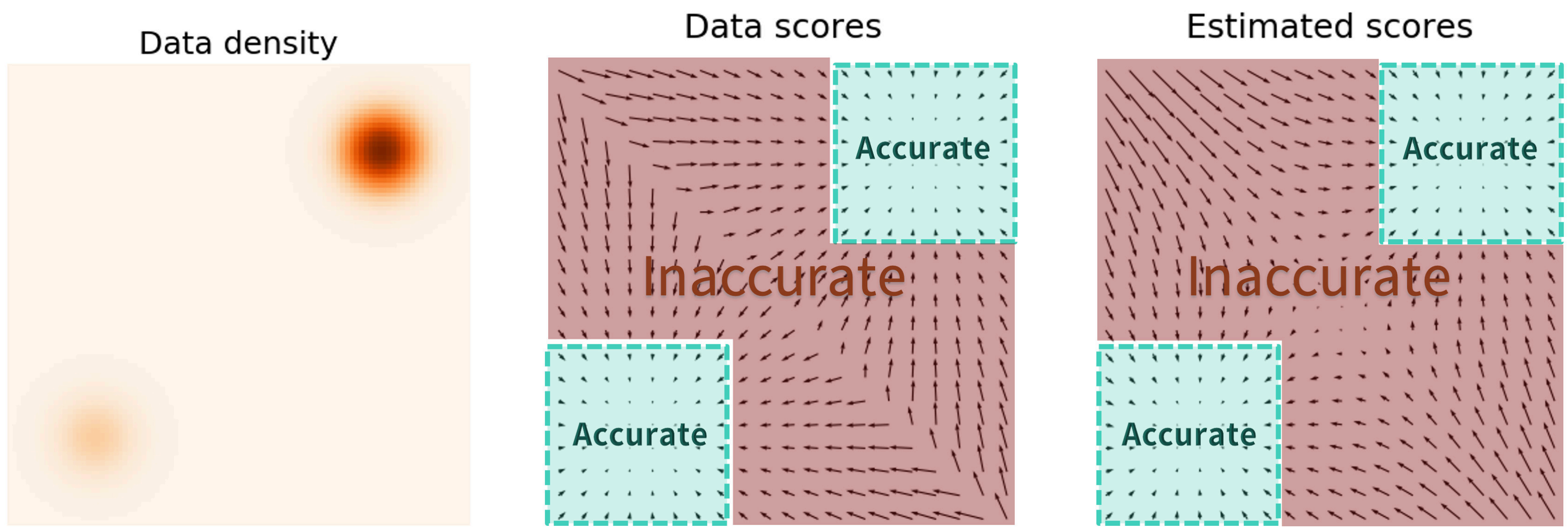
Scores

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

New samples

Langevin Dynamics

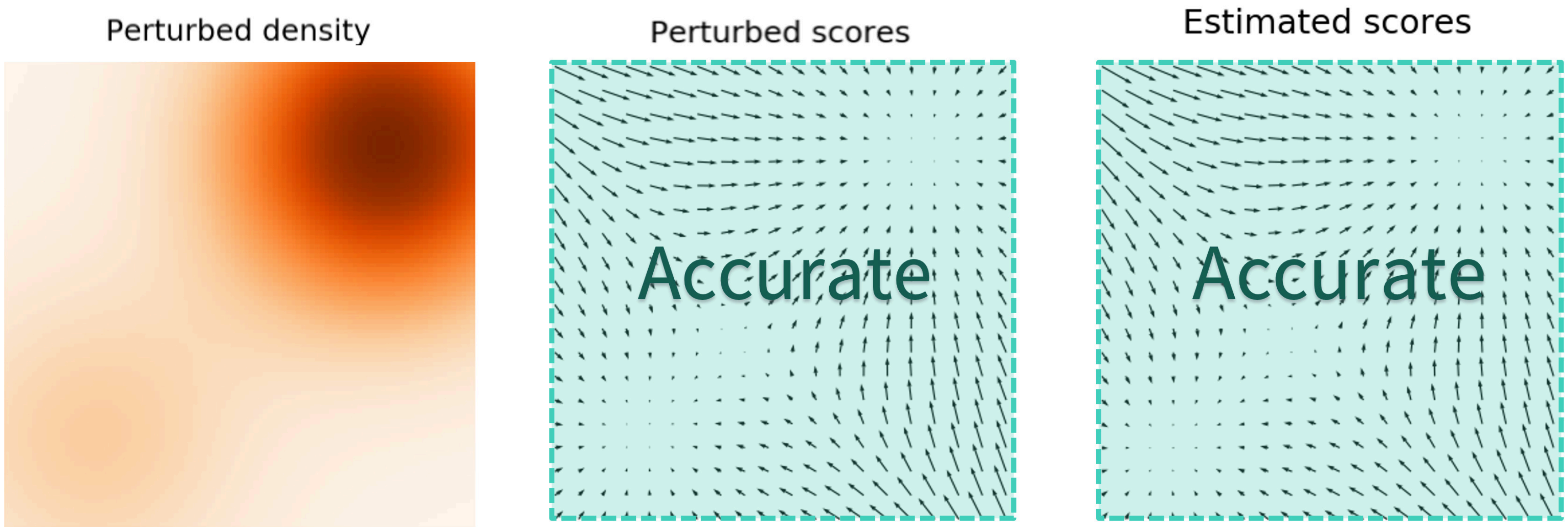
- For inference, samples are typically inside **low-density** regions
- The network s_θ is trained from data in high-density regions
- **Extrapolation** from low to high density is difficult → poor sample quality



Langevin Dynamics

- By increasing the noise level σ of the perturbed dataset, samples cover larger regions of the perturbed data space

- In this case, the perturbation is too large and $p(x) \not\approx p_\theta(x)$



Annealed Langevin Dynamics

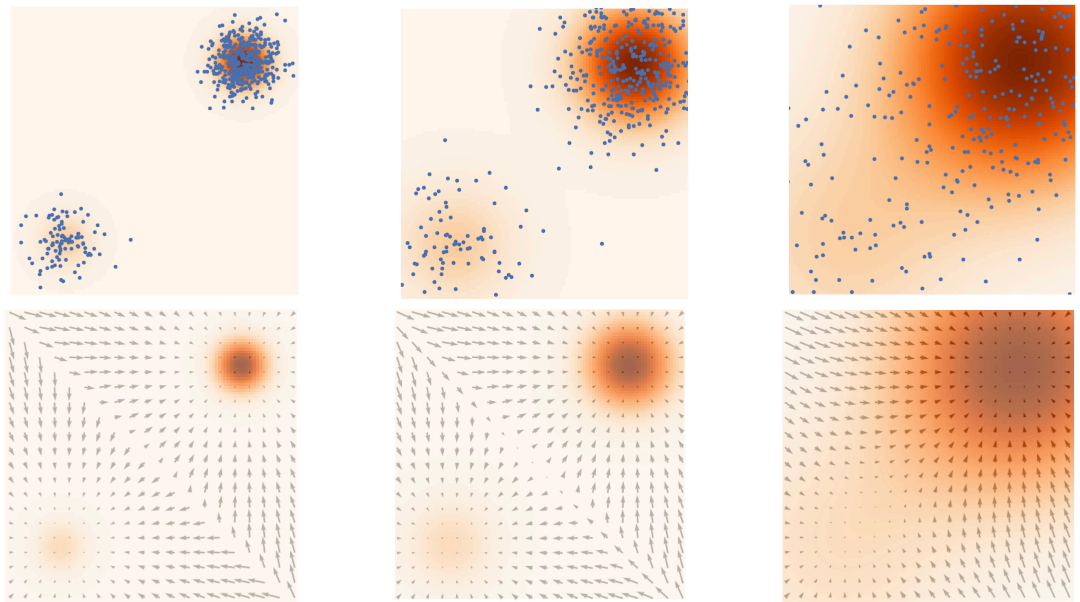
- Consider multiple noise scales

$$0 < \sigma_1 < \sigma_2 < \ldots < \sigma_L$$

  and train a network $s_\theta(x, \sigma_i)$ with the noise scale σ_i as additional input

- Repeatedly apply Langevin Dynamics for each noise scale, starting from the largest noise σ_L until the smallest noise σ_1

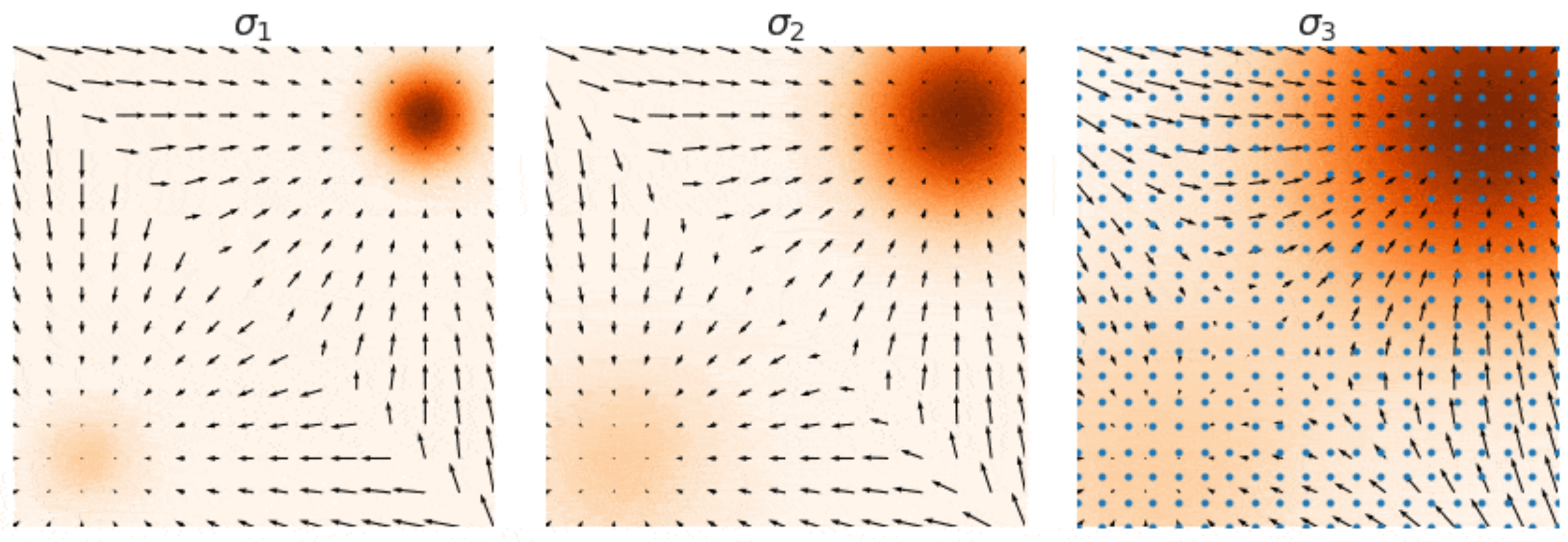
$\sigma_1 < \sigma_2 < \sigma_3$



Source: yang-song.net/blog/2021/score

Langevin Dynamics

Annealed Langevin Dynamics



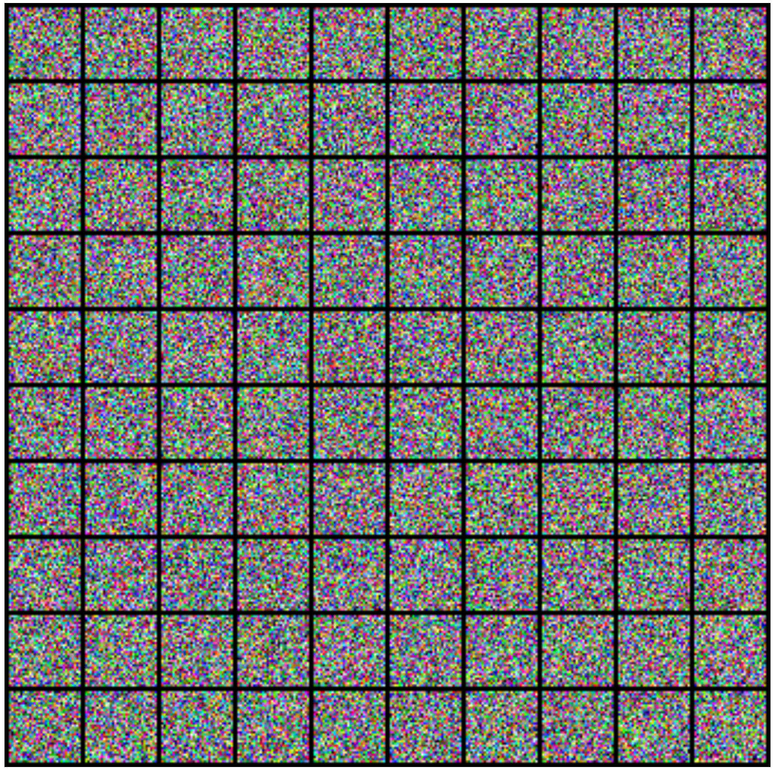
Source: yang-song.net/blog/2021/score

Annealed Langevin Dynamics

Summary

- Denoising score matching works well even for high-dimensional data such as images.
- No need to backpropagate gradients through many steps → method is much more scalable than CNFs
- Specifying a good sequence of noise scales is critical
- Inference requires many evaluations of $s_\theta(x, \sigma_i)$
- We can sample from $p(x)$ but not directly compute likelihoods
- No maximum likelihood training

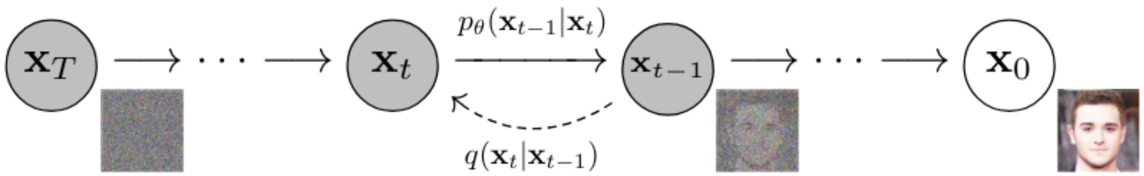
CIFAR 10



Source: yang-song.net/blog/2021/score

Diffusion Models

Denoising Diffusion Probabilistic Model



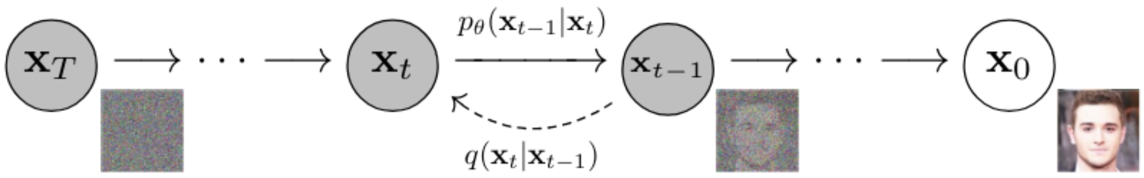
Ho et al. 2020

- Diffusion models are latent variable models of the form

$$p_\theta(x_0) = \int p_\theta(x_{0:T})dx_{1:T}$$

where $x_1, \ldots, x_T$ are latents with the same dimensionality as $x_0 \sim q(x_0)$

Denoising Diffusion Probabilistic Model



Ho et al. 2020

- Reverse process: (Markov chain with learned Gaussian transition)

$$p_\theta(x_{0:T}) := p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}\,|\,x_t) \quad \text{and} \quad p_\theta(x_{t-1}\,|\,x_t) = \mathcal{N}\left(\mu_\theta(x_t, t), \beta_t I\right)$$

- Forward process (Markov chain that adds Gaussian noise):

$$q(x_{1:T}\,|\,x_0) := \prod_{t=1}^{T} q(x_t\,|\,x_{t-1}) \quad \text{and} \quad q(x_t\,|\,x_{t-1}) = \mathcal{N}\left(\sqrt{1-\beta_t}x_{t-1}, \beta_t I\right)$$

Denoising Diffusion Probabilistic Model

- We set the noise scales β_t as hyperparameters (usually $T = 1000$, $\beta_0 = 10^{-4}, \beta_T = 0.02$)
- Given data x_0, we can sample the noisy latent x_t via

$$q(x_t \,|\, x_0) = \mathcal{N}(x_t, \sqrt{\overline{\alpha}_t} x_0, (1 - \overline{\alpha}_t) I))$$

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

- **Training objective (variational bound on the likelihood):**

$$\mathbb{E}[-\log p_\theta(x_0)] \leq \mathbb{E}\left[-\log p(x_T) - \sum_{t \geq 1} \log \frac{p_\theta(x_{t-1} \,|\, x_t)}{q(x_t \,|\, x_{t-1})}\right]$$

Denoising Diffusion Probabilistic Model

- We can reformulate the training objective as

$$\mathbb{E}\left[\underbrace{\boxed{\mathrm{KL}(q(x_T|x_0)\,||\,p(x_T))}}_{L_T} + \sum_{t>1}\underbrace{\boxed{\mathrm{KL}(q(x_{t-1}|x_t,x_0)\,||\,p_\theta(x_{t-1}|x_t))}}_{L_{t-1}} - \underbrace{\boxed{\log p_\theta(x_0|x_1)}}_{L_1}\right]$$

- L_T does not depend on θ
- L_1 is easy to train (continuous to discrete decoder)
- L_{t-1} is the KL-divergence between two Gaussian distributions

Denoising Diffusion Probabilistic Model

Analysing L_{t-1}

- The Gaussian for the forward process is

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

with

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t \quad \text{and} \quad \tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\beta_t$$

- L_{t-1} simplifies to matching the means of the forward and reverse process

$$L_{t-1} = \mathbb{E}\left[\frac{1}{2\beta_t} \,||\, \tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, x) \,||^2\right] + C$$

Denoising Diffusion Probabilistic Model

ϵ-prediction

- We can write

$$x_t(x_0, \epsilon) = \sqrt{\overline{\alpha}_t} x_0 + \sqrt{1 - \overline{\alpha}_t} \epsilon \quad \text{for} \quad \epsilon \sim \mathcal{N}(0,I)$$

- Instead of predicting the mean $\mu_\theta(x_t, t)$, predict the noise $\epsilon_\theta(x_t, t)$

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

- L_{t-1} changes to

$$L_{t-1} = \mathbb{E}_{x_0, \epsilon} \left[\frac{\beta_t^2}{2\beta_t \alpha_t (1 - \overline{\alpha}_t)} || \epsilon - \epsilon_\theta(\sqrt{\overline{\alpha}_t} x_0 + \sqrt{1 - \overline{\alpha}_t} \epsilon, t) ||^2 \right] + C$$

Denoising Diffusion Probabilistic Model

Training and Inference

- In practice the weightings of the individual terms are often dropped

$$L_{\mathrm{DM}}(\theta) := \mathbb{E}_{t,x_0,\epsilon}\left[\,||\,\epsilon - \ \epsilon_\theta(\underbrace{\sqrt{\overline{\alpha}_t}x_0 + \sqrt{1-\overline{\alpha}_t}\epsilon}_{x_t},\, t)\ ||^2\right]$$

Algorithm 1 Training

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

Algorithm 2 Sampling

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