Image: Synthetic ImageNet samples the diffusion model by Dhariwal and Nichol

COMP547

DEEP UNSUPERVISED LEARNING

Lecture #10 – Score-Based and Denoising Diffusion Models

KOÇ UNIVERSITY

Aykut Erdem // Koç University // Spring 2022

Previously on COMP547

- Motivation & Definition of Implicit Models
- Original GAN (Goodfellow et al, 2014)
- Evaluation: Parzen, Inception, Fréchet
- Theory of GANs
- GAN Progression
- Conditional GANs, Cycle-Consistent Adversarial Networks
- GANs and Representations
- Applications

Artificial faces synthesized by StyleGAN (Nvidia)

Lecture overview

- Energy-based models
- Score-based Models
- Denoising Diffusion Models

Disclaimer: Much of the material and slides for this lecture were borrowed from
—Stefano Ermon and Aditya Grover’s Stanford CS236 class
—Yang Song and Stefano Ermon’s talk titled "Generative Modeling by Estimating Gradients of the Data Distribution"
—Jascha Sohl-Dickstein’s talk "Deep Unsupervised Learning using Nonequilibrium Thermodynamics"
—Sangwoo Mo’s talk titled “Introduction to Diffusion Models”

Lecture overview

- Energy-based models
- Score-based Models
- Denoising Diffusion Models

Inspired by their work, we further investigated the relationship between diffusion models and score-based generative models in an ICLR 2021 paper [20]. We found that the sampling method of diffusion probabilistic models can be integrated with annealed Langevin dynamics of score-based models to create a unified and more powerful sampler (the Predictor-Corrector sampler). By generalizing the number of noise scales to infinity, we further proved that score-based generative models and diffusion probabilistic models can both be viewed as discretizations to stochastic differential equations determined by score functions. This work bridges both score-based generative modeling and diffusion probabilistic modeling into a unified framework.

Collectively, these latest developments seem to indicate that both score-based generative modeling with multiple noise perturbations and diffusion probabilistic models are different perspectives of the same model family, much like how wave mechanics and matrix mechanics are equivalent formulations of quantum mechanics in the history of physics [5]. The perspective of score matching and score-based models allows one to calculate log-likelihoods exactly, solve inverse problems naturally, and is directly connected to energy-based models, Schrödinger bridges and optimal transport [47]. The perspective of diffusion models is naturally connected to VAEs, lossy compression, and can be directly incorporated with variational probabilistic inference. This blog post focuses on the first perspective, but I highly recommend interested readers to learn about the alternative perspective of diffusion models as well (see a great blog by Lilian Weng).

https://yang-song.github.io/blog/2021/score/

Disclaimer: Much of the material and slides fo
—Stefano Ermon and Aditya Grover's Stanford CS236
—Yang Song and Stefano Ermon's talk titled "Generati
—Jascha Sohl-Dickstein's talk "Deep Unsupervised Le
—Sangwoo Mo's talk titled "Introduction to Diffusion Models"

Lecture overview

- **Energy-based models**
  - **Parametrizing probability distributions**
  - Energy-based generative modeling
  - Ising Model, Product of Experts, Restricted Boltzmann machine, Deep Boltzman Machines
  - Training and sampling from EBMs
- Score-based Models
- Denoising Diffusion Models

Parameterizing probability distributions

- Probability distributions $p(x)$ are a key building block in generative modeling.
  1. Non-negative: $p(x) \geq 0$
  2. Sum-to-one: $\sum_x p(x) = 1$ (or $\int p(x)dx = 1$ for continuous variables)

- Coming up with a non-negative function $p_\theta(\mathbf{x})$ is not hard.
  Given any function $f_\theta(\mathbf{x})$, we can choose
  - $g_\theta(\mathbf{x}) = f_\theta(\mathbf{x})^2$
  - $g_\theta(\mathbf{x}) = \exp(f_\theta(\mathbf{x}))$
  - $g_\theta(\mathbf{x}) = |f_\theta(\mathbf{x})|$
  - $g_\theta(\mathbf{x}) = \log(1 + \exp(f_\theta(\mathbf{x})))$

Parameterizing probability distributions

- Probability distributions $p(x)$ are a key building block in generative modeling.
  1. Non-negative: $p(x) \geq 0$
  2. Sum-to-one: $\sum_x p(x) = 1$ (or $\int p(x)dx = 1$ for continuous variables)

     Sum-to-one is key!

- Total "volume" is fixed: increasing $p(x_{train})$ guarantees that x_{train} becomes relatively more likely (compared to the rest)

- Problem:
  - $g_\theta(\mathbf{x}) \geq 0$ is easy, but $g_\theta(\mathbf{x})$ might not sum-to-one.
  - $\sum_x g_\theta(\mathbf{x}) = Z(\theta) \neq 1$ in general, so $g_\theta(\mathbf{x})$ is not a valid probability mass function or density

Parameterizing probability distributions

- **Problem:** $g_\theta(\mathbf{x}) \geq 0$ is easy, but $g_\theta(\mathbf{x})$ might not be normalized
- **Solution:** $p_\theta(\mathbf{x}) = \frac{1}{\mathrm{Volume}(g_\theta)} g_\theta(\mathbf{x}) = \frac{1}{\int g_\theta(\mathbf{x})\mathrm{d}\mathbf{x}} g_\theta(\mathbf{x})$

  Then by definition $\int p_\theta(\mathbf{x})\mathrm{d}\mathbf{x} = 1$
- **Example:** Choose $g_\theta(\mathbf{x})$ so that the volume is **analytically** as a function of θ.
  1. $g_{(\mu,\sigma)}(x) = e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. **Volume** is: $\int e^{-\frac{x-\mu}{2\sigma^2}} dx = \sqrt{2\pi\sigma^2}$ → **Gaussian**
  2. $g_\lambda(x) = e^{-\lambda x}$. **Volume** is: $\int_0^{+\infty} e^{-\lambda x} dx = \frac{1}{\lambda}$ → **Exponential**
  3. $g_\theta(x) = h(x) \exp\{\theta \cdot T(x)\}$. **Volume** is $\exp\{A(\theta)\}$, where $A(\theta) = \log \int h(x) \exp\{\theta \cdot T(x)\}\mathrm{d}\mathbf{x}$ → **Exponential family**
     - Normal, Poisson, exponential, Bernoulli
     - beta, gamma, Dirichlet, Wishart, etc.
- Function forms $g_\theta(\mathbf{x})$ need to allow **analytical** integration. Despite being restrictive, they are very useful as building blocks for more complex distributions.

Lecture overview

- **Energy-based models**
  - Parametrizing probability distributions
  - **Energy-based generative modeling**
  - Ising Model, Product of Experts, Restricted Boltzmann machine, Deep Boltzman Machines
  - Training and sampling from EBMs
- Score-based Models
- Denoising Diffusion Models

Likelihood based learning

- **Problem:** $g_\theta(\mathbf{x}) \geq 0$ is easy, but $g_\theta(\mathbf{x})$ might not be normalized
- **Solution:** $p_\theta(\mathbf{x}) = \frac{1}{\text{Volume}(g_\theta)} g_\theta(\mathbf{x}) = \frac{1}{\int g_\theta(\mathbf{x}) \mathrm{d}\mathbf{x}} g_\theta(\mathbf{x})$

  Typically, choose $g_\theta(\mathbf{x})$ so that we know the volume analytically. More complex models can be obtained by combining these building blocks.

  1. **Autoregressive:** Products of normalized objects $p_\theta(\mathbf{x}) p_{\theta'(\mathbf{x})}(\mathbf{y})$

     $$\int_\mathbf{x} \int_\mathbf{y} p_\theta(\mathbf{x}) p_{\theta'(\mathbf{x})}(\mathbf{y}) \mathrm{d}\mathbf{x}\mathrm{d}\mathbf{y} = \int_\mathbf{x} p_\theta(\mathbf{x}) \underbrace{\int_\mathbf{y} p_{\theta'(\mathbf{x})}(\mathbf{y}) \mathrm{d}\mathbf{y}}_{=1} \; \mathrm{d}\mathbf{x} = \int_\mathbf{x} p_\theta(\mathbf{x}) \mathrm{d}\mathbf{x} = 1$$

  2. **Latent variables:** Mixtures of normalized objects $\alpha p_\theta(\mathbf{x}) + (1-\alpha) p_{\theta'}(\mathbf{x})$

     $$\int_\mathbf{x} \alpha p_\theta(\mathbf{x}) + (1-\alpha) p_{\theta'}(\mathbf{x}) \mathrm{d}\mathbf{x} = \alpha + (1-\alpha) = 1$$

Likelihood based learning

- **Problem:** $g_\theta(\mathbf{x}) \geq 0$ is easy, but $g_\theta(\mathbf{x})$ might not be normalized
- **Solution:** $p_\theta(\mathbf{x}) = \frac{1}{\text{Volume}(g_\theta)} g_\theta(\mathbf{x}) = \frac{1}{\int g_\theta(\mathbf{x})\mathrm{d}\mathbf{x}} g_\theta(\mathbf{x})$

  Typically, choose $g_\theta(\mathbf{x})$ so that we know the volume analytically. More complex models can be obtained by combining these building blocks.

  1. **Autoregressive:** Products of normalized objects $p_\theta(\mathbf{x})p_{\theta'(\mathbf{x})}(\mathbf{y})$

     $$\int_{\mathbf{x}}\int_{\mathbf{y}} p_\theta(\mathbf{x})p_{\theta'(\mathbf{x})}(\mathbf{y})\mathrm{d}\mathbf{x}\mathrm{d}\mathbf{y} = \int_{\mathbf{x}} p_\theta(\mathbf{x}) \underbrace{\int_{\mathbf{y}} p_{\theta'(\mathbf{x})}(\mathbf{y})\mathrm{d}\mathbf{y}}_{=1} \; \mathrm{d}\mathbf{x} = \int_{\mathbf{x}} p_\theta(\mathbf{x})\mathrm{d}\mathbf{x} = 1$$

  2. **Latent variables:** Mixtures of normalized objects $\alpha p_\theta(\mathbf{x}) + (1-\alpha)p_{\theta'}(\mathbf{x})$

     $$\int_{\mathbf{x}} \alpha p_\theta(\mathbf{x}) + (1-\alpha)p_{\theta'}(\mathbf{x})\mathrm{d}\mathbf{x} = \alpha + (1-\alpha) = 1$$

How about using models where the "volume"/normalization constant of $g_\theta(\mathbf{x})$ is not easy to compute analytically?

Energy-based model

$$p_\theta(\mathbf{x}) = \frac{1}{\int \exp(f_\theta(\mathbf{x}))d\mathbf{x}} \exp\left(f_\theta(\mathbf{x})\right) = \frac{1}{Z(\theta)} \exp\left(f_\theta(\mathbf{x})\right)$$

The output of f_θ is a scalar value between $-\infty$ and ∞.

- The volume/normalization constant

$$Z(\theta) = \int \exp\left(f_\theta(\mathbf{x})\right) d\mathbf{x}$$

is also called the **partition** function. Why exponential (and not e.g. $f_\theta(\mathbf{x})^2$)?

1. Want to capture very large variations in probability. log-probability is the natural scale we want to work with. Otherwise need highly non-smooth f_θ.
2. Exponential families. Many common distributions can be written in this form.
3. These distributions arise under fairly general assumptions in statistical physics (maximum entropy, second law of thermodynamics).
   - $-f_\theta(\mathbf{x})$ is called the **energy**, hence the name.
   - Intuitively, configurations x with low energy (high $f_\theta(x)$) are more likely.

Energy-based model

$$p_\theta(\mathbf{x}) = \frac{1}{\int \exp(f_\theta(\mathbf{x}))d\mathbf{x}} \exp\left(f_\theta(\mathbf{x})\right) = \frac{1}{Z(\theta)} \exp\left(f_\theta(\mathbf{x})\right)$$

- Pros:
  - extreme flexibility: can use pretty much any function $f_\theta(\mathbf{x})$ you want
- Cons:
  - Sampling from $p_\theta(\mathbf{x})$ is hard
  - Evaluating and optimizing likelihood $p_\theta(\mathbf{x})$ is hard (learning is hard)
  - No feature learning (but can add latent variables)
- **Curse of dimensionality:** The fundamental issue is that computing $Z(\theta)$ numerically (when no analytic solution is available) scales exponentially in the number of dimensions of x.
- Nevertheless, **some tasks do not require knowing** $Z(\theta)$

Applications of Energy-based models

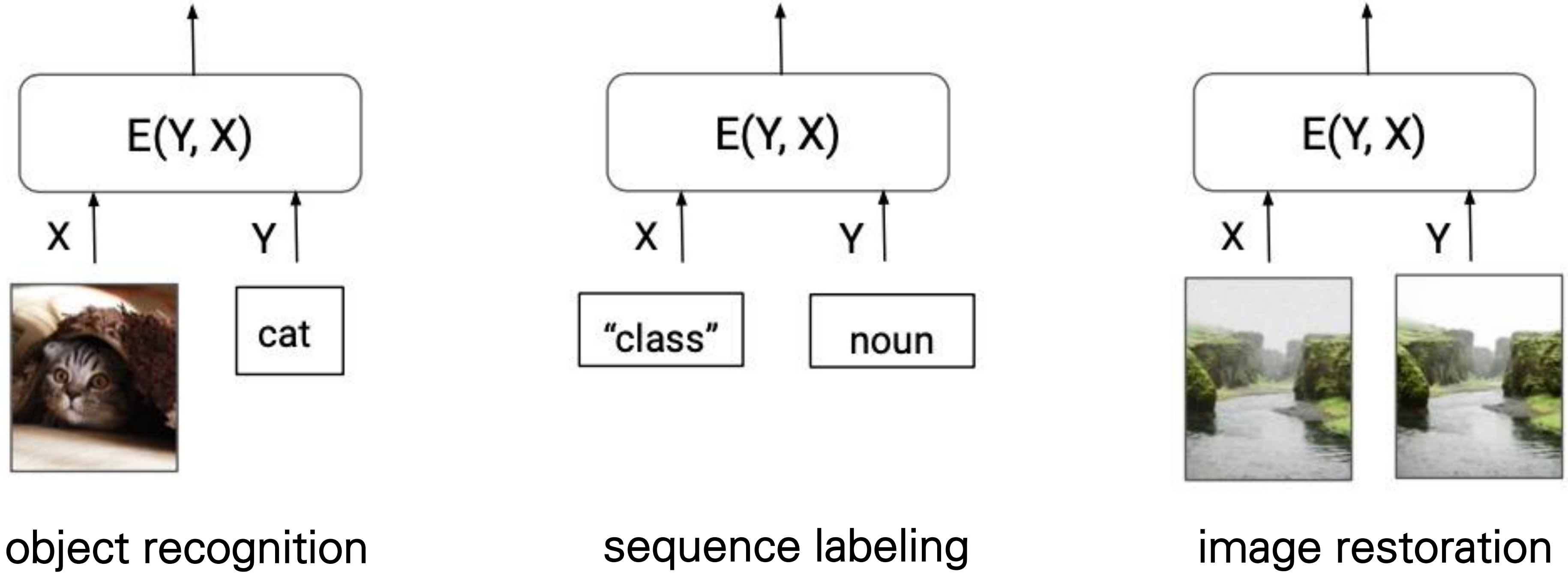
$$p_\theta(\mathbf{x}) = \frac{1}{\int \exp(f_\theta(\mathbf{x}))d\mathbf{x}} \exp\left(f_\theta(\mathbf{x})\right) = \frac{1}{Z(\theta)} \exp\left(f_\theta(\mathbf{x})\right)$$

- Given x, x′ evaluating $p_\theta(\mathbf{x})$ or $p_\theta\left(\mathbf{x}'\right)$ requires $Z(\theta)$.
- However, their ratio

$$\frac{p_\theta(\mathbf{x})}{p_\theta(\mathbf{x}')} = \exp\left(f_\theta(\mathbf{x}) - f_\theta\left(\mathbf{x}'\right)\right)$$

  does not involve $Z(\theta)$.
- This means we can easily check which one is more likely. Applications:
  - Anomaly detection
  - Denoising

Applications of Energy-based models



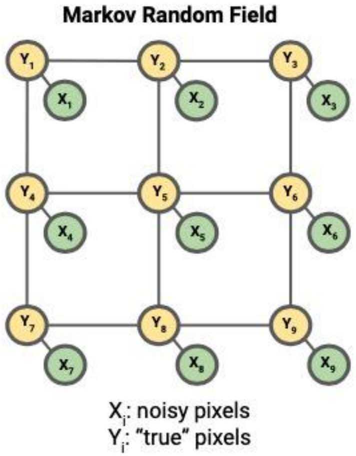
- Given a trained model, many applications require relative comparisons. Hence $Z(\theta)$ is not needed.

Lecture overview

- **Energy-based models**
  - Parametrizing probability distributions
  - Energy-based generative modeling
  - **Ising Model**, Product of Experts, Restricted Boltzmann machine, Deep Boltzman Machines
  - Training and sampling from EBMs
- Score-based Models
- Denoising Diffusion Models

Ising Model

- There is a true image $y \in \{0, 1\}^{3\times3}$, and a corrupted image $x \in \{0, 1\}^{3\times3}$. We know **x**, and want to somehow recover **y**.



- We model the joint probability distribution p(**y**, **x**) as

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \exp\left(\sum_i \psi_i\left(x_i, y_i\right) + \sum_{(i,j)\in E} \psi_{ij}\left(y_i, y_j\right)\right)$$

  - $\psi_i\left(x_i, y_i\right)$: the i-th corrupted pixel depends on the i-th original pixel
  - $\psi_{ij}\left(y_i, y_j\right)$: neighboring pixels tend to have the same value

- How did the original image **y** look like? Solution: maximize p(**y**|**x**). Or equivalently, maximize p(**y**, **x**).

Lecture overview

- **Energy-based models**
  - Parametrizing probability distributions
  - Energy-based generative modeling
  - Ising Model, **Product of Experts**, Restricted Boltzmann machine, Deep Boltzman Machines
  - Training and sampling from EBMs
- Score-Based Models
- Denoising Diffusion Models

Product of Experts

- Suppose you have trained several models $q_{\theta_1}(\mathbf{x}), r_{\theta_2}(\mathbf{x}), t_{\theta_3}(\mathbf{x})$. They can be different models (PixelCNN, Flow, etc.)
- Each one is like an expert that can be used to score how likely an input $\mathbf{x}$ is.
- Assuming the experts make their judgments indpendently, it is tempting to ensemble them as

$$p_{\theta_1}(\mathbf{x})q_{\theta_2}(\mathbf{x})r_{\theta_3}(\mathbf{x})$$

- To get a valid probability distribution, we need to normalize

$$p_{\theta_1,\theta_2,\theta_3}(\mathbf{x}) = \frac{1}{Z(\theta_1,\theta_2,\theta_3)} q_{\theta_1}(\mathbf{x})r_{\theta_2}(\mathbf{x})t_{\theta_3}(\mathbf{x})$$

- Note: similar to an AND operation (e.g., probability is zero as long as one model gives zero probability), unlike mixture models which behave more like OR

Product of Experts

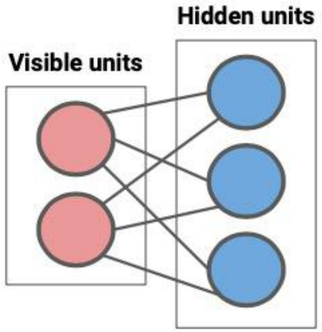
Image source: Du et al., 2020.

Lecture overview

- **Energy-based models**
  - Parametrizing probability distributions
  - Energy-based generative modeling
  - Ising Model, Product of Experts, **Restricted Boltzmann machine**, Deep Boltzman Machines
  - Training and sampling from EBMs
- Score-based Models
- Denoising Diffusion Models

Restricted Boltzmann machine (RBM)

- RBM: energy-based model with latent variables
- Two types of variables:
  - $\mathbf{x} \in \{0, 1\}^n$ are visible variables (e.g., pixel values)
  - $\mathbf{z} \in \{0, 1\}^m$ are latent ones
- The joint distribution is
  $p_{W,b,c}(\mathbf{x}, \mathbf{z}) = \frac{1}{Z} \exp\left(\mathbf{x}^T W \mathbf{z} + b\mathbf{x} + c\mathbf{z}\right) = \frac{1}{Z} \exp\left(\sum^n \sum^m x_i z_j w_{ij} + b\mathbf{x} + c\mathbf{z}\right)$



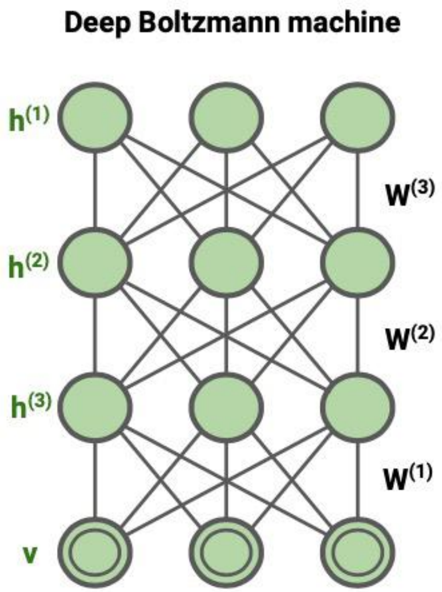
- Restricted because there are no visible-visible and hidden-hidden connections, i.e., x_i x_j or z_i z_j terms in the objective

Lecture overview

- **Energy-based models**
  - Parametrizing probability distributions
  - Energy-based generative modeling
  - Ising Model, Product of Experts, Restricted Boltzmann machine, **Deep Boltzman Machines**
  - Training and sampling from EBMs
- Score-based Models
- Denoising Diffusion Models

Deep Boltzmann Machines

- Stacked RBMs are one of the first deep generative models:



- Bottom layer variables v are pixel values. Layers above (**h**) represent "higher-level" features (corners, edges, etc.).
- Early deep neural networks for supervised learning had to be pre-trained like this to make them work.

Deep Boltzmann Machines: Samples

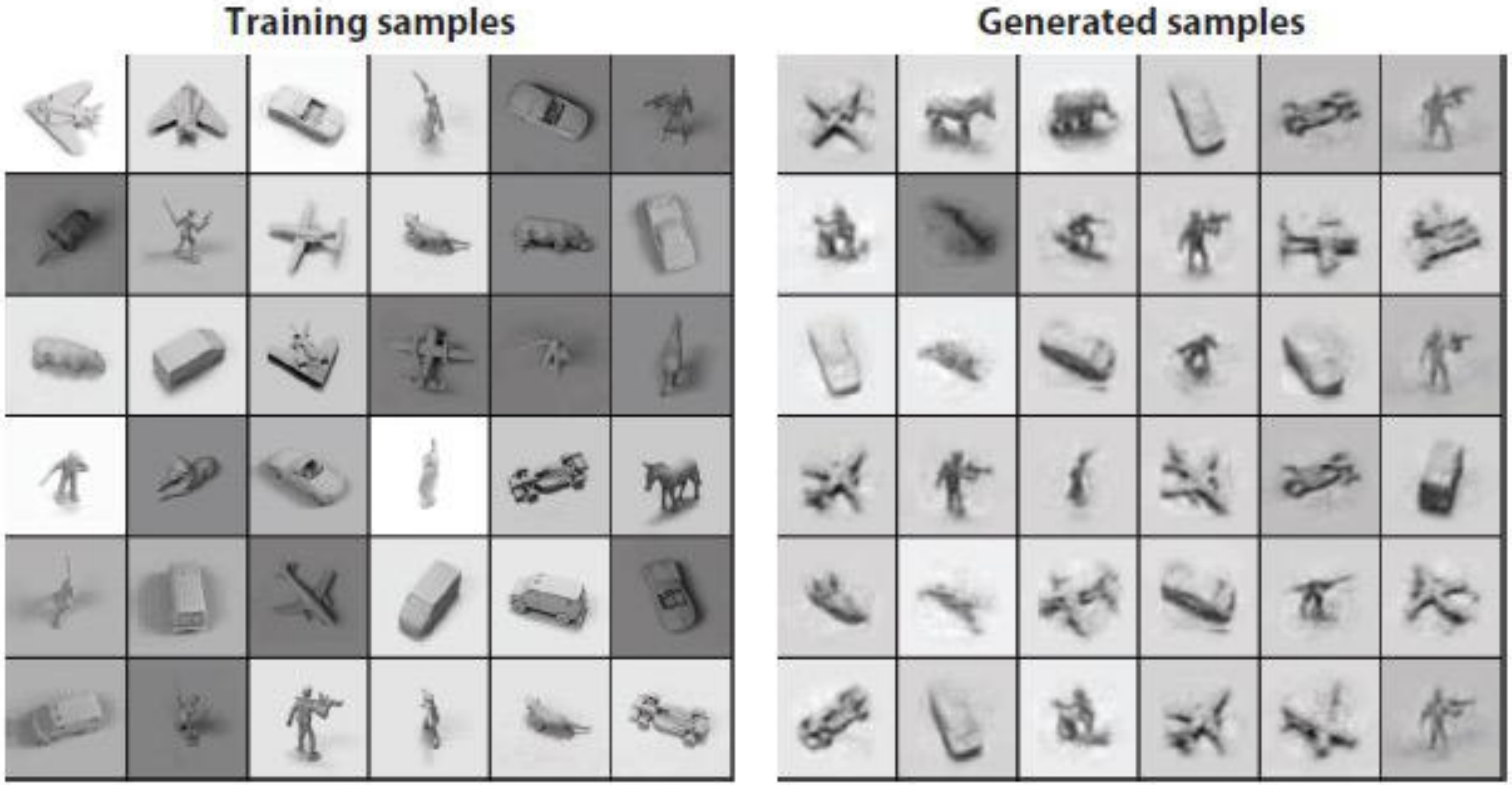


Image source: Salakhutdinov and Hinton, 2009.

Lecture overview

- **Energy-based models**
  - Parametrizing probability distributions
  - Energy-based generative modeling
  - Ising Model, Product of Experts, Restricted Boltzmann machine, Deep Boltzman Machines
  - **Training and sampling from EBMs**
- Score-based Models
- Denoising Diffusion Models

Energy-based model

$$p_\theta(\mathbf{x}) = \frac{1}{\int \exp(f_\theta(\mathbf{x}))d\mathbf{x}} \exp\left(f_\theta(\mathbf{x})\right) = \frac{1}{Z(\theta)} \exp\left(f_\theta(\mathbf{x})\right)$$

- Pros:
  - can plug in pretty much any function $f_\theta(\mathbf{x})$ you want
- Cons:
  - Sampling is hard
  - Evaluating likelihood (learning) is hard
  - No feature learning

- **Curse of dimensionality:** The fundamental issue is that computing $Z(\theta)$ numerically (when no analytic solution is available) scales exponentially in the number of dimensions of **x**

Computing the normalization constant is hard

- As an example, the RBM joint distribution is

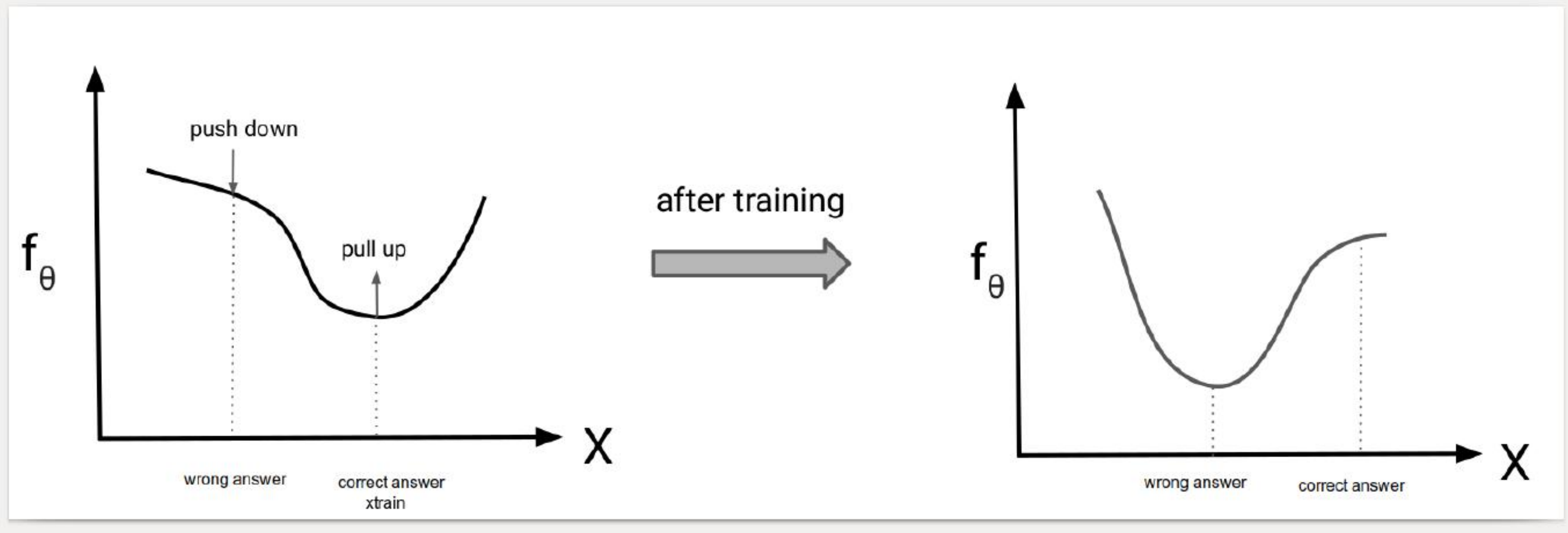
$$p_{W,b,c}(\mathbf{x}, \mathbf{z}) = \frac{1}{Z} \exp\left(\mathbf{x}^T W \mathbf{z} + b\mathbf{x} + c\mathbf{z}\right)$$

  where
  - $x \in \{0, 1\}^n$ are visible variables (e.g., pixel values)
  - $z \in \{0, 1\}^m$ are latent ones
- The normalization constant (the "volume") is

$$Z(W, b, c) = \sum_{\mathbf{x} \in \{0,1\}^n} \sum_{\mathbf{z} \in \{0,1\}^m} \exp\left(\mathbf{x}^T W \mathbf{z} + b\mathbf{x} + c\mathbf{z}\right)$$

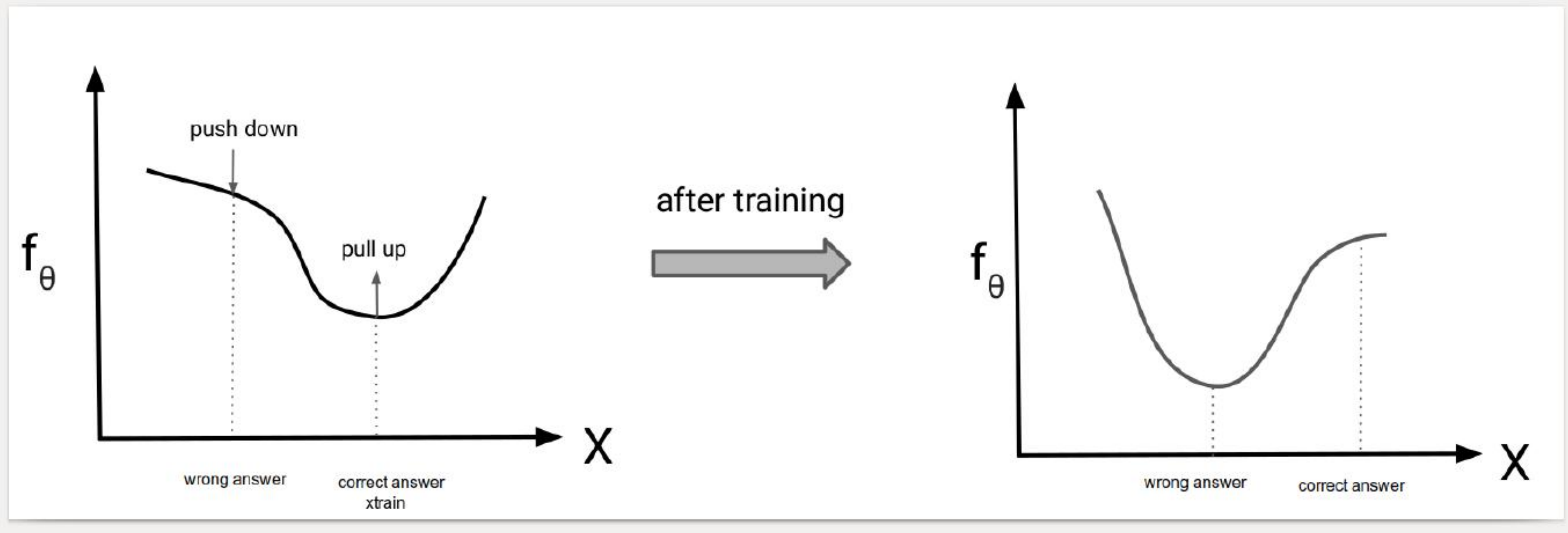
- **Note:** it is a well-defined function of the parameters W, b, c, but no simple closed-form. Takes time exponential in n, m to compute. This means that evaluating the objective function $p_{W,b,c}$ (x, z) for likelihood-based learning is hard.
- **Observation:** Optimizing the likelihood $p_{W\ b,c}$ (x, z) is difficult, but optimizing the un-normalized probability $\exp\left(\mathbf{x}^T W \mathbf{z} + b\mathbf{x} + c\mathbf{z}\right)$ (wrt trainable parameters W, b, c) is easy.

Training intuition



- **Goal**: $\text{maximize}\ \frac{\exp\{f_\theta(\mathbf{x}_{\text{train}})\}}{Z(\theta)}$. Increase numerator, decrease denominator.
- **Intuition**: because the model is not normalized, increasing the un-normalized log-probability $f_\theta(\mathbf{x}_{\text{train}})$ by changing θ does **not** guarantee that $\mathbf{x}_{\text{train}}$ becomes relatively more likely (compared to the rest).
- We also need to take into account the effect on other "wrong points" and try to "push them down" to also make $Z(\theta)$ small.

Constrastive Divergence



- **Goal**: $\text{maximize} \frac{\exp\{f_\theta(\mathbf{x}_{\text{train}})\}}{Z(\theta)}$
- **Idea**: Instead of evaluating.$Z(\theta)$ exactly, use a Monte Carlo estimate.
- Contrastive divergence algorithm: sample $x_{\text{sample}} \sim p_\theta$ take step on $\nabla_\theta \left(f_\theta\left(x_{\text{train}}\right) - f_\theta\left(x_{\text{sample}}\right)\right)$. Make training data more likely than typical sample from the model.

Contrastive Divergence

- Maximize log-likelihood: $\max_\theta f_\theta(x_{\text{train}}) - \log Z(\theta)$
- Gradient of log-likelihood:

$$
\begin{aligned}
& \nabla_\theta f_\theta(x_{\text{train}}) - \nabla_\theta \log Z(\theta) \\
= & \nabla_\theta f_\theta(x_{\text{train}}) - \frac{\nabla_\theta Z(\theta)}{Z(\theta)} \\
= & \nabla_\theta f_\theta(x_{\text{train}}) - \frac{1}{Z(\theta)} \int \nabla_\theta \exp\{f_\theta(x)\} \mathrm{d}x \\
= & \nabla_\theta f_\theta(x_{\text{train}}) - \frac{1}{Z(\theta)} \int \exp\{f_\theta(x)\} \nabla_\theta f_\theta(x) \mathrm{d}x \\
= & \nabla_\theta f_\theta(x_{\text{train}}) - \int \frac{\exp\{f_\theta(x)\}}{Z(\theta)} \nabla_\theta f_\theta(x) \mathrm{d}x \\
= & \nabla_\theta f_\theta(x_{\text{train}}) - E_{x_{\text{sample}}}[\nabla_\theta f_\theta(x_{\text{sample}})] \\
\approx & \nabla_\theta f_\theta(x_{\text{train}}) - \nabla_\theta f_\theta(x_{\text{sample}}),
\end{aligned}
$$

where $x_{\text{sample}} \sim \exp\{f_\theta(x_{\text{sample}})\}/Z(\theta)$

- How to sample?

Sampling from energy-based models

$$p_\theta(\mathbf{x}) = \frac{1}{\int \exp(f_\theta(\mathbf{x}))d\mathbf{x}} \exp\left(f_\theta(\mathbf{x})\right) = \frac{1}{Z(\theta)} \exp\left(f_\theta(\mathbf{x})\right)$$

- No direct way to sample like in autoregressive or flow models. Main issue: cannot easily compute how likely each possible sample is
- However, we can easily compare two samples x,x′.
- Use an iterative approach called Markov Chain Monte Carlo:
  1. Initialize x^0 randomly, t = 0
  2. Let $x' = x^t + \text{noise}$
     - If $f_\theta(x') > f_\theta(x^t)$, let $x^{t+1} = x'$
     - Else let $x^{t+1} = x'$ with probability $\exp(f_\theta(x') - f_\theta(x^t))$
  3. Goto step 2
- Works in theory, but can take a very long time to converge

Sampling from energy-based models

- For any continuous distribution $p_\theta(\mathbf{x})$, suppose we can compute its gradient (the **score function**) $\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})$
- Let $\pi(x)$ be a prior distribution that is easy to sample from.
- Langevin MCMC.
  - $\mathbf{x}^0 \sim \pi(\mathbf{x})$
  - Repeat $\mathbf{x}^{t+1} \sim \mathbf{x}^t + \epsilon \nabla_{\mathbf{x}} \log p_\theta\left(\mathbf{x}^t\right) + \sqrt{2\epsilon}\mathbf{z}^t$ for $t = 0, 1, 2, \cdots, T-1$, where $\mathbf{z}^t \sim \mathcal{N}(0, I)$.
  - If $\varepsilon \to 0$ and $T \to \infty$, we have $\mathbf{x}_T \sim p_\theta(\mathbf{x})$.
- Note that for energy-based models

$$
\begin{aligned}
\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) &= \nabla_{\mathbf{x}} f_\theta(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z(\theta)}_{=0} \\
&= \nabla_{\mathbf{x}} f_\theta(\mathbf{x})
\end{aligned}
$$

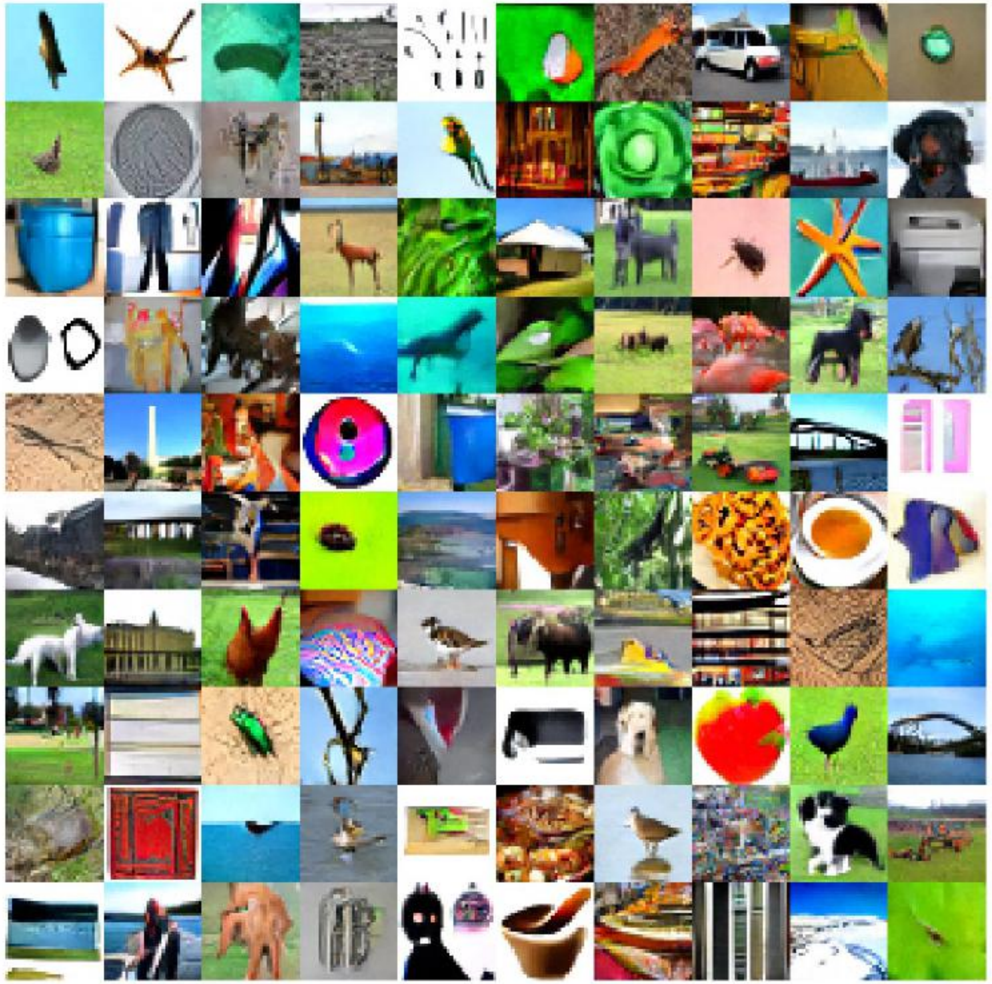
Modern energy-based models

Langevin sampling

Face samples

Image source: Nijkamp et al. 2019

Modern energy-based models



ImageNet samples

Image source: Du et al. 2019

Lecture overview

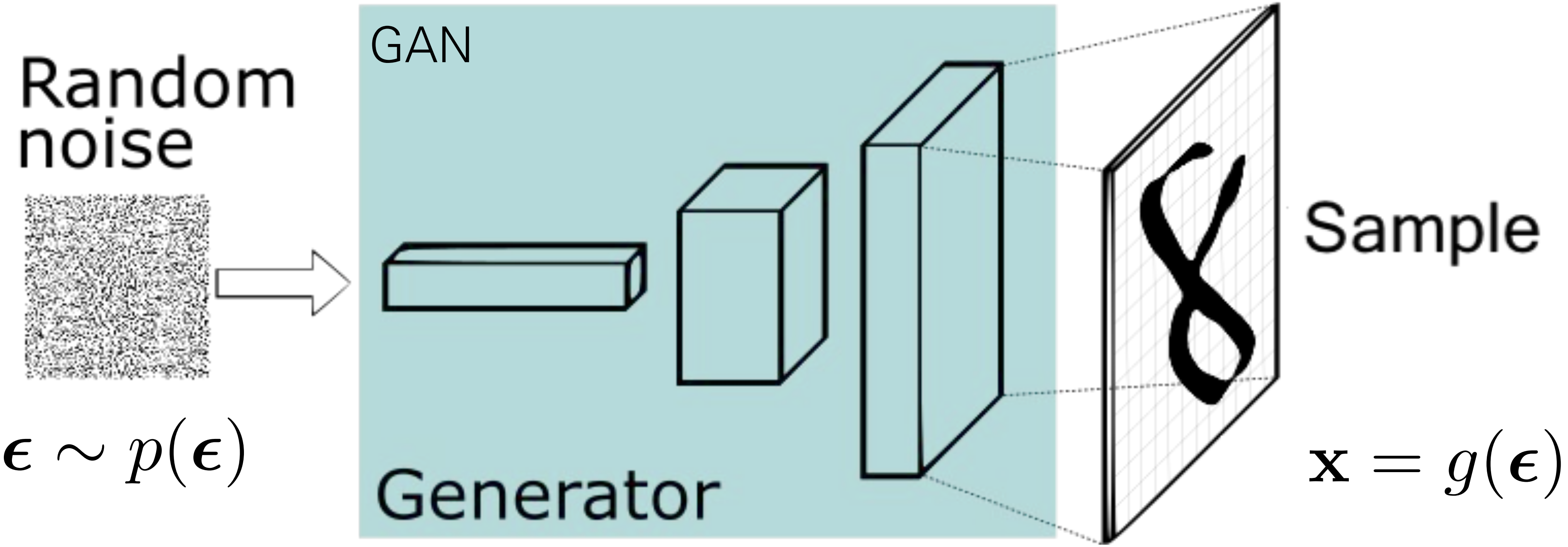
- Energy-based models
- **Score-based Models**
  - Probability distributions and Score functions
  - Score-based Generative Models
  - Sampling from a Score-based Model
  - Latent Score-based Generative Models
- Denoising Diffusion Models

Lecture overview

- Energy-based models
- **Score-based Models**
  - **Probability Distribution and Score functions**
  - Score-based Generative Models
  - Sampling from a Score-based Model
  - Latent Score-based Generative Models
- Denoising Diffusion Models

Representations of Probability Distributions

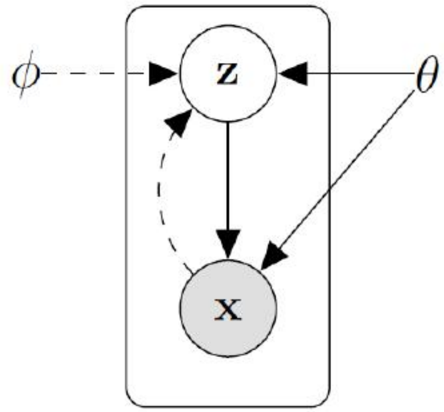
Implicit models: directly represent the sampling process



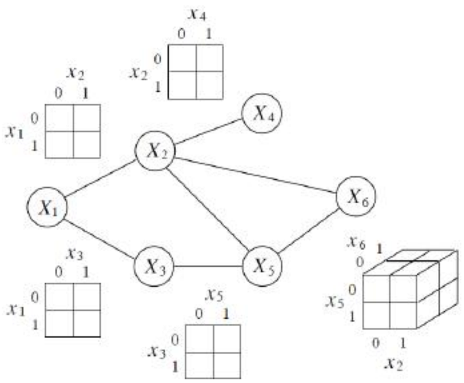
- **Cons:** hard to train, no likelihood, no principled model comparisons

Representations of Probability Distributions

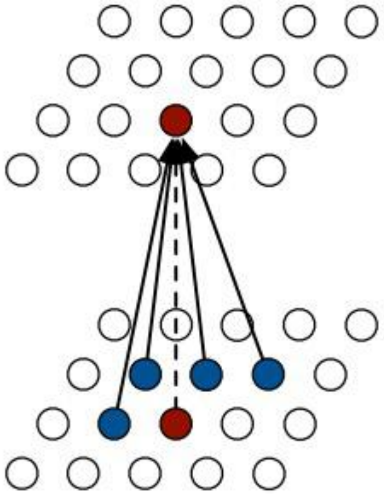
Explicit models: represent a probability density/mass function $p(\mathbf{x})$



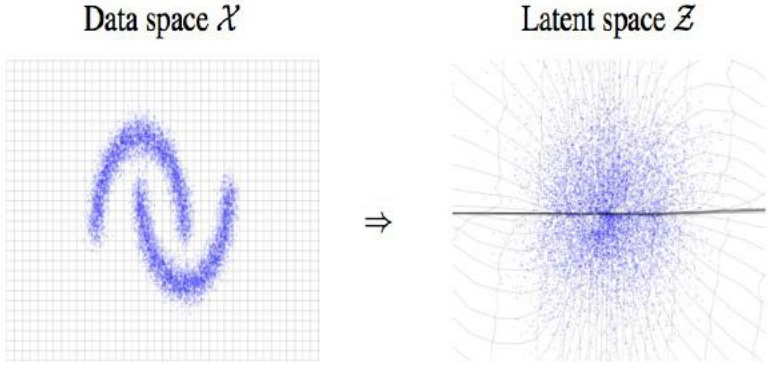
Bayesian networks (e.g., VAEs)



MRF



Autoregressive models



Flow models

- **Cons**: need to be normalized → balance expressivity and tractability

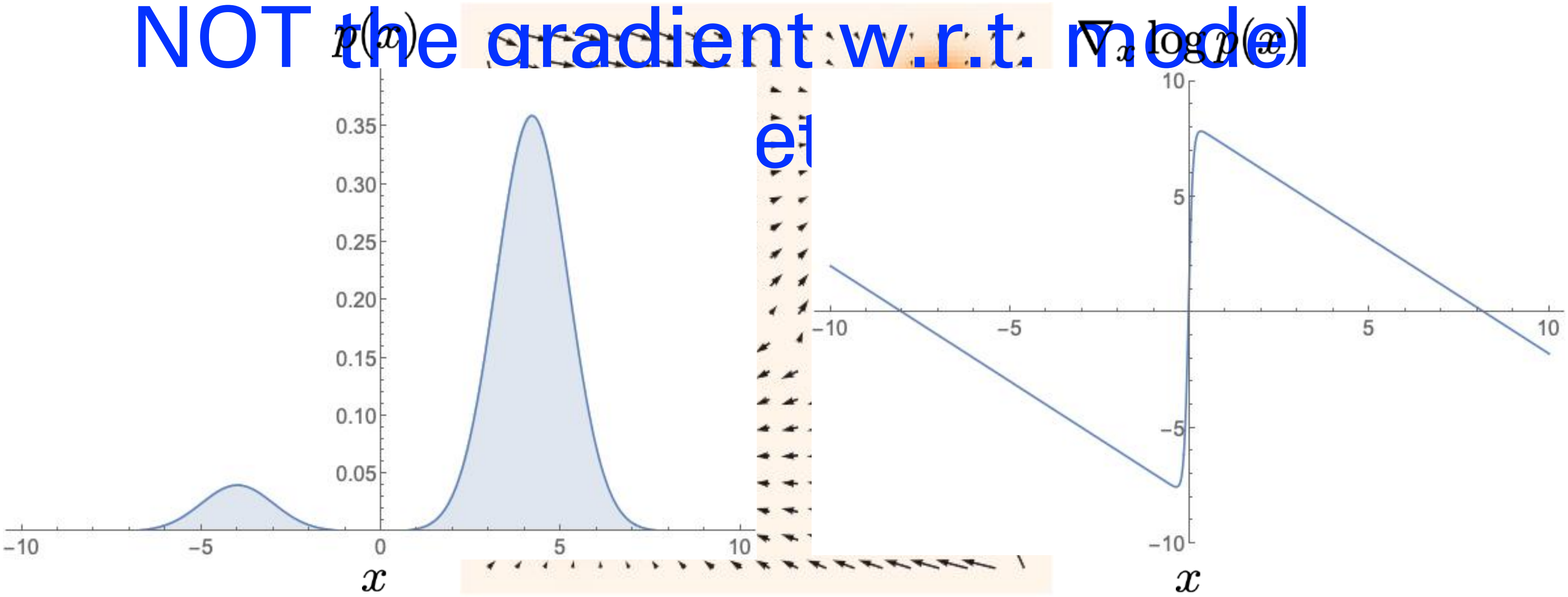
Representation of Probability Distributions

Alternative: The gradient of a probability density wrt the input dimensions

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}) \quad \text{Score}$$

NOT the gradient w.r.t. model

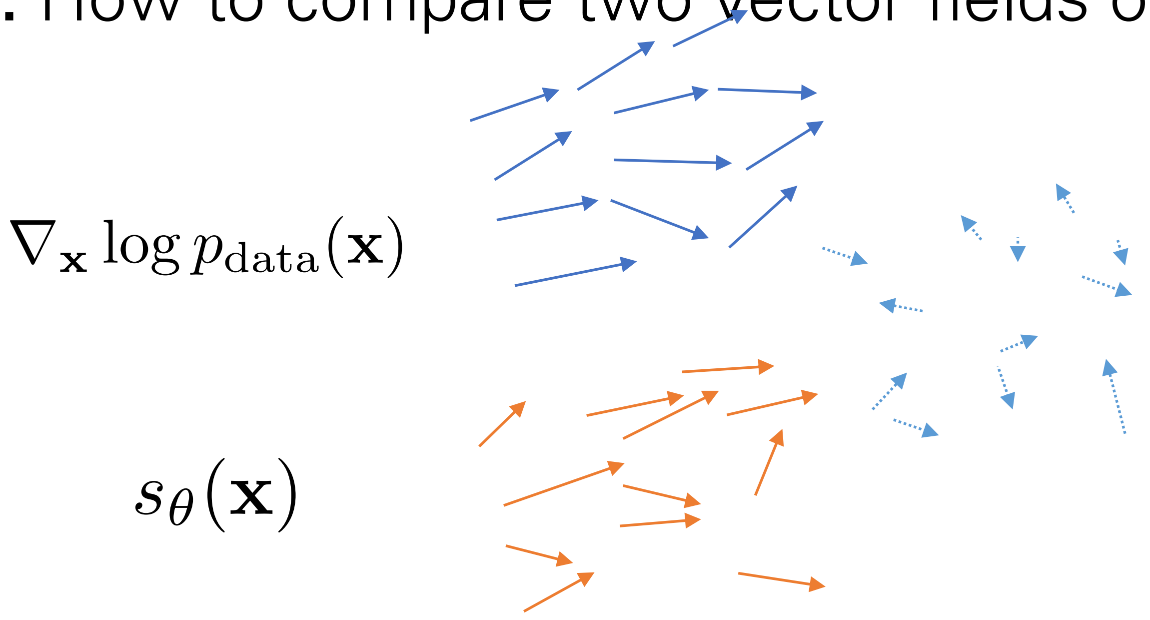
et



(pdf and score)

Score Estimation

- **Given:** i.i.d. samples $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \overset{\text{i.i.d.}}{\sim} p_{\text{data}}(\mathbf{x})$
- **Task:** Estimating the score $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$
- **Score Model:** A trainable vector-valued function $s_\theta(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}^D$
- **Objective:** How to compare two vector fields of scores?



Average
Euclidean distance
over the space

Score Estimation

- **Given:** i.i.d. samples $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \overset{\text{i.i.d.}}{\sim} p_{\text{data}}(\mathbf{x})$
- **Task:** Estimating the score $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$
- **Score Model:** A trainable vector-valued function $s_\theta(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}^D$
- **Objective:** How to compare two vector fields of scores?

?

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

(Fisher divergence)

- Integration by parts

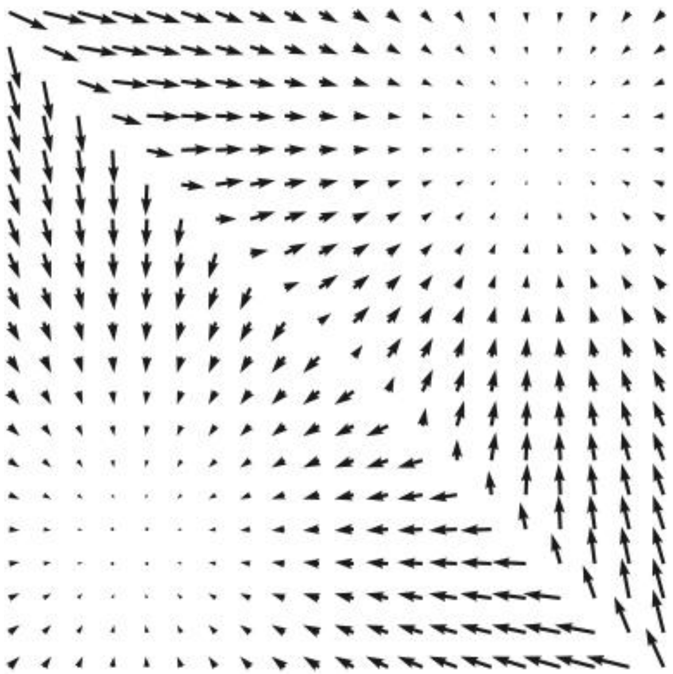
$$\mathbb{E}_{p_{\text{data}}}\left[\frac{1}{2}\|s_\theta(\mathbf{x})\|_2^2 + \text{trace}(\nabla_{\mathbf{x}} s_\theta(\mathbf{x}))\right]$$

$$\approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{2}\|s_\theta(\mathbf{x}_i)\|_2^2 + \text{trace}(\nabla_{\mathbf{x}} s_\theta(\mathbf{x}_i))\right]$$

Score Matching

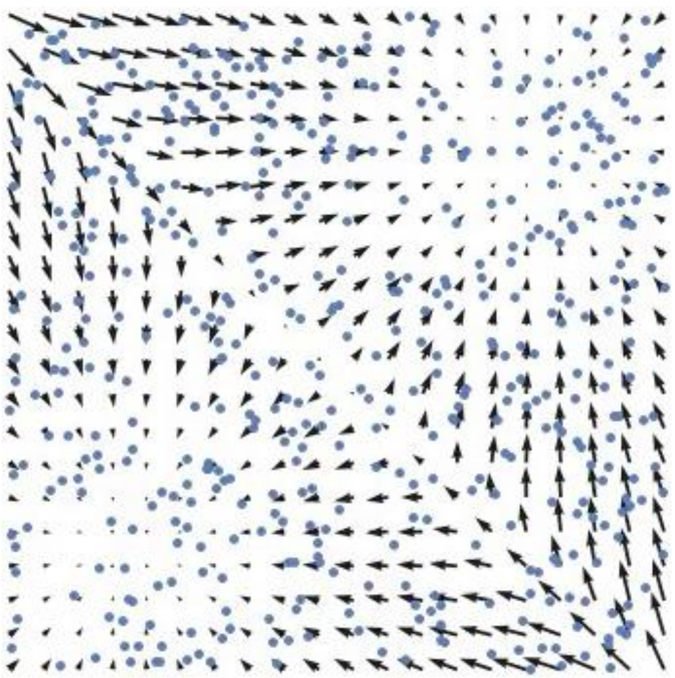
Hyvärinen (2005)

From Scores to Samples: Langevin Dynamics



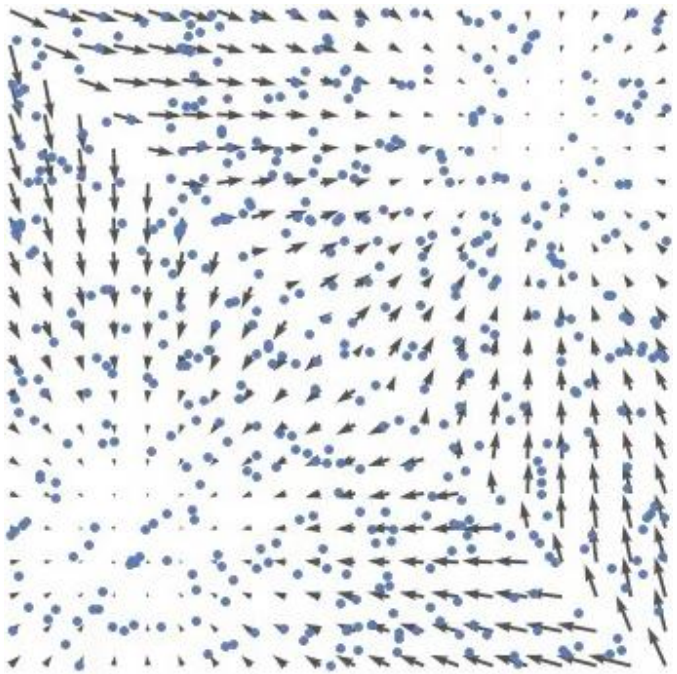
Scores

$$s_\theta(\mathbf{x})$$



Follow the scores

$$\tilde{\mathbf{x}}_{t+1} \leftarrow \tilde{\mathbf{x}}_t + \frac{\epsilon}{2} s_\theta(\tilde{\mathbf{x}}_t)$$



Follow noisy scores:
Langevin dynamics

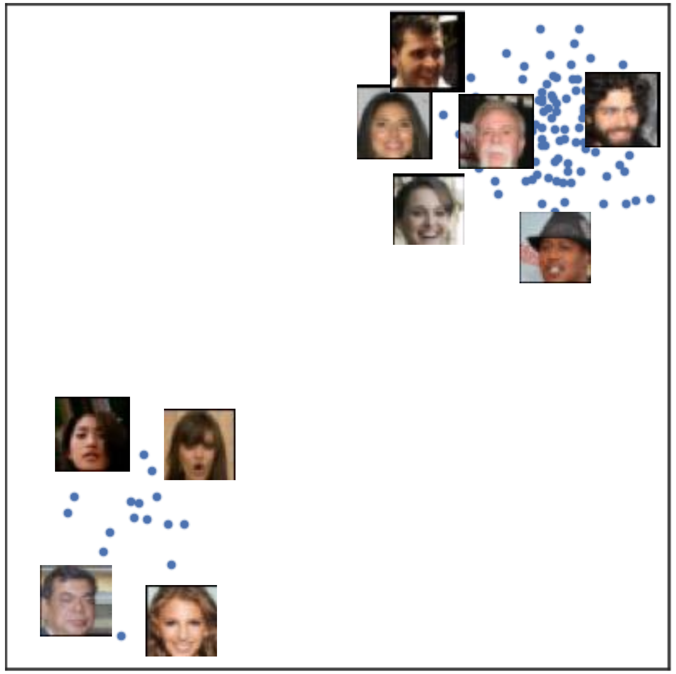
$$\mathbf{z}_t \sim \mathcal{N}(0, I)$$

$$\tilde{\mathbf{x}}_{t+1} \leftarrow \tilde{\mathbf{x}}_t + \frac{\epsilon}{2} s_\theta(\tilde{\mathbf{x}}_t) + \sqrt{\epsilon}\mathbf{z}_t$$

Lecture overview

- Energy-based models
- **Score-based Models**
  - Probability Distribution and Score functions
  - **Score-based Generative Models**
  - Sampling from a Score-based Model
  - Latent Score-based Generative Models
- Denoising Diffusion Models

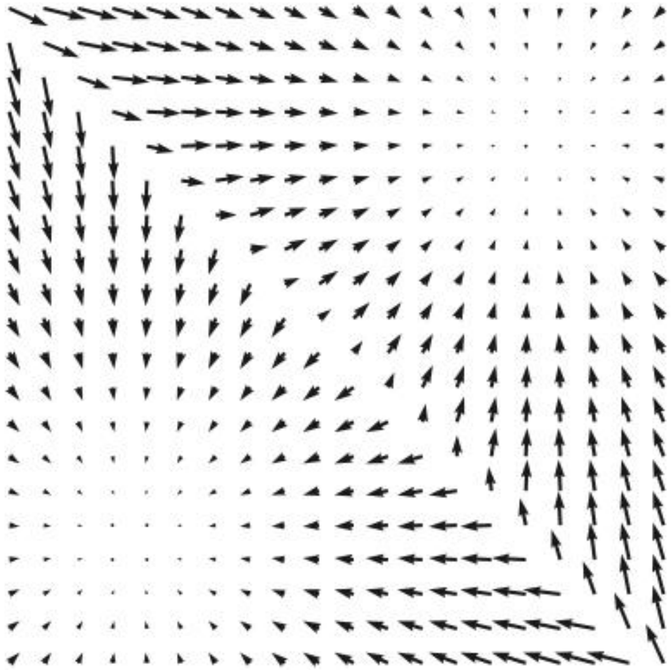
Score-Based Generative Modeling



Data samples

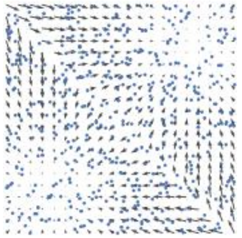
$$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \overset{\text{i.i.d.}}{\sim} p_{\text{data}}(\mathbf{x})$$

Score Matching



Scores

$$s_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$$

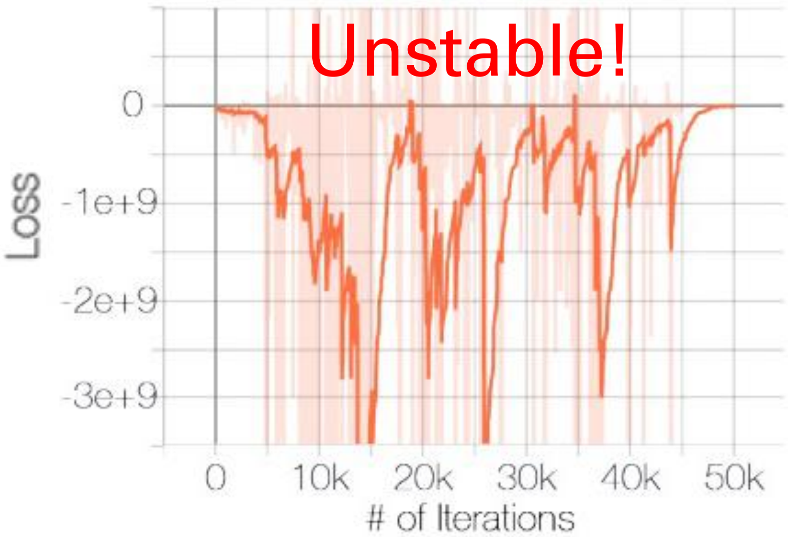


Langevin dynamics

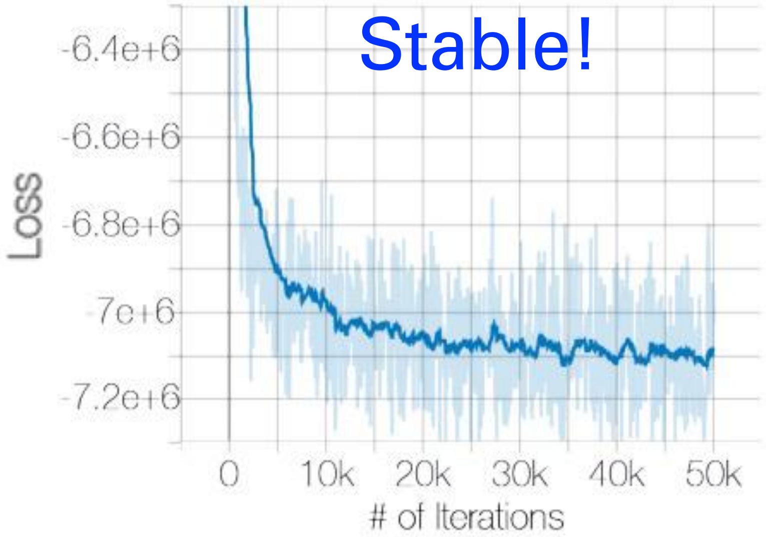
New samples

Adding Noise to Data for Well-Defined Scores

- Scores can be undefined when
  - The support of data distribution is on a low-dimensional manifold
  - The data distribution is discrete
- Solution: adding noise

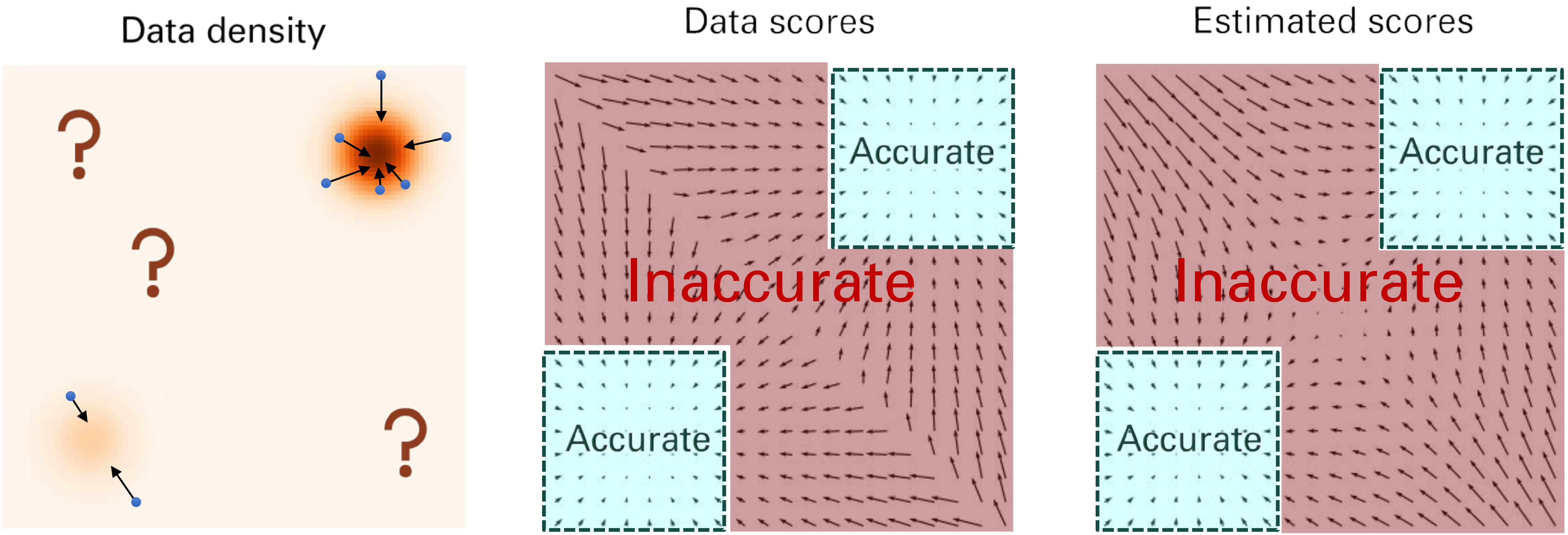


Data unperturbed



Data perturbed with $\mathcal{N}(0; 0.0001)$

Challenge in Low Data Density Regions

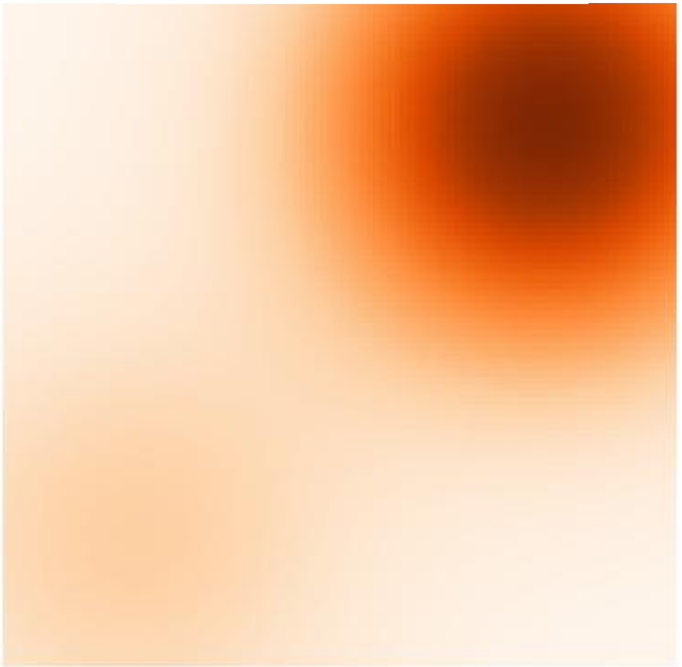


$$\frac{1}{2}\mathbb{E}_{p_{\text{data}}}[\|\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) - s_\theta(\mathbf{x})\|_2^2] \approx \frac{1}{2N}\sum_{i=1}^{N}\|\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}_i) - s_\theta(\mathbf{x}_i)\|_2^2$$

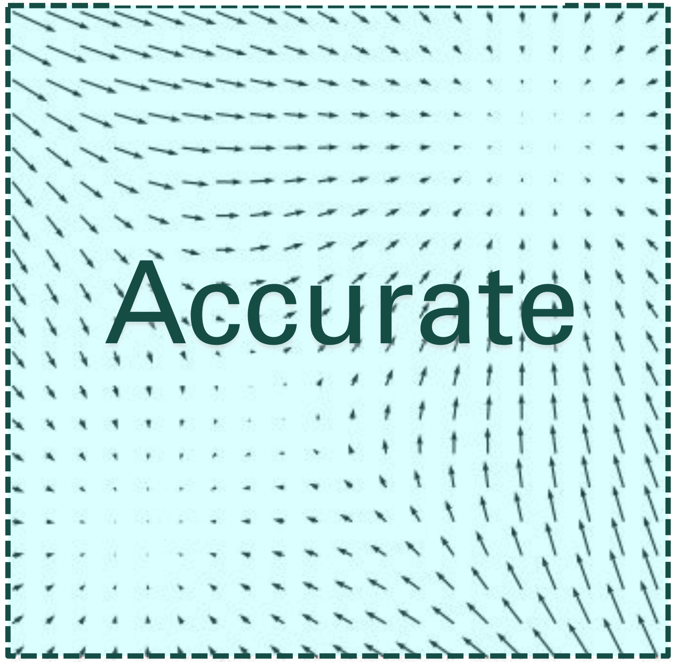
Adding Noise to Data for Better Score Estimation

- Random noise provides samples in low data density regions.

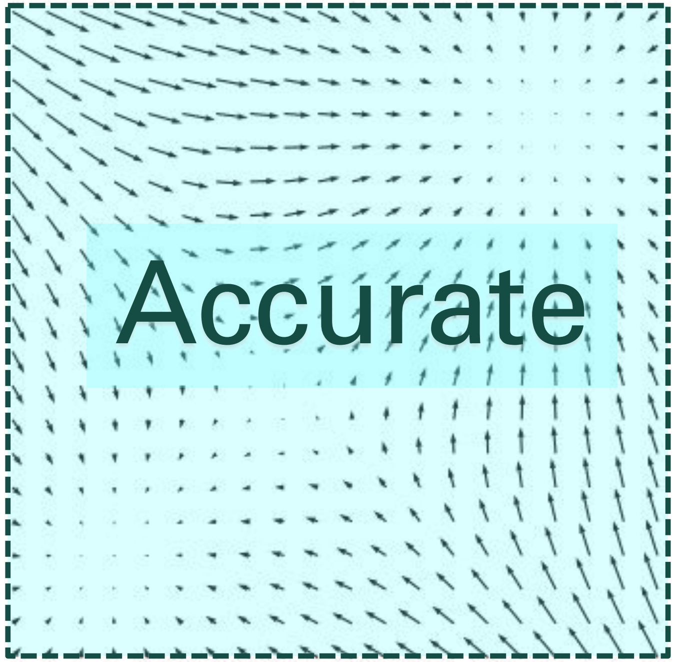
Perturbed density



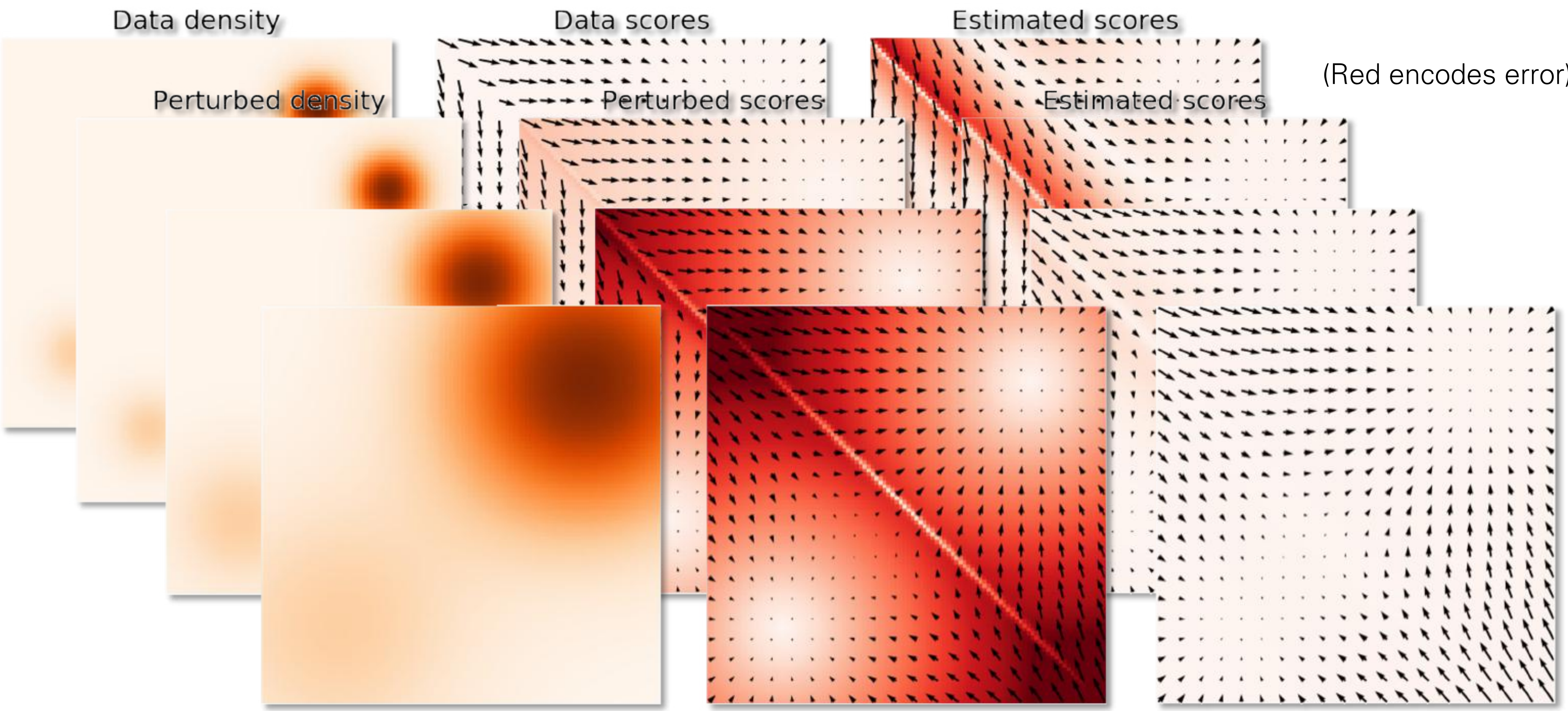
Perturbed scores



Estimated scores



Sample Quality vs. Estimation Accuracy

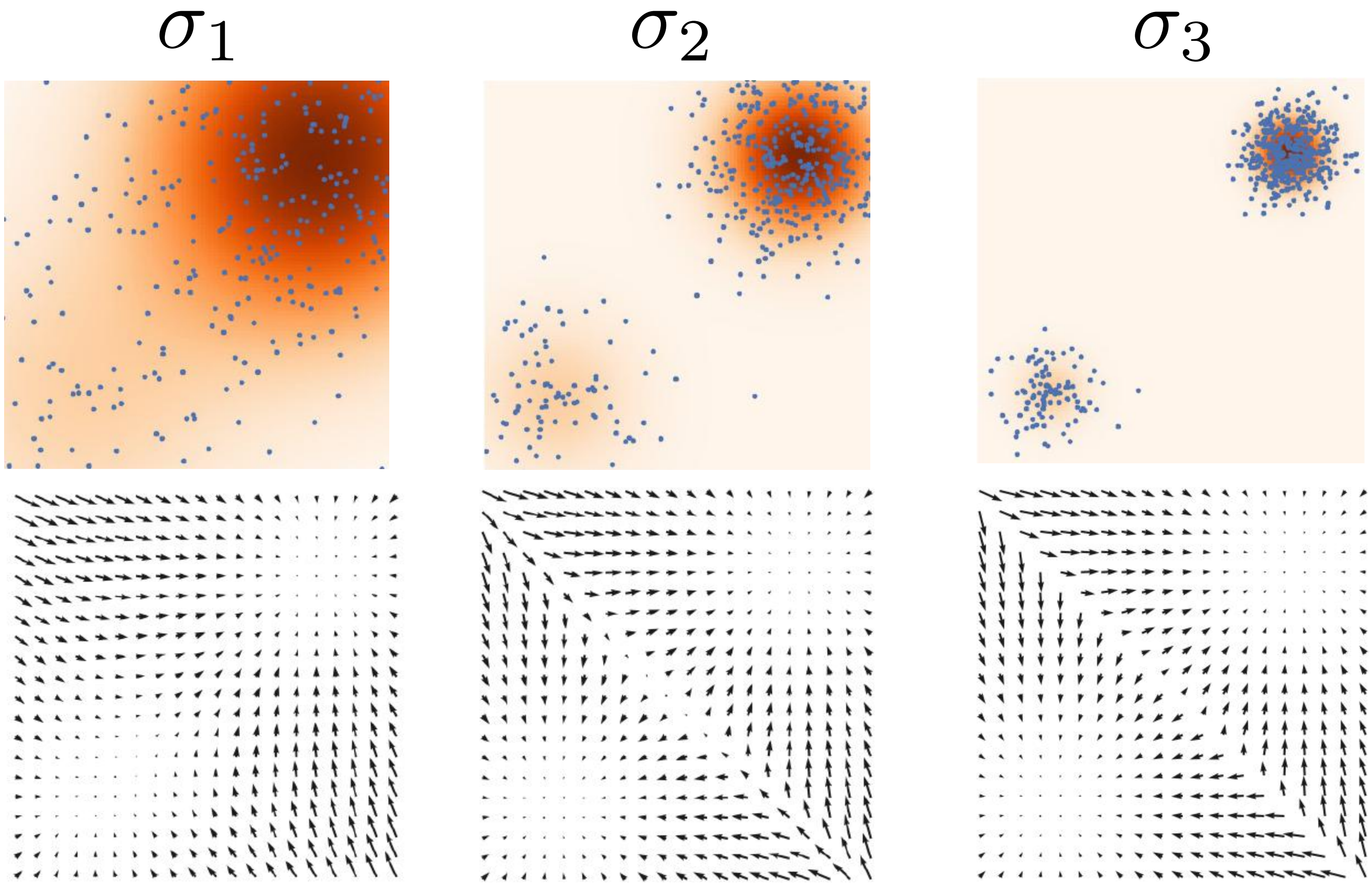


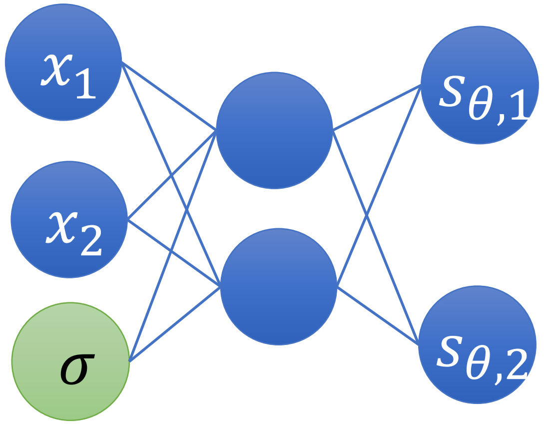
(Red encodes error)

Worse sample quality!

Better score estimation!

Joint Score Estimation via Noise Conditional Score Networks





Noise Conditional Score Network (NCSN)

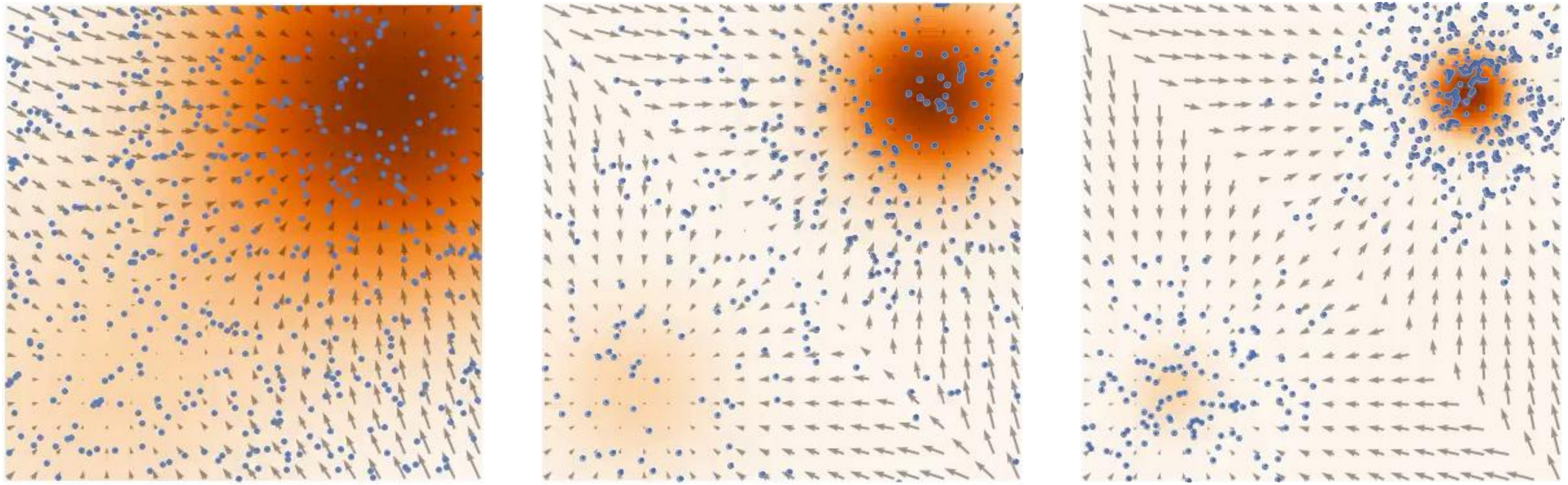
Lecture overview

- Energy-based models
- **Score-based Models**
  - Probability Distribution and Score functions
  - Score-based Generative Models
  - **Sampling from a Score-based Model**
  - Latent Score-based Generative Models
- Denoising Diffusion Models

Annealed Langevin Dynamics

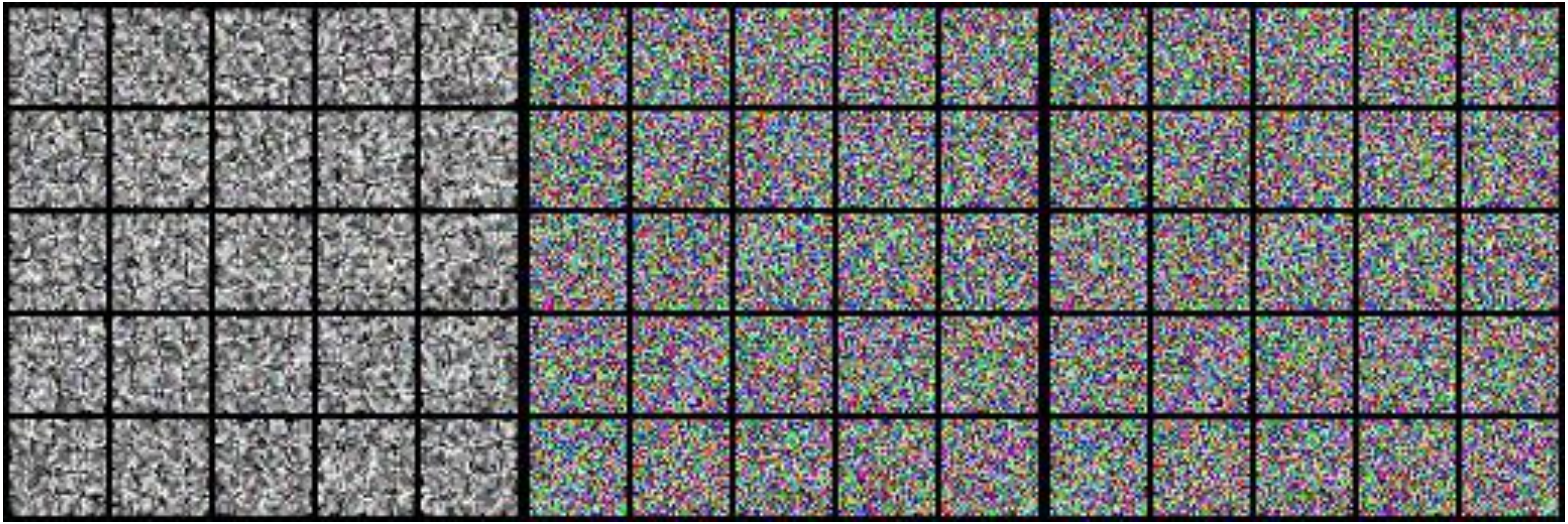
Joint Scores to Samples

- Sample using $\sigma_1, \sigma_2, \cdots, \sigma_L$ sequentially with Langevin dynamics.
- Anneal down the noise level.
- Samples used as initialization for the next level.



σ_1 σ_2 σ_3

Experiments: Sampling



Experiments: Sample Quality

CIFAR-10 Unconditional

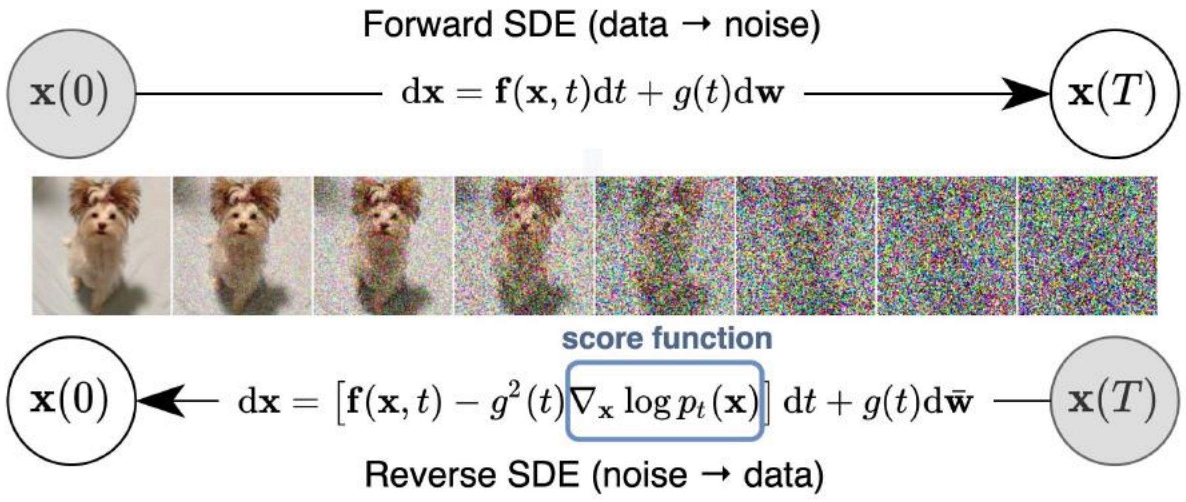
Model	Inception Score (higher is better)	FID score (lower is better)
PixelCNN	4.60	65.93
EBM	6.02	40.58
SNGAN	8.22 ± 0.05	21.7
ProgressiveGAN	8.80 ± 0.05	-
NCSN (ours)	8.87 ± 0.12	25.32

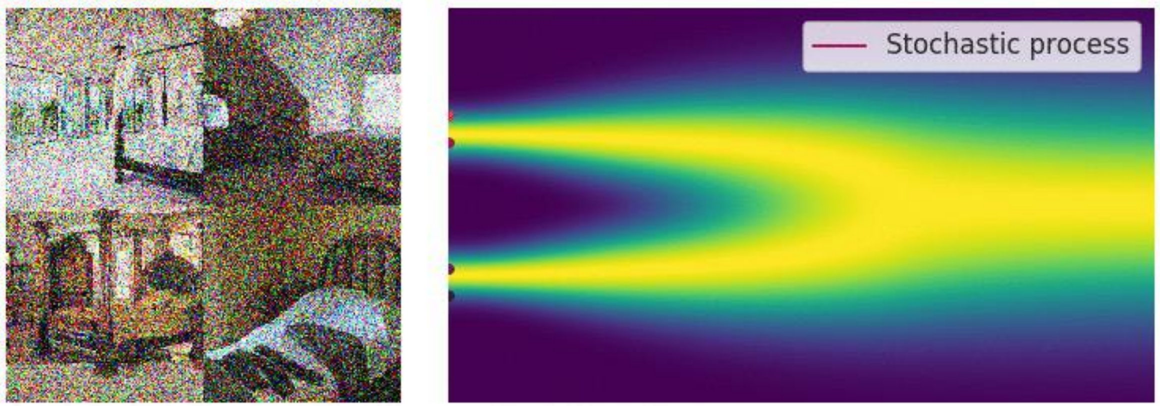
Experiments: Inpainting

Experiments: Inpainting

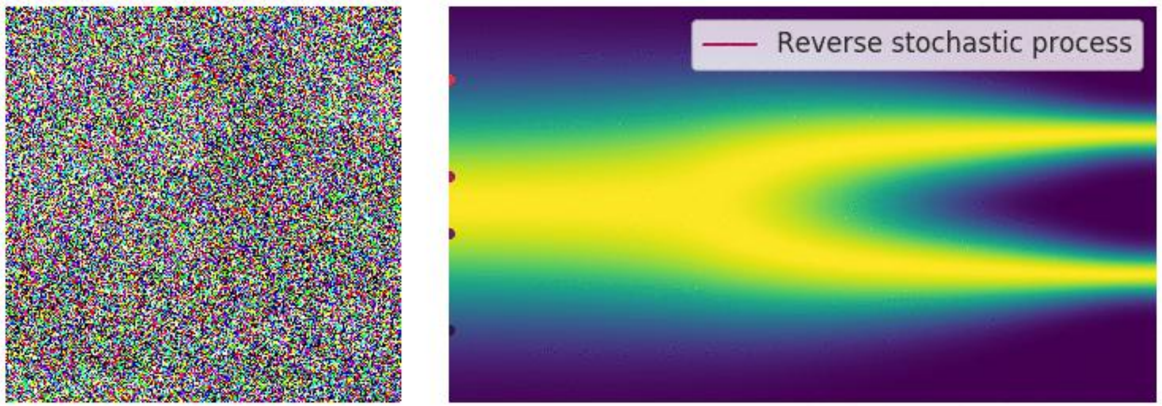
Reversing the SDE for sample generation

- One can generate samples by reversing the perturbation process with annealed Langevin dynamics.
- For infinite noise scales, we can analogously reverse the perturbation process for sample generation by using the reverse SDE.





Perturbing data to noise with a continuous-time stochastic process.



Generating data from noise by reversing the perturbation procedure.

1024 x 1024 samples on FFHQ dataset

256 x 256 samples on LSUN Bedroom

Sample Quality

Table 2: NLLs and FIDs (ODE) on CIFAR-10.

Model	NLL Test ↓	FID ↓
RealNVP (Dinh et al., 2016)	3.49	-
iResNet (Behrmann et al., 2019)	3.45	-
Glow (Kingma & Dhariwal, 2018)	3.35	-
MintNet (Song et al., 2019b)	3.32	-
Residual Flow (Chen et al., 2019)	3.28	46.37
FFJORD (Grathwohl et al., 2018)	3.40	-
Flow++ (Ho et al., 2019)	3.29	-
DDPM (L) (Ho et al., 2020)	$\leqslant 3.70^{*}$	13.51
DDPM (L_{simple}) (Ho et al., 2020)	$\leqslant 3.75^{*}$	3.17
DDPM	3.28	3.37
DDPM cont. (VP)	3.21	3.69
DDPM cont. (sub-VP)	3.05	3.56
DDPM++ cont. (VP)	3.16	3.93
DDPM++ cont. (sub-VP)	3.02	3.16
DDPM++ cont. (deep, VP)	3.13	3.08
DDPM++ cont. (deep, sub-VP)	**2.99**	**2.92**

Table 3: CIFAR-10 sample quality.

Model	FID↓	IS↑
Conditional		
BigGAN (Brock et al., 2018)	14.73	9.22
StyleGAN2-ADA (Karras et al., 2020a)	**2.42**	**10.14**
Unconditional		
StyleGAN2-ADA (Karras et al., 2020a)	2.92	9.83
NCSN (Song & Ermon, 2019)	25.32	8.87 ± .12
NCSNv2 (Song & Ermon, 2020)	10.87	8.40 ± .07
DDPM (Ho et al., 2020)	3.17	9.46 ± .11
DDPM++	2.78	9.64
DDPM++ cont. (VP)	2.55	9.58
DDPM++ cont. (sub-VP)	2.61	9.56
DDPM++ cont. (deep, VP)	2.41	9.68
DDPM++ cont. (deep, sub-VP)	2.41	9.57
NCSN++	2.45	9.73
NCSN++ cont. (VE)	2.38	9.83
NCSN++ cont. (deep, VE)	**2.20**	**9.89**

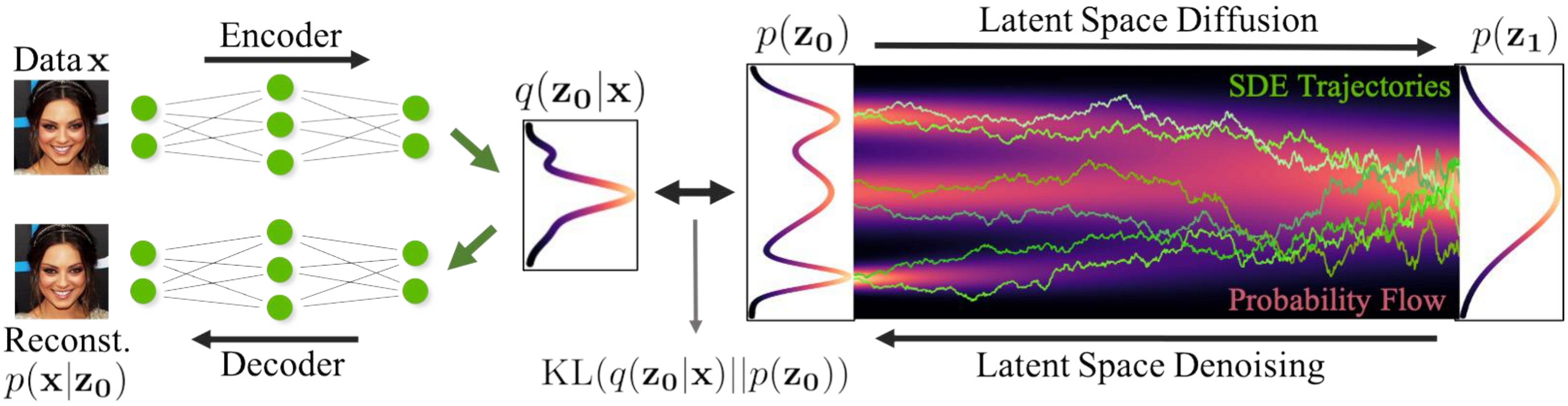
Lecture overview

- Energy-based models
- **Score-based Models**
  - Probability Distribution and Score functions
  - Score-based Generative Models
  - Sampling from a Score-based Model
  - **Latent Score-based Generative Models**
- Denoising Diffusion Models

Latent Score-based Generative Models

- Score-based generative models (SGMs) are applied directly in data space and <u>often require 1000s of network evaluations for sampling</u>.

- **Idea:** Can we train SGMs in a latent space?



Latent Score-based Generative Models

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\phi}, \boldsymbol{\theta}, \boldsymbol{\psi}) = \mathbb{E}_{q_\phi(\mathbf{z}_0|\mathbf{x})}[-\log p_\psi(\mathbf{x} \mid \mathbf{z}_0)] + \mathrm{KL}(q_\phi(\mathbf{z}_0 \mid \mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z}_0))$$

$$= \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_0|\mathbf{x})}\left[-\log p_{\boldsymbol{\psi}}(\mathbf{x}|\mathbf{z}_0)\right]}_{\text{reconstruction term}} + \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_0|\mathbf{x})}\left[\log q_{\boldsymbol{\phi}}(\mathbf{z}_0|\mathbf{x})\right]}_{\text{negative encoder entropy}} + \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_0|\mathbf{x})}\left[-\log p_{\boldsymbol{\theta}}(\mathbf{z}_0)\right]}_{\text{cross entropy}}$$

- A simple expression for the cross-entropy term in the variational loss
- A parameterization of the latent space score function, which mixes a Normal distribution with a learnable SGM.

Also employs a SDE-based variance reduction importance sampling schemes to stably train deep LSGMs.

Latent Score-based Generative Models

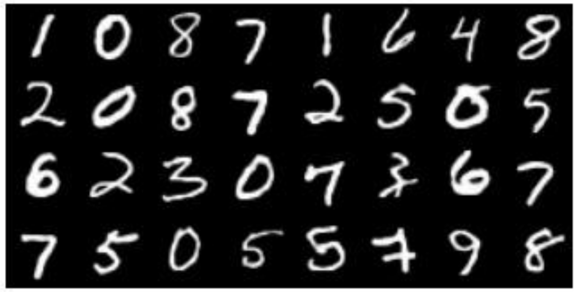
The evolution of the latent variables under the reverse-time generative process by feeding latent variables from different stages along the process to the decoder to map them back to image space

Latent SGM (LSGM) Samples

(a) CIFAR-10

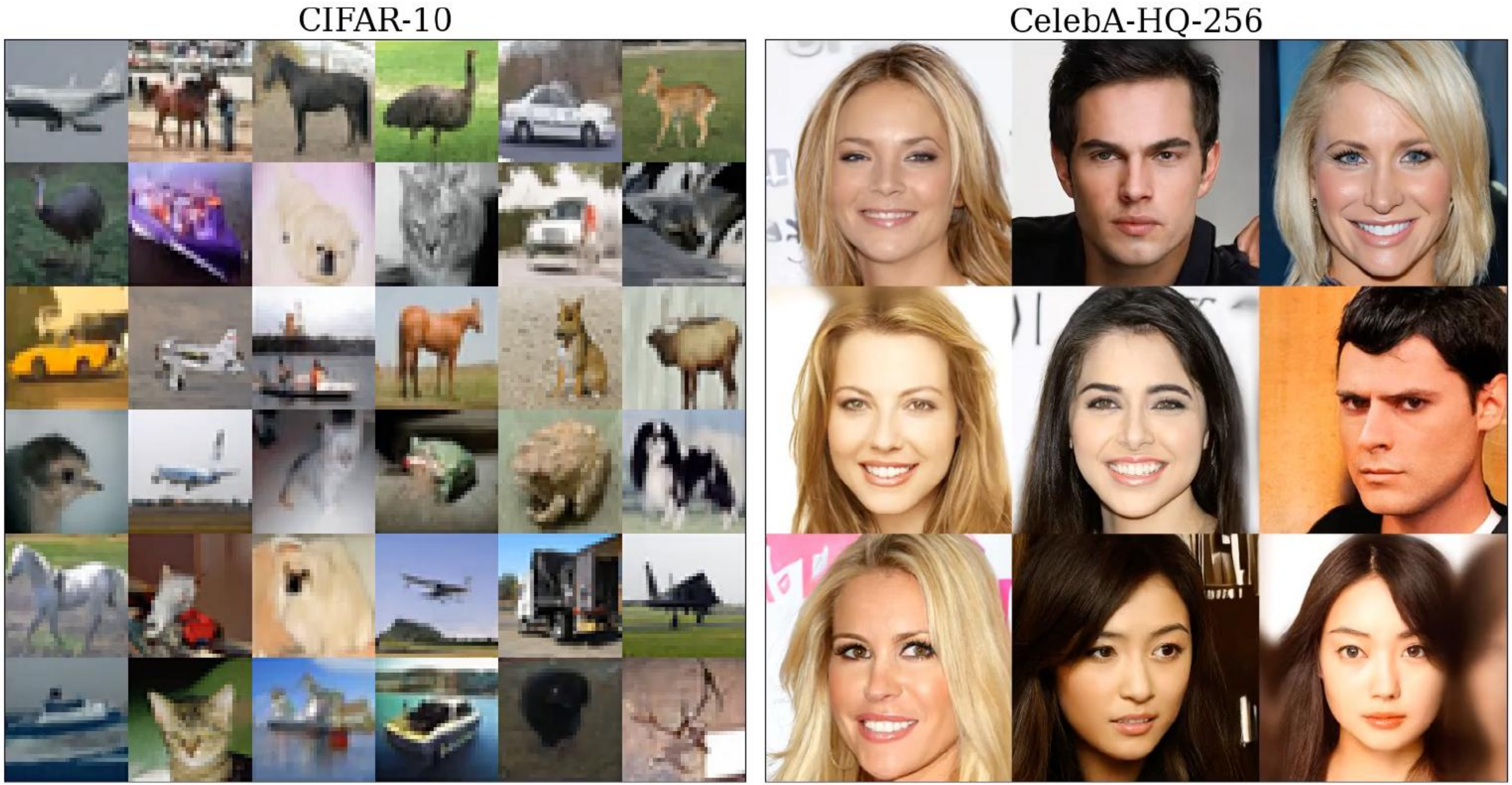
(b) CelebA-HQ-256

(c) OMNIGLOT



(d) MNIST

Traversing in the latent space of LSGM.



LSGM Sample Quality

Table 2: Generative performance on CIFAR-10.

	Method	NLL↓	FID↓
Ours	LSGM (FID)	≤3.43	**2.10**
	LSGM (NLL)	**≤2.87**	6.89
	LSGM (balanced)	≤2.95	2.17
	VAE Backbone	2.96	43.18
VAEs	VDVAE [21]	**2.87**	-
	NVAE [20]	2.91	23.49
	VAEBM [76]	-	12.19
	NCP-VAE [56]	-	24.08
	BIVA [48]	3.08	-
	DC-VAE [77]	-	17.90
Score	NCSN [3]	-	25.32
	Rec. Likelihood [40]	3.18	9.36
	DSM-ALS [39]	3.65	-
	DDPM [1]	3.75	3.17
	Improved DDPM [26]	2.94	11.47
	SDE (DDPM++) [2]	2.99	2.92
	SDE (NCSN++) [2]	-	2.20
Flows	VFlow [19]	2.98	-
	ANF [18]	3.05	-
Aut. Reg.	DistAug aug [78]	2.53	42.90
	Sp. Transformers [79]	2.80	-
	δ-VAE [80]	2.83	-
	PixelSNAIL [81]	2.85	-
	PixelCNN++ [82]	2.92	-
GANs	AutoGAN [83]	-	12.42
	StyleGAN2-ADA [84]	-	2.92

Table 3: Generative results on CelebA-HQ-256.

	Method	NLL↓	FID↓
Ours	LSGM	**≤0.70**	**7.22**
	VAE Backbone	**0.70**	30.87
VAEs	NVAE [20]	**0.70**	29.76
	VAEBM [76]	-	20.38
	NCP-VAE [56]	-	24.79
	DC-VAE [77]	-	15.80
Score	SDE [2]	-	**7.23**
Flows	GLOW [85]	1.03	68.93
Aut. Reg.	SPN [86]	0.61	-
GANs	Adv. LAE [87]	-	19.21
	VQ-GAN [64]	-	10.70
	PGGAN [88]	-	8.03

- LSGM obtains the SOTA FID score of 2.10 on CIFAR-10, outperforming previous GANs and SGMs.
- On CelebA-HQ-256, it is on a par with previous SGMs while being 50x to 600x faster in sampling.

Conclusion

- Score-based generative modeling
  - **No need to be normalized / invertible**
    - Flexible architecture choices
  - **No minimax optimization**
    - stable training
    - a natural measurement of training progress / model comparison

- **Adding noise** and **annealing the noise levels** are critical

- Better or comparable sample quality to GANs.

Related Work

- Generative Stochastic Networks (Bengio et al. (2013), Alain et al. (2016))
  - Sampling starts **close to data points.**
  - Need **MCMC during training** with walkback.

- Nonequilibrium Thermodynamics (Sohl-Dickstein et al. (2015)), Infusion Training (Bordes et al. (2017)), Variational Walkback (Goyal et al. (2017))
  - **Likelihood**-based training.
  - Need **MCMC during training.**

Experiments: Nearest Neighbors

Future Directions

- How to apply score-based generative modeling to discrete data?
- Theoretical guidance on how to choose noise levels?
- Better architecture for higher resolution image generation?
- Improved score estimation?

Lecture overview

- Energy-based models
- Score-based Models
- **Denoising Diffusion Models**
  - Diffusion Probabilistic Models
  - Denoising Diffusion Probabilistic Models
  - Denoising Diffusion Implicit Model
  - Applications

Lecture overview

- Energy-based models
- Score-based Models
- **Denoising Diffusion Models**
  - **Diffusion Probabilistic Models**
  - Denoising Diffusion probabilistic Models
  - Denoising Diffusion Implicit Model
  - Applications

Observation 1: Diffusion Destroys Structure

- Dye density represents probability density
- Goal: Learn structure probability density
- Observation: Diffusion destroys structure

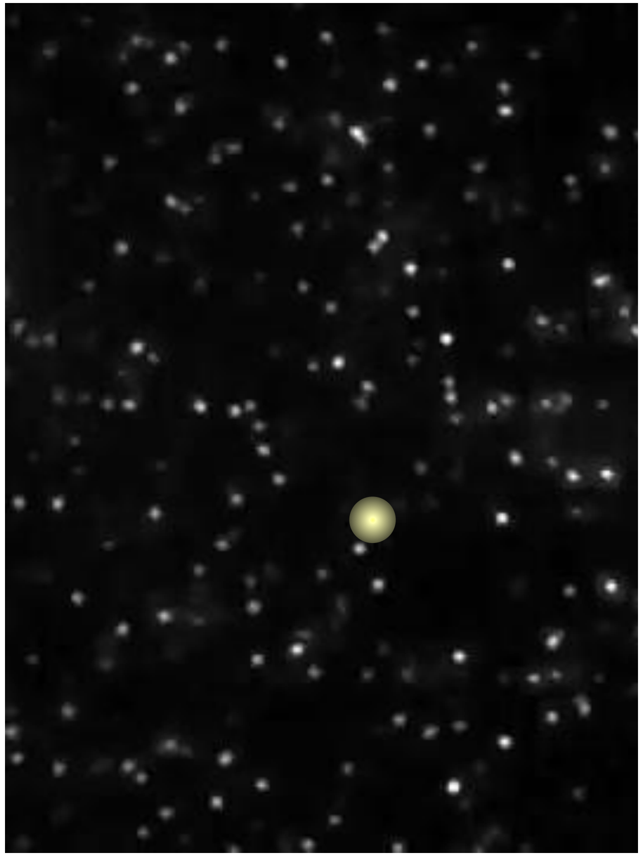
Data distribution → Uniform distribution

Idea: Recover Structure by Reversing Time

- What if we could reverse time?
- Recover data distribution by starting from uniform distribution and running dynamics backwards

Data distribution ← Uniform distribution

Observation 2: Microscopic Diffusion is Time Reversible



Nanoparticles in water

- Microscopic view
- Brownian motion
- Position updates are small Gaussians
  - Both forwards and backwards in time

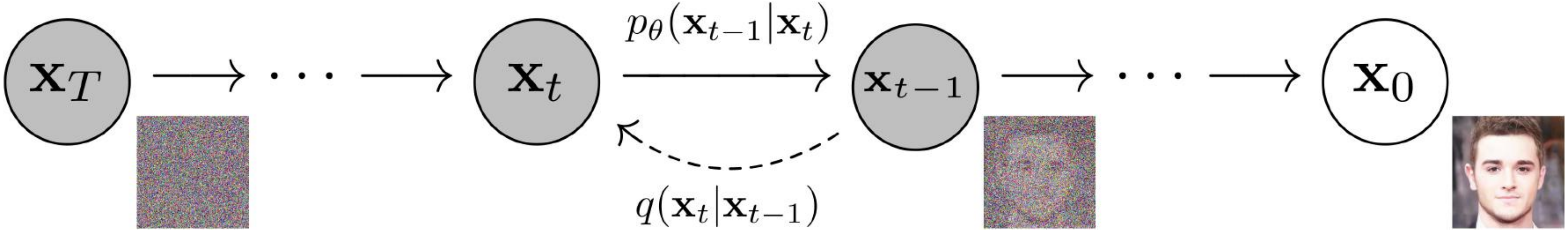
Video credit: Rutger Saly

Overview of Diffusion Probabilistic Models

- Destroy all structure in data distribution using diffusion process
- Learn reversal of diffusion process
  - Estimate function for mean and covariance of each step in the reverse diffusion process (binomial rate for binary data)

Diffusion Probabilistic Models

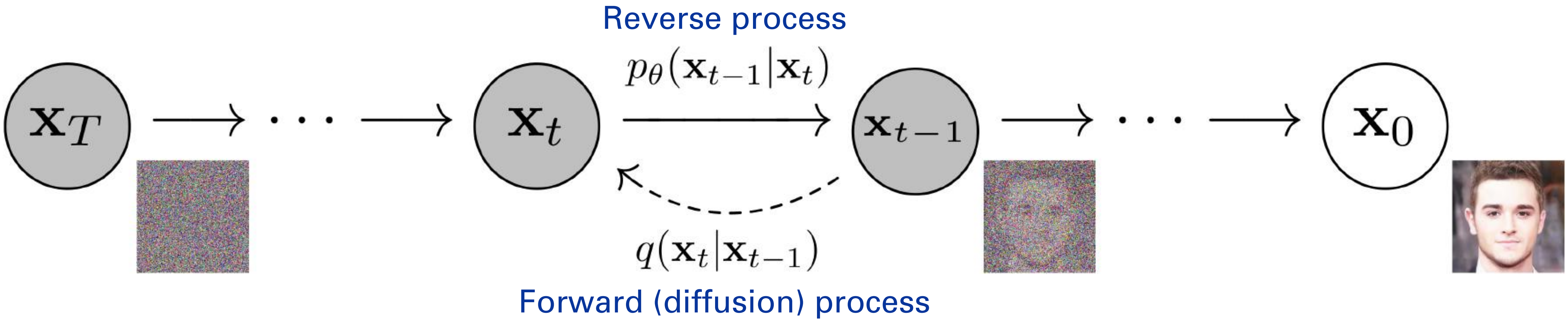
- Diffusion model aims to learn the reverse of noise generation procedure
  - **Forward step**: (Iteratively) Add noise to the original sample
    - →The sample $\mathbf{x}_0$ converges to the complete noise $\mathbf{x}_T$ (e.g., $\mathcal{N}(0, I)$)



Forward (diffusion) process

Diffusion Probabilistic Models

- Diffusion model aims to learn the reverse of noise generation procedure
  - **Forward step**: (Iteratively) Add noise to the original sample
    - →The sample $\mathbf{x}_0$ converges to the complete noise $\mathbf{x}_T$ (e.g., $\mathcal{N}(0, I)$)
  - **Reverse step**: Recover the original sample from the noise
    - →Note that it is the "generation" procedure



Diffusion Probabilistic Models

- Diffusion model aims to learn the reverse of noise generation procedure
  - **Forward step**: (Iteratively) Add noise to the original sample
    - → Technically, it is a product of conditional noise distributions $q(\mathbf{x}_t|\mathbf{x}_{t-1})$
      - Usually, the parameters β_t are fixed (one can jointly learn, but not beneficial)
      - Noise annealing (i.e., reducing noise scale $\beta_t < \beta_{t-1}$) is crucial to the performance

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

Diffusion Probabilistic Models

- Diffusion model aims to learn the reverse of noise generation procedure
  - **Forward step**: (Iteratively) Add noise to the original sample

    → Technically, it is a product of conditional noise distributions $q(\mathbf{x}_t|\mathbf{x}_{t-1})$

    - Usually, the parameters β_t are fixed (one can jointly learn, but not beneficial)
    - Noise annealing (i.e., reducing noise scale $\beta_t < \beta_{t-1}$) is crucial to the performance

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

  - **Reverse step**: Recover the original sample from the noise

    → It is also a product of conditional (de)noise distributions $p_\theta(\mathbf{x}_{t=1}|\mathbf{x}_t)$

    → Use the **learned** parameters: denoiser $\boldsymbol{\mu}_\theta$ (main part) and randomness $\boldsymbol{\Sigma}_\theta$

$$p_\theta\left(\mathbf{x}_{0:T}\right) := p\left(\mathbf{x}_T\right) \prod_{t=1}^{T} p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right), \quad p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) := \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)\right)$$

Diffusion Probabilistic Models

- Diffusion model aims to learn the reverse of noise generation procedure
  - **Forward step**: (Iteratively) Add noise to the original sample
  - **Reverse step**: Recover the original sample from the noise

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

  - **Training:** Minimize variational lower bound of the model

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

Diffusion Probabilistic Models

- Diffusion model aims to learn the reverse of noise generation procedure
  - **Forward step**: (Iteratively) Add noise to the original sample
  - **Reverse step**: Recover the original sample from the noise

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

  - **Training:** Minimize variational lower bound of the model

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

    - It can be decomposed to the **step-wise** losses (for each step *t*)

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

Diffusion Probabilistic Models

- Diffusion model aims to learn the reverse of noise generation procedure
  - **Training:** Minimize variational lower bound of the model
    - It can be decomposed to the **step-wise** losses (for each step *t*)

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

  - Here, the true reverse step $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ can be computed as a **closed form** of β_t
    - Note that we only define the true forward step

$$q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right)=\mathcal{N}\left(\mathbf{x}_{t-1} ; \tilde{\mu}_t\left(\mathbf{x}_t, \mathbf{x}_0\right), \tilde{\beta}_t^3 \mathbf{I}\right)$$

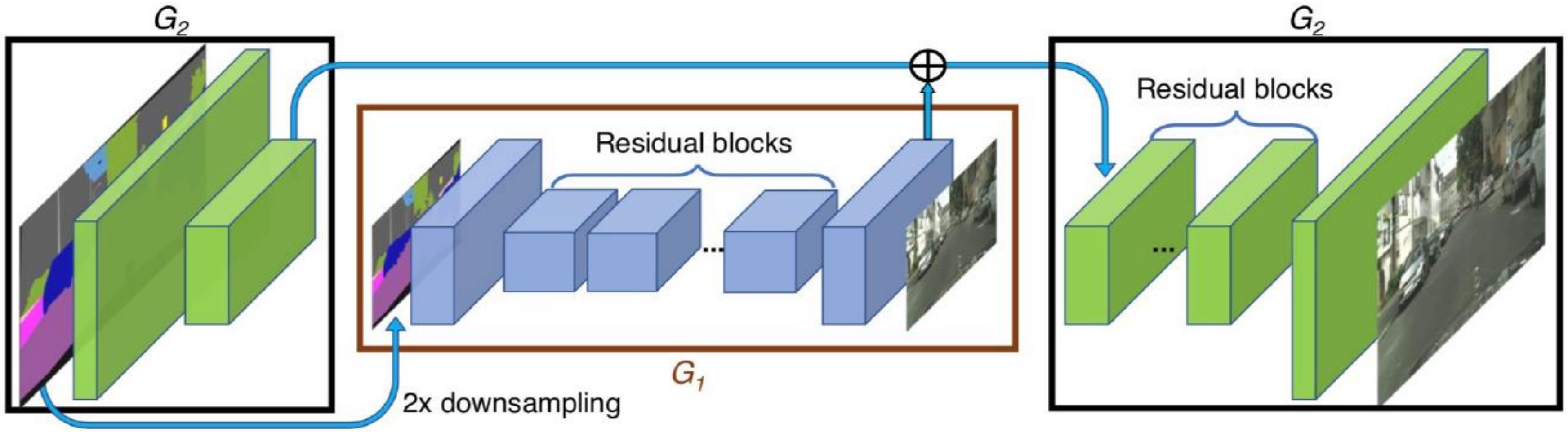
$$\text{where} \quad \tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right):=\tilde{\beta}_t^1 \mathbf{x}_0+\tilde{\beta}_t^2 \mathbf{x}_t$$

  - Since all distributions above are Gaussian, the KL divergences are tractable

Diffusion Probabilistic Models

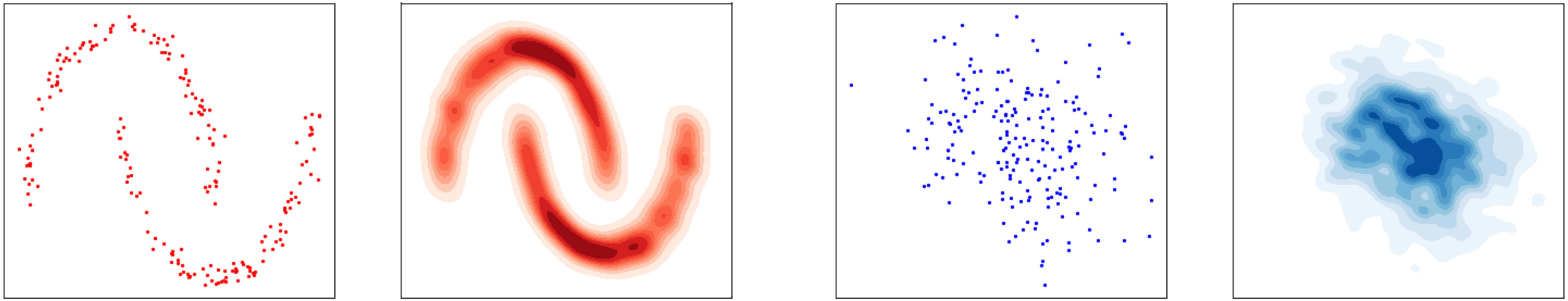
- Diffusion model aims to learn the reverse of noise generation procedure
  - **Network:** Use the image-to-image translation (e.g., U-Net) architectures
    - Recall that input is $\mathbf{x}_t$ and output is $\mathbf{x}_{t-1}$, both are images
    - It is expensive since both input and output are high-dimensional

    - Note that the denoiser $\mu_\theta(\mathbf{x}_t, t)$ shares weights, but conditioned by step t



Diffusion Probabilistic Models

- Diffusion model aims to learn the reverse of noise generation procedure
  - **Sampling:** Draw a random noise $\boldsymbol{x}_T$ then apply the reverse step $p(\mathbf{x}_{t=1}|\mathbf{x}_t)$
    - It often requires the hundreds of reverse steps (very slow)
  - Early and late steps change the high- and low-level attributes, respectively



Diffusion Probabilistic Models – CIFAR10

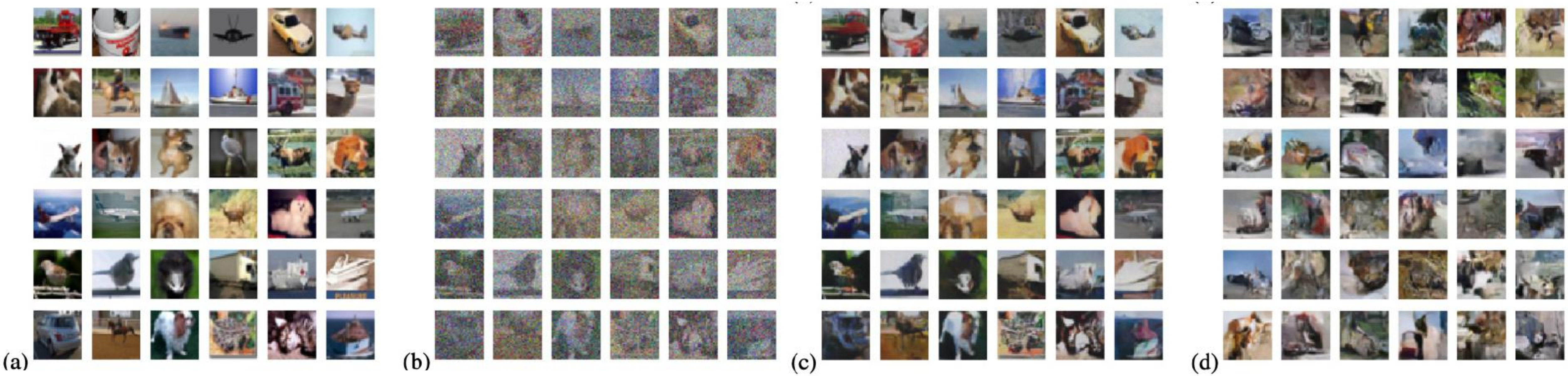


Figure 3. The proposed framework trained on the CIFAR-10 (Krizhevsky & Hinton, 2009) dataset. *(a)* Example holdout data (similar to training data). *(b)* Holdout data corrupted with Gaussian noise of variance 1 (SNR = 1). *(c)* Denoised images, generated by sampling from the posterior distribution over denoised images conditioned on the images in (b). *(d)* Samples generated by the diffusion model.

Lecture overview

- Energy-based models
- Score-based Models
- **Denoising Diffusion Models**
  - Diffusion Probabilistic Models
  - **Denoising Diffusion Probabilistic Models**
  - Denoising Diffusion Implicit Model
  - Applications

Denoising Diffusion Probabilistic Model

- DDPM reparametrizes the reverse distributions of diffusion models
  - Key idea: The original reverse step fully creates the denoiser $\mu_\theta(\mathbf{x}_t, t)$ from $\mathbf{x}_t$
    - However, $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ share most information, and thus it is redundant
      - Instead, create the **residual** $\epsilon_\theta(\mathbf{x}_t, t)$ and add to the original $\mathbf{x}_t$

Training resembles denoising score matching

Sampling resembles Langevin Dynamics

Initiated the diffusion model boom!

Denoising Diffusion Probabilistic Model

- DDPM reparametrizes the reverse distributions of diffusion models
  - Key idea: The original reverse step fully creates the denoiser $\mu_\theta(\mathbf{x}_t, t)$ from $\mathbf{x}_t$
    - However, $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ share most information, and thus it is redundant
      - Instead, create the **residual** $\epsilon_\theta(\mathbf{x}_t, t)$ and add to the original $\mathbf{x}_t$
  - Formally, DDPM reparametrizes the learned reverse distribution as

$$\boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right)=\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t-\frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta\left(\mathbf{x}_t, t\right)\right)$$

and the step-wise objective L_{t-1} can be reformulated as

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

Denoising Diffusion Probabilistic Model

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 $\mathbf{x}_0$

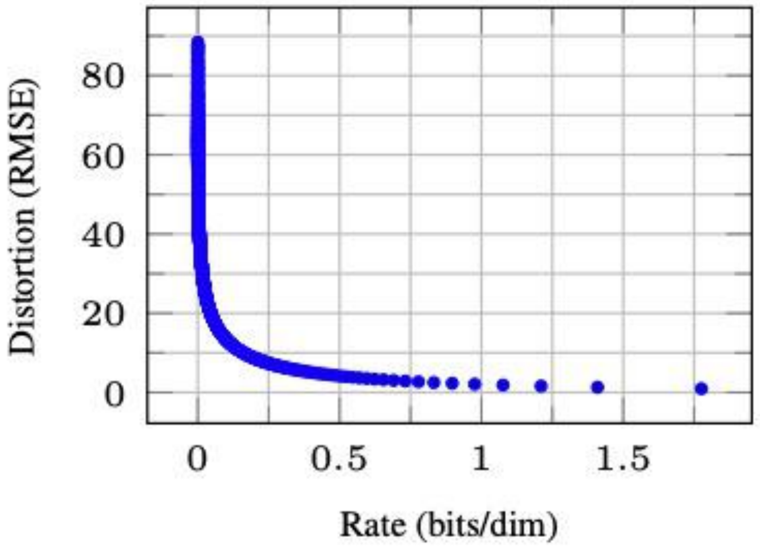
Denoising Diffusion Probabilistic Model

Unconditional CIFAR10 samples. Inception Score=9.46, FID=3.17.

Denoising Diffusion Probabilistic Model

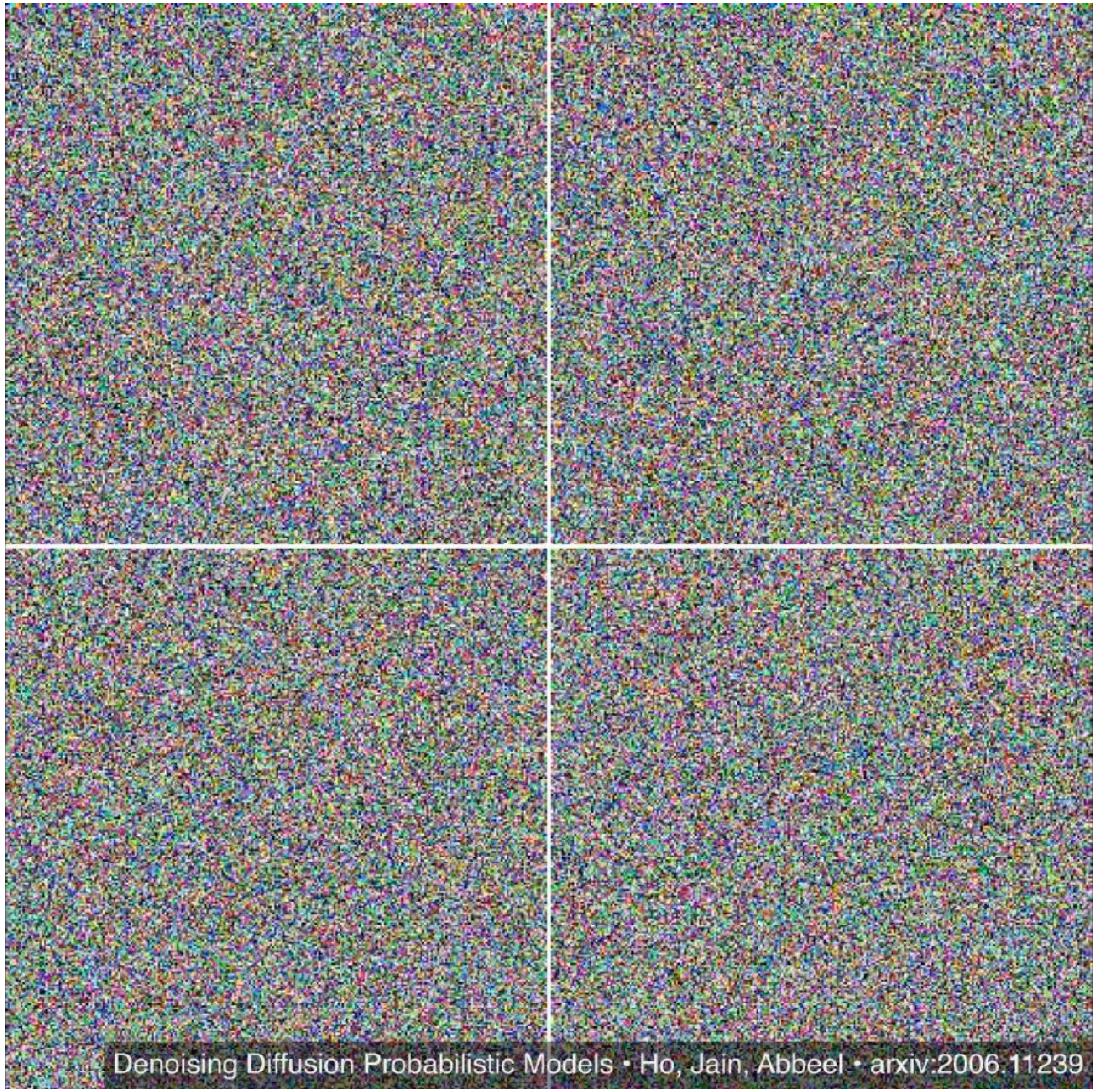
Table 1: CIFAR10 results. NLL measured in bits/dim.

Model	IS	FID	NLL Test (Train)
Conditional			
EBM [11]	8.30	37.9	
JEM [15]	8.76	38.4	
BigGAN [3]	9.22	14.73	
StyleGAN2 + ADA [28]	**10.06**	**2.67**	
Unconditional			
Diffusion (original) [50]			≤ 5.40
Gated PixelCNN [56]	4.60	65.93	3.03 (2.90)
Sparse Transformer [7]			**2.80**
PixelIQN [40]	5.29	49.46	
EBM [11]	6.78	38.2	
NCSNv2 [53]		31.75	
NCSN [52]	8.87 ± 0.12	25.32	
SNGAN [36]	8.22 ± 0.05	21.7	
SNGAN-DDLS [4]	9.09 ± 0.10	15.42	
StyleGAN2 + ADA [28]	$\mathbf{9.74} \pm 0.05$	3.26	
Ours (L, fixed isotropic $\mathbf{\Sigma}$)	7.67 ± 0.13	13.51	≤ 3.70 (3.69)
Ours (L_{simple})	9.46 ± 0.11	**3.17**	≤ 3.75 (3.72)



Denoising Diffusion Probabilistic Model

Denoising Diffusion Probabilistic Model



Lecture overview

- Energy-based models
- Score-Based Models
- **Denoising Diffusion Models**
  - Diffusion Probabilistic Models
  - Denoising Diffusion Probabilistic Models
  - **Denoising Diffusion Implicit Model**
  - Applications

Denoising Diffusion Implicit Model

- DDIM roughly sketches the final sample, then refine it with the reverse process

- **Motivation**:
  - Diffusion model is slow due to the iterative procedure
  - GAN/VAE creates the sample by one-shot forward operation
  - Can we combine the advantages for fast sampling of diffusion models?

- **Technical spoiler:**
  Instead of naively applying diffusion model upon GAN/VAE,
  DDIM proposes a principled approach of rough sketch + refinement

Denoising Diffusion Implicit Model

- DDIM roughly sketches the final sample, then refine it with the reverse process
  - Key idea:
    - Given $\mathbf{x}_t$, generate the rough sketch $\mathbf{x}_0$ and refine $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$
    - Unlike original diffusion model, it is not a Markovian structure

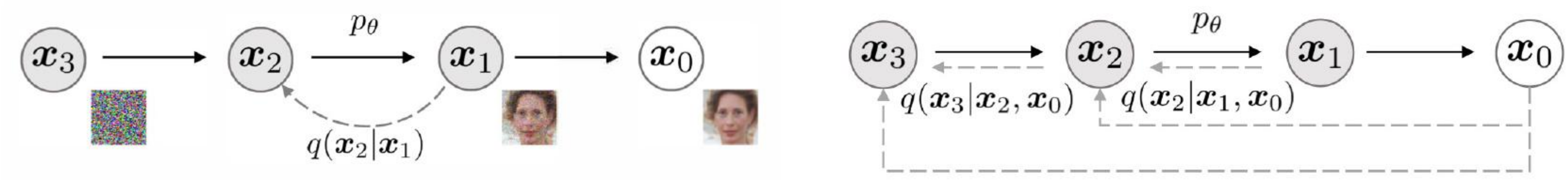
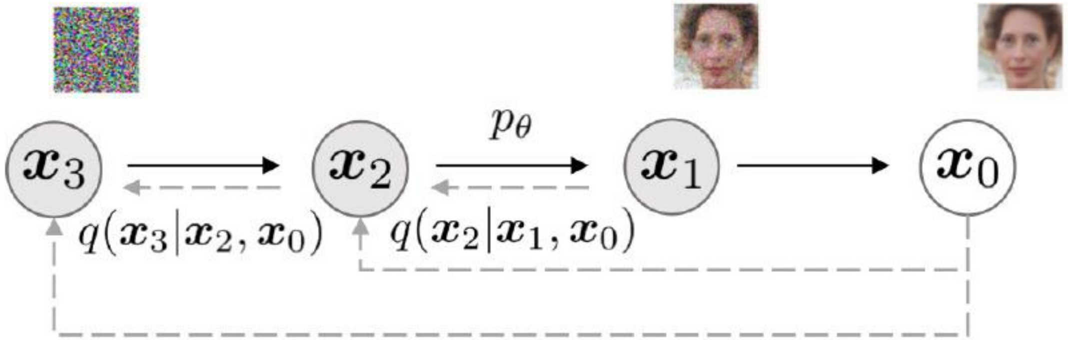


Figure 1: Graphical models for diffusion (left) and non-Markovian (right) inference models.

Denoising Diffusion Implicit Model

- DDIM roughly sketches the final sample, then refine it with the reverse process
  - **Key idea:** Given $\mathbf{x}_t$, generate the rough sketch $\mathbf{x}_0$ and refine $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$



  - **Formulation:** Define the forward distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ as

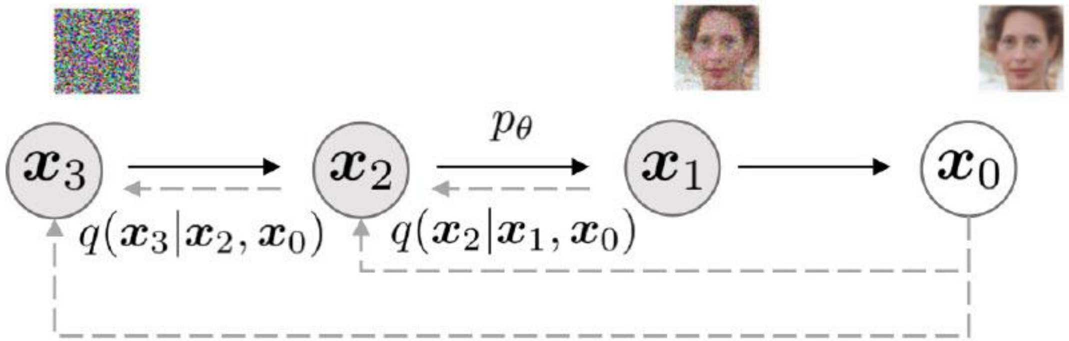
$$q_\sigma\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0\right) = \mathcal{N}\left(\sqrt{\alpha_{t-1}}\boldsymbol{x}_0 + \sqrt{1 - \alpha_{t-1} - \sigma_t^2} \cdot \frac{\boldsymbol{x}_t - \sqrt{\alpha_t}\boldsymbol{x}_0}{\sqrt{1-\alpha_t}}, \sigma_t^2 \boldsymbol{I}\right)$$

    then, the forward process is derived from Bayes' rule

$$q_\sigma\left(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{x}_0\right) = \frac{q_\sigma\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0\right) q_\sigma\left(\boldsymbol{x}_t \mid \boldsymbol{x}_0\right)}{q_\sigma\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_0\right)}$$

Denoising Diffusion Implicit Model

- DDIM roughly sketches the final sample, then refine it with the reverse process
  - **Key idea:** Given $\mathbf{x}_t$, generate the rough sketch $\mathbf{x}_0$ and refine $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$



  - **Formulation:** Forward process is $q_\sigma\left(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{x}_0\right)=\frac{q_\sigma\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0\right) q_\sigma\left(\boldsymbol{x}_t \mid \boldsymbol{x}_0\right)}{q_\sigma\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_0\right)}$

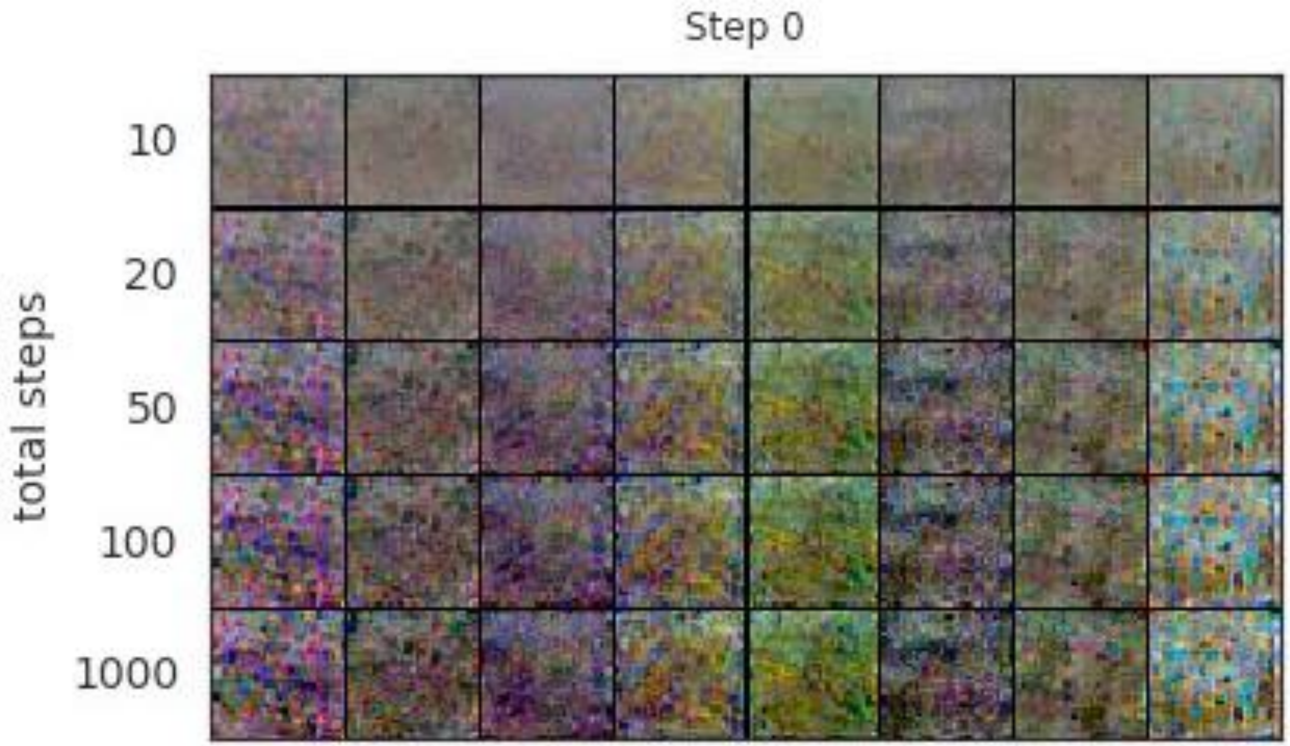
    and reverse process is $\boldsymbol{x}_{t-1}=\sqrt{\alpha_{t-1}} \underbrace{\left(\frac{\boldsymbol{x}_t-\sqrt{1-\alpha_t} \epsilon_\theta^{(t)}\left(\boldsymbol{x}_t\right)}{\sqrt{\alpha_t}}\right)}_{\text{"predicted } \boldsymbol{x}_0 \text{"}}+\underbrace{\sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \epsilon_\theta^{(t)}\left(\boldsymbol{x}_t\right)}_{\text{"direction pointing to } \boldsymbol{x}_t \text{"}}+\underbrace{\sigma_t \epsilon_t}_{\text{random noise}}$

  - **Training:** The variational lower bound of DDIM is identical to the one of DDPM
    - It is surprising since the forward/reverse formulation is totally different

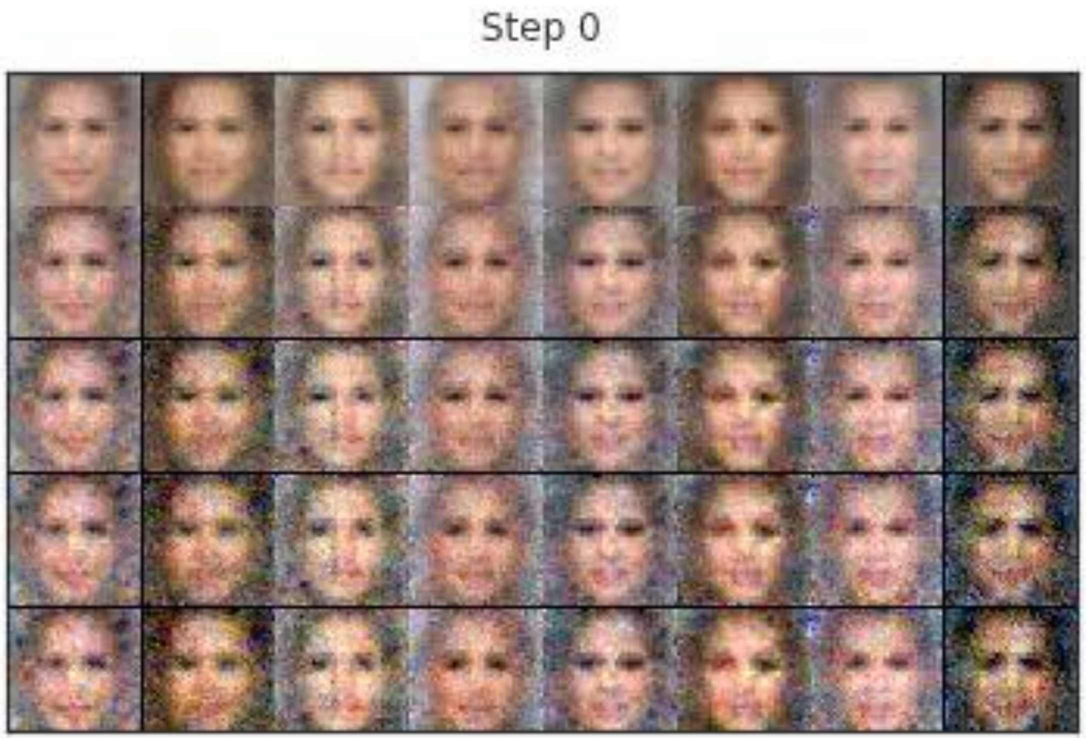
Denoising Diffusion Implicit Model

- DDIM significantly reduces the sampling steps of diffusion model
  - Creates the outline of the sample after only 10 steps (DDPM needs hundreds)

Denoising Diffusion Implicit Model

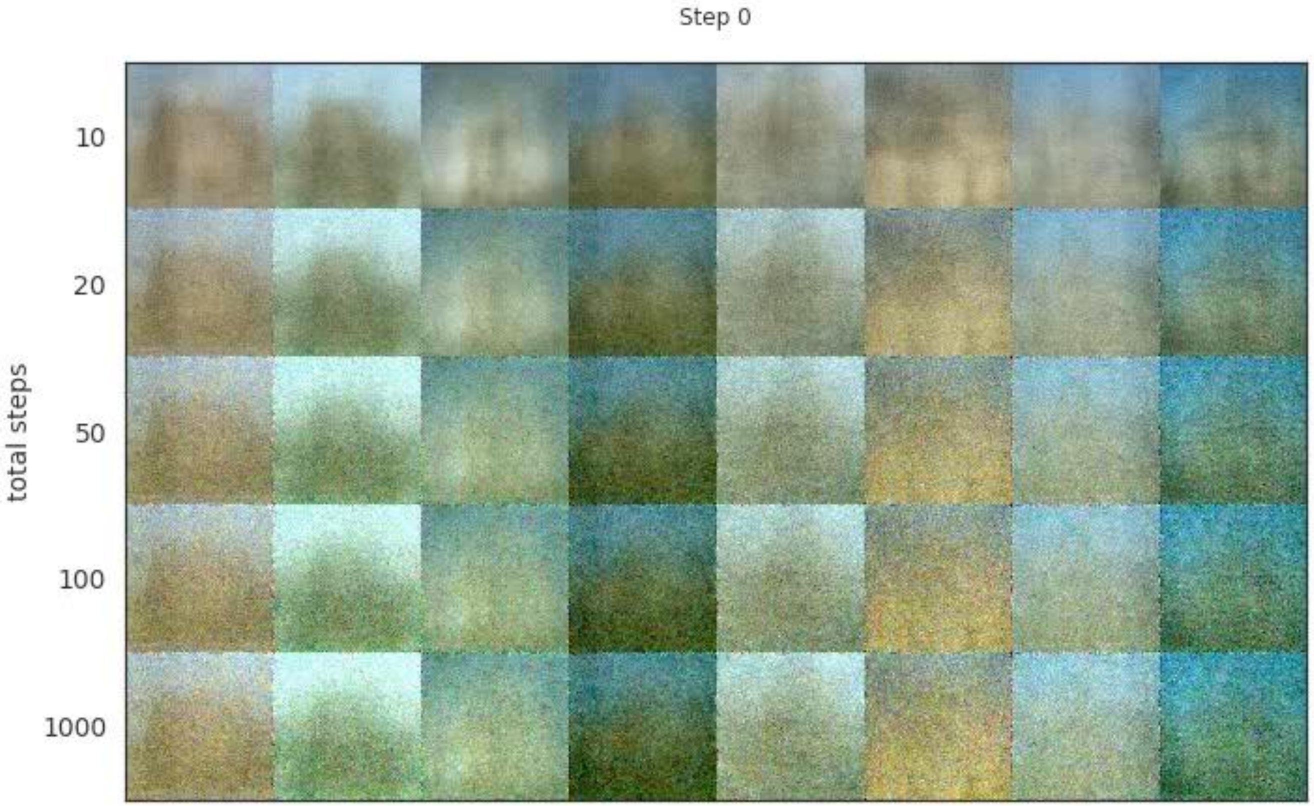


Generating CIFAR10 samples

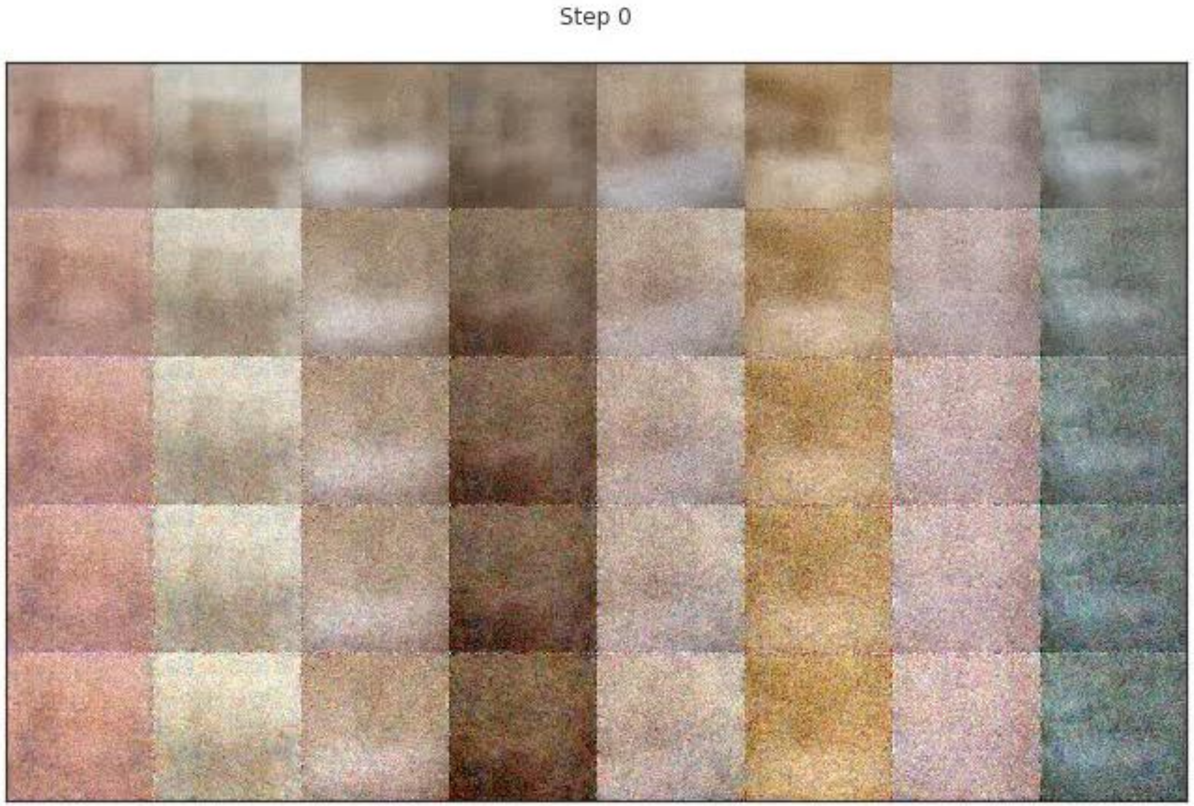


Generating CelebA samples

Denoising Diffusion Implicit Model



Generating LSUN Church samples



Generating LSUN Bedroom samples

Denoising Diffusion Implicit Model

- DDIM significantly reduces the sampling steps of diffusion model
  - Creates the outline of the sample after only 10 steps (DDPM needs hundreds)

Table 1: CIFAR10 and CelebA image generation measured in FID. $\eta = 1.0$ and $\hat{\sigma}$ are cases of DDPM (although Ho et al. (2020) only considered $T = 1000$ steps, and $S < T$ can be seen as simulating DDPMs trained with S steps), and $\eta = 0.0$ indicates DDIM.

		CIFAR10 (32 × 32)					CelebA (64 × 64)				
	S	10	20	50	100	1000	10	20	50	100	1000
η	0.0	**13.36**	**6.84**	**4.67**	**4.16**	4.04	**17.33**	**13.73**	**9.17**	**6.53**	3.51
	0.2	14.04	7.11	4.77	4.25	4.09	17.66	14.11	9.51	6.79	3.64
	0.5	16.66	8.35	5.25	4.46	4.29	19.86	16.06	11.01	8.09	4.28
	1.0	41.07	18.36	8.01	5.78	4.73	33.12	26.03	18.48	13.93	5.98
$\hat{\sigma}$		367.43	133.37	32.72	9.99	**3.17**	299.71	183.83	71.71	45.20	**3.26**

Lecture overview

- Energy-based models
- Score-based Models
- **Denoising Diffusion Models**
  - Diffusion Probabilistic Models
  - Denoising Diffusion Probabilistic Models
  - Denoising Diffusion Implicit Model
  - **Applications**

Diffusion Models for Image Generation

- Beat BigGAN and StyleGAN on generating high-resolution images

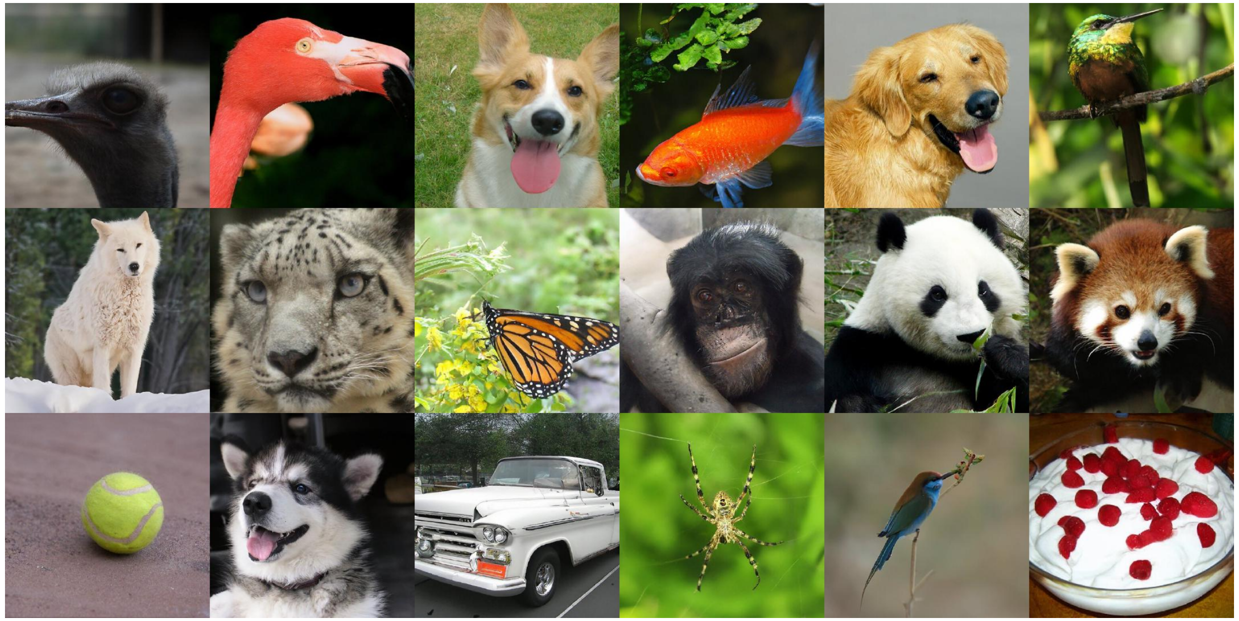
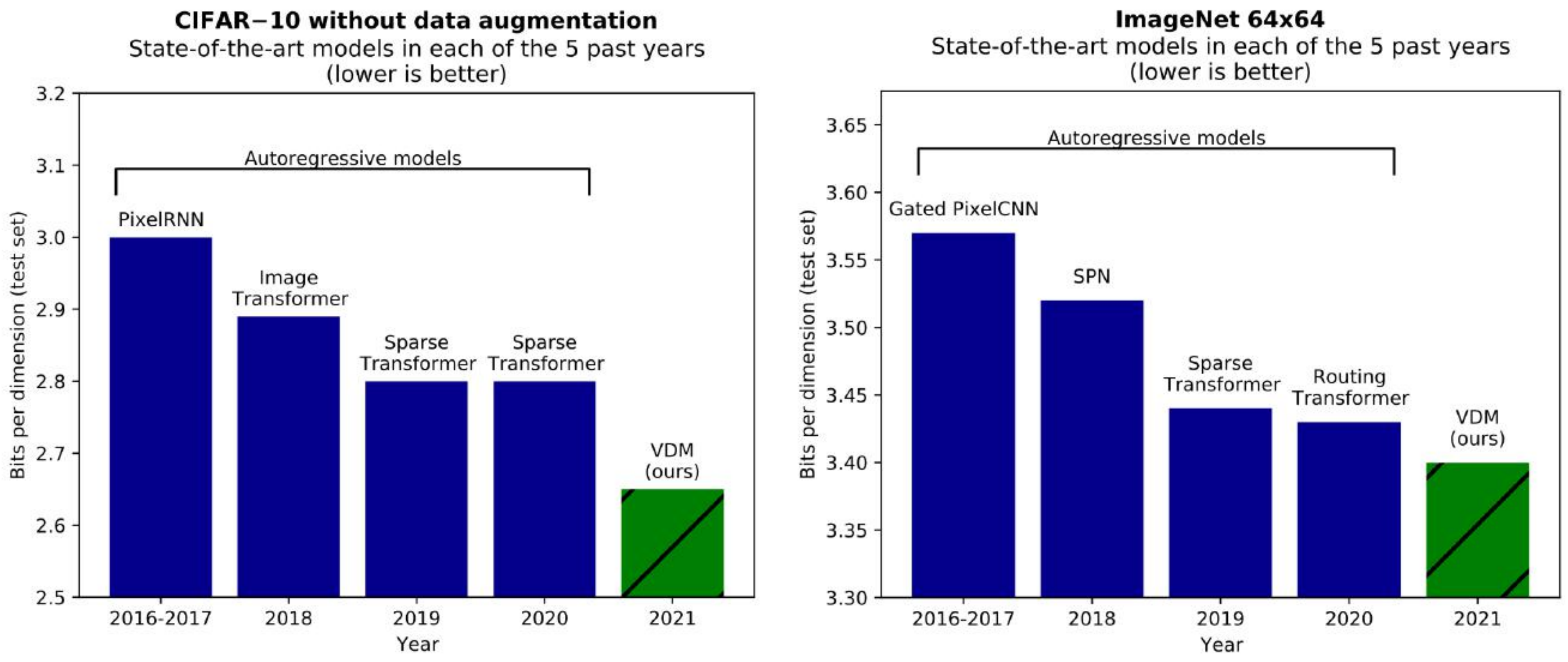


Figure 1: Selected samples from our best ImageNet 512×512 model (FID 3.85)

Model	FID	sFID	Prec	Rec
LSUN Bedrooms 256×256				
DCTransformer† [42]	6.40	6.66	0.44	**0.56**
DDPM [25]	4.89	9.07	0.60	0.45
IDDPM [43]	4.24	8.21	0.62	0.46
StyleGAN [27]	2.35	6.62	0.59	0.48
ADM (dropout)	**1.90**	**5.59**	**0.66**	0.51
ImageNet 512×512				
BigGAN-deep [5]	8.43	8.13	**0.88**	0.29
ADM	23.24	10.19	0.73	**0.60**
ADM-G (25 steps)	8.41	9.67	0.83	0.47
ADM-G	**7.72**	**6.57**	0.87	0.42

Diffusion Models for Density Estimation

- Beat autoregressive models on likelihood score



(a) CIFAR-10 without data augmentation (b) ImageNet 64x64

Figure 1: Autoregressive generative models were long dominant in standard image density estimation benchmarks. In contrast, we propose a family of diffusion-based generative models, *Variational Diffusion Models* (VDMs), that outperforms contemporary autoregressive models in these benchmarks.

Diffusion Models for Image Editing

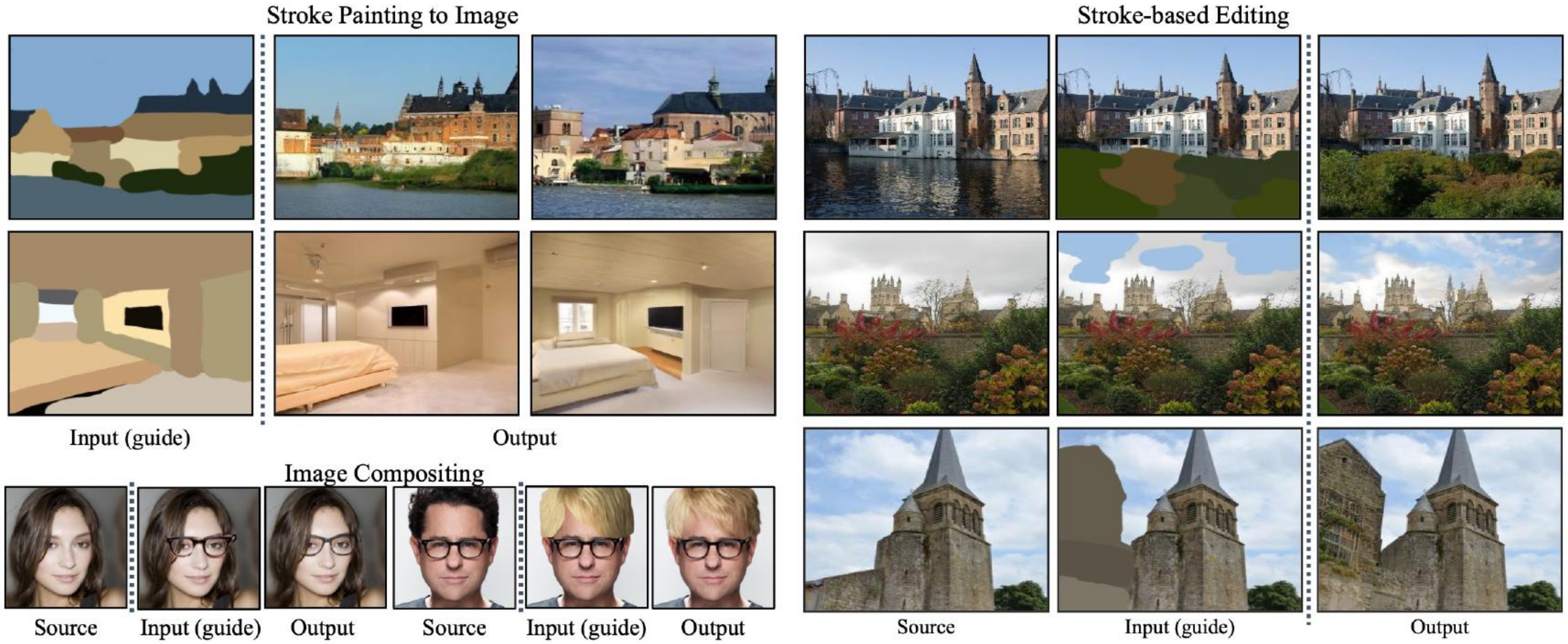


Figure 1: Stochastic Differential Editing (SDEdit) is a **unified** image synthesis and editing framework based on stochastic differential equations. SDEdit allows stroke painting to image, image compositing, and stroke-based editing **without** task-specific model training and loss functions.

Diffusion Models for Image Editing

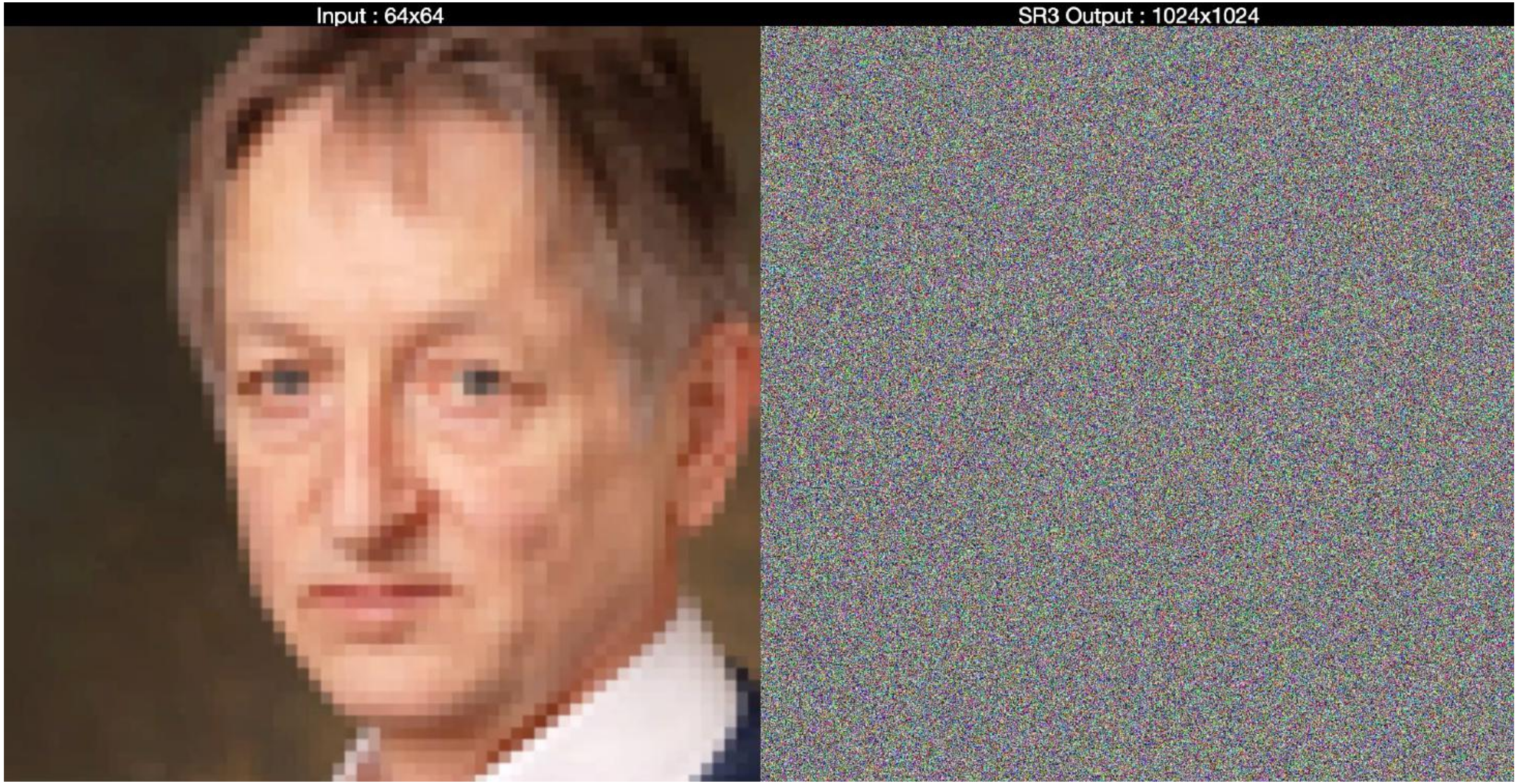
“zebras roaming in the field”

“a girl hugging a corgi on a pedestal”

“a man with red hair”

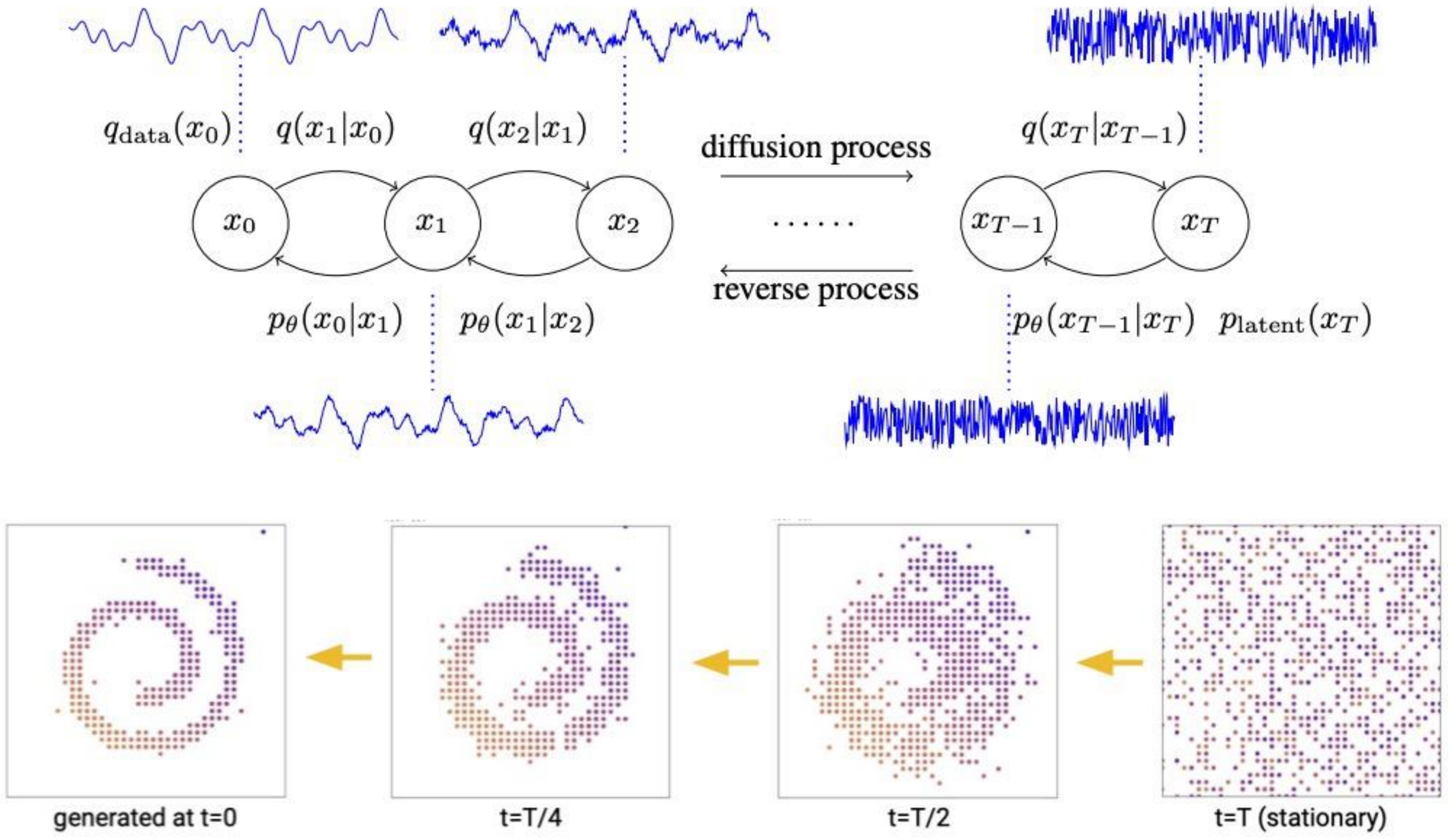
“a vase of flowers”

Diffusion Models for Super Resolution



Results of a SR3 model (64×64 → 512×512), trained on FFHQ, and applied to images outside of the training set.

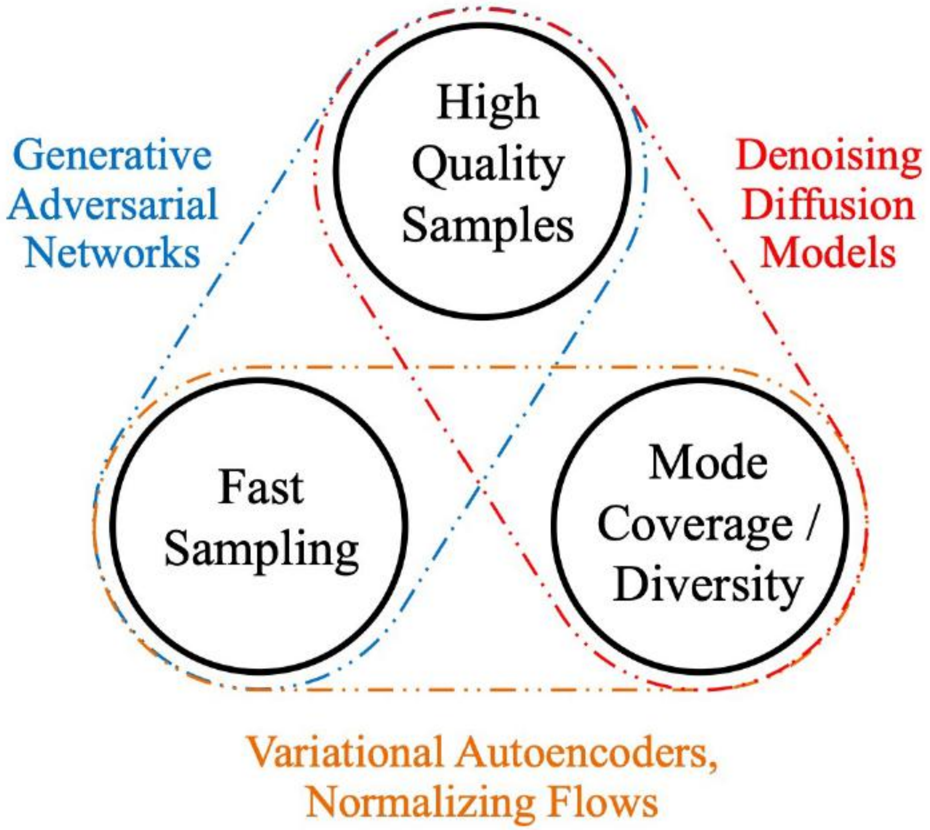
Diffusion Models are also effective for non-visual domains



https://github.com/heejkoo/Awesome-Diffusion-Models

Diffusion Model is All We Need?

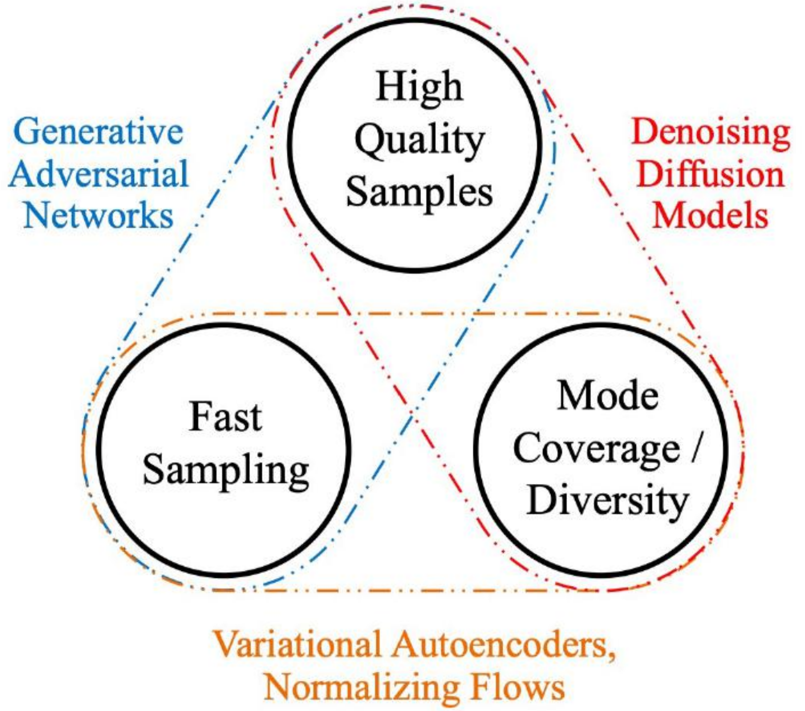
- Trilemma of generative models: Quality vs. Diversity vs. Speed
  - Diffusion model produces diverse and high-quality samples, but generations is slow



Summary

- New golden era of generative models
  - Competition of various approaches: GAN, VAE, flow, diffusion model[1]
  - Also, lots of hybrid approaches (e.g., score SDE = diffusion + continuous flow)

- Which model to use?
  - **Diffusion model** seems to be a nice option for high-quality generation
  - However, **GAN** is (currently) still a more practical solution which needs fast sampling (e.g., real-time apps.)



Next lecture:
Strengths and Weaknesses of Current Models