



Diffusion and Score-Based Generative Models

CVPR2022 Tutorial

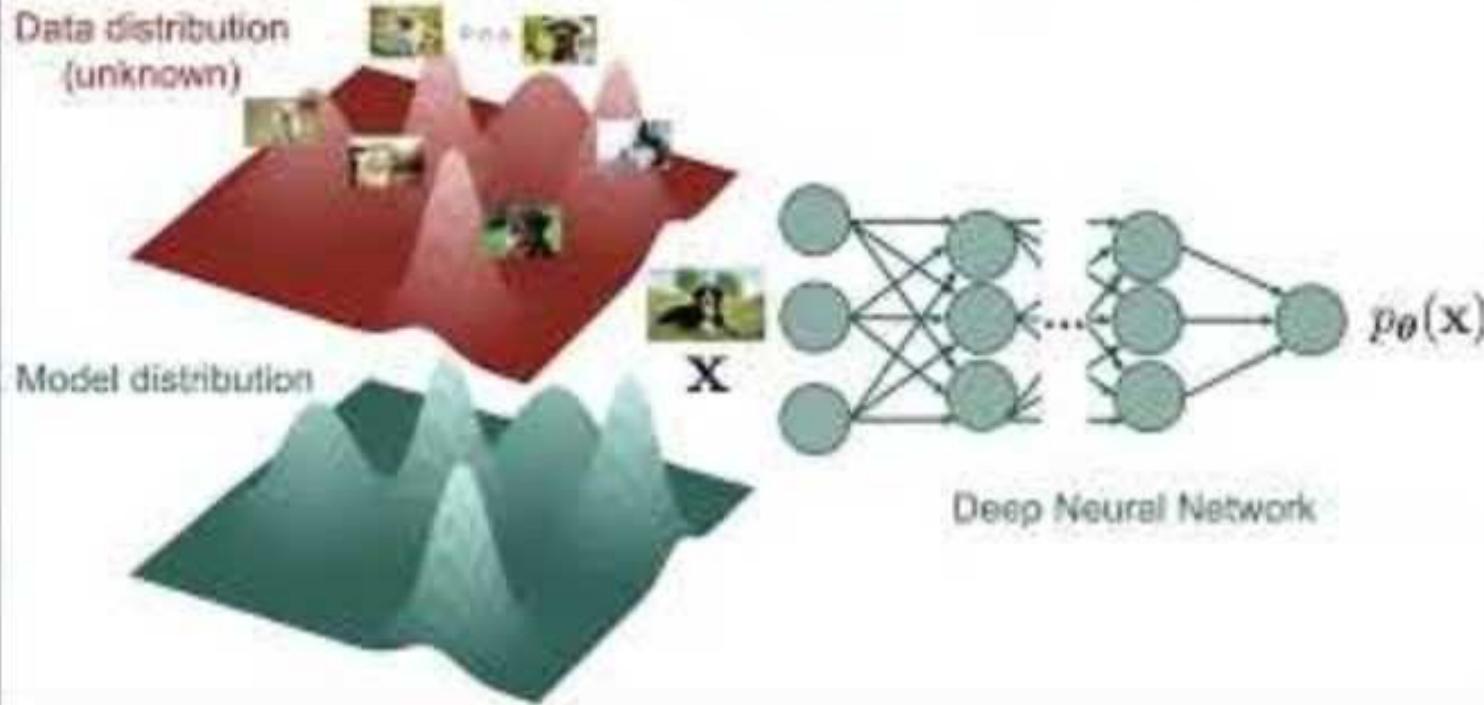
Yang Song

Artificial Intelligence
Creating the Future

Dong-A University

Division of Computer Engineering &
Artificial Intelligence

The key challenge for building complex generative models



December 12, 2022

Diffusion and Score-Based
Generative Models

Yang Song
Stanford University

[Model Distribution]

Estimating the probability distribution of data



...

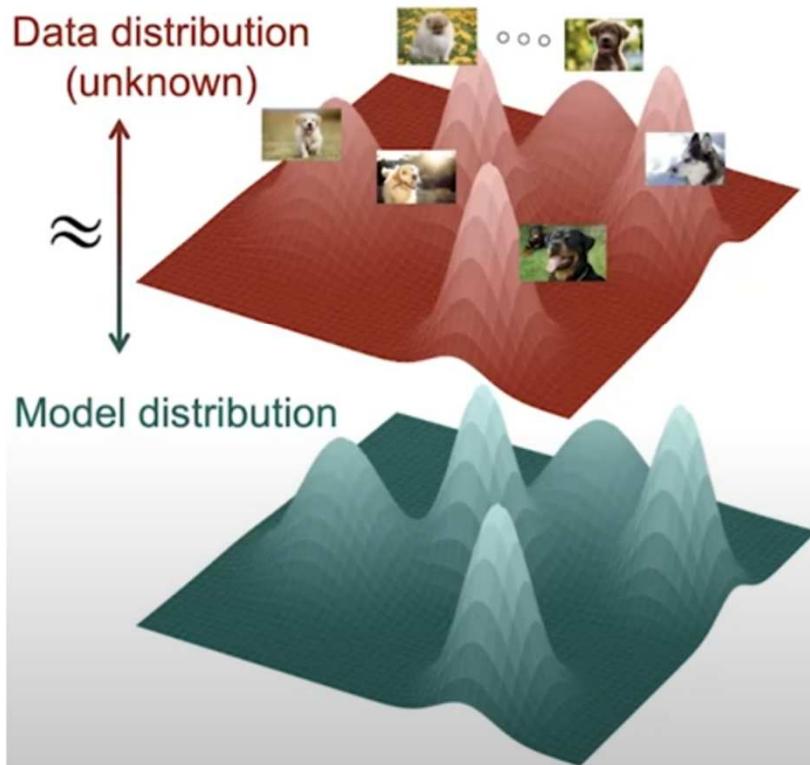


Data samples

- Almost generative models follow the same pipeline.
- The basic idea is to **estimate the probability distribution of data**.
- In order to build a deep generative model, the first thing we need to do is to **collect a large data set**.
- And as a running example, let's suppose the data set contains many images of dogs.

[Model Distribution]

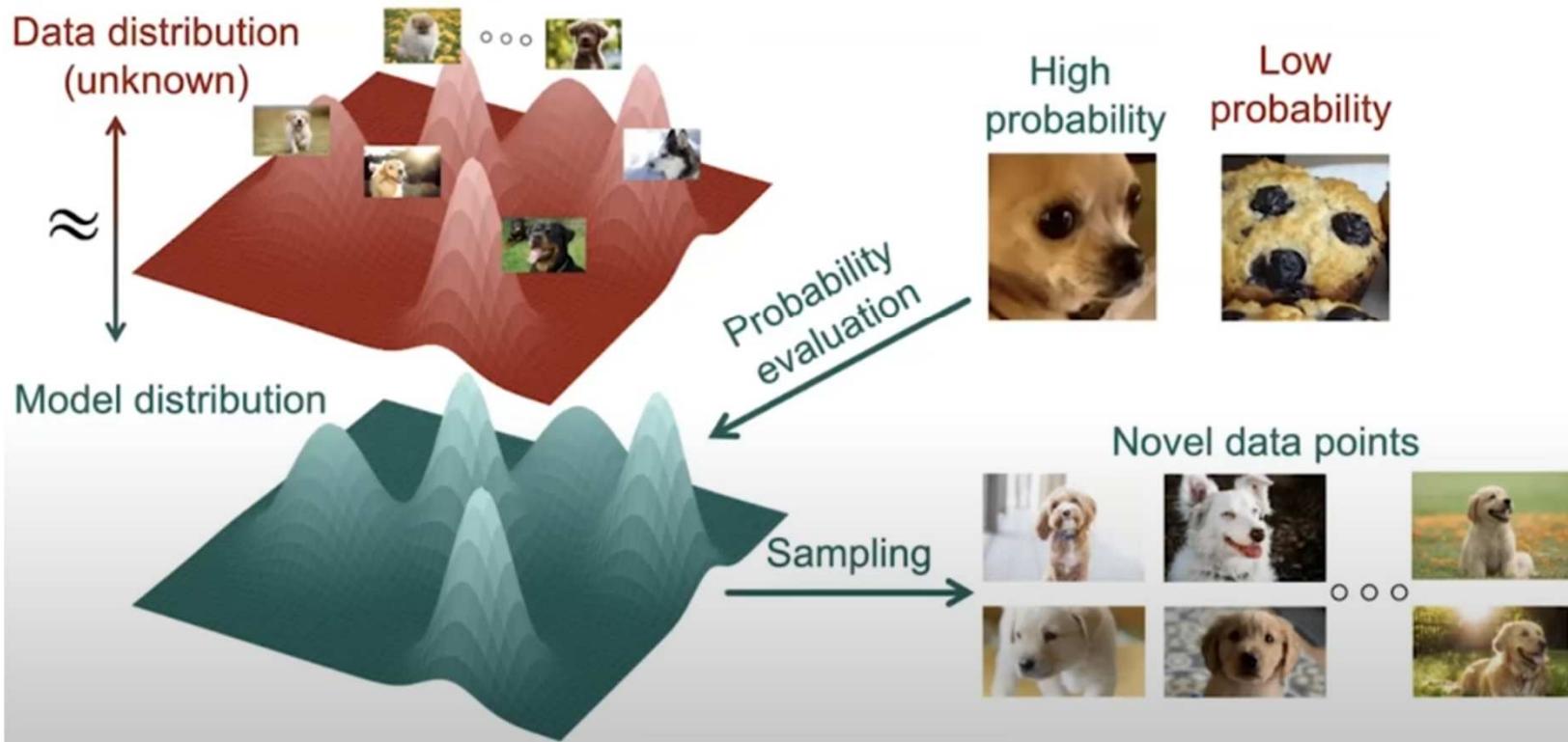
Estimating the probability distribution of data



- A typical assumption in statistics and machine learning is that all those data points in our training data set come from some underlying data distribution.
- In other words, those data points are basically ID samples from this data distribution, but we don't have the analytical form of the data distribution, and we have to estimate it.
- And to estimate this data distribution, we have to create a model. This model represents parameterized probability distribution, which we call the model distribution.
- And we hope to tune this model parameter to make sure this model distribution is close to the data distribution in a certain sense.
- So if this model distribution is very close to the data distribution, then we can use the model for many important applications.

[Model Distribution]

Estimating the probability distribution of data



[Sampling]

- And one example is, of course, we can generate an unlimited number of novel data points just by sampling from this model distribution.

[Probability Evaluation]

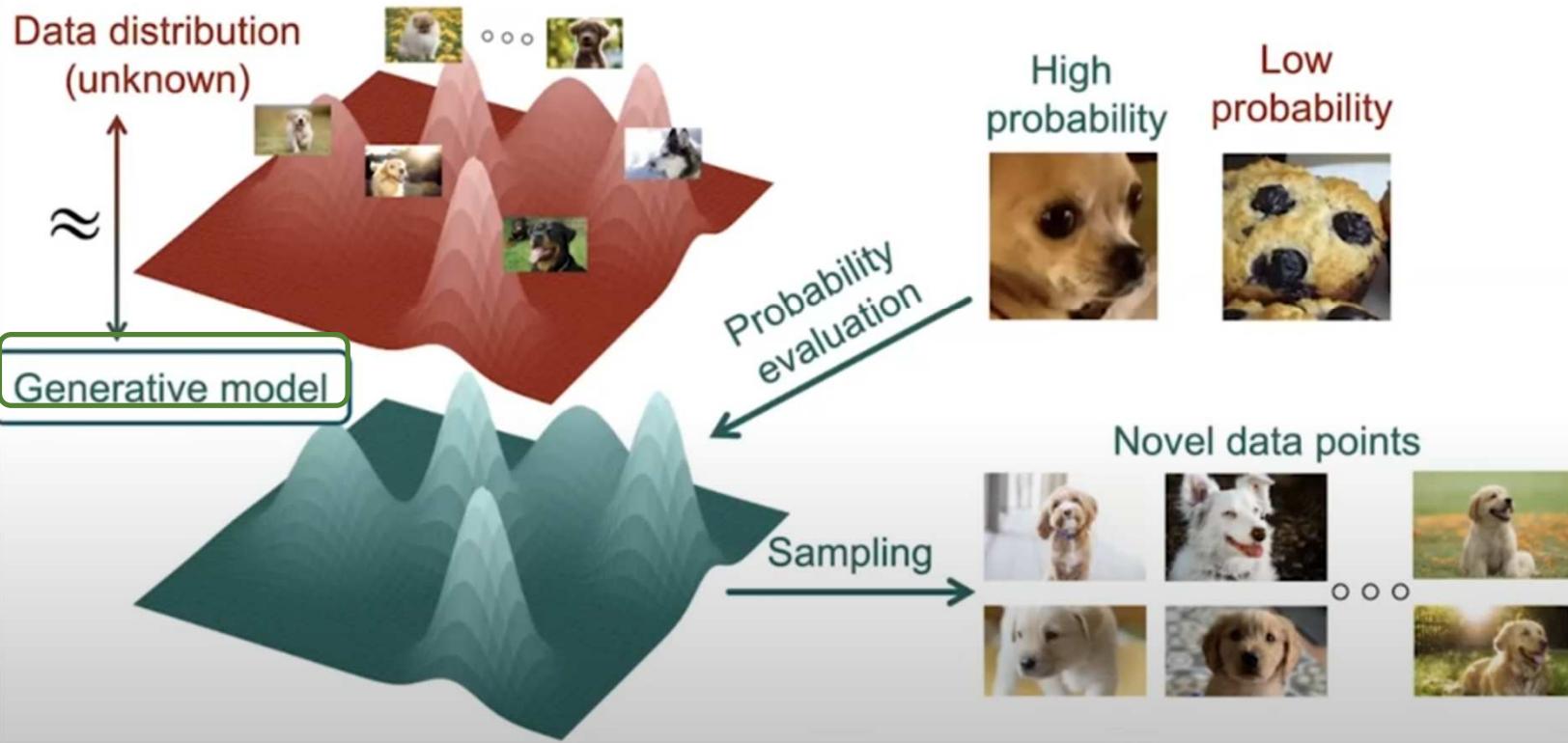
- Another application is we can use this model distribution to compute the probability value for any potential data point.
- So as an example for a data point, like a picture of a chihuahua, because it is a picture of a dog, it is actually within our data distribution. And therefore, this model distribution usually assigns high probability values for such data points. For some irrelevant data point, like a picture of a muffin, because it is not a picture of a dog, a good model distribution will add lower probability values to such images.

[Generative Model]

- So because this model distribution provides a way to generate novel data points, we also refer to it as a generative model.

[Model Distribution]

Estimating the probability distribution of data



[Sampling]

- And one example is, of course, we can generate an unlimited number of novel data points just by sampling from this model distribution.

[Probability Evaluation]

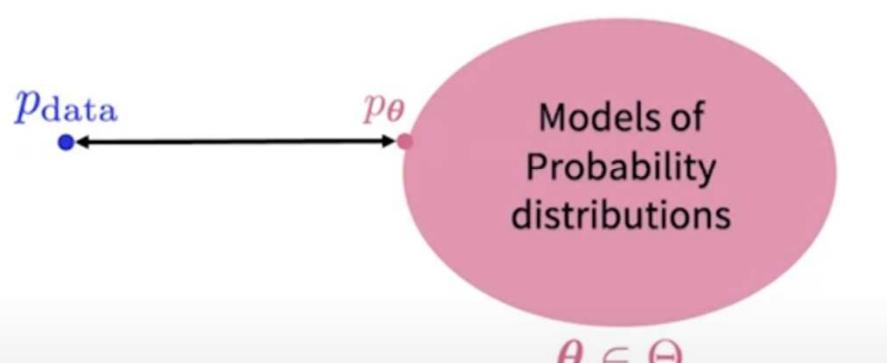
- Another application is we can use this model distribution to compute the probability value for any potential data point.
- So as an example for a data point, like a picture of a chihuahua, because it is a picture of a dog, it is actually within our data distribution. And therefore, this model distribution usually assigns high probability values for such data points. For some irrelevant data point, like a picture of a muffin, because it is not a picture of a dog, a good model distribution will add lower probability values to such images.

[Generative Model]

- So because this model distribution provides a way to generate novel data points, we also refer to it as a generative model.

[Model Distribution]

Training generative models



$$\mathbf{x}_i \stackrel{\text{i.i.d.}}{\sim} p_{\text{data}}$$
$$i = 1, 2, \dots, N$$

- How can we train those generative models?
- As we know, when we have a large data set, we may formalize the problem a little bit further.
- We can use a simple x_i to represent each data point in the data set and we have a total of N data points. And our model provides a family of probability distributions.
- We hope to find a single probability distribution inside this huge family by minimizing the distance from P_{θ} to P_{data} .
- And afterwards, we can just generate samples from P_{data} .

[Data Distribution]

The key challenge for building complex generative models

Data distribution
(unknown)



X
Data distribution is
extremely complex for
high dimensional data.

Model distribution



?
How to build a complex
model to fit the data
distribution?

- However, there is one key challenge associated with this framework.
- That is, our data distribution can be extremely complicated, especially for data in high dimensions.
- So consider how complicated it might be for distributions of images, video, audio. It might have millions of dimensions.
- And as a result, we have to build a very powerful model distribution in order to estimate our data distribution.
- So **how can we build a powerful model distribution?**

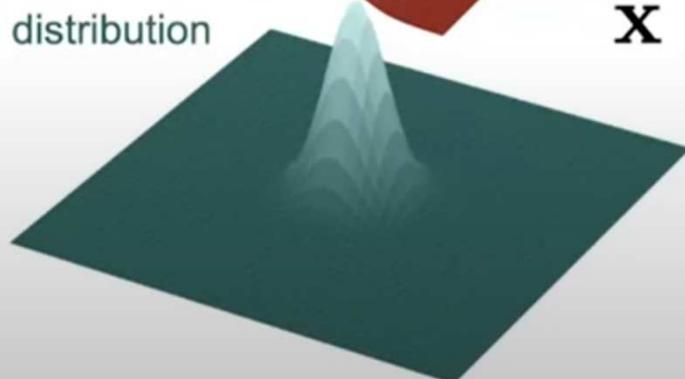
[Data Distribution]

The key challenge for building complex generative models

Data distribution
(unknown)



Model distribution



$$\mathbf{x}$$
$$p_{\mu}(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}} e^{-\frac{\|\mathbf{x}-\mu\|_2^2}{2}}$$

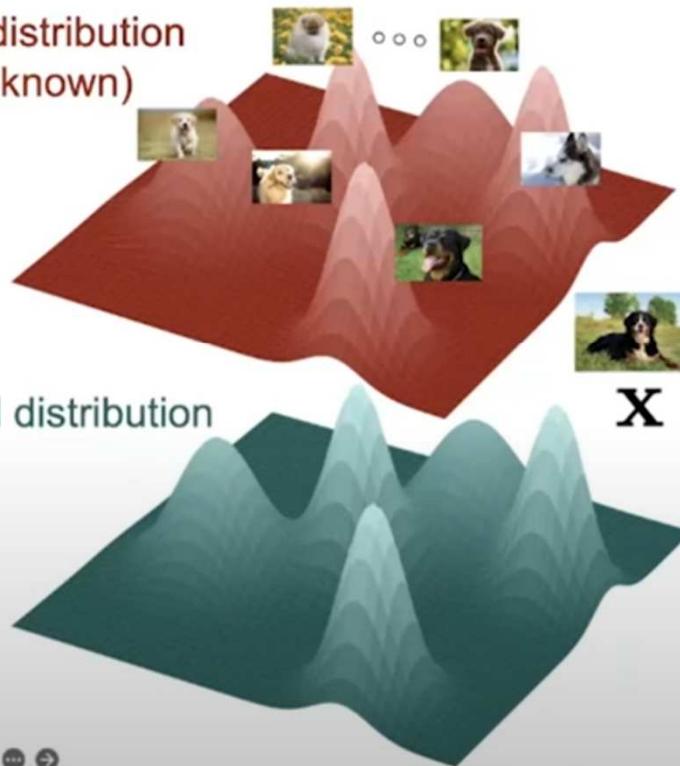
A computational graph diagram showing four nodes connected to a final output node labeled \mathbf{x} . The first three nodes are grouped together and have arrows pointing to the final node.

- Let's recall that in statistics we often work with simple distributions, such as Gaussian distribution.
- Of course, a Gaussian distribution is too simple. It won't be able to approximate our complicated data distribution. But it serves as a good starting point.
- A **Gaussian distribution** is basically a **computational graph** that has **two layers**. *The first layer corresponds to the input data point. The second layer is a single unit that basically gives you the probability density function of this Gaussian distribution.*
- This computation is very simple, and the middle in this slide denotes the mean parameter of discussing distribution. By changing the parameter to middle, you are basically changing the mean of this Gaussian.
- But as we said, Gaussian models are too simple. How can we make a more complicated model?
- Well, a very natural idea is **to leverage a bigger and deeper computational graph**.

[Data Distribution]

The key challenge for building complex generative models

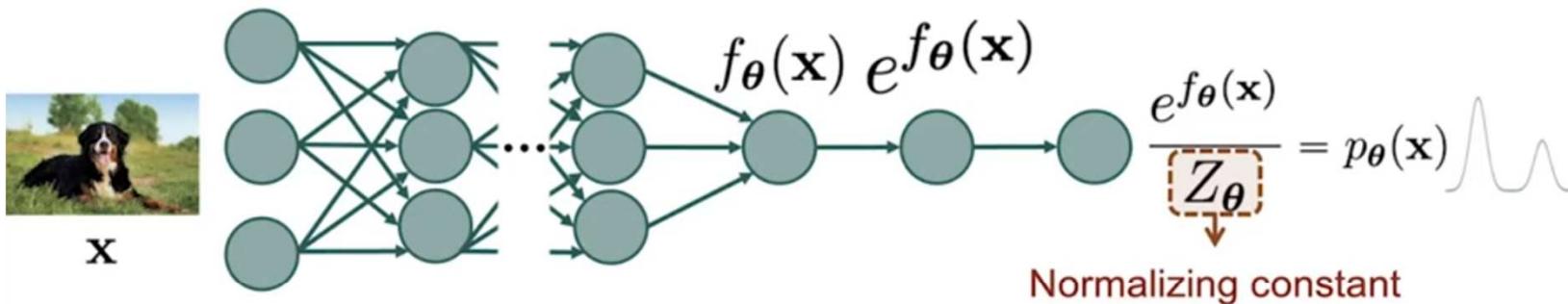
Data distribution
(unknown)



- And we also call it a **deep neural network**. So we hope to use that deep neural network to represent a complicated probability distribution P_θ , where theta denotes the weights in this deep neural network.
- And when we use deep neural networks to build those powerful generative models, we obtain **deep generative models**.

[Data Distribution]

The key challenge for building complex generative models

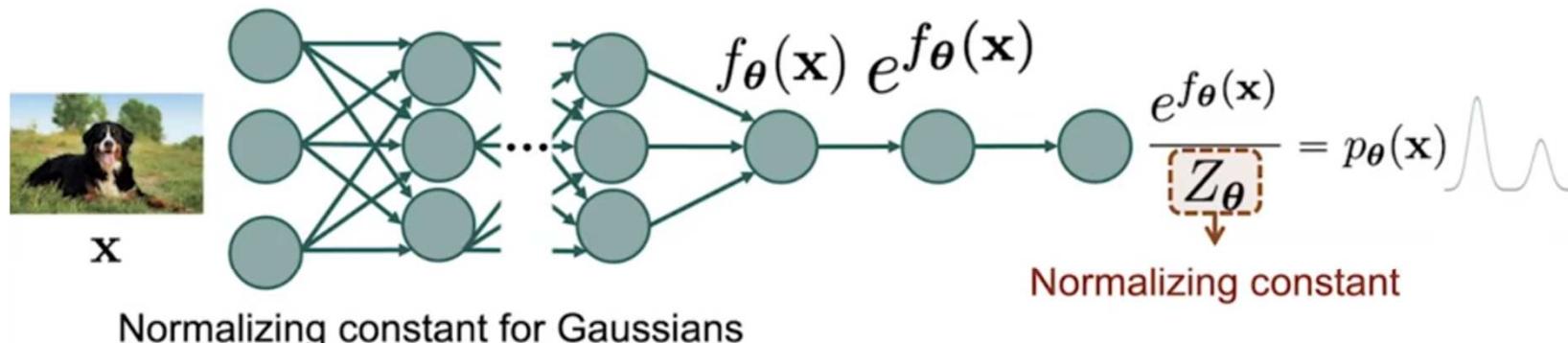


- By definition, this normalizing constants should be computed by evaluating the high dimension integral of the exponential function of our theta over all possible values of x in the space.

- But it is actually **nontrivial** to use a **deep neural network** to directly represent a probability distribution because we typically view a deep neural network as a black box that converts a high dimension input \mathbf{x} to a typically one-dimensional output f_{theta} .
- So this output value f_{theta} does not directly model distribution because it may not be positive everywhere.
- Our first step to convert this into our probability density is to take the exponential of the output. So then the output becomes positive.
- And then we can normalize the output by dividing by a constant Z_{theta} in order to construct a probability distribution which has positive values everywhere and is also properly normalized.
- So the denominator here is called **the normalizing constant**.

[Data Distribution]

The key challenge for building complex generative models



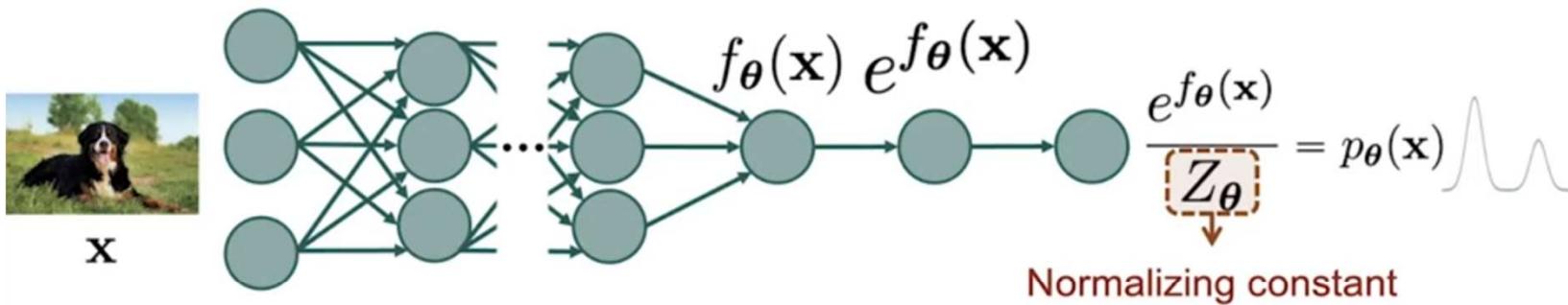
Normalizing constant for Gaussians

$$Z_{\mu} = \frac{1}{(2\pi)^{d/2}}$$

- In the special case of Gaussian models, this normalizing constant is very simple to compute because f_{θ} in Gaussian models has a very simple form.
- So we can directly compute the integral in closed form.

[Data Distribution]

The key challenge for building complex generative models

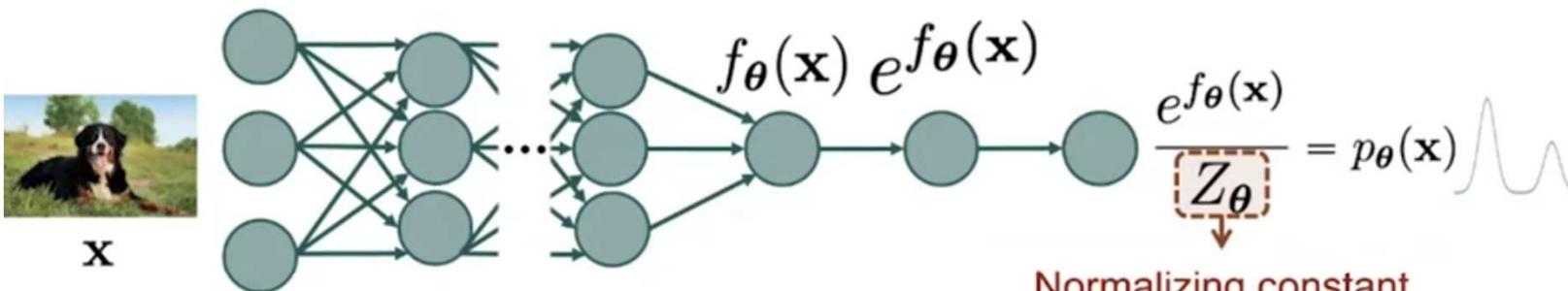


$$Z_{\theta} = \int e^{f_{\theta}(\mathbf{x})} d\mathbf{x}$$

- We can directly compute the integral in closed form.
- But when we are trying to handle more powerful deep neural network models, this normalizing constant becomes intractable to compute.

[Data Distribution]

The key challenge for building complex generative models

**x**

$$Z_{\theta} = \int e^{f_{\theta}(x)} dx$$



#P-complete even for discrete variables

Ludwig Boltzmann
(1844-1906)Willard Gibbs
(1839-1903)Gustav Zeuner
(1828-1907)Johannes van der Waals
(1837-1923)

Thermodynamics & Statistical Mechanics

- And as a quick example, even if we consider a simplified case where \mathbf{x} is discrete and in which case the integral becomes a summation, **computing this normalizing constant** is still a **P-complete problem**, which is at least as hard as NP-complete.
- And this difficulty is by no means the unique challenge in deep generative modeling.
- You can find many similar challenges in adjacent fields, such as thermodynamics and statistical mechanics. And people have been studying this problem for quite a while.

[Deep Genetic Models]

Tackling the intractable normalizing constant

Approximating the normalizing constant

- Energy-based models [Ackley et al. 1985, LeCun et al. 2006]

Using restricted neural network models

- Autoregressive models [Bengio & Bengio 2000, van den Oord et al. 2016]
- Normalizing flow models [Dinh et al. 2014, Rezende & Mohamed 2015]
- Variational auto-encoders [Kingma & Welling 2014, Rezende et al. 2014]

Modeling the Generation Process Only

- Generative Adversarial Networks (GANS) [Goodfellow et al. 2014]



Inaccurate probability evaluation



Restricted model family



Cannot evaluate probabilities

- In the current literature of deep generative models, there are mostly three approaches to **address this intractable normalizing constant difficulty**.
- And as a result, we can actually categorize **deep generative models** into **three different categories of families**.
 - The first category is based on **approximating this normalizing constants** using approaches such as Markov chain Monte Carlo.
 - One typical example inside this family is **energy-based models trained by contrastive divergence**.
 - The disadvantage of this direction is then because we have to approximate this normalizing constant, we **cannot compute the probability value accurately**, since the probability value requires dividing by this approximate normalizing constant.

[Deep Genetic Models]

Tackling the intractable normalizing constant

Approximating the normalizing constant

- Energy-based models [Ackley et al. 1985, LeCun et al. 2006]

Using restricted neural network models

- Autoregressive models [Bengio & Bengio 2000, van den Oord et al. 2016]
- Normalizing flow models [Dinh et al. 2014, Rezende & Mohamed 2015]
- Variational auto-encoders [Kingma & Welling 2014, Rezende et al. 2014]

Modeling the Generation Process Only

- Generative Adversarial Networks (GANS) [Goodfellow et al. 2014]



Inaccurate probability evaluation



Restricted model family



Cannot evaluate probabilities

- The second major approach is based on using **restricted neural network models**, such that this normalizing constant is tractable by construction.
- There are a few examples inside this family, but the challenges are once we restrict our neural network models, we also **limit the flexibility of deep generative models** that we can potentially build along this direction.

[Deep Genetic Models]

Tackling the intractable normalizing constant

Approximating the normalizing constant

- Energy-based models [Ackley et al. 1985, LeCun et al. 2006]

Using restricted neural network models

- Autoregressive models [Bengio & Bengio 2000, van den Oord et al. 2016]
- Normalizing flow models [Dinh et al. 2014, Rezende & Mohamed 2015]
- Variational auto-encoders [Kingma & Welling 2014, Rezende et al. 2014]

Modeling the Generation Process Only

- Generative Adversarial Networks (GANS) [Goodfellow et al. 2014]



Inaccurate probability evaluation



Restricted model family



Cannot evaluate probabilities

- The last category is based on **modeling the data generating process** directly instead of modeling the probability density function.
- The most predominant example in this family is generative adversarial networks.
- However, because those approaches to not model the underlying data distribution, they cannot give us accurate probability values.

[Deep Genetic Models]

Desiderata of better generative modeling



Inaccurate probability evaluation



Restricted model family

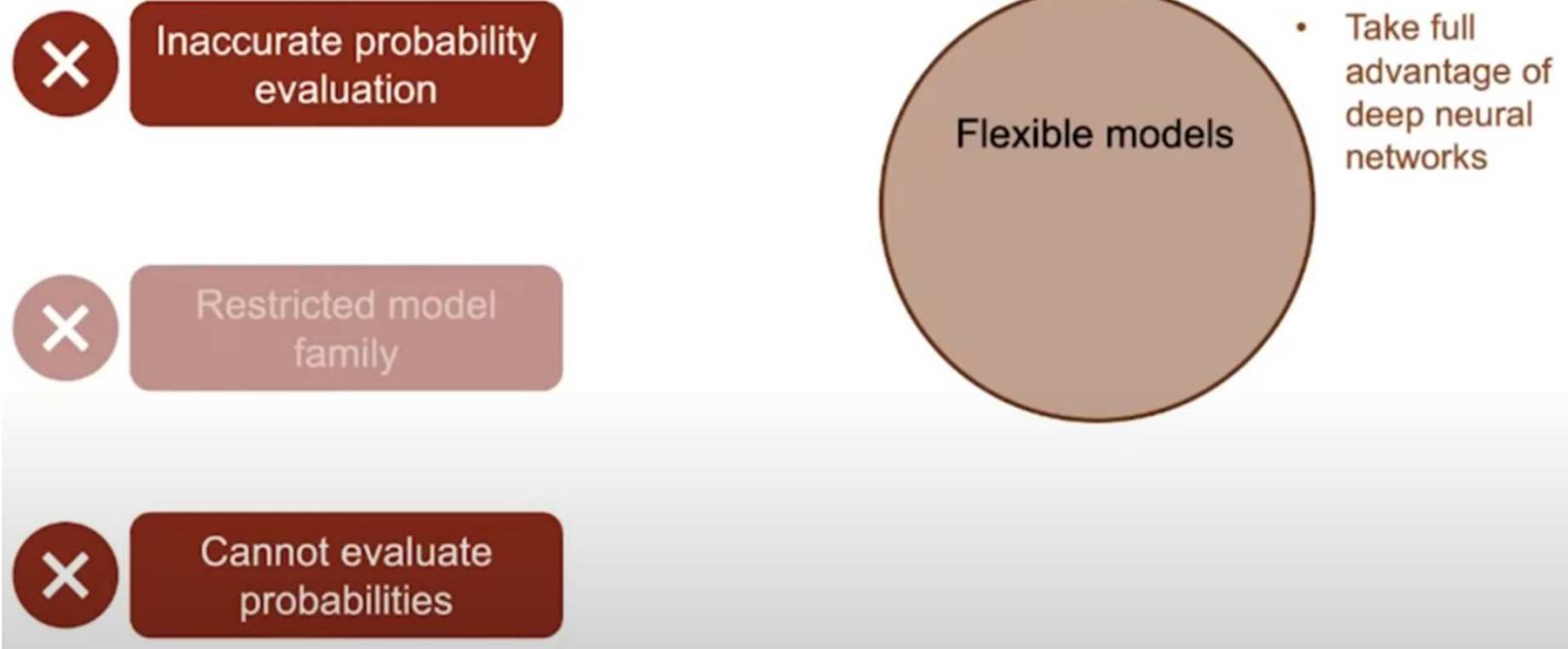


Cannot evaluate probabilities

- These are a few challenges associated with previous generative modeling frameworks.
- And if we want to address those difficulties by proposing **a better framework of generative modeling**, then we require this better framework to satisfy certain desiderata.

[Deep Genetic Models]

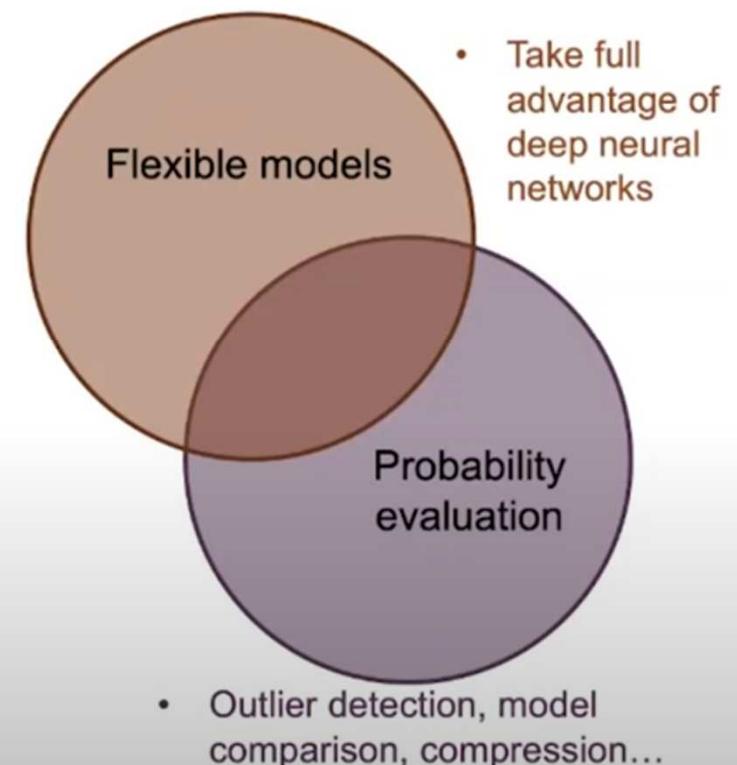
Desiderata of better generative modeling



[Deep Genetic Models]

Desiderata of better generative modeling

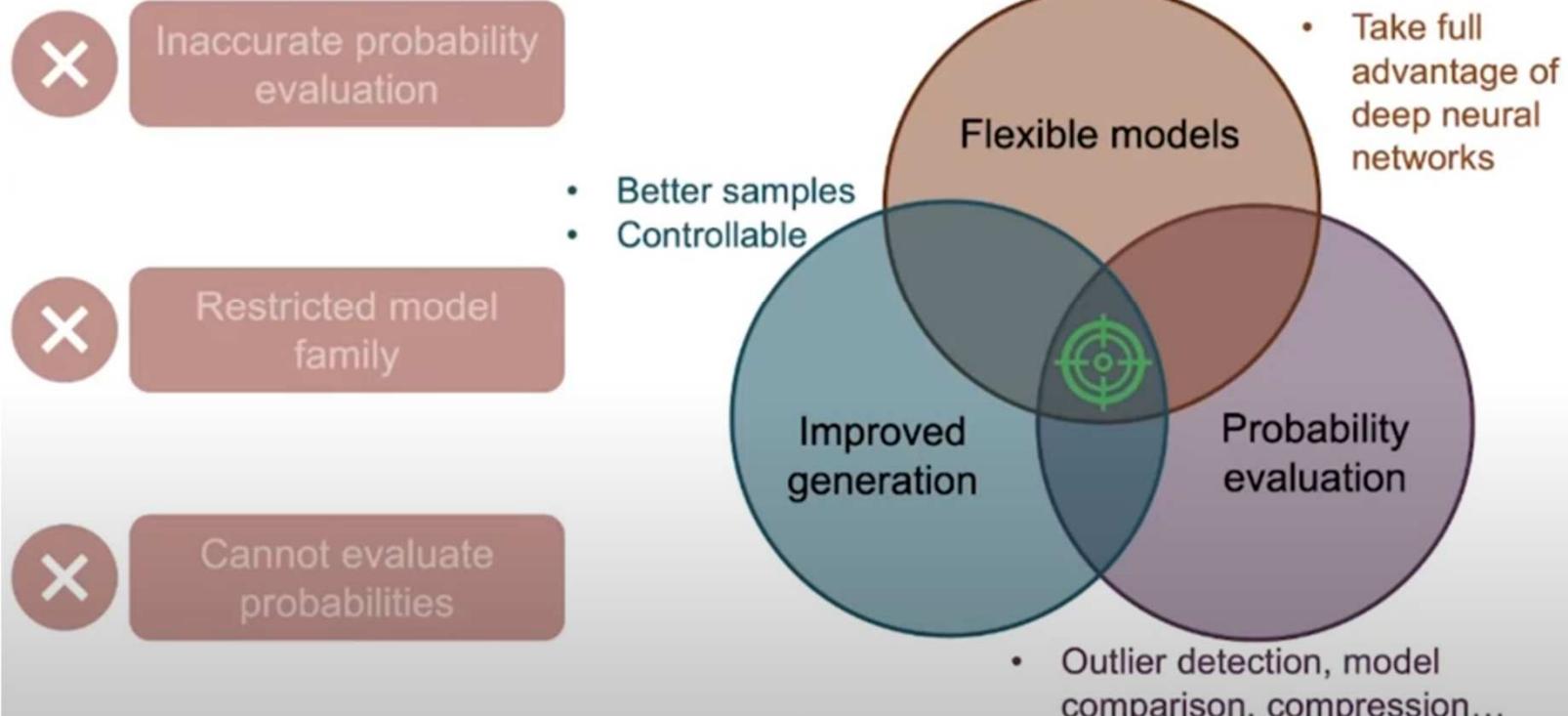
-  Inaccurate probability evaluation
-  Restricted model family
-  Cannot evaluate probabilities



- The second desideratum is we hope to **evaluate probability values accurately** using this new framework of generative modeling.
- If we can evaluate the probability values accurately, we can address the rest of the challenges on the left side. And then moreover, those accurate probability values are very important for applications such as outlier detection, model comparison, or lossless compression.

[Deep Genetic Models]

Desiderata of better generative modeling



- And finally, because we are aiming to build a **more powerful framework of general models**, we of course want to generate samples with better quality.
- So not only do we want to generate samples with **better quality**, we also want to **control this generation process** in a principled way so that we may use this generative model for numerous downstream applications.
- And one example is medical image reconstruction, which I will discuss briefly later in the tutorial.
- So now, in today's talk, I will show you **one such framework that satisfies all three desiderata** listed here.
- And the key of this framework is to work with **score functions** to **represent our probability distribution**.

[Score Functions]

Our proposal: working with score functions

$$p(\mathbf{x})$$

Probability density function


$$\nabla_{\mathbf{x}} \log p(\mathbf{x})$$

(Stein) score function

- What is the score function?
- Well, suppose we have a continuous probability distribution where we use $p_{\mathbf{x}}$ to represent the probability density function.
- We define the **score function** as the **gradient of log px**.
- This quantity has multiple names. It can be called a **Stein score function** to differentiate from Fisher score functions that typically appear in statistics. It can also be called as the score function or simply score.
- Be careful **this gradient is taken with respect to the random variable \mathbf{x} , it is not taken with respect to any model parameter like theta.**
- What does our score function look like?

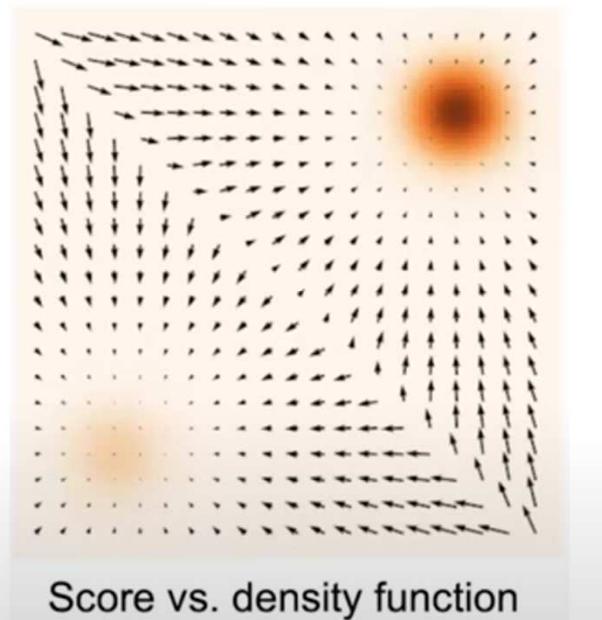
[Score Functions]

Our proposal: working with score functions

$p(\mathbf{x})$
Probability density function



$\nabla_{\mathbf{x}} \log p(\mathbf{x})$
(Stein) score function



- Let's consider a simple example which is a mixture of two Gaussians. This figure shows a density function and the score function for this mixture of Gaussian distribution.
- The density function is a color coded, where darker color indicates higher density. The score function is a vector field that gives the direction where the density function grows most quickly.
- So **given the density function**, we can **compute the score function** very easily because we can just take the derivative.
- Conversely, **with the score function**, we can also **recover the density function** in principle by computing integrals.
- So mathematically this score function preserves all the information in the density function. *They are equivalent to [INAUDIBLE].. But computationally, this score function is much easier to work with compared to the density function.*

[Outline]

Score-based generative modeling: outline

Flexible models

- Bypass the normalizing constant
- Principled statistical methods

[Song et al. UAI 2019 oral]

Improved generation

- Higher sample quality than GANs
- Controllable generation

[Song & Ermon. NeurIPS 2019 oral]
[Song & Ermon. NeurIPS 2020]
[Song et al. ICLR 2021 oral]
(Outstanding Paper Award)
[Song et al. ICLR 2022]

Probability evaluation

- Accurate probability evaluation
- Better estimation of data probabilities

[Song et al. ICLR 2021 oral]
[Song et al. NeurIPS 2021 spotlight]

- When we work with the score function for representing probability distributions, we get our score-based generative models. And I will show you that this score-based generative model has multiple advantages.
- [First] it allows very **flexible models** because the score functions actually **do not need to be normalized at all**. Which means you can use a very flexible neural net models to represent this score function, and we can **learn such models or score functions from data using principle statistical approaches**.

[Flexible models]

Score-based generative modeling: outline

Flexible models

- Bypass the normalizing constant
- Principled statistical methods

[Song et al. UAI 2019 oral]

Improved generation

- Higher sample quality than GANs
- Controllable generation

[Song & Ermon. NeurIPS 2019 oral]
[Song & Ermon. NeurIPS 2020]
[Song et al. ICLR 2021 oral]
(Outstanding Paper Award)
[Song et al. ICLR 2022]

Probability evaluation

- Accurate probability evaluation
- Better estimation of data probabilities

[Song et al. ICLR 2021 oral]
[Song et al. NeurIPS 2021 spotlight]

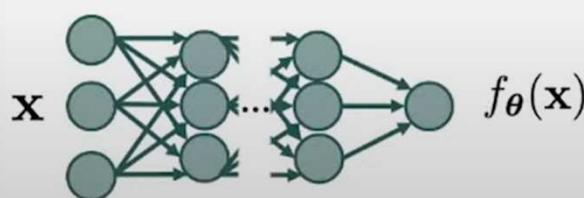
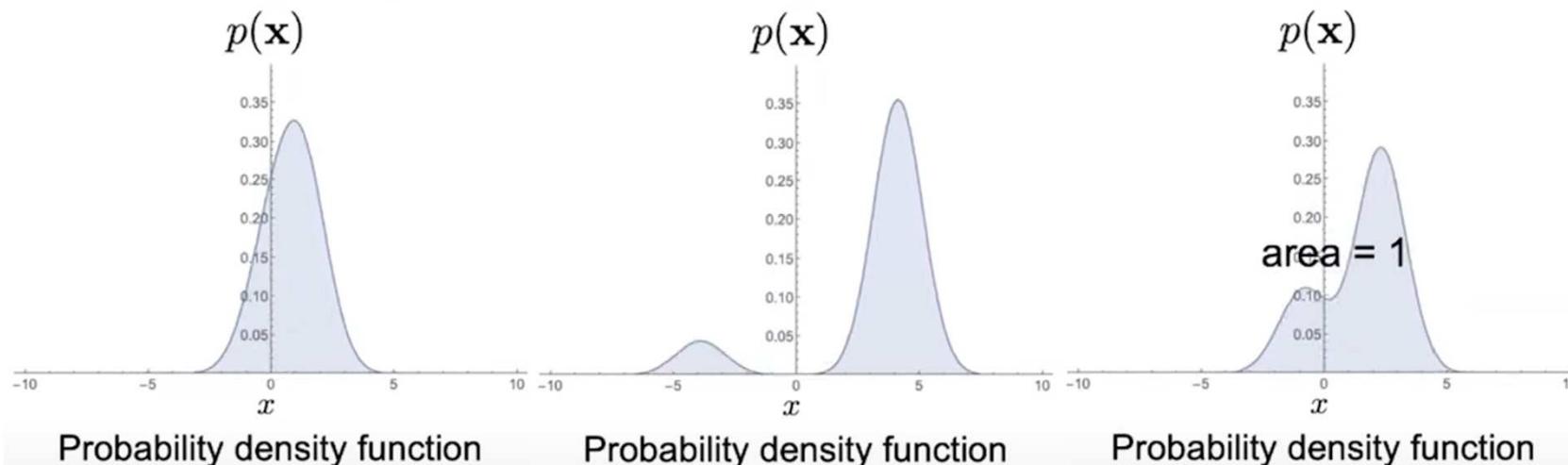
- [Second] We can **directly generate samples from those models of score functions**, and those samples could have surprisingly **good quality** and can be even better than [INAUDIBLE] in many situations. And moreover, we can **control the sample generation process** in a principal way for many important applications.

- [Third], even if we only have the model of the score function, we can still **compute the probability values accurately**. And empirically, we can even **obtain better probability values** compared to those models that directly work with probability density functions.
- So in the rest of the tutorial, I will first focus on how score-based generative modeling allows very flexible models.

[Flexible models]

Flexible models

Score functions bypass the normalizing constant



- Recall that one major difficulty in deep generative modeling is due to the **intractable normalizing constant problem** when we are trying to model the probability density function.

Indeed, if we want to model this probability distribution using a normalized probability model, then no matter how we change our model parameter in some of the architectures or other configurations, we **always have to ensure that the distribution represented is fully normalized**.

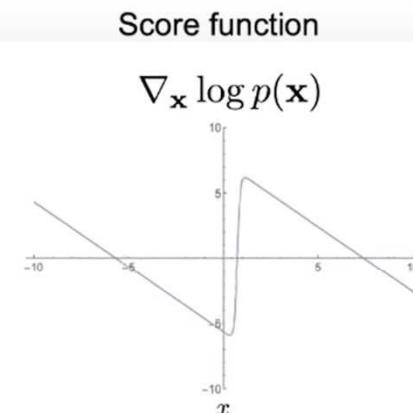
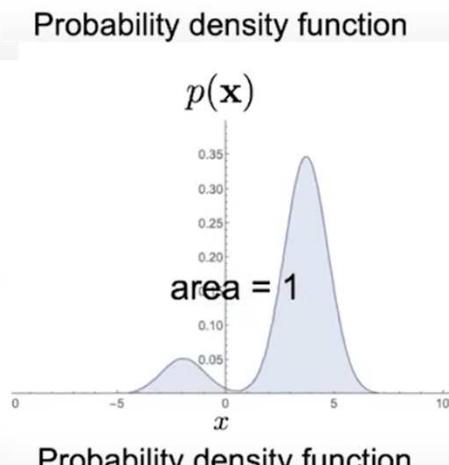
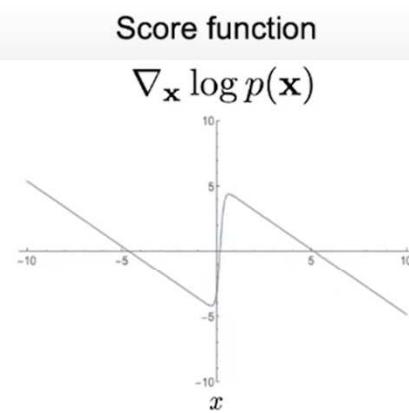
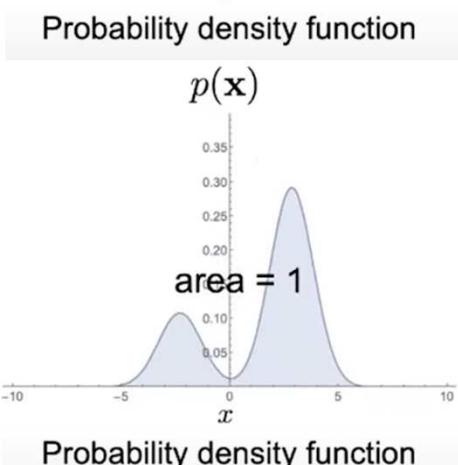
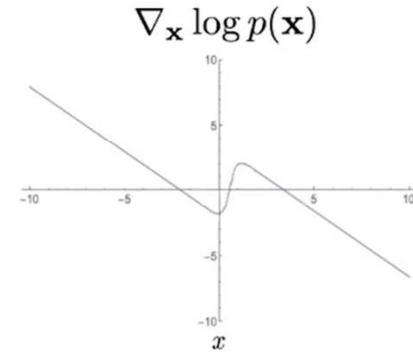
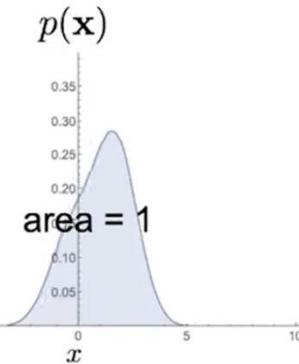
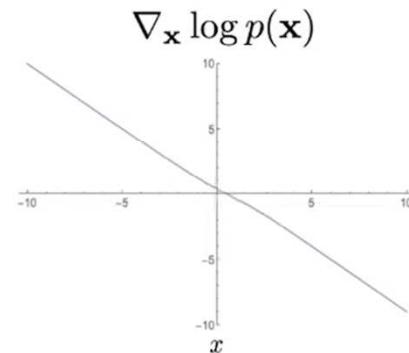
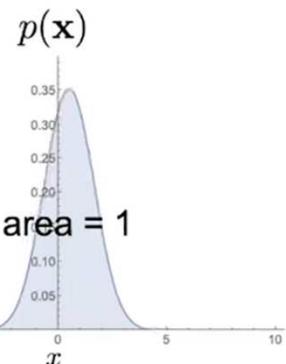
In other words, **the area below this curve has to be 1**.

And due to this constraint, when we use the deep neural network model to those density functions, we **always have to deal with this intractable normalizing constant difficulty**.

[Flexible models]

Flexible models

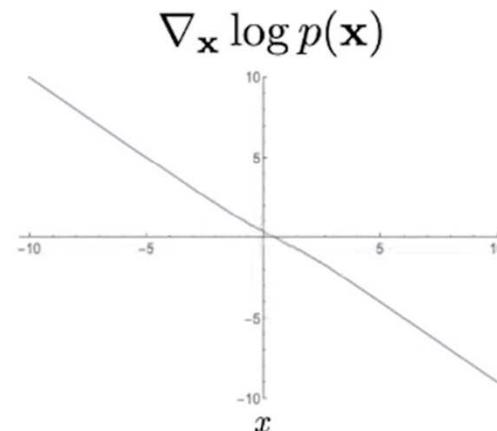
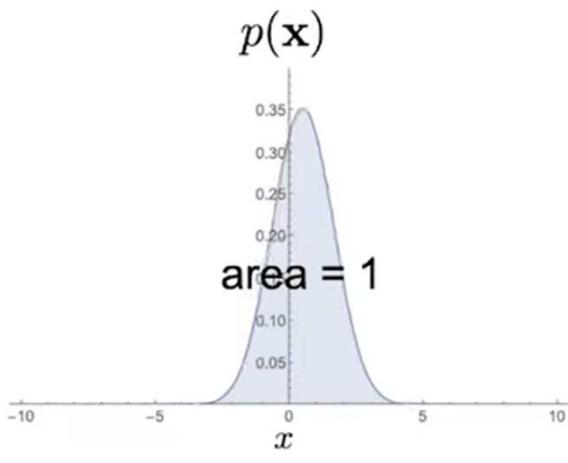
Score functions bypass the normalizing constant



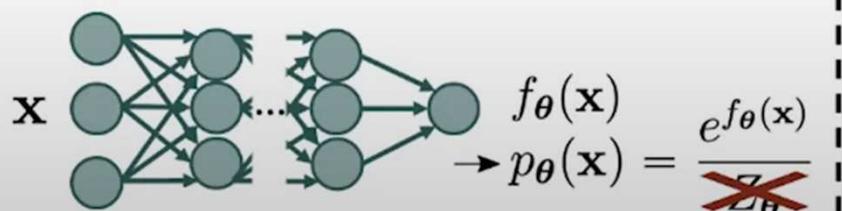
[Flexible models]

Flexible models

Score functions bypass the normalizing constant



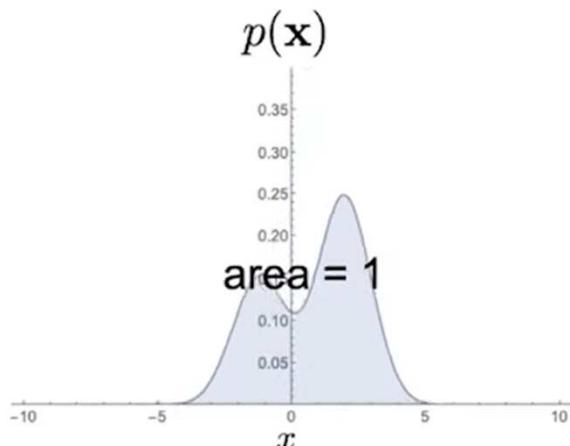
- But in contrast, if we **model the same distribution through the score functions**, then, as the animation shows, **there is no such normalization restriction**.
- In fact, if we **compute a score function** there is no such **normalization restriction**.



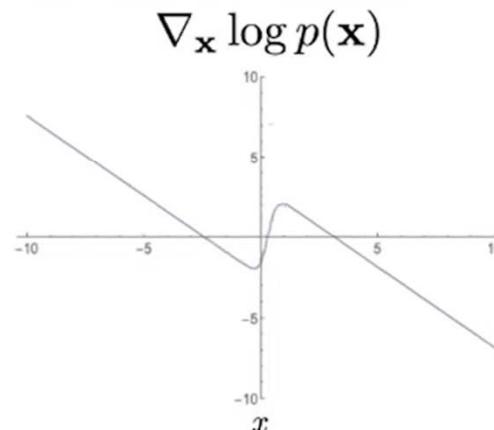
[Flexible models]

Flexible models

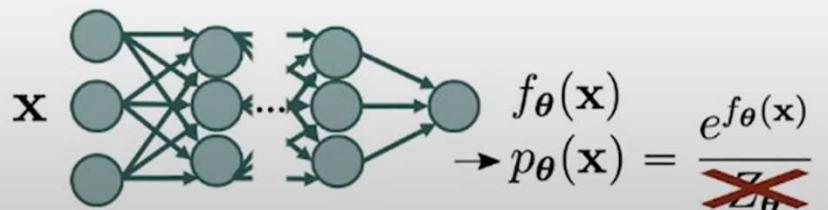
Score functions bypass the normalizing constant



Probability density function



Score function



$$\nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}) = \nabla_{\mathbf{x}} f_{\theta}(\mathbf{x}) - \boxed{\nabla_{\mathbf{x}} \log Z_{\theta}},$$

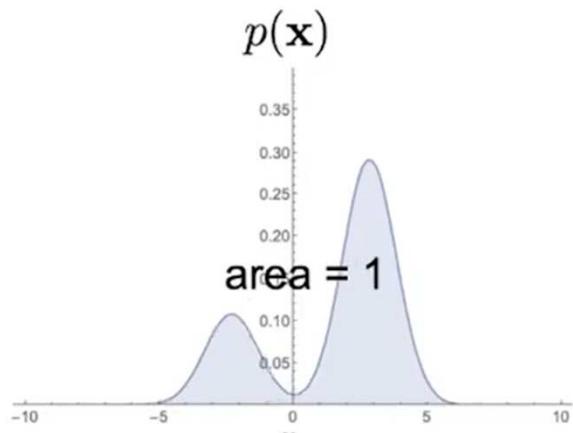
\downarrow
0

- If we compute a score function for the neural network on the left side, we notice that the **score function is the difference of two terms**.
- Only the second term involves the **intractable normalizing constant**. But the second term is **always 0** because the **gradient of any constant** is always 0.

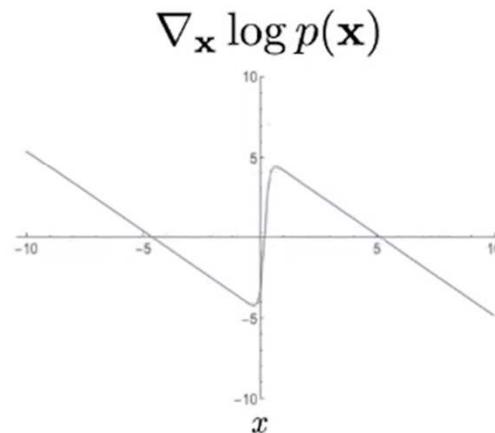
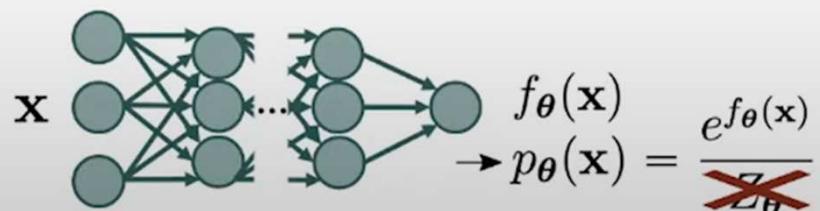
[Flexible models]

Flexible models

Score functions bypass the normalizing constant



Probability density function



Score function

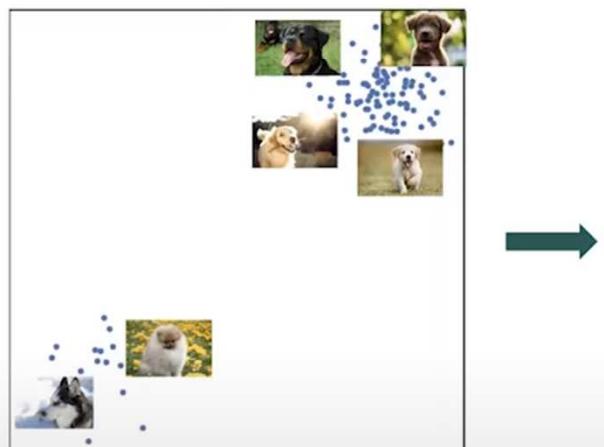
$$\begin{aligned}\nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}) &= \nabla_{\mathbf{x}} f_{\theta}(\mathbf{x}) - \boxed{\nabla_{\mathbf{x}} \log Z_{\theta}} \\ &= \nabla_{\mathbf{x}} f_{\theta}(\mathbf{x}) \\ &= s_{\theta}(\mathbf{x})\end{aligned}$$

- As a result, **the score function equals the gradient of the deep neural network**.
- And as you might know, those gradients of deep neural networks can be easily computed with automatic differentiation or with backpropagation. So this is a very efficient operation.
- And from the [INAUDIBLE], we use a simple **s_theta** to denote such a **deep neural network model for the score function**, and we call it our **score model**.

[Flexible models]

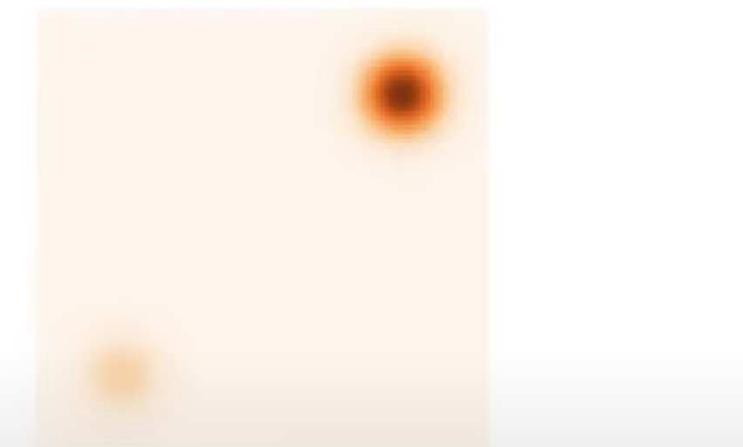
Flexible models

Score models can be estimated from data



Training data

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



Probability density function

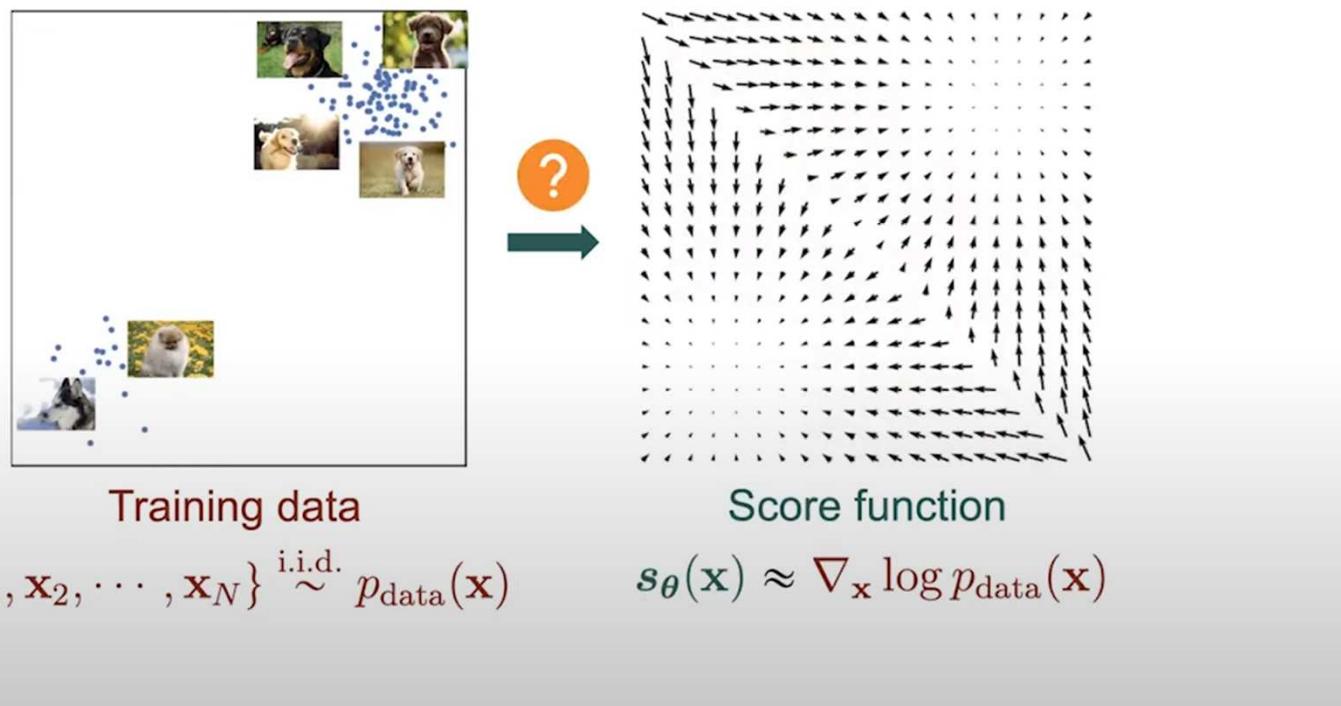
$$p_{\theta}(\mathbf{x}) \approx p_{\text{data}}(\mathbf{x})$$

- Suppose we have collected a large trained data set, and again we use $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ to denote each point in this data set.
- We assume the underlying data density is given by P_{data} .
- With our knowledge in statistics, we know that we can **train our properly normalized statistical model to estimate the underlying data density** using methods such as **maximum likelihood**.

[Flexible models]

Flexible models

Score models can be estimated from data



- But because we want to work with **score functions**, we want to develop a **similar approach** that can allow us to **train our score model to estimate the underlying score function** from a **limited set of training data points**.
- And we have formulated this problem **score estimation**.

[Flexible models]

Flexible models

Score models can be estimated from data

Given: $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\} \stackrel{\text{i.i.d.}}{\sim} p_{\text{data}}(\mathbf{x})$

Goal: $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$

Score Model: $s_{\theta}(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^d \approx \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$

Objective: How to compare two vector fields of scores?

- Mathematically, we are given a bunch of data points which are assumed to be IID sampled from the data distribution P_{data} .
- Our goal is to estimate this score function of the data density.
- We are given a **score model**. This is assumed to be a **deep neural network model** that maps the deep dimensional input to a deep dimensional output, and we hope to **train this score model** such that it **approximates our ground truth score function of the data distribution**.
- So how can we train this score model to be close to our ground truth data score function?
- Well, we need to minimize a certain objective. **This objective has to compare two vector fields of score functions.**

[Flexible models]

Flexible models

Score models can be estimated from data

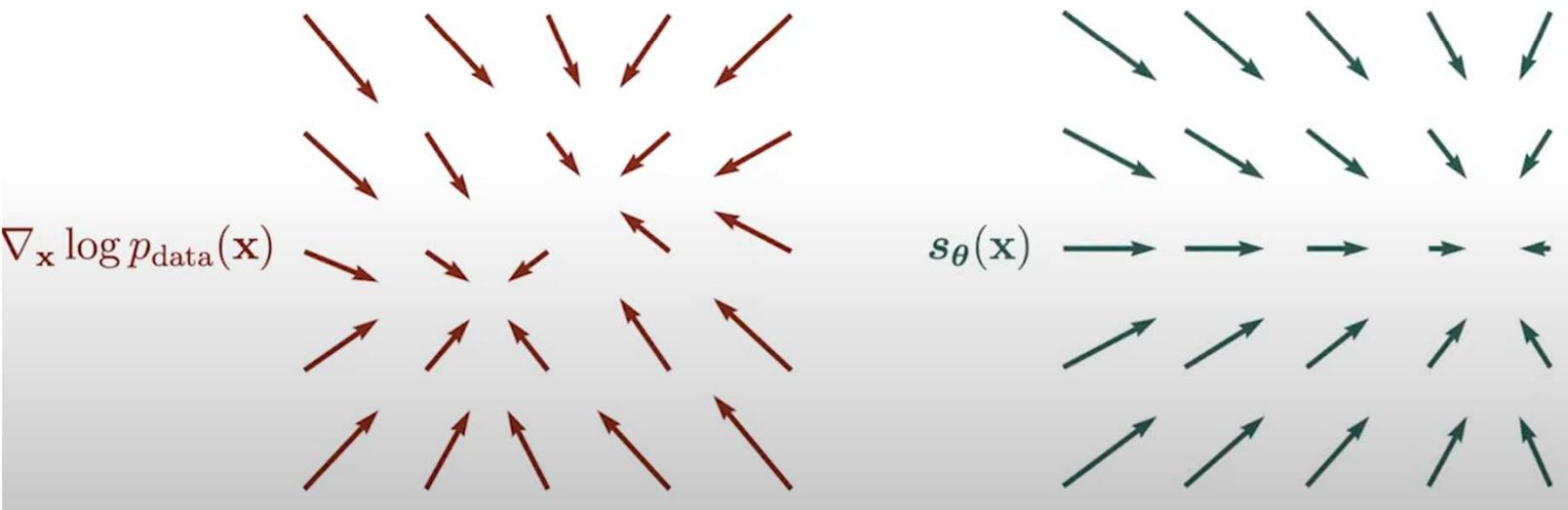
Given: $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\} \stackrel{\text{i.i.d.}}{\sim} p_{\text{data}}(\mathbf{x})$

Goal: $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$

Score Model: $s_{\theta}(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^d \approx \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$

Objective: How to compare two vector fields of scores?

- Here, one vector field is **the ground truth data score function**.
- The other vector field is **predicted by our score model**.
- How can we compare the difference?
Let's recall that those two vector fields actually lie in the same space.



[Flexible models]

Flexible models

Score models can be estimated from data

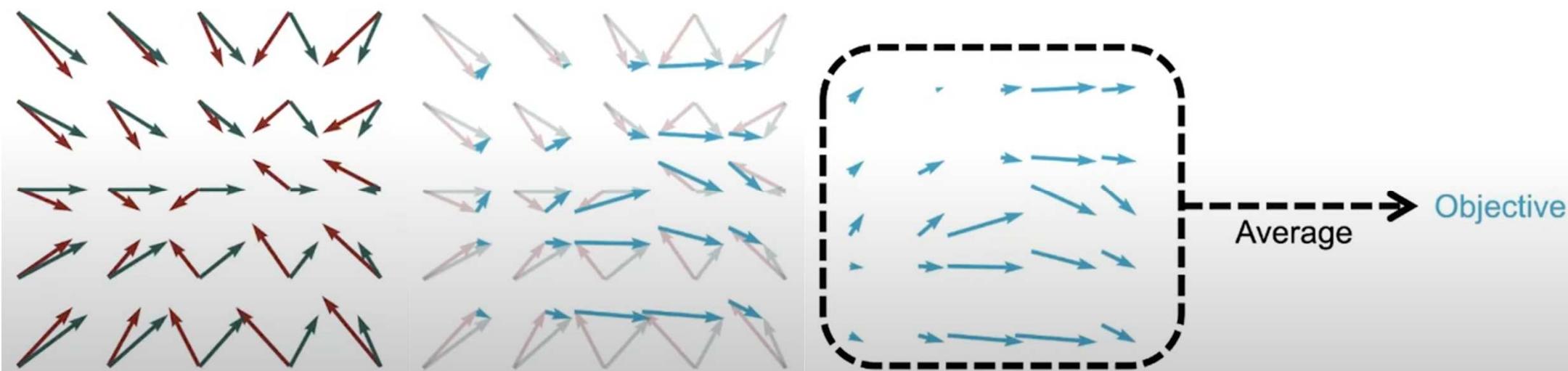
Given: $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\} \stackrel{\text{i.i.d.}}{\sim} p_{\text{data}}(\mathbf{x})$

Goal: $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$

Score Model: $s_{\theta}(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^d \approx \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$

Objective: How to compare two vector fields of scores?

- So we might be able to compute the difference vectors between those pairs of vectors from the original vector fields.
- And then we can [INAUDIBLE] over the densities of those difference vectors to form a **single scalar-valued objective**.



[Flexible models]

Flexible models

Score models can be estimated from data

Given: $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\} \stackrel{\text{i.i.d.}}{\sim} p_{\text{data}}(\mathbf{x})$

Goal: $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$

Score Model: $s_{\theta}(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^d \approx \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$

Objective: How to compare two vector fields of scores?

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

(Fisher divergence)

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

(Fisher divergence)

- Mathematically, we can capture this intuition with the Fisher divergence objective.
- **Fisher divergence** is essentially an expected squared Euclidean distance between the **data score** and the **model score averaged over samples from the data distribution**.
- However, Fisher divergence cannot be directly computed because **we don't know the ground truth value of the data score function**.
- But luckily there is a way to address this challenge,

[Flexible models]

Flexible models

Score models can be estimated from data

Given: $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\} \stackrel{\text{i.i.d.}}{\sim} p_{\text{data}}(\mathbf{x})$

Goal: $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$

Score Model: $s_{\theta}(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^d \approx \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$

Objective: How to compare two vector fields of scores?

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

(Fisher divergence)

Integration by parts
(Gauss's theorem)

Score Matching [Hyvärinen 2005]:

$$\begin{aligned} & \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\frac{1}{2} \|s_{\theta}(\mathbf{x})\|_2^2 + \text{trace} \left(\underbrace{\nabla_{\mathbf{x}} s_{\theta}(\mathbf{x})}_{\text{Jacobian of } s_{\theta}(\mathbf{x})} \right) \right] \\ & \approx \frac{1}{N} \sum_{i=1}^N \left[\frac{1}{2} \|s_{\theta}(\mathbf{x}_i)\|_2^2 + \text{trace}(\nabla_{\mathbf{x}} s_{\theta}(\mathbf{x}_i)) \right] \end{aligned}$$

trace 함수 : 대각선 요소의 합

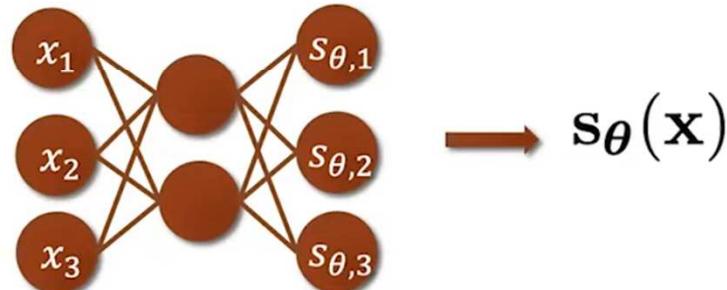
- This method is called score matching. Score matching uses integration by parts of Gauss's theorem to convert Fisher divergence into the following equivalent objective.
- The objective at the bottom is equivalent to Fisher divergence up to a constant.
- But since constants do not affect optimization, their score matching objective defines the same optimum as the Fisher divergence.
- In a score matching objective, there is no dependency on the score function of the data distribution anymore.
- Moreover, the expectation in score matching can be efficiently approximated using the empirical mean over the training data set.
- So, so far, so good.
- However, the score matching objective

[Flexible models]

Flexible models

Score Matching is not scalable

- Deep score models



- Compute $\|s_{\theta}(\mathbf{x})\|_2^2$ and $\text{trace}(\nabla_{\mathbf{x}} s_{\theta}(\mathbf{x}))$

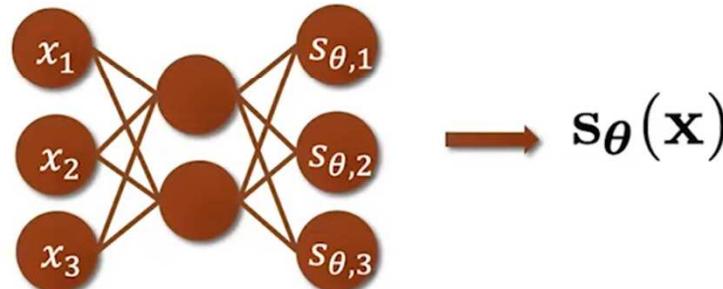
- However, **the score matching objective is not scalable to compute**, especially when you want to **use deep neural networks** to model high-dimensional data points.
- Let's suppose **our score function is parameterized by our deep neural network**, which we call **deep score models**.
- In order to use score matching we have to **compute** two terms, where one term is **the squared Euclidean norm of the score model output**. The second term is the choice of the **Jacobian of the score model**.

[Flexible models]

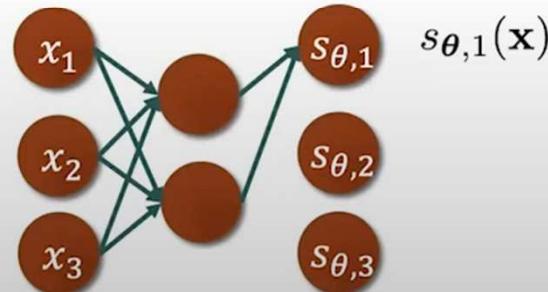
Flexible models

Score Matching is not scalable

- Deep score models



- Compute $\|s_{\theta}(\mathbf{x})\|_2^2$ and $\text{trace}(\nabla_{\mathbf{x}}s_{\theta}(\mathbf{x}))$



- For the first term, it is super simple to compute and very efficient because we just need [INAUDIBLE] forward propagation to get the output. Then we can compute the squared L2 norm very efficiently.
- For the second term things become a little bit more complicated because we **need one forward propagation to compute the first element of the score function output**, and we need a **backpropagation** to compute the first element on the diagonal of this Jacobian.

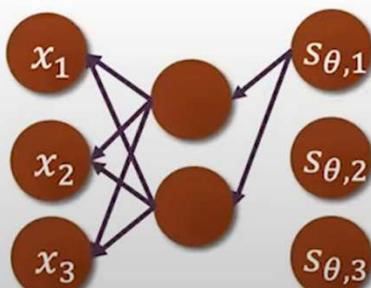
[Flexible models]

Flexible models

Score Matching is not scalable

- Compute $\|s_{\theta}(\mathbf{x})\|_2^2$ and $\text{trace}(\nabla_{\mathbf{x}} s_{\theta}(\mathbf{x}))$

$$\frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_1}$$

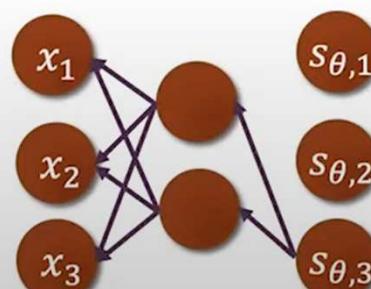


$$\nabla_{\mathbf{x}} s_{\theta}(\mathbf{x}) = \begin{pmatrix} \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_3} \\ \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_3} \\ \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_3} \end{pmatrix}$$

$$\frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_1}$$

$$\frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_2}$$

$$\frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_3}$$



$$\nabla_{\mathbf{x}} s_{\theta}(\mathbf{x}) = \begin{pmatrix} \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_3} \\ \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_3} \\ \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_3} \end{pmatrix}$$

- This procedure has to be repeated multiple times until we have recovered all diagonal elements on the Jacobian. Then we can sum over the diagonal elements to get the trace.
- This whole procedure requires a lot of backpropagations.

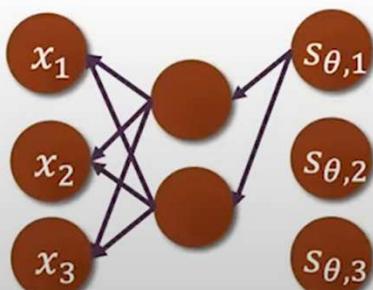
[Flexible models]

Flexible models

Score Matching is not scalable

- Compute $\|s_\theta(\mathbf{x})\|_2^2$ and $\text{trace}(\nabla_{\mathbf{x}} s_\theta(\mathbf{x}))$

$$\frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_1}$$

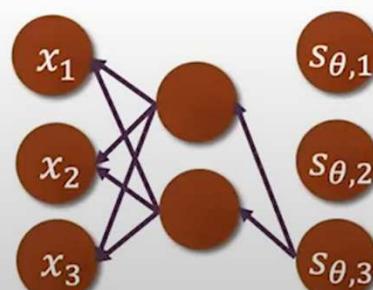


$$\nabla_{\mathbf{x}} s_\theta(\mathbf{x}) = \begin{pmatrix} \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_3} \\ \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_3} \\ \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_3} \end{pmatrix}$$

$$\frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_1}$$

$$\frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_2}$$

$$\frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_3}$$



$O(\#\text{dimensions of } \mathbf{x})$

Backprops!

$$\nabla_{\mathbf{x}} s_\theta(\mathbf{x}) = \begin{pmatrix} \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_3} \\ \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_3} \\ \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_3} \end{pmatrix}$$

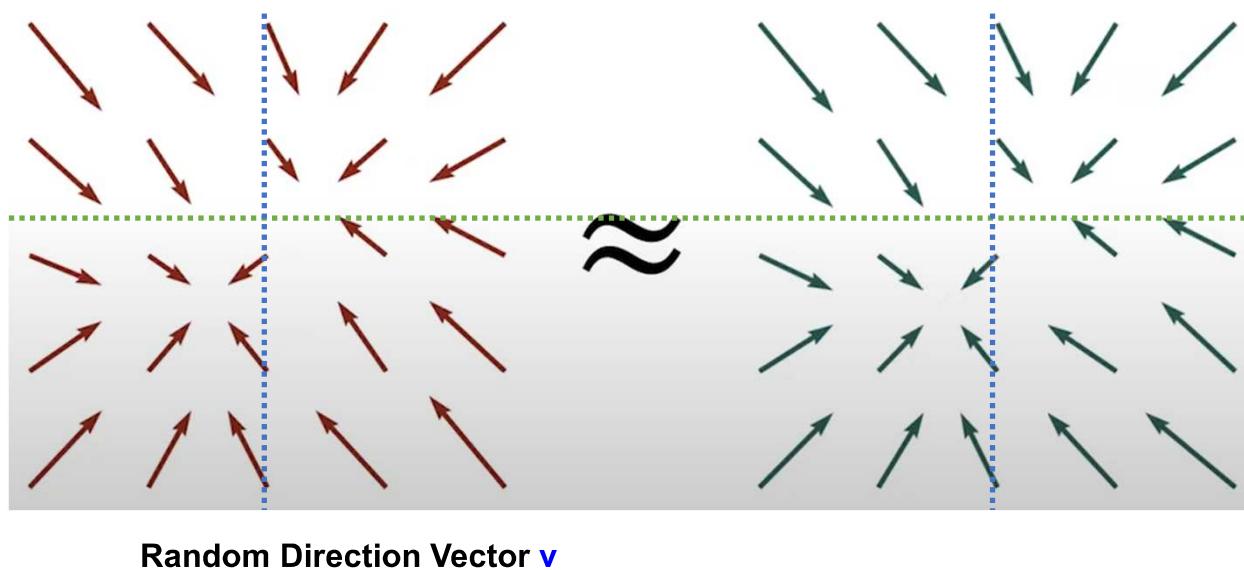
- This whole procedure requires a lot of backpropagations.
- And the number of backpropagations actually is proportional to the dimensionality of our data point.
- For modeling high-dimension data like images, we might need to deal with medians and dimensions.
- And this means score matching in its naive form is not scalable.

[Score Model – Sliced Score Matching]

Flexible models

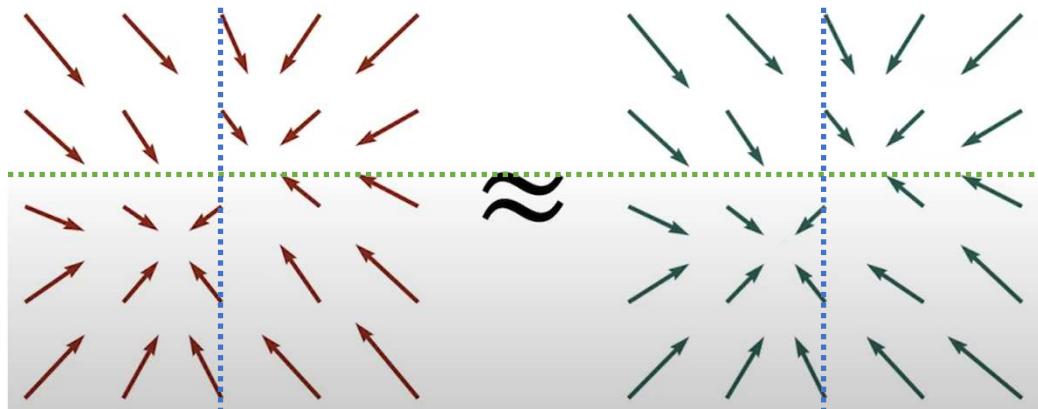
Sliced score matching

- **Intuition:** one dimensional problems should be easier
- **Idea:** project onto random directions

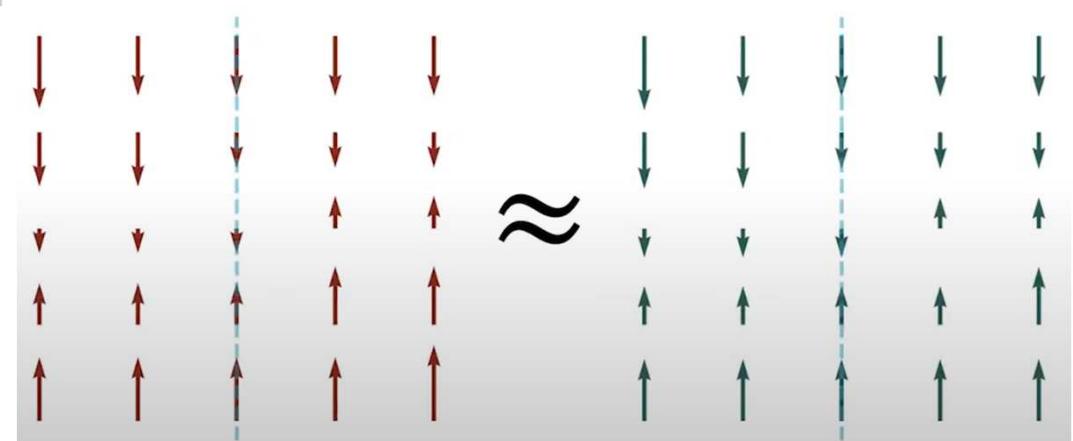
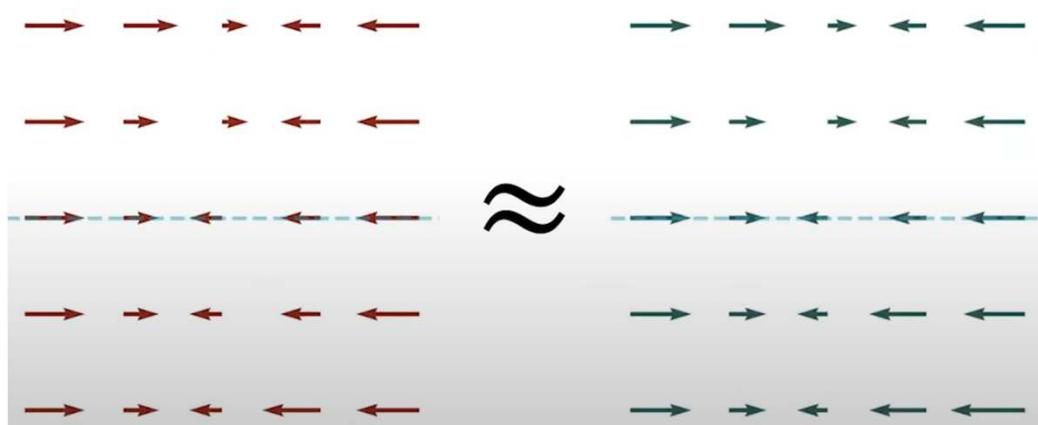


- To address this challenge, we actually propose a more efficient variant of score matching which we term **sliced score matching**.
- The basic intuition is that **one-dimensional problem** should be much easier to solve than those high-dimensional problems.
- How can we convert a high-dimensional problem to a one-dimensional problem?
- Well, we can leverage **random projections**. We **project the high-dimensional vector fields to run directions**. Then we get one-dimensional scalar fields.
- Suppose those two high-dimensional vector fields are close to each other. Then we can **project them along random one-dimensional directions**. This gives us one-dimensional scalar fields. Those scalar fields will also be close to each other.

[Score Model – Sliced Score Matching]



Random
Direction
Vector v



- [Random Project from mD to 1D]
- We **project the high-dimensional vector fields to run directions**. Then we get one-dimensional scalar fields.
- Suppose those two high-dimensional vector fields are close to each other. Then we can **project them along random one-dimensional directions**. This gives us one-dimensional scalar fields. Those scalar fields will also be close to each other.

[Score Model – Sliced Score Matching]

Flexible models

Sliced score matching

- **Intuition:** one dimensional problems should be easier
- **Idea:** project onto random directions
- **Randomized objective: Sliced Fisher Divergence**

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

- Integration by parts → **Sliced Score Matching:**

$$\mathbb{E}_{p_{\mathbf{v}}} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\boxed{\mathbf{v}^T \nabla_{\mathbf{x}} \mathbf{s}_{\theta}(\mathbf{x}) \mathbf{v}} + \frac{1}{2} (\mathbf{v}^T \mathbf{s}_{\theta}(\mathbf{x}))^2 \right]$$

↓
Scalable!

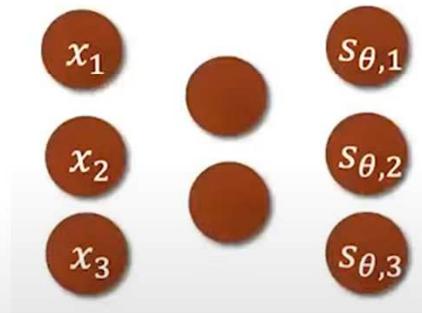
- We can capture this intuition with the concept of sliced Fisher divergence.
- Here \mathbf{v} denotes **the projection direction**. It is a vector. And $p_{\mathbf{v}}$ denotes **the distribution of those projection directions**.
- We compute the inner product [INAUDIBLE] \mathbf{v} and those two school functions and measure the resulting difference between them.
- And we can again leverage integration by parts to eliminate the dependency on the ground truth data score. This gives us **the sliced score matching objective**.
- And in sliced score matching, there is **no trace of a Jacobian** anymore. Instead, we have **vector Jacobian vector product**. This term is much more **scalable** to compute.

[Score Model – Sliced Score Matching]

Flexible models

Computing Jacobian-vector products is scalable

$$\mathbf{v}^\top \nabla_{\mathbf{x}} \mathbf{s}_{\theta}(\mathbf{x}) \mathbf{v} = \mathbf{v}^\top \nabla_{\mathbf{x}} (\mathbf{v}^\top \mathbf{s}_{\theta}(\mathbf{x}))$$



- This is actually not hard to see because we can rewrite the vector Jacobian vector product as an alternative for on the right-hand side.
- This just requires us to swap the location of \mathbf{v} and \mathbf{S}_{θ} within the gradient operator.

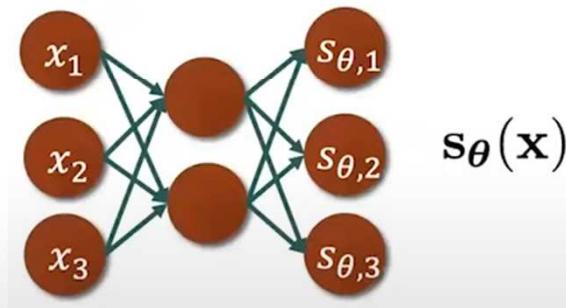
- So now I will show you how to compute this vector Jacobian vector product very efficiently.

[Score Model – Sliced Score Matching]

Flexible models

Computing Jacobian-vector products is scalable

$$\mathbf{v}^\top \nabla_{\mathbf{x}} \mathbf{s}_{\theta}(\mathbf{x}) \mathbf{v} = \mathbf{v}^\top \nabla_{\mathbf{x}} (\mathbf{v}^\top \boxed{\mathbf{s}_{\theta}(\mathbf{x})})$$



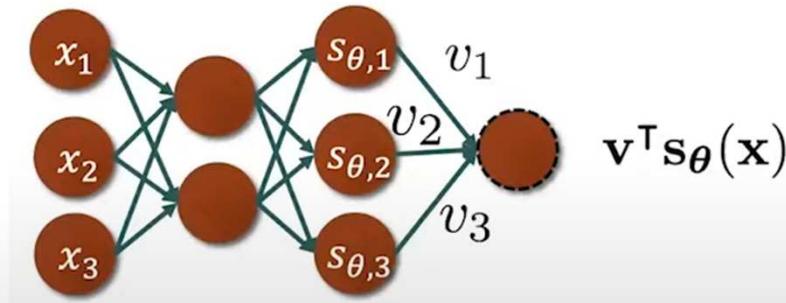
- First, we just need one forward propagation to get the output of s_{θ} ,

[Score Model – Sliced Score Matching]

Flexible models

Computing Jacobian-vector products is scalable

$$\mathbf{v}^\top \nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x}) \mathbf{v} = \mathbf{v}^\top \nabla_{\mathbf{x}} (\mathbf{v}^\top \mathbf{s}_\theta(\mathbf{x}))$$



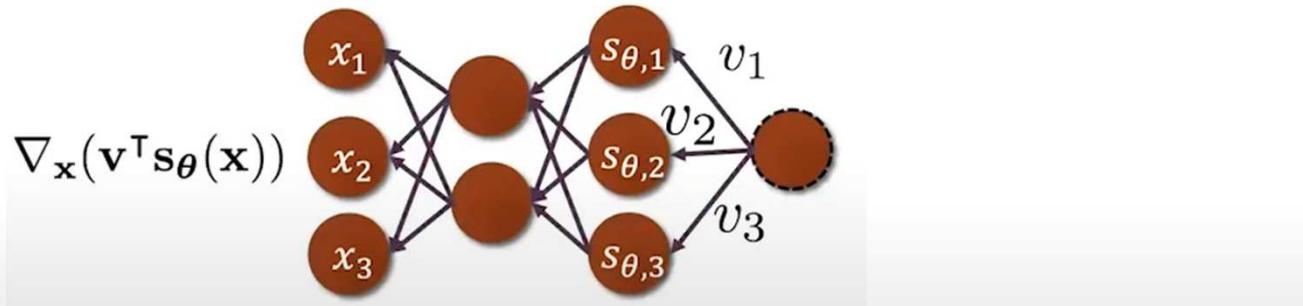
- and then we can directly compute the inner product between \mathbf{v} and $\mathbf{s}_\theta(\mathbf{x})$.
- So this amounts to adding one additional neuron to the computational graph.
- And next, we can compute that gradient

[Score Model – Sliced Score Matching]

Flexible models

Computing Jacobian-vector products is scalable

$$\mathbf{v}^\top \nabla_{\mathbf{x}} \mathbf{s}_{\theta}(\mathbf{x}) \mathbf{v} = \mathbf{v}^\top \nabla_{\mathbf{x}} (\mathbf{v}^\top \mathbf{s}_{\theta}(\mathbf{x}))$$



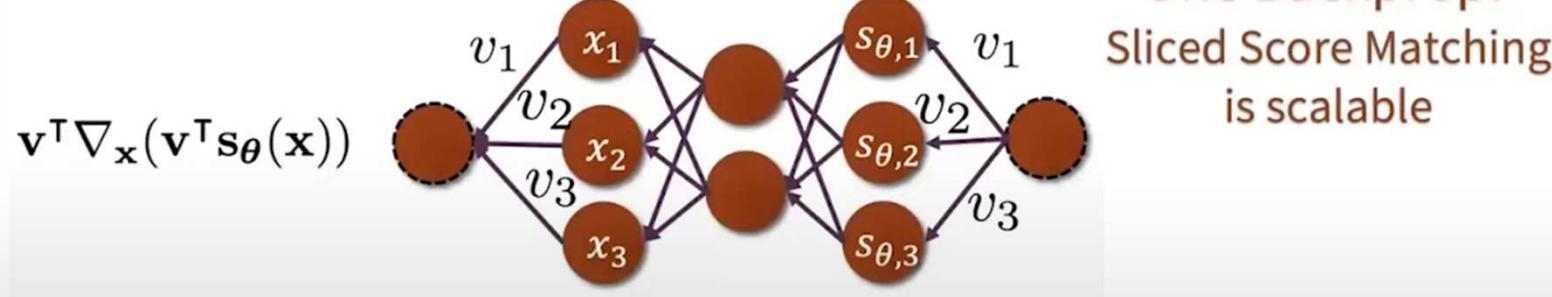
- And next, we can compute that gradient by doing one backpropagation.

[Score Model – Sliced Score Matching]

Flexible models

Computing Jacobian-vector products is scalable

$$\mathbf{v}^\top \nabla_{\mathbf{x}} \mathbf{s}_{\theta}(\mathbf{x}) \mathbf{v} = \boxed{\mathbf{v}^\top \nabla_{\mathbf{x}} (\mathbf{v}^\top \mathbf{s}_{\theta}(\mathbf{x}))}$$



Song*, Garg*, Shi, Ermon. "Sliced Score Matching: A Scalable Approach to Density and Score Estimation." UAI 2019.

- And as the last step, we just need to computer the inner products in the [INAUDIBLE] gradient.
- So the whole procedure only requires **one backpropagation**, which is much more **efficient** compared to the vanilla form of score matching.

[Score Model – Sliced Score Matching]

Flexible models

Sliced score matching

- Sample a minibatch of datapoints $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\} \sim p_{\text{data}}(\mathbf{x})$
- Sample a minibatch of projection directions $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\} \sim p_{\mathbf{v}}$
- Estimate the sliced score matching loss with empirical means

$$\frac{1}{n} \sum_{i=1}^n \left[\mathbf{v}_i^\top \nabla_{\mathbf{x}} s_\theta(\mathbf{x}_i) \mathbf{v}_i + \frac{1}{2} (\mathbf{v}_i^\top s_\theta(\mathbf{x}_i))^2 \right]$$

- This is how sliced score matching works in practice.
- We just sample a minibatch of data points from our data set.
- And for each data point, we sample one single projection direction from our distribution of $p_{\mathbf{v}}$.
- And then we form the empirical estimate of the sliced score matching training objective using the empirical mean over our sample data points and those projection directions.

[Score Model – Sliced Score Matching]

Flexible models

Sliced score matching

- Sample a minibatch of datapoints $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\} \sim p_{\text{data}}(\mathbf{x})$
- Sample a minibatch of projection directions $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\} \sim p_{\mathbf{v}}$
- Estimate the sliced score matching loss with empirical means

$$\frac{1}{n} \sum_{i=1}^n \left[\mathbf{v}_i^\top \nabla_{\mathbf{x}} s_\theta(\mathbf{x}_i) \mathbf{v}_i + \frac{1}{2} (\mathbf{v}_i^\top s_\theta(\mathbf{x}_i))^2 \right]$$

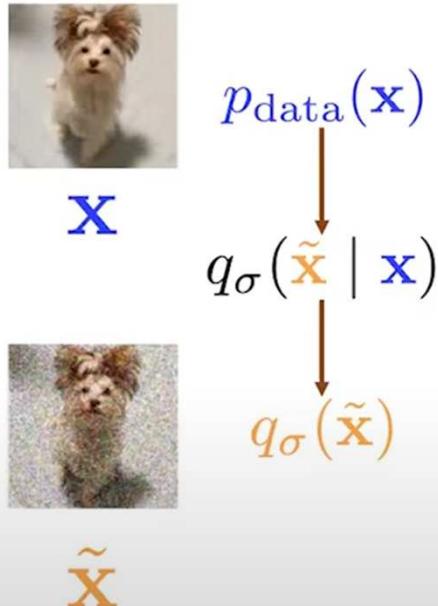
- The projection distribution is typically Gaussian or Rademacher
- Stochastic gradient descent
- Can use more projections per datapoint to boost performance

- So the projection distribution $p_{\mathbf{v}}$ is typically a simple standard Gaussian distribution or sometimes better you can use Rademacher distributions, which are uniform distributed sine vectors.
- And then we can use stochastic gradient descent to minimize our empirical objective for sliced score matching.
- And if you want a better performance or equivalently lower variance of our training objective, you could potentially use more projections per data point.
- So that concludes the discussion of sliced score matching.

[Score Model – Denoising Score Matching]

Flexible models

Denoising score matching

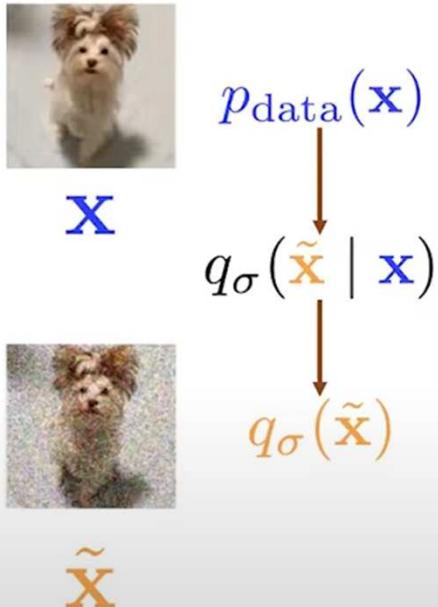


- There exists another approach called **denoising score matching** that can also bypass the computational challenge of vanilla score matching.
- The idea of denoising score matching is to **add additional noise to the data point** to help us compute the choice of a Jacobian term.
- To perform to denoising score matching, we need to **design a perturbation kernel** which we denote as **q_{sigma}** . So $\tilde{\mathbf{x}}$ denotes the perturbed data point, and \mathbf{x} denotes the original noise-free data point. **Sigma** can typically be a Gaussian distribution with means x and a standard deviation σ .
- So after converting this perturbation kernel with our original data distribution, we get a **noisy data distribution to sigma of x_{tilde}** .

[Score Model – Denoising Score Matching]

Flexible models

Denoising score matching



- Matching the score of a noise-perturbed distribution

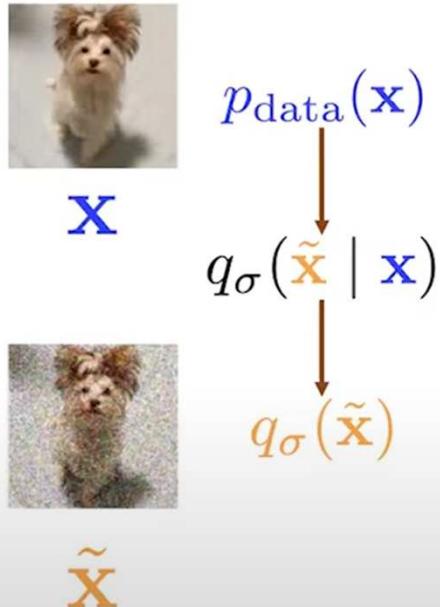
$$\frac{1}{2} \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}})} [\|\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}) - s_\theta(\tilde{\mathbf{x}})\|_2^2]$$

- The key idea of denoising score matching is to **estimate the scope function of this noise data density** instead of the score function of the original data density.
- Of course, when sigma is very small, you can approximately view the score function of the noise density as the equivalent to the scope function of the noise-free density.
- The magic happens when you estimate this score function of a noisy distribution.
- You can use some arithmetic derivation to write down an equivalent form to the denoising score matching objective, which I give at the bottom of this slide.

[Score Model – Denoising Score Matching]

Flexible models

Denoising score matching



- Matching the score of a noise-perturbed distribution

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

- Denoising score matching (Vincent 2011)

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

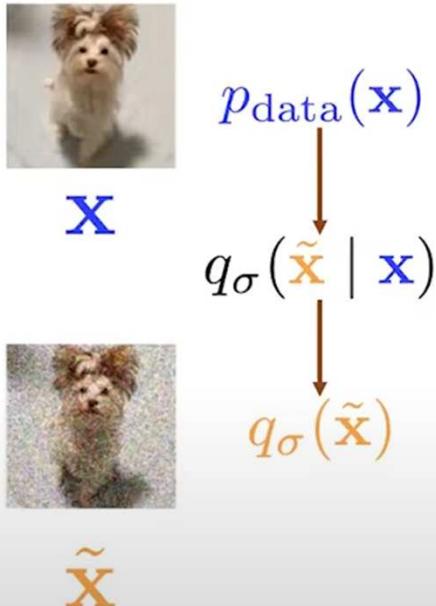
Scalable

- So you can use some arithmetic derivation to write down an equivalent form to the denoising score matching objective, which I give at the bottom of this slide.
- In this new form, what we need to compute is just **the gradient of the perturbation kernel**.
- Because we designed the perturbation kernel by hand, usually this perturbation kernel is a **fully tractable distribution**. Computing this gradient is very efficient, and it can be done analytically.

[Score Model – Denoising Score Matching]

Flexible models

Denoising score matching



- Matching the score of a noise-perturbed distribution

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

- Denoising score matching (Vincent 2011)

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

- **Cannot estimate scores of noise-free distributions!**

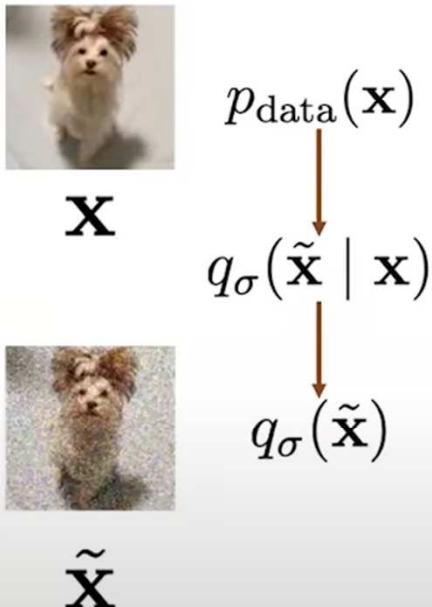
- But the [INAUDIBLE] known as score matching is that **since it requires adding noise to data points, it cannot estimate scores of the noise-free distributions.**
- And what's worse, **when you're trying to lower the magnitude of the noise, the variance of denoising score matching objective actually becomes bigger and bigger and eventually explodes.**
- **There is no easy way to use denoising score matching for noise-free score estimation.**

[Score Model – Denoising Score Matching]

Flexible models

Denoising score matching

- Denoising score matching (Vincent 2011): Matching the score of a noise-perturbed distribution



$$\begin{aligned}
 & - \int q_\sigma(\tilde{\mathbf{x}}) \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}})^\top \mathbf{s}_\theta(\tilde{\mathbf{x}}) d\tilde{\mathbf{x}} \\
 &= - \int q_\sigma(\tilde{\mathbf{x}}) \frac{1}{q_\sigma(\tilde{\mathbf{x}})} \nabla_{\tilde{\mathbf{x}}} q_\sigma(\tilde{\mathbf{x}})^\top \mathbf{s}_\theta(\tilde{\mathbf{x}}) d\tilde{\mathbf{x}} \\
 &= - \int \nabla_{\tilde{\mathbf{x}}} q_\sigma(\tilde{\mathbf{x}})^\top \mathbf{s}_\theta(\tilde{\mathbf{x}}) d\tilde{\mathbf{x}} \\
 &= - \int \nabla_{\tilde{\mathbf{x}}} \left(\int p_{\text{data}}(\mathbf{x}) q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) d\mathbf{x} \right)^\top \mathbf{s}_\theta(\tilde{\mathbf{x}}) d\tilde{\mathbf{x}} \\
 &= - \int \left(\int p_{\text{data}}(\mathbf{x}) \nabla_{\tilde{\mathbf{x}}} q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) d\mathbf{x} \right)^\top \mathbf{s}_\theta(\tilde{\mathbf{x}}) d\tilde{\mathbf{x}} \\
 &= - \int \left(\int p_{\text{data}}(\mathbf{x}) q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) d\mathbf{x} \right)^\top \mathbf{s}_\theta(\tilde{\mathbf{x}}) d\tilde{\mathbf{x}} \\
 &= - \iint p_{\text{data}}(\mathbf{x}) q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})^\top \mathbf{s}_\theta(\tilde{\mathbf{x}}) d\mathbf{x} d\tilde{\mathbf{x}} \\
 &= - E_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x}), \tilde{\mathbf{x}} \sim q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})} [\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})^\top \mathbf{s}_\theta(\tilde{\mathbf{x}})]
 \end{aligned}$$

- So we can actually derive the formula with denoising score matching very easily.
- But I guess due to time reasons, we have skip this part.
- And it's not hard to find this derivation from the original paper of denoising score matching.

[Score Model – Denoising Score Matching]

Flexible models

Denoising score matching

- Sample a minibatch of datapoints $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\} \sim p_{\text{data}}(\mathbf{x})$
- Sample a minibatch of perturbed datapoints $\{\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \dots, \tilde{\mathbf{x}}_n\} \sim q_\sigma(\tilde{\mathbf{x}})$
 $\tilde{\mathbf{x}}_i \sim q_\sigma(\tilde{\mathbf{x}}_i | \mathbf{x}_i)$
- Estimate the denoising score matching loss with empirical means

$$\frac{1}{2n} \sum_{i=1}^n [\|s_\theta(\tilde{\mathbf{x}}_i) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}_i | \mathbf{x}_i)\|_2^2]$$

- As a conclusion, when you want to apply denoising score matching, you follow a similar procedure as sliced score matching.
- First of all, you sample a minibatch of data points from the data density.
- And then you sample a minibatch of perturbed data points.
- So usually for one data point, you sample a single perturbed data point by adding the additional amount of noise to the chosen data point.
- And then you can form the empirical estimation of the denoising score matching loss by approximating the expectation using empirical means.

[Score Model – Denoising Score Matching]

Flexible models

Denoising score matching

- Sample a minibatch of datapoints $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\} \sim p_{\text{data}}(\mathbf{x})$
- Sample a minibatch of perturbed datapoints $\{\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \dots, \tilde{\mathbf{x}}_n\} \sim q_\sigma(\tilde{\mathbf{x}})$
 $\tilde{\mathbf{x}}_i \sim q_\sigma(\tilde{\mathbf{x}}_i | \mathbf{x}_i)$
- Estimate the denoising score matching loss with empirical means

$$\frac{1}{2n} \sum_{i=1}^n [\|s_\theta(\tilde{\mathbf{x}}_i) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}_i | \mathbf{x}_i)\|_2^2]$$

- If Gaussian perturbation

$$\frac{1}{2n} \sum_{i=1}^n \left[\|s_\theta(\tilde{\mathbf{x}}_i) + \frac{\tilde{\mathbf{x}}_i - \mathbf{x}_i}{\sigma^2}\|_2^2 \right]$$

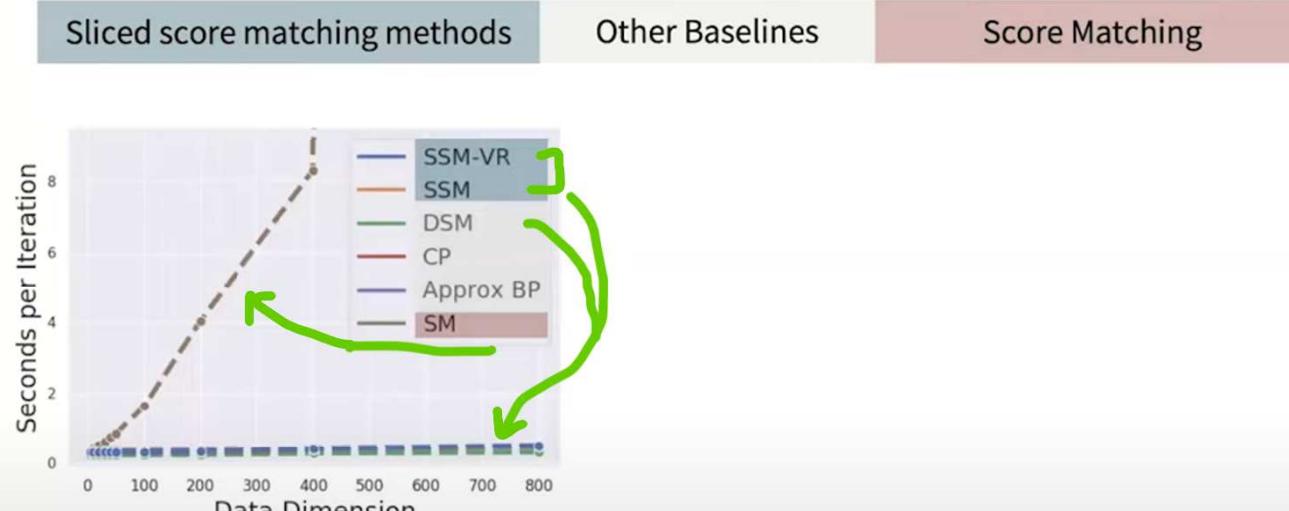
- Stochastic gradient descent
- Need to choose a very small σ !

- In the special case of Gaussian perturbations, you can further simplify the denoising score matching loss function.
- Then you can just apply stochastic gradient descent to minimize this objective function to train your score model.
- In practice, if you want it to work well for estimating score functions of noise-free data densities, you need to choose a very small sigma.
- But as I said, when sigma is very small, the variance of this objective will explode. So there is a tradeoff, and you need to find the sweet spot.

[Score Model]

Flexible models

Results: Sliced Score Matching for EBMs



Efficiency

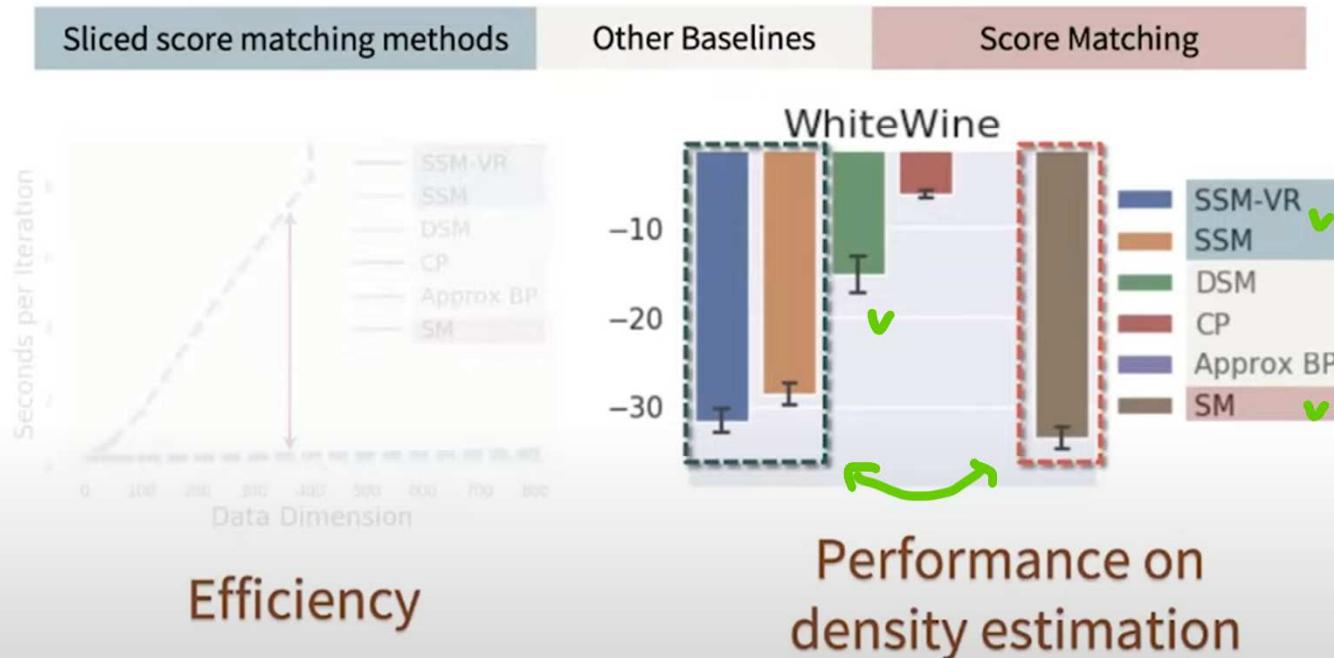
Song*, Garg*, Shi, Ermon. "Sliced Score Matching: A Scalable Approach to Density and Score Estimation." UAI 2019.

- Some experimental results.
- We first want to compare the computational efficiency of **sliced score matching (SSM)** and also **denoising score matching (DSM)** versus the vanilla version **score matching (SM)**.
- We consider the problem of training **energy-based models (EBM)**, or, equivalently, we are considering the problem of training score functions from noise-free data.
- The first figure shows how much time is needed to perform each iteration of various algorithms as a function of data dimensionality.
- When data dimensionality increases, all those algorithms will take more time to perform one training iteration.
- Clearly, score matching (SM) scales the worst. In contrast, Sliced Score Matching (SSM) and Denoising Score Matching (DSM), they scale much more preferably compared to score matching.

[Score Model]

Flexible models

Results: Sliced Score Matching for EBMs



Song*, Garg*, Shi, Ermon. "Sliced Score Matching: A Scalable Approach to Density and Score Estimation." UAI 2019.

- And in terms of the actual **performance of score estimation**, we report the results on the left figure. The performance is better when the number is lower.
- Comparing sliced score matching and score matching, you can see that even though sliced score matching takes much less time to compute, they can still obtain more or less comparable performance as score matching in terms of score estimation.
- Really, we gain a lot of computational boost at a small cost of the accuracy in score estimation.
- For Denoising Score Matching (DSM) because you have to inject noise to the data point, the performance in score estimation is not as good as sliced score matching when you want to estimate the score function of a clean data points.
- So everything is where we expected.

[Flexible Models]

Score-based generative modeling: outline



Flexible models

- Bypass the normalizing constant
- Principled statistical methods

[Song et al. UAI 2019 oral]

Improved generation

- Higher sample quality than GANs
- Controllable generation

[Song & Ermon. NeurIPS 2019 oral]
[Song & Ermon. NeurIPS 2020]
[Song et al. ICLR 2021 oral]
(Outstanding Paper Award)
[Song et al. ICLR 2022]

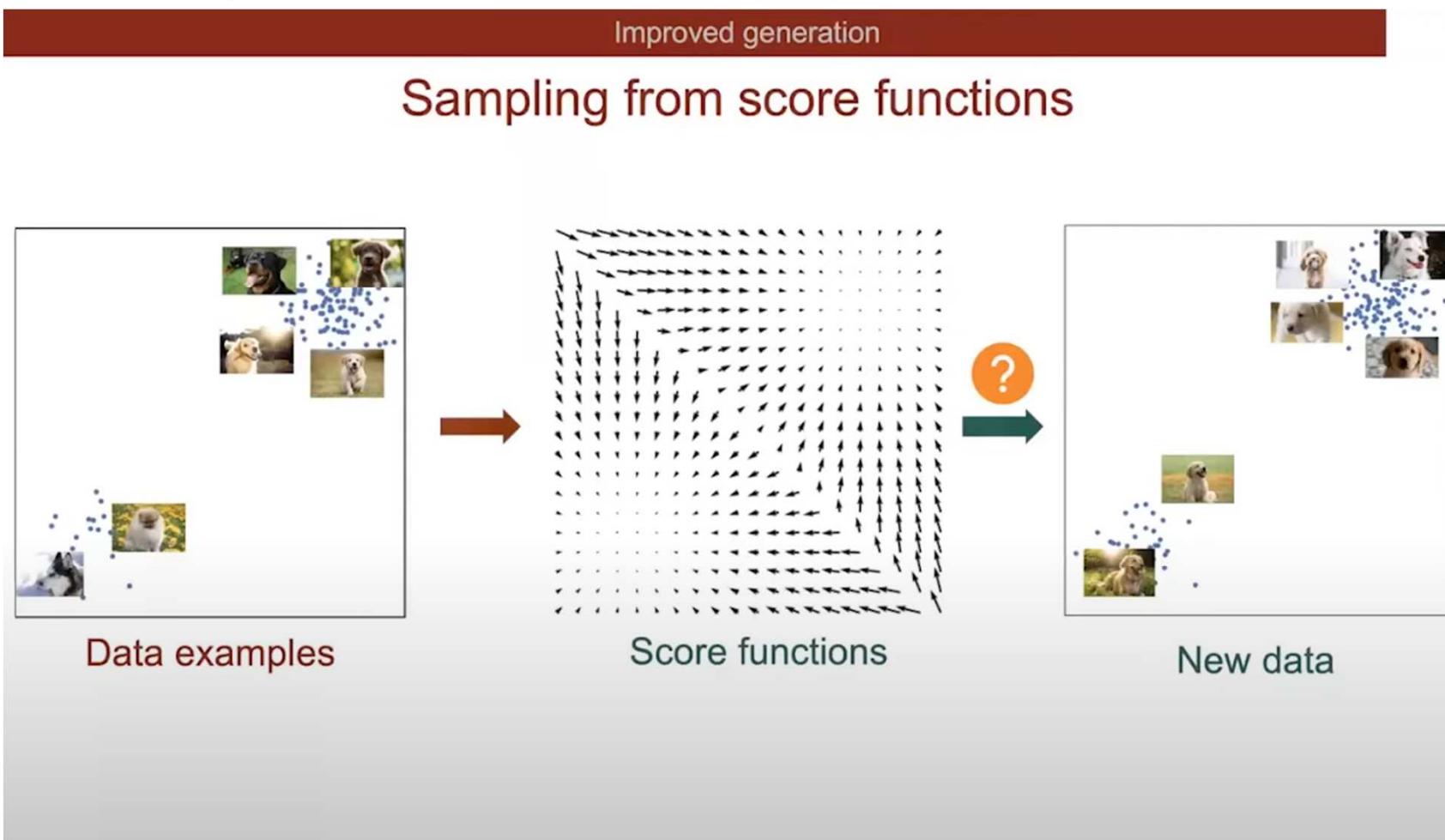
Probability evaluation

- Accurate probability evaluation
- Better estimation of data probabilities

[Song et al. ICLR 2021 oral]
[Song et al. NeurIPS 2021 spotlight]

- So now I have discussed how working with score functions allow very flexible models because score functions bypass the challenge of a normalizing constant, and we can use principled statistical methods like score matching, sliced score matching, or denoising score matching to train those score models from data.

[Improved generation]

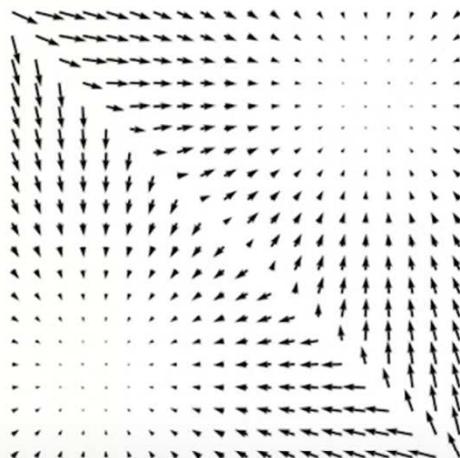


- As a quick recap, we know that, given a large training data set, we can use principaled statistical methods like score matching to train our score model to estimate the underlying score function.
- In order to **build our generative model**, we have to find a certain approach to **create new data points from the given vector field of score functions**.
- So how can we do this?
- Well, suppose we are already given the score function.

[Improved generation]

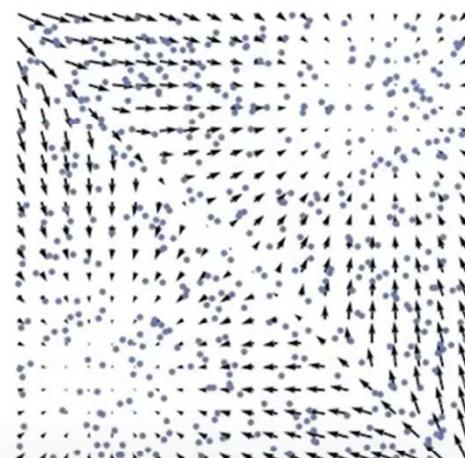
Improved generation

Sampling from score functions: Langevin dynamics



Score function

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



Follow the scores

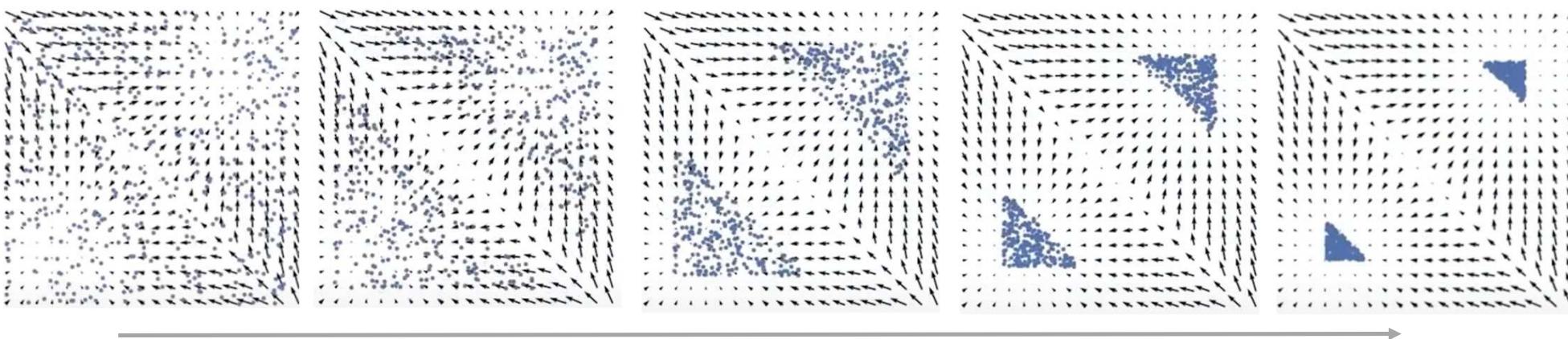
$$\tilde{x}_{t+1} \leftarrow \tilde{x}_t + \frac{\epsilon}{2} s_{\theta}(\tilde{x}_t)$$

- Imagine there are many random points scattered across the stairs.
- Can we move those random points to form samples from the score function?
- Well, **one idea** is we can potentially **move those points by following the directions predicted by the score function**.

[Improved generation]

Improved generation

Sampling from score functions: Langevin dynamics



Follow the scores

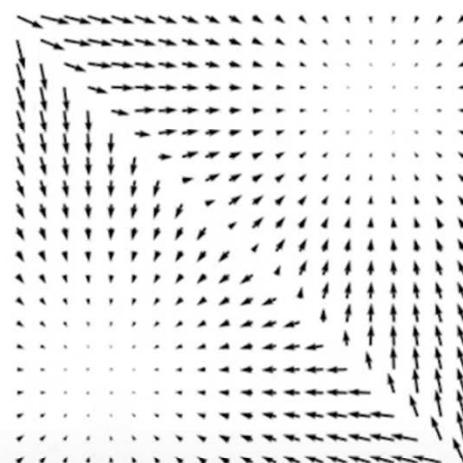
$$\tilde{\mathbf{x}}_{t+1} \leftarrow \tilde{\mathbf{x}}_t + \frac{\epsilon}{2} \mathbf{s}_\theta(\tilde{\mathbf{x}}_t)$$

- However, **this will not give us valid samples** because all of those points will eventually collapse into each other.

[Improved generation]

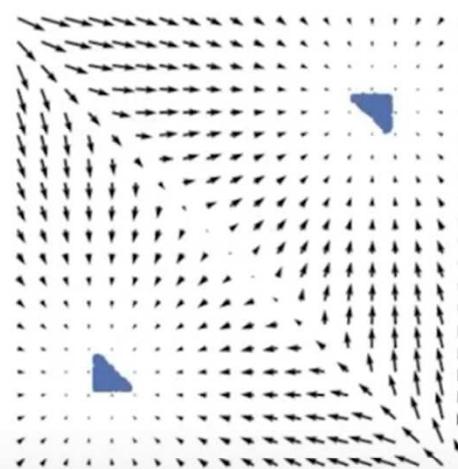
Improved generation

Sampling from score functions: Langevin dynamics



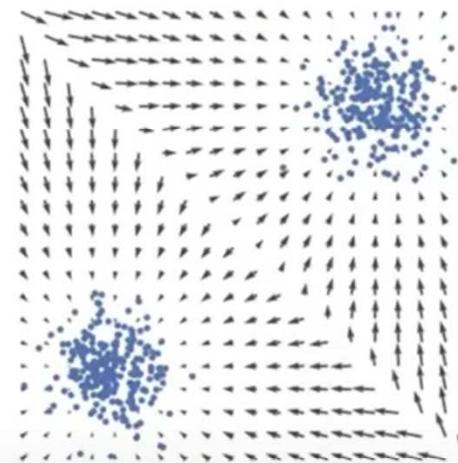
Score function

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



Follow the scores

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



Follow the noisy scores

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

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

- But this problem can be addressed if we follow a noise inversion of the score function.
- Equivalently, we want to inject Gaussian noise to our score function and follow those noise perturbed score functions.
- This method is the well-known approach of **Langevin dynamics**.

Langevin dynamics

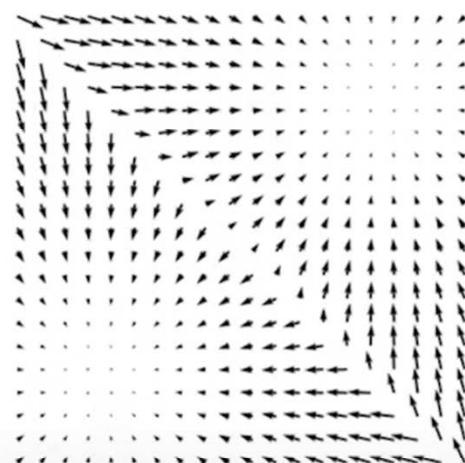
[Parisi 1981]

[Grenander and Miller, 1994]

[Improved generation]

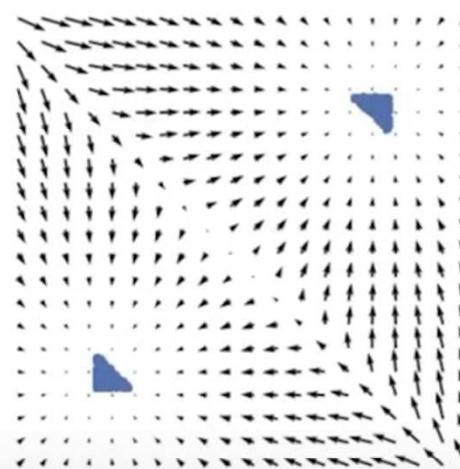
Improved generation

Sampling from score functions: Langevin dynamics



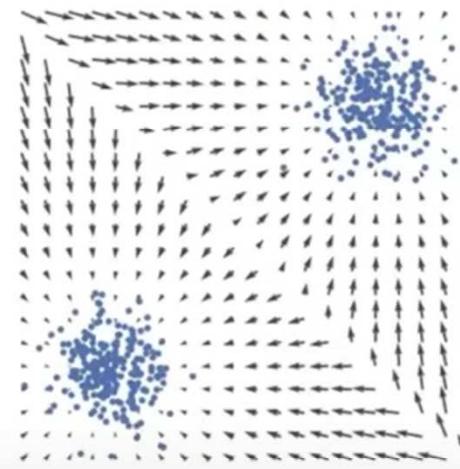
Score function

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



Follow the scores

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



Follow the noisy scores



Lan
[Grenan
Correct samples
guaranteed
4]

- And it is also well known that if we keep this sampling procedure long enough to reach convergence, and if we set the step size to be very, very small, then **Langevin dynamics** [INAUDIBLE] to give you the **correct samples from the score function**.
- This is the details of **Langevin sampling**.

[Improved generation]

Improved generation

Langevin dynamics sampling

- Sample from $p(\mathbf{x})$ using only the score $\nabla_{\mathbf{x}} \log p(\mathbf{x})$
- Initialize $\mathbf{x}^0 \sim \pi(\mathbf{x})$
- Repeat for $t \leftarrow 1, 2, \dots, T$

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

$$\mathbf{x}^t \leftarrow \mathbf{x}^{t-1} + \frac{\epsilon}{2} \nabla_{\mathbf{x}} \log p(\mathbf{x}^{t-1}) + \sqrt{\epsilon} \mathbf{z}^t$$

- The goal is to **sample from some density $p(\mathbf{x})$ using only the score function-- the gradient of $p(\mathbf{x})$.**
- The procedure of Langevin dynamics is as follows
- First, we initialize our sample from some prior distribution. This prior distribution can be very simple. It can be a Gaussian distribution or a uniform distribution.
- Then we repeat the following procedure multiple times.
- In each of the sampling steps, we first generate a random Gaussian vector from the standard Gaussian distribution.
- And then we modify \mathbf{x} according to following recurrence equation; We basically update the previous sample using our score function plus a scaled version of the Gaussian noise vector.

[Improved generation]

Improved generation

Langevin dynamics sampling

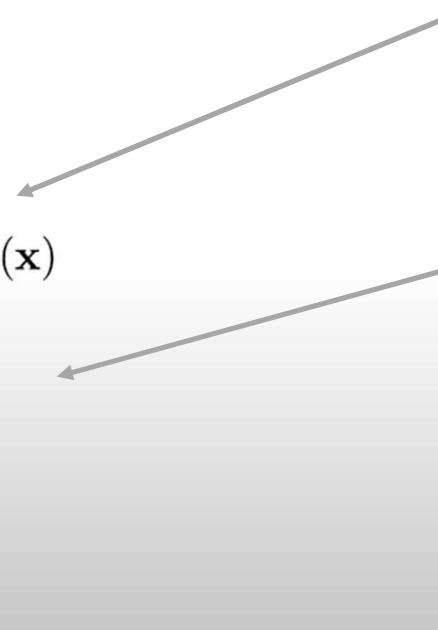
- Sample from $p(\mathbf{x})$ using only the score $\nabla_{\mathbf{x}} \log p(\mathbf{x})$
- Initialize $\mathbf{x}^0 \sim \pi(\mathbf{x})$
- Repeat for $t \leftarrow 1, 2, \dots, T$

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

$$\mathbf{x}^t \leftarrow \mathbf{x}^{t-1} + \frac{\epsilon}{2} \nabla_{\mathbf{x}} \log p(\mathbf{x}^{t-1}) + \sqrt{\epsilon} \mathbf{z}^t$$

- If $\epsilon \rightarrow 0$ and $T \rightarrow \infty$, we are guaranteed to have $\mathbf{x}^T \sim p(\mathbf{x})$
- Langevin dynamics + score estimation

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



- If you set **epsilon** to something very **close to 0**, and if you set the total number of iterations, **T**, to be **large** enough, then we are **guaranteed to obtain a valid sample from the underlying density of the score function**.
- Now we know **score matching** can **estimate the score function data**.
- **Langevin dynamics** can generate samples from the score function.
- It becomes very natural to just replace the score function in Langevin dynamics with our score model, and then we can generate data samples - we define a new generative model.

[Improved generation]

Improved generation

Score matching + Langevin dynamics

CIFAR-10 data



Model samples



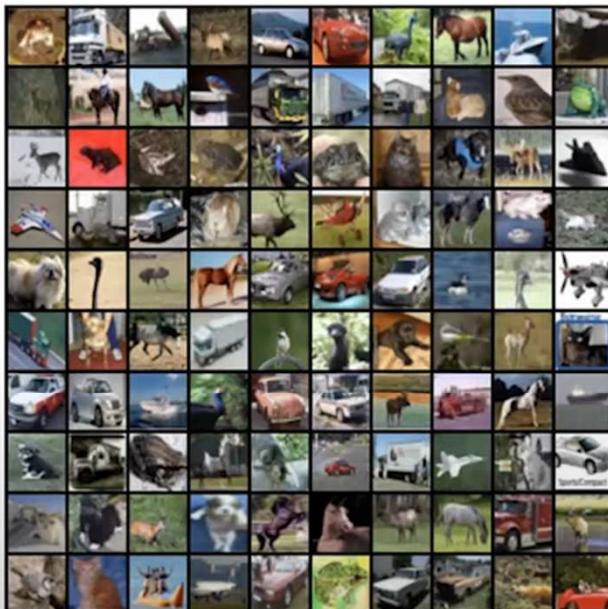
- This approach sounds very nice from the theoretical perspective, but it does not work well in practice.
- So here are the results of combining score matching and Langevin dynamics naively.
- The left figure shows some images from the CIFAR-10 data set. CIFAR-10 is a data set that contains many images of size 32x32
- The right figure shows you newly generated samples by combining score matching and Langevin dynamics naively. Clearly you can see that the newly generated samples do not look realistic at all.

[Improved generation]

Improved generation

Score matching + Langevin dynamics

CIFAR-10 data



Model samples



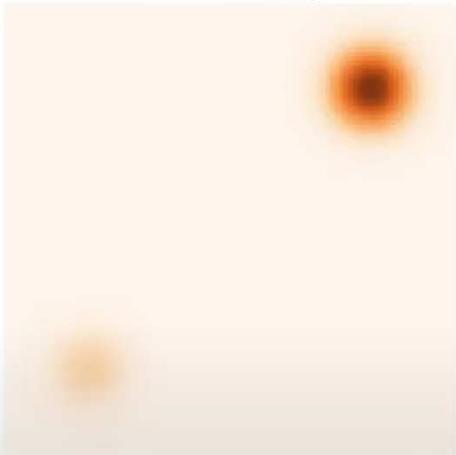
- This approach sounds very nice from the theoretical perspective, but it does not work well in practice.
- So here are the results of combining score matching and Langevin dynamics naively.
- The left figure shows some images from the CIFAR-10 data set. CIFAR-10 is a data set that contains many images of size 32x32
- The right figure shows you newly generated samples by combining score matching and Langevin dynamics naively. Clearly you can see that the newly generated samples do not look realistic at all.
- There has to be something very wrong with this simple naive approach.

[Improved generation]

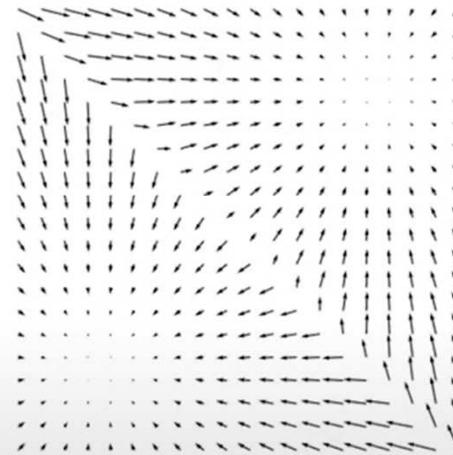
Improved generation

Challenge in low data density regions

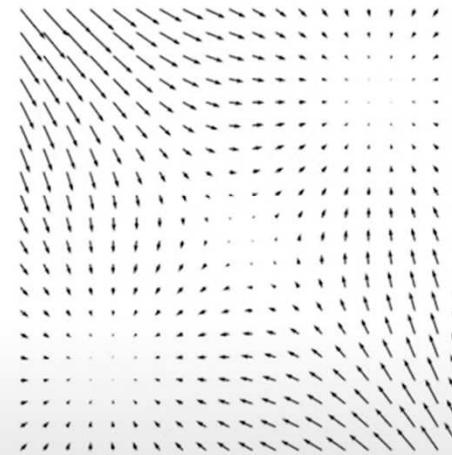
Data density



Data scores



Estimated scores



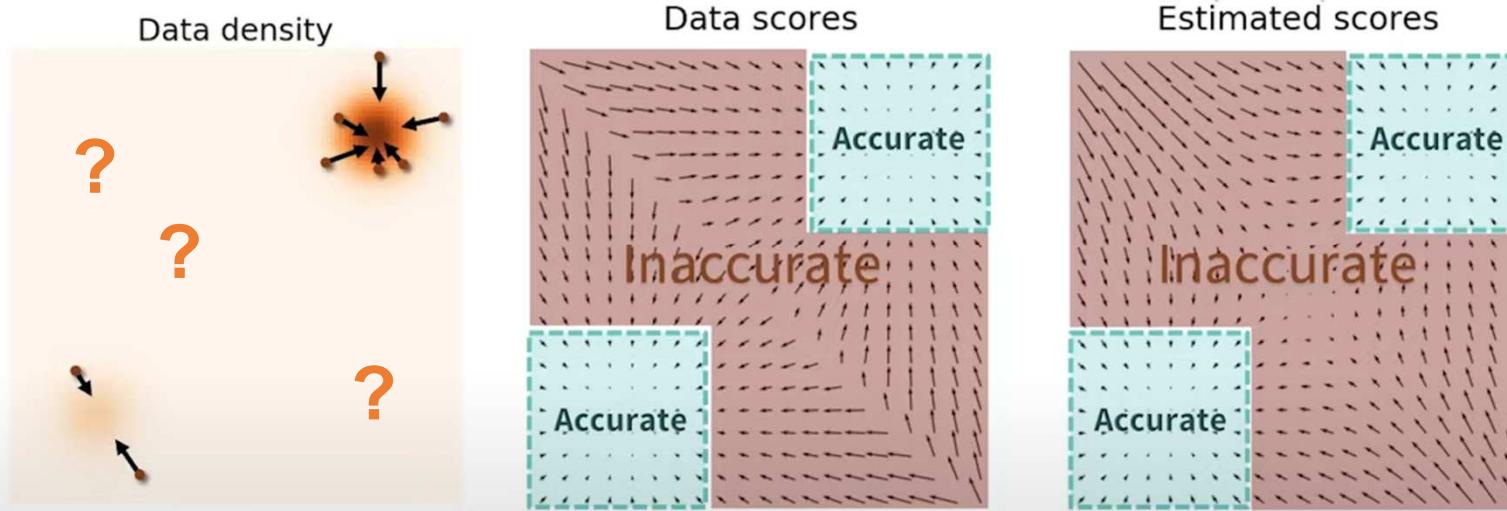
- In our research, we identified several challenges.
- One interesting challenge is it is **hard to estimate score functions accurately in low data density regions**.
- To illustrate this challenge, let's consider the prior example of a mixture of Gaussian distribution again.
- The left figure shows you the ground truth density function. Middle figure shows you the ground truth score function. The rightmost figure gives the estimated score function from score matching.

[Song and Ermon. NeurIPS 2019 (oral)]

[Improved generation]

Improved generation

Challenge in low data density regions



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

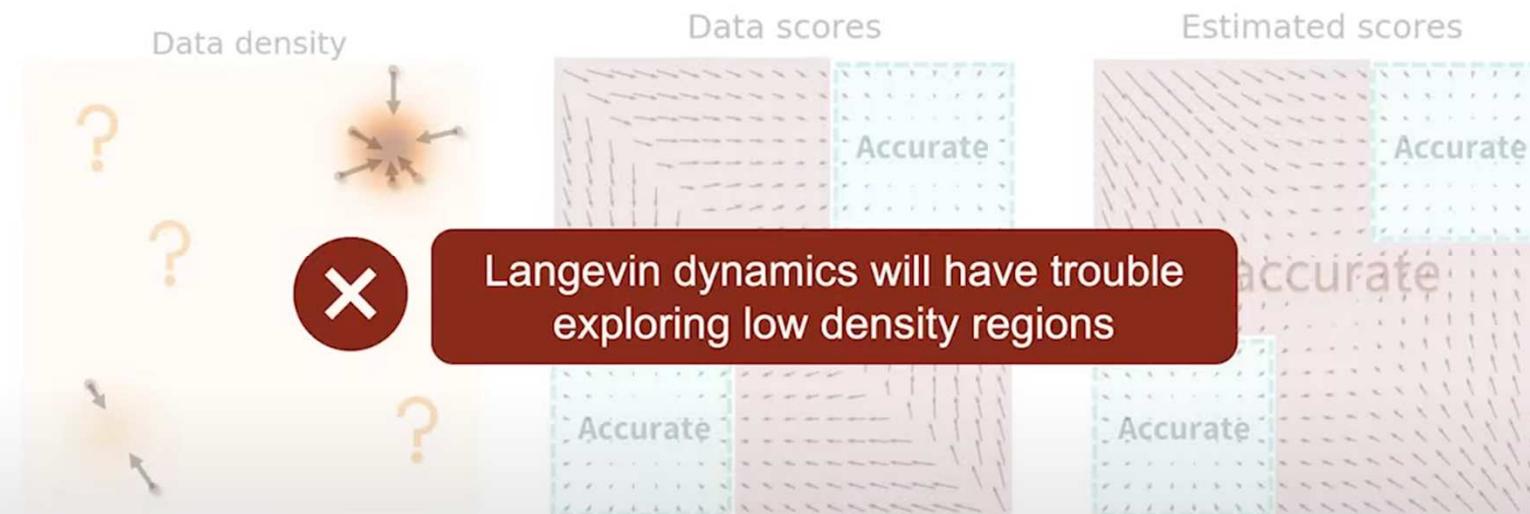
[Song and Ermon. NeurIPS 2019 (oral)]

- If you compare those two vector fields, it's clear that the estimated scores are accurate in high data density regions, which are given by those green boxes.
- But for low data density regions, the estimated scores are not accurate at all.
- This is not totally unexpected because we **use score matching to train our score model**, and **score matching compares the difference between the ground truth and the model only at samples from the data distribution**.
- In low data density regions, we don't know how many samples, and therefore we don't have enough information to infer the true score functions in those regions.

[Improved generation]

Improved generation

Challenge in low data density regions



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

[Song and Ermon. NeurIPS 2019 (oral)]

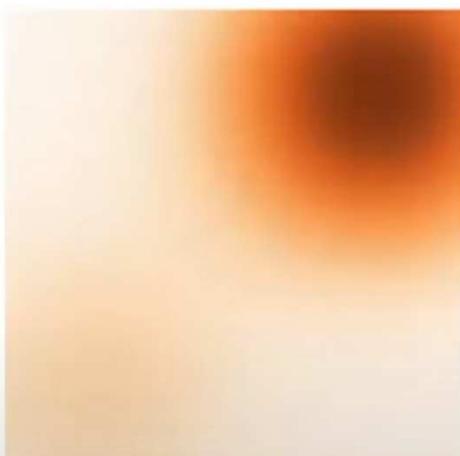
- And this is a **huge obstacle** for **Langevin dynamics** to provide high quality samples because Langevin dynamics will have a lot of trouble exploring and navigating those **low data density regions**.

[Improved generation]

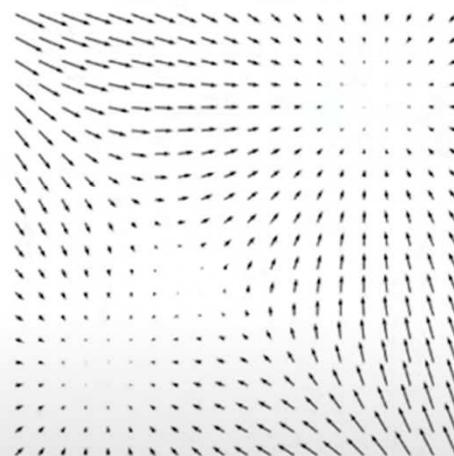
Improved generation

Improving score estimation by adding noise

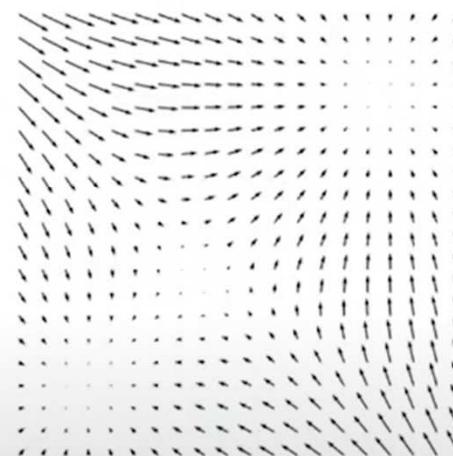
Perturbed density



Perturbed scores



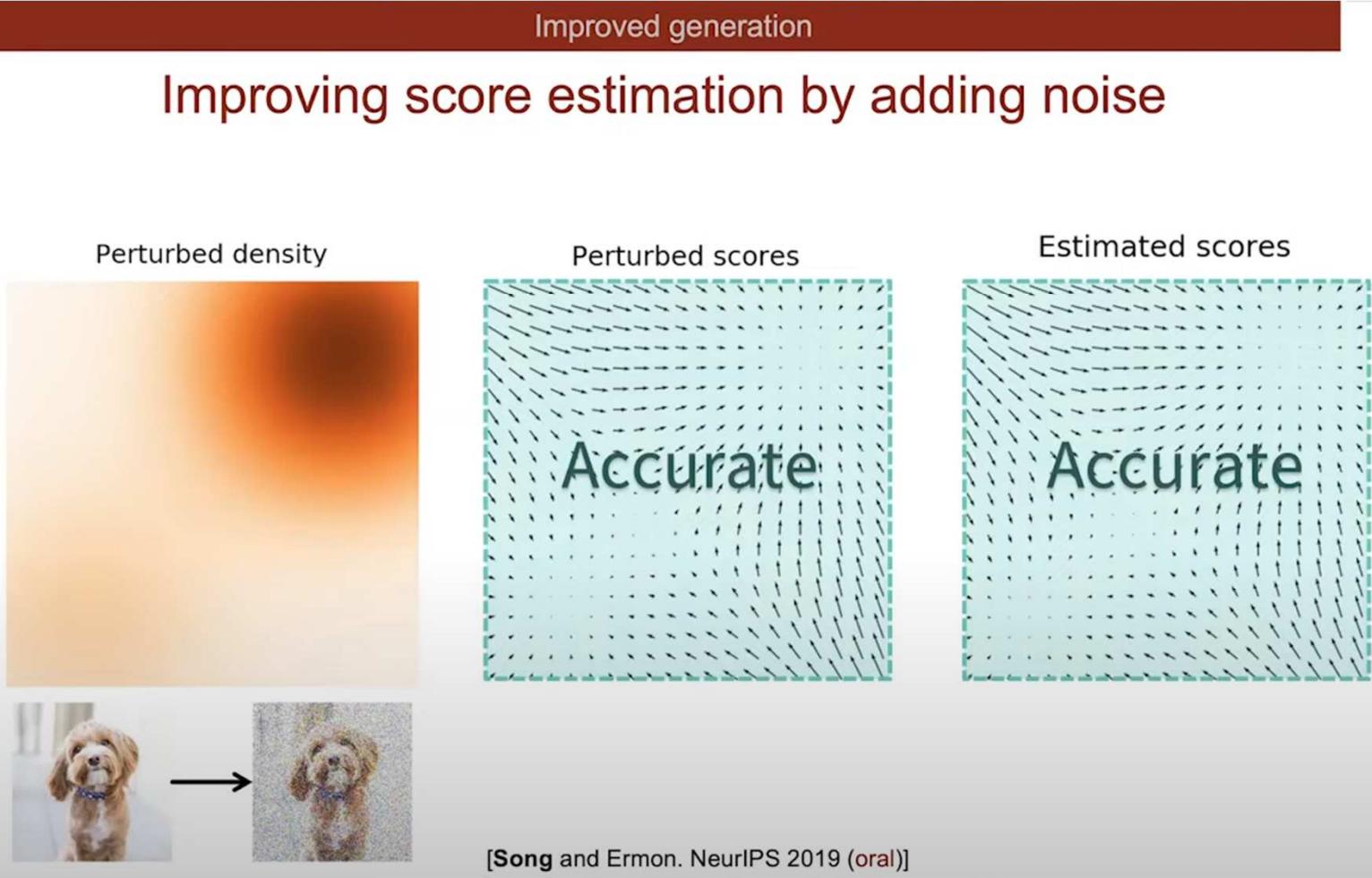
Estimated scores



[Song and Ermon. NeurIPS 2019 (oral)]

- So how can we address this challenge?
- One idea is to **inject Gaussian noise to perturb our data points**. So after adding enough Gaussian noise, we **perturb the data points to everywhere in the space**.
- This means **the size of low data density regions becomes smaller**.

[Improved generation]



- In the context of image generation, adding additional Gaussian noise means we **inject Gaussian noise to perturb each pixel of the image**.
- So in this toy example, you can see that, **after injecting the right amount of Gaussian noise, the estimated scores now become accurate almost everywhere**.

[Improved generation]

Improved generation

Improving score estimation by adding noise

Perturbed density

Perturbed scores

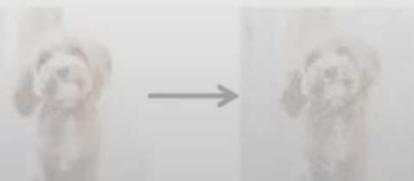
Estimated scores



High noise provides useful directional information for Langevin dynamics.



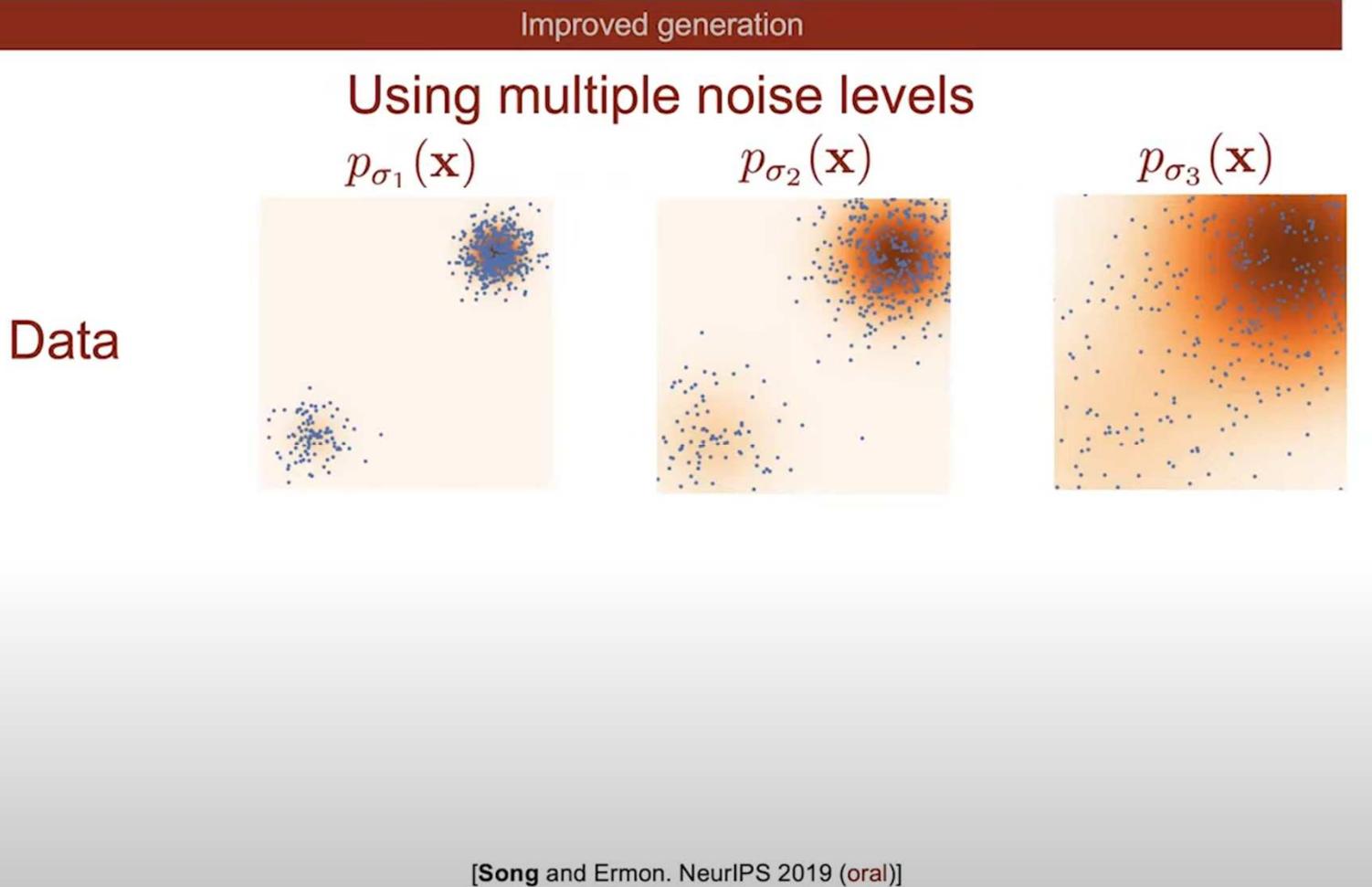
But perturbed density no longer approximates the true data density.



[Song and Ermon. NeurIPS 2019 (oral)]

- This phenomenon is very promising because it at least says that the **score function of noisy data densities** are much easier to be estimated **accurately**, and those score functions of noisy densities could **provide valuable directional information to guide Langevin dynamics to move from low data density region to high data density regions.**
- But simply injecting Gaussian noise will not solve all the problems. Because of perturbation of data points, those noisy data distances **are no longer good approximations to the original true data density**.

[Improved generation]



- To solve this problem, we propose to use a **multiple sequence of different noise levels**.
- So as a toy example, we consider **three noise levels** from sigma_1 to sigma_3.
- We used Gaussian noise on mean 0 and standard deviation from sigma_1 to sigma_3 to perturb our training data set.
- And this will give us three noisy training data sets.
- For Each noisy data set, there will be a corresponding noise data density, which we denote as p_{σ_1} to p_{σ_3}

[Improved generation]

Improved generation

Using multiple noise levels

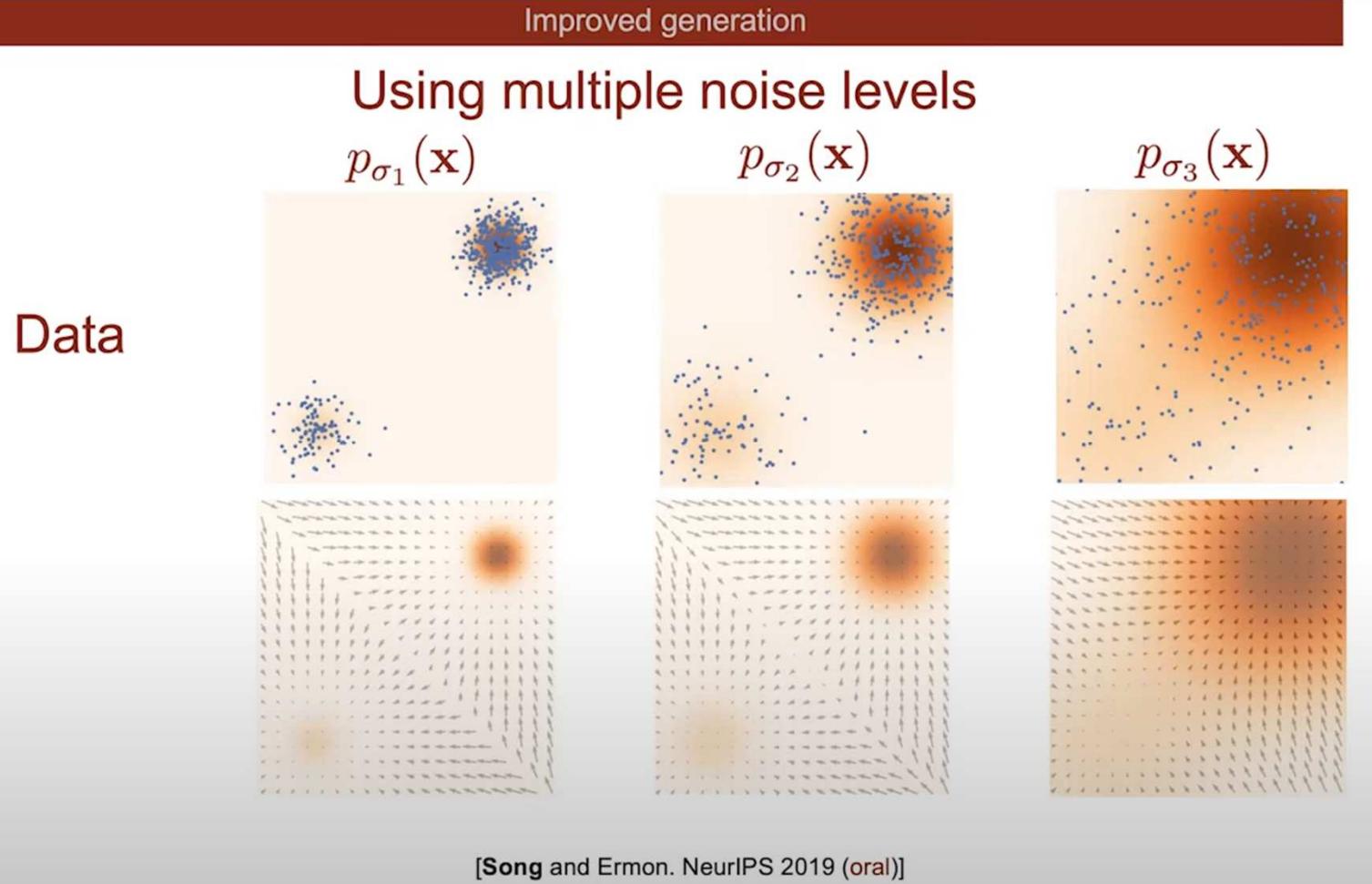
- So in the context of images, perturbation using multiple levels of noise will give you a sequence of images demonstrated here.

Data



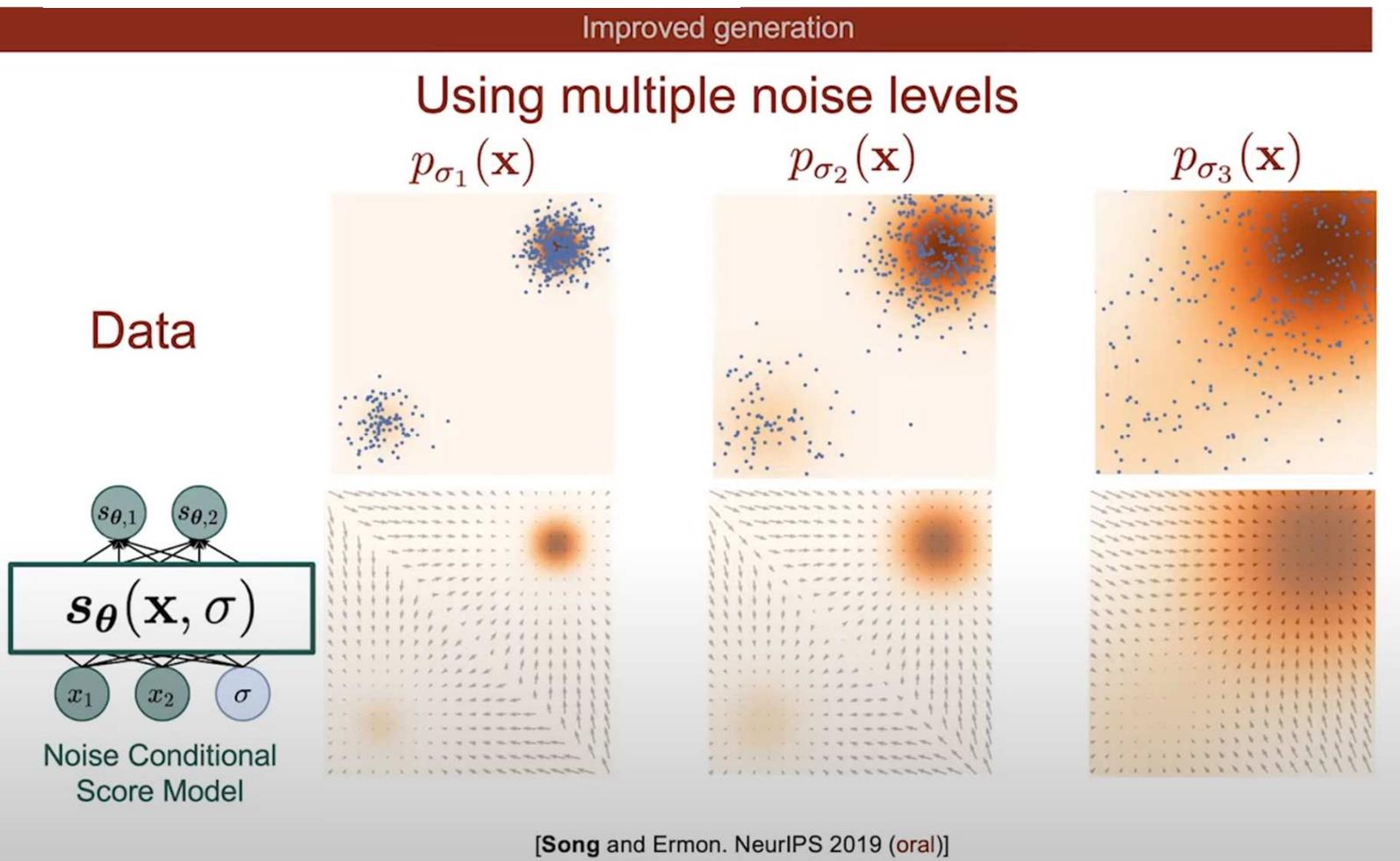
[Song and Ermon. NeurIPS 2019 (oral)]

[Improved generation]



- After obtaining those noisy data sets, we want to estimate the underlying density.
- We want to estimate the underlying function of the corresponding noisy to the densities.
- **How can we estimate three noisy score functions?**
- Well, the most naive approach is we train three networks, and each network is responsible for estimating the score function of a single noise level.
- But this is not a scalable solution. Because in practice, we might require much more noise levels.
- For example, our image generation would typically require hundreds to thousands of noise levels.

[Improved generation]



- A more scalable solution is to consider a **conditional score model**, which we call a ***noise conditional score model***.
- A noise conditional score model is a simple modification to our score model.
- It takes **noise level sigma** as one **additional input dimension** to the model.
- The **output corresponds to the score function of the data density perturbed with noise level sigma**.
- How can we train this noise conditional score model. Well, again, we can leverage the idea of **score matching**.

[Improved generation]

Improved generation

Using multiple noise levels

$p_{\sigma_1}(\mathbf{x})$
 $p_{\sigma_2}(\mathbf{x})$
 $p_{\sigma_3}(\mathbf{x})$

Data

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

$s_{\theta,1}$ $s_{\theta,2}$

x_1 x_2 σ

Noise Conditional Score Model

Positive weighting function

$$\frac{1}{N} \sum_{i=1}^N \underbrace{\lambda(\sigma_i)}_{\text{Noise level}} \underbrace{\mathbb{E}_{p_{\sigma_i}(\mathbf{x})} [\|\nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x}) - s_{\theta}(\mathbf{x}, \sigma_i)\|_2^2]}_{\text{Score matching loss}}$$

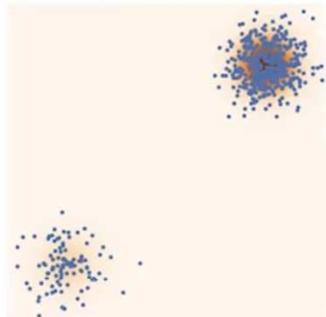
[Song and Ermon. NeurIPS 2019 (oral)]

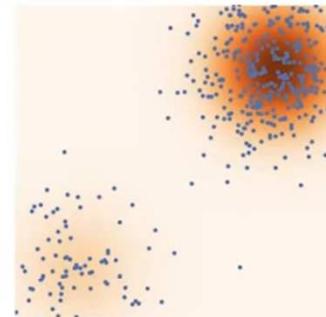
- We have an important modification with score matching to jointly train the score model across all levels.
- In this modification we have a summation with **score matching losses**.
- We have **one score matching loss** for each noise level σ_i and we have **a positive weighting function**, $\lambda(\sigma_i)$.
- The value of this weighting function is typically chosen using [INAUDIBLE] heuristics. It can also be derived using principled analysis of the problem.
- We have **this positive weighting function** just to **balance the scales of score matching loss across all noise levels**, and this is **helpful for optimization**.
- By minimizing this modified score matching loss, if our optimizer is powerful enough, and if our model is expressive enough, then we will obtain accurate score estimation for all noise labels.

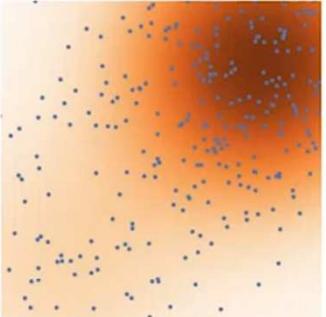
[Improved generation]

Improved generation

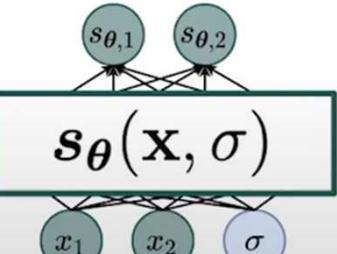
Using multiple noise levels

$p_{\sigma_1}(\mathbf{x})$


$p_{\sigma_2}(\mathbf{x})$


$p_{\sigma_3}(\mathbf{x})$


Data

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


Positive weighting function

$$\frac{1}{N} \sum_{i=1}^N \underbrace{\lambda(\sigma_i) \mathbb{E}_{p_{\sigma_i}(\mathbf{x})} [\|\nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x}) - s_{\theta}(\mathbf{x}, \sigma_i)\|_2^2]}_{\text{Score matching loss}}$$

Noise level

[Song and Ermon. NeurIPS 2019 (oral)]

- We have an important modification with score matching to jointly train the score model across all levels.
- In this modification we have a summation with **score matching losses**.
- We have **one score matching loss** for each noise level σ_i and we have **a positive weighting function**, $\lambda(\sigma_i)$.
- The value of this weighting function is typically chosen using [INAUDIBLE] heuristics. It can also be derived using principled analysis of the problem.
- We have **this positive weighting function** just to **balance the scales of score matching loss across all noise levels**, and this is **helpful for optimization**.
- By minimizing this modified score matching loss, if our optimizer is powerful enough, and if our model is expressive enough, then we will obtain accurate score estimation for all noise labels.

[Improved generation]

Improved generation

Using multiple noise levels

$p_{\sigma_1}(\mathbf{x})$ $p_{\sigma_2}(\mathbf{x})$ $p_{\sigma_3}(\mathbf{x})$

Data

Noise Conditional Score Model

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

$s_{\theta,1}$ $s_{\theta,2}$

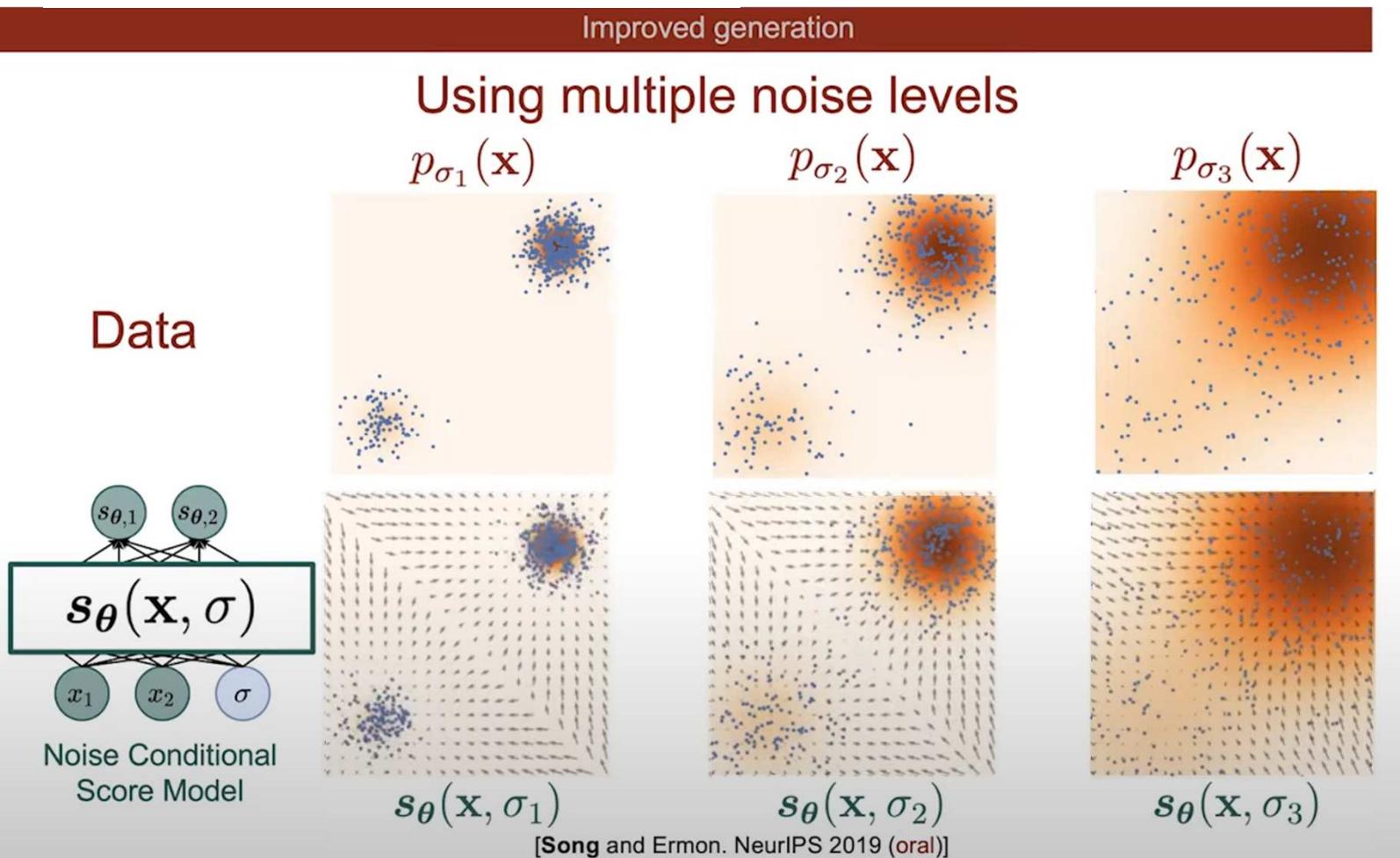
x_1 x_2 σ

- A generalization to the training objective of diffusion probabilistic models [Sohl-Dickstein et al. 2015]
- First unveiled by DDPM [Ho et al. 2020]

[Song and Ermon. NeurIPS 2019 (oral)]

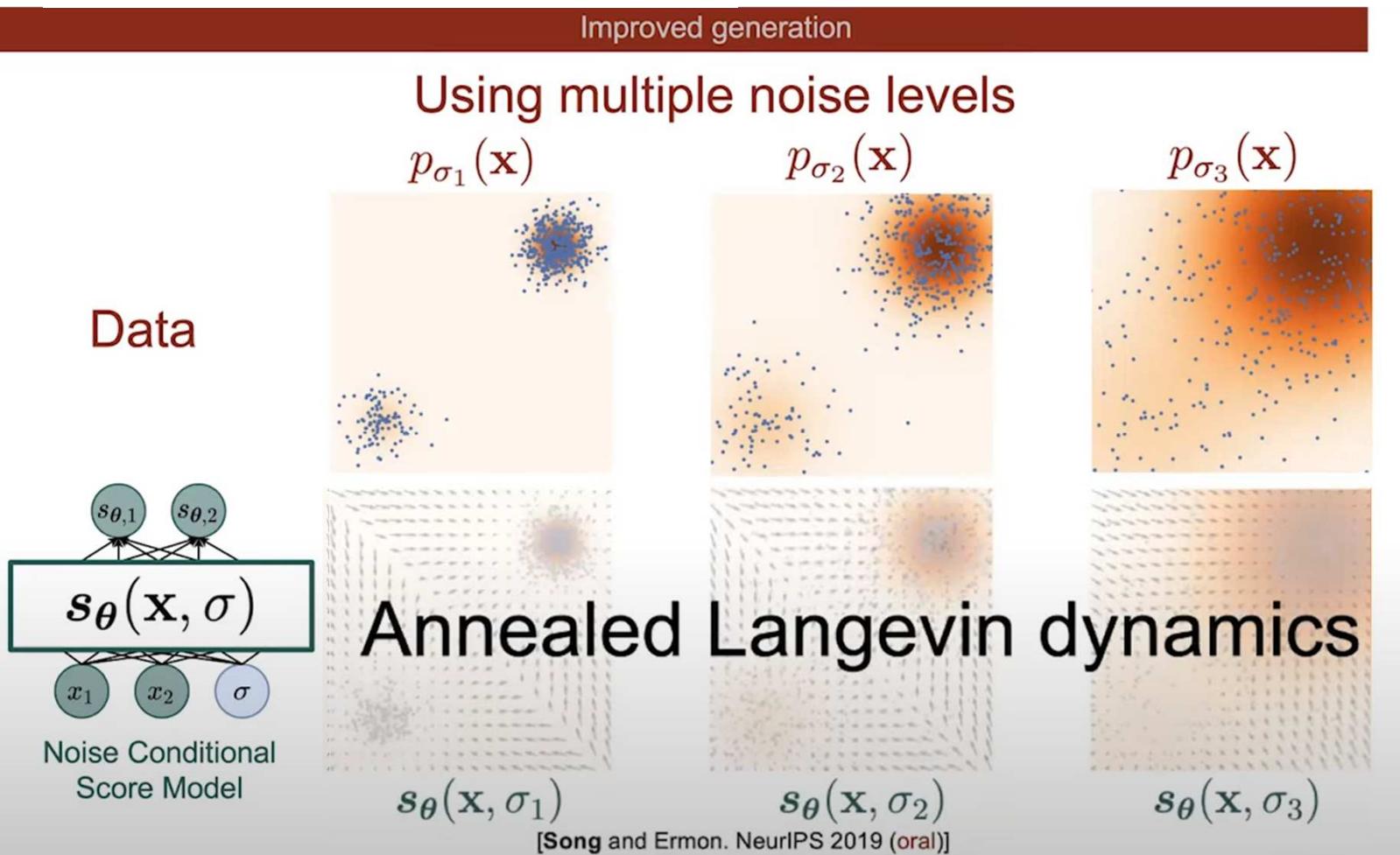
- After training this noise-conditional score model, **how do we generate samples?**
- Well, one additional note is that **this mixture of score matching loss function** is actually a generalization to the training objective of the first version diffusion probabilistic models proposed in 2015.
- And this **connects score-based generative models to diffusion models**.
- The **connection between score-based models and diffusion models** was first unveiled by the **DDPM** paper, which was in 2020.

[Improved generation]



- So now, let's return to the question of **how to sample from the noise-conditional score model after training with the score matching loss.**
- Well, we can still use Langevin dynamics.
- We can **first apply Langevin dynamics to sample from the score model with the biggest perturbation noise.**
- And the samples will be used as the **initialization to sample from the score model of the next noise level.**
- And then we **continue** in this fashion until finally with generate samples from the **score function with the smallest noise level.**

[Improved generation]



- We call this sampling procedure **annealed Langevin dynamics** because the rough intuition is we hope to **gradually anneal down the temperature of our data density** to gradually reduce the noise level.

[Improved generation]

Improved generation

Score-based generative modeling in the real world



[Song and Ermon. NeurIPS 2019 (oral)]

- And this is what it looks like when we apply this approach to modeling real images.
- So it's quite remarkable that we can start from a random noise, then modify those images according to the score model, and this can eventually give us nice looking samples.

[Improved generation]

Improved generation		
Quantitative results on CIFAR0-10		
Model	Inception	FID
CIFAR-10 Unconditional		
PixelCNN [59]	4.60	65.93
PixelIQN [42]	5.29	49.46
EBM [12]	6.02	40.58
WGAN-GP [18]	$7.86 \pm .07$	36.4
MoLM [45]	$7.90 \pm .10$	18.9
SNGAN [36]	$8.22 \pm .05$	21.7
ProgressiveGAN [25]	$8.80 \pm .05$	-
NCSN (Ours)	$8.87 \pm .12$	25.32

[Song and Ermon. NeurIPS 2019 (oral)]

- And this is the result of this simple noise conditional score model approach in 2019.
- So we provided similar scores of CIFAR-10 data set. Those inception scores and FIDs are important quantities for comparing the performance of different generative models in terms of sample quality.
- This was the first time that a different method can outperform GANs in terms of achieving higher inception score.
- Of course FID score is still lagging behind GANs, but at that time it was quite surprising that a simple proof can already outperform GANs in one important metric, which is the inception score.

[Improved generation]

Improved generation

GANs were the best for sample generation



Yann LeCun

Turing Award 2018

“(GANs are) the most interesting idea in the last 10 years in Machine Learning”

Years of extensive engineering from Google, Nvidia, Facebook, OpenAI...



Beat GANs on CIFAR-10 for the first time! (in Inception Score)

[Song and Ermon. NeurIPS 2019 (oral)]

- So why it is important to outperform GANs? Because GANs were the best generative model for sample generation, especially for images, for quite a while.
- As Turing Award winner, Yann LeCun, has said, are the most interesting idea in the last 10 years in machine learning.
- Indeed GANs have attracted a lot of research efforts from big corporations and universities, and people have improved GANs so much-- spent so much engineering effort on it— it is amazing to see that GANs can generate very nice-looking images.
- But it is quite amazing that we can actually outperform GANs with score-based generative models. And with the resources available in academia,

[Improved generation]

Improved generation

GANs were the best for sample generation



Yann LeCun
Turing Award 2018

“(GANs are) the most interesting idea in the last 10 years in Machine Learning”

Years of extensive engineering from Google, Nvidia, Facebook, OpenAI...

- So why it is important to outperform GANs? Because GANs were the best generative model for sample generation, especially for images, for quite a while.
- As Turing Award winner, Yann LeCun, has said, are the most interesting idea in the last 10 years in machine learning.
- Indeed GANs have attracted a lot of research efforts from big corporations and universities, and people have improved GANs so much-- spent so much engineering effort on it— it is amazing to see that GANs can generate very nice-looking images.

[Song and Ermon. NeurIPS 2019 (oral)]

[Improved generation]

Improved generation

GANs were the best for sample generation



Yann LeCun
Turing Award 2018

“(GANs are) the most interesting idea in the last 10 years in Machine Learning”

Years of extensive engineering from Google, Nvidia, Facebook, OpenAI...

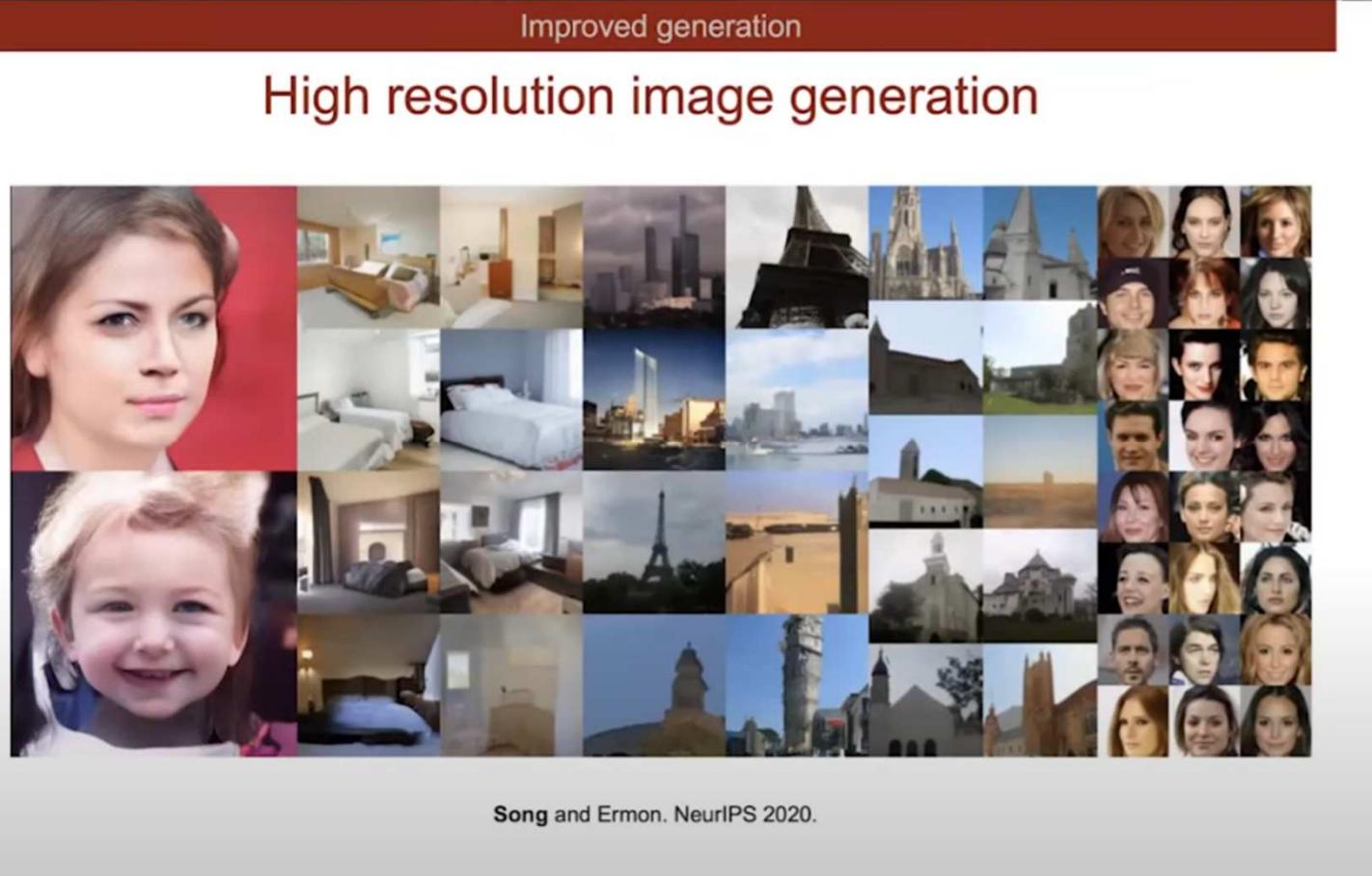
- But it is quite amazing that we can actually outperform GANs with score-based generative models.
- And with the resources available in academia, we actually do not have much capability to tune those kind of score models well enough, so especially considering the imbalance between computer resources and engineering efforts spent on GANs and score-based models, I consider this a very surprising achievement.



Beat GANs on CIFAR-10 for the first time! (in Inception Score)

[Song and Ermon. NeurIPS 2019 (oral)]

[Improved generation]



Song and Ermon. NeurIPS 2020.

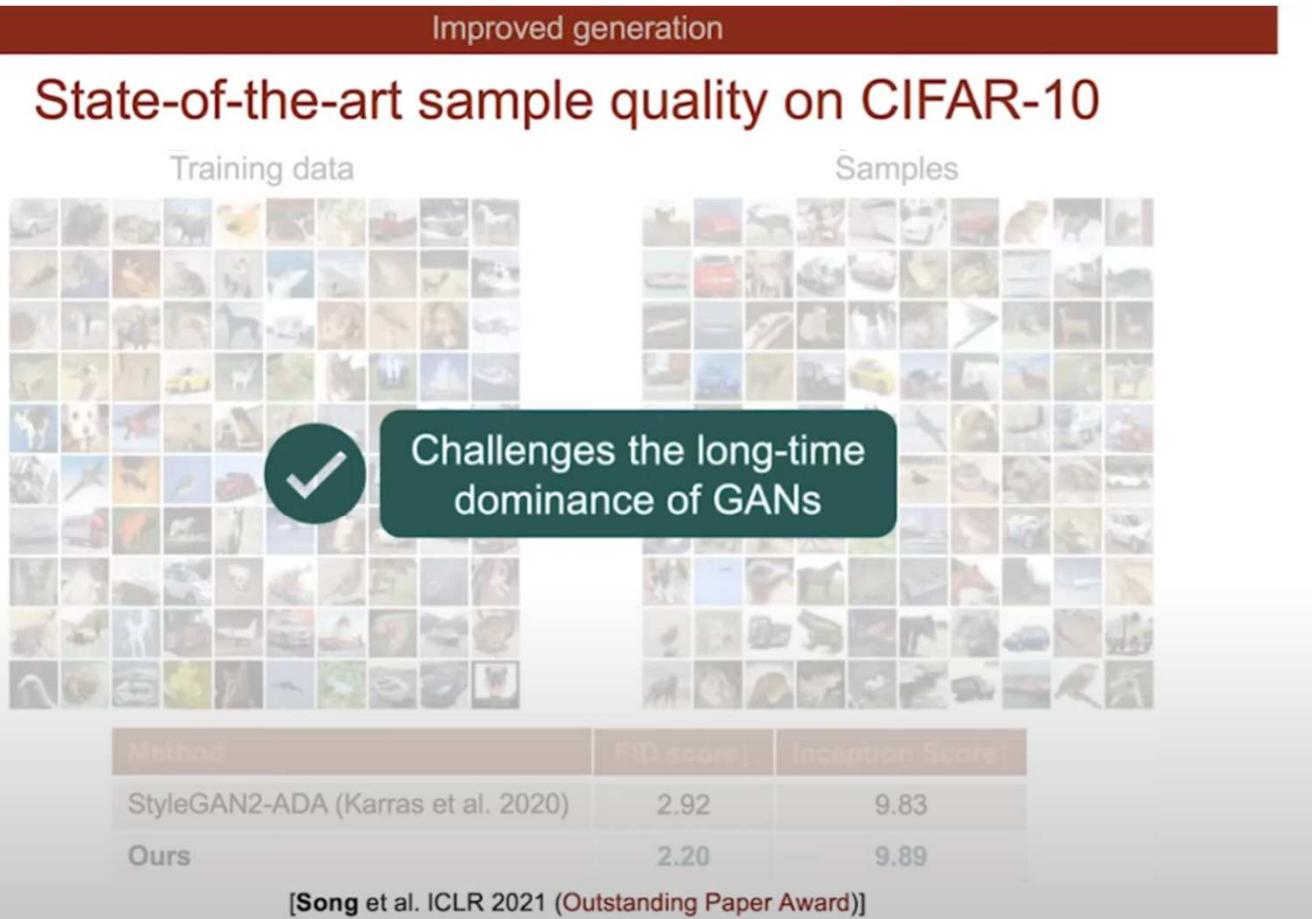
- So, of course, noise-conditional score models can be applied to other types of image generation tasks, including images of different objects and of different resolution.
- With some later development score matching techniques and neural network model architectures, we can further improve the sample quality of CIFAR-10.
- And Of course, nowadays, diffusion models

[Improved generation]



- Nowadays, diffusion models have captured so much attention, and people are now working on diffusion, trying to improve their various perspectives.
- It's not unexpected that people are achieving better and better quality using diffusion models or score-based models.
- In this work, we, again, data set of CIFAR-10.
- The left figure shows some existing training images from the CIFAR-10 data set.
- The right figure shows the newly generated samples from this improved approach. So now you can see new regenerative samples look very realistic and very diverse. They are also different from existing training images. You cannot generate such images by simply memorizing the training data set.

[Improved generation]



- And again, we compare with the best approach in terms of FID scores and inception scores. So now we are able to outperform the best GAN approach in terms of both FID scores and inception scores.
 - And this means score-based models can challenge **the long-time dominance of GANs in every generation**.

[Improved generation]

Improved generation

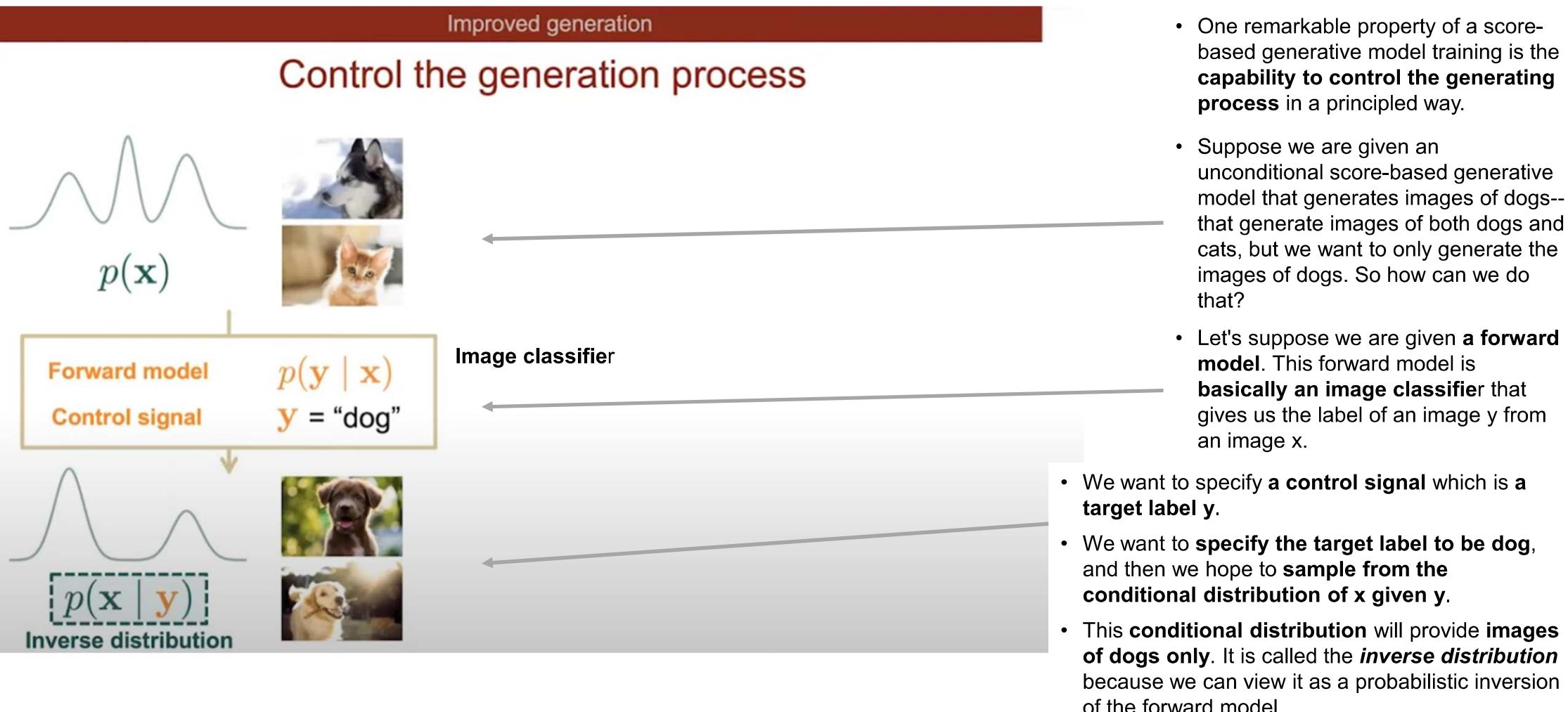
High-fidelity generation of 1024x1024 images



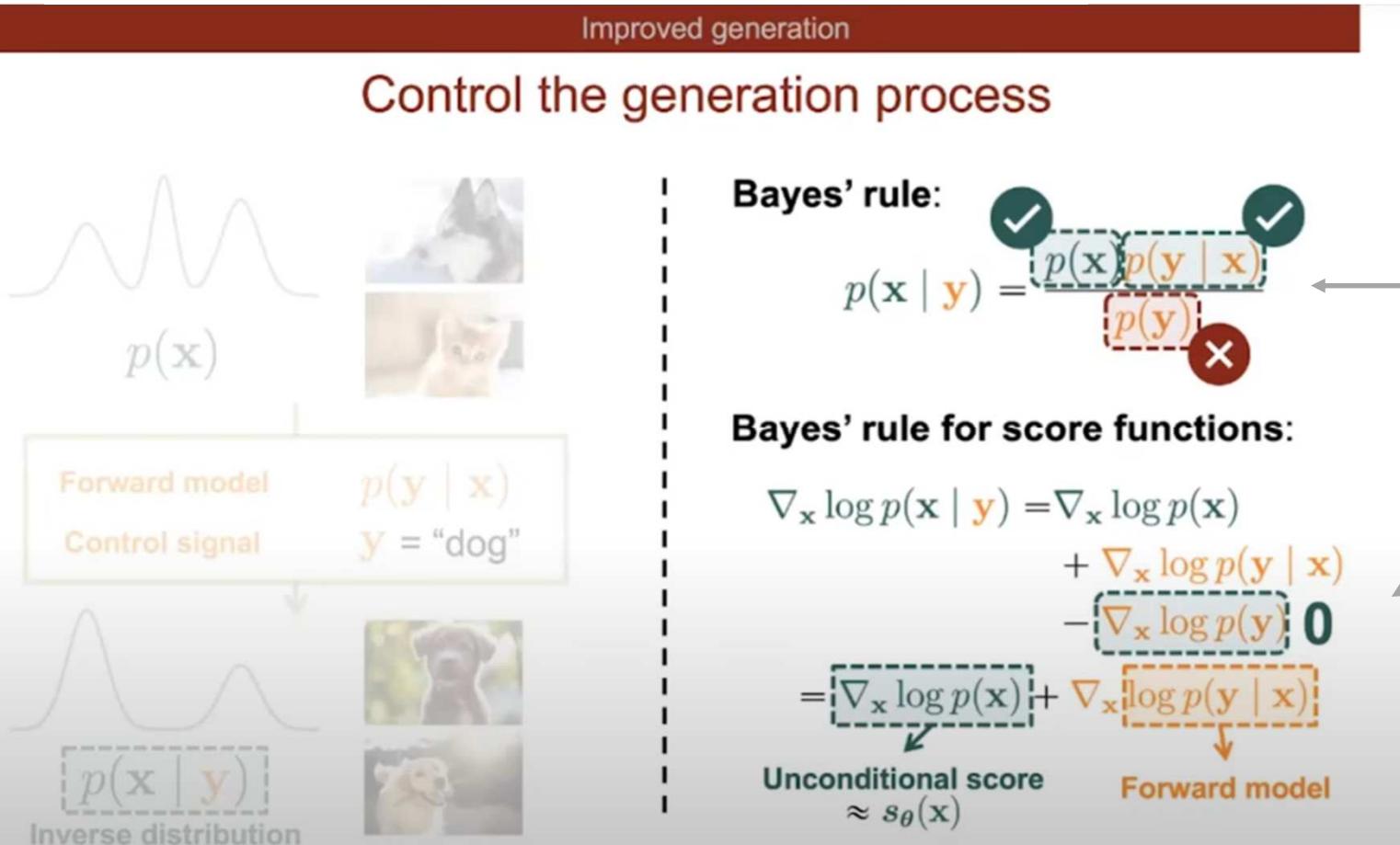
[Song et al. ICLR 2021 (Outstanding Paper Award)]

- The same approach can be extended to generate images of very large resolution.
- So here are two samples generated from a score-based model. Each one has the resolution of 1,024 by 1,024.
- And here are more such samples from the same model with same resolution. So you can see the samples are very high quality quite comparable to the best GAN approaches [INAUDIBLE] time.

[Improved generation - Inverse Distribution]



[Improved generation - Inverse Distribution]



- How can we obtain this inverse distribution?
- The standard approach is to leverage the Bayes's rule.
- In Bayes's rule we have access to the unconditional distribution $p_{\mathbf{x}}$.
- We are given the forward model, but we don't know the denominator.
- This denominator is exactly the normalizing constant of the inverse distribution.
- This means we can use score functions to again bypass this challenge in Bayes's rule, and we can derive the Bayes's rule for score functions very easily. So the derivation is very simple.
- We just take the logarithm on both sides of Bayes's rule and then take the gradient
- Again, we can find that the only term that depends on the denominator goes away.

[Improved generation - Inverse Distribution]

Improved generation

Control the generation process

**Bayes' rule:**

$$p(\mathbf{x} | \mathbf{y}) = \frac{p(\mathbf{x}) p(\mathbf{y} | \mathbf{x})}{p(\mathbf{y})}$$

✓ ✓
p(x) p(y | x)
p(y) ✗

Bayes' rule for score functions:

$$\begin{aligned} \nabla_{\mathbf{x}} \log p(\mathbf{x} | \mathbf{y}) &= \nabla_{\mathbf{x}} \log p(\mathbf{x}) \\ &\quad + \nabla_{\mathbf{x}} \log p(\mathbf{y} | \mathbf{x}) \\ &\quad - \boxed{\nabla_{\mathbf{x}} \log p(\mathbf{y})} \mathbf{0} \\ &= \boxed{\nabla_{\mathbf{x}} \log p(\mathbf{x})} + \boxed{\nabla_{\mathbf{x}} \log p(\mathbf{y} | \mathbf{x})} \\ \text{Unconditional score} &\approx s_{\theta}(\mathbf{x}) \qquad \qquad \qquad \text{Forward model} \end{aligned}$$

- This means we can **use score functions** to again bypass this challenge in Bayes's rule, and we can derive the **Bayes's rule for score functions** very easily.
- We just take the logarithm on both sides of Bayes's rule and then take the gradient. Again, we can find that the only term that depends on the denominator goes away.
- **The score function of the inverse distribution** now becomes a simple summation with two terms.
- The first term is the **unconditional score function** that can be estimated by training an **unconditional score model**.
- The second term is the **gradient of the log forward model**.

[Improved generation - Inverse Distribution]

Improved generation

Control the generation process

**Bayes' rule:**

$$p(x | y) = \frac{p(x)p(y | x)}{p(y)}$$

✓ ✓ ✗

Bayes' rule for score functions:

$$\begin{aligned} \nabla_x \log p(x | y) &= \nabla_x \log p(x) \\ &\quad + \nabla_x \log p(y | x) \\ &\quad - \boxed{\nabla_x \log p(y)} 0 \\ &= \boxed{\nabla_x \log p(x)} + \boxed{\nabla_x \log p(y | x)} \end{aligned}$$

Plug in different forward models for the same score model

✓

- In this particular application, **conditional image generation**, the **forward model** is the **classifier**, and the **gradient** can be very **easy to compute using backpropagation**.
- In some other applications, this forward model might be manually specified, and the gradient is actually analytically tractable in most cases.
- The nice thing of this decomposition is now we can plug in different forward models or exactly the same score model. Which means we only **need to train an unconditional score model once**.
- Then we can repurpose this unconditional score model for various conditional generation applications just by switching the forward model.

[Improved generation - Inverse Distribution]

Improved generation

Controllable Generation: class-conditional generation

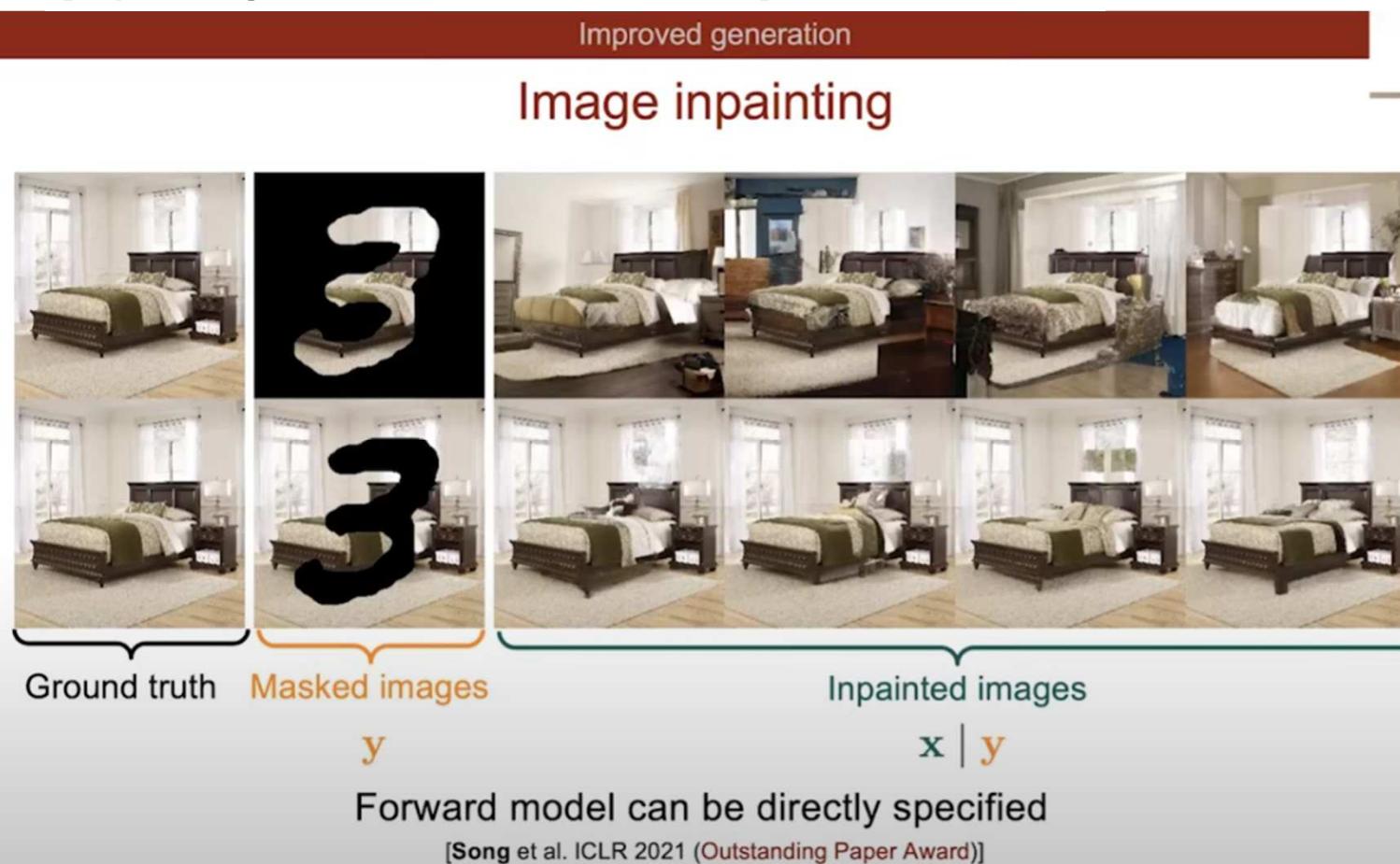
- y is the **class label**
- $p_t(y | x)$ is a time-dependent classifier (classifier-guidance)



[Song et al. ICLR 2021 (Outstanding Paper Award)]

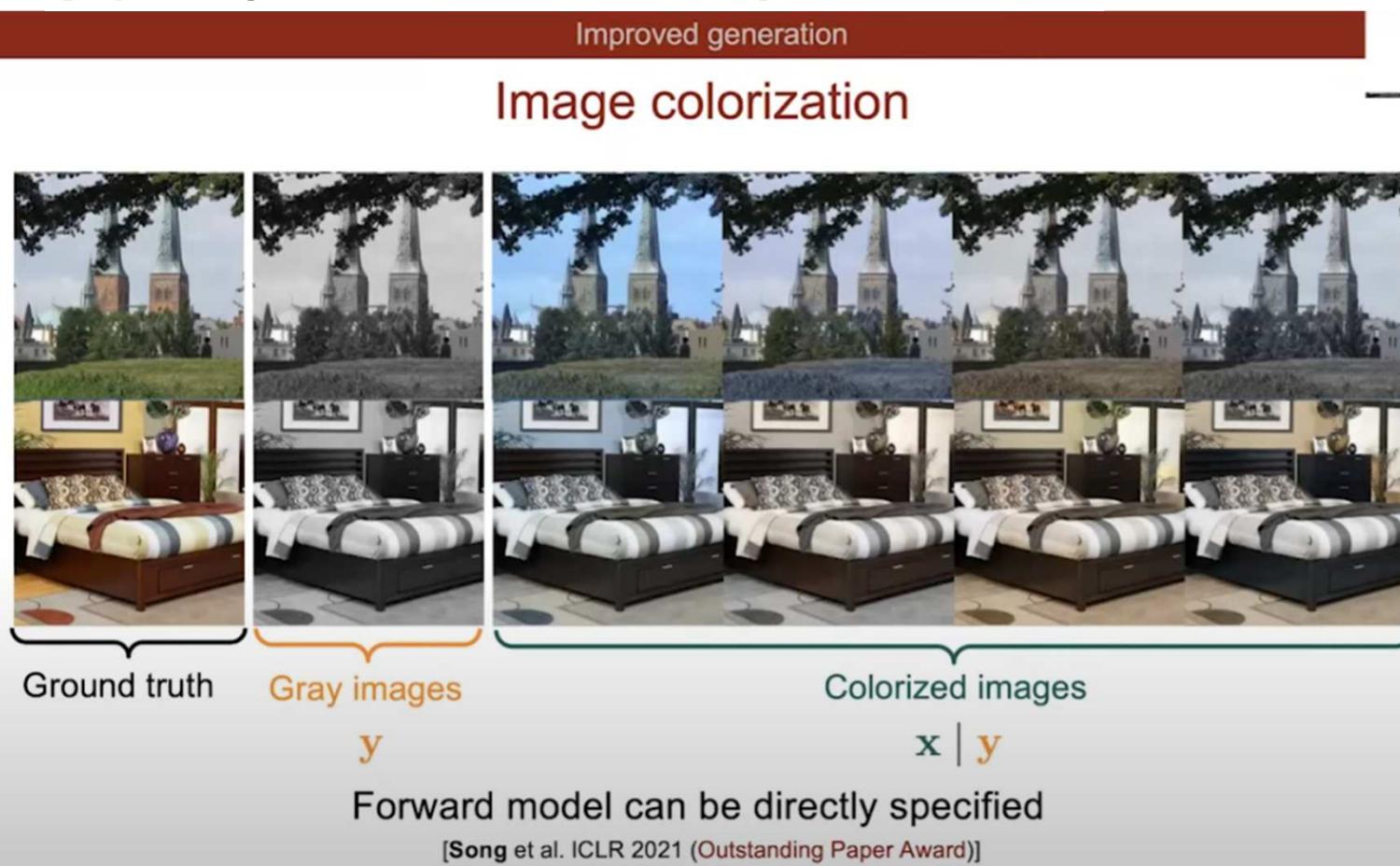
- This is one example.
- We can train **one unconditional score model** of CIFAR-10 images, then couple it with a **classifier** to generate **class conditional samples**.
- Here, **the forward model is the time-dependent classifier**. It is time dependent because we consider a **sequence of score functions**, and this means we need to have become a **sequence of classifiers**.
- So the figures demonstrates the **conditional generation results** of CIFAR-10 [INAUDIBLE] all from an **unconditional score-based model**.
- And this approach has to been further developed as **classifier guidance** or **classifier-free guidance** in subsequent works, and nowadays, it is the standard technique used in all the text-to-image generation approaches, like DALL-E 2 or Imagen.

[Improved generation - Inverse Distribution]



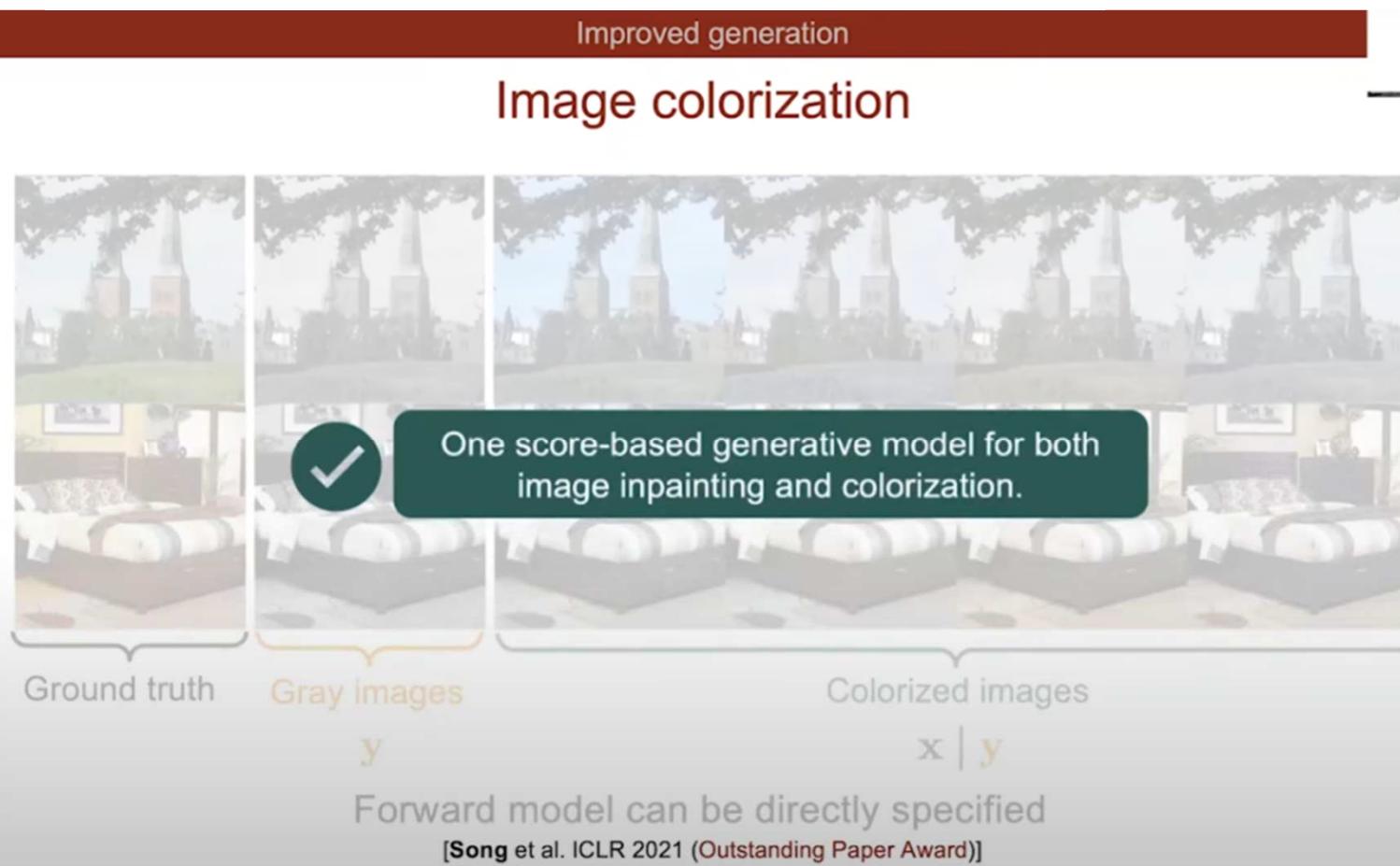
- We can use an unconditional score-based model for imaging painting.
- Here the **control signal** is the **masked image**.
- We only know some partial regions of the image.
- And we want to sample from the inverse distribution, which gives us completed images from a partially-observed image.
- In this case, the forward model can be directly specified using our domain expertise. So there is no need to train a separate model for this task.

[Improved generation - Inverse Distribution]



- Similarly, we can apply unconditional score-based models for image colorization.
- And again, in this figure you can see that our **control signal** is **the gray image**, and we can infer the colorized images from the gray images.
- Now, further models can be specified manually.
- For this image in painting and the image colorization tasks, we were actually using the same unconditional score model.

[Improved generation - Inverse Distribution]



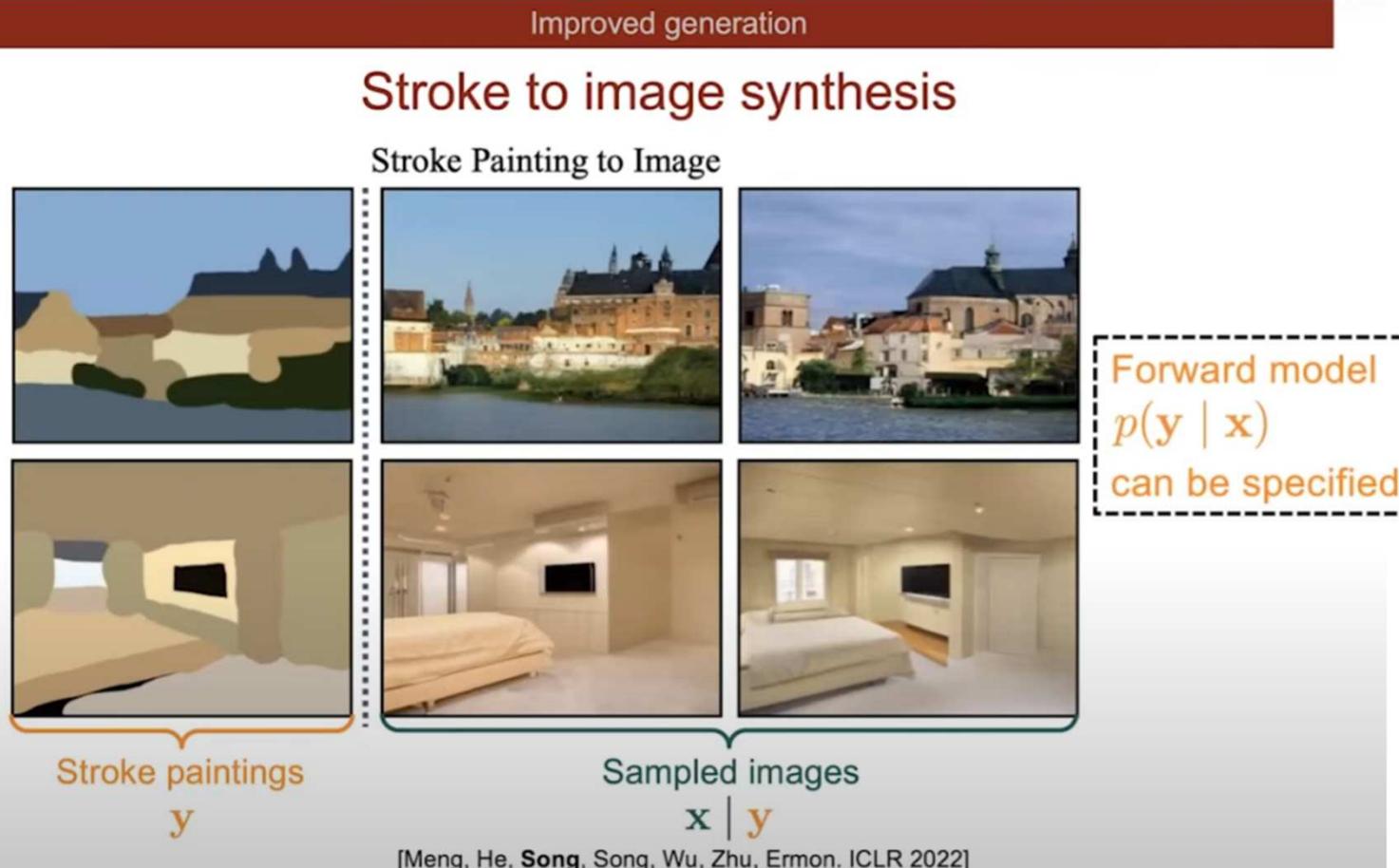
- So this means that **one score-based model** can be used for both **imaging painting and the colorization**, demonstrating the **flexibility** of this **decomposition of the inverse score function**.

[Improved generation - Inverse Distribution]



- And again, we can apply this approach to **larger-scale examples**, such as colorization for images from resolution 1,024 by 1,024.

[Improved generation - Inverse Distribution]



- So the same approach can be applied to convert stroke paintings to realistic images, and here is one example.
- Now, the **stroke paintings** become **the control signal**, and we use an unconditional score-based model trained on realistic images only.
- They have no idea of what a stroke painting looks like. We can **develop the forward model by manual specification** using our domain expertise.

[Improved generation - Inverse Distribution]

Improved generation

Language-guided image generation

y **x | y**

(Prompt)
Treehouse in the style of Studio Ghibli animation

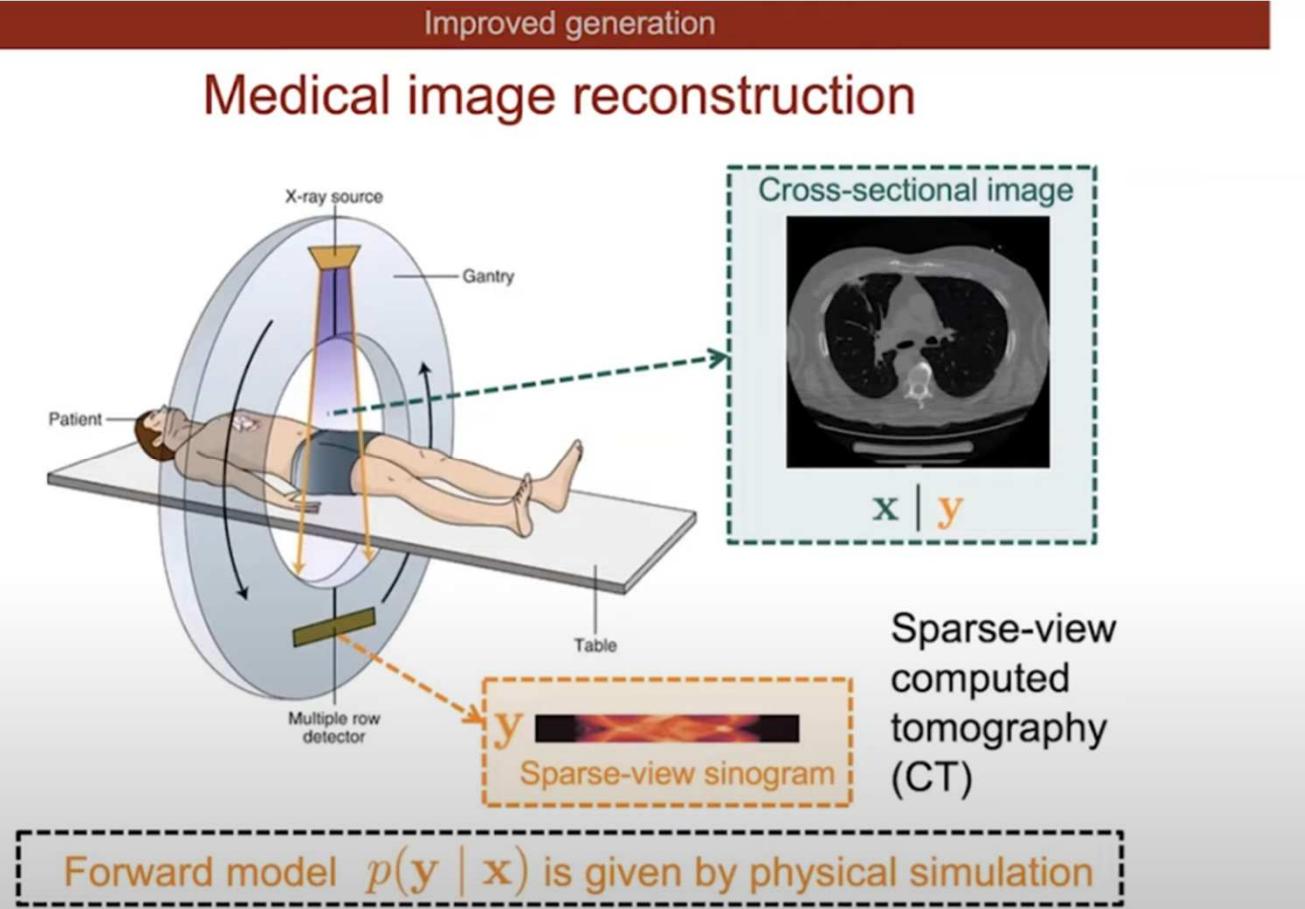
Forward model
 $p(y | x)$
is an image captioning neural network.



[Work by @danielrussruss]

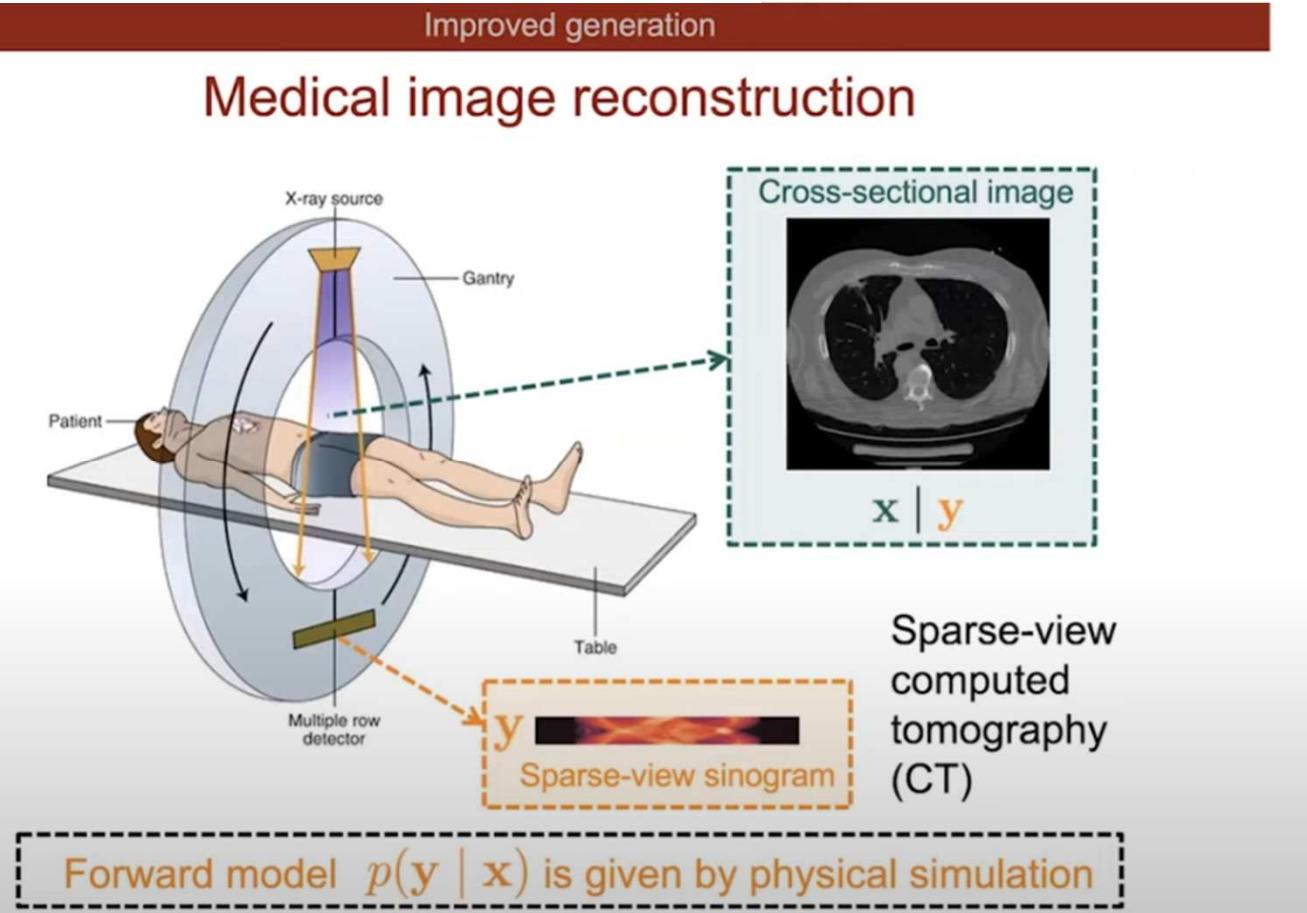
- And this is another example, language-guided image generation.
- In this case, we are based on an **unconditional score-based model** and **the control signal becomes a language description**— tree house in the style of Studio Ghibli animation.
- The forward model is given by an **image captioning neural network**.
- In this example, **the score model has no knowledge of language at all, but it is capable of generating spatial images that conform with the language description**.

[Conditional Score based Generation Example]



- We can apply conditional score-based generation for medical image reconstruction. We consider the special problem of computed tomography. In this case, we use X-rays to shoot through a human body.
- Those X-rays will hit the detector to form observations called sparse-view sonogram.
- We can invert this physical procedure to obtain those cross-sectional medical image.
- So here the **control signal** is a **sonogram**. The inverse distribution gives you the **conditional distribution of medical images given the sonogram**.
- We want to consider the problem of sparse-view computed tomography, meaning that we want to use as few X-rays as possible to reduce radiation.

[Conditional Score based Generation Example]



- This is a very simple task for generative models because **by training an unconditional generative model on large-scale medical images**, our generative models can **actually learn what a typical medical image looks like**.
- It can learn very useful image prior, and this can be subsequently used to reduce the number of X-ray projections.
- In this case, **the forward model** is given by **physical simulation**.
- So there is **no need to train any separate conditional model** on **capturing this forward model**.
- And we can have some results on real-world CT data sets.

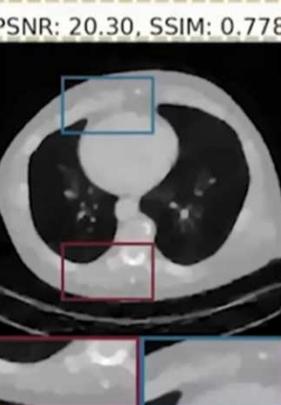
[Conditional Score based Generation Example]

Improved generation

Medical image reconstruction

Sparse-view CT (just 23 projections)

PSNR: 20.30, SSIM: 0.778



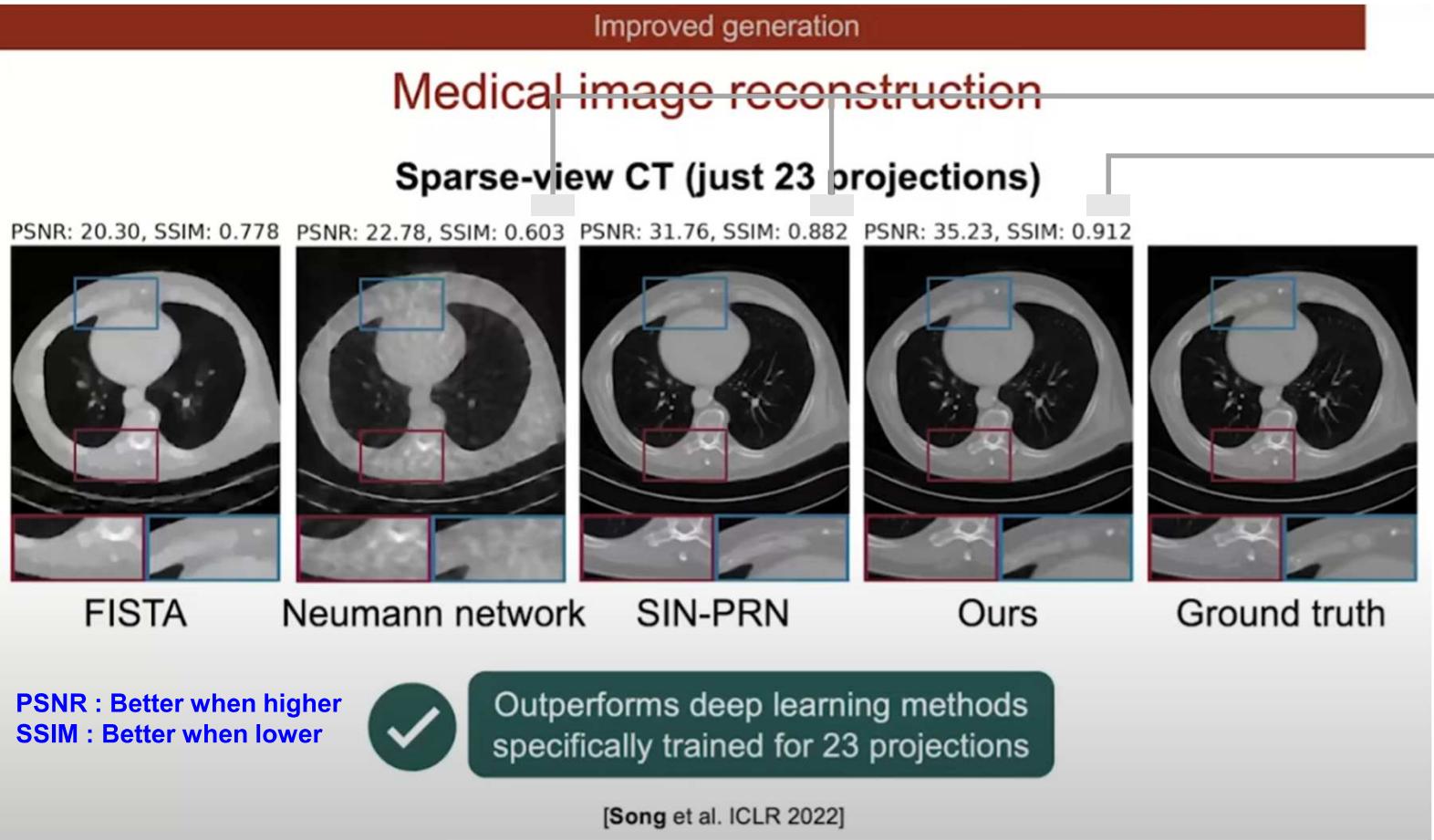
PSNR \uparrow : Better when higher
SSIM \downarrow : Better when lower

FISTA

[Song et al. ICLR 2022]

- We can have some results on real-world CT data sets.
- We consider the task of using 23 projections, while in contrast typical traditional approaches require hundreds to thousands of projections.
- FISTA : This is the result of a traditional approach based on compressed sensing.
- So using only 23 projections, you can see the medical image is quite blurry.
- Quantitatively, we compared the performance of different algorithms using PSNR and SSIM.

[Conditional Score based Generation Example]



- Neumann Network, SIN-PRN : Here are the results of two deep neural network-based approaches. So those methods are based on mapping projections directly to images. They are kind of limited to a particular training setting. In this case, since they are trained on 23 projections, it is hard to adapt them to a different number of projections later.
- Ours : This is our fully unsupervised approach. Because we only try one unconditional score-based model, we do not train any particular model associated with these 23 projections.
- So that means we can adapt the same model to different settings, like changing the number of projections later after training

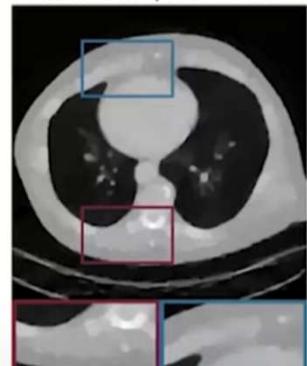
[Conditional Score based Generation Example]

Improved generation

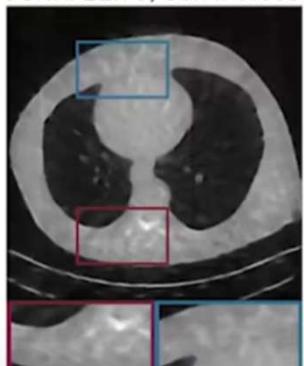
Medical image reconstruction

Sparse-view CT (just 23 projections)

PSNR: 20.30, SSIM: 0.778 PSNR: 22.78, SSIM: 0.603 PSNR: 31.76, SSIM: 0.882 PSNR: 35.23, SSIM: 0.912



FISTA



Neumann network



SIN-PRN



Ours



Ground truth

- Both qualitatively and quantitatively, we can see that **this score-based medical image reconstruction approach** can actually **outperformed** other deep learning methods.

- Even though this generative approach is fully unsupervised, it does not bind to a particular experimental setting. While in contrast, existing deep learning methods come to be limited to a specific experimental setting.

- So similar success has also been observed on accelerated magnetic resonance imaging as well.

PSNR : Better when higher
SSIM : Better when lowerOutperforms deep learning methods
specifically trained for 23 projections

Similar success on accelerated MRI

[Jalal et al., NeurIPS 2021]

[Chung & Ye, 2021]

[Song et al., ICLR 2022]

[Song et al. ICLR 2022]

[Conditional Score based Generation Example]

Improved generation

State-of-the-art performance on various other tasks

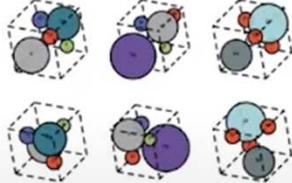
Image generation
[Dhariwal & Nichol, 2021]



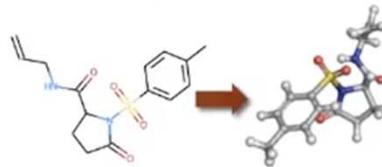
Text-to-speech generation
[Tae et al., 2021]



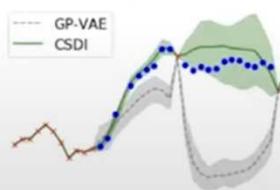
Material design
[Xie et al., 2021]



Molecular conformation prediction
[Xu, Yu, Song, et al., ICLR 2022]



Time series prediction
[Tashiro, Song, Song, Ermon, NeurIPS 2021]



Shape generation
[Cai et al., 2020]



<https://scorebasedgenerativemodeling.github.io/>

- There has been numerous developments of score-based models or diffusion models. We have obtained state-of-the-art performance on many other tasks. And this is already kind of outdated at this time, but I think it's worth mentioning anyways.
- So we can generate high-quality images for a much more complicated data set like imageNet.
- And we can obtain outstanding performance audio syntheses, text-to-speech generation, material design-- this is actually a paper by MIT researchers-- and also shape generation.
- We can also use score-based approaches for molecular confirmation prediction and time series prediction.
- And there is a website of score based generative modeling.github.io that includes a list of relevant works trying to build upon the technology of diffusion and trying to improve the methodology of score-based generative models.

[Probability Evaluation]

Score-based generative modeling: outline



Flexible models

- Bypass the normalizing constant
- Principled statistical methods

[Song et al. UAI 2019 oral]



Improved generation

- Higher sample quality than GANs
- Controllable generation

[Song & Ermon. NeurIPS 2019 oral]
[Song & Ermon. NeurIPS 2020]
[Song et al. ICLR 2021 oral]
(Outstanding Paper Award)
[Song et al. ICLR 2022]

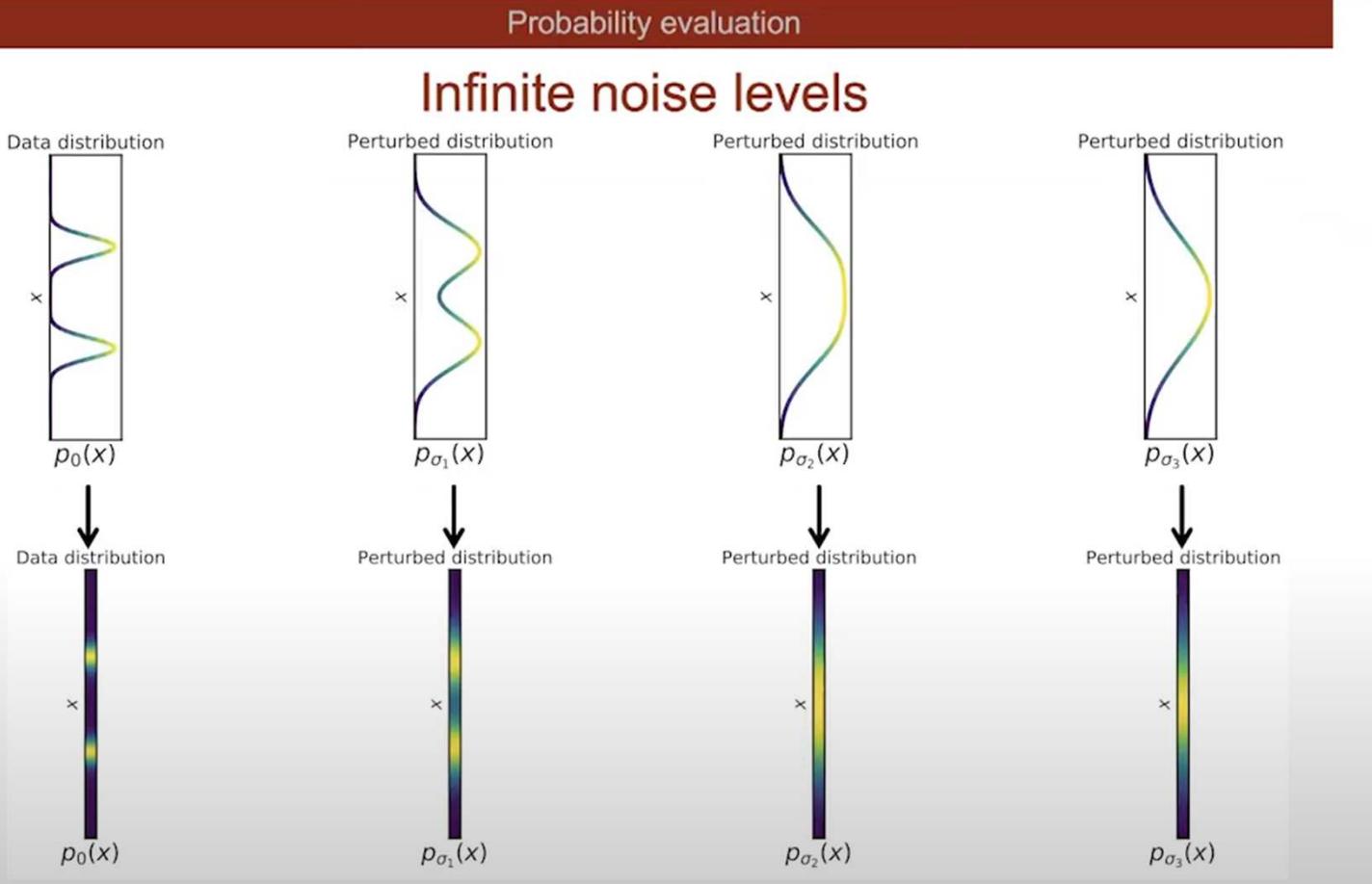
Probability evaluation

- Accurate probability evaluation
- Better estimation of data probabilities

[Song et al. ICLR 2021 oral]
[Song et al. NeurIPS 2021 spotlight]

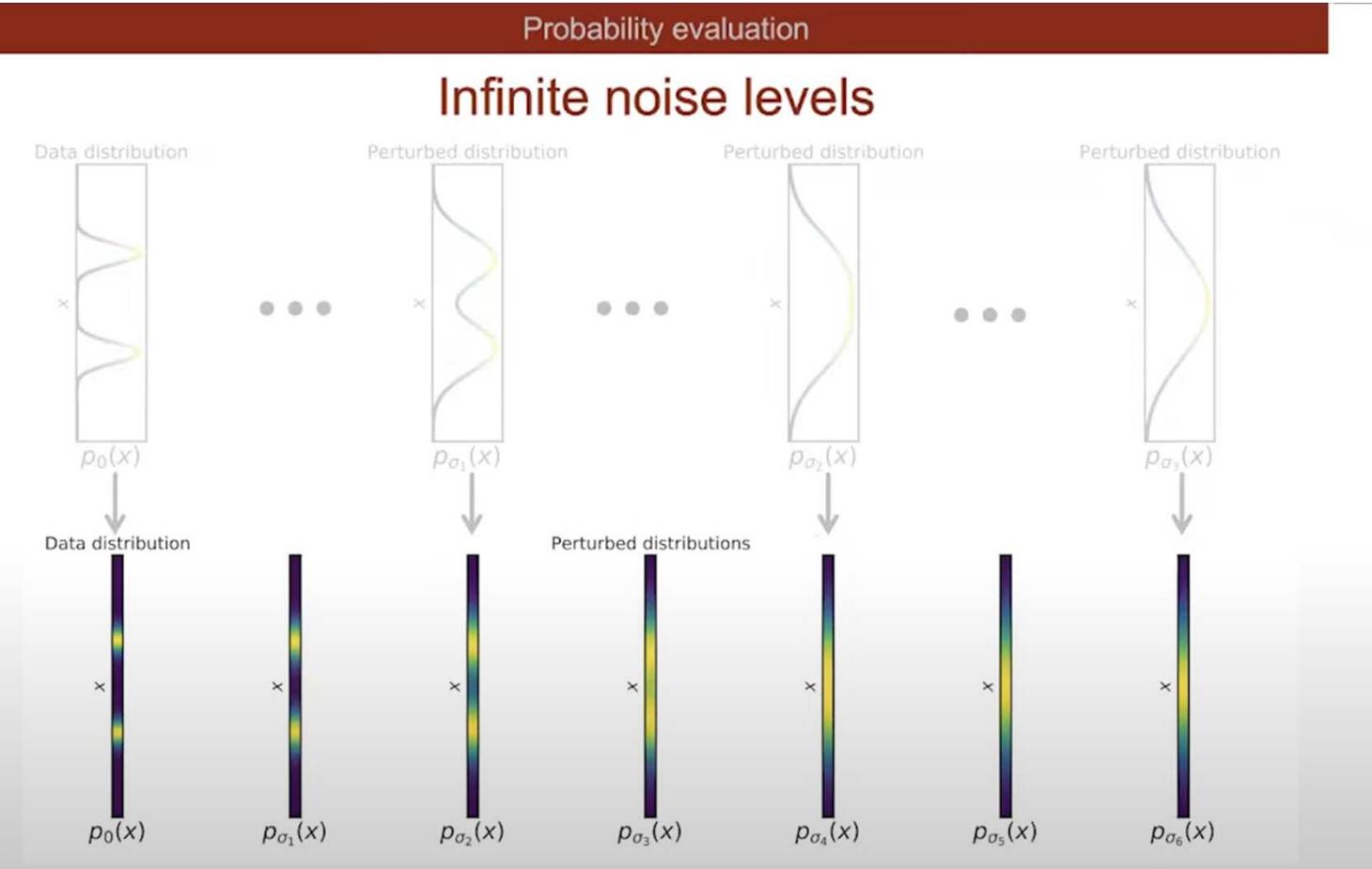
- Now I am talking about how score-based generative modeling allows **flexible model architectures**, allows **improved sample quality**, with a controllable generation procedure.
- In the last part of a tutorial, I will talk about **how we can compute probability values accurately**, and **how we can outperform existing likelihood-based generative models in terms of density estimation**.

[Probability Evaluation]



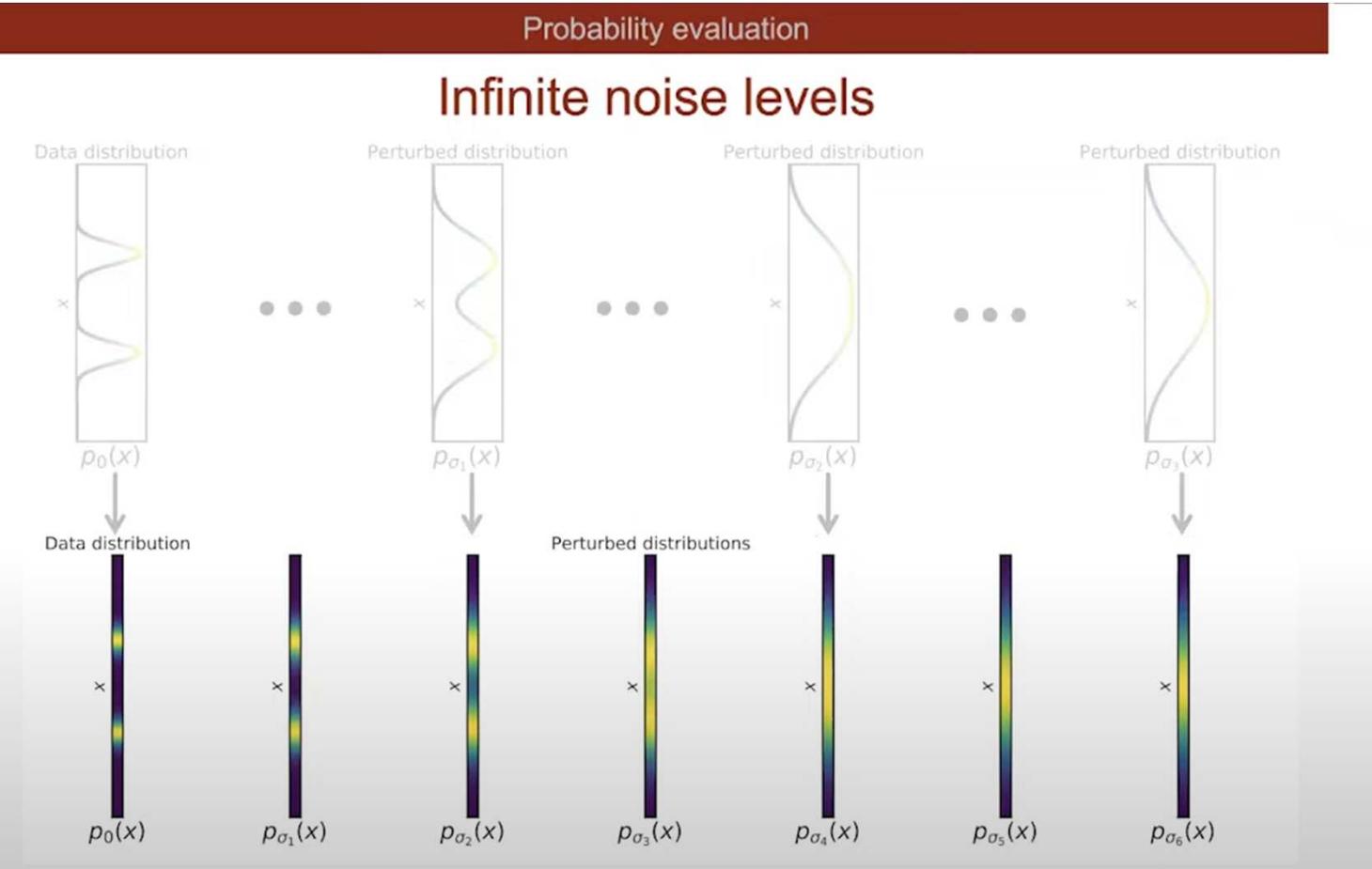
- So in order to **compute accurately probability values**, we have to **generalize the previous framework** from **using a finite number of noise levels** to **using an infinite number of noise levels**.
- Let's get some intuition first by assuming our data distribution is a one-dimensional mixture of two Gaussians.
- Let's start **with three noise levels**. We have sigma one to sigma three. And we use Gaussian noise of a standard deviation from sigma one to sigma three to perturb our data distribution.
- So if the noise level is large enough, we can convert any data distribution into a simple Gaussian distribution.
- We may use a one-dimensional heat map to represent each of those noisy data densities.

[Probability Evaluation]



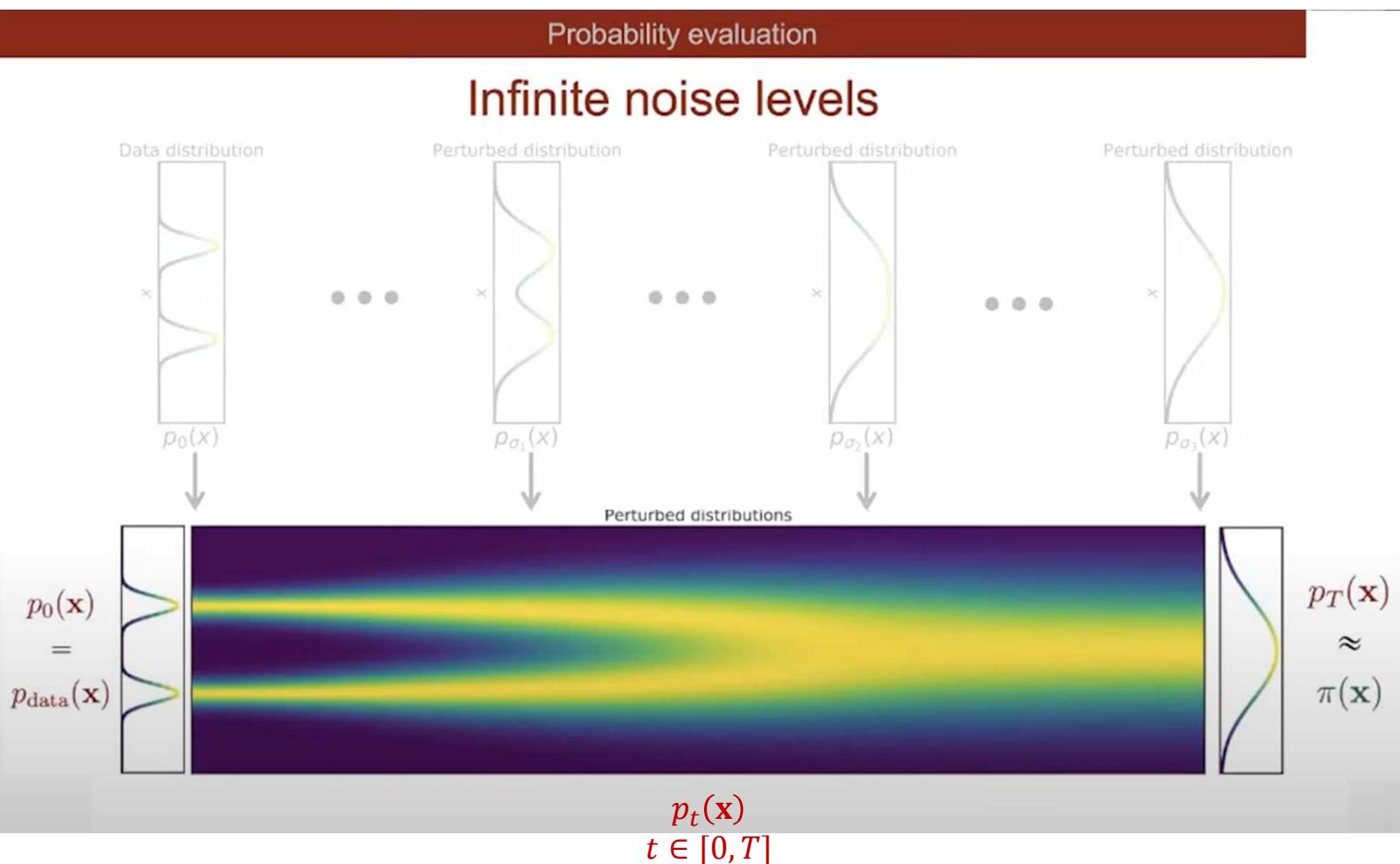
- With more noise levels, we have more heat maps.

[Probability Evaluation]



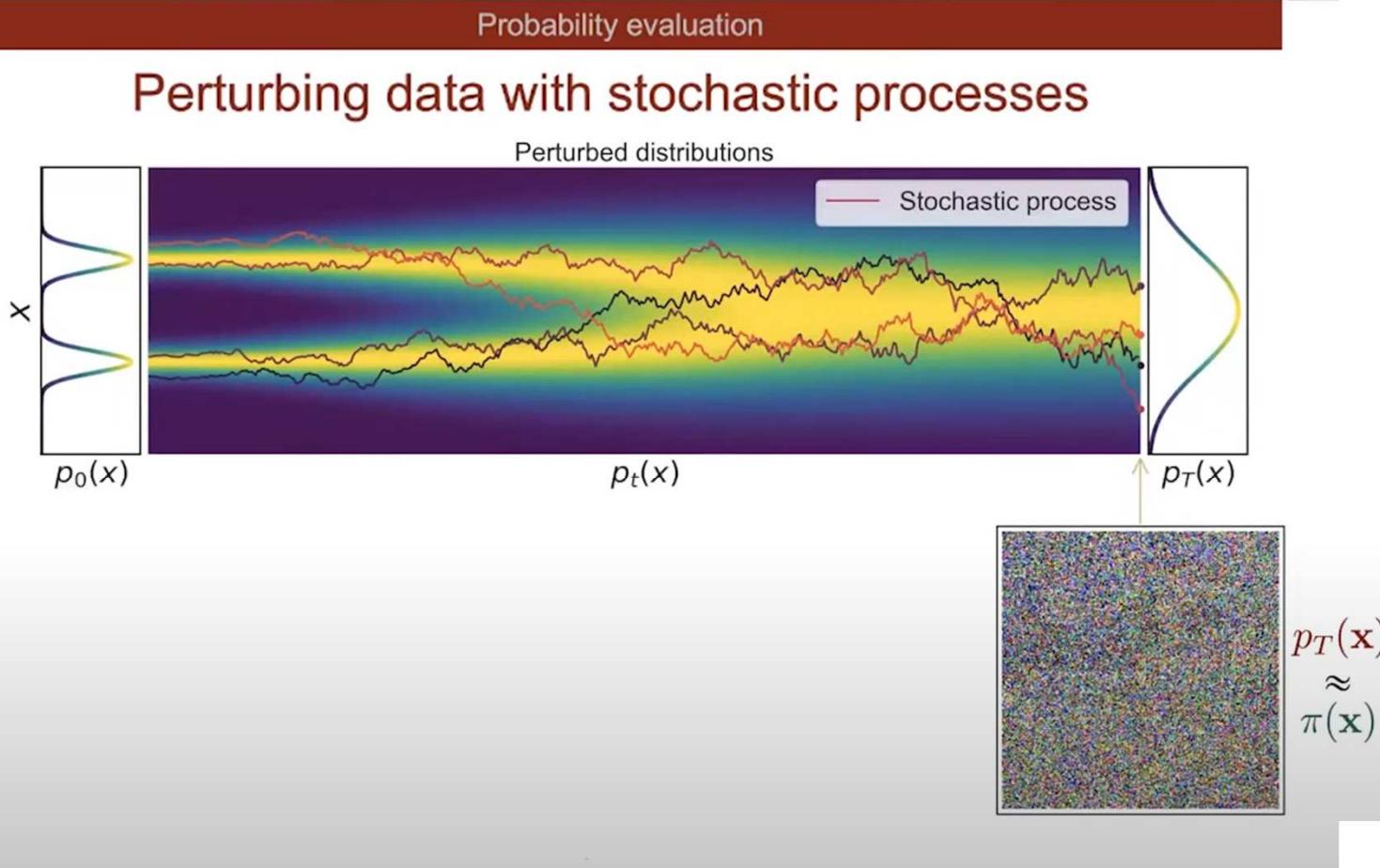
- With more noise levels, we have more heat maps.

[Probability Evaluation]



- In the limit of the infinite noise levels, we have a continuous, two-dimensional heat map that represents an infinite number of noisy data densities.
- We use p_t to represent each of those noisy data densities, where t is a continuous parameter ranging between 0 and capital T , where capital T , capital T is the fixed constant.
- When t 's at 0, t_0 is the same of the data density because we do not inject any Gaussian noise and this time instance
- When t is capital T , the capital T contains a lot of Gaussian noise.
- It would be close a simple Gaussian distribution, which we denote as $\pi(x)$.

[Probability Evaluation]

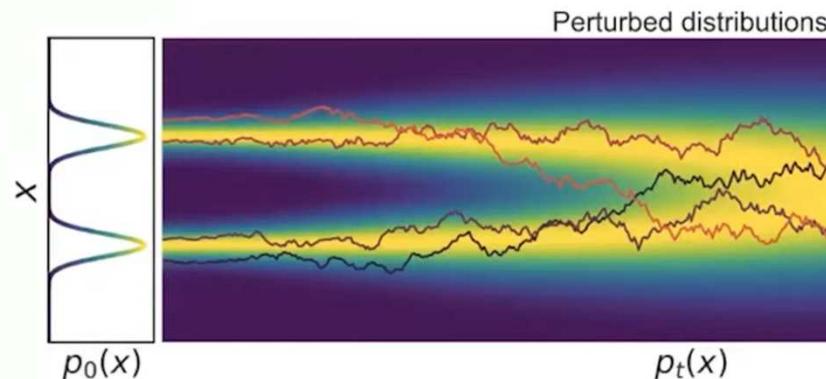


- So suppose we are given this **sequence of infinite number of noise levels**.
- **How do we generate noisy data sets for training our score models?**
- Well, we need to leverage the intuition or **stochastic processes**.
- We progressively inject the Gaussian to perturb our train data sets.
- So after enough perturbation, eventually we will obtain very noisy images which are close to samples from a simple Gaussian distribution.
- **So the trajectory of those noisy data sets form the trajectories of a stochastic process.**

[Probability Evaluation]

Probability evaluation

Perturbing data with stochastic processes



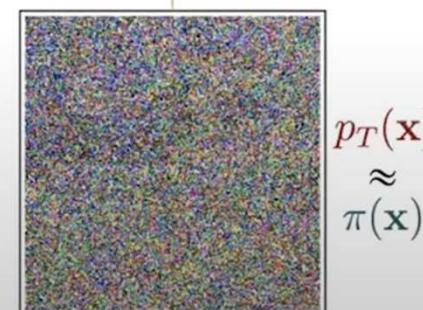
Stochastic process

$$\{\mathbf{x}_t\}_{t \in [0, T]}$$



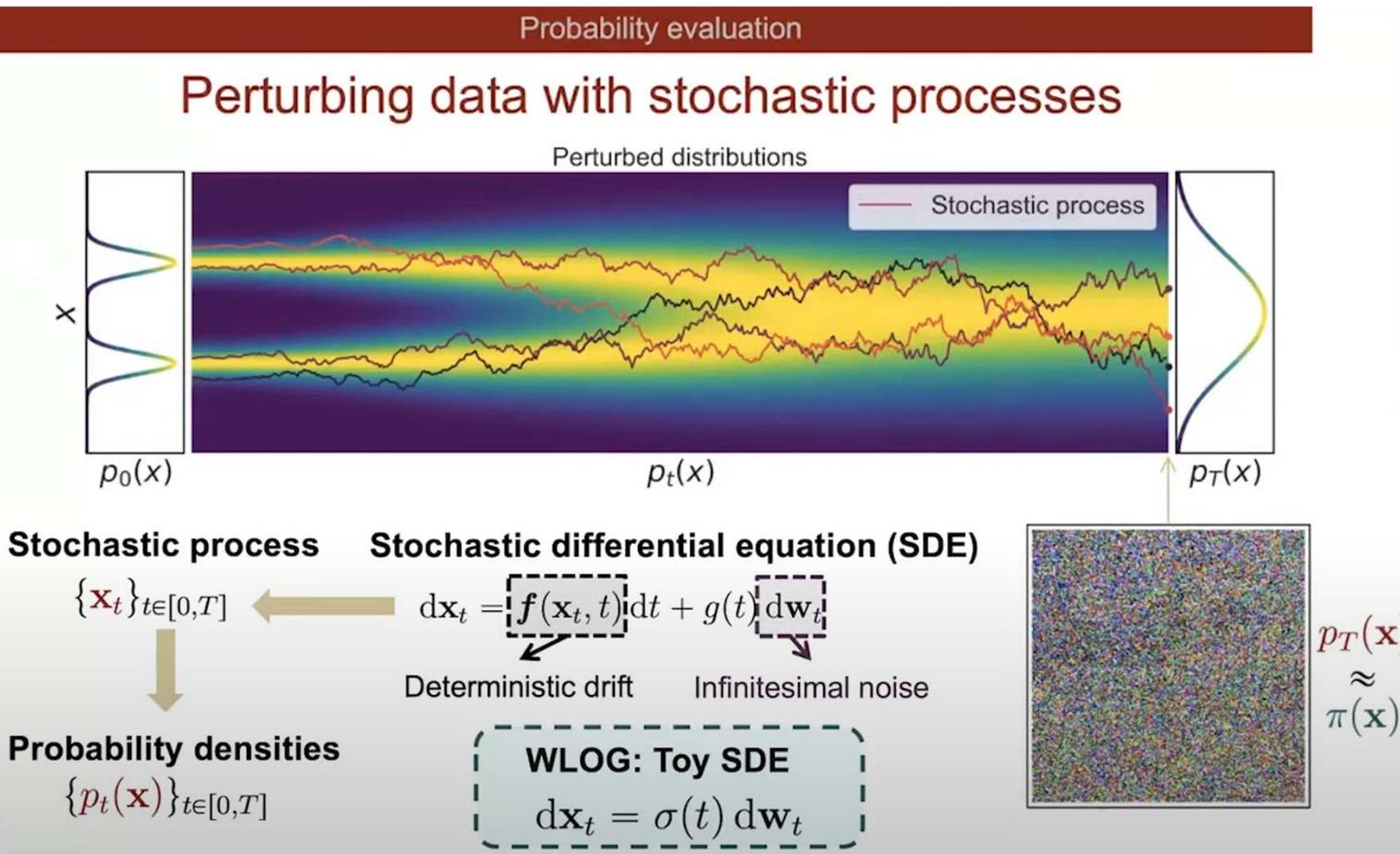
Probability densities

$$\{p_t(\mathbf{x})\}_{t \in [0, T]}$$



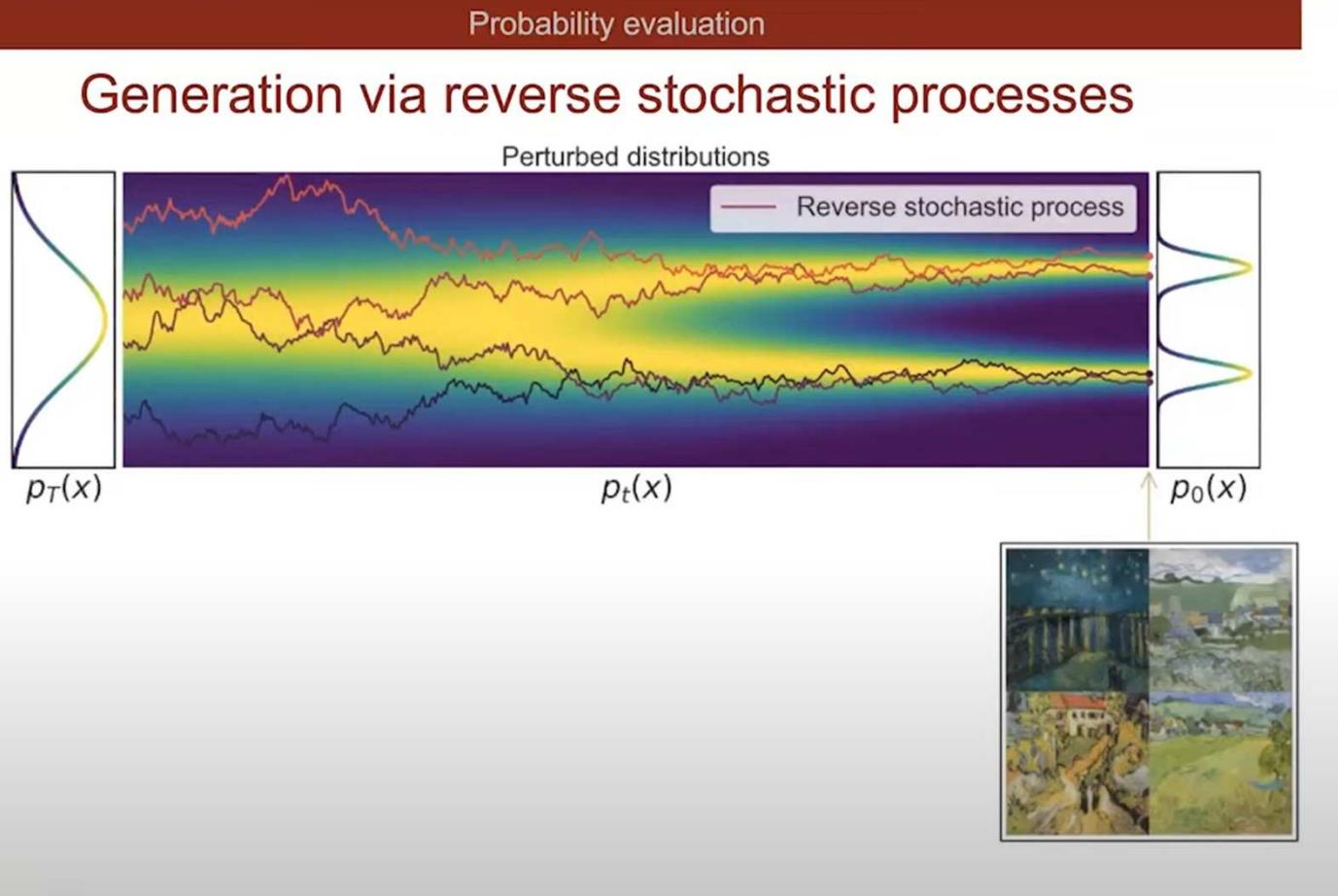
- A stochastic process is basically a **collection of a infinite number of random variables**.
- Here those random variables are indexed by the continuous parameter t .
- For each random variable, there will be a **corresponding probability density**. So one stochastic process corresponds to an **infinite number of probability densities**.
- So how do we choose the right stochastic process such that it represents an infinite number of noisy data densities?
- Well, we use the term of a stochastic differential
- A stochastic process is basically a **collection of a infinite number of random variables**.
- Here those random variables are indexed by the continuous parameter t .

[Probability Evaluation]



- Well, we use **the term of a stochastic differential equation**. A stochastic differential equation is very similar to an ordinary differential equation, but it has one additional stochastic term.
- In the general form of SDEs, we have **one deterministic drift term** that **controls the deterministic properties of the stochastic process**.
- We have **one stochastic term** which involves **dwt**, the infinitesimal **Gaussian noise**.
- So without loss of generality, we consider the following toy formulation of the SDE which does not have the deterministic drift term. And it has a very simple stochastic term, sigma t.
- We can use **sigma t** as a continuous time generalization of noise table **sigma i**, which we introduce it before.

[Probability Evaluation]

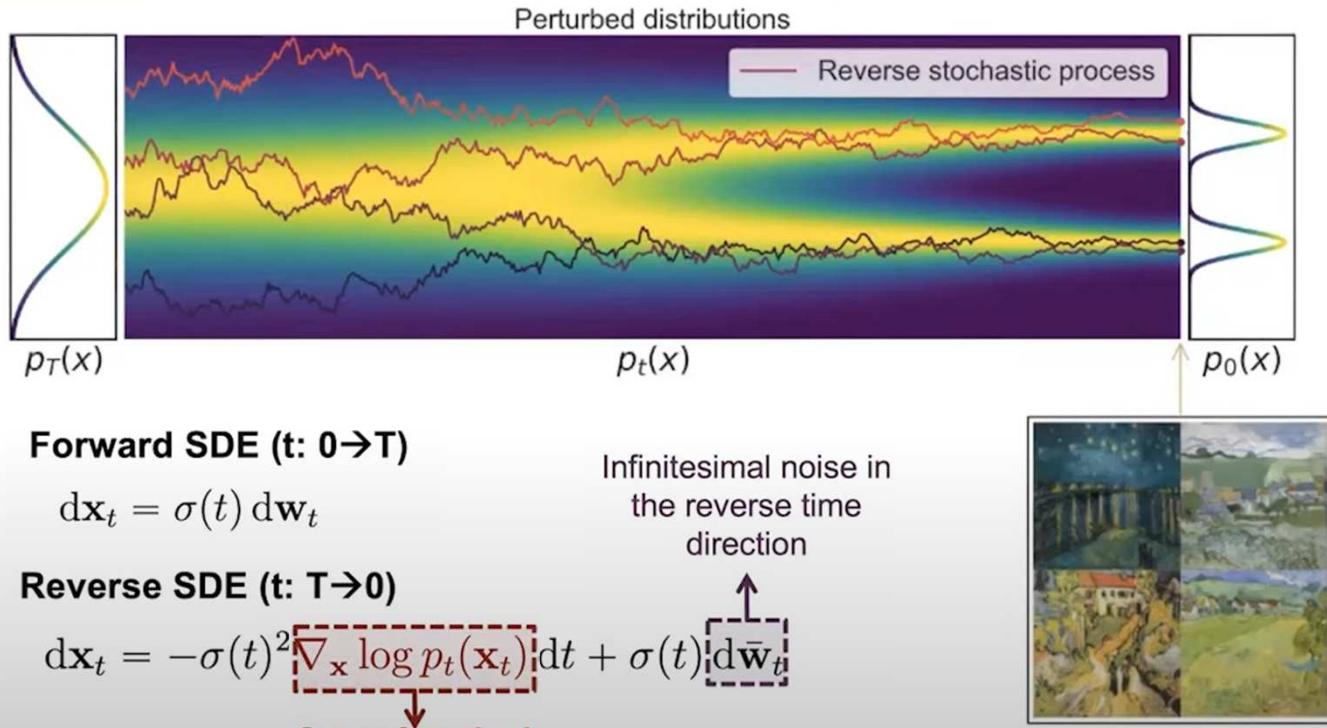


- Now we have an infinite number of noisy data sets. How do we generate samples, suppose we can estimate their score functions?
- Well, the **sample generation process amounts to a time reversal of the perturbation process.**
- By reversing the perturbation process, we can start from Gaussian noise, then **progressively denoise** to generate noise free samples.
- How do we obtain this reverse stochastic process?

[Probability Evaluation]

Probability evaluation

Generation via reverse stochastic processes

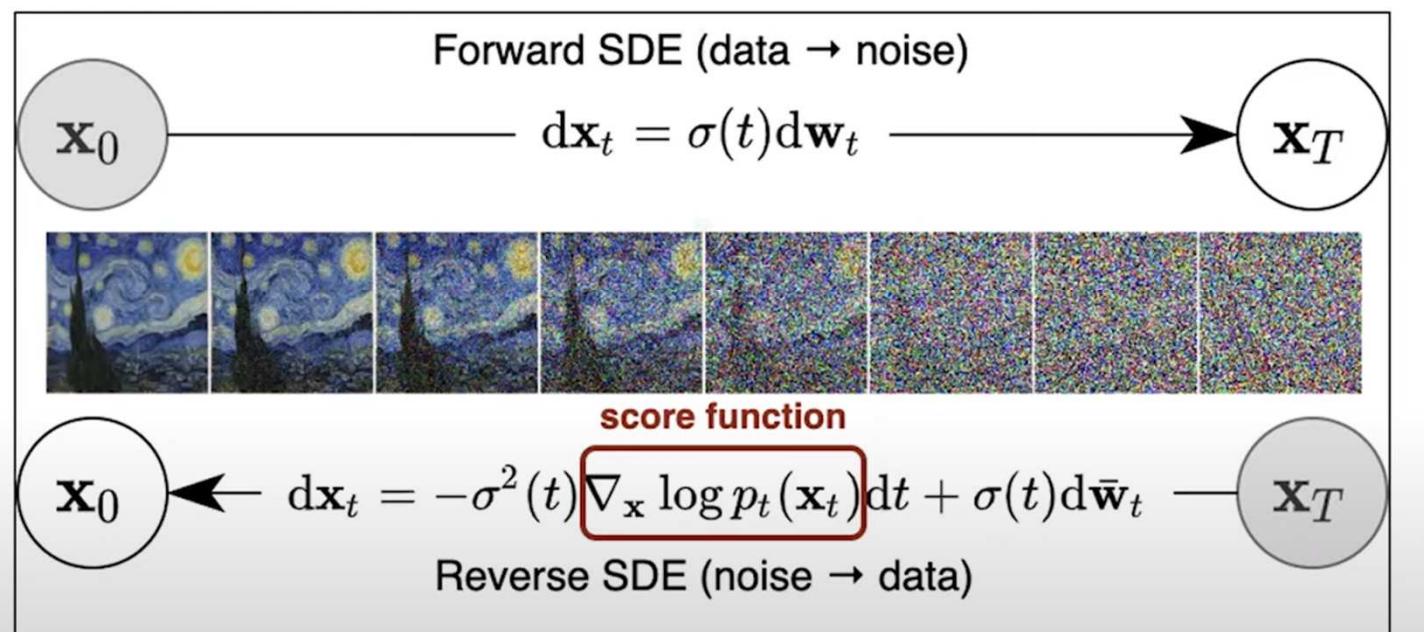


- Recall that our forward stochastic process is given by a stochastic differential equation.
- It turns out that any stochastic differential equation of that form can be reversed in analytical form, and this gives us **the reverse stochastic differential equation**.
- The reverse stochastic differential equation depends on an infinitesimal noise term, $d\bar{w}_t$.
- So this is very similar to dwt , but it maxes only when time flows backwards.
- It also depends on a score function of the noisy data density, p_t .

[Probability Evaluation]

Probability evaluation

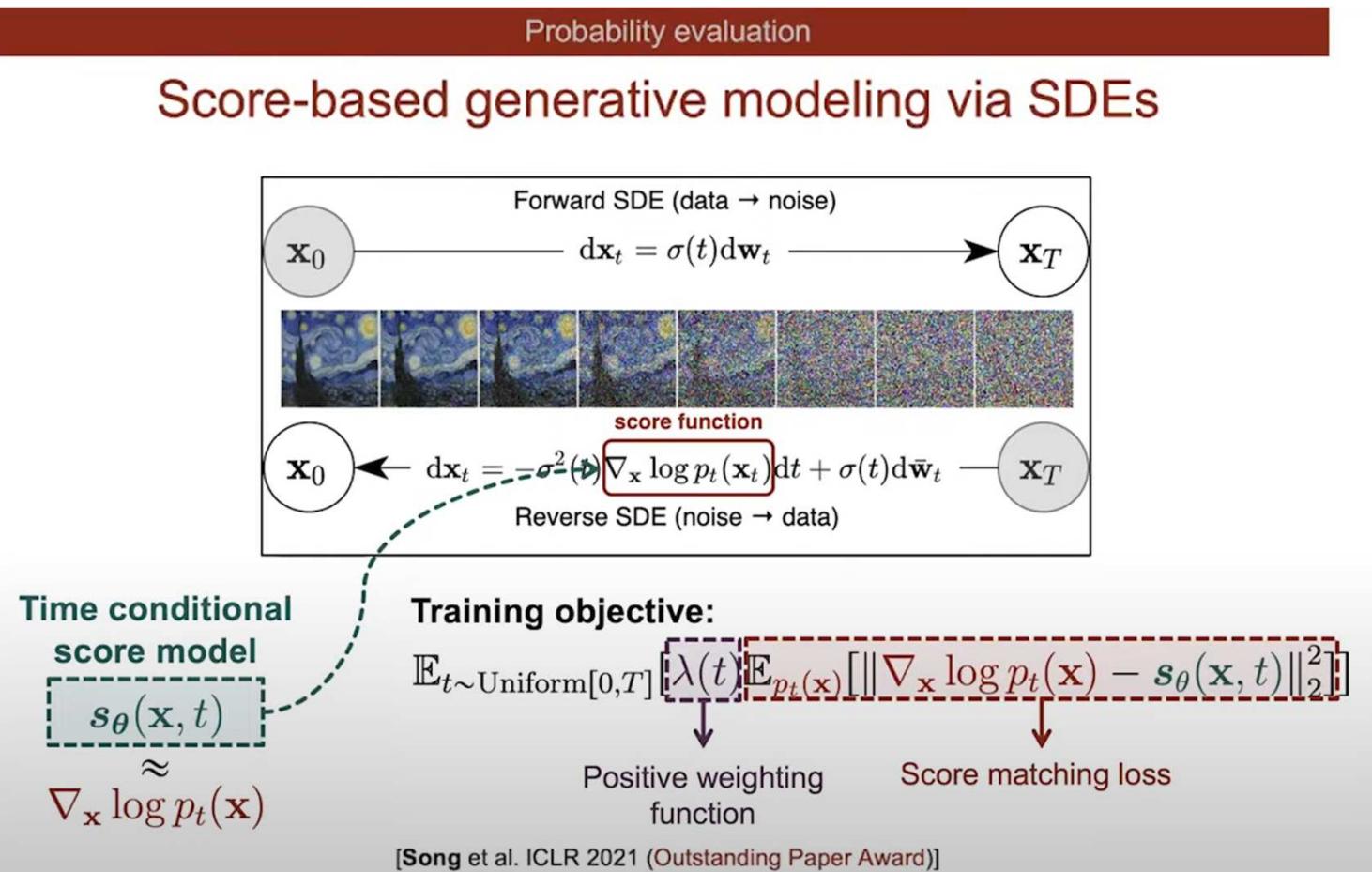
Score-based generative modeling via SDEs



- So now with the forward and backward SDEs, we can generalize the previous scope as the generative modeling approach for using-- yielding noise levels.

[Song et al. ICLR 2021 (Outstanding Paper Award)]

[Probability Evaluation]



- In this formulation, **the key is to estimate the score function**, which we accomplish by modeling the time--by **parameterizing the time-conditional score model**.
- We hope to **train this time-conditioned model to approximate a function of the data density and time instant, t**.
- And again, **the training procedure depends on score matching**.
- We have **one score matching loss for any time instant, t**.
- We have a positive working function, **lambda t**, to **balance the optimization procedure**.
- And we have a generalized summation to an expectation over t.

[Probability Evaluation]

Probability evaluation

Score-based generative modeling via SDEs

- Time-dependent score-based model

$$\mathbf{s}_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$$

- Training:

$$\mathbb{E}_{t \in \mathcal{U}(0, T)} [\lambda(t) \mathbb{E}_{p_t(\mathbf{x})} [\|\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x}, t)\|_2^2]]$$

- Reverse-time SDE

$$d\mathbf{x} = -\sigma^2(t) \mathbf{s}_\theta(\mathbf{x}, t) dt + \sigma(t) d\bar{\mathbf{w}}$$

- Euler-Maruyama (analogous to Euler for ODEs)

$$\mathbf{x} \leftarrow \mathbf{x} - \sigma(t)^2 \mathbf{s}_\theta(\mathbf{x}, t) \Delta t + \sigma(t) \mathbf{z} \quad (\mathbf{z} \sim \mathcal{N}(\mathbf{0}, |\Delta t| \mathbf{I}))$$

$$t \leftarrow t + \Delta t$$

Song, Sohl-Dickstein, Kingma, Kumar, Ermon, Poole. "Score-Based Generative Modeling through Stochastic Differential Equations." ICLR 2021.

- After training with this score mentioned objective, we obtain a **good time-conditional score-based model** that **approximates the score function of noisy data densities**.

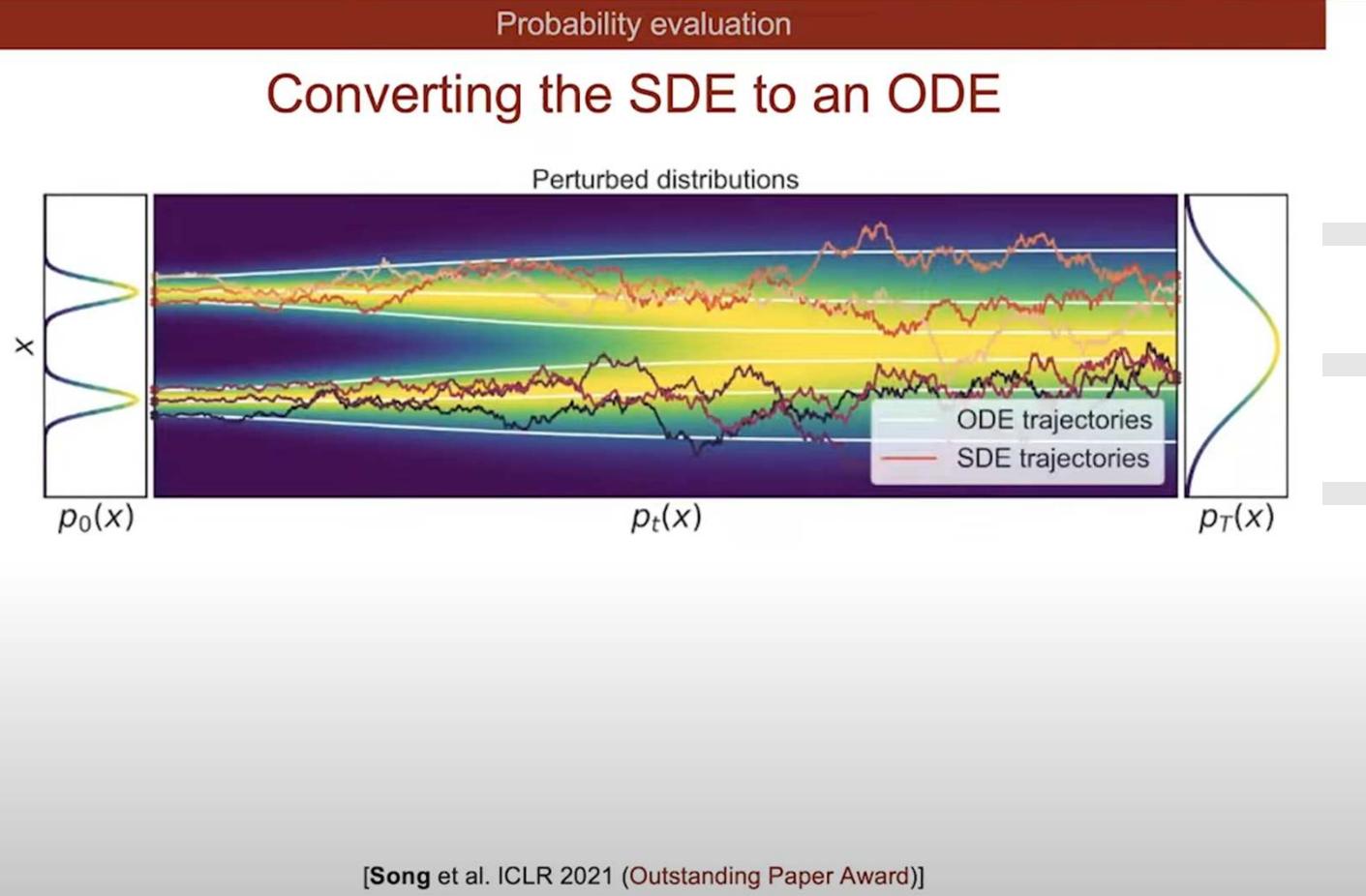
- And of course training involves **minimizing the score mentioned objective**, and we can train it efficiently using the denoising [INAUDIBLE] or [INAUDIBLE]..

- After training, we can **plug our time-conditional score model** into the reverse time SDE.

- Then, we can use **any numerical SD solver** to **solve this reverse time SDE for sample generation**.

- And one simple approach is the **Euler-Maruyama approach**, which is **stochastic generalization to the classical Euler solver for ordinary differential equations**.

[Probability Evaluation]



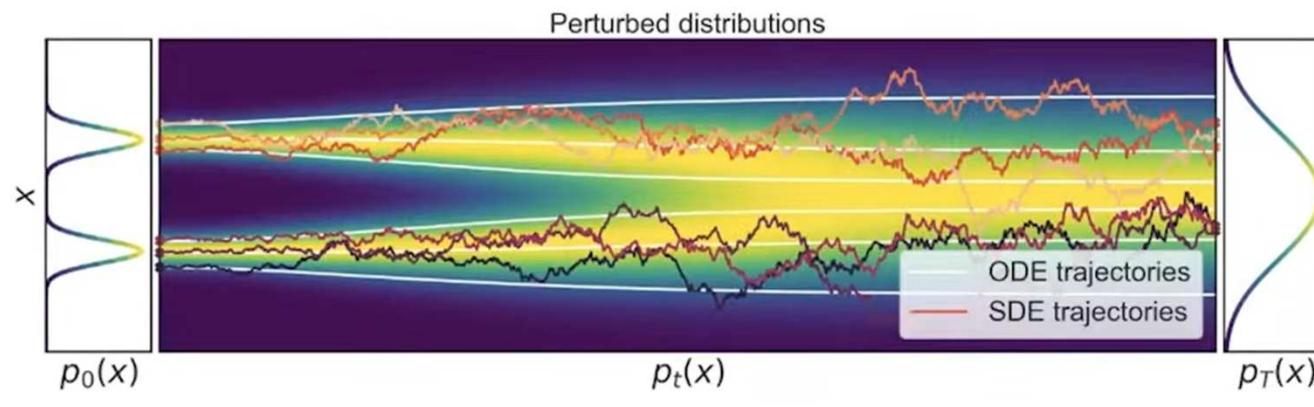
- With this **continuous SDE approach**, we not only **improve empirical performance**, we can but also finally discuss how we can **compute the accurate probability values**. And this requires to **convert the stochastic differential equation to an ordinary differential equation**.
- So with the right SDE, we can convert any data distribution to the Gaussian distribution.
- It turns out we can do the same thing by using **ordinary differential equations**.
- The **trajectory** of the **ODE** and the **SDE** look quite different from each other, but **actually they can share the same set of marginal densities** that's given by the background.

[Song et al. ICLR 2021 (Outstanding Paper Award)]

[Probability Evaluation]

Probability evaluation

Converting the SDE to an ODE

**SDE**

$$dx_t = \sigma(t) dw_t$$

**Ordinary differential equation
(probability flow ODE)**

$$\frac{dx_t}{dt} = -\frac{1}{2}\sigma(t)^2 \nabla_x \log p_t(x_t)$$

Score function
 $\approx s_\theta(x, t)$

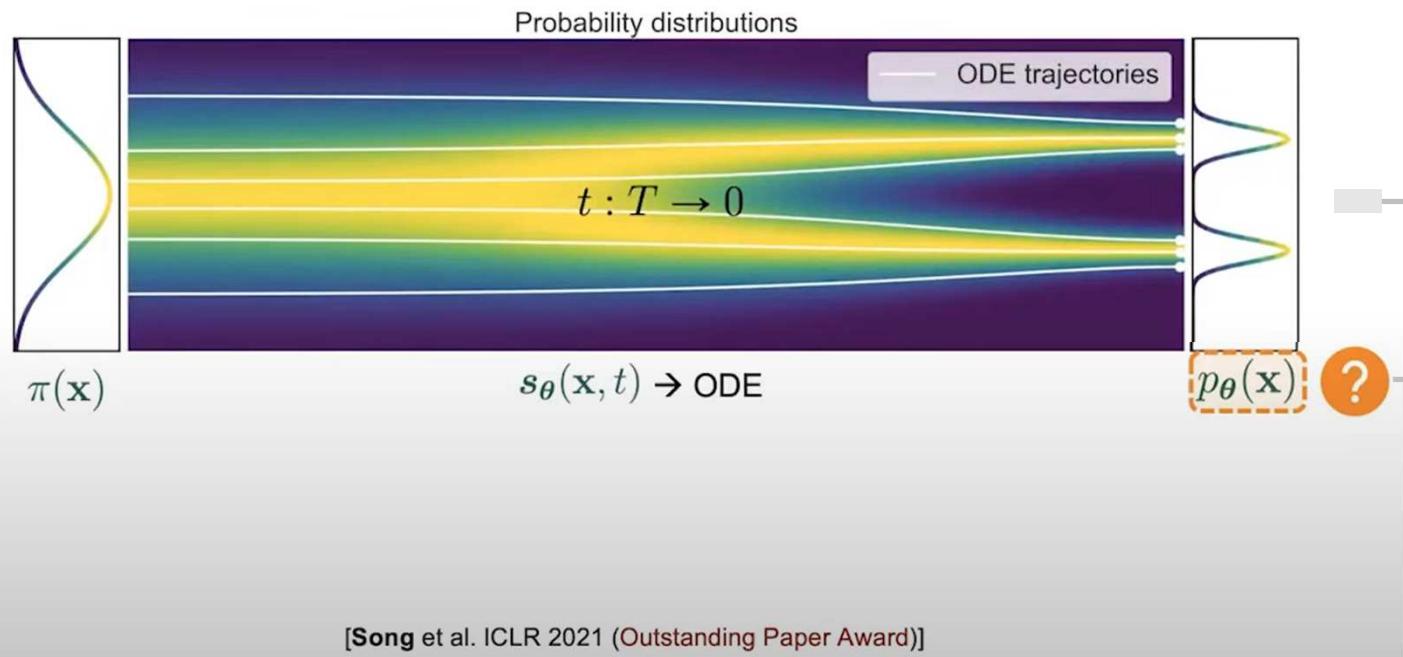
[Song et al. ICLR 2021 (Outstanding Paper Award)]

- So for any SDE of this form, we show that the corresponding ordinary differential equation, named **probability flow ODE**, has a form on the right side.
- So again, **this ODE only relies on the score function**.
- And since we have the time-conditional score model, we can plug it into the ODE, and then we can solve the ODE in various ways.

[Probability Evaluation]

Probability evaluation

Evaluating the probabilities with ODEs



- So after substituting **this model** into the **ODE**, we can solve the probability flow, or the bank crossing time bar, starting from some samples from the Gaussian distribution.
- This **ODE trajectory** gradually **converts our Gaussian vectors into high quality image samples**.
- And we **did not** use the resulting **distribution of samples from this ODE solver as p data**.
- So now, I will show you **how to compute the accurate value of p data**.

[Probability Evaluation]

Probability evaluation

Computing the exact likelihood with ODEs

Likelihood:

$$\pi(\mathbf{x}) \xrightarrow[t : T \rightarrow 0]{s_{\theta}(\mathbf{x}, t)} p_{\theta}(\mathbf{x})$$

Applications of likelihood:

- Lossless compression (Witten et al. 1987, Townsend et al. 2019)
- Unsupervised anomaly detection (Pimental et al. 2014, Song et al. 2018)
- Generative classification (Ng & Jordan 2002, Zimmermann, Schott, Song, et al. 2021)
- Density estimation (Silverman, 1986)

- Recall that we define this probability distribution in this way. We have **prior Gaussian distribution**. We have **time-conditional score model**.

- By **solving the ODE**, we get the **distribution p data**, so this is actually applications.

- So why we need to **compute the exact likelihood**? Because they have many useful applications. We also mentioned this briefly before. This includes lossless compression, unsupervised anomaly detection, generative classification, density estimation, and so on.

[Probability Evaluation]

Probability evaluation

Computing the exact likelihood with ODEs

Likelihood:

$$\pi(\mathbf{x}) \xrightarrow[t : T \rightarrow 0]{s_{\theta}(\mathbf{x}, t)} p_{\theta}(\mathbf{x})$$

Applications of likelihood:

- Lossless compression (Witten et al. 1987, Townsend et al. 2019)
- Unsupervised anomaly detection (Pimental et al. 2014, Song et al. 2018)
- Generative classification (Ng & Jordan 2002, Zimmermann, Schott, Song, et al. 2021)
- Density estimation (Silverman, 1986)

Probability flow ODE allows exact likelihood computation:

$$\log p_{\theta}(\mathbf{x}_0) = \log \pi(\mathbf{x}_T) - \frac{1}{2} \int_0^T \sigma(t)^2 \text{trace}(\nabla_{\mathbf{x}} s_{\theta}(\mathbf{x}, t)) dt$$

ODE Solver
Unbiased Estimator

(Change-of-variable formula for ODEs, Chen et al. 2018)
 [Song et al. ICLR 2021 (Outstanding Paper Award)]

- A formula for this likelihood is given by the following equation.
- This equation connects **log p** data--any data point \mathbf{x}_0 --with the **log prior distribution**, **log pi**, and also **one-dimensional integral** that involves the choice of the **Jacobian of the score model**.
- The **trace** can be computed using an **unbiased estimator**.
- And the **integral** can be computed using an **ODE solver**. This integral is simple to evaluate, because it is a one-dimensional integral.

[Probability Evaluation]

Probability evaluation

Solving ODEs for Sampling

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)	$\leq 3.70^*$	13.51
DDPM (L_{simple}) (Ho et al., 2020)	$\leq 3.75^*$	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

models trained
with score matching

- Here are some results of computing the density with this ODE approach. So our results are highlighted with the green box.
- And we report results in negative log likelihoods, which are better when lower.
- In this table, you can see that we can achieve lower negative log likelihood on almost all previous approaches, even though our methods are not explicitly trained for maximum likelihood.

black-box ODE
Solvers for sampling

[Song et al. ICLR 2021 (Outstanding Paper Award)]

[Probability Evaluation]

Probability evaluation

Efficient maximum likelihood training

Theorem (informal): Connection between the Kullback-Leibler (KL) divergence and score matching.

$$\text{KL}(p_{\text{data}} \parallel p_{\theta}) \leq \frac{1}{2} \mathbb{E}_{t \sim \text{Uniform}[0, T]} [\sigma(t)^2 \mathbb{E}_{p_t(\mathbf{x})} [\|\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) - s_{\theta}(\mathbf{x}, t)\|_2^2]] \\ + \boxed{\text{KL}(p_T \parallel \pi)} \approx 0$$

- I mentioned that the weighting function on the t can be chosen using some theoretically principled approach. And indeed we can do that to specifically maximize maximum likelihood.
- So there is a theorem we shouldn't-- there is an important connection between the KL divergence and the score matching objective, and this connection looks like below.
- Here the second term, KL divergence from p capital T to π , is approximately 0 if capital T is large enough. This term does not affect optimization, because it does not depend on model parameter theta.

[Song et al. NeurIPS 2021 (spotlight)]

[Probability Evaluation]

Probability evaluation

Efficient maximum likelihood training

Theorem (informal): Connection between the Kullback-Leibler (KL) divergence and score matching.

$$\text{KL}(p_{\text{data}} \parallel p_{\theta}) \leq \frac{1}{2} \mathbb{E}_{t \sim \text{Uniform}[0,T]} [\sigma(t)^2 \mathbb{E}_{p_t(\mathbf{x})} [\|\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) - s_{\theta}(\mathbf{x}, t)\|_2^2]] + \text{KL}(p_T \parallel \pi) \approx 0$$

“Likelihood weighting”



Efficient surrogate loss for maximum likelihood training

[Song et al. NeurIPS 2021 (spotlight)]

- The first term is exactly our score matching objective, but with a different weighting function.
- So this weighting function is sigma t squared, which we call **the likelihood of weighting** because **KL divergence is directly related to maximal likelihood training**.
- By minimizing the score matching loss function with this particular likelihood of weighting function, we're actually implicitly maximizing likelihoods.
- Because **this score matching loss function** is very **efficient to optimize**, this also gives a way **for efficient maximum likelihood training for score-based diffusion models**.

[Probability Evaluation]

Probability evaluation

Achieving highest probabilities on test data

Negative log-probability ↓ (bits/dim)

Method	CIFAR-10	ImageNet 32x32
PixelSNAIL [Chen et al. 2018]	2.85	3.80
Delta-VAE [Razavi et al. 2019]	2.83	3.77
Sparse Transformer [Child et al. 2019]	2.80	–
Ours	2.83	3.76



Challenges years of dominance of autoregressive models and VAEs

[Song et al. NeurIPS 2021 (spotlight)]

- With this approach, we can further improve the density values on several tested data sets. Again, we report results in negative log probability, which is lower-- which is better when lower.
- Here are some existing results. Those are state-of-the-art likelihood based generative models that achieve very good likelihood of values to image data, CIFAR-10 and ImageNet.
- And here is our result, which achieves very good likelihood 2.83 [INAUDIBLE]. This is second to the state-of-the-art. We also achieved a new state-of-the-art likelihood of ImageNet 32x32.
- This demonstrates the score-based generative models, or diffusion models, can not only challenge the dominance of GANs on image generation quality, but can also challenge the dominance of other regression models and AVEs, obtaining high likelihood values.

[Probability Evaluation]

Probability evaluation

Probability flow ODE: latent space manipulation

Interpolation



Temperature scaling

[Song et al. ICLR 2021 (Outstanding Paper Award)]

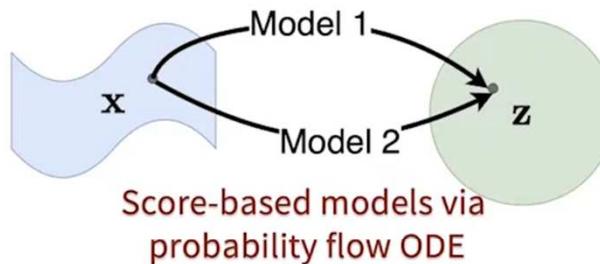
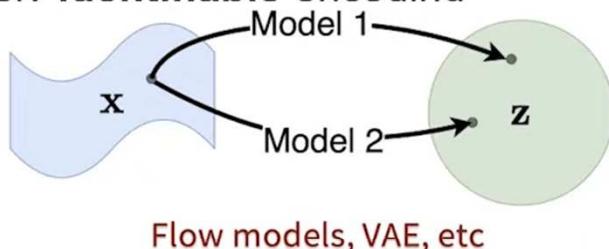
- So aside from probability evaluation, there also a few nice properties with the probability flow ODE.
- One example is, we can perform **latent space manipulation** because this ODE actually **connects score-based models to normalizing flows**, or **latent space generative models**.
- We can manipulate the latent space for applications such as **image interpolation**, which we show on the left side, or **temperature scaling**, which we show on the right side.

[Probability Evaluation]

Probability evaluation

Probability flow ODE: uniquely identifiable encoding

- Uniquely **identifiable** encoding



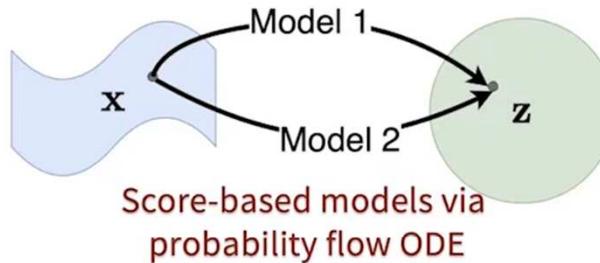
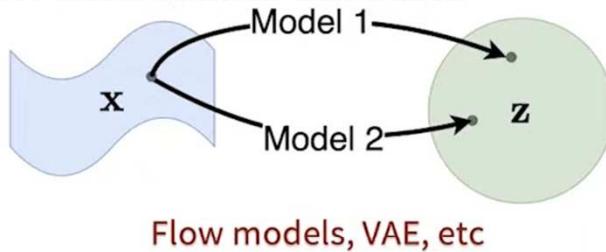
- There is **one unique property** associated with **the probability flow ODE**. That is it recovers that encode that is **uniquely identifiable**.
- So what does it mean?
- For traditional latent space generative models, such as AVEs, GANs, or normalizing flows, if you train two models with different architectures, or if you train them with different optimizers, then they will map the same image, the same datapoint x , to different latent code, z .
- But in the case of that probability flow ODE, things are a bit different. Even if you have different model architectures or different optimizers, as long as the architectures and optimizers are good enough, they will map the same data point into the same latent code, z .

[Probability Evaluation]

Probability evaluation

Probability flow ODE: uniquely identifiable encoding

- Uniquely **identifiable** encoding



- No trainable parameters in the probability flow ODE!

$$dx = -\frac{1}{2}\sigma(t)^2 \nabla_x \log p_t(x) dt$$

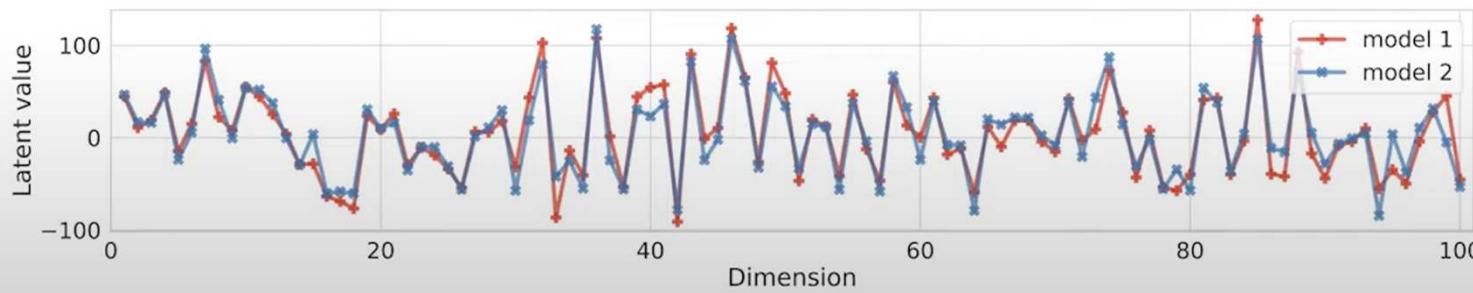
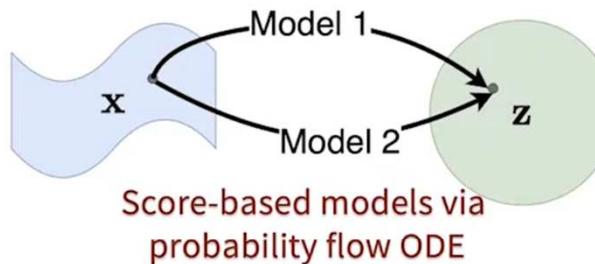
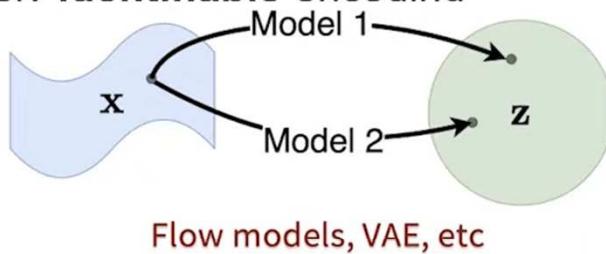
- And this is because the **probability flow ODE itself actually does not depend on a model parameter at all.**
- So once we have fixed the forward process, this probability flow ODE is also fixed. And the [INAUDIBLE] between the data point x and the latent code z is also fixed.

[Probability Evaluation]

Probability evaluation

Probability flow ODE: uniquely identifiable encoding

- Uniquely **identifiable** encoding



[Song et al. ICLR 2021 (Outstanding Paper Award)]

- So here are some experiment results.
- We trained two model architectures on the same CIFAR-10 data set.
- And we plot the first 100 dimensions of the latent code for fixed CIFAR-10 image input.
- You can see that the latent code is almost the same, even though we were using two different model architectures trained separately.

[Probability Evaluation]

Score-based generative modeling: summary

Flexible models

- Bypass the normalizing constant
- Principled statistical methods

[Song et al. UAI 2019 oral]

Improved generation

- Higher sample quality than GANs
- Controllable generation

[Song & Ermon. NeurIPS 2019 oral]
[Song & Ermon. NeurIPS 2020]
[Song et al. ICLR 2021 oral]
(Outstanding Paper Award)
[Song et al. ICLR 2022]

Probability evaluation

- Accurate probability evaluation
- Better estimation of data probabilities

[Song et al. ICLR 2021 oral]
[Song et al. NeurIPS 2021 spotlight]

- So as a summary, we have talked about score-based generative models. It has multiple desirable properties.
- The first, it allows very **flexible neural network models**. Because the score functions can bypass the normalizing constant, and we can train those flexible models from data with principle statistical methods.
- And second, we can **generate samples with a very high quality** that can even surpass GANs in many challenging image interaction tasks. And moreover, we can control the selection process for important applications in conditional image generation, and also inverse problem solving.
- And finally, we can **compute the probability values accurately**, even though we **only have models of the score function**. And empirically, we can even obtain **better density estimation performance** than existing likelihood-based generative models.