Diffusion and Score-Based Generative Models

CVPR2022 Tutorial

Yang Song

Artificial Intelligence

Creating the Future

Dong-A University

Division of Computer Engineering & Artificial Intelligence

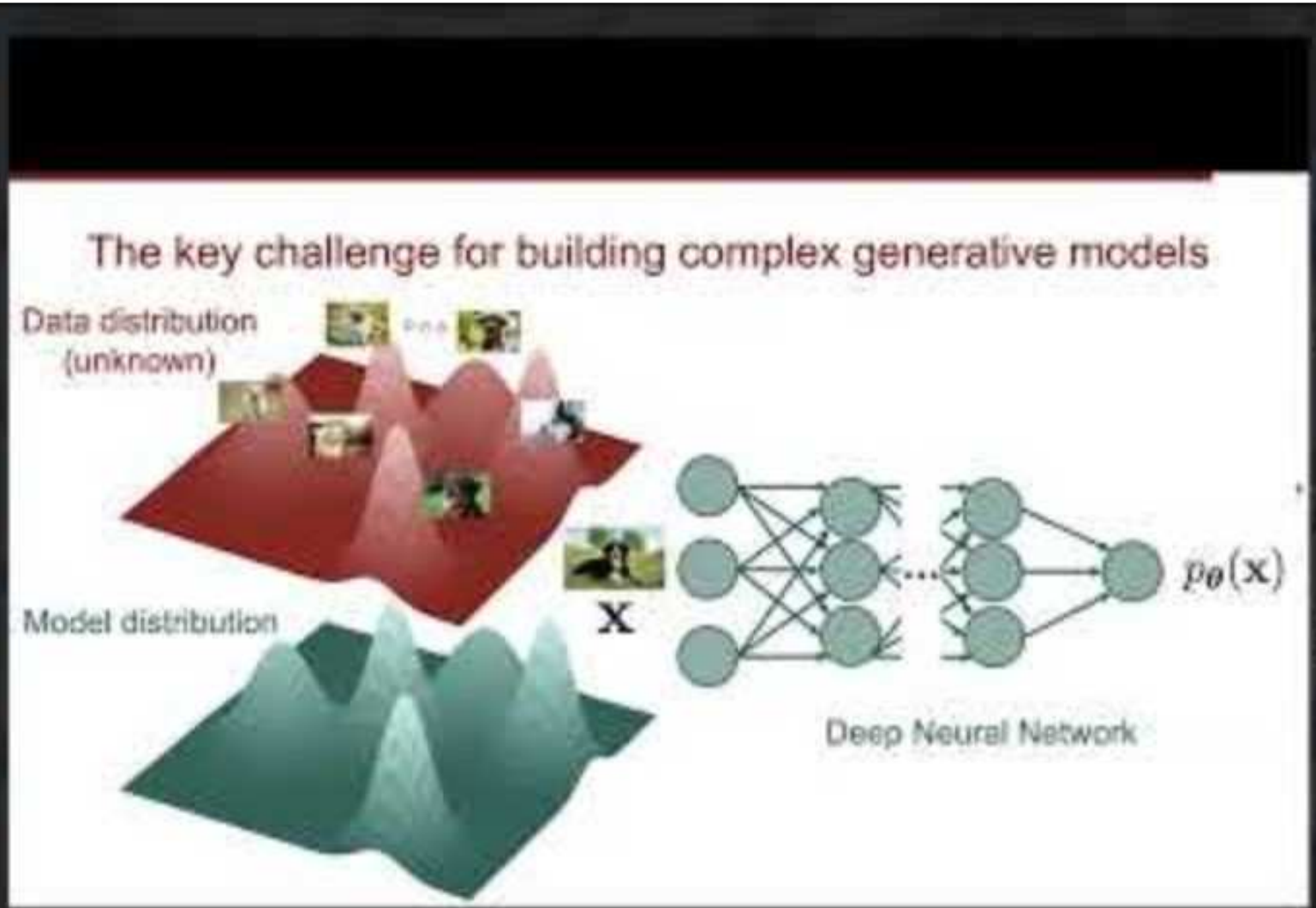
References

Main

- CVPR2022 Tutorial – Yang Song
- https://www.youtube.com/watch?v=wMmqCMwuM2Q

Blog

- 고급생성모델입문서, 조의현, 위키독스
- https://wikidocs.net/book/14332



December 12, 2022

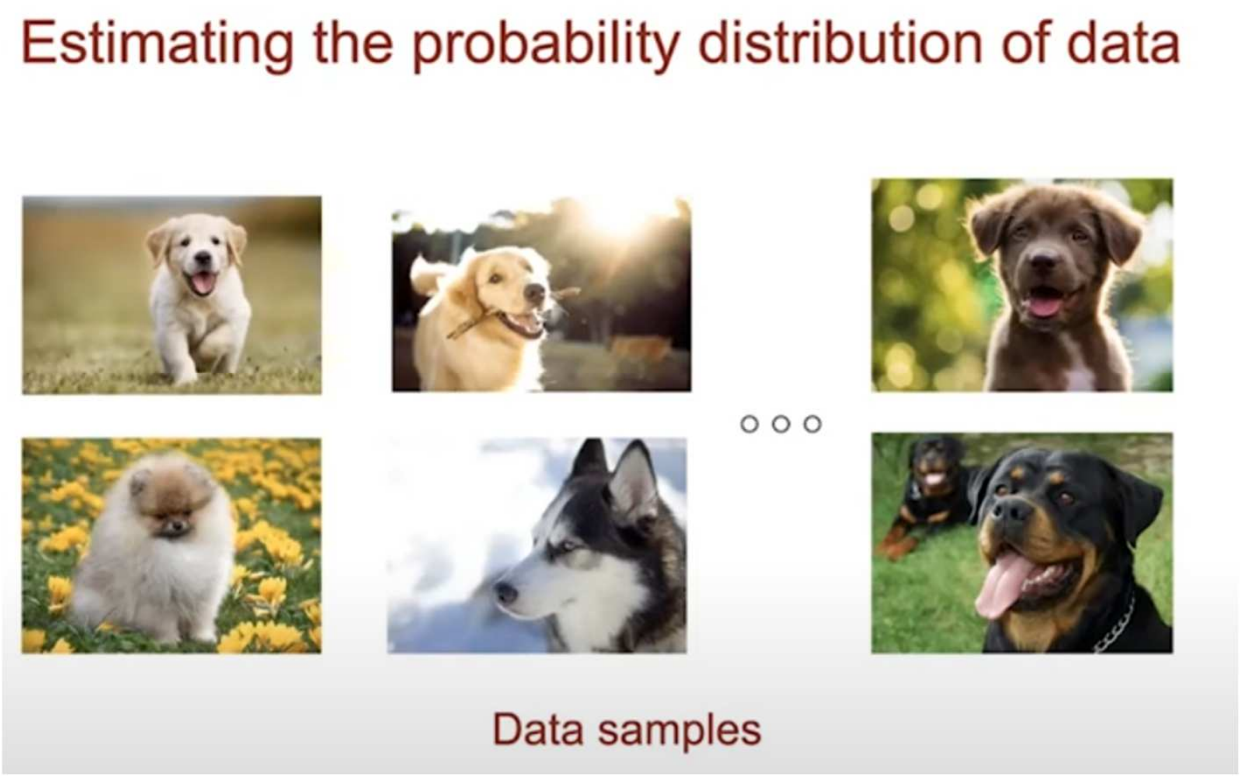
Diffusion and Score-Based Generative Models

Yang Song

Stanford University

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

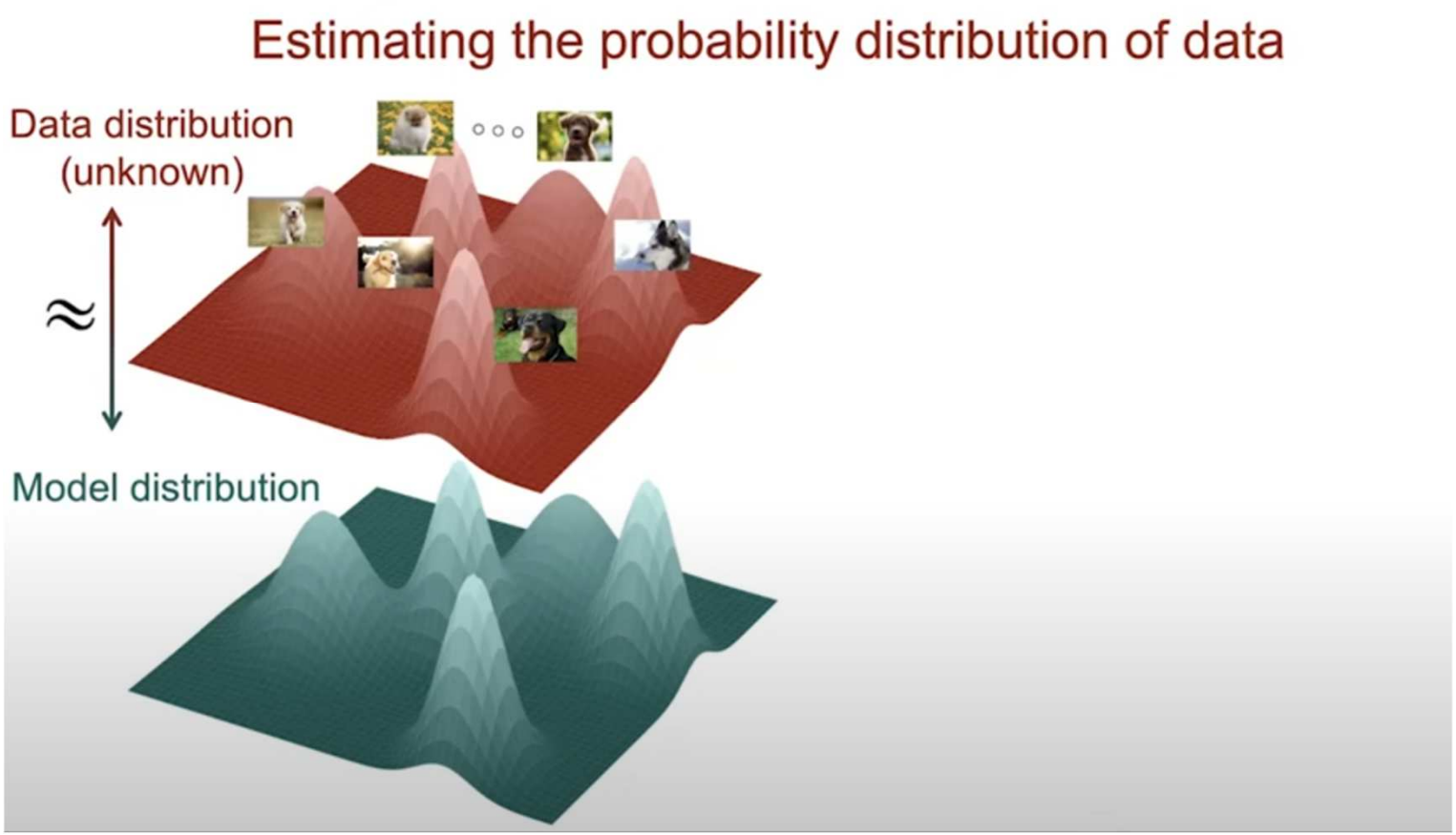
[Model Distribution]



- 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

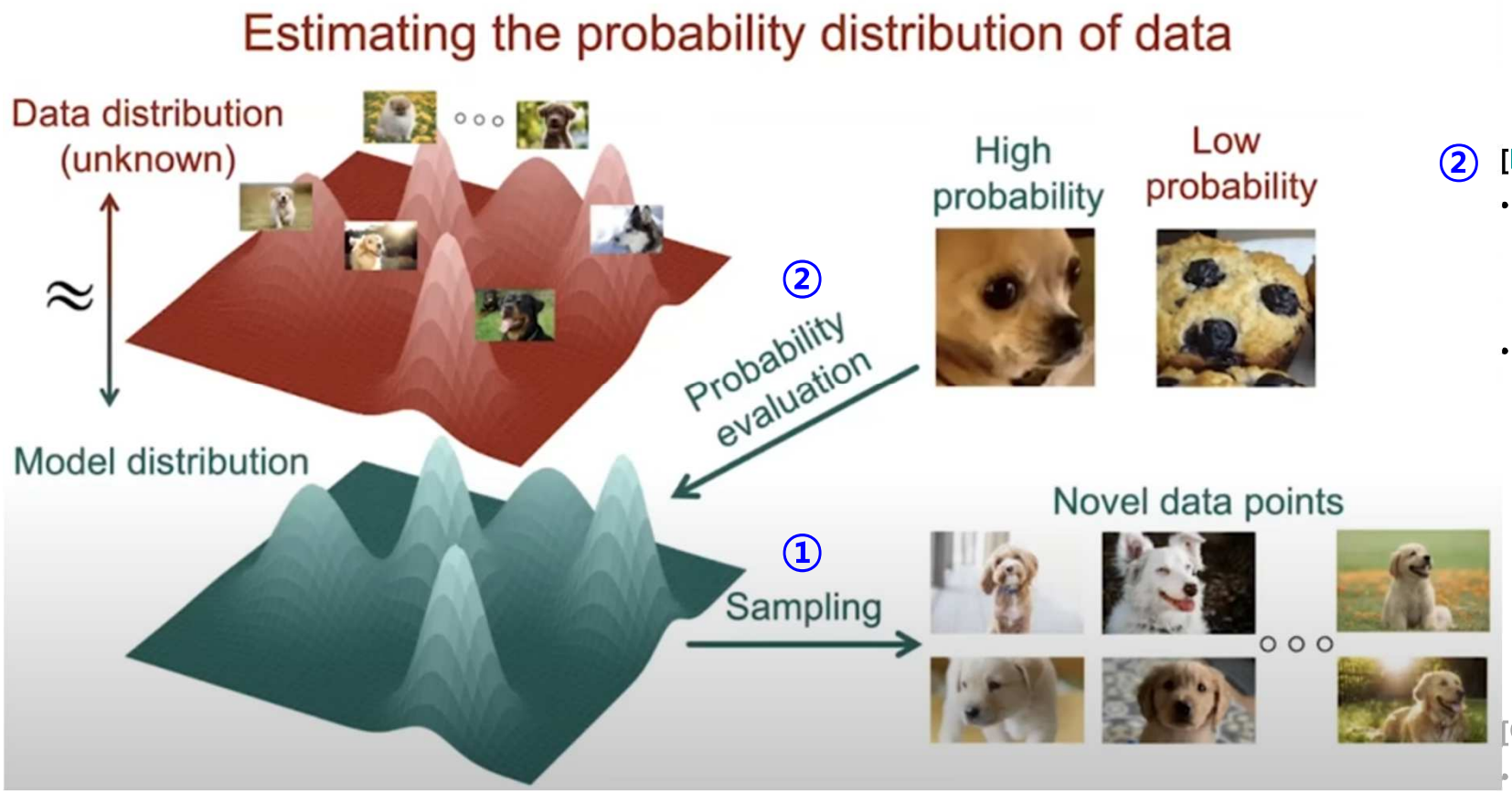
[Model Distribution]



- 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 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Model Distribution]



① **[Sampling]**

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

② **[Probability Evaluation]**

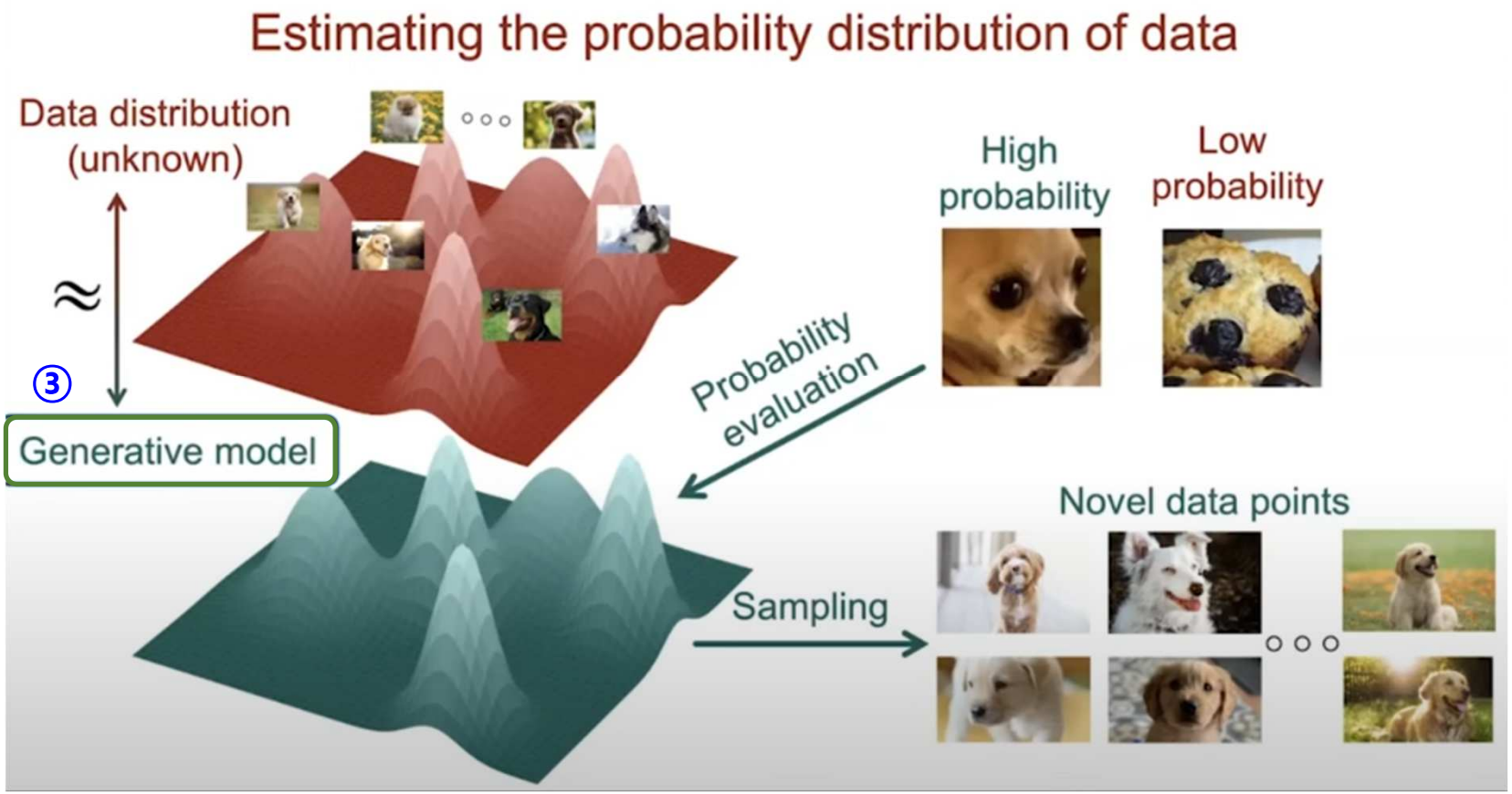
- Another application is we can use this model distribution to **compute the probability value for any potential data point**.
- So as an example for a data point, like a picture of a chihuahua, because it is a picture of a dog, it is actually within our data distribution. And therefore, this model distribution 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 l**ower 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Model Distribution]



[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.

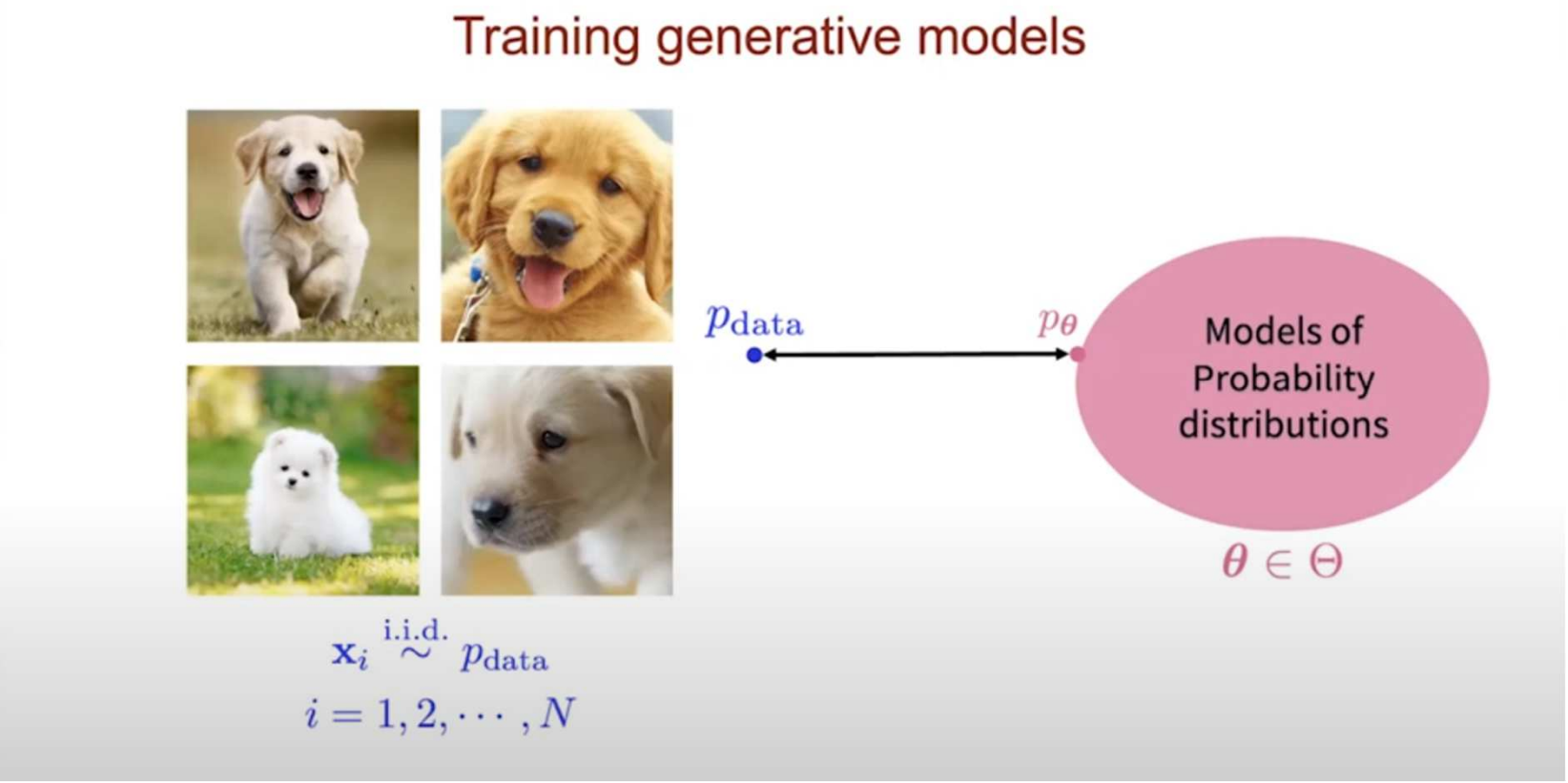
③

[Generative Model]

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

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Model Distribution]

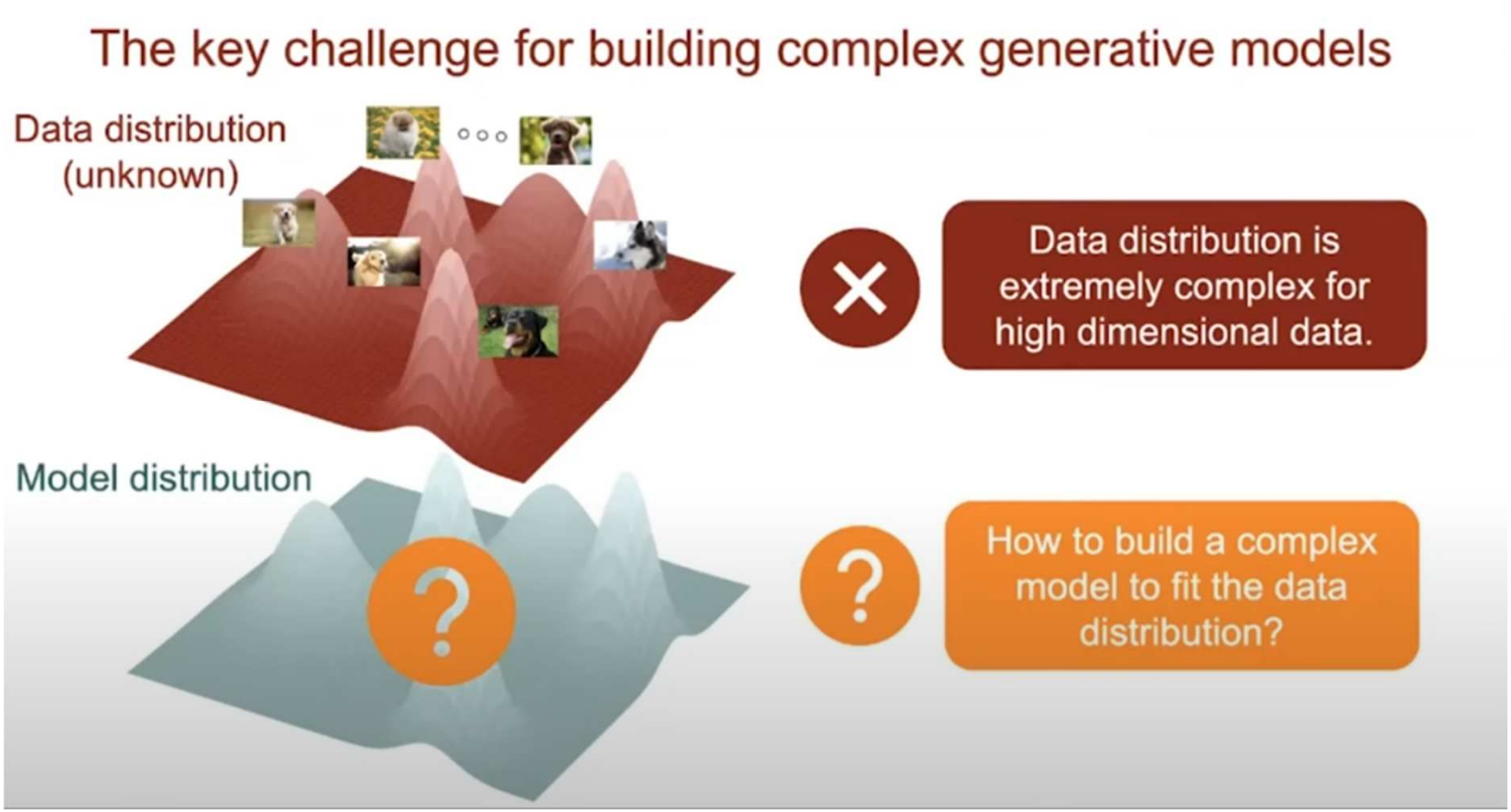


- **How can we train those generative models?**

- 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_theta to P_data**.
- And afterwards, we can just **generate samples from P_theta**.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Data Distribution]

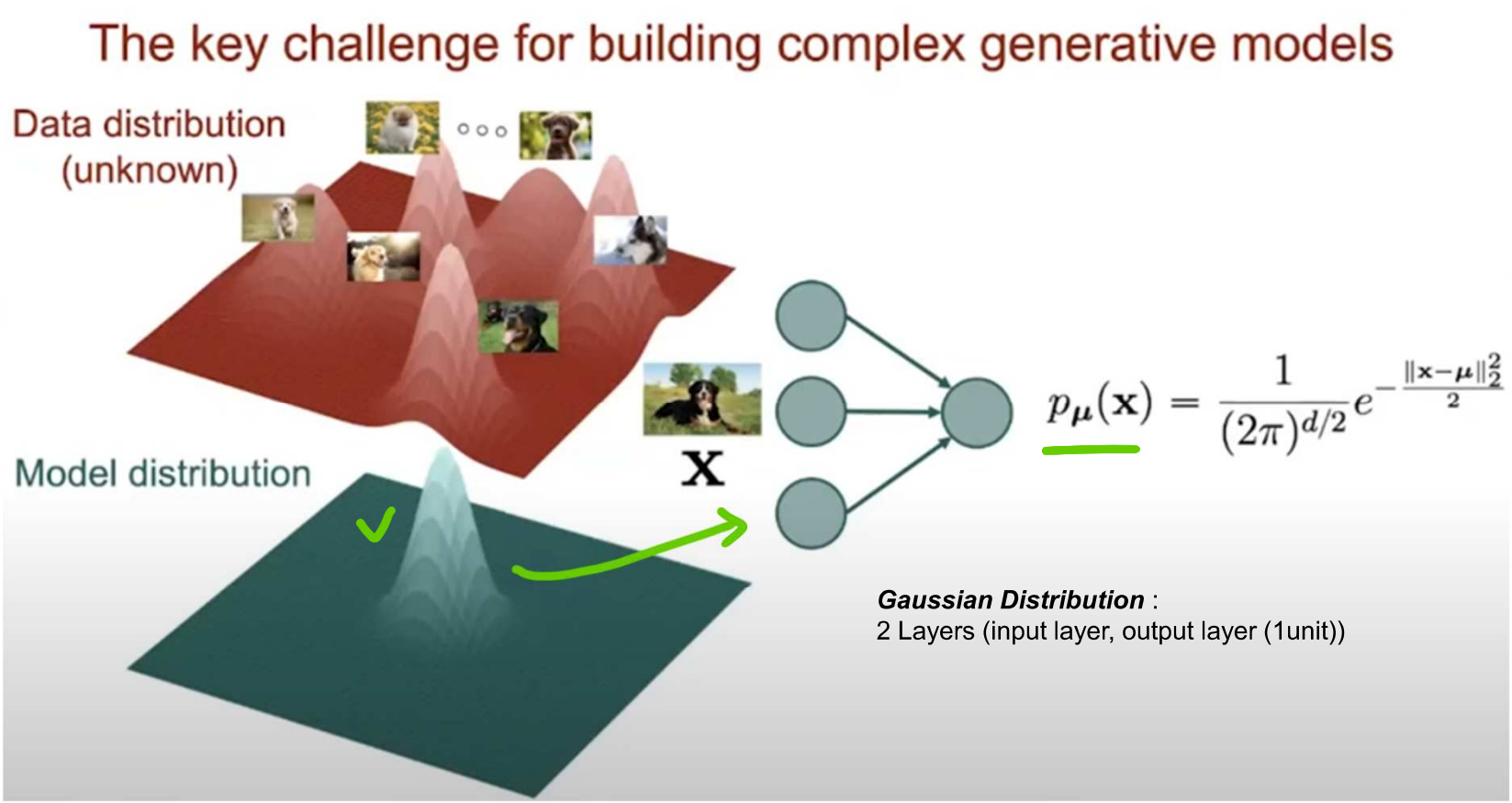


- However, there is one key challenge associated with this framework.
- That is, **our data distribution** can be **extremely complicated**, especially for data is **high dimensions**.
- So consider how complicated it might be for distributions of images, video, audio. It might have millions of dimensions.
- 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?**

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Data Distribution]



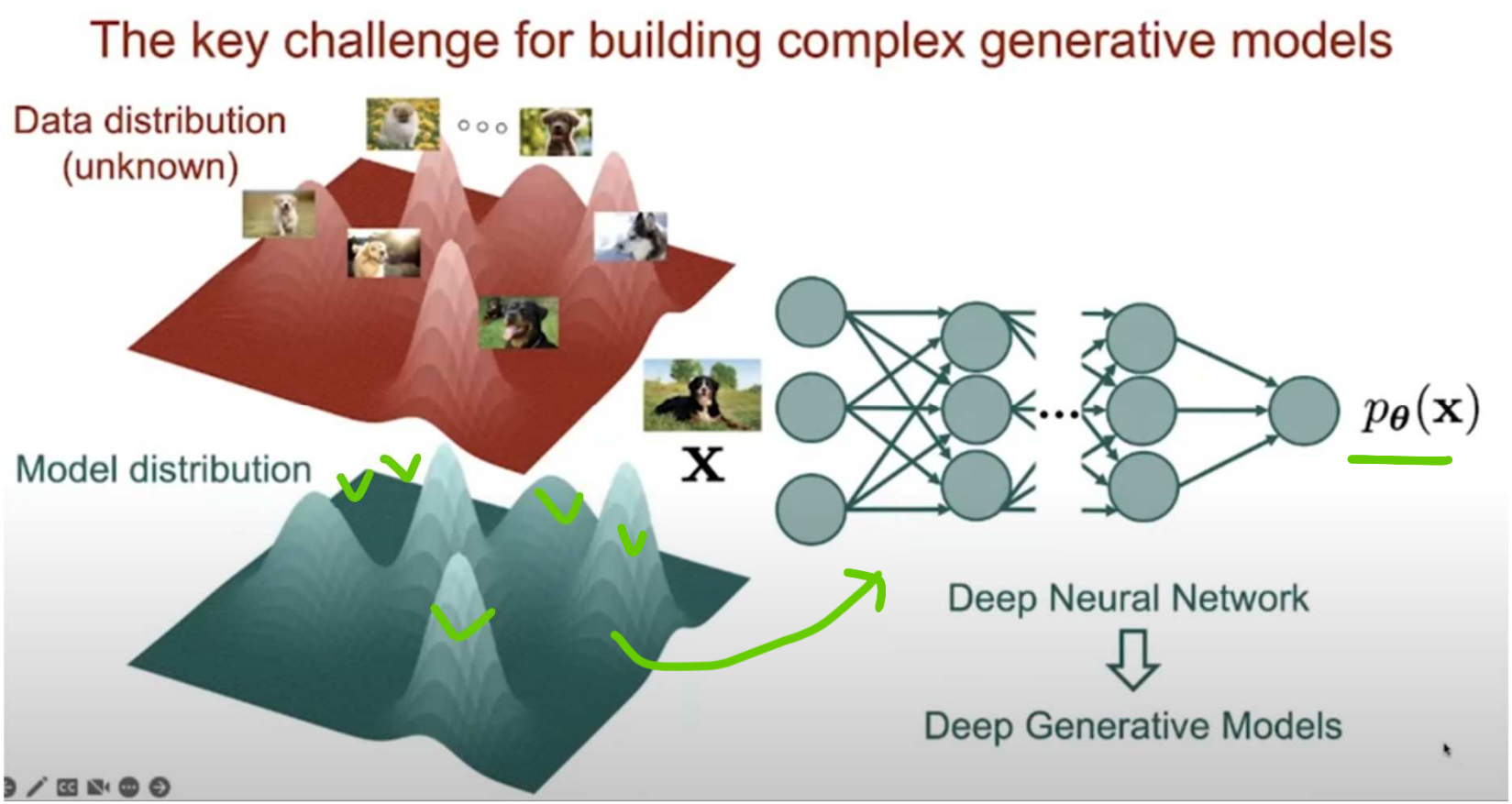
Gaussian Distribution :
2 Layers (input layer, output layer (1unit))

- 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.

- 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**.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Data Distribution]



- 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**.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

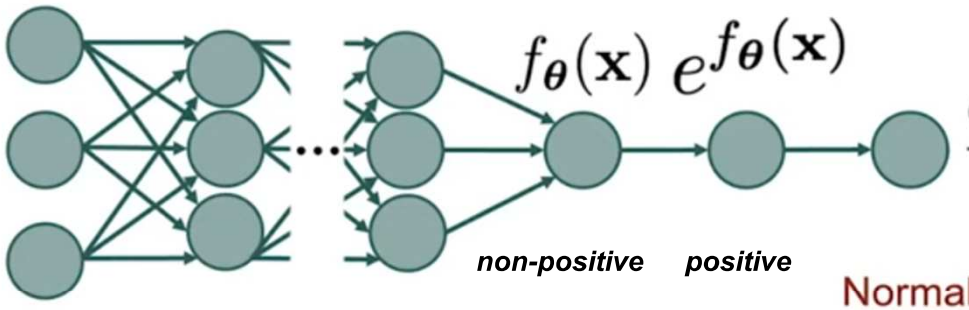
[Data Distribution]

The key challenge for building complex generative models

Build to represent the Probability Distribution

$\mathbf{x}$

High dimension



1 dimension

Probability Distribution

$$\frac{e^{f_\theta(\mathbf{x})}}{Z_\theta} = p_\theta(\mathbf{x})$$

Normalizing constant

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

☞ ***Energy-based model (EBM)***

- 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 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**.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Data Distribution]

The key challenge for building complex generative models

$\mathbf{x}$

$f_{\boldsymbol{\theta}}(\mathbf{x})$ $e^{f_{\boldsymbol{\theta}}(\mathbf{x})}$

$$\frac{e^{f_{\boldsymbol{\theta}}(\mathbf{x})}}{Z_{\boldsymbol{\theta}}} = p_{\boldsymbol{\theta}}(\mathbf{x})$$

Normalizing constant

Normalizing constant for Gaussians

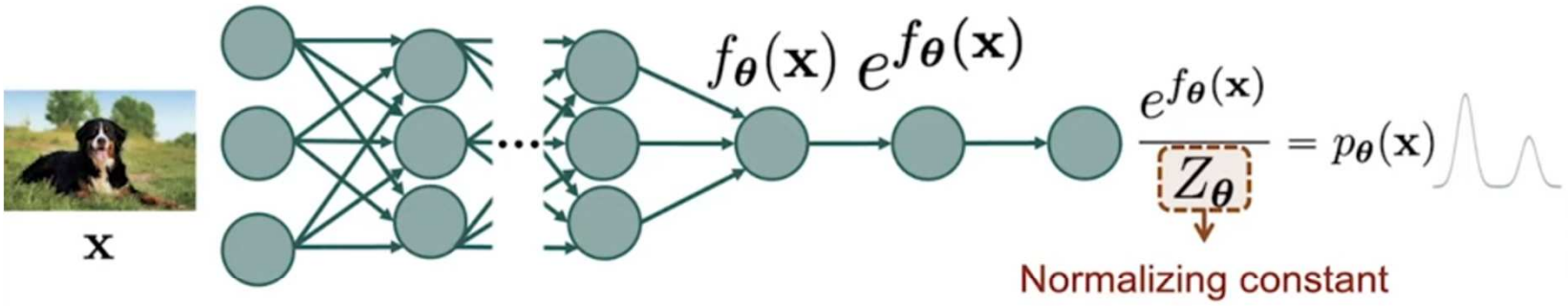
$$Z_{\mu} = \frac{1}{(2\pi)^{d/2}}$$

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

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Data Distribution]

The key challenge for building complex generative models



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

Intractable

- 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**.

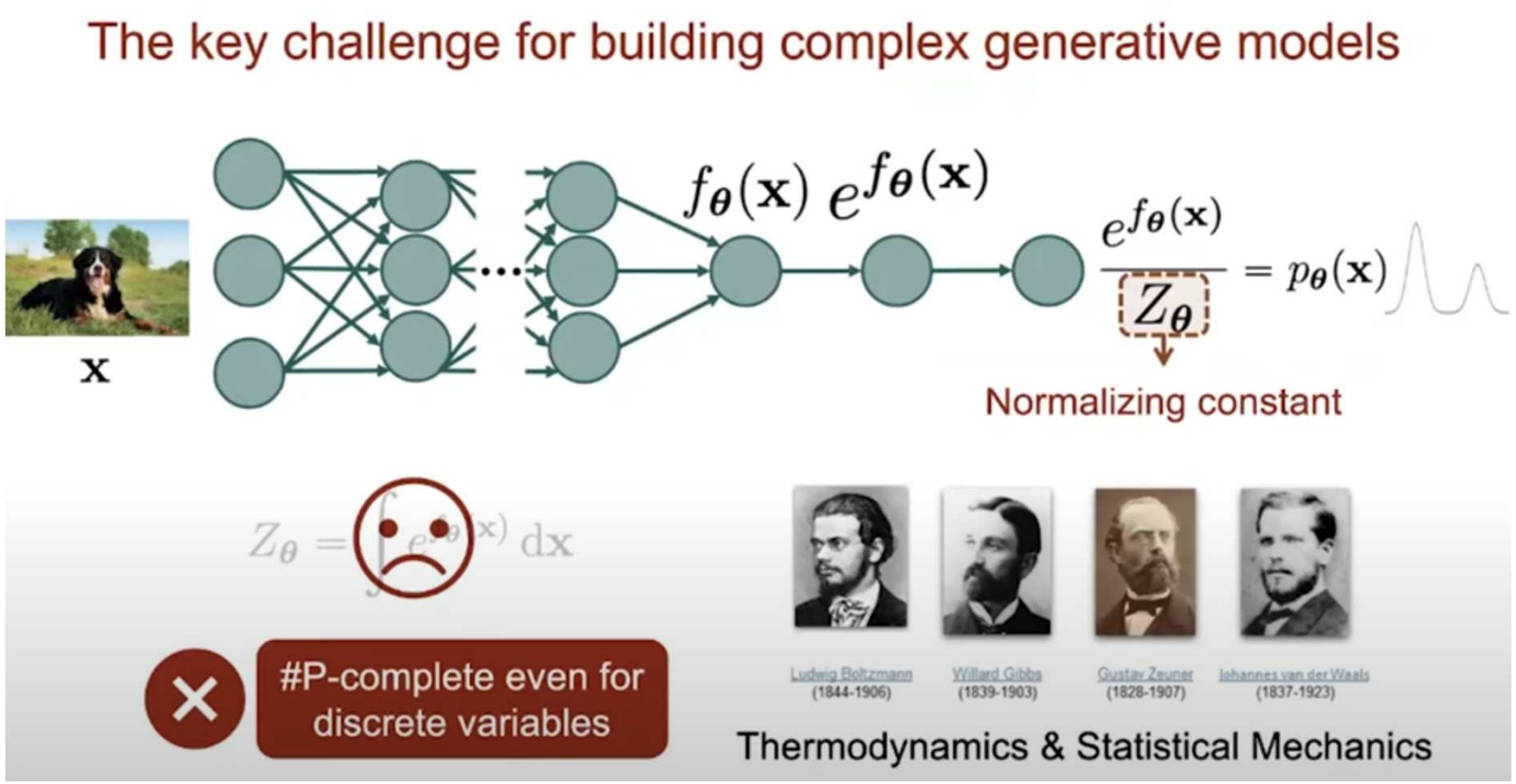
Normalizing Constant $Z_\theta = \int \exp(f_\theta(x))dx$이 계산이 어려운 intractable한 분포

➢ 생성 모델을 학습 데이터에 대해 학습시키기 위해서는 log likelihood $\log p_\theta(x)$의 gradient ∇_θ를 파라미터 θ에 더해주는 gradient optimization 단계를 거쳐야 하므로 계산이 불가능한 분포 $Z_\theta = \int \exp(f_\theta(x))dx$를 미분하는 방법을 찾아야 함.

☞ [출처] 고급생성모델입문서, https://wikidocs.net/230104

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

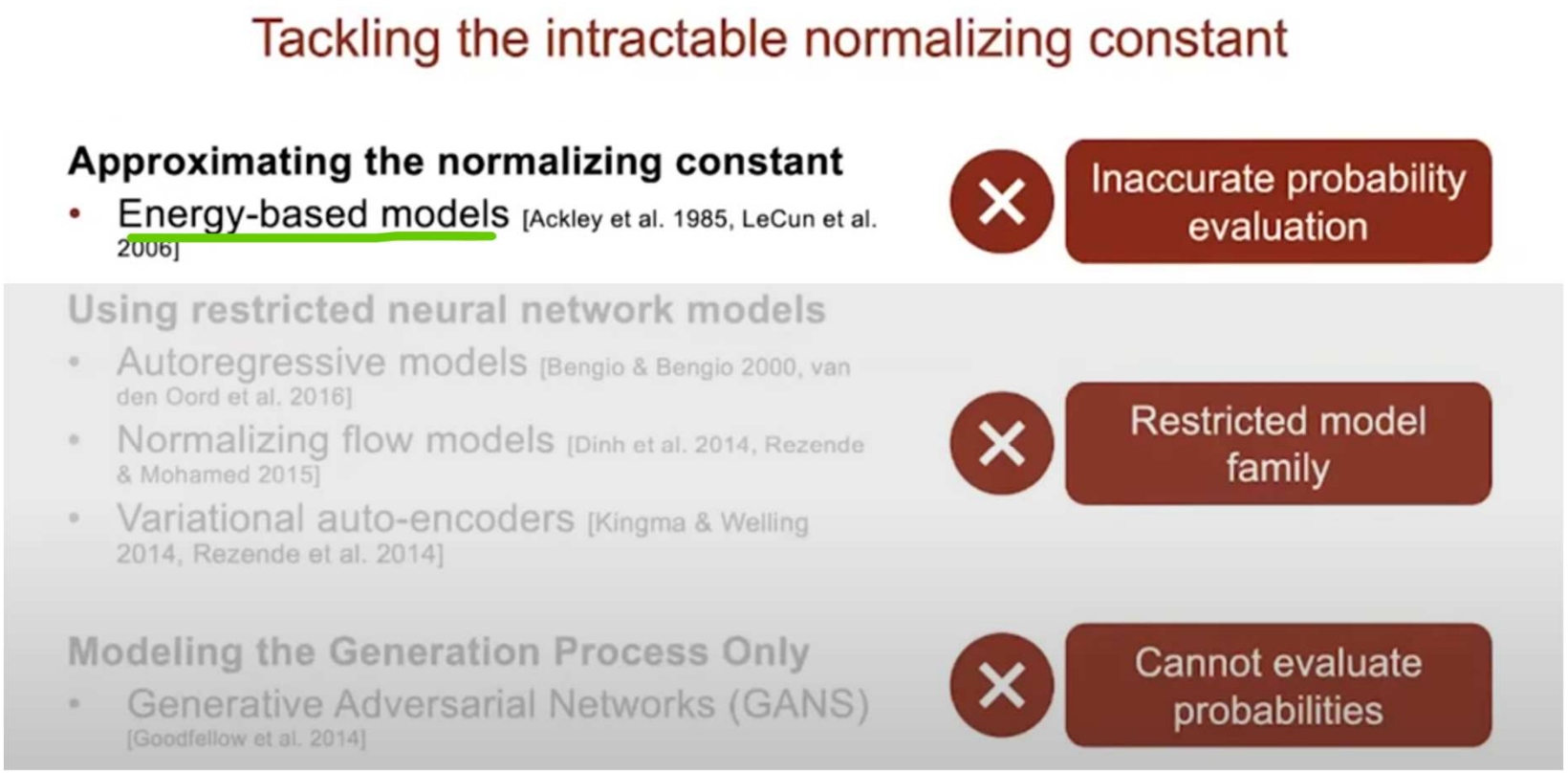
[Data Distribution]



- And as a quick example, even if we consider **a simplified case where 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Deep Genetic Models]



☞ Energy-based models (EBM) - 고급생성모델입문서, https://wikidocs.net/230097

- 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Deep Genetic Models]

Tackling the intractable normalizing constant

Approximating the normalizing constant

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

✕ Inaccurate probability evaluation

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]

✕ Restricted model family

Modeling the Generation Process Only

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

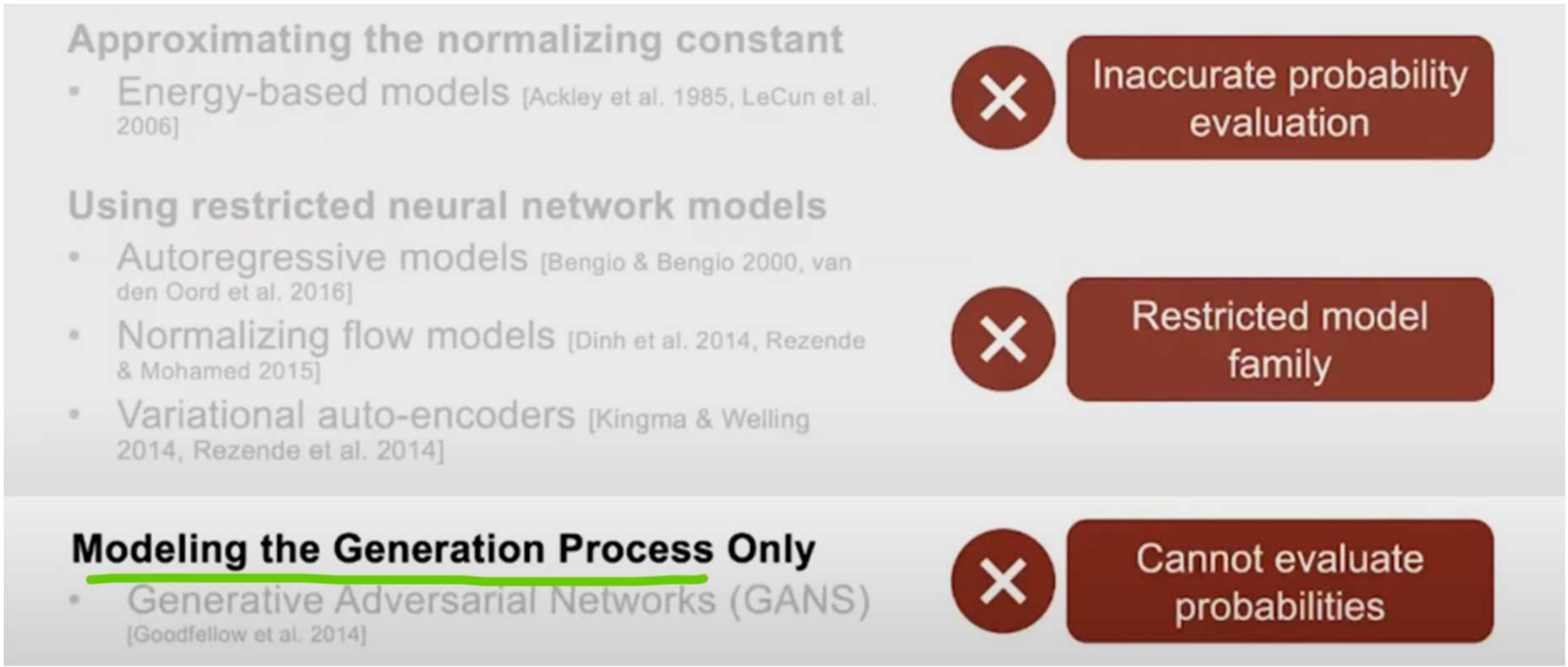
✕ Cannot evaluate probabilities

- 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Deep Genetic Models]

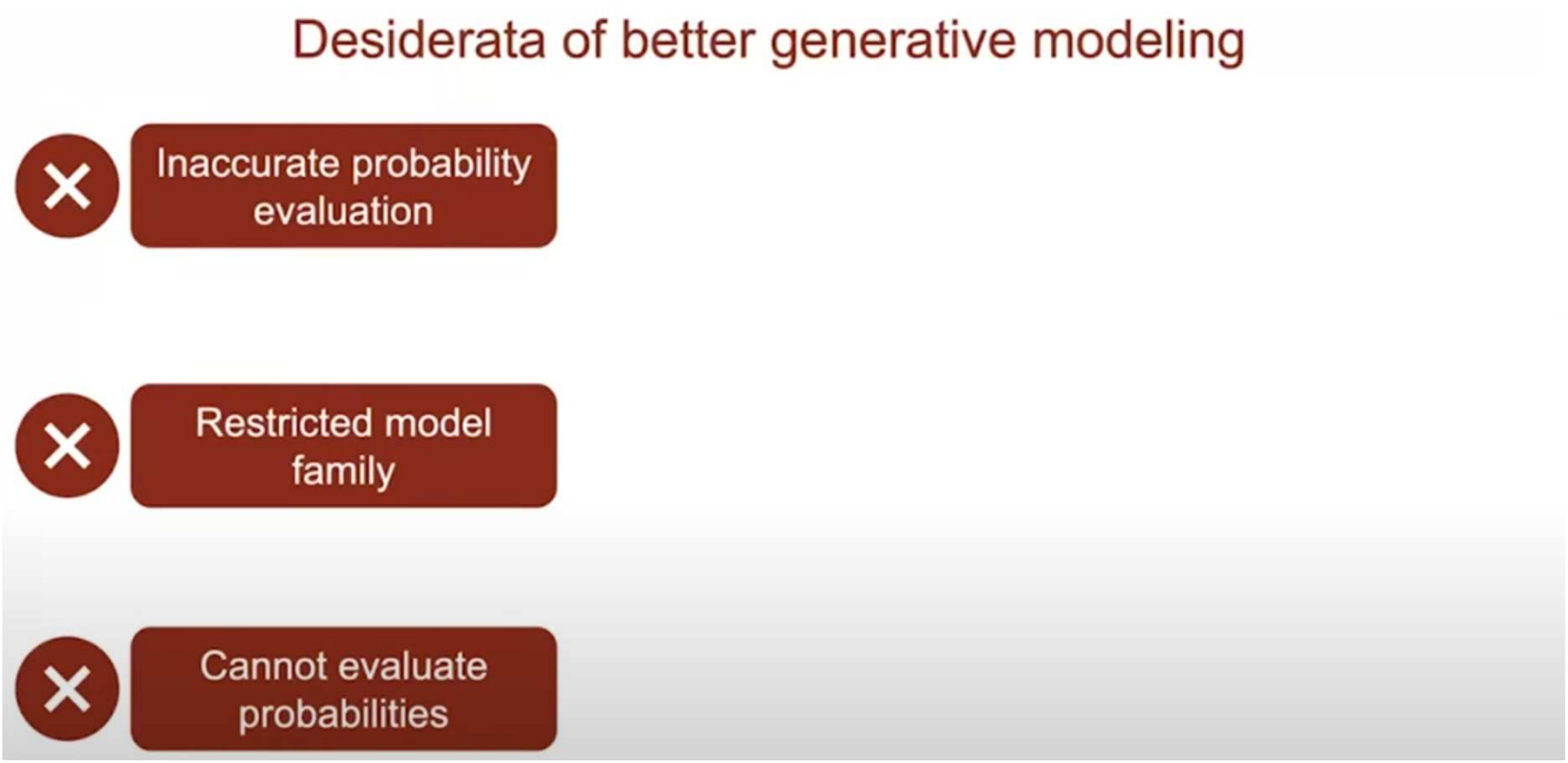
Tackling the intractable normalizing constant



- 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**.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

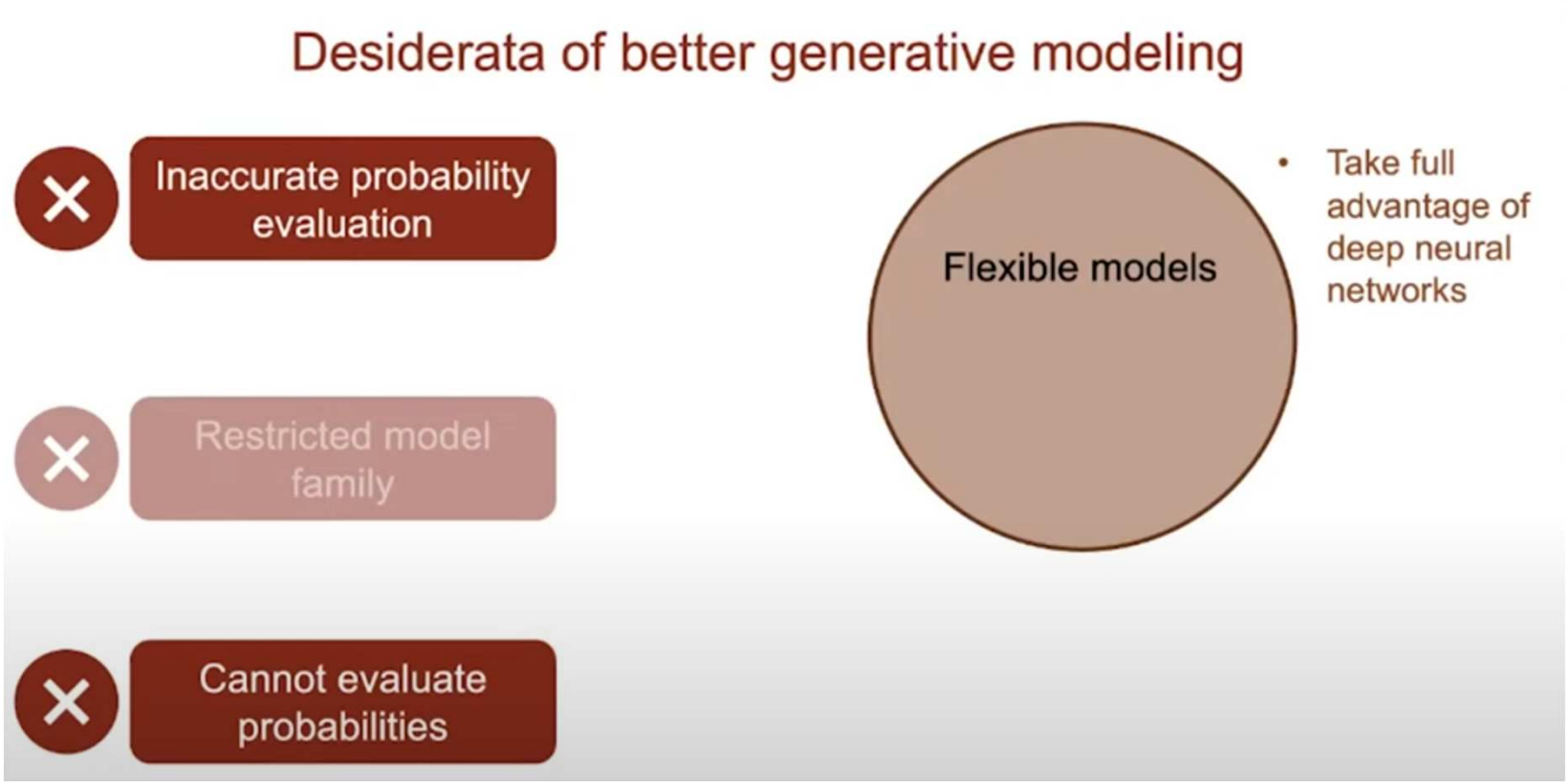
[Deep Genetic Models]



- 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

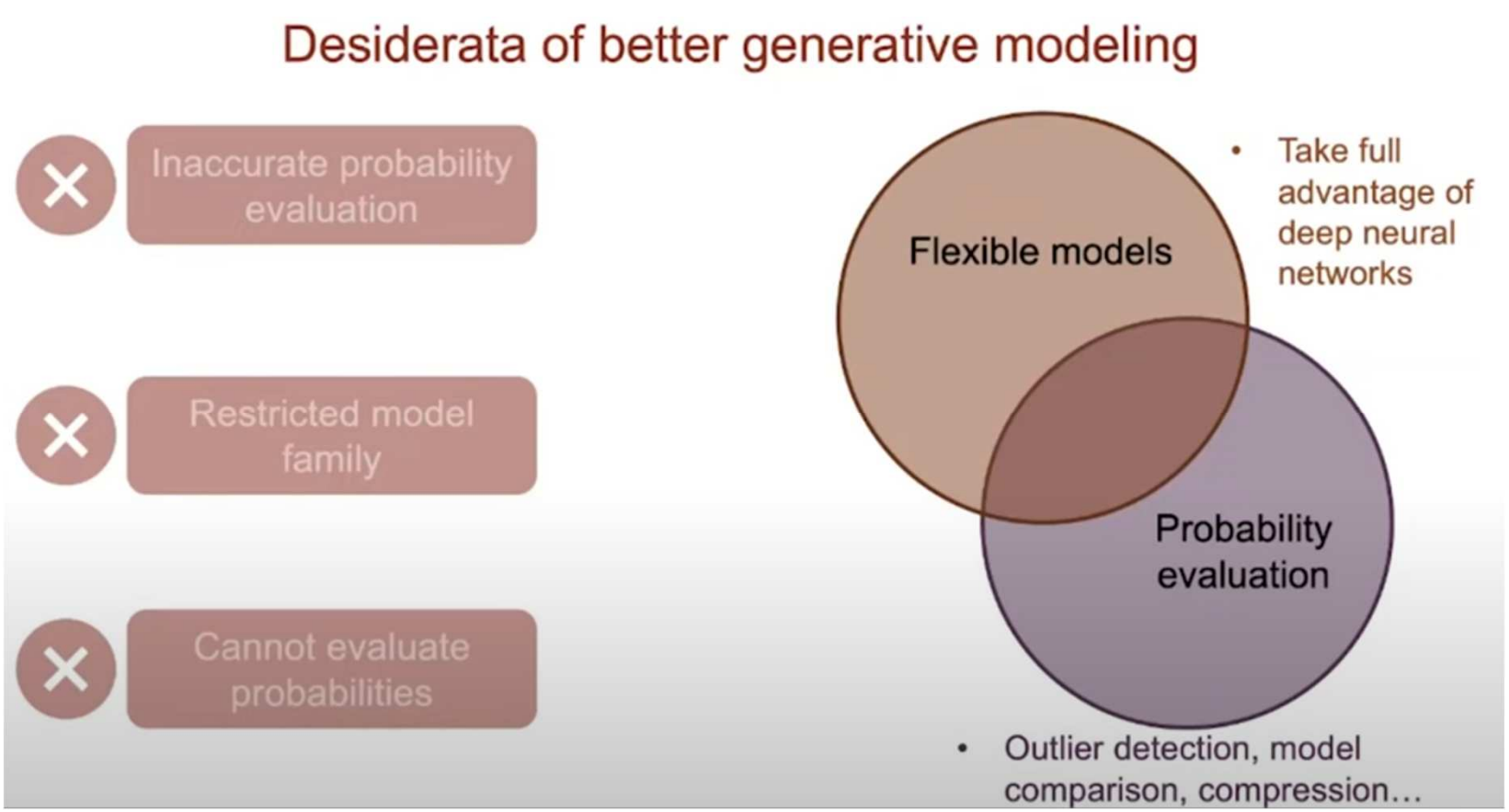
[Deep Genetic Models]



- And one thing is we hope that this better framework can allow us to use a very **flexible neural network models to parameterize this distribution**.
- So this not only addresses the second challenge on our side, but also allows us to take full advantage of the deep learning revolution to leverage very powerful deep neural networks to build our deep generative models.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

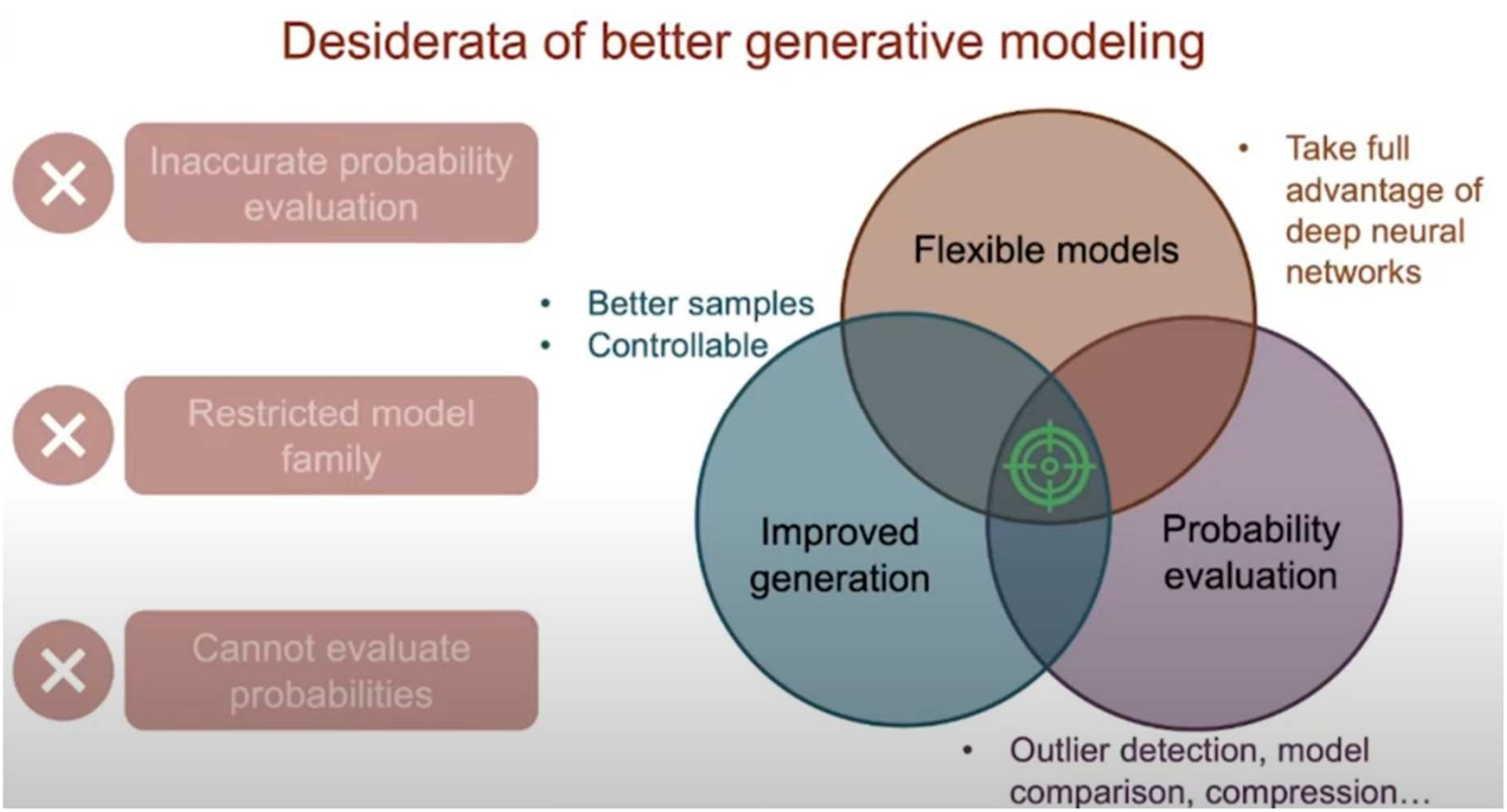
[Deep Genetic Models]



- 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Deep Genetic Models]



- And finally, because we are aiming to build **a more powerful framework of general models.**
- 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**.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Functions]

Our proposal: working with score functions

$p(\mathbf{x})$

Probability density function

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

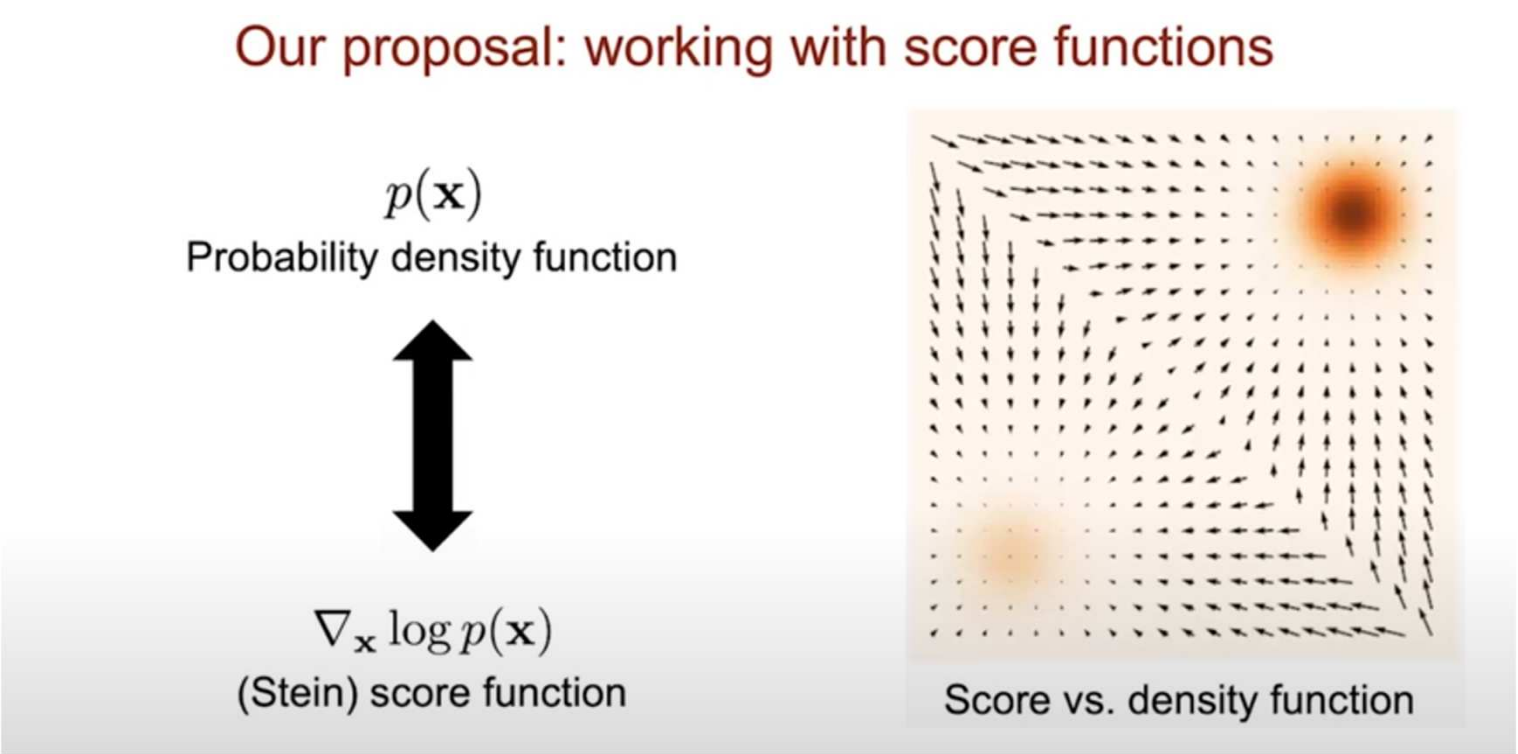
(Stein) score function

- What is the score function?
- Well, suppose we have a continuous probability distribution where we use **p_x**
- to represent the probability density function.
- We define the **score function** as the **gradient of log p_x**.
- 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 x**, it is **not taken with respect to any model parameter like theta**.

- What does our score function look like?

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

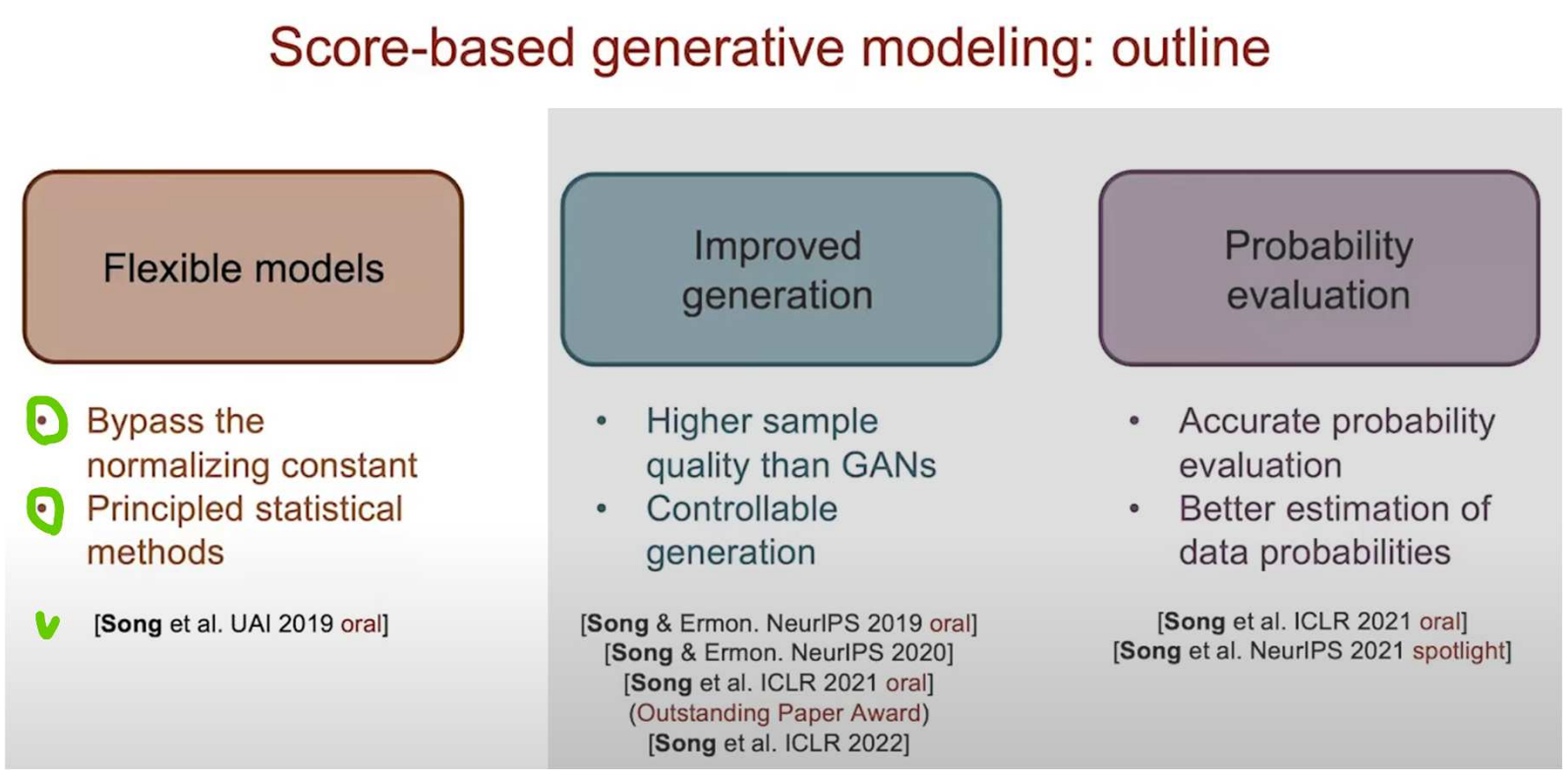
[Score Functions]



- 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 text 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. But computationally, this score function is much easier to work with compared to the density function.**

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Outline]

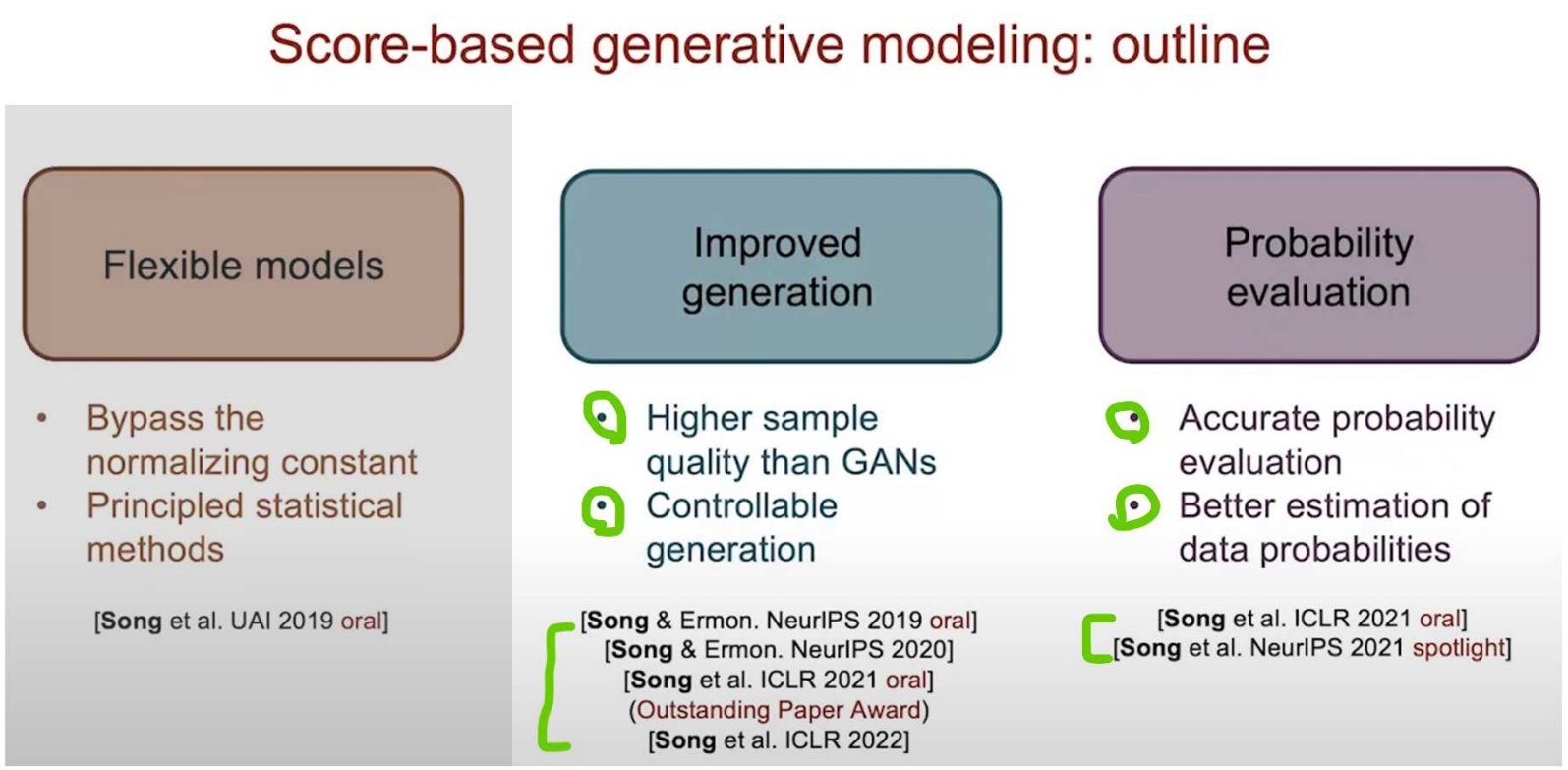


- When we work with the score function for representing probability distributions, we get our **score-based generative models**. And I will show you that this score-based generative model has multiple advantages.

- **[First]** it allows very **flexible models** because the score functions actually **do not need to be normalized at all**.
- Which means ① we can **use a very flexible neural net models to represent this score function**, and ② we can **learn such models or score functions from data using principled statistical approaches**.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Outline]

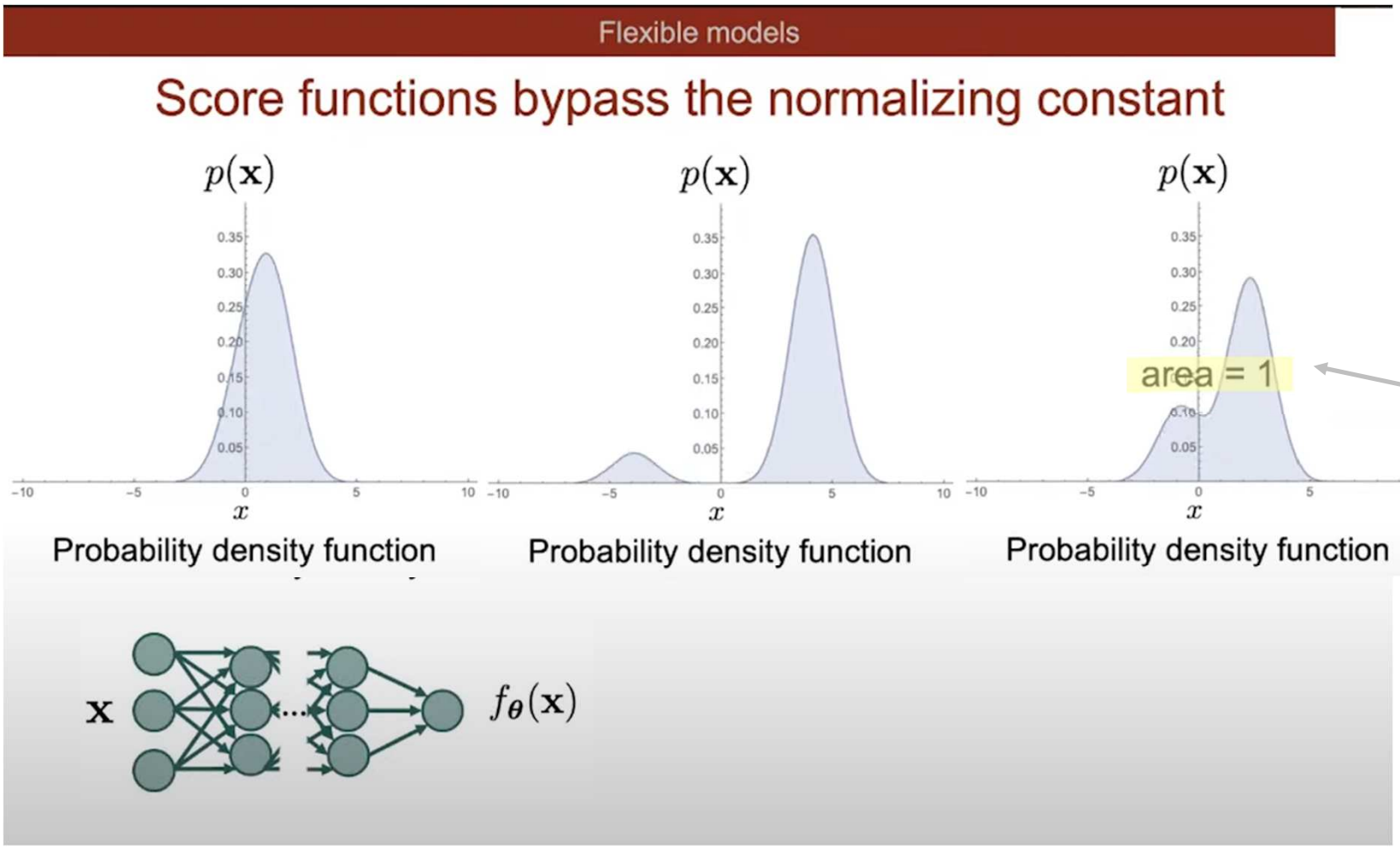


- **[Second]** ① We can **directly generate samples from those models of score functions**, and those samples could have surprisingly **good quality** and can be even better than [INAUDIBLE] in many situations. And moreover, ② we can **control the sample generation process** in a principal way for many important applications.

- **[Third]**, even if we only have the model of the score function, ① we can still **compute the probability values accurately**. And empirically, ② 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Flexible models]



- 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**.

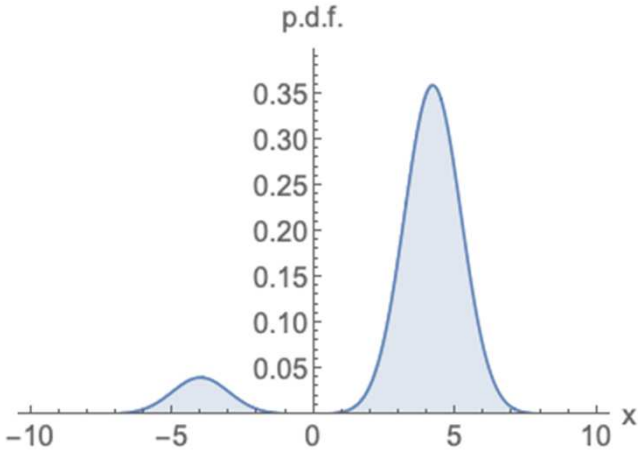
  And due to this constraint, when we use the deep neural network model to those density functions, we **always have to deal with this intractable normalizing constant difficulty**.

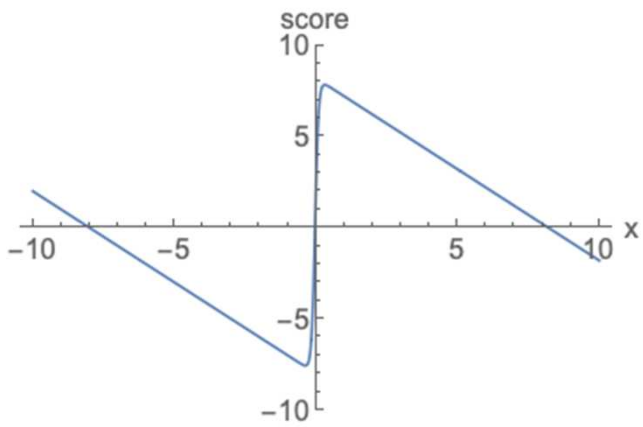
CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Flexible models] https://yang-song.net/blog/2021/score/

Flexible models

Score functions bypass the normalizing constant



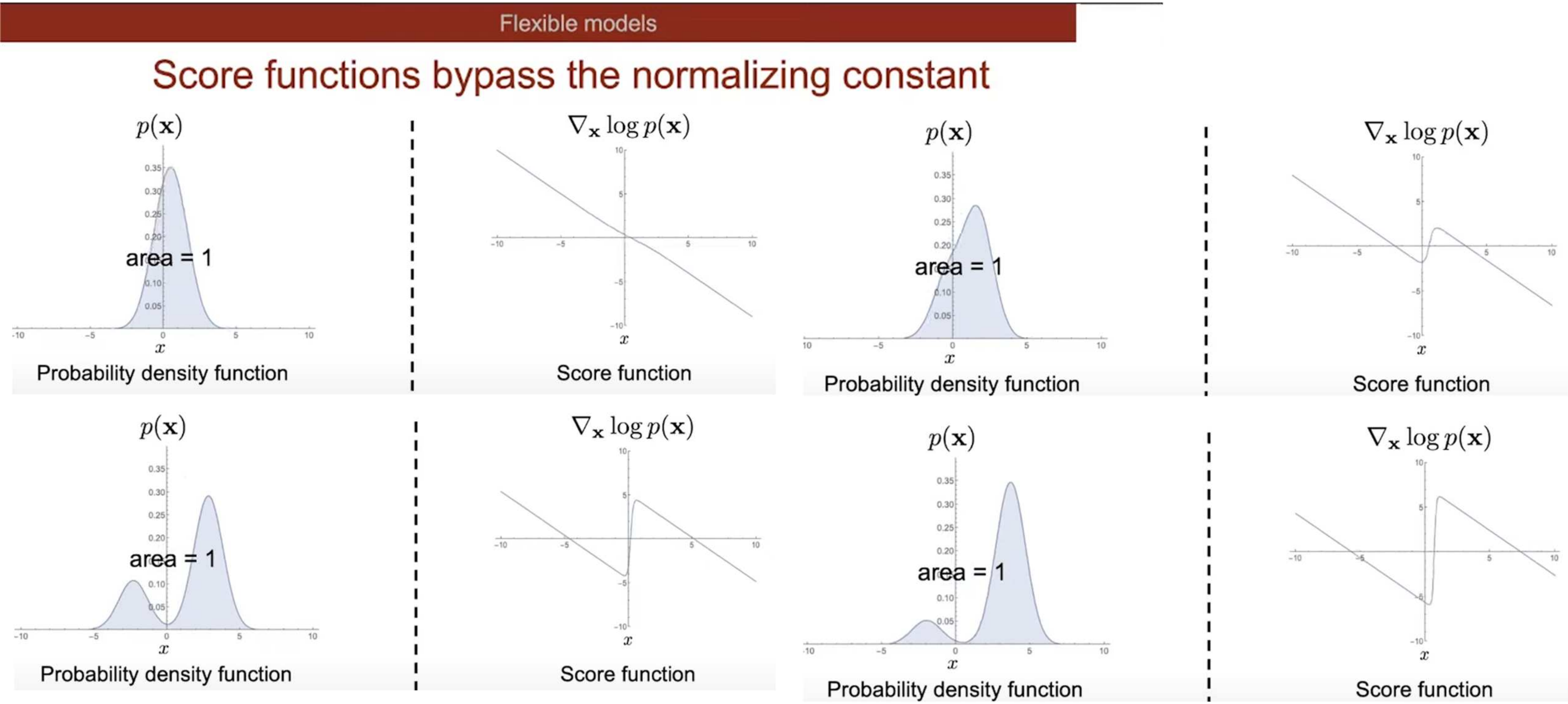


Parameterizing probability density functions.
No matter how you change the model family and parameters, it has to be normalized (area under the curve must integrate to one).

Parameterizing score functions.
No need to worry about normalization.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Flexible models]



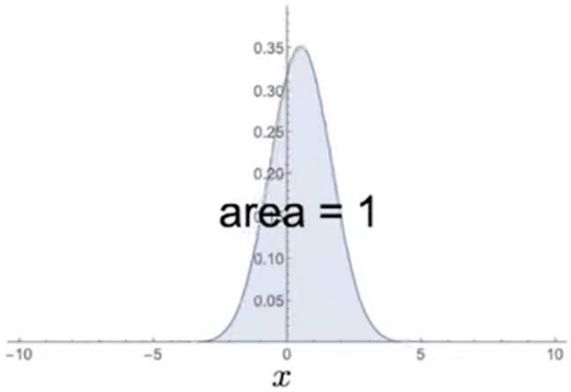
CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Flexible models]

Flexible models

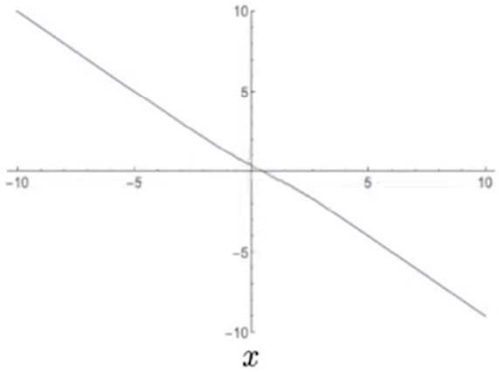
Score functions bypass the normalizing constant

$p(\mathbf{x})$

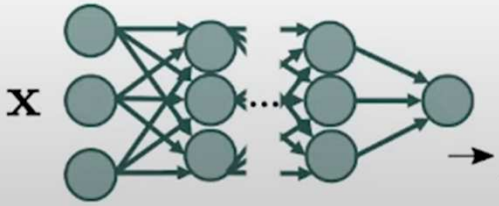


Probability density function

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



Score function



$$f_\theta(\mathbf{x}) \rightarrow p_\theta(\mathbf{x}) = \frac{e^{f_\theta(\mathbf{x})}}{\cancel{Z_\theta}}$$

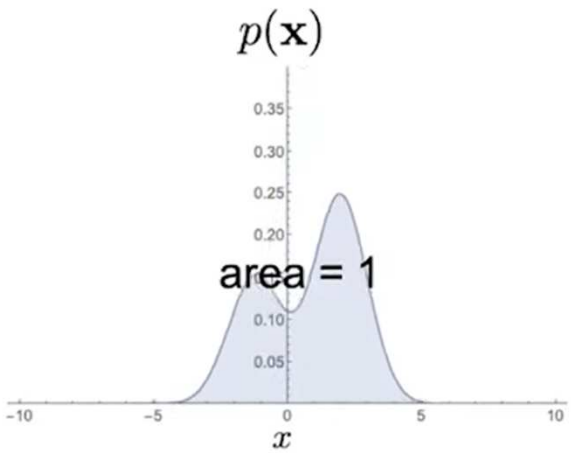
- But in contrast, if we **model the same distribution through the score functions**, then, as the animation shows, **there is no such normalization restriction**.
- In fact, **if we compute a score function, there is no such normalization restriction**.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

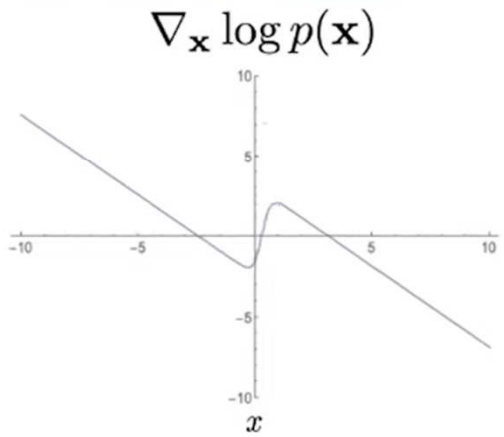
[Flexible models]

Flexible models

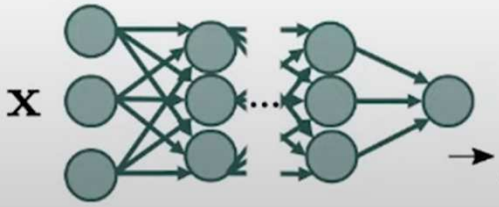
Score functions bypass the normalizing constant



Probability density function



Score function



$$f_{\boldsymbol{\theta}}(\mathbf{x}) \rightarrow p_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{e^{f_{\boldsymbol{\theta}}(\mathbf{x})}}{Z_{\boldsymbol{\theta}}}$$

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

$\nabla_{\mathbf{x}} \log Z_{\boldsymbol{\theta}} = 0$

- 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.

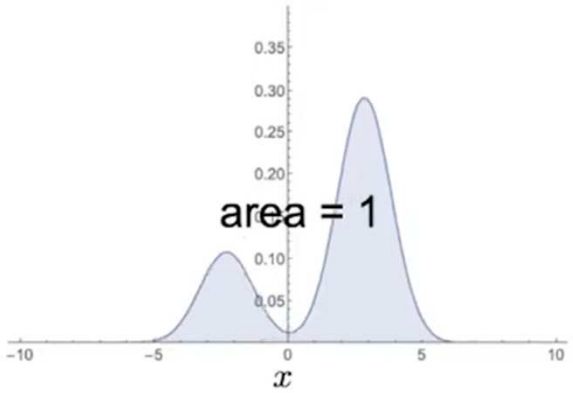
CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Flexible models]

Flexible models

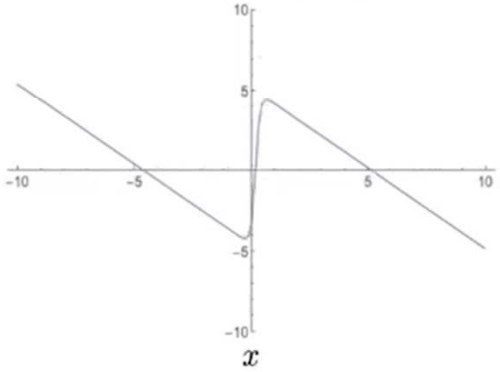
Score functions bypass the normalizing constant

$p(\mathbf{x})$

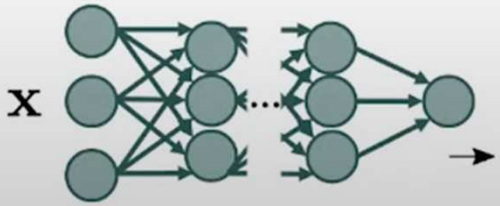


Probability density function

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



Score function



$$f_{\theta}(\mathbf{x}) \rightarrow p_{\theta}(\mathbf{x}) = \frac{e^{f_{\theta}(\mathbf{x})}}{\cancel{Z_{\theta}}}$$

$$\begin{aligned} \nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}) &= \nabla_{\mathbf{x}} f_{\theta}(\mathbf{x}) - \boxed{\nabla_{\mathbf{x}} \log Z_{\theta}} \\ &= \nabla_{\mathbf{x}} f_{\theta}(\mathbf{x}) \qquad \boxed{\nabla_{\mathbf{x}} \log Z_{\theta}} \rightarrow \mathbf{0} \\ &= \boldsymbol{s}_{\theta}(\mathbf{x}) \end{aligned}$$

- 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 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**.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

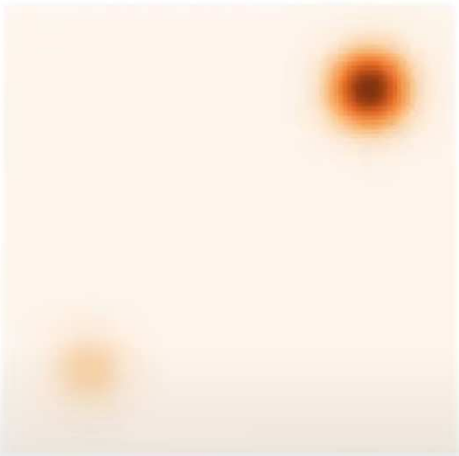
[Flexible models]

Flexible models

Score models can be estimated from data

Training data

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



Probability density function

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

- Suppose we have collected a large trained data set, and again we use x1, x2, to xN 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 <u>properly normalized statistical model</u>** <u>to</u> **<u>estimate the underlying data density</u>** <u>using methods such as</u> **<u>maximum likelihood</u>**.

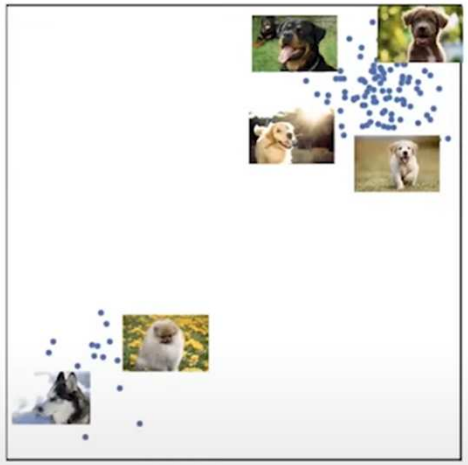
Normalized statistical model
- probability density estimation

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Flexible models]

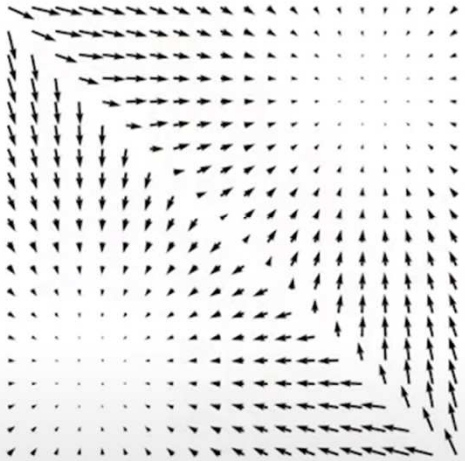
Flexible models

Score models can be estimated from data



Training data

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



Score function

$$\boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$$

- 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**.

Score model
- Score estimation

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Flexible models]

Flexible models

Score models can be estimated from data

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

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

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

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.**

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Flexible models]

Flexible models

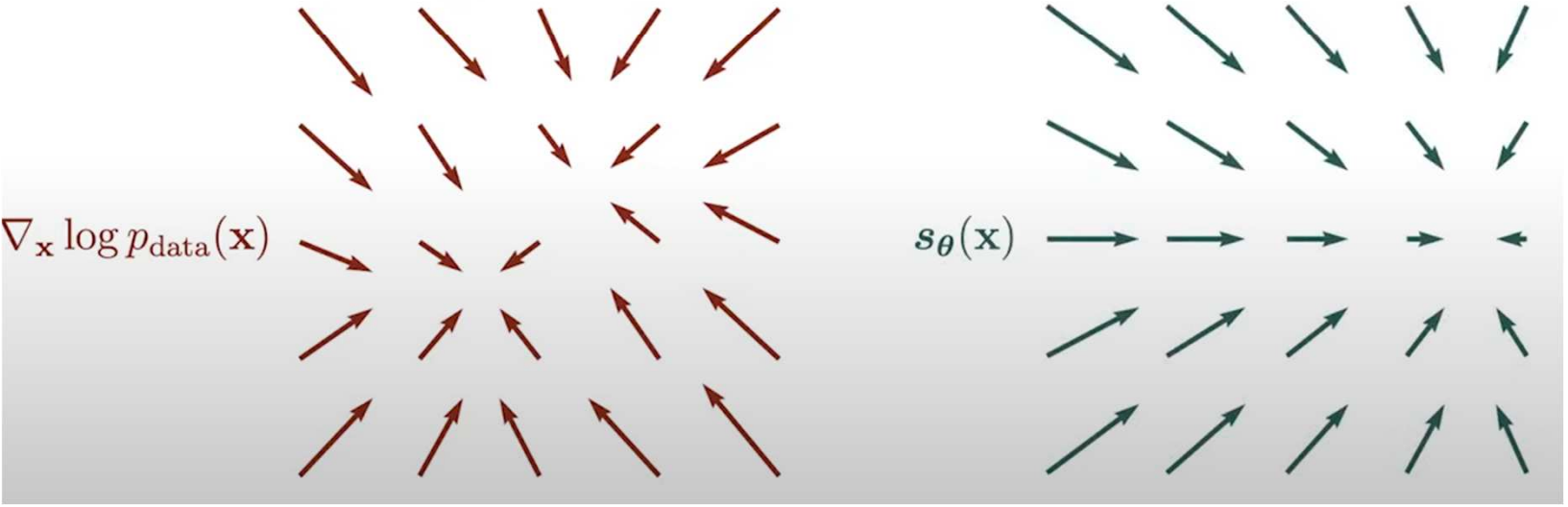
Score models can be estimated from data

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

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

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

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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Flexible models]

Flexible models

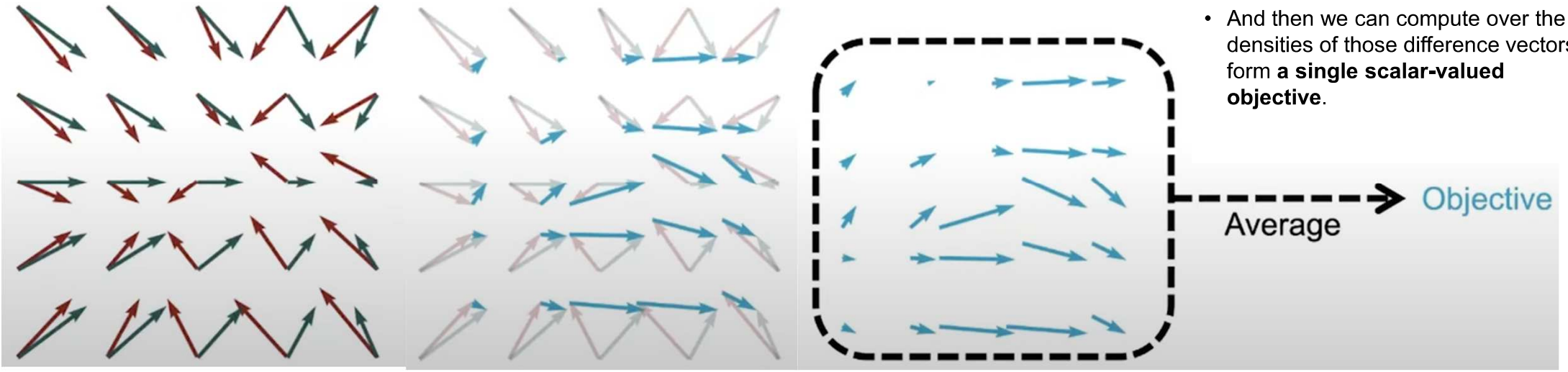
Score models can be estimated from data

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

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

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

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 compute over the densities of those difference vectors to form **a single scalar-valued objective**.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Flexible models]

Flexible models

Score models can be estimated from data

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

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

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

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)

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

(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,

- Fisher Divergence : 두 분포의 gradient간의 Euclidean Distance의 차이를 의미하며, $\nabla_x \log p_{data}(x) \approx s_\theta(x)$에 가까워질수록 0에 수렴함

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Flexible models]

Flexible models

Score models can be estimated from data

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

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

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

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]:

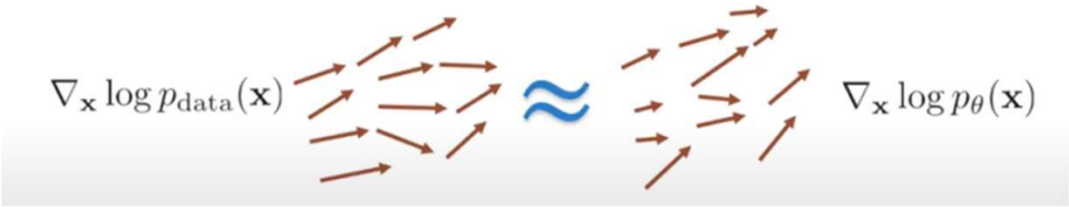
$$\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\frac{1}{2}\|s_\theta(\mathbf{x})\|_2^2 + \text{trace}(\underbrace{\nabla_{\mathbf{x}} s_\theta(\mathbf{x})}_{\text{Jacobian of } s_\theta(\mathbf{x})})\right]$$

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

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.

[Score Matching] 고급생성모델 입문서 https://wikidocs.net/230524



$$\because s_\theta(x) = \nabla_x \log p_\theta(x)$$

score-based 모델 $s_\theta(x) = \nabla_x \log p_\theta(x)$을 데이터의 gradient $\nabla_x \log p_{data}(x)$에 근사하도록 학습하기 위해 KL-Divergence, JS Divergence와 같이 두 확률 분포의 gradient 간의 l2-norm의 차이를 수치화하는 **Fisher Divergence**을 학습 목표로 설정합니다.

$$\frac{1}{2} E_{x \sim p_{data}(x)}[||\nabla_x \log p_{data}(x) - s_\theta(x)||_2^2]$$

Fisher Divergence는 두 분포의 gradient간의 유클리디안 거리의 차이를 의미하며, $\nabla_x \log p_{data}(x) \approx s_\theta(x)$에 가까워질수록 0에 수렴합니다.

Changes of Variables

모델을 학습하기 이전 하나의 문제점을 짚고 넘어가야 합니다. 그것은 바로 저희가 확률 분포 $\log p_{data}(x)$를 알지 못한다는 점이고, 특히 gradient인 $\nabla_x \log p_{data}(x)$은 더더욱 알기 어렵습니다.

이 문제를 해결하기 위해 (Hyvarinen, 2005)은 $\nabla_x \log p_{data}(x)$와 $s_\theta(x)$간의 l2-norm function을 Score function의 식 $s_\theta(x)$으로 근사하는 학습 목표를 설정했습니다. 두 가지 변수 $\nabla_x \log p_{data}(x)$와 $s_\theta(x)$을 $s_\theta(x)$으로 압축하여 표현하는 방법을 Changes of Variables Trick이라고 명칭합니다.

$$E_{x \sim p_{data}(x)}[\frac{1}{2}||s_\theta(x)||_2^2 + tr(\nabla_x s_\theta(x))]$$

즉 기존 학습목표에서 $\nabla_x \log p_{data}(x)$을 제거하고 계산할 수 있는 $s_\theta(x)$의 식으로 나타냈고, $s_\theta(x)$만으로 표현한 변형된 학습 목표를 **Score Matching Objective**이라고 합니다.

$s_\theta(x)$에 대한 학습 목표에 데이터셋 $D = [x_1, x_2, \cdots, x_n]$을 가지고 몬테 카를로 방법을 적용해 학습 목표 $J(\theta, D)$를 다음과 같이 나타낼 수 있습니다.

$$J(\theta, D) = \frac{1}{n} \sum_{i=1}^{n} [\frac{1}{2}||s_\theta(x)||_2^2 + tr(\nabla_x s_\theta(x))]$$

- log likelihood $\log p(x)$이 아닌 $\nabla_x \log p_\theta(x)$을 구하므로 $L(\theta, D)$이 아닌 $J(\theta, D)$로 나타냅니다.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

Change of Variables 증명

$$\frac{1}{2}E_{x\sim p_{data}(x)}[||\nabla_x \log p_{data}(x) - s_\theta(x)||_2^2]$$

$$= \frac{1}{2}\int_{-\infty}^{\infty} p(x)(\nabla_x \log p(x))^2 dx + \frac{1}{2}\int_{-\infty}^{\infty} p(x)s_\theta(x)^2 dx - \int_{-\infty}^{\infty} p(x)\nabla_x \log p(x) s_\theta(x)^2 dx$$

첫 번째 식인 $\frac{1}{2}\int_{-\infty}^{\infty} p(x)(\nabla_x \log p(x))^2 dx$ 은 상수이므로 제거할 수 있습니다. 또한, 세번째 식은 다음과 같이 나누어 단순화 할 수 있습니다.

$$\int_{-\infty}^{\infty} p(x)\nabla_x \log p(x) s_\theta(x)^2 dx = \int_{-\infty}^{\infty} p(x)\frac{\nabla_x p(x)}{p(x)} s_\theta(x)^2 dx = \int_{-\infty}^{\infty} \nabla_x p(x) s_\theta(x)^2 dx$$

$$\therefore \left[\int_{-\infty}^{\infty} p(x)s_\theta(x)dx\right] - \int_{-\infty}^{\infty} p(x)\nabla_\theta s_\theta(x)dx$$

세번째 식을 단순화한 식의 첫 번째 식 $[p(x)s_\theta(x)]_{-\infty}^{\infty}$은 x가 어떤 유한한 숫자보다 커지면 사라지는 **bounded support**의 특징을 가지고 있으므로 0으로 수렴합니다.

즉 남은 식을 종합하면 $\frac{1}{2}\int_{-\infty}^{\infty} p(x)s_\theta(x)^2 dx$와 $\int_{-\infty}^{\infty} p(x)\nabla_x s_\theta(x) dx$이 남고, 이 두 식을 정리하면 저희가 앞서 change of variables 식으로 구한 학습 목표를 구할 수 있습니다.

$$E_{x\sim p_{data}(x)}[\frac{1}{2}||s_\theta(x)||_2^2 + tr(\nabla_x s_\theta(x))]$$

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Matching] 고급생성모델 입문서 https://wikidocs.net/230524

$$\frac{1}{n}\sum_{i=1}^{n}[\frac{1}{2}||s_\theta(x)||_2^2 + tr(\nabla_x s_\theta(x))]$$

비록 Change of Variables Theorem을 사용해 두 가지 변수 $\nabla_x \log p_{data}(x)$와 $s_\theta(x)$을 $s_\theta(x)$으로 압축하여 표현한 score matching function을 구했지만, 여전히 $\nabla_x s_\theta(x)$의 대각합(trace) $tr(\nabla_x s_\theta(x))$을 구하기 어렵다는 문제가 남아있습니다.

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

이는 대각합 $tr(\nabla_x s_\theta(x))$을 구하기 위해서는 데이터셋 D에 존재하는 모든 x_i에 대해 gradient을 구해야 하기 때문입니다.

이 때문에 저차원의 데이터에 대해서는 계산이 용이하지만, 고차원의 데이터에 대해서는 연산에 문제가 발생합니다. 즉 대각합 $tr(\nabla_x s_\theta(x))$은 확장성 오류(scalability issue)를 가지고 있습니다.

To solve the scalability issue of score matching

① **Sliced Score Matching**

② **Denoising Score Matching**

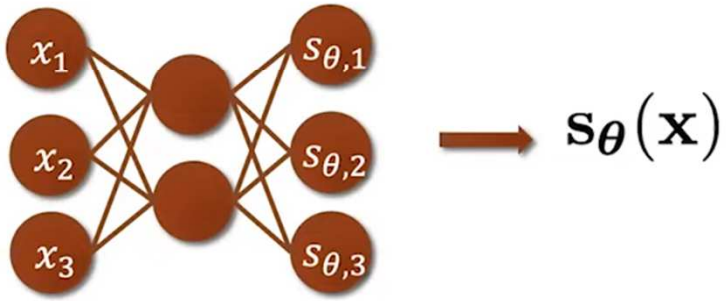
CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Flexible models]

Flexible models

Score Matching is not scalable

- Deep score models



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

- 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 Jacobin of the score model**.

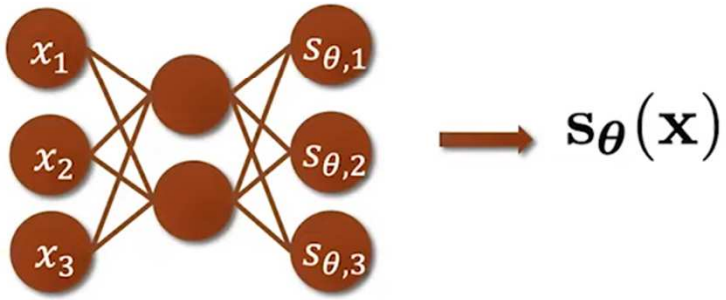
CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Flexible models]

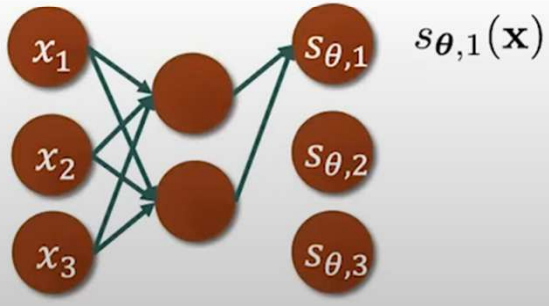
Flexible models

Score Matching is not scalable

- Deep score models



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



- For **the first term**, it is super simple to compute and very efficient because we just need the 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**.

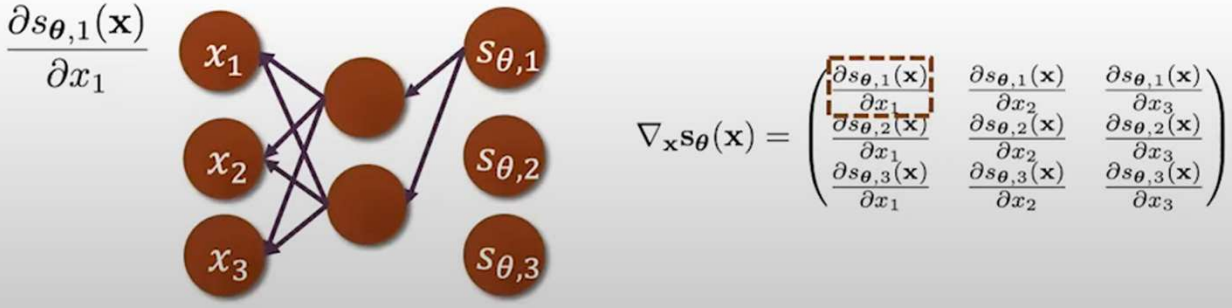
CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

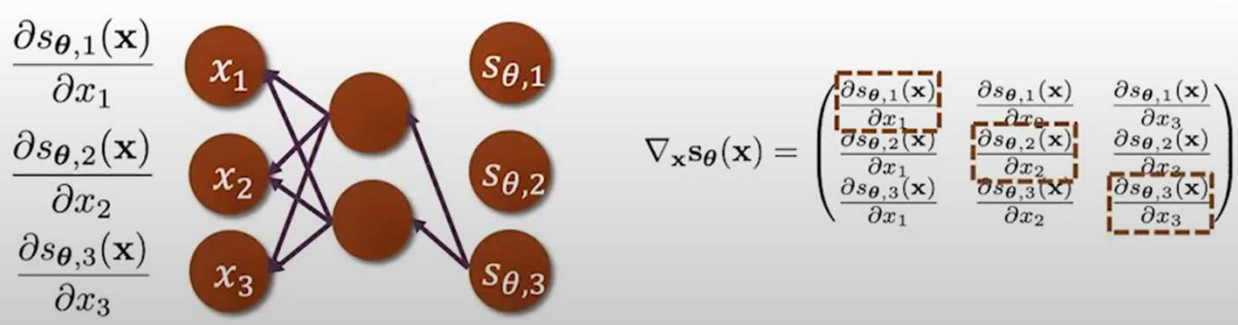
[Flexible models]

Flexible models

Score Matching is not scalable

- Compute $\|\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x})\|_2^2$ and $\operatorname{trace}(\nabla_{\mathbf{x}}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}))$





- 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**.

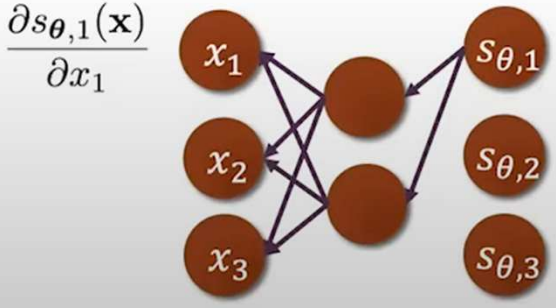
CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Flexible models]

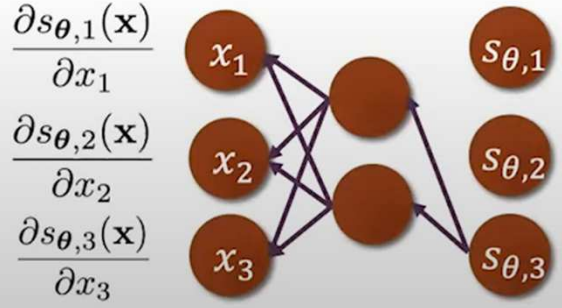
Flexible models

Score Matching is not scalable

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



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



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

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

- 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 high dimensions.
- This means **score matching in its naive form is not scalable**.

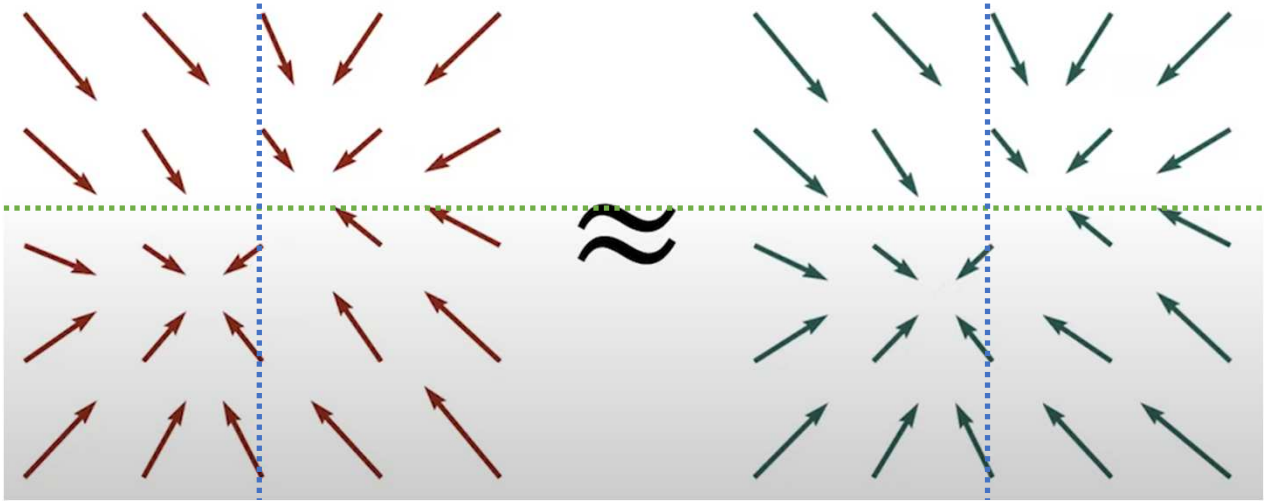
CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Sliced Score Matching]

Flexible models

Sliced score matching

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

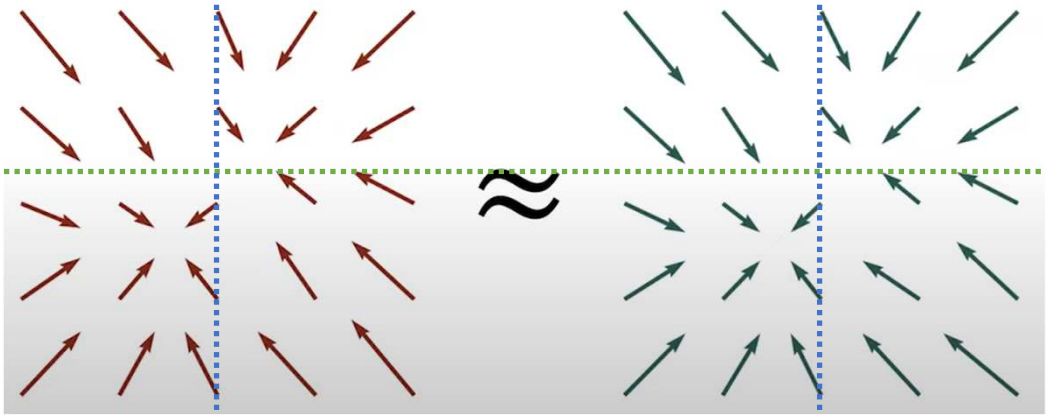


Random Direction Vector v

- 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 random 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.

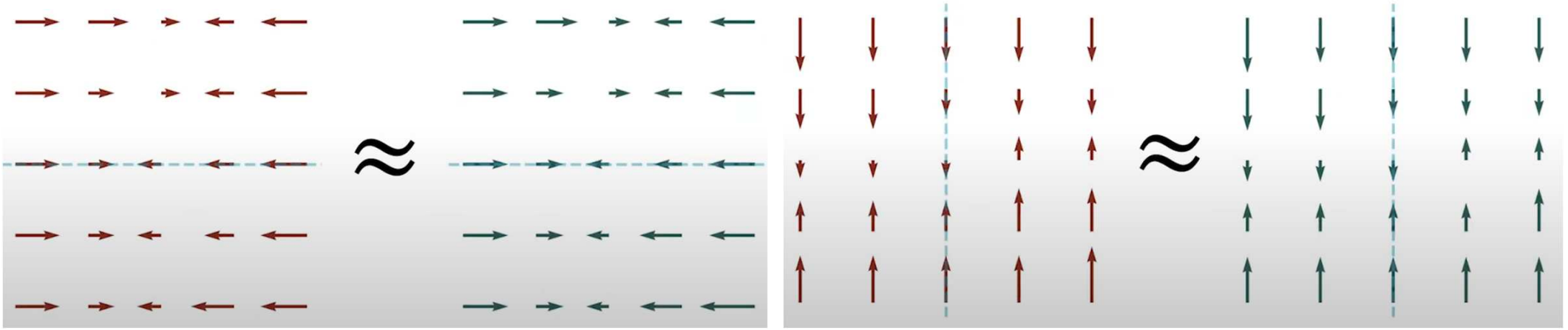
CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Sliced Score Matching]



Random Direction Vector v

- [Random Project from mD to 1D]
- We **project the high-dimensional vector fields to random 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.



Projection to Random Direction Vector v

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[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**

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

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

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

Scalable!

- We can capture this intuition with the concept of **sliced Fisher divergence**.
- Here **v** denotes **the projection direction**. It is a vector. And **pv** denotes **the distribution of those projection directions**.
- We compute the inner product of **v** and those two score 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.

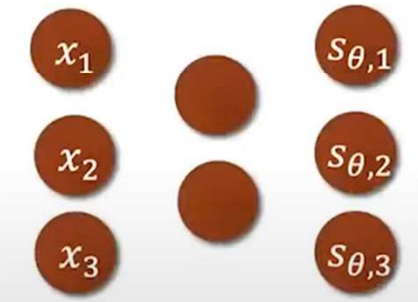
CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Sliced Score Matching]

Flexible models

Computing Jacobian-vector products is scalable

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



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

- This is actually not hard to see because we can rewrite **the Jacobian vector product** as an alternative form on the right-hand side.
- This just requires us **to swap the location of v and S_theta within the gradient operator**.
- So now I will show you how to compute this vector Jacobian vector product very efficiently.

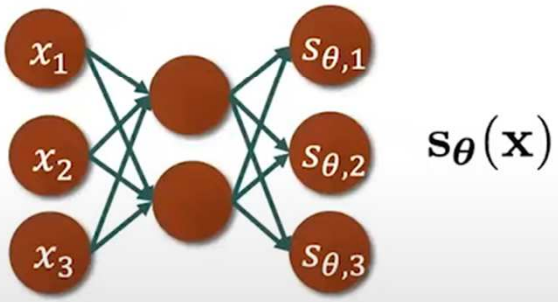
CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Sliced Score Matching]

Flexible models

Computing Jacobian-vector products is scalable

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



$\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x})$

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

- First, we just need one forward propagation to get the output of s_theta,

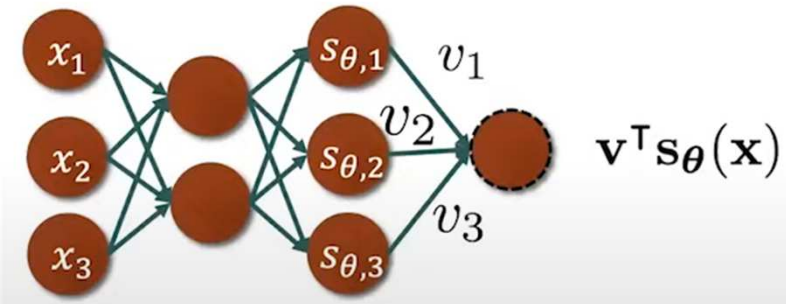
CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Sliced Score Matching]

Flexible models

Computing Jacobian-vector products is scalable

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



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

- And then we can directly compute **the inner product between v and s_theta**.
- So this amounts to adding one additional neuron to the computational graph.

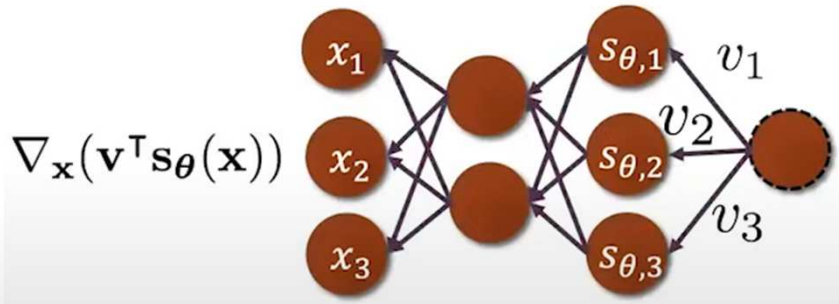
CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Sliced Score Matching]

Flexible models

Computing Jacobian-vector products is scalable

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



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

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

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

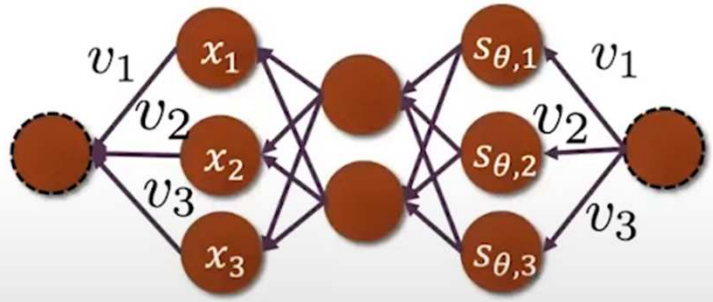
[Score Model – Sliced Score Matching]

Flexible models

Computing Jacobian-vector products is scalable

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

$\mathbf{v}^\intercal \nabla_{\mathbf{x}} (\mathbf{v}^\intercal \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}))$



One Backprop!
Sliced Score Matching is scalable

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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Sliced Score Matching]

Flexible models

Sliced score matching

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

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

- This is how sliced score matching works in practice.

① We just **sample a minibatch of data points from our data set**.

② And for each data point, we **sample one single projection direction from our distribution of pv**.

③ 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Sliced Score Matching]

Flexible models

Sliced score matching

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

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

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

- So the projection distribution **pv** 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Denoising Score Matching]

고급생성모델 입문서 https://wikidocs.net/230524

Denoising Score Matching은 Change of Variables Trick을 사용한 이전 Score Matching 방법들과 다르게 $\nabla_x \log p_{data}(x)$을 계산하기 위해 데이터 x에 간단한 가우시안 노이즈를 추가해 계산 불가능한 $\nabla_x \log p_{data}(x)$을 계산이 가능한 $\nabla_{\tilde{x}} \log q_\sigma(\tilde{x})$식으로 변형합니다.

$$\tilde{x} = x + \sigma \circ \epsilon \text{ for } x \sim p(x), \epsilon \sim N(0,1)$$

- 데이터 $x \rightarrow \tilde{x}$
- 확률 분포 $p_{data}(x) \rightarrow q_\sigma(\tilde{x})$

기존 분포 $p(x)$에 가우시안 노이즈를 추가한 $q_\sigma(\tilde{x})$와 score function $s_\theta(x)$의 학습 목표는 다음과 같습니다. (Vincent, 2011)

$$\frac{1}{2} E_{\tilde{x} \sim q_\sigma(\tilde{x})}[||\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}) - s_\theta(\tilde{x})||_2^2] = \frac{1}{2} E_{x \sim p} E_{\tilde{x} \sim q_\sigma(\tilde{x}|x)}[||\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x) - s_\theta(\tilde{x})||_2^2]$$

즉 Change of Variables Trick을 사용하지 않으면서 $\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x)$와 $s_\theta(\tilde{x})$의 l2-norm을 학습할 수 있습니다. 또한 가우시안 노이즈의 gradient $\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x)$는 $\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x) = -\frac{x-\tilde{x}}{\sigma^2}$이므로 계산이 복잡하지 않습니다.

하지만 Denoising Score Matching 방법을 사용하면 $p_{data}(x)$ 확률 분포가 아닌 가우지안 노이즈를 추가한 $q_\sigma(\tilde{x})$분포에 대해 학습이 이루어지기 때문에, 이 노이지한 분포에서 노이즈를 제외하는 Denoising Process을 적용해야 합니다.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Denoising Score Matching]

Flexible models

Denoising score matching

$\mathbf{x}$

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

$q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$

$\tilde{\mathbf{x}}$

$q_\sigma(\tilde{\mathbf{x}})$

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

$\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}})$

- 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 the denoising score matching, we need to **design a perturbation kernel** which we denote as **q_sigma**. So **x_tilde** denotes the perturbed data point, and **x** denotes the original noise-free data point. **Sigma** can typically be a Gaussian distribution with means x and a standard deviation sigma.
- So after converting this perturbation kernel with our original data distribution, we get **a noisy data distribution** to **q_sigma of x_tilde**.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Denoising Score Matching]

Flexible models

Denoising score matching

$\mathbf{x}$

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

$q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$

$\tilde{\mathbf{x}}$

$q_\sigma(\tilde{\mathbf{x}})$

- Matching the score of a noise-perturbed distribution

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

- 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Denoising Score Matching]

Flexible models

Denoising score matching

$\mathbf{x}$

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

$q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$

$\tilde{\mathbf{x}}$

$q_\sigma(\tilde{\mathbf{x}})$

- 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)

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

Scalable

- So you can use some arithmetic derivation to write down an equivalent form to **the denoising score matching objective**.
- 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Denoising Score Matching]

Flexible models

Denoising score matching

$\mathbf{x}$

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

$q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$

$\tilde{\mathbf{x}}$

$q_\sigma(\tilde{\mathbf{x}})$

- 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 what are 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**.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Denoising Score Matching]

Flexible models

Denoising score matching

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

$\mathbf{x}$

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

$q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$

$\tilde{\mathbf{x}}$

$q_\sigma(\tilde{\mathbf{x}})$

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

Middle term of Denoising score matching objective

- 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Denoising Score Matching]

Flexible models

Denoising score matching

- Sample a minibatch of datapoints $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\} \sim p_{\text{data}}(\mathbf{x})$
- Sample a minibatch of perturbed datapoints $\{\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \cdots, \tilde{\mathbf{x}}_n\} \sim q_\sigma(\tilde{\mathbf{x}})$

$$\tilde{\mathbf{x}}_i \sim q_\sigma(\tilde{\mathbf{x}}_i \mid \mathbf{x}_i)$$

- Estimate the denoising score matching loss with empirical means

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

- 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Denoising Score Matching]

Flexible models

Denoising score matching

- Sample a minibatch of datapoints $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\} \sim p_{\text{data}}(\mathbf{x})$
- Sample a minibatch of perturbed datapoints $\{\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \cdots, \tilde{\mathbf{x}}_n\} \sim q_\sigma(\tilde{\mathbf{x}})$

$$\tilde{\mathbf{x}}_i \sim q_\sigma(\tilde{\mathbf{x}}_i \mid \mathbf{x}_i)$$

- Estimate the denoising score matching loss with empirical means

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

- If Gaussian perturbation

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

- 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.

Diffusion and Score-Based Generative Models

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model – Summary]

고급생성모델 입문서 https://wikidocs.net/232611

Score-Based 모델은 $\nabla_x \log p_{data}(x)$에 근사하는 score function $s_\theta(x) = \nabla_x \log p_\theta(x)$을 학습하는 생성 모델입니다.

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

즉 $\nabla_x \log p_{data}(x)$와 $s_\theta(x)$ 분포 차이를 Fisher Divergence로 나타낸 학습 목표는 다음과 같습니다.

$$\frac{1}{2} E_{x \sim p_{data}(x)}[||\nabla_x \log p_{data}(x) - s_\theta(x)||_2^2]$$

데이터셋 $D = [x_1, x_2, \cdots, x_n]$을 학습 목표에 몬테 카를로 방법 적용해 모델의 파라미터 θ를 업데이트 할 수 있습니다.

$$J(\theta, D) = \frac{1}{n} \sum_{i=1}^{n} [\frac{1}{2} ||s_\theta(x^{(i)})||_2^2 + tr(\nabla_x s_\theta(x^{(i)}))]$$

Score Based Model Gradients

- Sliced Score Matching

$$E_{x \sim v_{q(v)}} E_{x \sim p_{data}(x)}[v^T \nabla_x s_\theta(x) v + \frac{1}{2}(v^T s_\theta(x))^2]$$

$$J_{sliced}(\theta, D) = \frac{1}{n} \sum_{i=1}^{n} [v^T \nabla_x s_\theta(x^{(i)}) v + \frac{1}{2}(v^T s_\theta(x^{(i)}))^2]$$

$$\nabla_\theta J_{sliced}(\theta, D) = \frac{1}{n} \sum_{i=1}^{n} [\nabla_\theta (v^T \nabla_{x^{(i)}} s_\theta(x^{(i)}) v) + \nabla_\theta (\frac{1}{2}(v^T s_\theta(x^{(i)}))^2)]$$

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \circ \nabla_\theta J(\theta^{(t)})$$

- Denoising Score Matching

$$\frac{1}{2} E_{x \sim p} E_{\tilde{x} \sim q_\sigma(\tilde{x}|x)}[||\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x) - s_\theta(\tilde{x})||_2^2]$$

$$J_{denoised}(\theta, D) = \frac{1}{n} \sum_{i=1}^{n} ||\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}^{(i)}|x^{(i)}) - s_\theta(\tilde{x}^{(i)})||_2^2$$

$$\nabla_\theta J_{denoised}(\theta, D) = \frac{1}{n} \sum_{i=1}^{n} ||\nabla_\theta (\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}^{(i)}|x^{(i)})) - \nabla_\theta (s_\theta(\tilde{x}^{(i)}))||_2^2$$

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \circ \nabla_\theta J(\theta^{(t)})$$

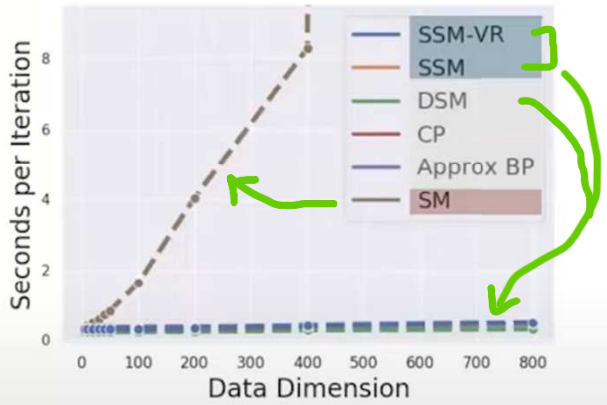
CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model]

Flexible models

Results: Sliced Score Matching for EBMs

Sliced score matching methods | Other Baselines | Score Matching



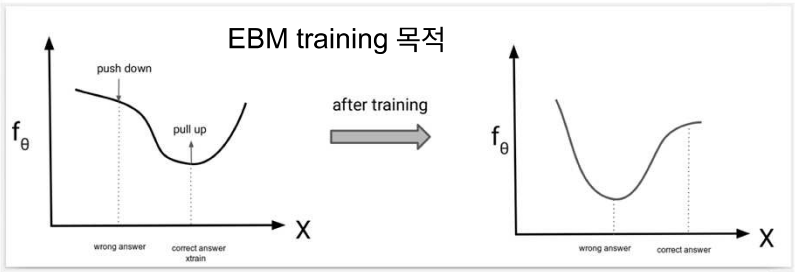
Efficiency

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

Energy-based models (EBM) : 데이터의 확률 분포를 energy function을 활용해 정의한 모델로, observed (혹은 more likely) data point에는 더 적은 에너지를, outlier (혹은 unlikely) data point에는 더 높은 에너지를 할당

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

Z(θ) : normalizing constant



- 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.

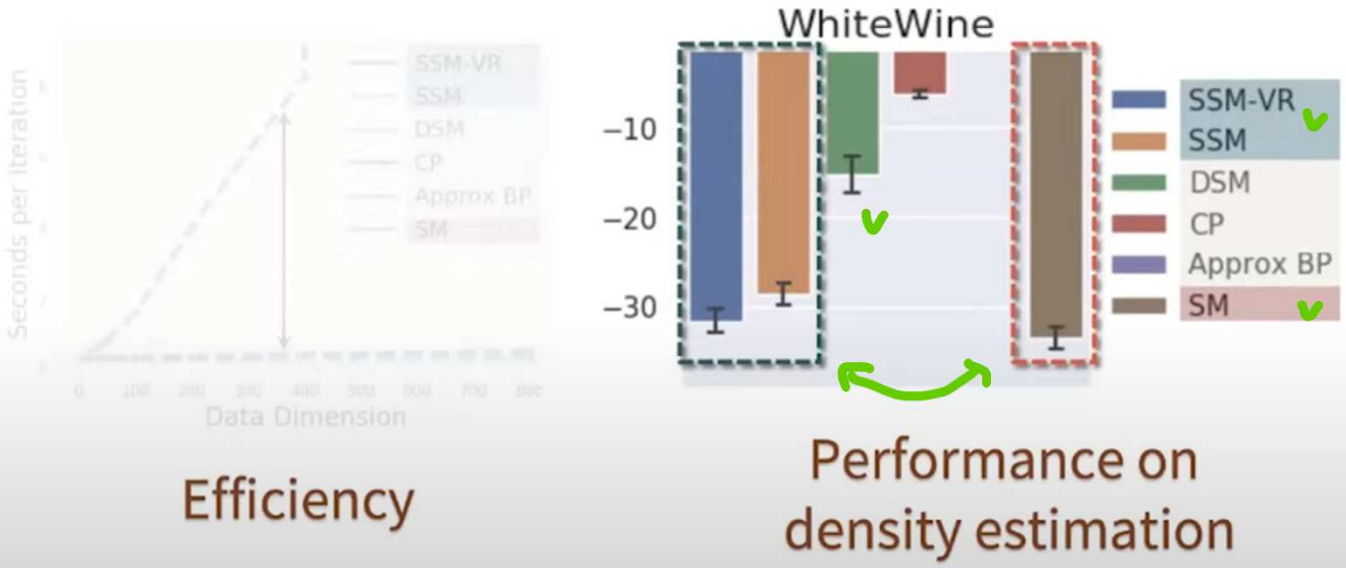
CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Score Model]

Flexible models

Results: Sliced Score Matching for EBMs

Sliced score matching methods | Other Baselines | Score Matching

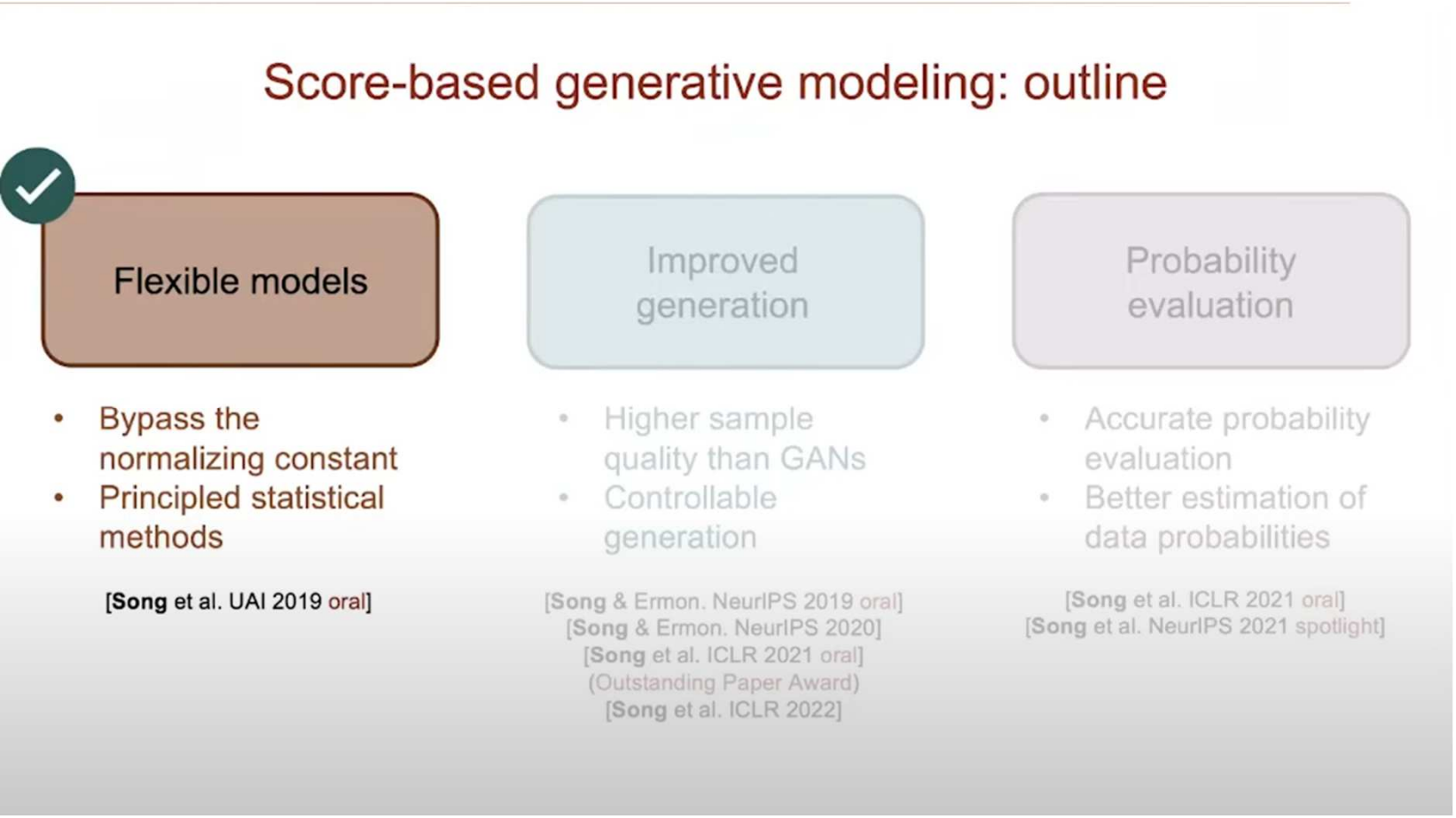


Song*, Garg*, Shi, Ermon. "Sliced Score Matching: A Scalable Approach to Density and Score Estimation." UAI 2019.

- 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.

CVPR2022 Tutorial – Yang Song
https://www.youtube.com/watch?v=wMmqCMwuM2Q

[Flexible Models]



- 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**.