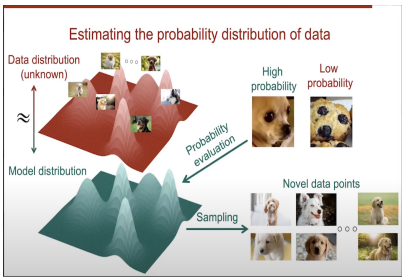
Diffusion and Score-Based Generative Models

- A typical assumption in statistics and Machine learning is that all data points in our training data set come from some underlying data distribution.



In other words, those data points are basically i.i.d. (independent and identically distributed) samples from given data distribution.

However, we don't have the analytical form of the data distribution of training dataset, and we need estimate it.

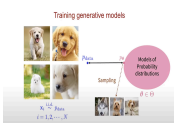
To estimate this data distribution, we need to create a model that represents parameterized probability distribution, which we call "the model distribution"

We hope to tune this model parameter to make sure this "model distribution" is close to the data distribution in a certain sense

If this model distribution is very close to the data distribution, then we can use the model for many applications, such as generating unlimited number of novel data points just by sampling from this model distribution.

Because this model distribution provides a way to generate novel data points, we also refer to it as a "Generative model"

Training Generative models



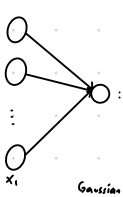
Our model provides a family of probability distributions. And we hope to find a single probability distribution inside huge family by minimizing the distances from P_{data} to P_θ

Key challenge for building complex generative models.

① P_{data} is extremely complex for high dimensional data.

↳ How to build a complex model to fit the P_{data}?

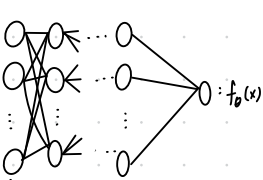
1. Gaussian distribution: It's too simple to approximate complicated P_{data}, but it serves as a good starting point.



$$: P_\theta(x) = \frac{1}{(2\pi)^{d/2}} e^{-\frac{\|x-\mu\|^2}{2\sigma^2}}$$

Gaussian distribution is basically a computational graph that has two layers, where the first layer corresponds to the input data point and second layer is a single unit that basically gives you the probability density function of Gaussian distribution

2. But, Gaussian models are too simple. Let's leverage a bigger and deeper Network



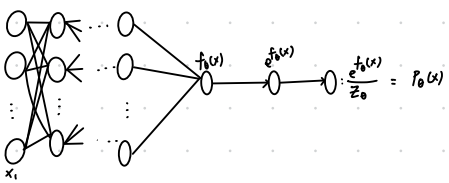
Deep Neural Network

But, it's actually non trivial to use 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 dimensional input x to a typically one dimensional output $f_\theta(x)$.

The output value $f_\theta(x)$ does not directly model the distribution because it may not be positive everywhere. So, first step to convert $f_\theta(x)$ into a probability density is to

① take the exponential of the output. $e^{f_\theta(x)}$, so then the output becomes positive.

② Then, we normalize the output by dividing by a constant Z_θ, in order to construct a probability distribution which has positive values everywhere and also properly normalized.



Z_θ : Normalizing Constant

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

In the special case of Gaussian models

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

However in Deep Neural Network

$\int e^{f_\theta(x)} dx$ becomes intractable to compute

And, this difficulty is the unique challenge in deep generative modeling

Tackling the intractable normalizing constant

① Approximating the normalizing constant.
- Energy based models

Disadvantage : Inaccurate probability evaluation

② Using restricted Neural Network models
- VAE

Disadvantages : Restricted model family

③ Modeling the Generation Process Only
- GANs

Disadvantages: Cannot evaluate probabilities

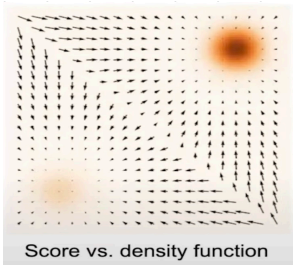
④ Working with score functions

Working with Score functions

$p(x)$: Probability density function

$\nabla_x \log p(x)$: (Stein) Score function

* Be careful this gradient is taken w.r.t the r.v 'x', not 'θ'



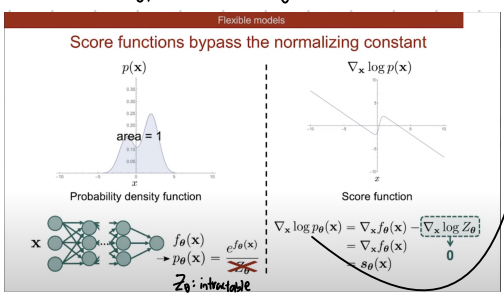
Score function is a vector field that gives the direction where the density function grows most quickly.

So, given the density function, we can compute the score function easily

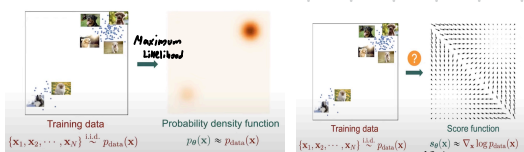
Conversely, with score function, we can recover pdf by computing integrals

Mathematically, this score function preserves all the information in the pdf.

Score functions bypass the normalizing constant



$$\nabla_x \log p_\theta(x) = \nabla_x \log\left(\frac{e^{f_\theta(x)}}{Z_\theta}\right)$$
$$= \nabla_x \log\left(e^{f_\theta(x)}\right) - \nabla_x \log(Z_\theta)$$
$$= \nabla_x f_\theta(x) - \nabla_x \log(Z_\theta)$$
$$= \nabla_x f_\theta(x) - 0$$
$$= s_\theta(x) : \text{Score Model}$$

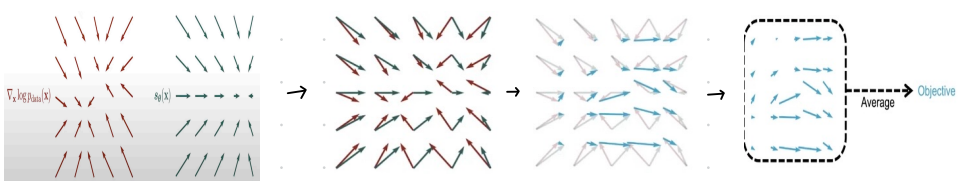


$$\nabla_x \log p_\theta(x) \approx \nabla_x \log p_{\text{data}}(x)$$

We train $p_\theta(x)$ to estimate $p_{\text{data}}(x)$ (the underlying data distribution) using methods such as Maximum Likelihood.

Score models can be estimated from data.

- Given : $\{x_1, x_2, \ldots, x_N\} \overset{i.i.d}{\sim} p_{\text{data}}(x)$
- Goal : $\nabla_x \log p_{\text{data}}(x)$
- Score model : $s_\theta(x) : \mathbb{R}^d \to \mathbb{R}^d \approx \nabla_x \log p_{\text{data}}(x)$
- Objective: How do we compare two vector fields of scores?



Let's recall that those two vector fields ($s_\theta(x)$ and $\nabla_x \log p_{\text{data}}(x)$) actually lie in the same space.

Mathematically, we can capture this intuition with the Fisher divergence objective (Fisher divergence is essentially an expected squared Euclidean distance between data score and the model score averaged over samples from the data)

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

However, Fisher divergence cannot be directly computed because we don't have ground truth value of $p_{\text{data}}(x)$

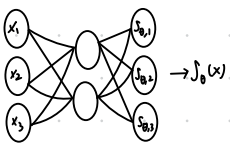
↓

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

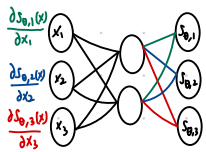
Jacobian of $s_\theta(x)$

$$\frac{1}{2}\mathbb{E}_{p_{data}(x)}\left[\|\nabla_x \log p_{data}(x) - s_\theta(x)\|_2^2\right] = \mathbb{E}_{p_{data}(x)}\left[\frac{1}{2}\|s_\theta(x)\|_2^2 + \text{trace}\left(\nabla_x s_\theta(x)\right)\right]$$
$$\approx \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\|s_\theta(x_i)\|_2^2 + \text{trace}\left(\nabla_x s_\theta(x_i)\right)\Big]$$

Deep Score Models



Compute $\text{trace}\left(\nabla_x s_\theta(x)\right)$

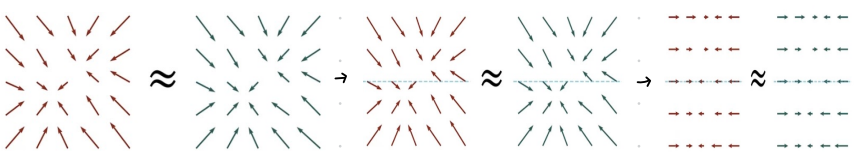


$$\nabla_x s_\theta(x) = \begin{pmatrix} \frac{\partial s_{\theta,1}(x)}{\partial x_1} & \frac{\partial s_{\theta,1}(x)}{\partial x_2} & \frac{\partial s_{\theta,1}(x)}{\partial x_3} \\ \frac{\partial s_{\theta,2}(x)}{\partial x_1} & \frac{\partial s_{\theta,2}(x)}{\partial x_2} & \frac{\partial s_{\theta,2}(x)}{\partial x_3} \\ \frac{\partial s_{\theta,3}(x)}{\partial x_1} & \frac{\partial s_{\theta,3}(x)}{\partial x_2} & \frac{\partial s_{\theta,3}(x)}{\partial x_3} \end{pmatrix} \quad \text{trace}\left(\nabla_x s_\theta(x)\right) = \begin{pmatrix} \frac{\partial s_{\theta,1}(x)}{\partial x_1} & \frac{\partial s_{\theta,1}(x)}{\partial x_2} & \frac{\partial s_{\theta,1}(x)}{\partial x_3} \\ \frac{\partial s_{\theta,2}(x)}{\partial x_1} & \frac{\partial s_{\theta,2}(x)}{\partial x_2} & \frac{\partial s_{\theta,2}(x)}{\partial x_3} \\ \frac{\partial s_{\theta,3}(x)}{\partial x_1} & \frac{\partial s_{\theta,3}(x)}{\partial x_2} & \frac{\partial s_{\theta,3}(x)}{\partial x_3} \end{pmatrix}$$

Computing $\text{trace}\left(\nabla_x s_\theta(x)\right)$ requires the dimensionality of input data point. Hence, score matching in its naive form is not scalable

Sliced Score Matching

- Intuition : One dimensional problems should be easier.
- Idea: Project onto random directions

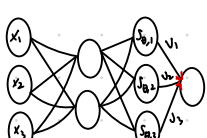


- Random Objective: Sliced Fisher Divergence

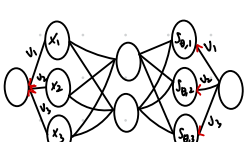
: $\frac{1}{2}\mathbb{E}_{p_v}\mathbb{E}_{p_{data}(x)}\left[\left(v^T\nabla_x \log p_{data}(x) - v^T s_\theta(x)\right)^2\right]$, where v denotes the projection vector, p_v denotes the distribution of projection directions

Sliced Score Matching :

$$\mathbb{E}_{p_v}\mathbb{E}_{p_{data}(x)}\left[\underbrace{v^T\nabla_x s_\theta(x)\, v}_{\text{Scalable}} + \frac{1}{2}\left(v^T s_\theta(x)\right)^2\right]$$



$v^T s_\theta(x)$



: One Backprop!

$v^T\nabla_x\left(v^T s_\theta(x)\right)$

Sliced score matching

① Sample a mini-batch of data points $\{x_1, x_2, \dots, x_N\} \sim p_{data}(x)$

② Sample a mini-batch of projection directions $\{v_1, v_2, \dots, v_N\} \sim p_v$

③ Estimate the sliced score matching loss with empirical means:

$$\frac{1}{N}\sum_{i=1}^{N}\left[v_i^T \nabla_x s_\theta(x_i) v_i + \frac{1}{2}(v_i^T s_\theta(x_i))^2\right]$$

④ The projection distribution is typically Gaussian or Rademacher

⑤ SGD

⑥ Can use more projections per datapoint to boost performance

---

Denoising score matching

- Can also bypass the computational challenge of vanilla score matching.

$p_{data}(x)$

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

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

Idea: Add additional noise to the data point to help us compute the choice of Jacobian term

- Matching the score of a noise-perturbed distribution

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

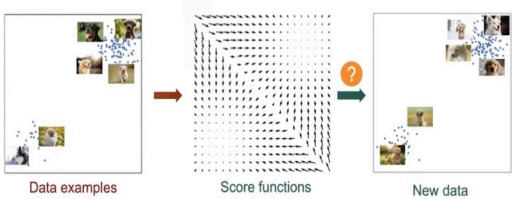
In other words, the key idea of denoising score matching is to estimate the score function of this noise data density ($s_\theta(\tilde{x})$) instead of the score function of the original data density ($s_\theta(x)$)

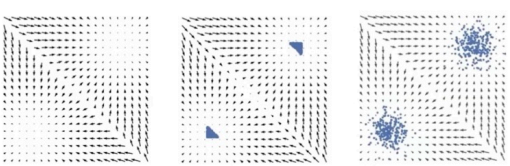
- Denoising score matching

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

- Cannot estimate scores of noise-free distributions.

Sampling from score functions. Langevin dynamics





Score function

$s_\theta(x)$

Follow the scores

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

Follow the noisy scores : Langevin dynamics

$z_t \sim \mathcal{N}(0, I)$

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

Follow the scores: Move all points by following the directions predicted by the score function. However, it won't give us valid samples because all of those points will eventually collapse into each other.

Follow the noisy version of the score function: Inject Gaussian noise to score function and follow those noise perturbed score function

Langevin dynamics sampling

- Goal: Sample from $p(x)$ using only the score $\nabla_x \log p(x)$
- Initialize $x^0 \sim \pi(x)$
- Repeat for $t \leftarrow 1, 2, \ldots, T$

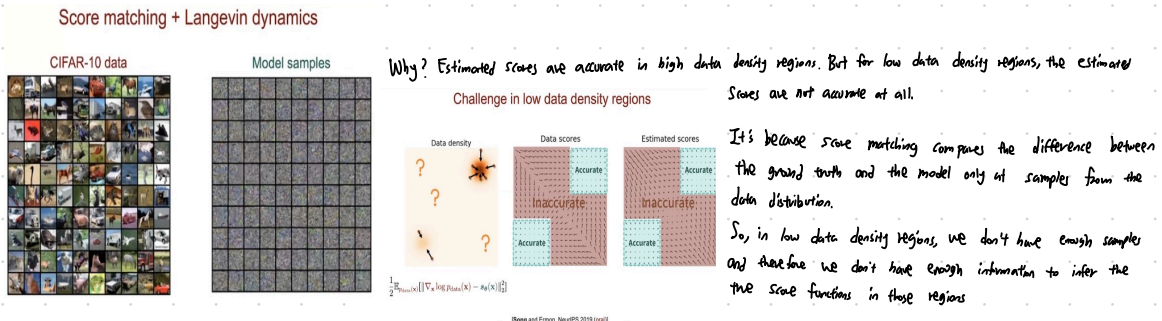
  $z^t \sim N(0, I)$

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

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

  $s_\theta(x) \approx \nabla_x \log p(x)$

However, with this naive approach, we get following results.



Why? Estimated scores are accurate in high data density regions. But for low data density regions, the estimated scores are not accurate at all.

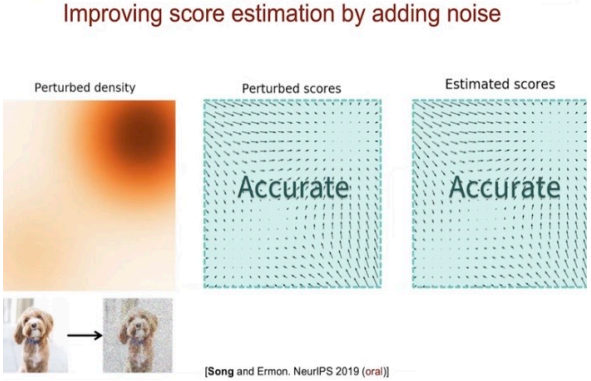
It's because score matching compares the difference between the ground truth and the model only at samples from the data distribution.

So, in low data density regions, we don't have enough samples and therefore we don't have enough information to infer the the score functions in those regions

It's a huge obstacle for Langevin dynamics to provide high quality samples

How can we address this challenge?

1) Inject Gaussian noise to perturb our data points. 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.

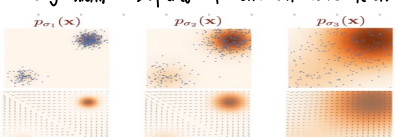


This phenomenon says that the score function of noisy data densities are much easier to be estimated accurately.

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

However, simply injecting Gaussian noise will not solve all the problems. Perturbed density no longer approximates the true data density

: Using multiple sequence of different noise levels (learn low data density region efficiently)



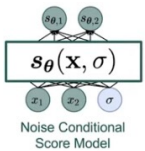
After obtaining noisy data sets, we want to estimate the underlying score function corresponding noisy data densities.

How can we estimate noisy score function?

Naive approach: train 3 networks, and each network is responsible for estimating the score function of a single noise level.

But, naive approach is not a scalable solution, because in practice, we might require much more noise levels.

Scalable solution: Noise Conditional Score Model



It's a simple modification to vanilla 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 to train this noise conditional score model? : We can leverage the idea of score matching

$$: \frac{1}{N}\sum_{i=1}^{N} \underbrace{\lambda(\sigma_i)}_{\text{Positive Weighting function}} \underbrace{\mathbb{E}_{p_{\sigma_i}(x)}\left[\|\nabla_x \log p_{\sigma_i}(x) - s_\theta(x, \sigma_i)\|_2^2\right]}_{\text{Score matching loss}}$$

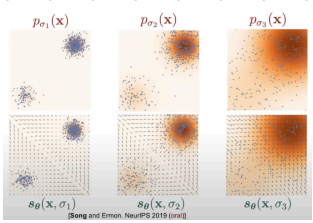
By minimizing this modified score matching loss, we will obtain accurate score estimation for all noise levels

Now, how to generate sample from noise-conditional score model after training with the score matching loss?

: Annealed Langevin dynamics

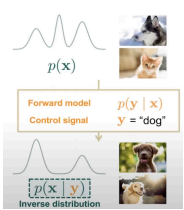
① First, apply Langevin dynamics to sample from the score model with the biggest perturbation noise, and the samples will be used as the initialization to sample from the score model of the next noise level

② Continue until it finally generate samples from the score function with the smallest noise level.



One remarkable property of a score-based generative model is the capability to control the generating process in a principled way

Control the generation process



How to obtain inverse distribution?

Bayes' rule: $P(x|y) = \frac{P(x)P(y|x)}{P(y)}$

Bayes' rule for score functions:

$$\nabla_x \log P(x|y) = \nabla_x \log P(x) + \nabla_x \log P(y|x) - \nabla_x \log P(y)$$

$$= \nabla_x \log P(x) + \nabla_x \log P(y|x) - 0$$

$$\approx s_\theta(x) + \nabla_x \log P(y|x)$$