CS167: Machine Learning

Generative Modeling: Diffusion

November, 18th, 2024

Drake UNIVERSITY

Recap: Generative Model

- Generative modeling can be used on many downstream tasks and applications

- Image generation: generate new image after training from a given image dataset

GAN-based

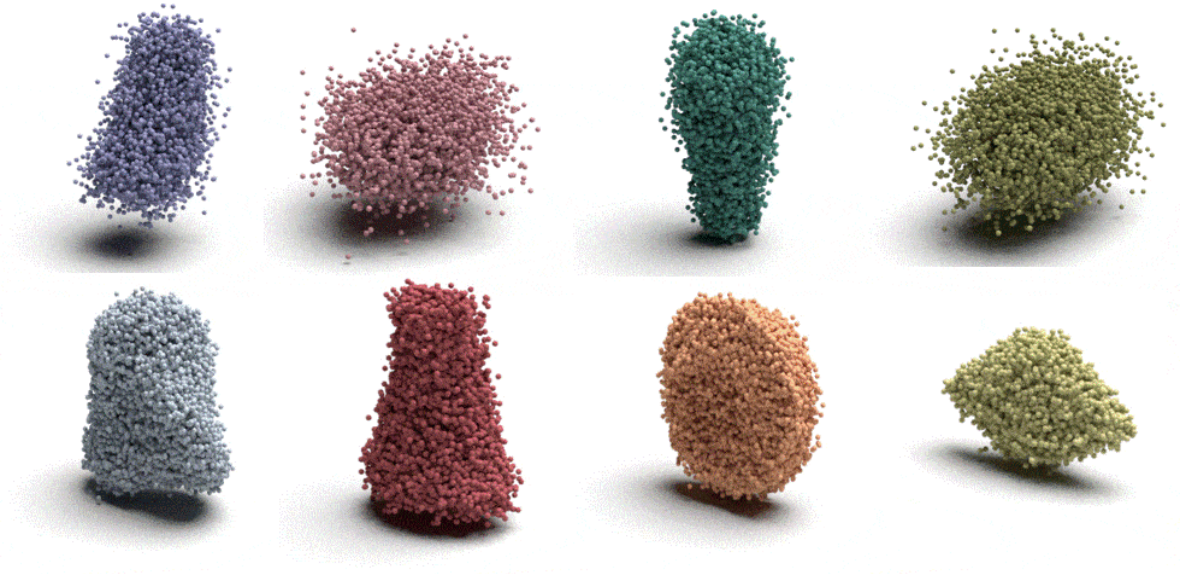
Diffusion-based

Diffusion Models Beat GANs on Image Synthesis - arxiv'21

Recap: Generative Model

- Generative modeling have achieved state-of-the-art performance on many downstream tasks and applications.

- Shape generation: generate shapes from 3D point-cloud



Learning Gradient Fields for Shape Generation - ECCV'20

Recap: Generative Model

- Generative modeling can be used on many downstream tasks and applications

- Audio synthesis: generate audio that sounds realistic

DiffWave: A Versatile Diffusion Model for Audio Synthesis - ICML'21

- Music generation: generate new music

Symbolic Music Generation with Diffusion Models

Recap: Generative Model: Likelihood-Based Model

- Generative modeling is based on representation of probability distribution
- Existing generative modeling techniques can be classified into two broad categories
  - Implicit generative models (GAN)
  - Likelihood-based models (VAE)
  - Score-based models (*diffusion process)

Generative Model: Implicit Model

- Generative modeling is based on representation of probability distribution
- **Implicit models:**
  - probability distribution is represented by a model of its sampling process
  - generative Adversarial Network (GAN) is an example of implicit generative model. It implicitly represents a distribution over all objects that can be produced by the generator network
  - Limitations
    - requires adversarial training which is very unstable (eg, we have seen example of mode collapse)

Generative Model: Likelihood-Based Model

- Generative modeling is based on representation of probability distribution
- **Likelihood-based models:**
  - directly learns the distribution's probability density function (PDF) via maximum likelihood estimation method
  - Variational Autoencoder (VAE) is an example of likelihood-based generative model
  - Limitations
    - rely on surrogate objectives to approximate maximum likelihood training
    - require strong restriction on the model architecture for tractable normalization

Probability Density Function (PDF) Estimation

- Data: $\{x_1, x_2, \ldots, x_n\}$
- True probability density function (PDF): $P(x)$
- We can model this true probability density function (PDF) with the help of a real-valued function parameterized by θ: $f_\theta(x)$
  - So the estimated probability density function (PDF)

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

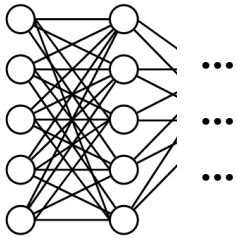
Neural network

Unnormalized probability model

normalizing constant

PDF Estimation using Deep Neural Network (DNN)

x



All the network weights is collectively: θ

$f_\theta(x)$

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

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

normalizing constant

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

Computation for this term
Is intractable for most DNN models

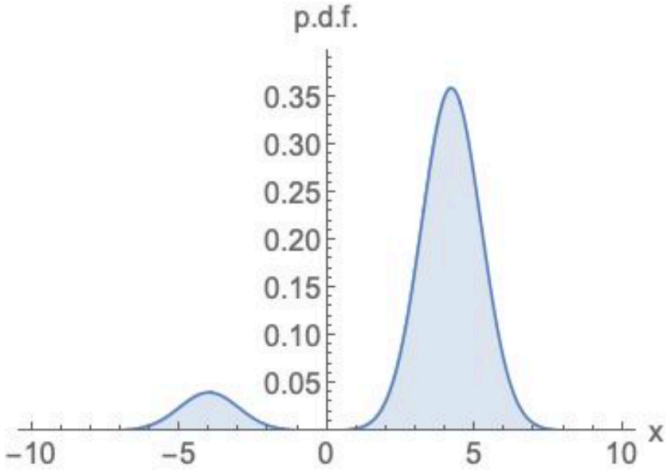
- Intractable normalizing constant Z_θ can be tackled using the followings:
  - Approximate normalizing constant
    - VAEs do that leading to inaccurate probability distribution
  - Restricting the DNN to model simpler PDF as Gaussian distribution
    - Caveat: then this restricted model will be unable to model complex data distribution
  - Generative Adversarial Networks (GAN):
    - Caveat: cannot evaluate probability
- Try alternative method that does not require modeling normalizing constant

Generative Model: Score-Based Model

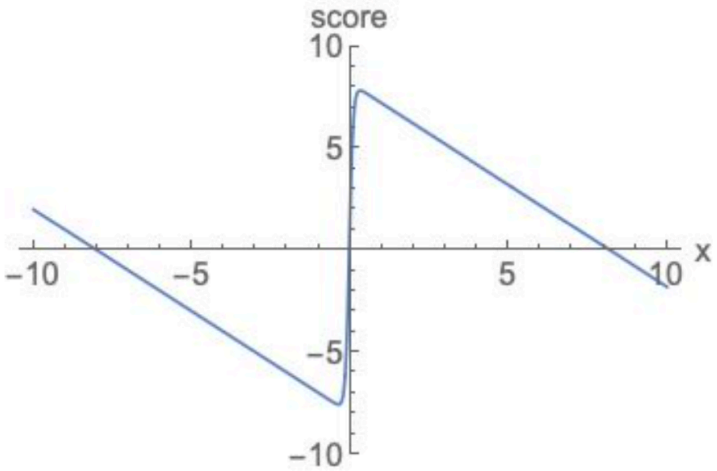
- Score-based models
  - Instead of modeling probability density function (PDF), it models gradient of the log probability density function (log PDF)
  - *Don't require tractable normalizing constant*
  - *Score function of a distribution:* $P(x) = \nabla_x logp(x)$

    gradient of the log pdf
    - this quantity is known as *stein score function*
  - Score-based model: $S_\theta(x) \approx \nabla_x logp(x)$
    - this model tries to approximate gradient of the log probability density function
    - Gradient with respect to x and not with respect to θ

Generative Model: Score-Based Model

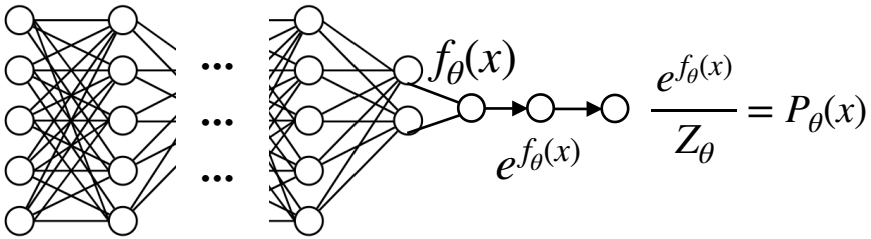


1D probability density function



1D score function

x



All the network weights is collectively: θ

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

$$= \nabla_x f_\theta(x) log_e e - \nabla_x log Z_\theta$$

$$= \nabla_x f_\theta(x) - \nabla_x log Z_\theta$$

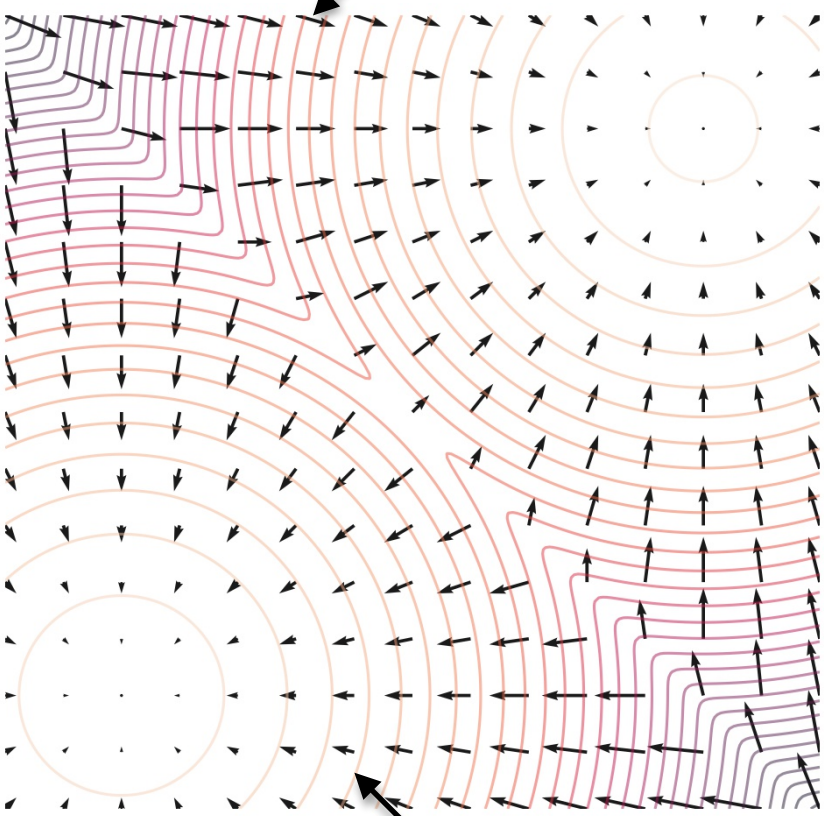
$$= \nabla_x f_\theta(x)$$

0

Derivative of constant is zero

Generative Model: Score-Based Model

- Score vs. probability density function
  - Imagine a probability distribution of a mixture of two 2D Gaussians
  - *probability density function* (contours) and it's *score function* (the vector field)
  - *Score function:*
    - it points in the direction where density function has fastest growth



Generative Model: Score-Based Model

beyond scope of cs195

method that minimizes Fisher Divergence (FD) without knowledge of the ground-truth data score

beyond scope of cs195

an iterative process to draw samples from a trained score-based model

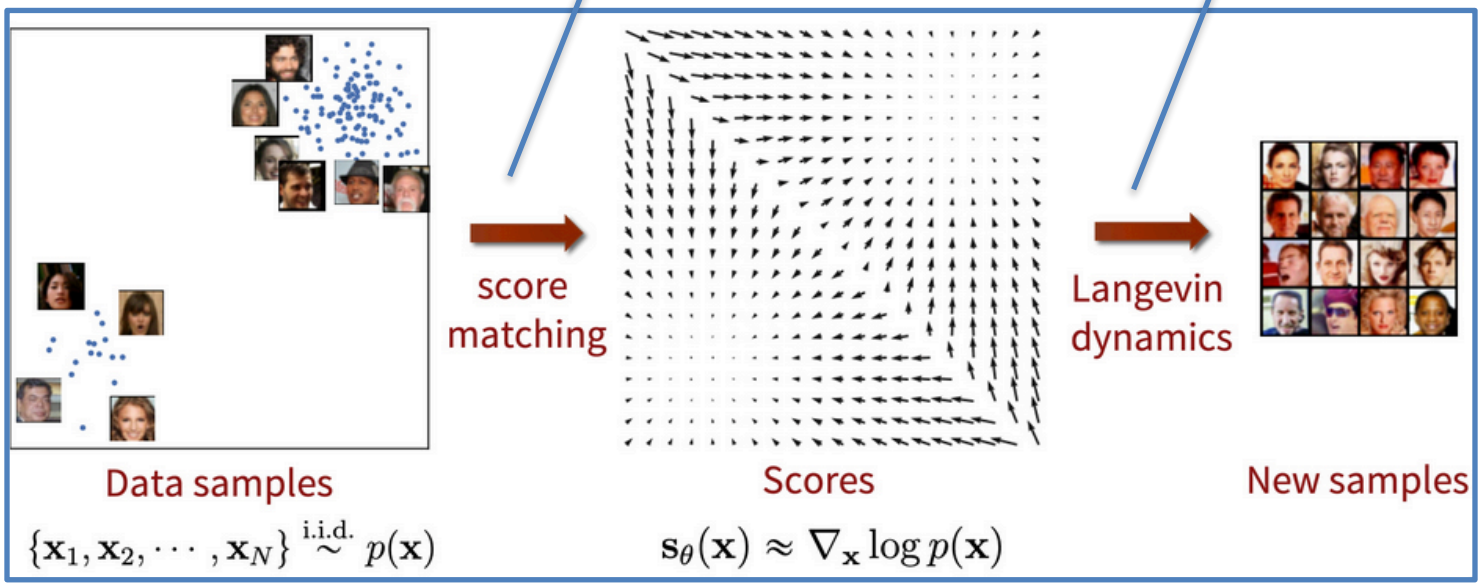


Figure: train a score-based model with score matching, and then produce samples via Langevin dynamics

Generative Model: Score-Based Model

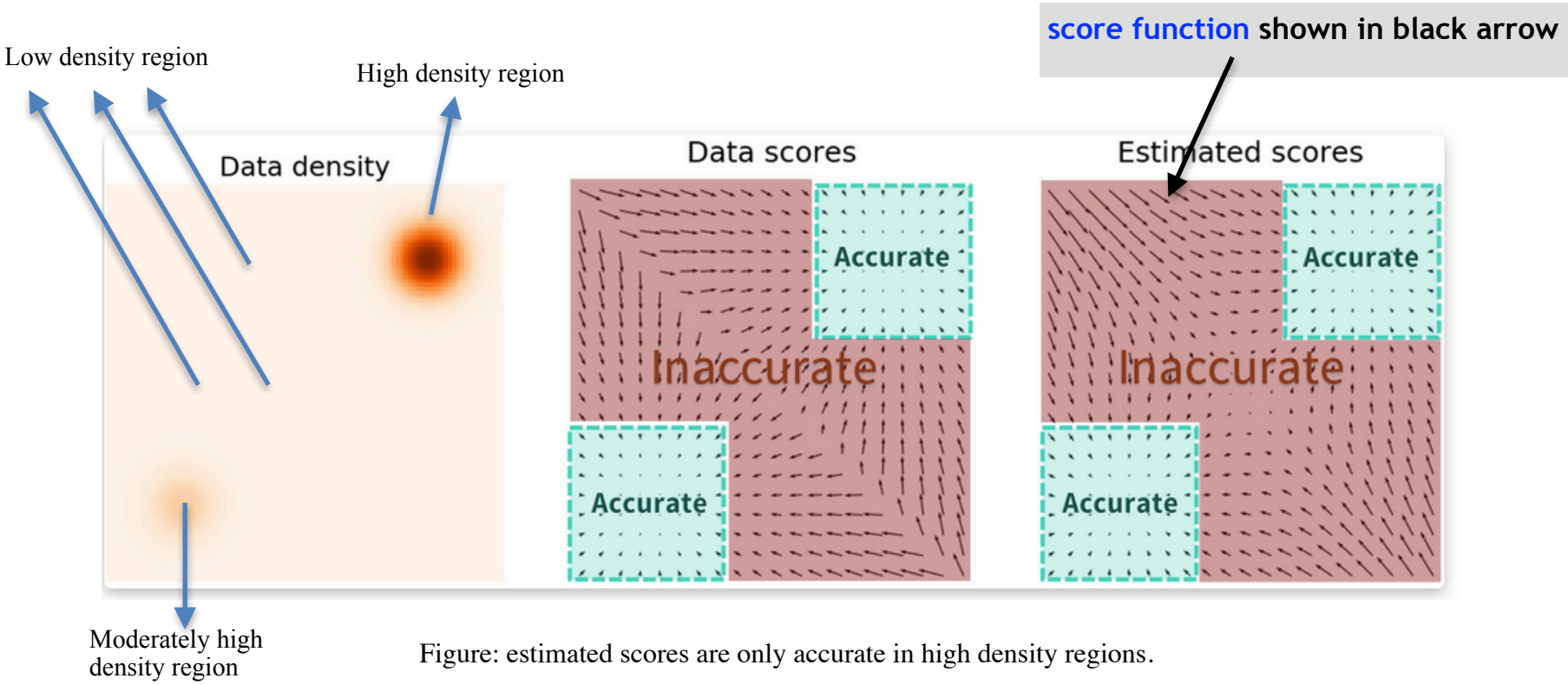


Figure: estimated scores are only accurate in high density regions.

- **Challenges:** the estimated score functions are inaccurate in low density regions

Generative Model: Score-Based Model

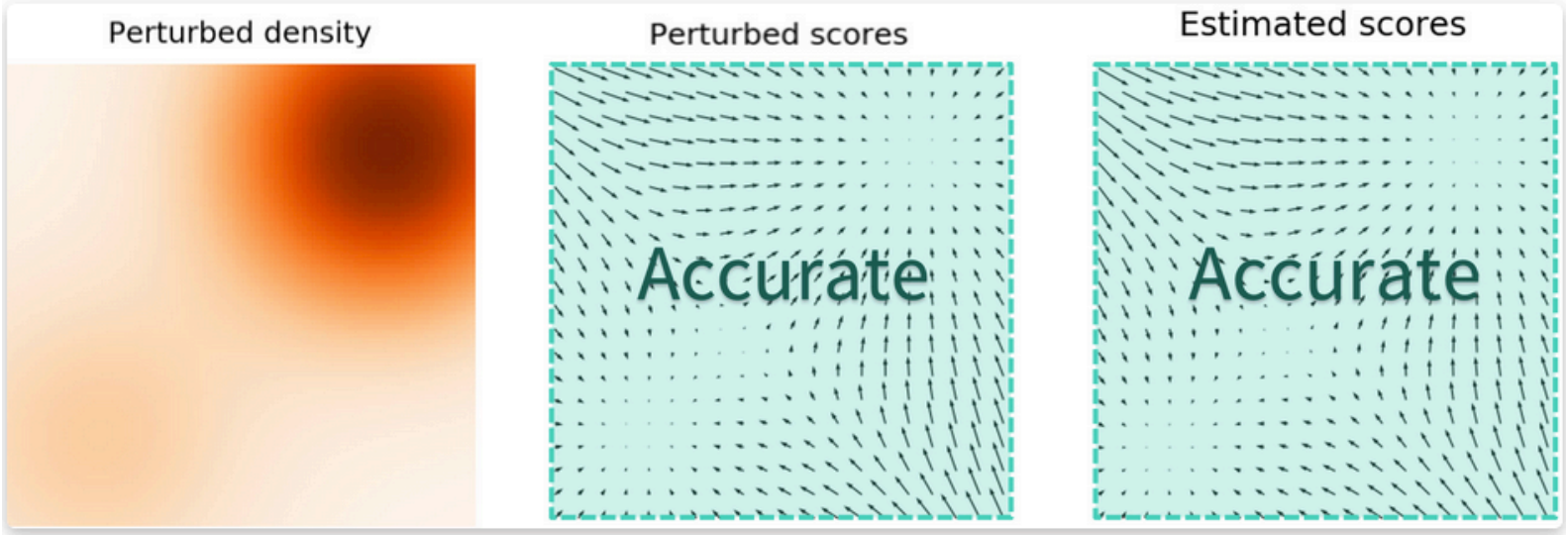


Figure: estimated scores are accurate everywhere for the noise-perturbed data distribution due to reduced low data density regions.

- **Solution:** bypass the difficulty of accurate score estimation in regions of low data density by **perturbing** data points with noise and train score-based models on the noisy data points instead.

Generative Model: Noise Conditional Score-Based Model

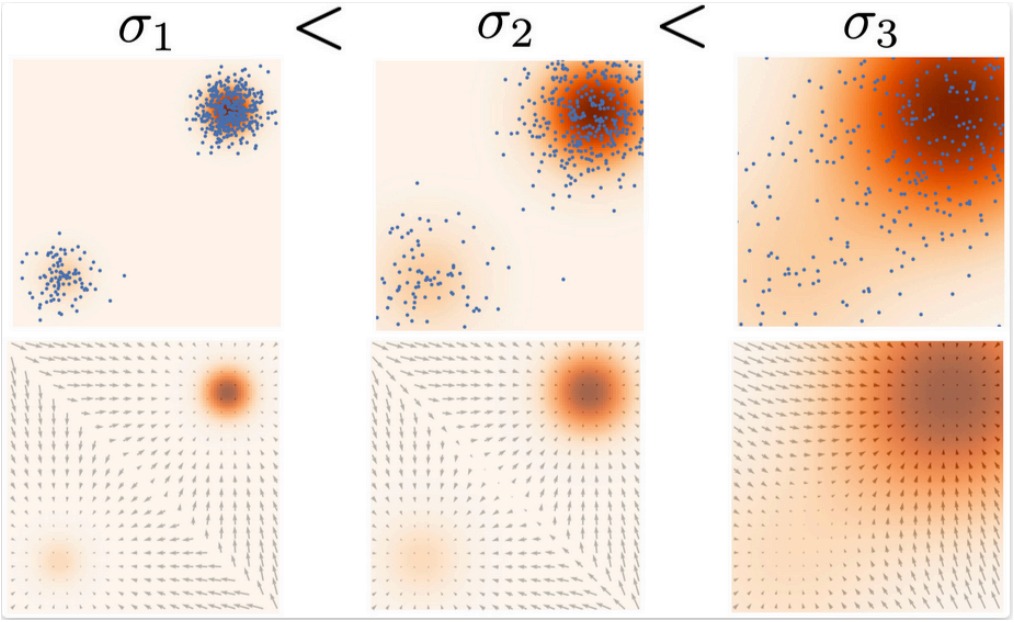
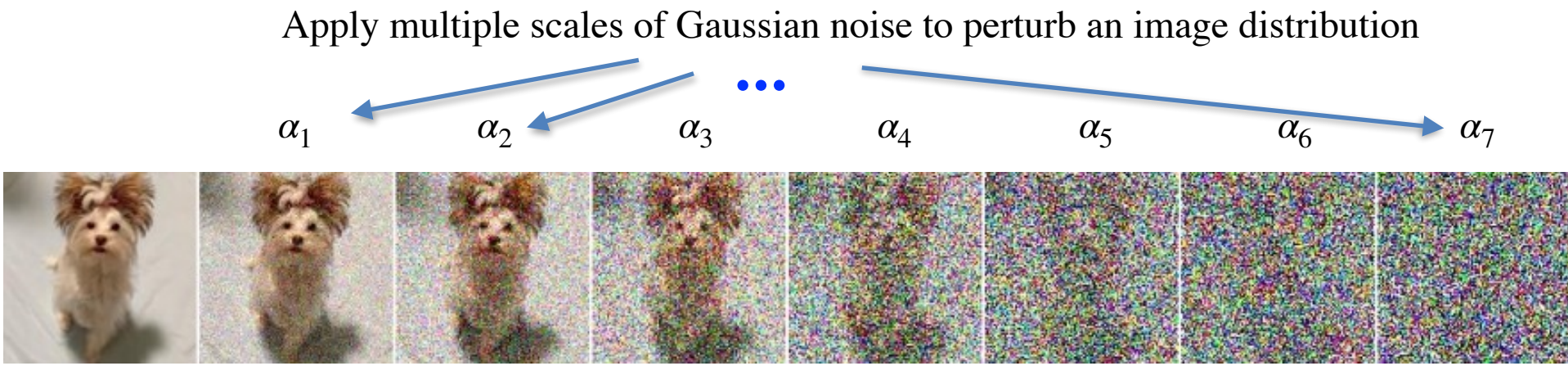


Figure: apply multiple scales of Gaussian noise to perturb the data distribution (**first row**) jointly estimate the score functions for all of them (**second row**)

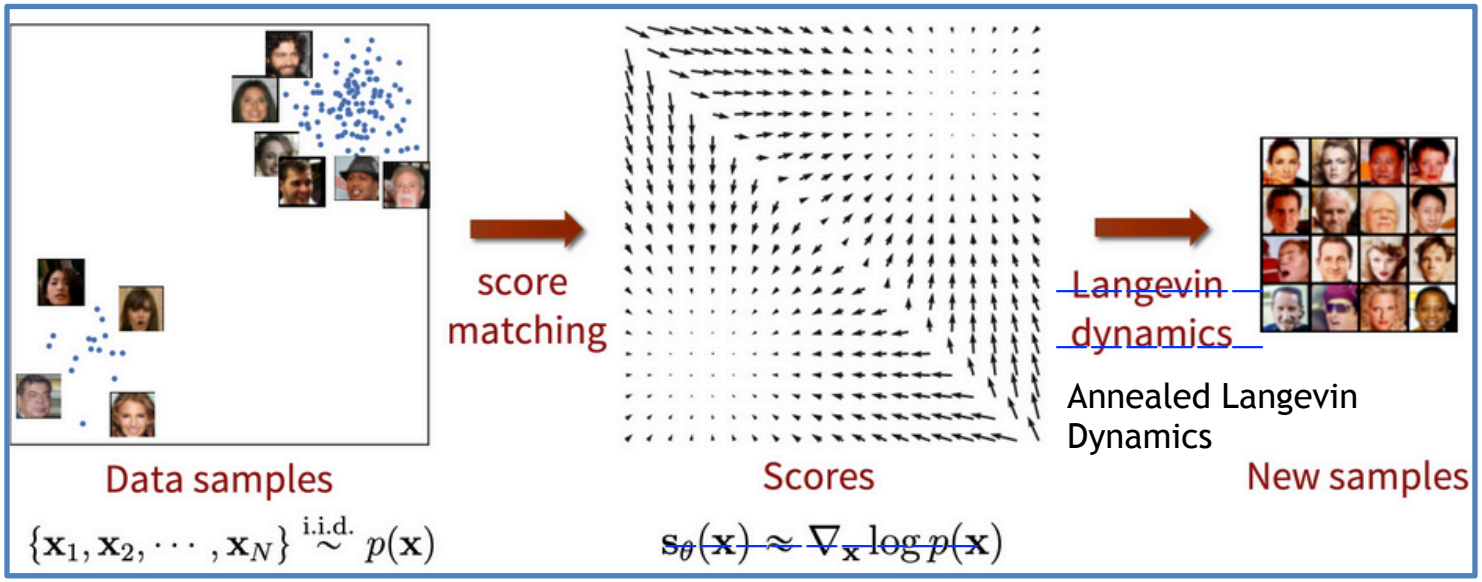
- **Solution:**
  - perturb the data distribution in a sequence of ***L=(100 or 1000)*** Gaussian noises (alpha values in the above figure and in that order)
  - Like before, train score-based models on this noisy data points.
    - It is called noise conditional score-based model

Generative Model: Noise Conditional Score-Based Model



- **Solution:**
  - For example, perturb an image in a sequence of ***L*** Gaussian noises (alpha values in the above figure and in that order)
  - Like before, train score-based models on this noisy data points.
    - It is called noise conditional score-based model

Generative Model: Score-Based Model



$$s_\theta(x, i) = \nabla_x log p_{\sigma_i}(x)$$

Figure: train a score-based model with score matching using noise conditional score network, and then produce samples via Annealed Langevin dynamics

UNet

Generative Model: New Sample Generation

Follow the reverse order (*L step process*) during inference: start with large Gaussian noise, modify the image according to the scores, and generate new sample

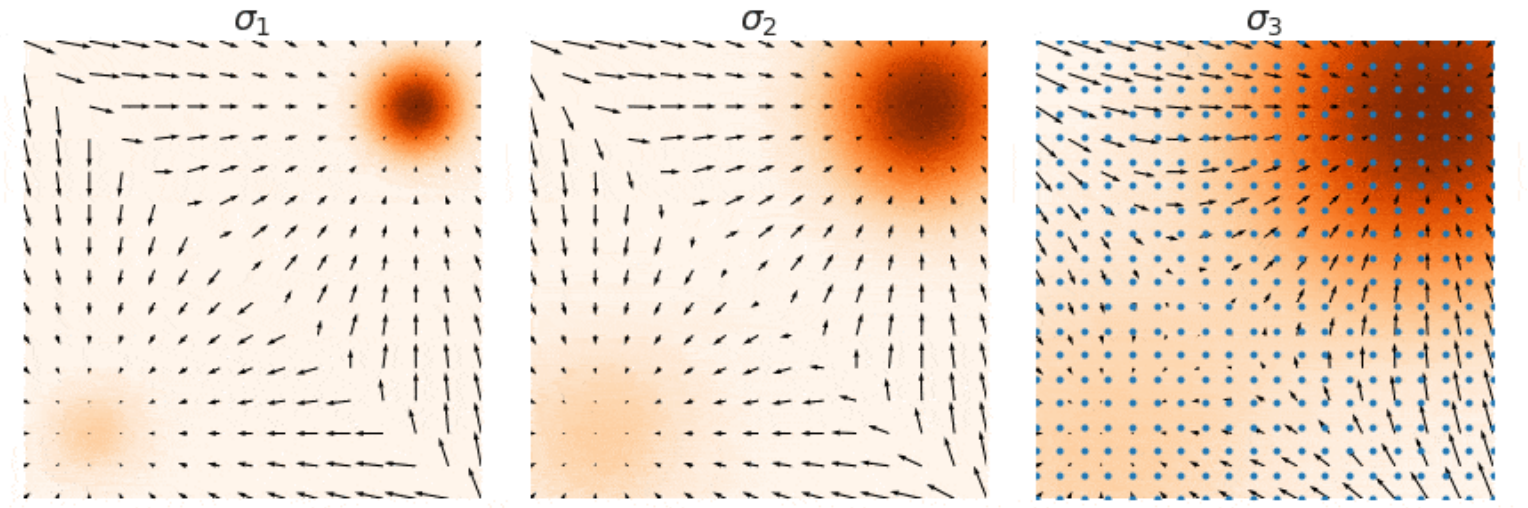


Figure: generate a new sample for a mixture of two Gaussian distribution. From right to left, generate new sample from largest noise conditioned distribution (third column), then second largest noisy condition distribution (middle), and finally the smallest noise conditioned distribution (left)

Generative Model: New Sample Generation

Follow the reverse order (*L step process*) during inference: start with large Gaussian noise, modify the image according to the scores, and generate new sample

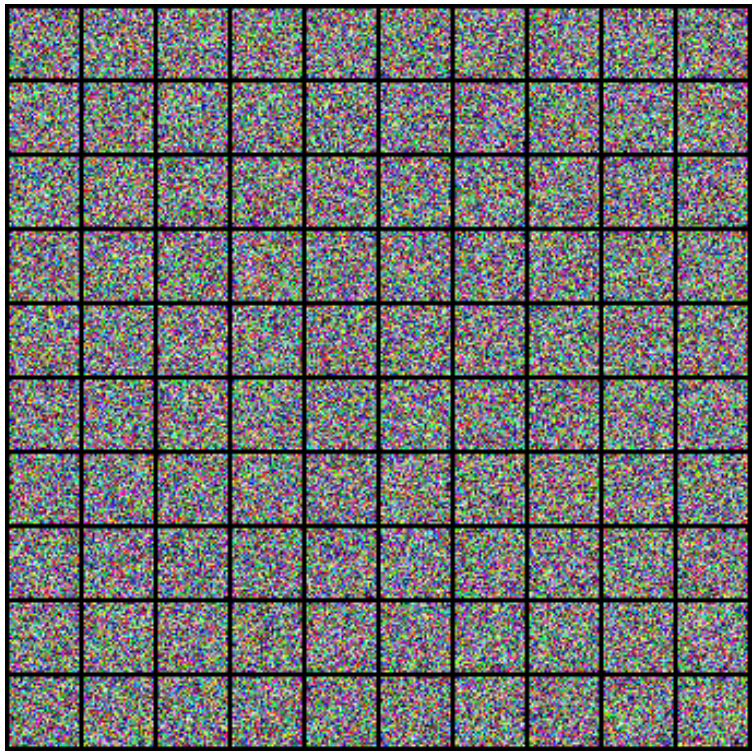


Figure: generate a new sample for a image data distribution for CIFAR-10 dataset

Diffusion Process

- Goal: learn structure of the probability density of a dataset

- Recover structure by reverse time; in other words, recover the data distribution by:
  - starting from uniform distribution and running backward dynamics

Diffusion Process: Algorithm

- **Forward process:** destroy all structure in data using diffusion process

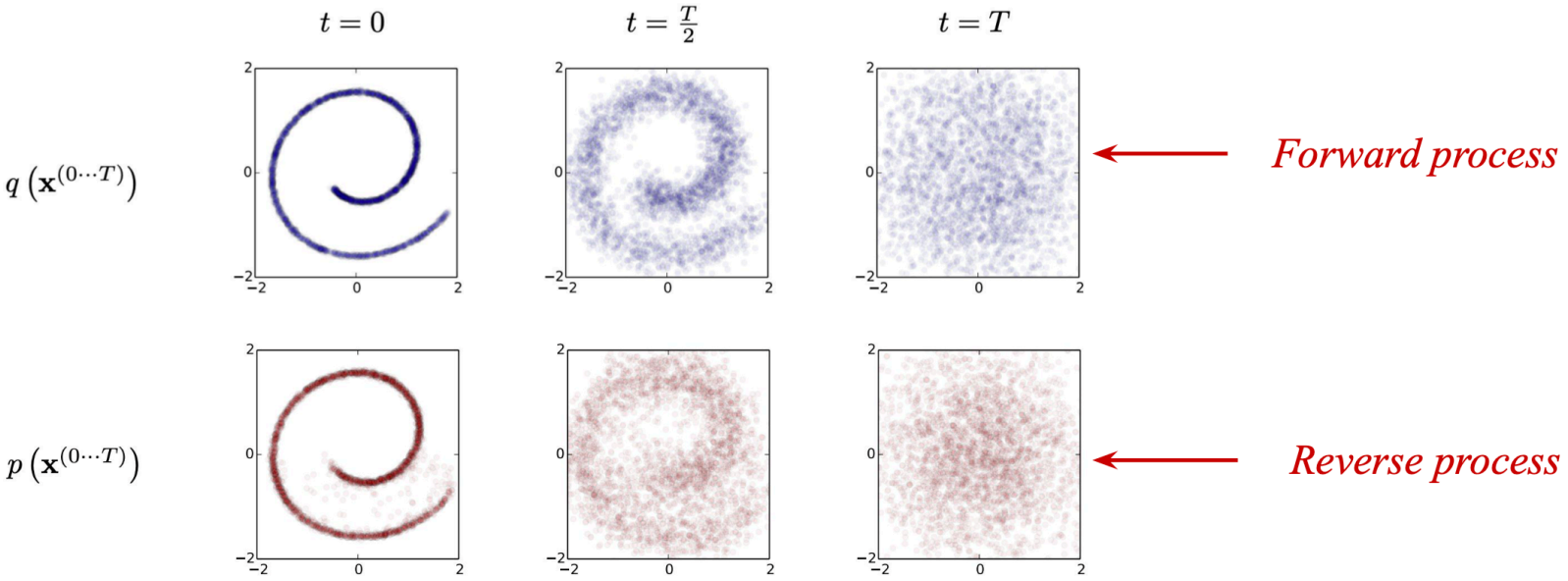
- **Reverse process:** train a DNN model to reverse the diffusion process

UNet

Diffusion Process: Algorithm

- **Forward process:** destroy all structure in data using diffusion process
- **Reverse process:** train a DNN model to reverse the diffusion process

UNet



slide inspired from J. Sohl-Dickstein

Diffusion Process: Algorithm

- **Forward process:** converts any complex data distribution into a tractable distribution

- **Reverse process:** learn a finite-time reversal of this diffusion process → generative model

PyTorch Implementation: Diffusion Model

- Open the notebook on blackboard