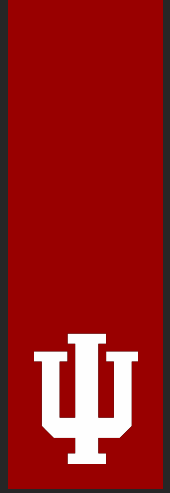


CSCI- B 657 - Discussion Section

Vibhas Vats

Drawback of VAEs

- Objective function of VAEs

Encoder: LV given input image x_i

N(0,1) **prior**

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

- If KL-div is large ⇒ no proper optimization ⇒ poor image generation

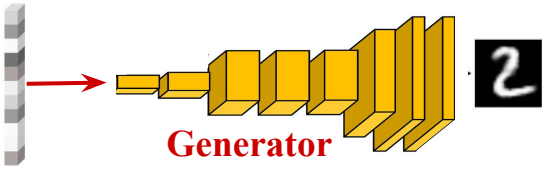
Why?

  - very strong prior distribution *p(z)* → prior distribution *different* than the data distribution
    - if the prior is **not** proper → poor optimization → poor reconstruction
  - another probabilistic explanation exits
    - involving evidence lower bound (ELBO) ✖

- GANs can produce better results

How GAN does it?

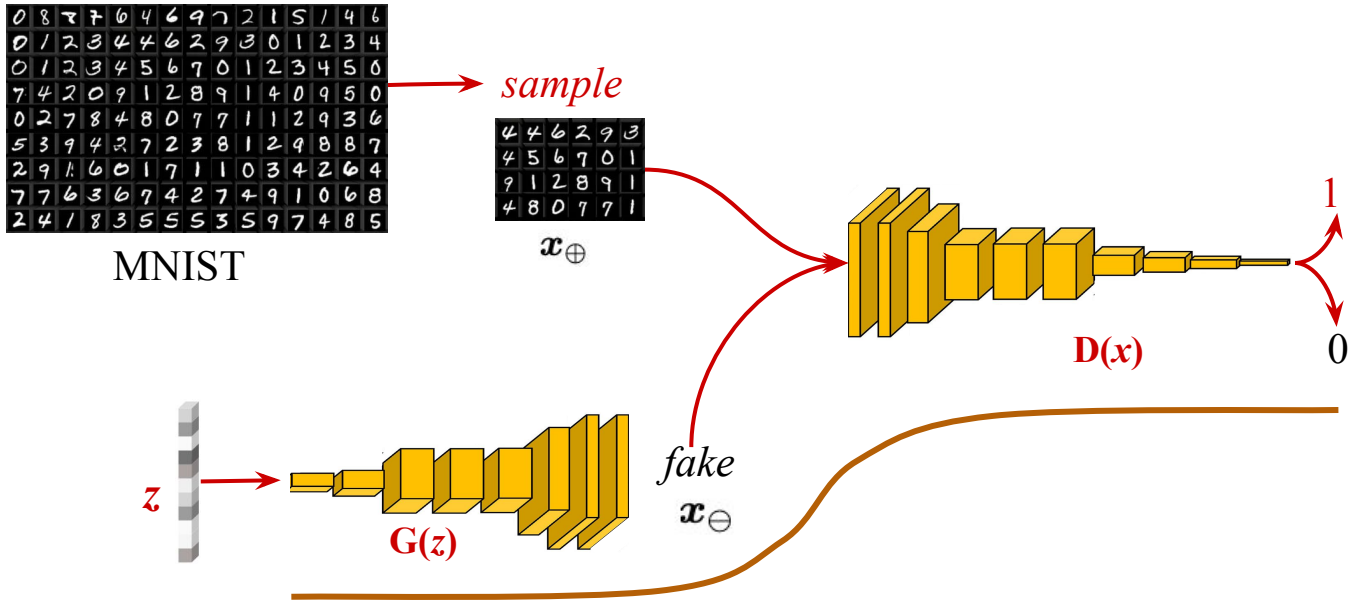
- **Generator (G):**
  - from an *N-dimensional code* → generate images
  - *main goal*: to produce images that looks real
    - real to whom? → **to human perception**
- But we can't have human in a training loop!
  - Another network → acts as *proxy* for human
  - **aim**: to discriminate b/w *real* and *fake* images
- **Discriminator (D):**
  - How to train such a network?
    - *input*:
      - a set of *real* images → from data distribution
      - and a set of *fake* images → output of G
    - ask to perform a *binary classification*:
      - real → 1, fake → 0
      - Sigmoid is used for binary classification



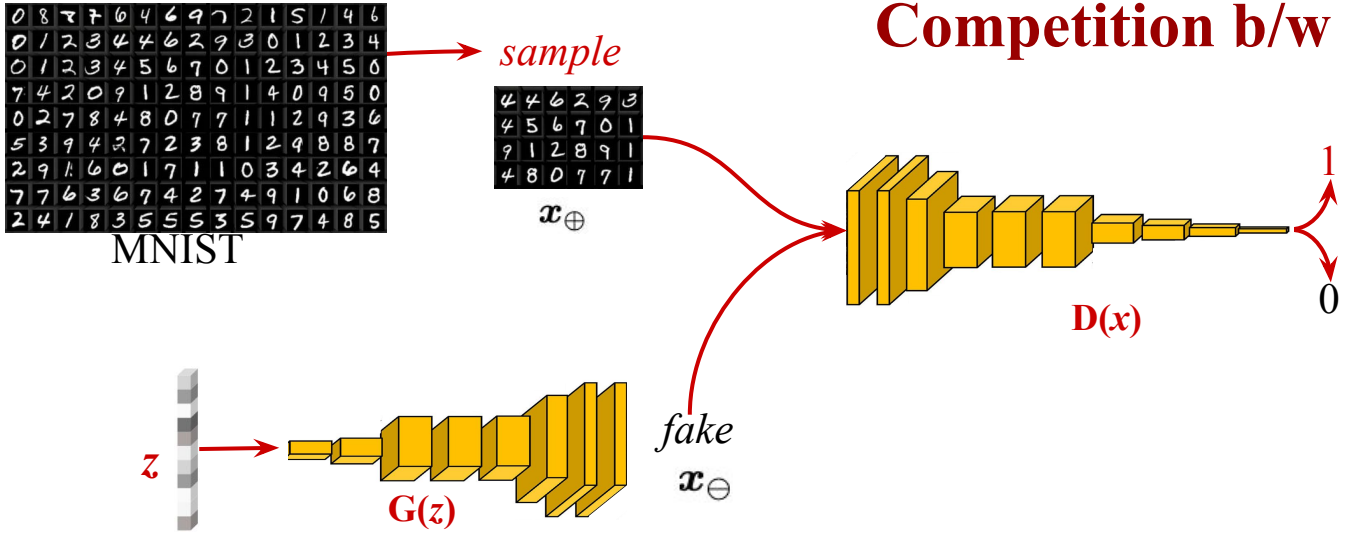
Competition b/w G and D

- **Goal of D:**→ to correctly classify real and fake images
- **Goal of G:**→ to fool the D
  - fool the D to classify a fake as real
- This has to be done simultaneously!

- If G can fool D
  - and we know that D is good enough
  - then, G understand the data distribution



Competition b/w G and D



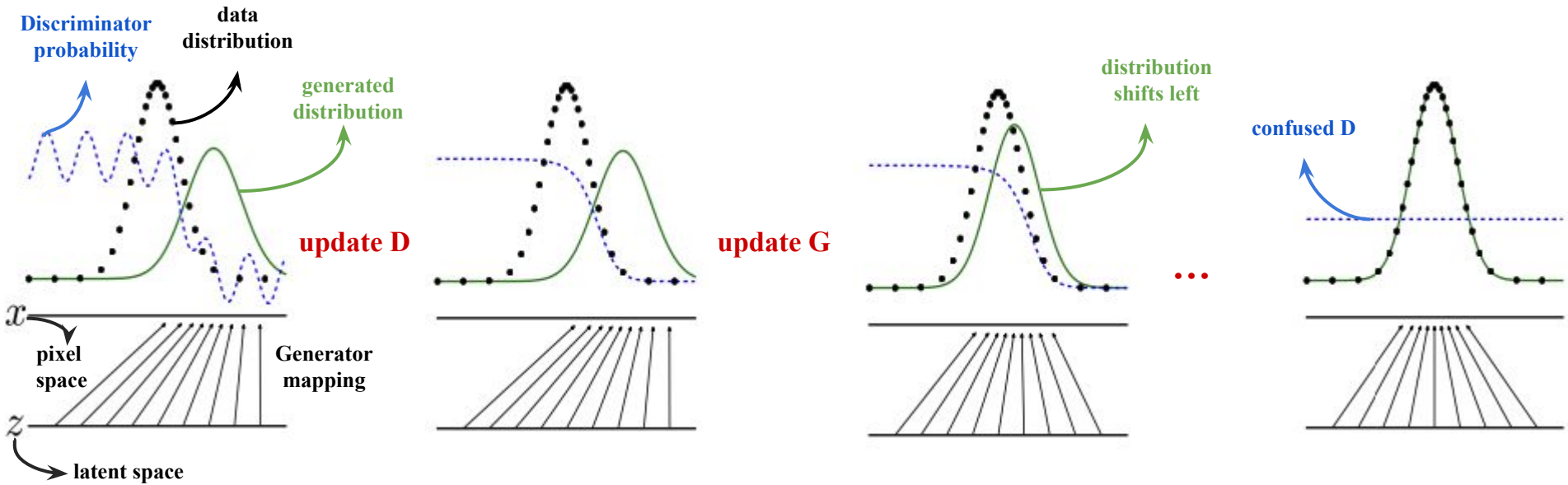
- If the generator does a good job
  - creates fake examples that looks like real
  - harms discriminator: **D(*x*–)** ↑
- **Dilemma**:
  - discriminator needs the generator to be good → to be able to learn from confusing fake examples
  - but, by definition a good generator promotes **D(*x*–)** ↑
    - harms the performance of the discriminator ⇒ ***2-player minimax game***

- If the discriminator does a good job
  - becomes good at discerning ***x***+ and ***x***–
  - **D(*x*+)** ↑ and,
  - harms generator: **D(*x*–)** ↓

GAN Optimization

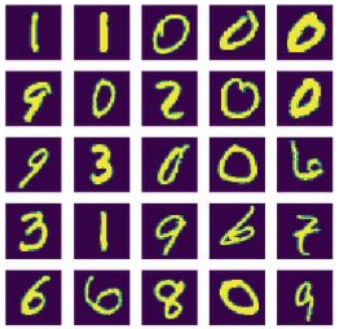
- Aim of GAN optimization is to achieve **Nash equilibrium**:
  - **D(x+) = D(x–) = 0.5**
  - discriminator can't tell real and fake apart!!
    - *meaning*: generator can generate images similar to real ones

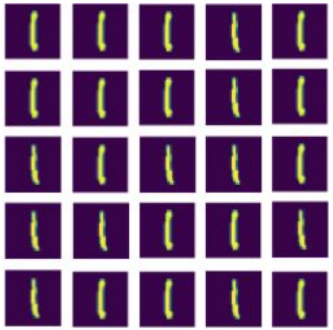
Nobel Prize 1994:
"for pioneering analysis of equilibria in the theory of non-cooperative games"

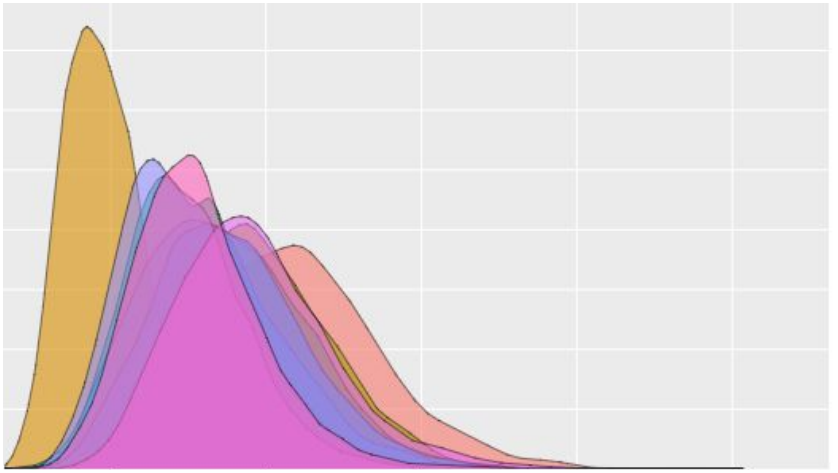


Mode Collapse

- Which set of generator images do you prefer?

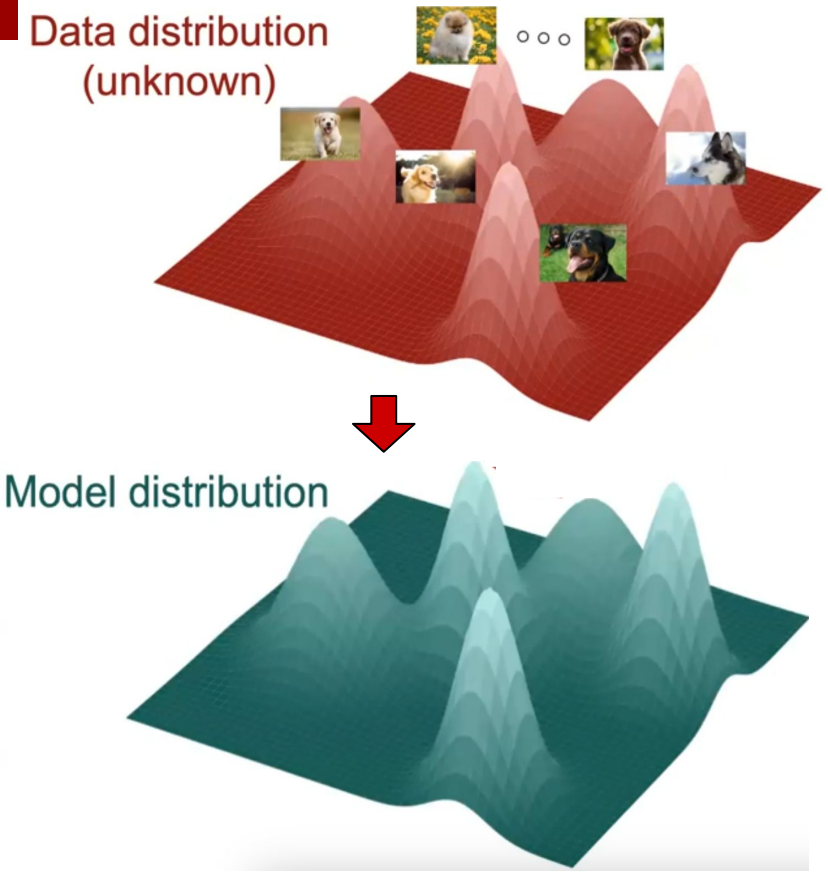


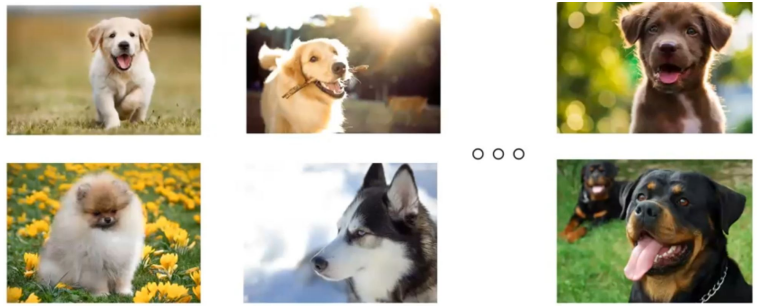




- A too much optimized generator can get stuck at a model of the generative model
  - it will generate real-looking fake images but mostly single class
  - initially D will classify it ‘real’
  - but then it realizes and catches it up, but the G can jump to another **mode** …
- GAN generates sharp realistic images
  - but these images do not represent the *true data distribution*
- Very careful optimization is needed to handle this

Estimating Probability Distribution of Data





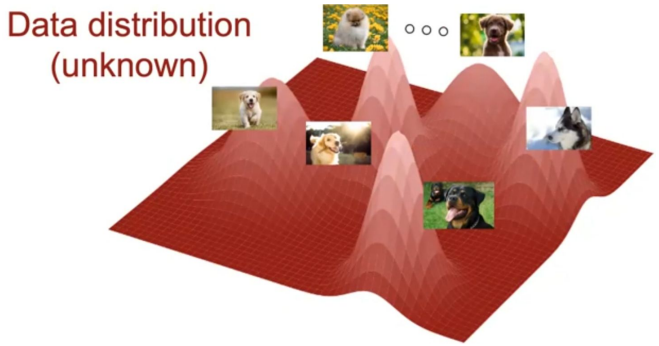
Dataset

Generate

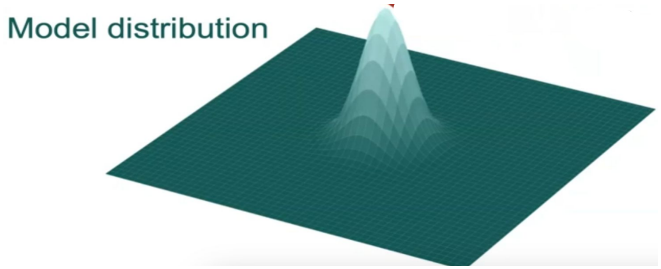
Novel data points

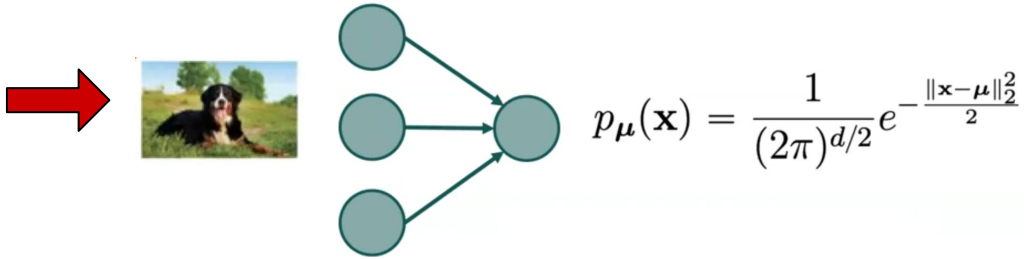
slide inspired from Yang Song

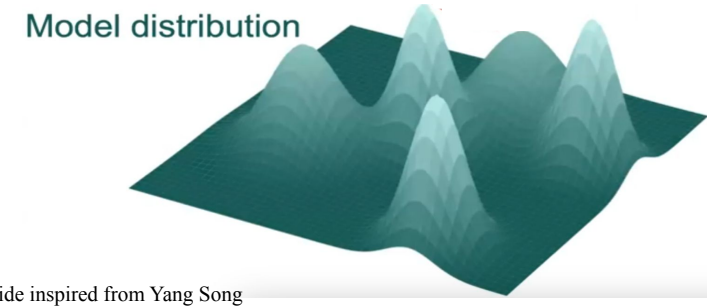
Complex Generative Models

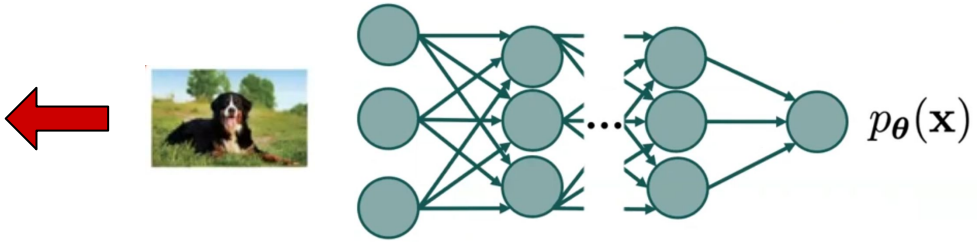


- Data Distribution is extremely complex for high dimensional data.
- How to build a complex model to approximate the data distribution?
  - start small !









slide inspired from Yang Song

Generative Model Categorization

- Based on representation of probability distribution:
  - existing generative modeling techniques → two categories
    - *Likelihood-based models*:
      - directly learns the distribution's prob. density fn via max. likelihood → VAEs
      - limitations:
        - require strong restriction on the model architecture for tractable normalization
        - rely on surrogate objectives to approximate maximum likelihood training
    - *Implicit generative models*:
      - prob. dist. is represented by a model of its sampling process → GANs
      - limitations:
        - require adversarial training → unstable and lead to mode collapse

PDF Estimation

- Data: $\{x_1, x_2, ..., x_n\}$
- Underlying data distribution: $P(x)$
- $f_\theta(x) \in R$: a real valued function parameterized by θ

Unnormalized Prob. model

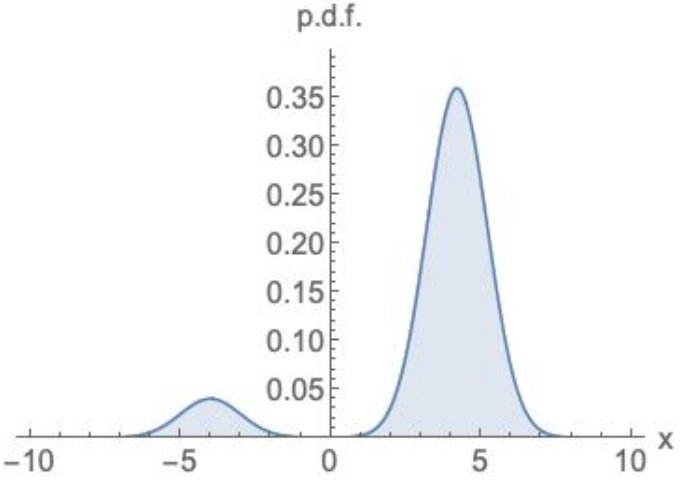
Probability Density Function (PDF): $P_\theta(x) = \frac{e^{f_\theta(\mathbf{x})}}{Z_\theta}$

Neural Network

Normalizing constant

- Z is dependent on θ
- PDF should integrate to 1

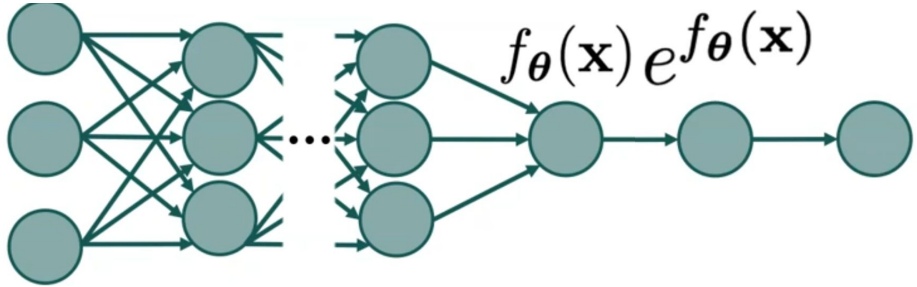
- Parameterized PDF:
  - no matter how you change the model family and parameters, it has to be **normalized**.



PDF Estimation with DL Models

- DL model can approximate function in any range (-∞ to +∞)
- Probability density function is positive → require some modifications

$\mathbf{x}$



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

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

Normalizing Constant

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

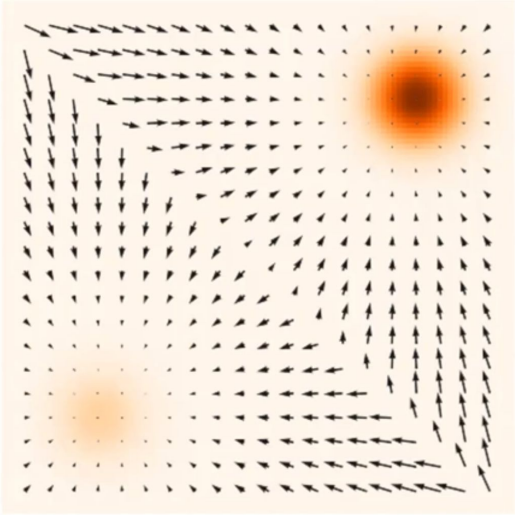
- Tackling intractable normalizing constant:
  - *Approximate normalizing constant*:
    - VAEs: leads to inaccurate probability evaluation
  - *Using restricted neural network models*:
    - highly restricted model family → can't capture complex data distribution
  - *Generative adversarial networks*:
    - can not evaluate probabilities
- Alternative method → score-based models

Intractable for most DL models 😞

slide inspired from Yang Song

Score-Based Models

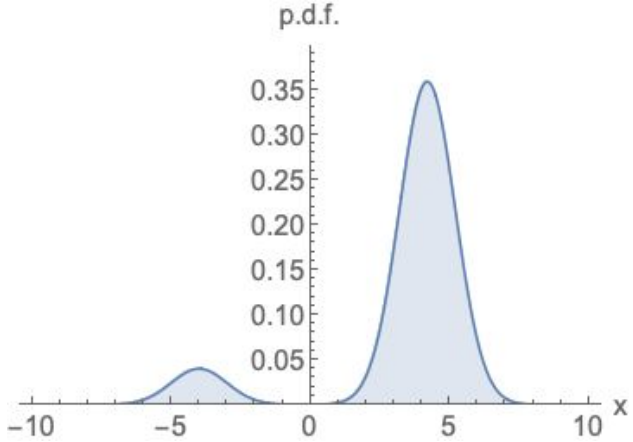
- **Proposed method: Score matching**
  - *model gradient of the log probability density function* → score function
  - do not require tractable normalizing constant
  - learning by score matching
  - Score function of a distribution ***P(x)*** = $\nabla_{\mathbf{x}} \log p(\mathbf{x})$
  - Score-based model: $\boldsymbol{S_\theta(x)} \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$
    - gradient w.r.t $\boldsymbol{x}$, not θ
    - parameterized without normalizing constant
- **Score vs. Density function:**
  - density: two Gaussian distributions – color coded
  - score: vector field of gradients
    - points in the direction where density function has fastest growth
  - with derivative and integration: move from one to other



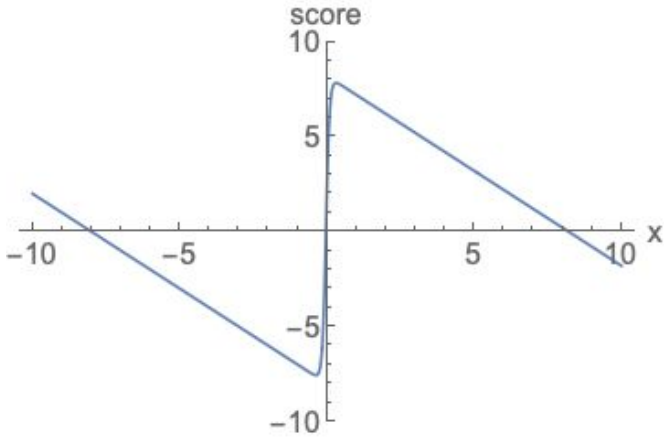
Score vs. density function

slide inspired from Yang Song

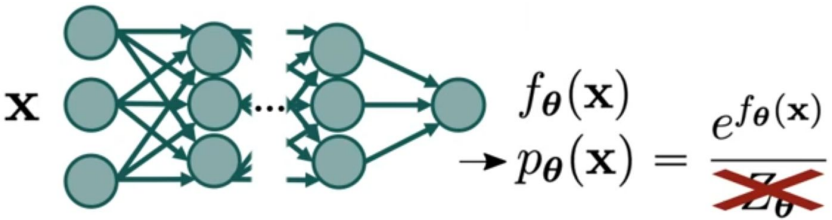
Score-Based Models



Probability Density Function



Score Function



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

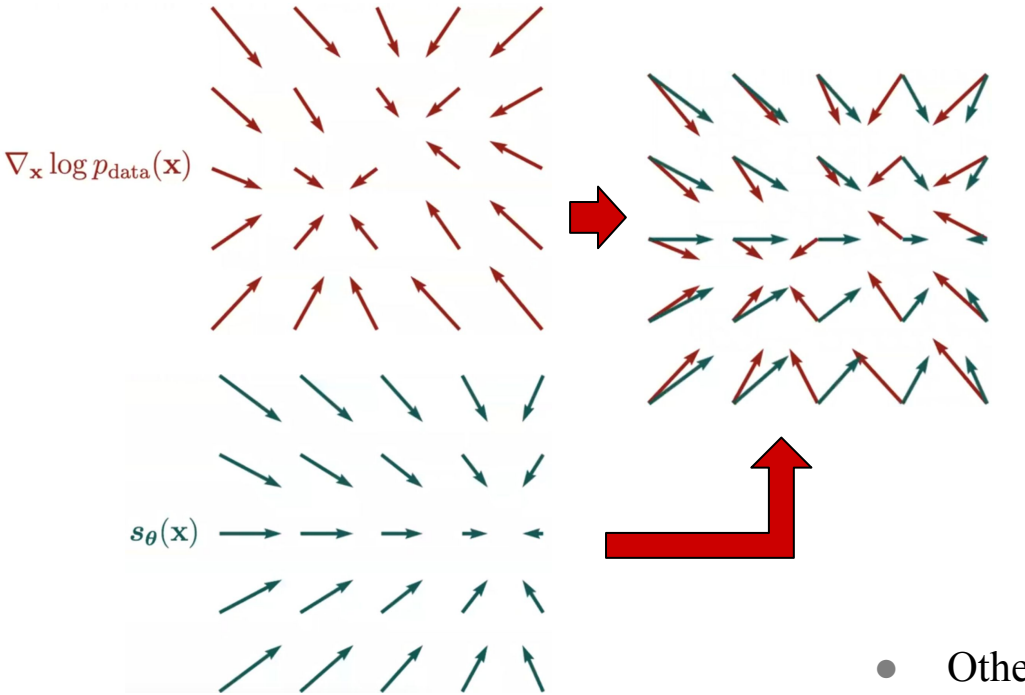
$\nabla_{\mathbf{x}} \log Z_\theta = 0$

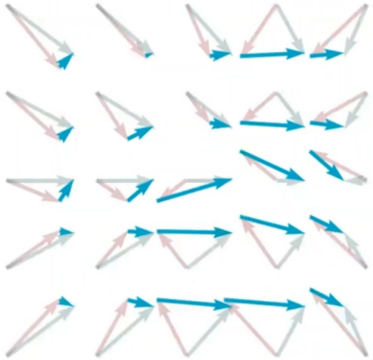
slide inspired from Yang Song

Training Score-Based Models

- **Given:** $\{x_1, x_2, \ldots, x_n\} \overset{i.i.d}{\sim} P_{data}(x)$
- **Goal:** $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$

- **Score Model:** $S_\theta(x) : R^d \rightarrow R^d \approx \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$
- **Objective:** compare two vector fields. How?





Avg → Objective

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

(Fisher divergence)

- Other methods to deal with that
  - Sliced score matching (Song et al. 2019)

slide inspired from Yang Song

Score Matching

- Family of method that minimizes Fisher Divergence (FD):
  - without knowledge of the ground truth data score
  - optimized with stochastic gradient descent
  - do not require adversarial optimization
- Modeling flexibility of FD:
  - $S_\theta(x)$ doesn't have to be an actual score fn. of any normalized distribution
  - compares L2 distance b/w ground-truth data score and estimated score
  - no restrictions on the form or complexity of $S_\theta(x)$
    - it should only be a vector valued function
- Train score-based model with score matching, but how to generate new samples?

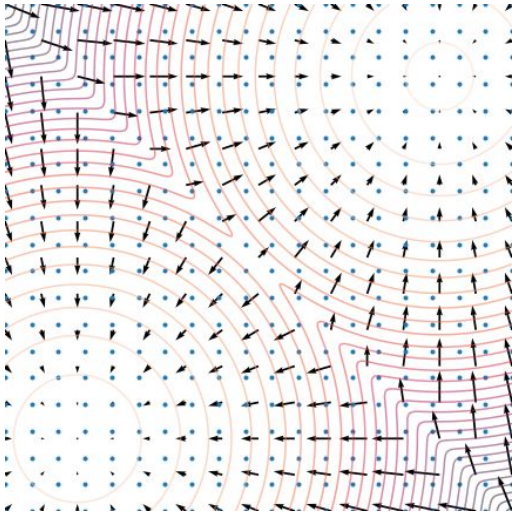
Sampling – Langevin Dynamics

- An iterative process to draw samples from a trained score-based model $\boldsymbol{S_\theta(x)}$
- It provides an MCMC procedure to samples from a distribution $P(x)$ using only score function.
- It initializes the chain from an arbitrary prior distribution $x_0 \sim \boldsymbol{\pi}(x)$
  - then iteratively samples

Noise ~ $N(0,1)$

$$X_{i+1} \leftarrow X_i + \epsilon \quad \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \sqrt{2}\,\epsilon\, z_i, \qquad i = 0, 1, \ldots, K$$

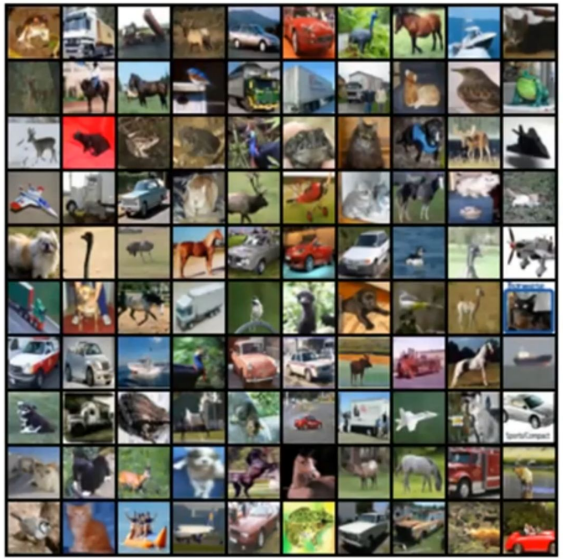
- When $\epsilon \to 0$ and $K \to \infty$:
  - X_K obtained using above equation converges to a samples from $P(x)$
- In practice:
  - error is negligible when ϵ is sufficiently small and K is sufficiently large



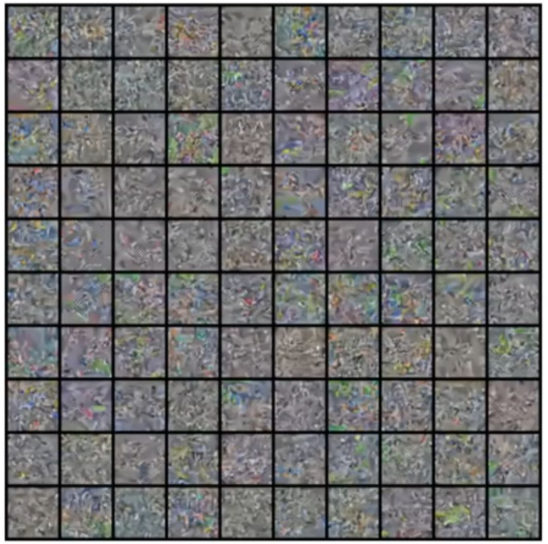
slide inspired from Yang Song

Naive Score-Based Generative Models

CIFAR-10 data



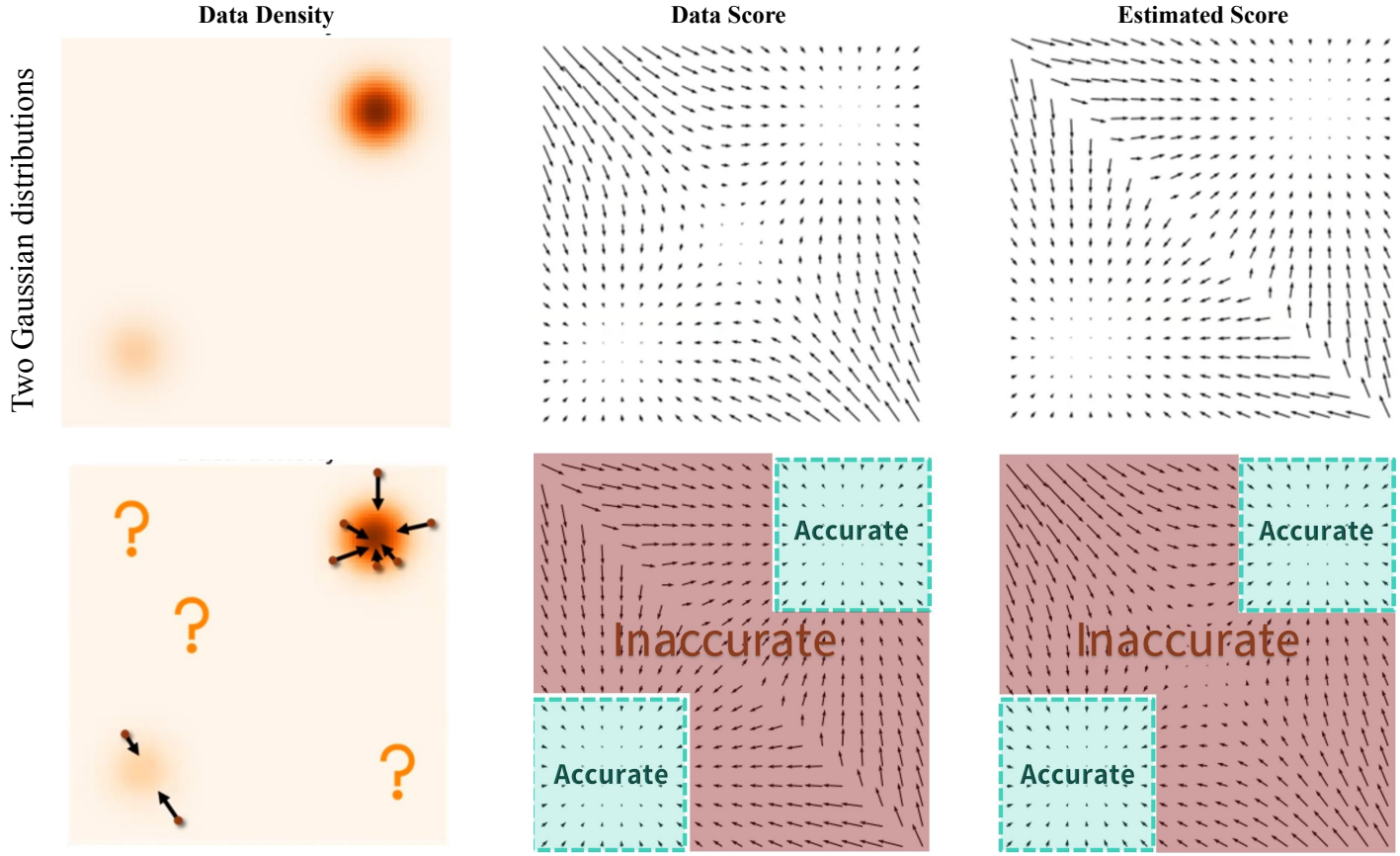
Model samples



- Score matching + Langevin dynamics ⇒ doesn't work as expected !!
- **Reason**:
  - poor score estimation at low data density regions $\nabla_{\mathbf{x}} \log p(\mathbf{x})$

slide inspired from Yang Song

Naive Score-Based Generative Models

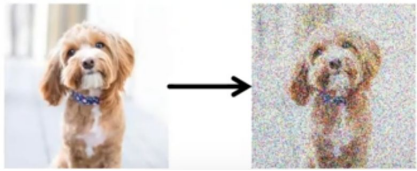


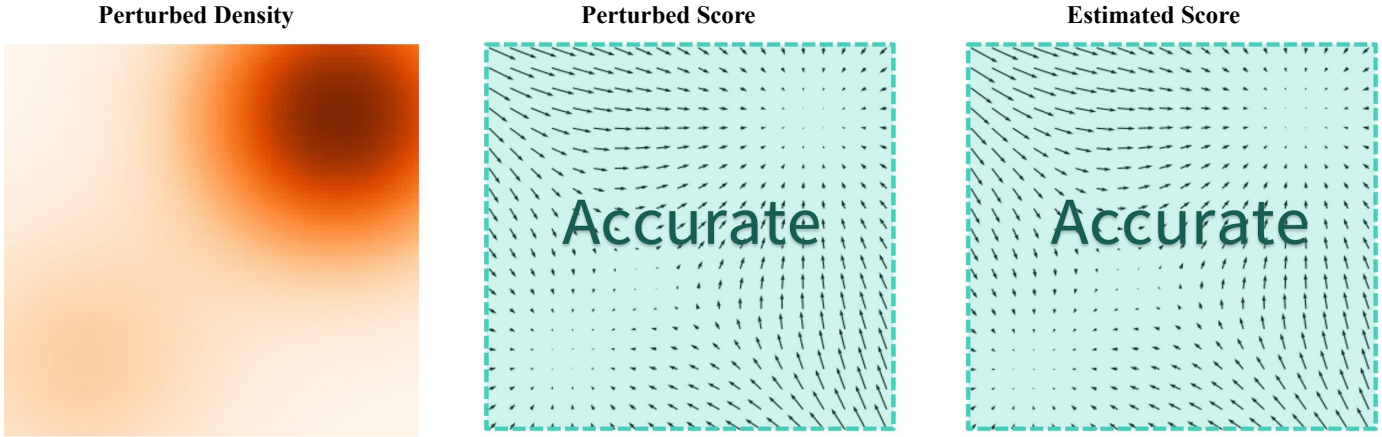
- Absence of sample in low data density regions makes the estimate inaccurate

slide inspired from Yang Song

Improving Score Estimation

- Perturb data points with noise and train score-based models on the noisy data points
- When noise magnitude is sufficiently large → it can *populate low data density regions*
  - improve the accuracy of of estimated score in that region
  - eg. mixture of Gaussians perturbed by additional Gaussian noise





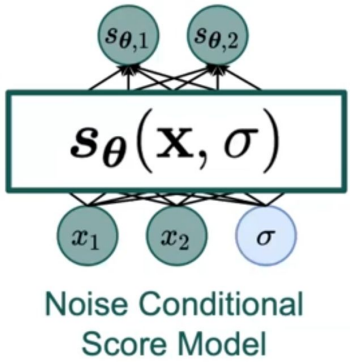
- Sequential application of noise

- How to choose an appropriate noise scale?

slide inspired from Yang Song

Choosing Noise Scale

- **Large noise:** cover more low density regions for better score estimation
  - but, it over corrupts the data → changing significantly from the original distribution
- **Low noise:** causes less corruption of the original data distribution
  - but, doesn't not cover the low density regions well !!
- Best of both world ⇒ *Multiple scales of noise perturbations simultaneously*
  - perturbing with Gaussian noise with L increasing standard deviations: $\boldsymbol{\sigma}_1 < \boldsymbol{\sigma}_2 < \ldots < \boldsymbol{\sigma}_L$
  - *First*: perturb P*(x)* with each of the Gaussian noise $N(0, \boldsymbol{\sigma}_i^2 I)$, $i = 1, 2, \ldots, L$
  - *Next*: estimate the score function of the each noisy-perturbed x by training
    - noise conditional score-based model $\boldsymbol{S_\theta(x, i)}$ with score-matching, simultaneously



Positive weighting function

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

slide inspired from Yang Song

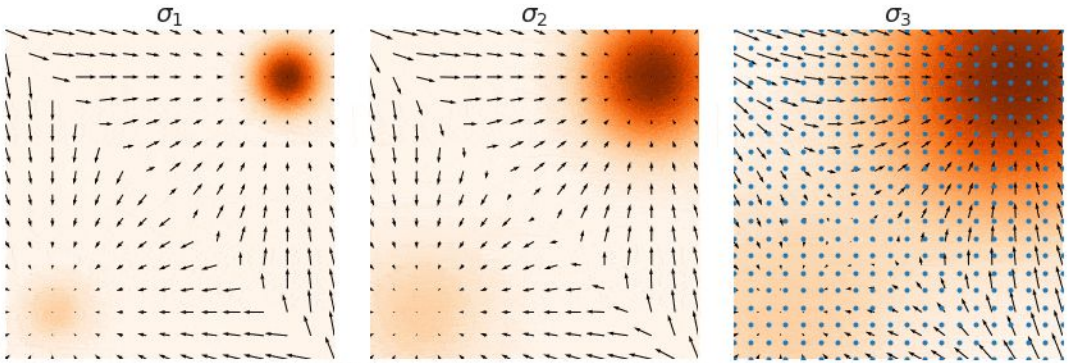
Sampling – Annealed Langevin Dynamics

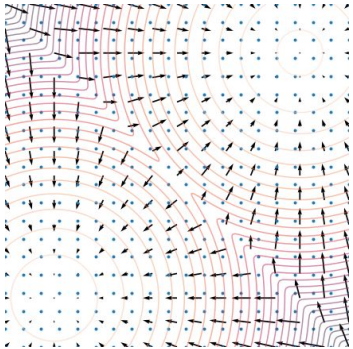
- **Annealed Langevin dynamics:**
  - combine a sequence of Langevin chains with gradually decreasing noise scales.
  - Start with largest noise sample → slowly move towards smaller noise samples
  - starting with largest noise model provides accurate estimate from the beginning
  - slowly moving to smaller noise samples keeps this accuracy consistent

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$.

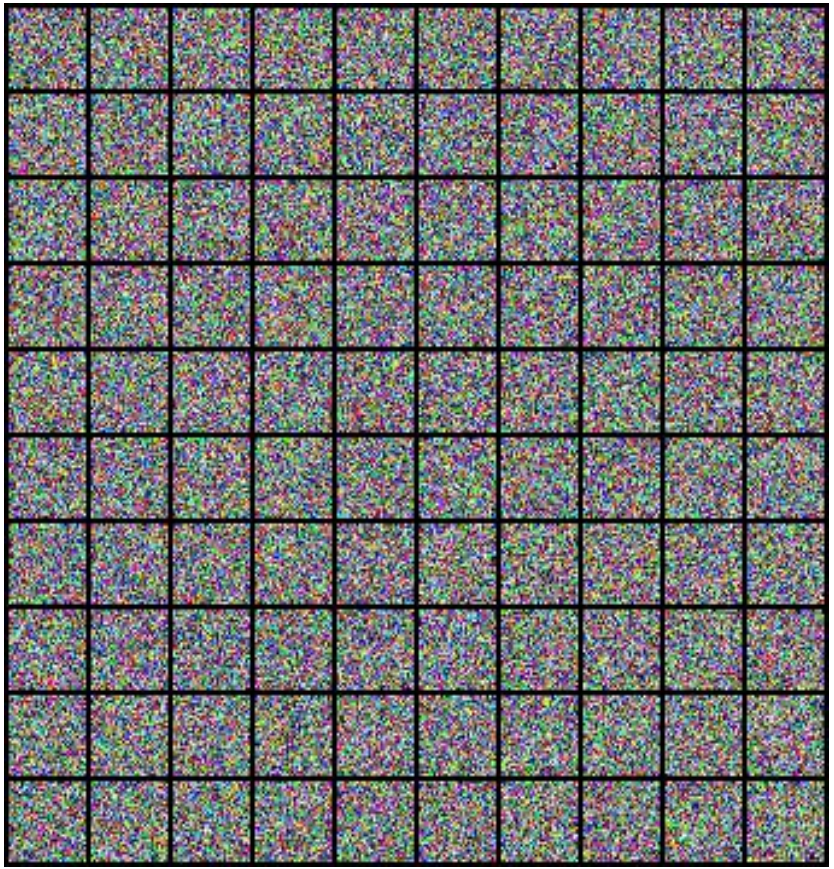
Initialize $\tilde{\mathbf{x}}_0$
for $i \leftarrow 1$ to L **do**
  $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2/\sigma_L^2$ ▷ α_i is the step size.
  for $t \leftarrow 1$ to T **do**
    Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
    $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\,\mathbf{z}_t$
  end for
  $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
end for
return $\tilde{\mathbf{x}}_T$

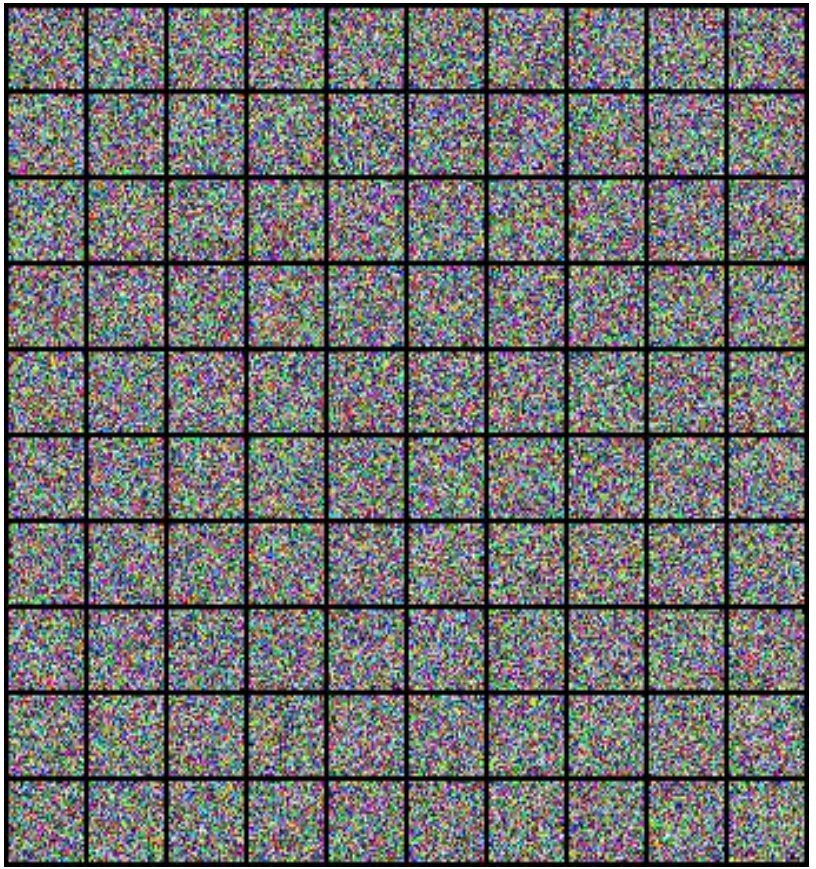




slide inspired from Yang Song

Samples Outputs





Diffusion Process - Intro

- **Goal**: learn structure of the probability density of a dataset
- Dye density represents probability density
  - each dye molecule is a sample
  - density of dye molecules represents probability density of data in that region
- **Observation**:
  - Diffusion process destroy structure in dataset

slide inspired from J. Sohl-Dickstein

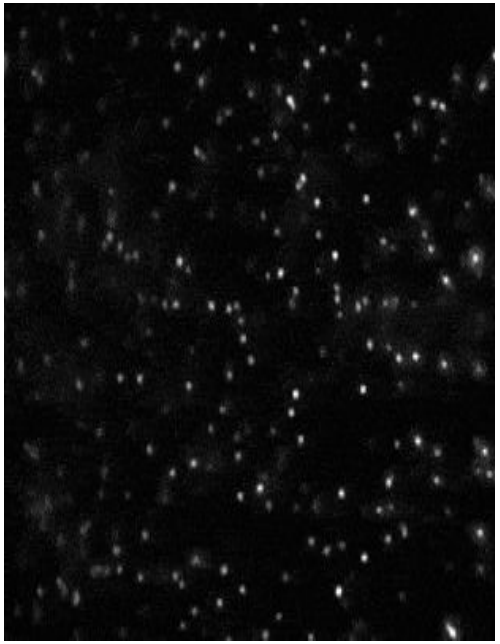
Diffusion Process - Intro

- **Core Idea**: Recover structure by reverse time
- Recover data distribution by
  - starting from uniform distribution and running backward dynamics
- Using Markov Chain, learn the Markov process for the reverse process exactly the way it was destroyed
- How exactly?
  - lets see diffusion more closely

slide inspired from J. Sohl-Dickstein

Microscopic View of Diffusion

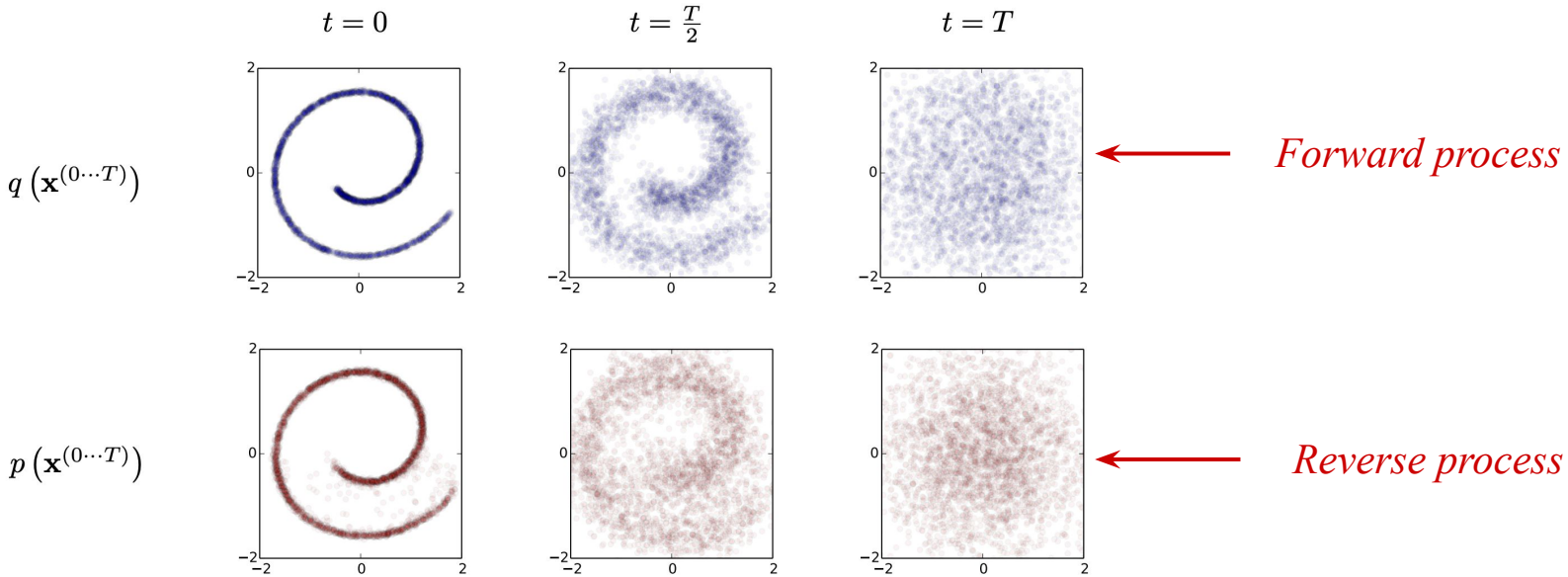
- Brownian motion of dye molecules:
  - seems similar for forward and backward in time
- **Observation**:
  - Microscopic diffusion is time reversible
- Can we predict the next location of a molecule?
  - **Assume**: position updates are small Gaussian
  - Take a data point:
    - draw a next location from a small Gaussian centered around its current location
- For every small diffusion steps, the functional form of the reverse process is identical
- Reverse process also consists of drawing a Gaussian with same mean and small covariance close to the original data point



slide inspired from J. Sohl-Dickstein

Algorithm

- **Steps**:
  - *Forward process*: Destroy all structure in data using diffusion process
    - symmetrically and slowly
  - *Reverse process*: Train Deep Learning model to reverse the diffusion process
  - Reverse diffusion process constitutes the generative model of data



slide inspired from J. Sohl-Dickstein

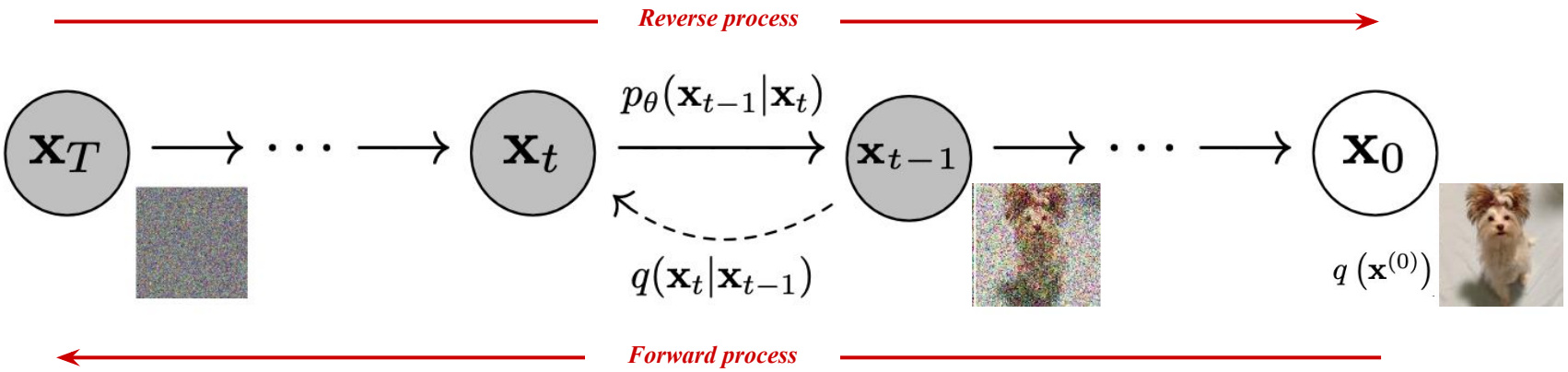
Tractability Vs. Flexibility

- *Tractable models* are easy to fit to data and can be analytically evaluated
  - but are unable to aptly describe structure in rich datasets
- *Flexible models* can arbitrarily fit any data
  - but are hard to evaluate
- Define a *flexible model* $\phi(x)$, yielding the flexible distribution $P(x) = \phi(x)/Z$
  - Z is the normalization constant → intractable to estimate
  - require specific process to estimate
- **Diffusion Probabilistic Models:**
  - Markov Chain gradually converts one distribution into another
  - extreme flexibility in model structure
  - exact sampling
- A generative Markov chain is learned to convert:
  - a *simple known distribution* → into a *target distribution (data distribution)*
  - the probabilistic model is defined as the endpoint of the Markov chain

Algorithm

- A forward process converts any complex data distribution into a tractable distribution

- Learn a finite-time reversal of this diffusion process → generative model



Mathematical Formulation

Markov Diffusion Kernel

Diffusion rate

$$\pi(\mathbf{y}) = \int d\mathbf{y}' T_\pi(\mathbf{y}|\mathbf{y}';\beta)\pi(\mathbf{y}')$$

Tractable distribution

$$q\left(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}\right) = T_\pi\left(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)};\beta_t\right)$$

T steps of diffusion

$$q\left(\mathbf{x}^{(0\cdots T)}\right) = q\left(\mathbf{x}^{(0)}\right)\prod_{t=1}^{T} q\left(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}\right)$$

Input data distribution

Forward Process

$$p\left(\mathbf{x}^{(T)}\right) = \pi\left(\mathbf{x}^{(T)}\right)$$

Tractable distribution

Finite time reversal

$$p\left(\mathbf{x}^{(0\cdots T)}\right) = p\left(\mathbf{x}^{(T)}\right)\prod_{t=1}^{T} p\left(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}\right)$$

Reverse Process

- Training amounts to maximizing the model log likelihood

Thank you!

INDIANA UNIVERSITY BLOOMINGTON