CS 760: Machine Learning
Generative Models

Misha Khodak

University of Wisconsin-Madison
29 October 2025

Outline

- **Intro to Generative Models**
  - Histograms, Parametrizing Distributions
- **Flow-based Models**
  - Transformations, training, sampling
- **Generative Adversarial Networks (GANs)**
  - Generators, discriminators, training, examples
- **Diffusion Models**

Outline

- **Intro to Generative Models**
  - Histograms, Parametrizing Distributions
- **Flow-based Models**
  - Transformations, training, sampling
- **Generative Adversarial Networks (GANs)**
  - Generators, discriminators, training, examples
- **Diffusion Models**

Generative Models

- Goal: learn an underlying process for (unlabeled) data.

- Recall generative vs discriminative modeling from Naive Bayes lecture.

Applications: Generate Images

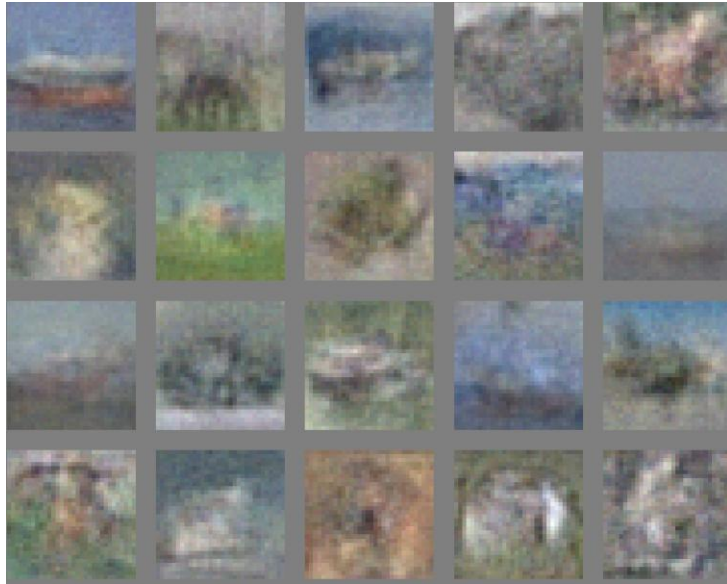- Old idea---tremendous growth
- Historical evolution:

2006: Hinton et al   2013: Kingma & Welling

Applications: Generate Images

- More recently, GAN models: 2014
  - Goodfellow et al

Applications: Generate Images

- More recently, GAN models
  - StyleGAN, Karras, Laine, Aila, 2018

Applications: Generate Images

- Today, Diffusion Models: DALL·E 3, Stable Diffusion 3.5

Prompt: ~*~aesthetic~*~ #boho #fashion, full-body 30-something woman laying on microfloral grass, candid pose, overlay reads
Stable Diffusion 3.5, cheerful cursive typography font.

Applications: Generate Images/Video

- GANs can also generate video
  - Plus transfer:

CycleGAN: Zhu, Park, Isola & Efros, 2017

Applications: Generate Video

- GANs can also generate video (DVD-GAN, Clark et al)

Applications: Generate Video

- Diffusion model can generate long videos up to a minute long with high visual quality.

[OpenAI Sora]

Additional Applications

- **Compress** data
  - Can often do better than fixed methods like JPEG
  - Similar to nonlinear dimensionality reduction

- Obtain **good representations**
  - Then can fine-tune for particular tasks
  - Unlabeled data is cheap, labeled data is not.

Goal: Learn a Distribution

- Want to estimate p_{data} from samples

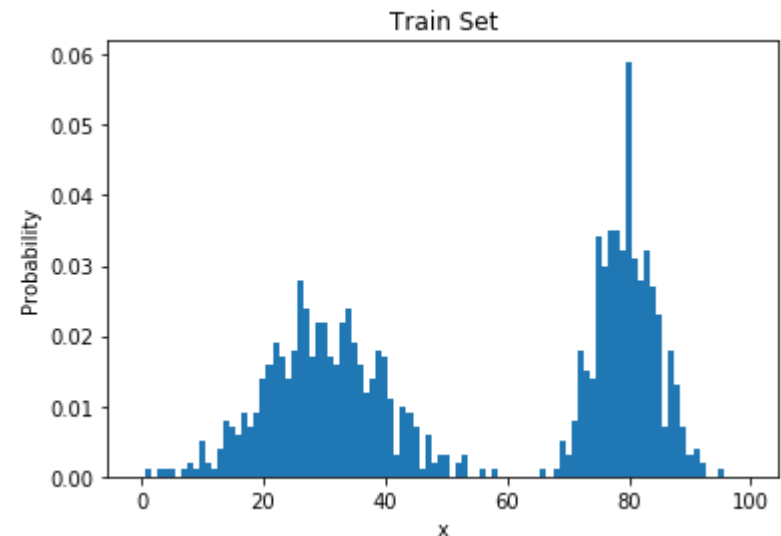$$x^{(1)}, x^{(2)}, \ldots, x^{(n)} \sim p_{\text{data}}(x)$$

- Desired abilities:
  - **Inference**: compute $p(x)$ for some x
  - **Sampling**: obtain a sample from $p(x)$

Goal: Learn a Distribution

- Want to estimate p_{data} from samples

$$x^{(1)}, x^{(2)}, \ldots, x^{(n)} \sim p_{\text{data}}(x)$$

- **One way**: build a histogram:
- Bin data space into k groups.
  - Estimate p_1, p_2, ..., p_k
- Train this model:
  - Count times bin i appears in dataset

Histograms: Inference & Samples

- **Inference**: check our estimate of p_i

- **Sampling**: straightforward, select bin i with probability p_i, then select uniformly from bin i.

- But …
  - inefficient in high dimensions

Parametrizing Distributions

- Don't store each probability, store $p_\theta(x)$

- One approach: likelihood-based
  - We know how to train with **maximum likelihood**

$$\arg\min_\theta -\frac{1}{n}\sum_{i=1}^{n}\log p_\theta(x^{(i)})$$

Parametrizing Distributions

- One approach: likelihood-based
  - We know how to train with **maximum likelihood**

  - Then, train with SGD

  - Just need to make some choices for $p_\theta(x)$
    - For example, recall Gaussian mixture models.
    - But many types of data have more complex underlying distributions.

Parametrizing Distributions: Autoregressive models

- e.g. recurrent neural networks, transformers.
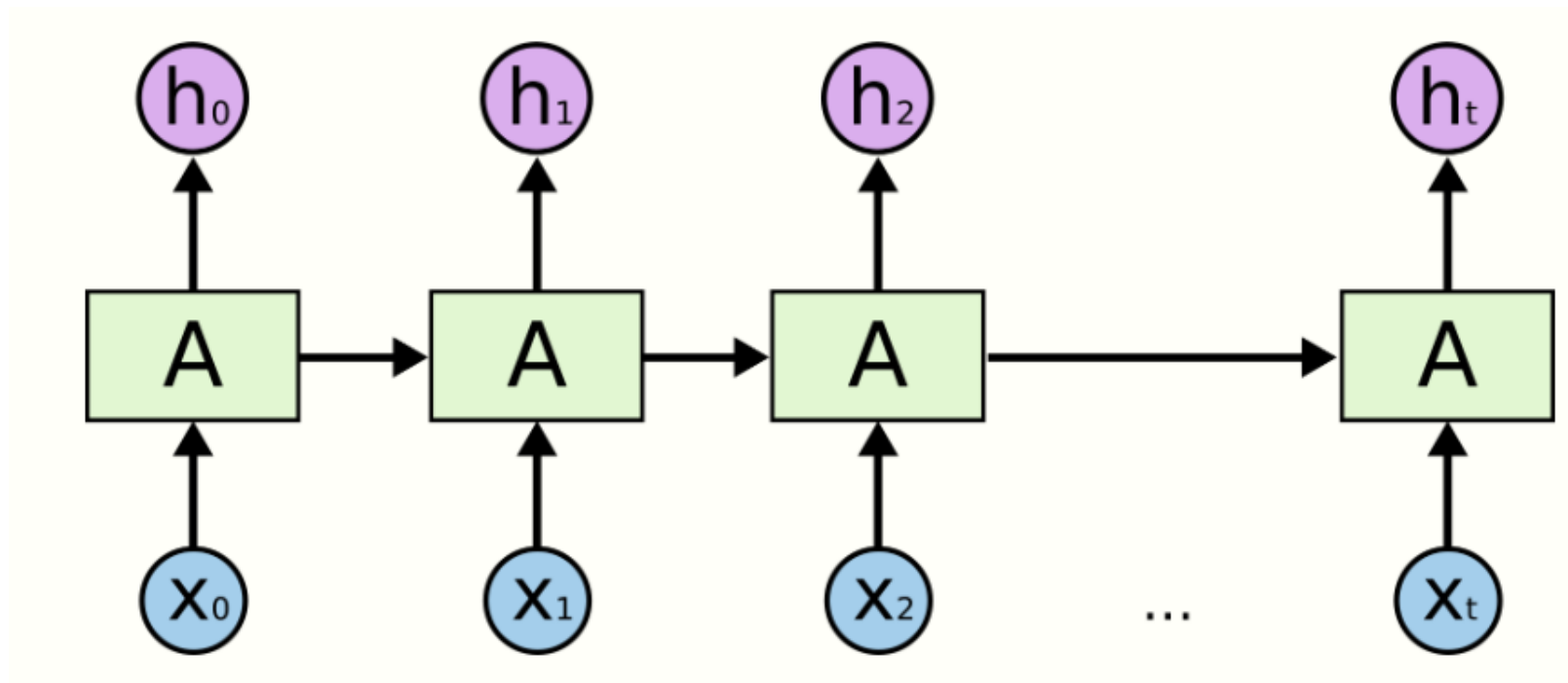
Outline

- **Intro to Generative Models**
  - Histograms, Parametrizing Distributions
- **Flow-based Models**
  - Transformations, training, sampling
- **Generative Adversarial Networks (GANs)**
  - Generators, discriminators, training, examples
- **Diffusion Models**

Flow Models

- One way to specify $p_\theta(x)$

- Use a latent variable z with a "simple" (e.g Gaussian) distribution.

- Then use a "complex" transformation, $x = f_\theta(z)$.

Flow Models

- We will need to compute the inverse transformation and take its derivative as well (for training).
- So compose multiple "simple" transformations

$$x = f_{\theta_k}(f_{\theta_{k-1}}(\ldots f_{\theta_1}(z)))$$

$$z = f_{\theta_1}^{-1}(f_{\theta_2}^{-1}(\ldots f_{\theta_k}^{-1}(x)))$$

Flow Models

- Transform a simple distribution to a complex one via a chain of invertible transformations (the "flow")
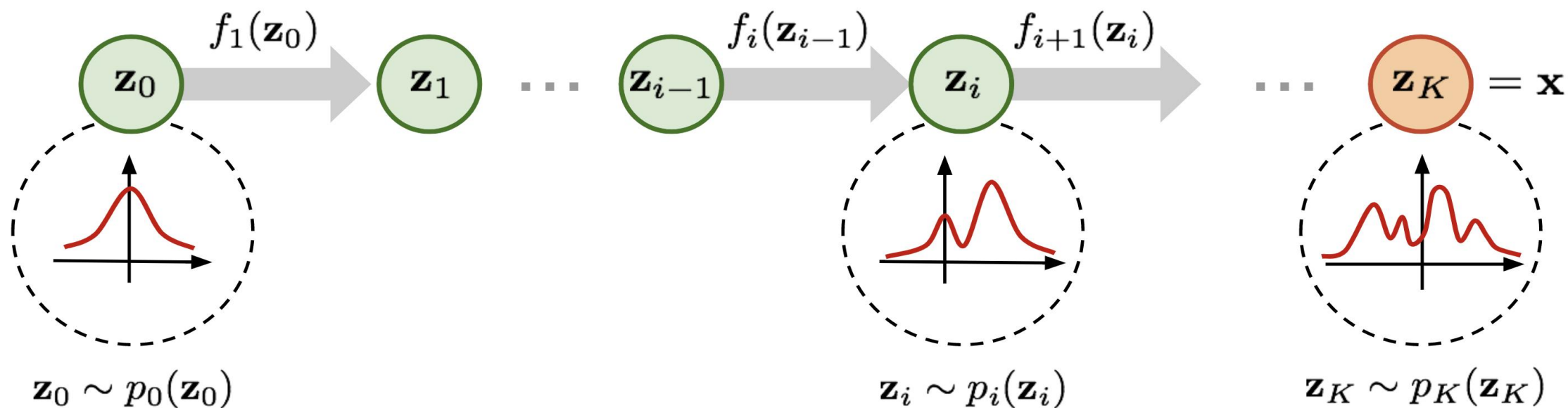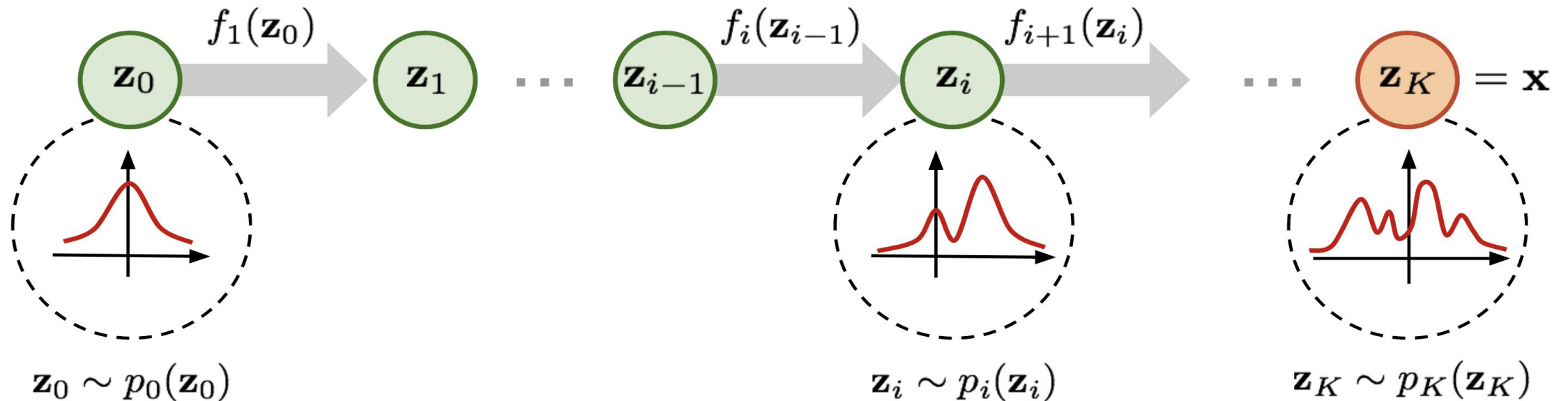

image from Lilian Weng

Flow Models: How to sample?

- Sample from z (the latent variable)---has a simple distribution that lets us do it: Gaussian, uniform, etc.

- Then run the sample z through the flow to get a sample x

Flow Models: How to train?

- Relationship between $p_x(x)$ and $p_z(z)$ (densities of x and z), given that $x = f_\theta(z)$?

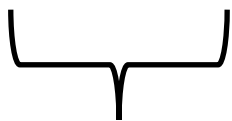$$p_x(x) = p_z(f_\theta^{-1}(x)) \left| \frac{\partial f_\theta^{-1}(x)}{\partial x} \right|$$

[change of variables]

Determinant of Jacobian matrix

Flow Models: Training

$$\max_{\theta} \sum_i \log\left(p_x(x^{(i)}; \theta)\right) = \max_{\theta} \left(\sum_i \log\left(p_z(f_\theta^{-1}(x^{(i)}))\right) + \log\left|\frac{\partial f_\theta^{-1}(x^{(i)})}{\partial x}\right|\right)$$

Maximum Likelihood
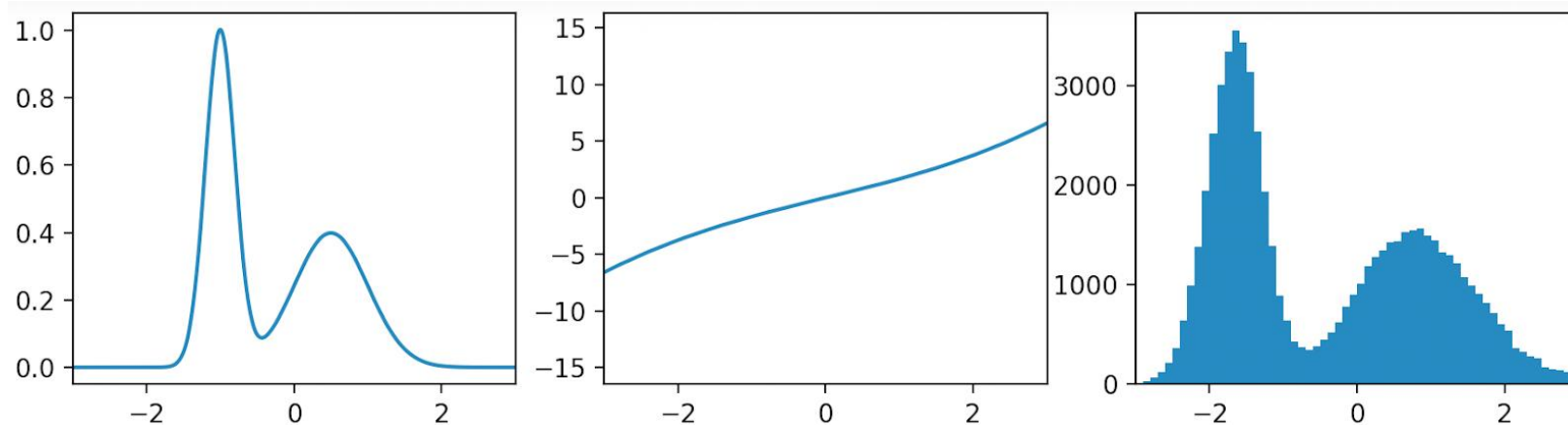
Latent variable version

Determinant of Jacobian matrix

Flows: Example

- Inverse flow to a Gaussian (right)

- **Before** training:

- **After** training:

Inverse Flow

image from UC Berkeley: Deep Unsupervised Training
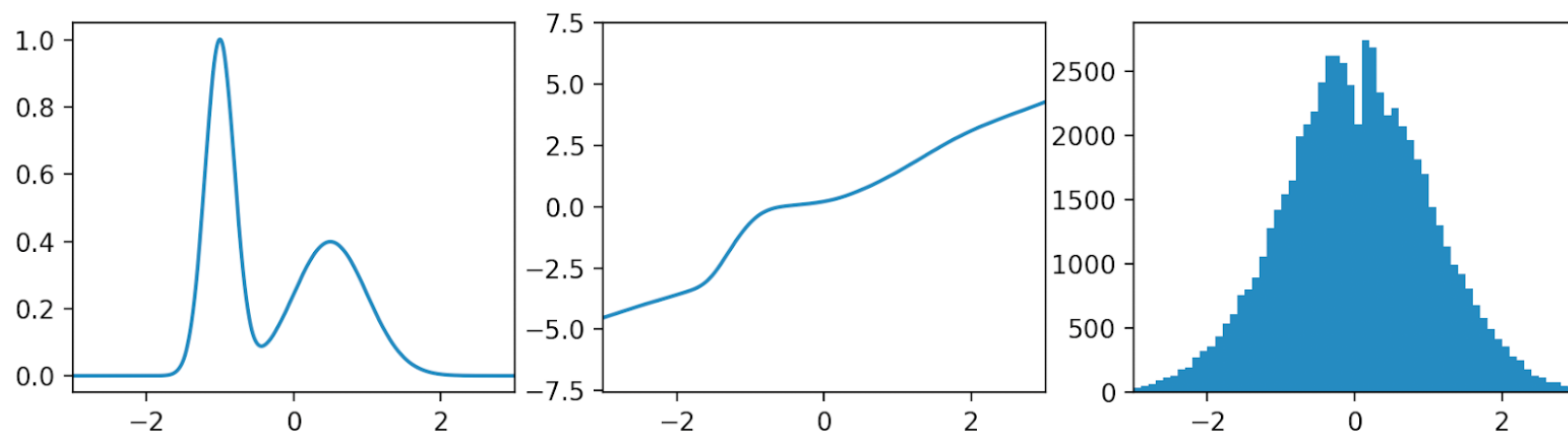
Flows: Transformations

- What kind of f transformations should we use?
- Many choices:
  - Affine: $f(x) = A^{-1}(x - b)$
  - Elementwise: $f(x_1, ..., x_d) = (f(x_1), ..., f(x_d))$
  - Splines

- Desirable properties:
  - Invertible
  - Differentiable

Outline

- **Intro to Generative Models**
  - Histograms, Parametrizing Distributions
- **Flow-based Models**
  - Transformations, training, sampling
- **Generative Adversarial Networks (GANs)**
  - Generators, discriminators, training, examples
- **Diffusion Models**

GANs: Generative Adversarial Networks

- So far we've been modeling the density…
  - What if we just want to get high-quality samples?

- GANs do this.
  - Think of art forgery
  - Left: original
  - Right: forged version
  - Two-player game:
    - **Generator** wants to pass off the discriminator as an original
    - **Discriminator** wants to distinguish forgery from original

GANs: Basic Setup
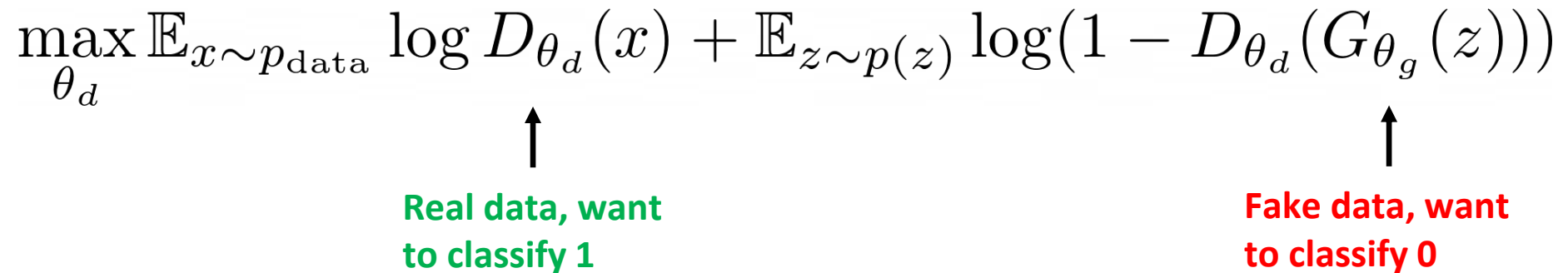
- Let's set up networks that implement this idea:
  - **Discriminator** network
  - **Generator** network

image from Stanford CS231n / Emily Denton
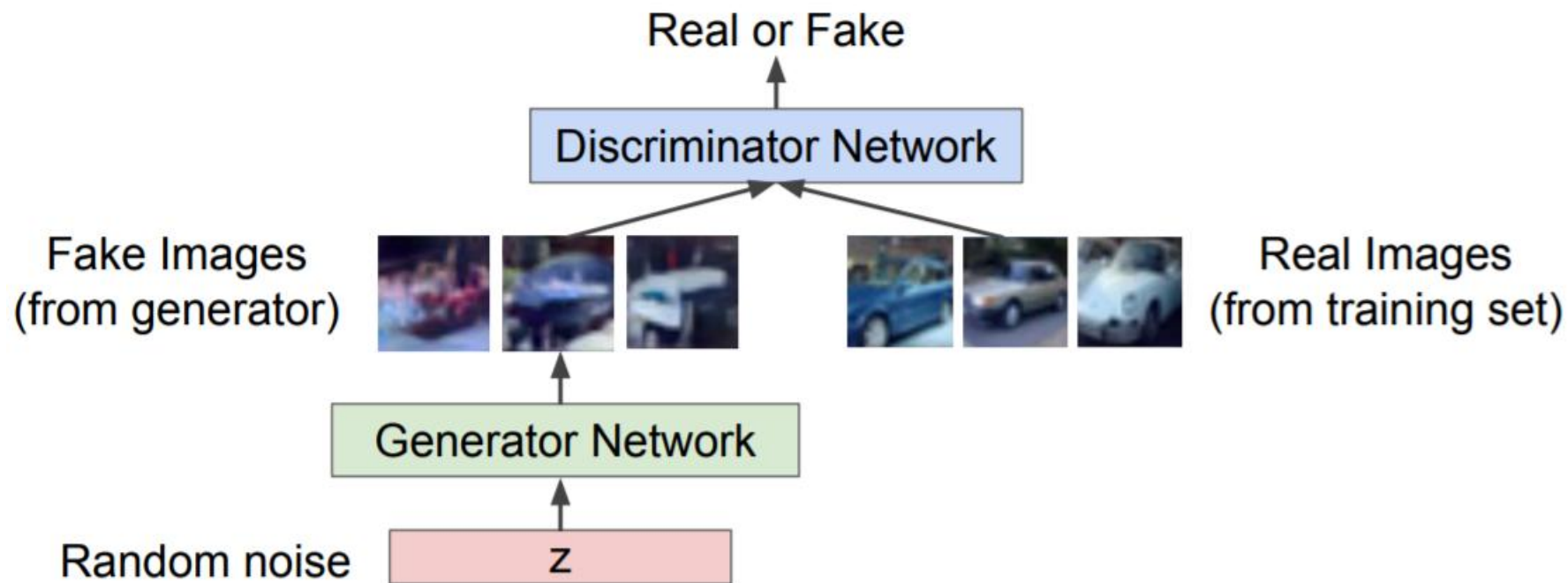
GAN Training: Discriminator

- How to train these networks? Two sets of parameters to learn: θ$_d$ (discriminator) and θ$_g$ (generator)

- Let's **fix** the generator. What should the discriminator do?
  - Distinguish fake and real data: binary classification.
  - Use the cross-entropy loss, we get

$$\max_{\theta_d} \mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Real data, want to classify 1

Fake data, want to classify 0

GAN Training: Generator & Discriminator

- How to train these networks? Two sets of parameters to learn: θ_d (discriminator) and θ_g (generator)

- This makes the discriminator better, but also want to make the generator more capable of fooling it:
  - Minimax game! Train jointly.

$$\min_{\theta_g} \max_{\theta_d} \mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

↑ **Real data, want to classify 1**

↑ **Fake data, want to classify 0**

GAN Training: Alternating Training

- So we have an optimization goal:

$$\min_{\theta_g} \max_{\theta_d} \mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

- Alternate training:
  - **Gradient ascent**: *fix generator*, make the discriminator better:

$$\max_{\theta_d} \mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

  - **Gradient descent**: *fix discriminator*, make the generator better

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

GAN Training: Issues

- Training often not stable
- Many tricks to help with this:
  - Replace the generator training with

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

  - Better gradient shape
  - Choose number of alternating steps carefully

- Can still be challenging.

GAN Architectures

- **Discriminator**: image classification, use a **CNN**
- What should **generator** look like
  - Input: noise vector z.
  - Output: an image (i.e. a 3-channel x width x height volume)
  - Similar to a reversed CNN pattern...
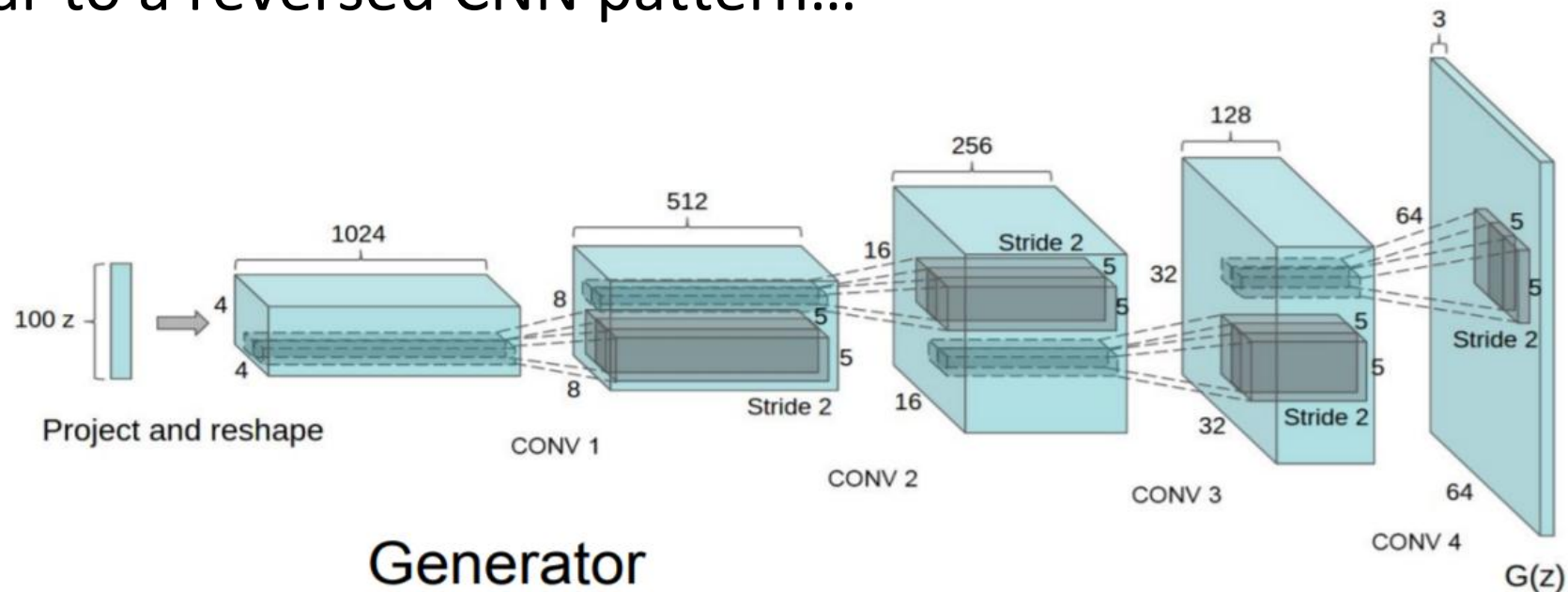
Generator

GANs: Example

- Output of a GAN after 5 epochs of training:
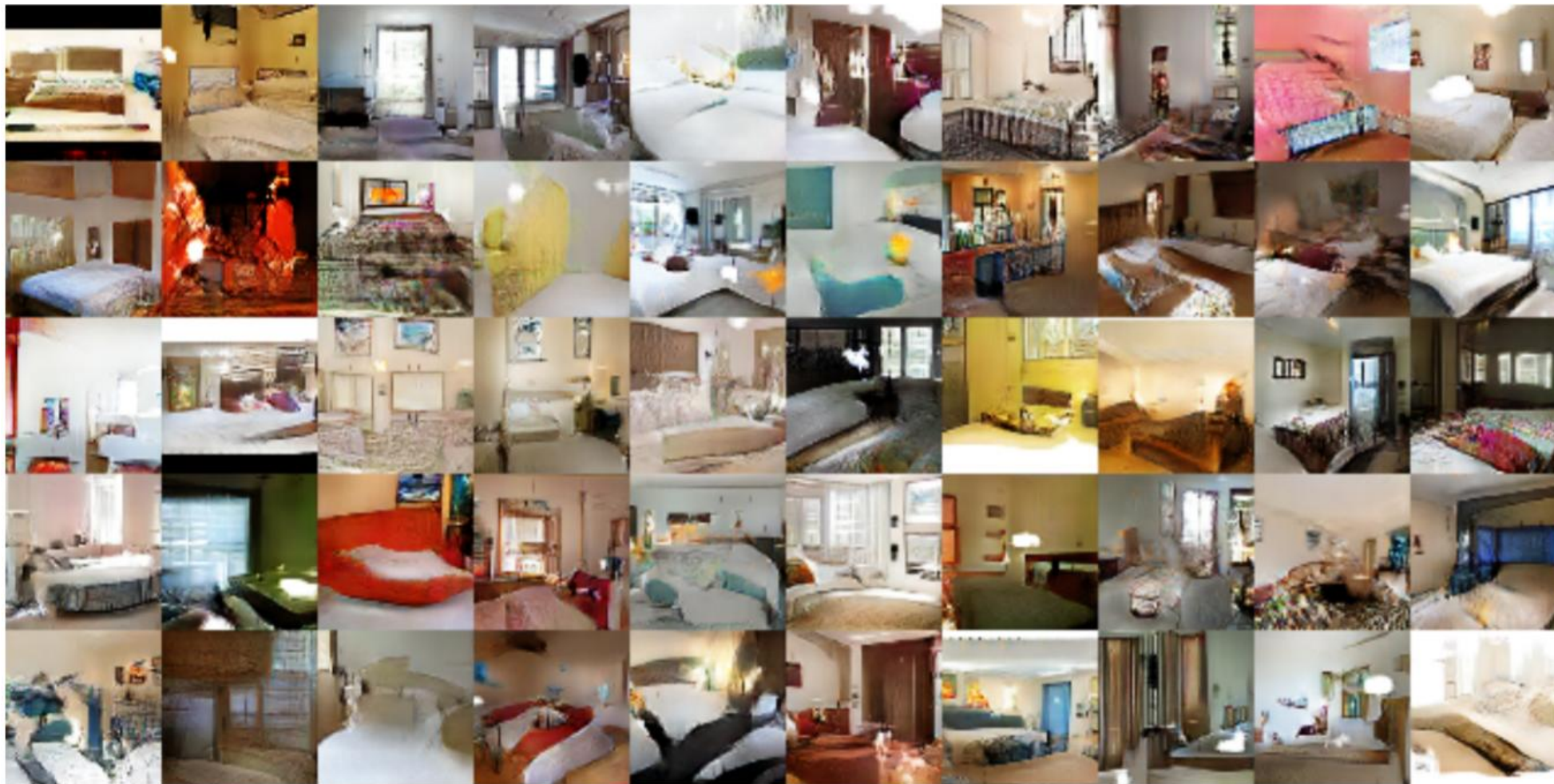
Outline

- **Intro to Generative Models**
  - Histograms, Parametrizing Distributions
- **Flow-based Models**
  - Transformations, training, sampling
- **Generative Adversarial Networks (GANs)**
  - Generators, discriminators, training, examples
- **Diffusion Models**

Diffusion Models (optional)

- **Learning to generate by denoising**

- Denoising diffusion models consist of two processes:
  - Forward diffusion process that gradually adds noise to input
  - Reverse denoising process that learns to generate data by denoising

Forward diffusion process (fixed)

Data

Noise

Reverse denoising process (generative)
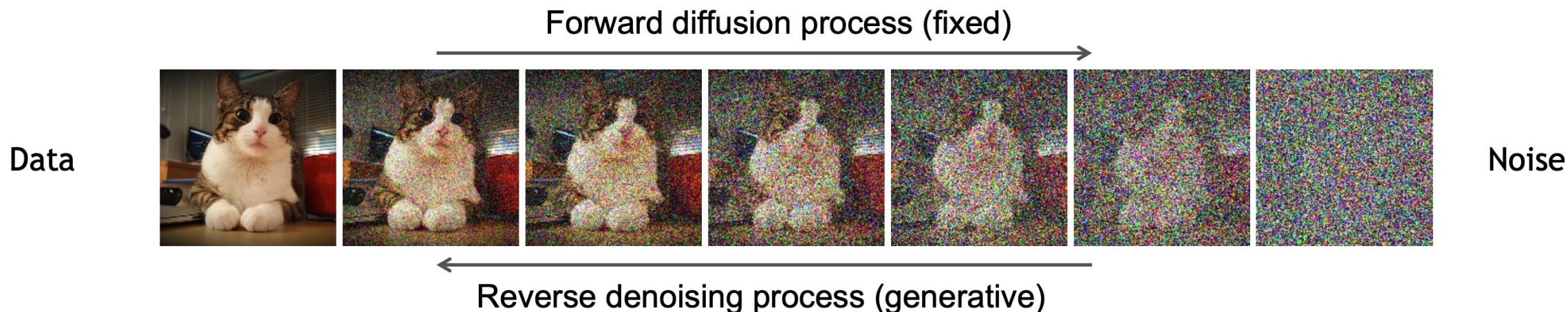
Diffusion Models (optional)

- The formal definition of the forward process in T steps:

Forward diffusion process (fixed)

Data ... Noise

$\mathbf{x}_0$ $\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$ ... $\mathbf{x}_T$

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad \Rightarrow \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \qquad \text{(joint)}$$
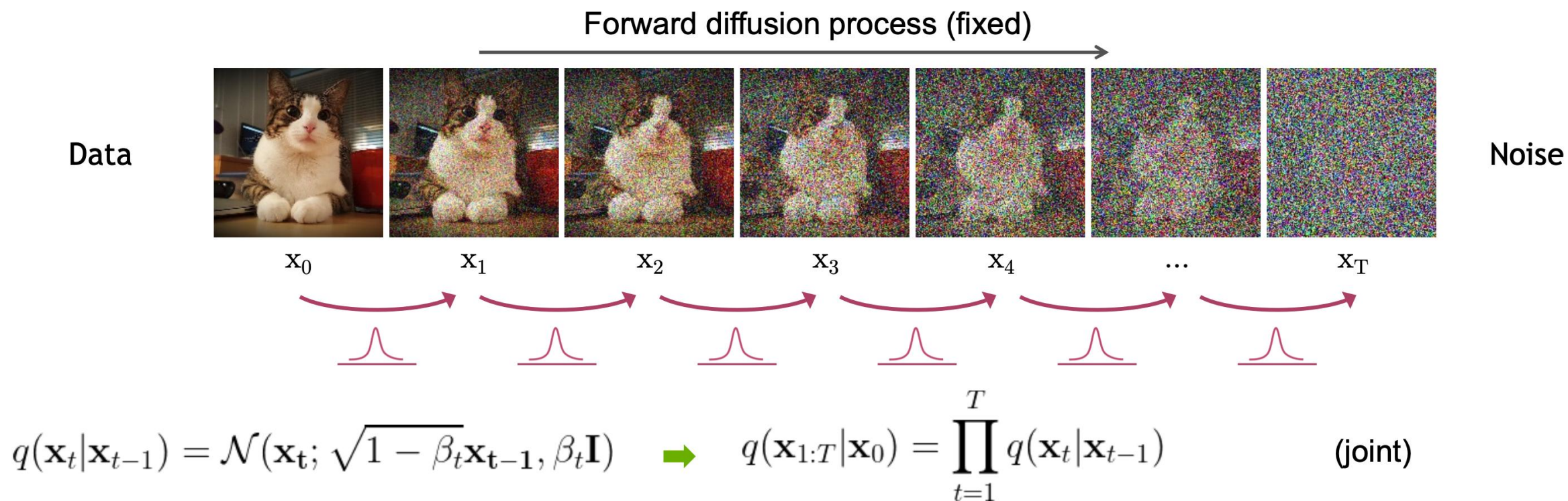
Diffusion Models (optional)

- Diffusion Kernel

Forward diffusion process (fixed)

Data

Noise

x_0 x_1 x_2 x_3 x_4 ... x_T
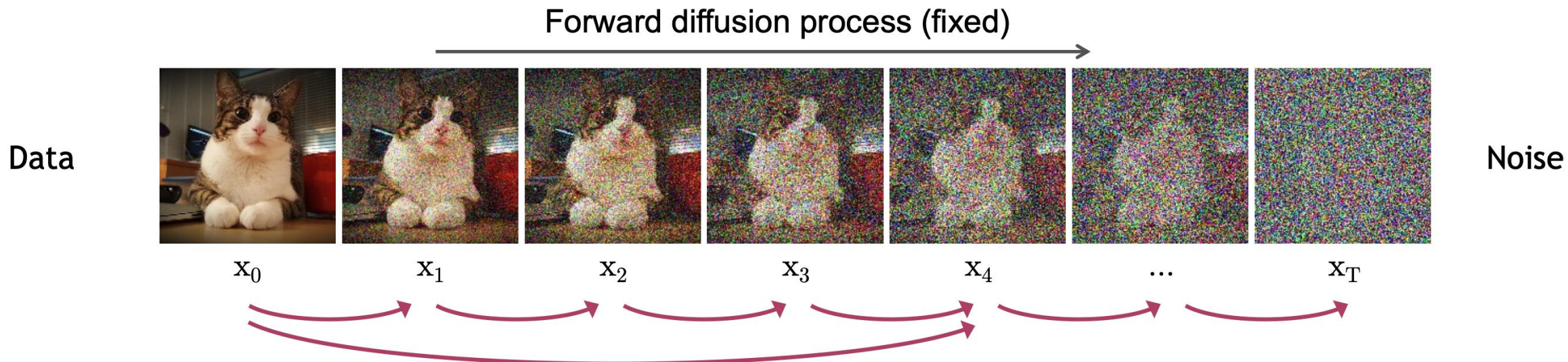
Define $\bar{\alpha}_t = \prod_{s=1}^{t}(1 - \beta_s)$ ➡ $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I}))$ (Diffusion Kernel)

For sampling: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{(1 - \bar{\alpha}_t)}\,\epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

β_t values schedule (i.e., the noise schedule) is designed such that $\bar{\alpha}_T \to 0$ and $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$
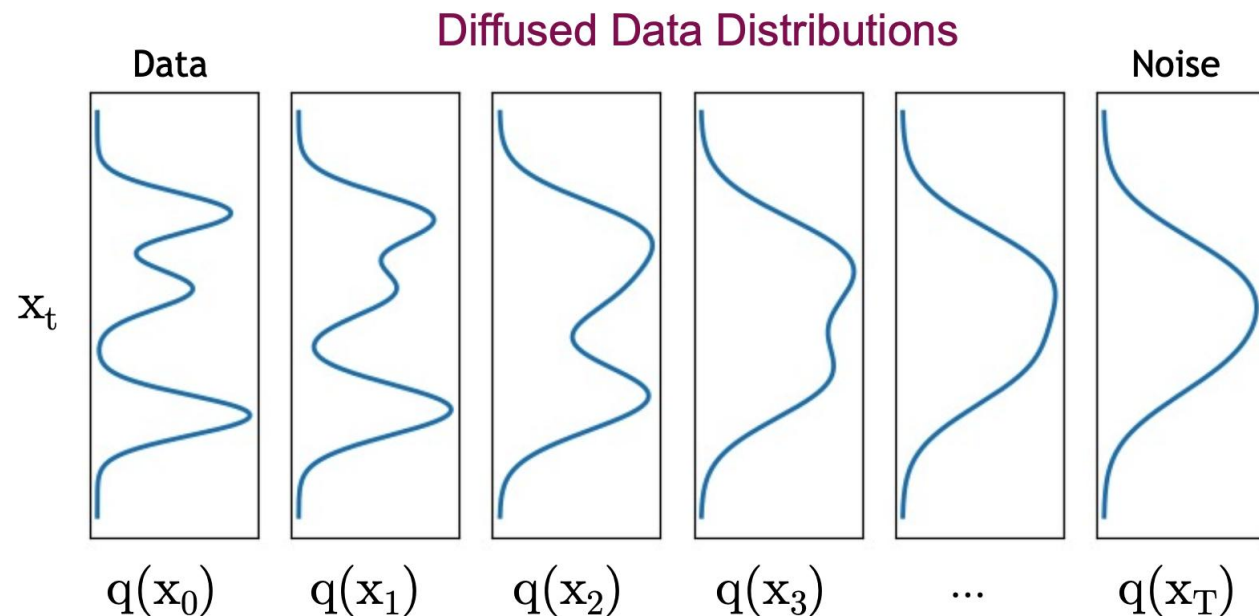
Diffusion Models (optional)

- What happens to a distribution in the forward diffusion?

So far, we discussed the diffusion kernel $q(\mathbf{x}_t|\mathbf{x}_0)$ but what about $q(\mathbf{x}_t)$?

Diffused Data Distributions

$$q(\mathbf{x}_t) = \int \underbrace{q(\mathbf{x}_0, \mathbf{x}_t)}_{\text{Joint dist.}} d\mathbf{x}_0 = \int \underbrace{q(\mathbf{x}_0)}_{\substack{\text{Input} \\ \text{data dist.}}} \underbrace{q(\mathbf{x}_t|\mathbf{x}_0)}_{\substack{\text{Diffusion} \\ \text{kernel}}} d\mathbf{x}_0$$

Diffused data dist.

The diffusion kernel is Gaussian convolution.

$q(x_0)$ $q(x_1)$ $q(x_2)$ $q(x_3)$ ... $q(x_T)$

We can sample $\mathbf{x}_t \sim q(\mathbf{x}_t)$ by first sampling $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ and then sampling $\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0)$ (i.e., ancestral sampling).

Diffusion Models (optional)

- Generative Learning by Denoising

Recall, that the diffusion parameters are designed such that $q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$
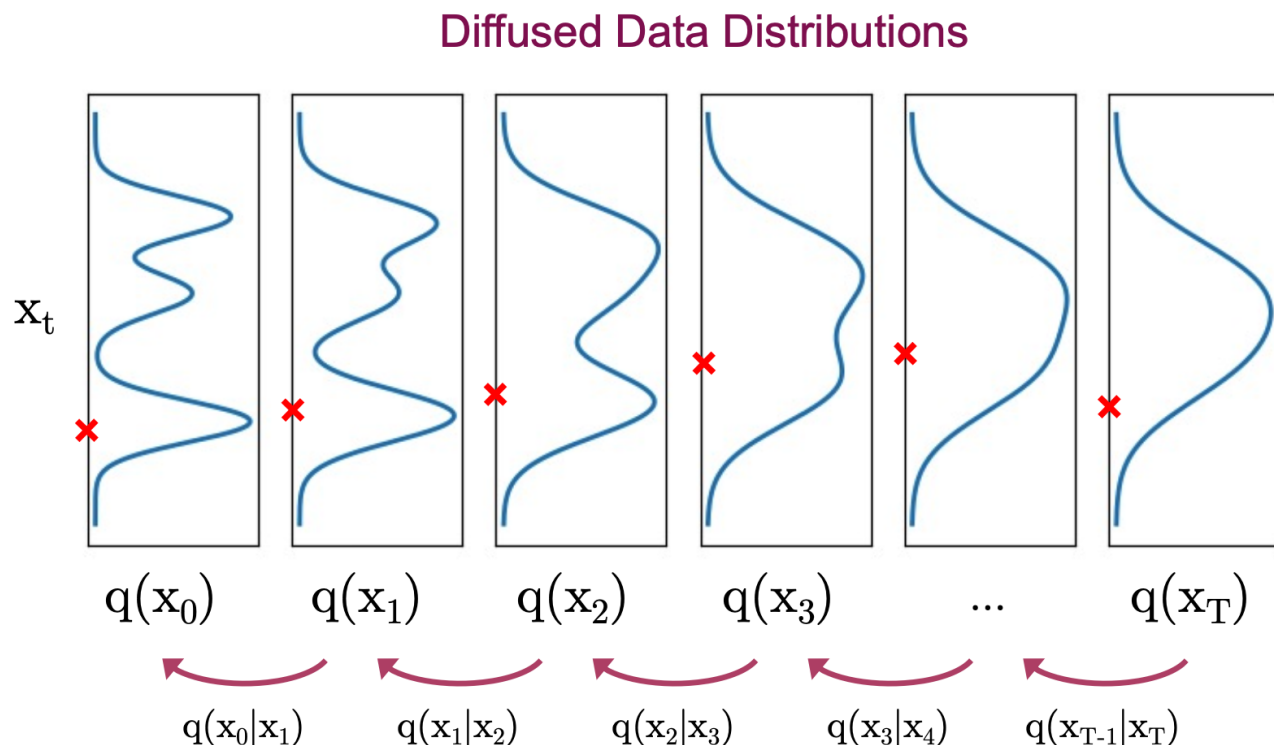

Diffused Data Distributions

Generation:

Sample $\mathbf{x}_T \sim \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

Iteratively sample $\mathbf{x}_{t-1} \sim \underbrace{q(\mathbf{x}_{t-1}|\mathbf{x}_t)}_{\text{True Denoising Dist.}}$

$q(\mathbf{x}_0) \quad q(\mathbf{x}_1) \quad q(\mathbf{x}_2) \quad q(\mathbf{x}_3) \quad \dots \quad q(\mathbf{x}_T)$

$q(\mathbf{x}_0|\mathbf{x}_1) \quad q(\mathbf{x}_1|\mathbf{x}_2) \quad q(\mathbf{x}_2|\mathbf{x}_3) \quad q(\mathbf{x}_3|\mathbf{x}_4) \quad q(\mathbf{x}_{T-1}|\mathbf{x}_T)$

In general, $q(\mathbf{x}_{t-1}|\mathbf{x}_t) \propto q(\mathbf{x}_{t-1})q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is intractable.

Can we approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$? Yes, we can use a Normal distribution if β_t is small in each forward diffusion step.
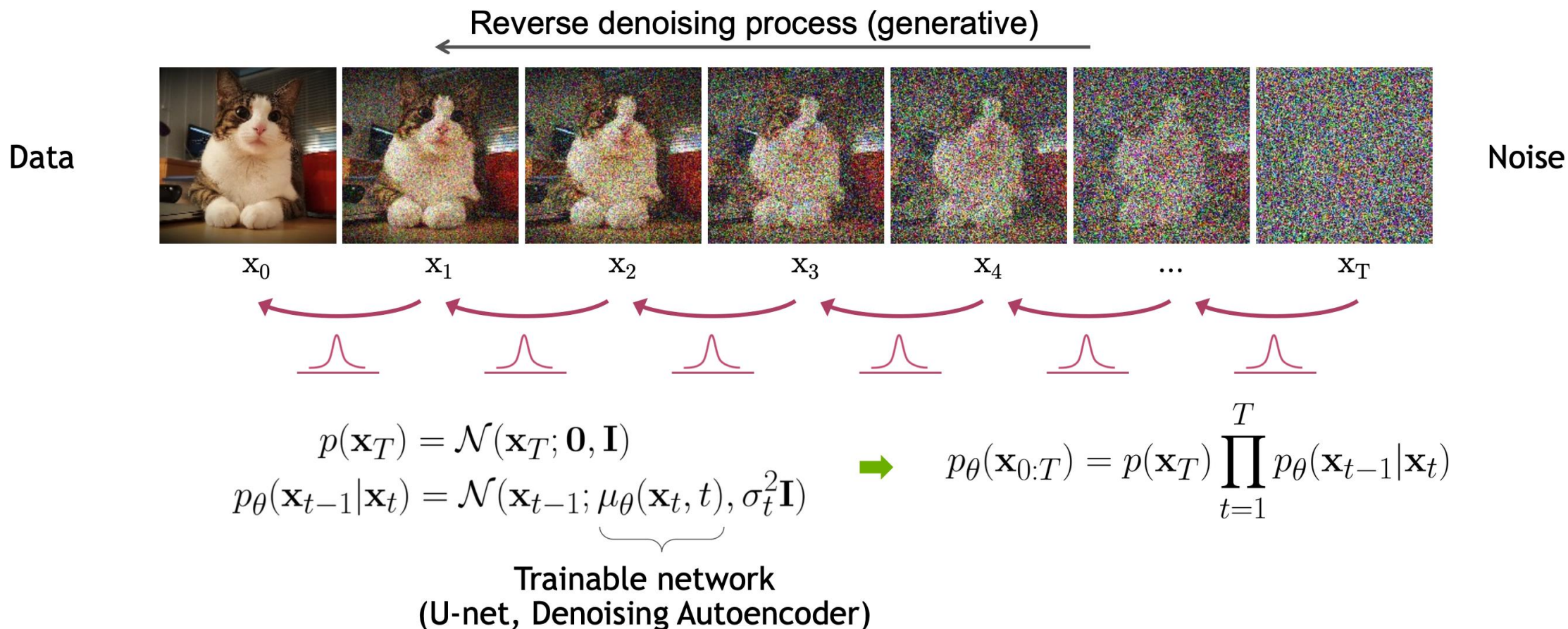
Diffusion Models (optional)

- Reverse Denoising Process

Formal definition of forward and reverse processes in T steps:

Reverse denoising process (generative)

Data

Noise

$\mathbf{x}_0 \quad \mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \mathbf{x}_4 \quad \dots \quad \mathbf{x}_T$

$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \underbrace{\mu_\theta(\mathbf{x}_t, t)}, \sigma_t^2 \mathbf{I})$$

Trainable network
(U-net, Denoising Autoencoder)

$$\rightarrow \quad p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$

Thanks Everyone!

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, Pedro Domingos, Jerry Zhu, Yingyu Liang, Volodymyr Kuleshov, Fei-Fei Li, Justin Johnson, Serena Yeung, Pieter Abbeel, Peter Chen, Jonathan Ho, Aravind Srinivas, Ruiqi Gao, Tengyang Xie