# CS 839: Foundation Models
# **Diffusion Models**

Fred Sala

University of Wisconsin-Madison

**Oct. 30, 2025**

# Announcements

- **Logistics:**
  - Project info out
  - Come chat about presentation/project proposals.
  - Class roadmap:

| | |
|---|---|
| Thursday Oct. 30 | Diffusion Models |
| Tuesday Nov. 4 | Scaling & Scaling Laws |
| Thursday Nov. 6 | Security, Privacy, Toxicity + Future Areas |

# Outline

- **Generative Models Overview**
  - Basic idea, complexity challenges, overview of major image generation techniques, intuitions
- **Normalizing Flows & GANs**
  - Normalizing flow transformations, training, sampling, GAN generators, discriminators, training
- **Diffusion Models**
  - Overall intuition, score-based training, controlling and latent space formulations, extensions

# Outline

- **Generative Models Overview**
  - Basic idea, complexity challenges, overview of major image generation techniques, intuitions
- **Normalizing Flows & GANs**
  - Normalizing flow transformations, training, sampling, GAN generators, discriminators, training
- **Diffusion Models**
  - Overall intuition, score-based training, controlling and latent space formulations, extensions

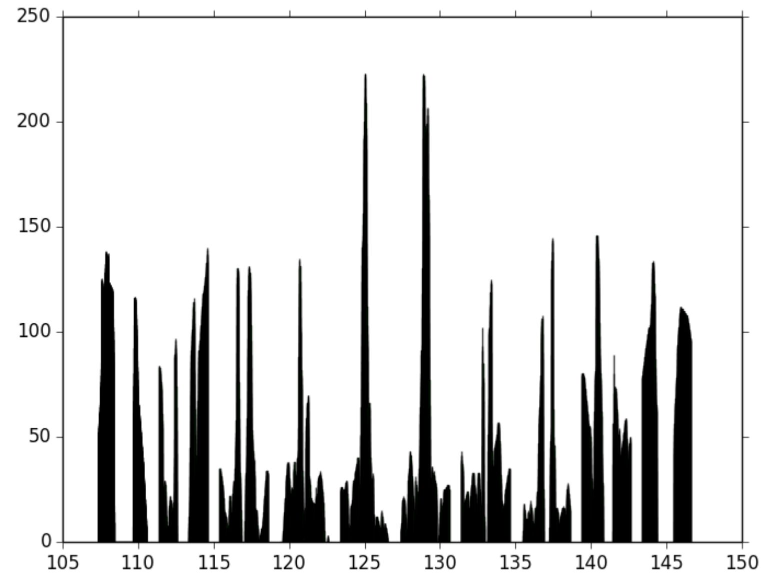
# **Goal**: Learn a Distribution

- Want to estimate $p_{data}$ from samples

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

- Useful abilities to have:
  - **Inference**: compute p(x) for some x
  - **Sampling**: obtain a sample from p(x)

- As always need efficiency for this too…



# Directly Modeling the Distribution

- Want to estimate $p_{data}$ from samples

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

- One straightforward idea: **parametrize the pdf of the distribution**. To train, maximize the log likelihood

$$\max_{\theta} \sum_{i=1}^{N} \log p_{\theta}(x_i).$$

- However, we'll face some challenges…
  - Why? Both training and inference can be complex

# **Goal**: Learn a Distribution

- Want to estimate $p_{data}$ from samples

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

- Let's set

$$p_\theta(x) = \frac{1}{Z}\exp(f_\theta(x))$$

**Energy function**

- Have to deal with the normalizing **partition function Z**,

$$Z_\theta = \int \exp(f_\theta(x))dx$$

**Usually intractable!**

# Getting Around the Partition Function

**All** gen. modeling techniques must deal with this. How?

- Avoid modeling the pdf explicitly
  - → **GANs**
- Choose special choices of p/f that keeps Z tractable
  - → Certain **normalizing flows**
- Use approximations
  - → **VAEs**, using ELBO-style bounds
- Obtain training objectives that sidestep maximum likelihood
  - → **GANs**, **score-based diffusion models**

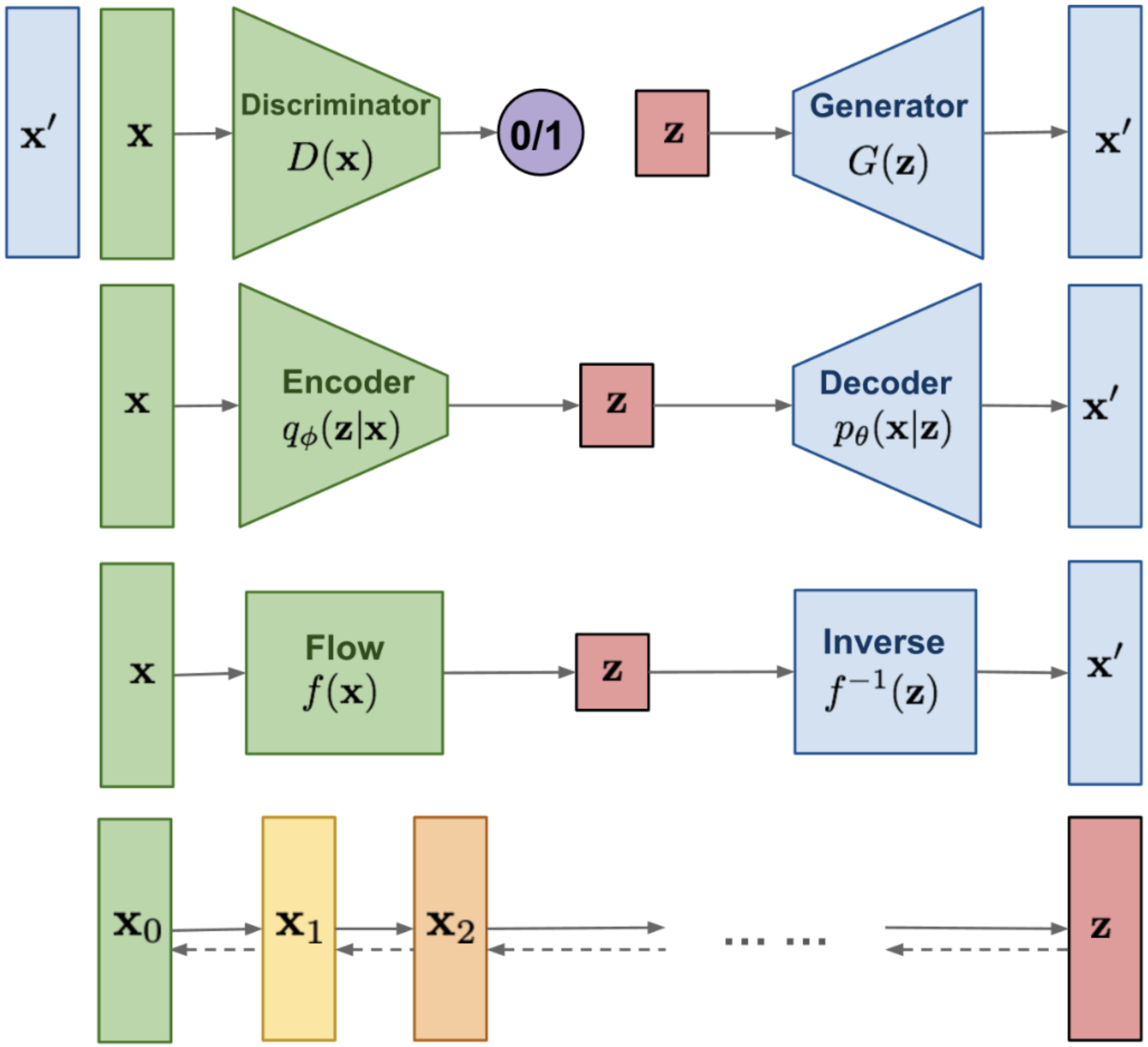
# Generative Modeling Approaches

**GAN:** Adversarial training

**VAE:** maximize variational lower bound

**Flow-based models:** Invertible transform of distributions

**Diffusion models:** Gradually add Gaussian noise and then reverse

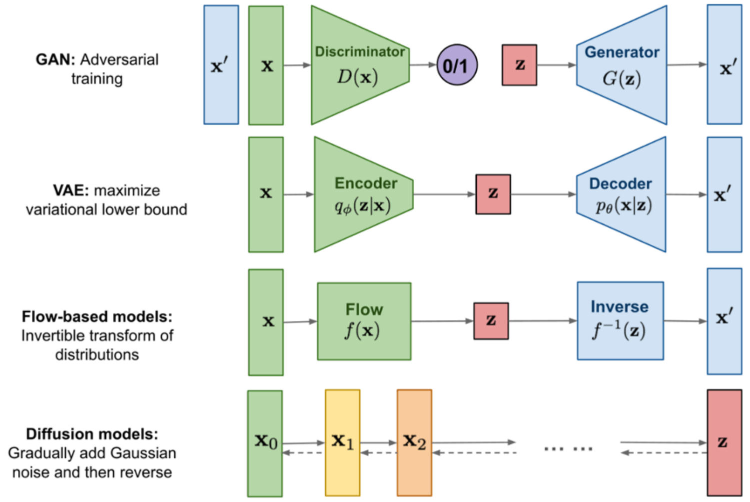



Lilian Weng

# Generative Modeling Intuitions

We can think of GMs as doing two things:

- "**Mapping**" a **simple** (fake) distribution into a **complex** (real) distribution
  - Why? Sample from simple distribution, then transform with learned map
  - "**Latent space**" interpretation
- Learning to undo noise or undo a particular transformation
  - Related to self-supervised learning



Lilian Weng

Combine with previous training considerations to get various techniques

# Break & Questions

# Outline

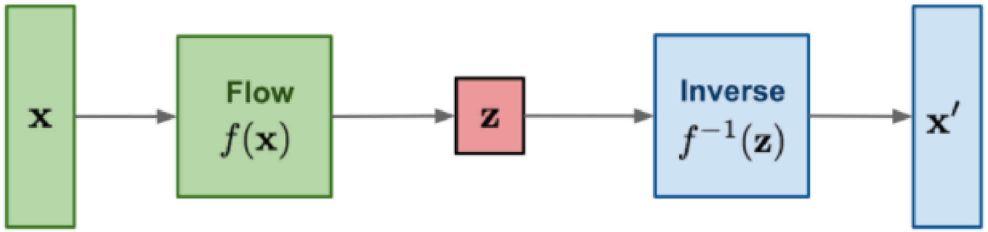
- **Generative Models Overview**
  - Basic idea, complexity challenges, overview of major image generation techniques, intuitions
- **Normalizing Flows & GANs**
  - Normalizing flow transformations, training, sampling, GAN generators, discriminators, training
- **Diffusion Models**
  - Overall intuition, score-based training, controlling and latent space formulations, extensions

# Flow Models

- Want to fit $p_\theta(x)$, as we described
- Some goals:
  - Good fit for the data
  - Computing a probability: the actual value of $p_\theta(x)$ for some x
  - Ability to sample
  - Also: a **latent representation**

- Won't model $p_\theta(x)$ directly… instead we'll get some latent variable z

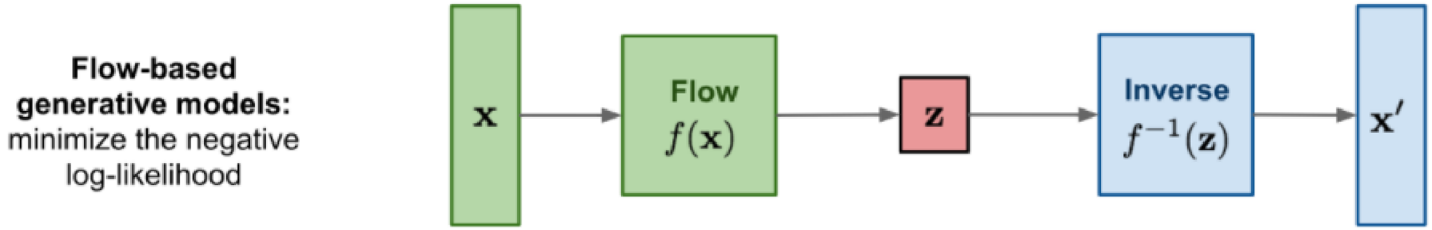
**Flow-based generative models:** minimize the negative log-likelihood

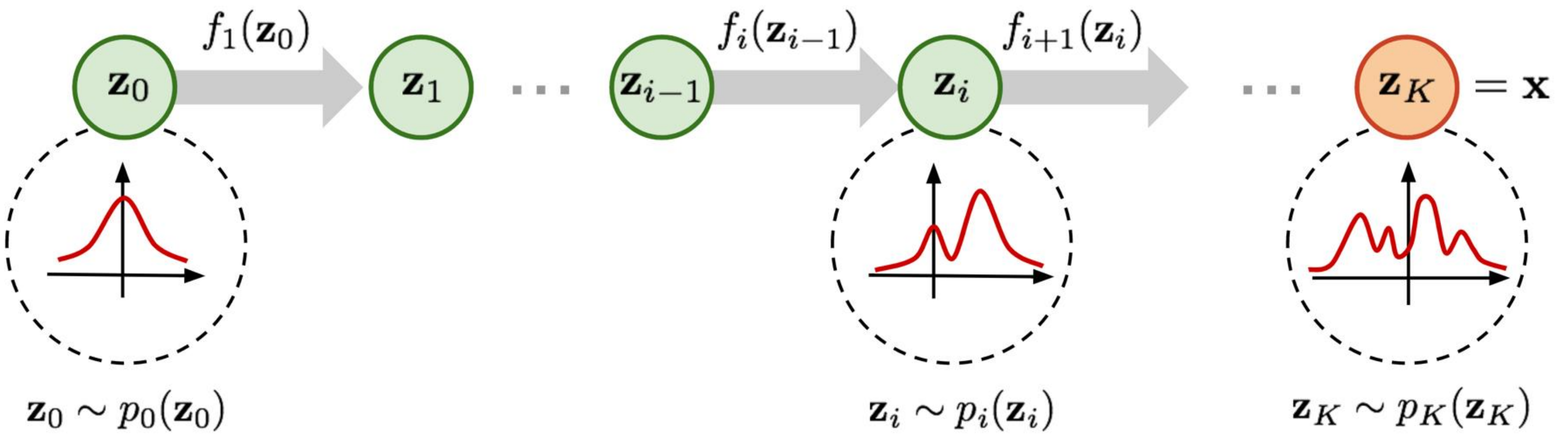



Lilian Weng

# Flow Models

- **Key idea**: transform a simple distribution to complex
  - Use a chain of transformations (the "flow")

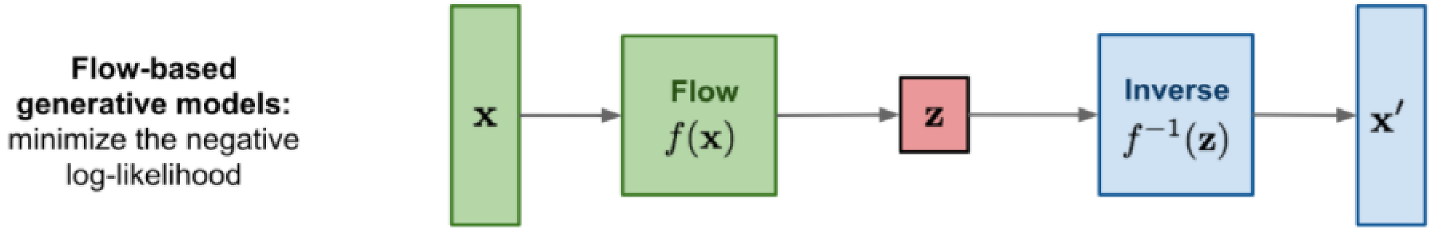



Lilian Weng

# Flow Models

- **Key idea**: transform a simple distribution to complex
  - Use a chain of invertible transformations (the “flow”)




  - 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 inverse flow to get a sample x
  - How to train? Let’s see…

# **Flow Models:** Density Relationships

- **Key idea**: transform a simple distribution to complex
  - Use a chain of transformations (the "flow")

- How does each transformation affect the density p?

Latent variable → $z$; Transformation → $f_\theta$

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

$$p_\theta(x)\, dx = p(z)\, dz$$

$$p_\theta(x) = p(f_\theta(x)) \left| \frac{\partial f_\theta(x)}{\partial x} \right|$$

Determinant of Jacobian matrix

# **Flow Models:** Training

- **Key idea**: transform a simple distribution to complex
  - Use a chain of transformations (the "flow")

- How does training change?
  - **Idea**: might be easier to optimize $p_Z$

$$\max_\theta \sum_i \log p_\theta(x^{(i)}) = \max_\theta \sum_i \log p_Z(f_\theta(x^{(i)})) + \log \left| \frac{\partial f_\theta}{\partial x}(x^{(i)}) \right|$$

**Maximum Likelihood**

**Latent variable version**

**Determinant of Jacobian matrix**

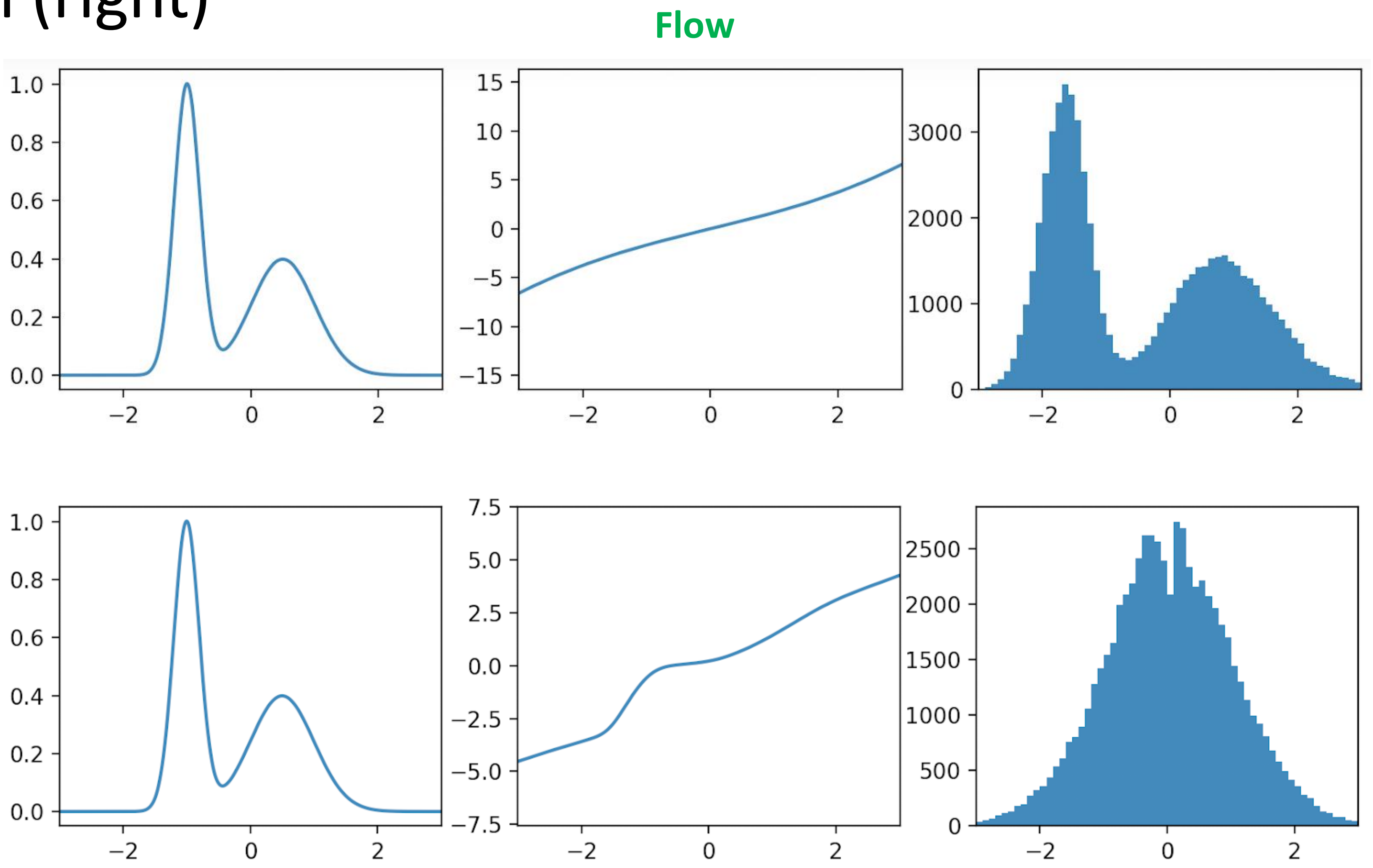
Can extend to many chained transformations…

# **Flows**: Example

- Flow to a Gaussian (right)

- **Before** training:

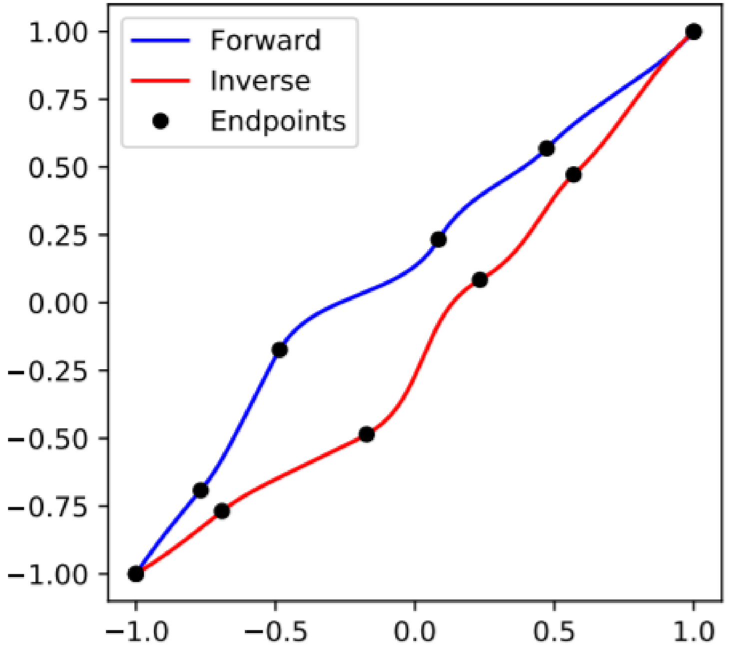
- **After** training:




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: $f(x_1, \ldots, x_d) = (f(x_1), \ldots, f(x_d))$
  - Splines:
- Desirable properties:
  - Invertible
  - Differentiable (forward and inverse)




(a) Forward and inverse transformer

Papamakarios et al' 21

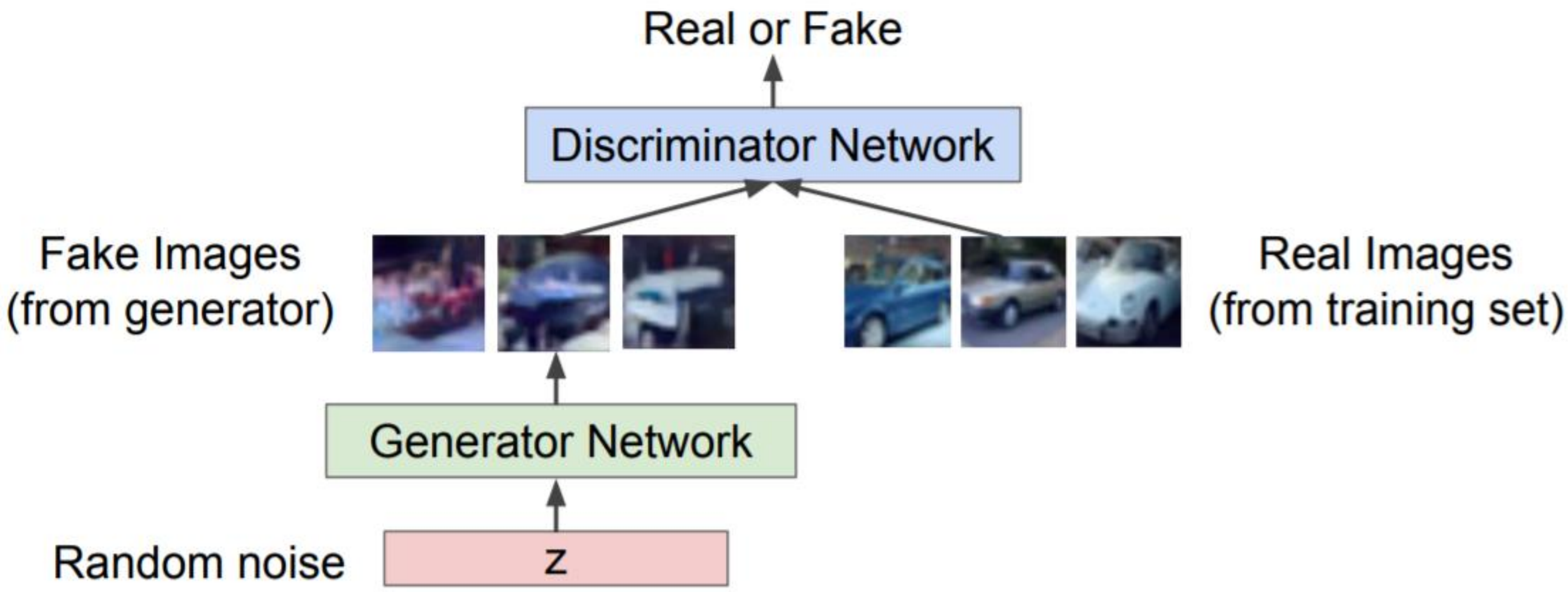
# **GANs**: Generative Adversarial Networks

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

- GANs do this. Based on a clever idea:
  - Art forgery: very common through history
  - Left: original
  - Right: forged version
  - Two-player game. **Forger** wants to pass off the forgery as an original; **investigator** wants to distinguish forgery from original

# **GANs**: Basic Setup

- Let’s set up networks that implement this idea:
  - Discriminator network: like the **investigator**
  - Generator network: like the **forger**




Stanford CS231n / Emily Denton

# **GAN** Training: Discriminator

- How to train these networks? Two sets of parameters to learn: $\theta_d$ (discriminator) and $\theta_g$ (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: $\theta_d$ (discriminator) and $\theta_g$ (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

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

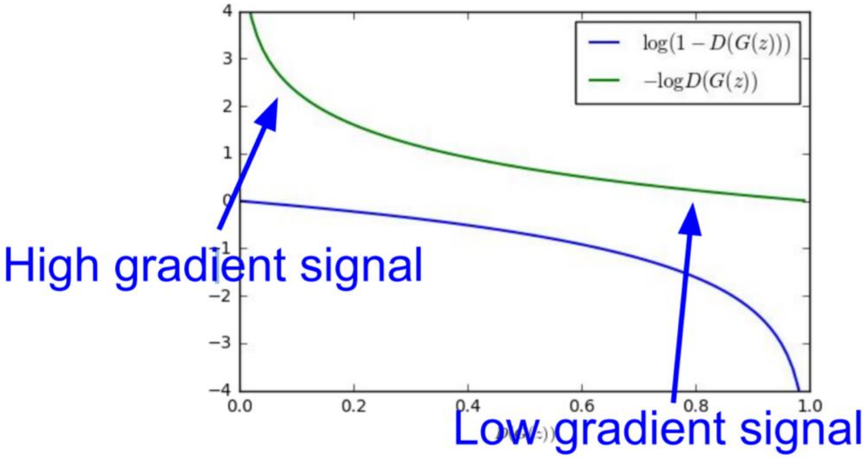
# **GAN** Training: Issues

- 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 alt. steps carefully

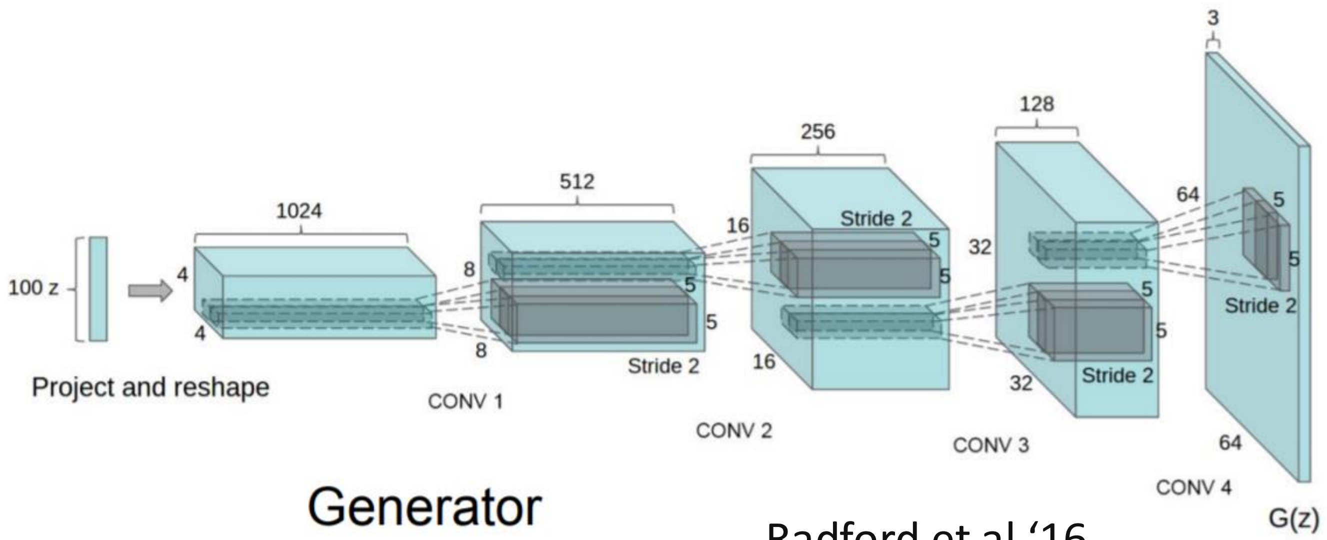
- Can still be challenging.




Stanford CS231n

# **GAN** Architectures

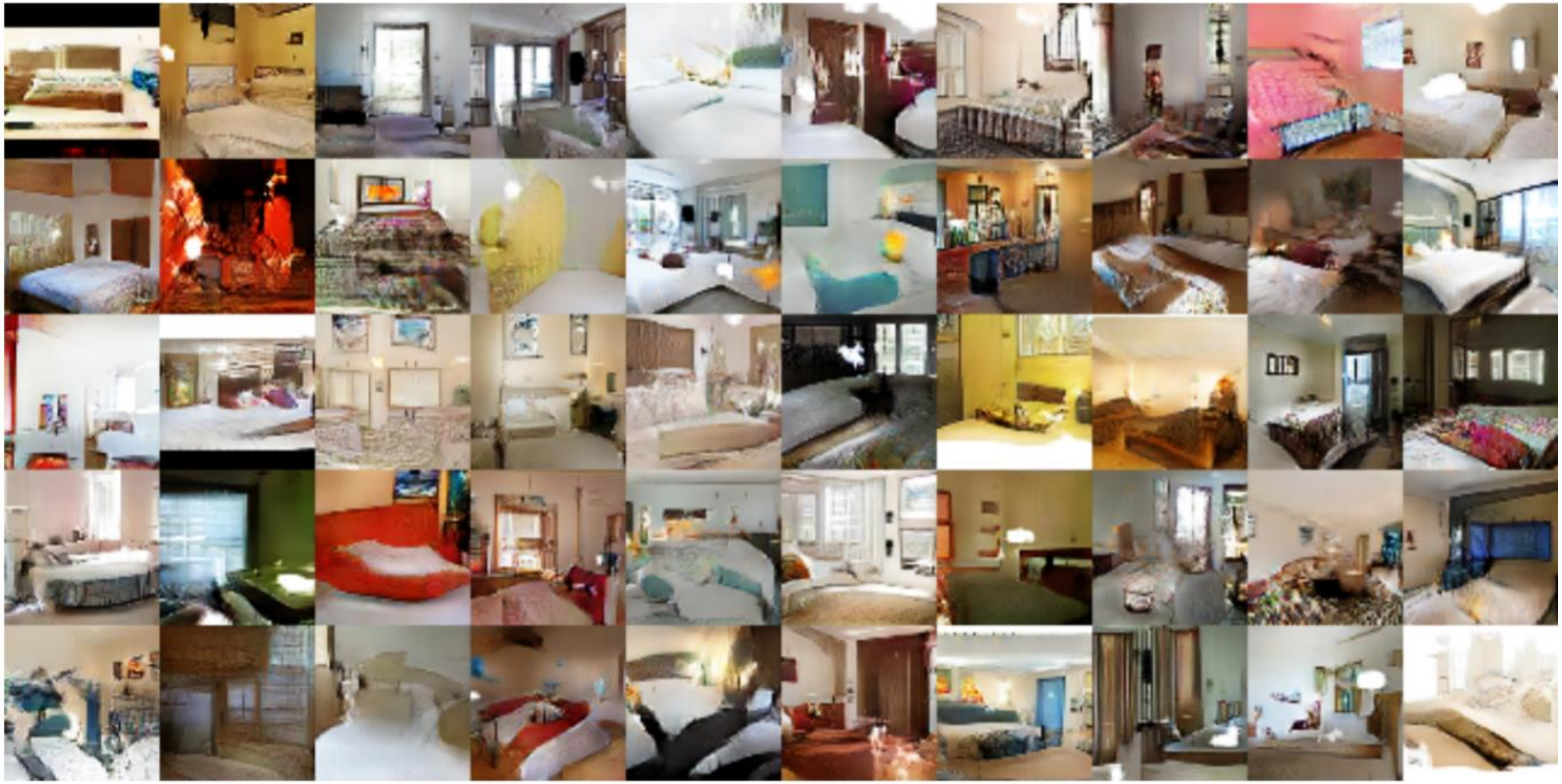
- So far we haven't commented on what the networks are
- **Discriminator**: image classification, use a **CNN**
- What should **generator** look like
  - Input: noise vector z. Output: an image (ie, volume 3 x width x height)
  - Can just reverse our CNN pattern…




Radford et al '16

# **GANs**: Example

- From Radford's paper, with 5 epochs of training:



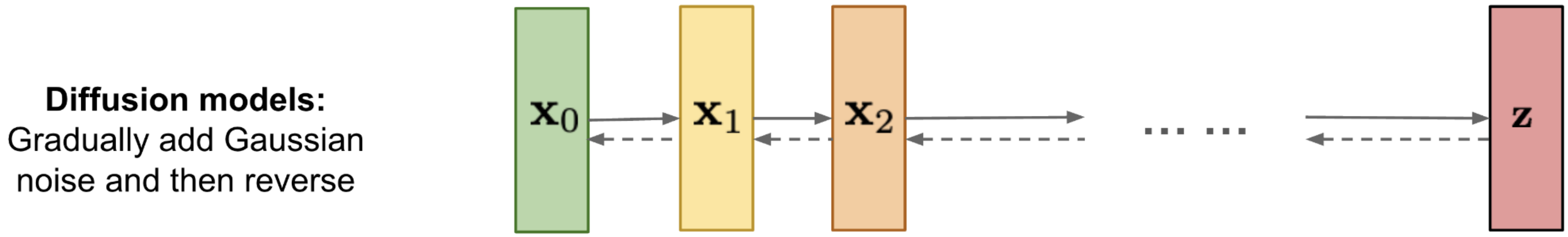
# Break & Questions

# Outline

- **Generative Models Overview**
  - Basic idea, complexity challenges, overview of major image generation techniques, intuitions
- **Normalizing Flows & GANs**
  - Normalizing flow transformations, training, sampling, GAN generators, discriminators, training
- **Diffusion Models**
  - Overall intuition, score-based training, controlling and latent space formulations, extensions

# Diffusion Models Idea

- Let’s return to something that looks like a normalizing flow,




Lilian Weng

- Really a large family of techniques that share some common properties
  - But have been derived from different starting principles / desired properties

# Score-Based Generative Models

- How do we avoid running into the partition function?
- Let’s not model the pdf
- Instead, model the “**score**”

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

- Score: gradient of the log likelihood with respect to the data.
- Goal: train s such that

$$\mathbf{s}_\theta(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})$$

# Score-Based Generative Models

Instead, model the "**score**" $\nabla_{\mathbf{x}} \log p(\mathbf{x})$

Goal: train s such that

$$\mathbf{s}_\theta(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})$$

- Why does this avoid the partition function?
- Let's plug in our energy-based function from earlier. We get:

Gradient w.r.t. **x**, **not** θ

$$\mathbf{s}_\theta(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) = -\nabla_{\mathbf{x}} f_\theta(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z_\theta}_{=0} = -\nabla_{\mathbf{x}} f_\theta(\mathbf{x}).$$

# Training & Inference for Score-Based Models

- Training: can directly run M.S.E. as a loss,

$$\mathbb{E}_{p(\mathbf{x})}[\|\nabla_{\mathbf{x}} \log p(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x})\|_2^2]$$

- We usually can't access the left hand term, but techniques for training despite this
- Inference: special methods that can sample, like Langevin dynamics

$$\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \sqrt{2\epsilon}\, \mathbf{z}_i$$

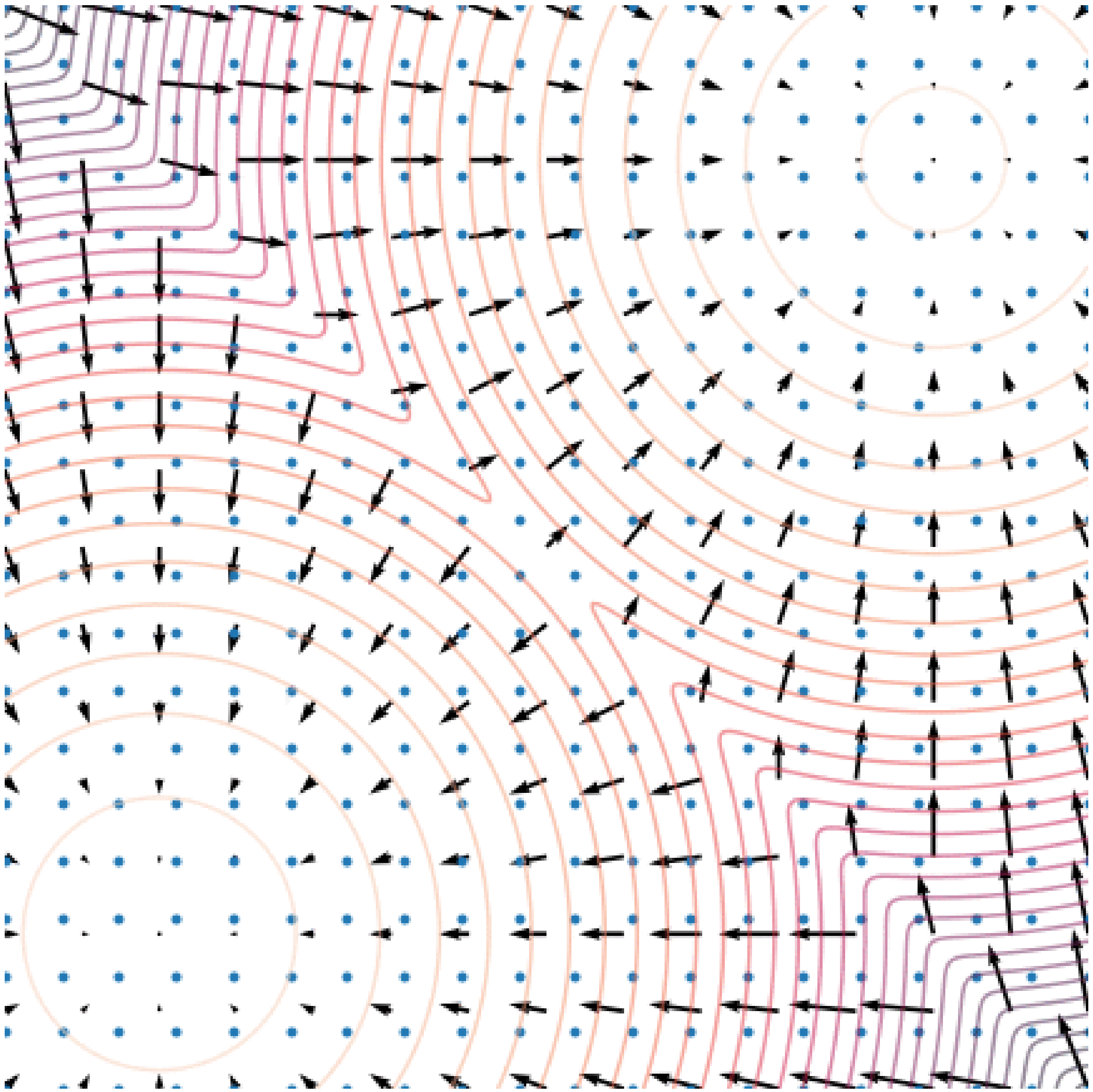
Sample Iterates

Learned score function

Noise

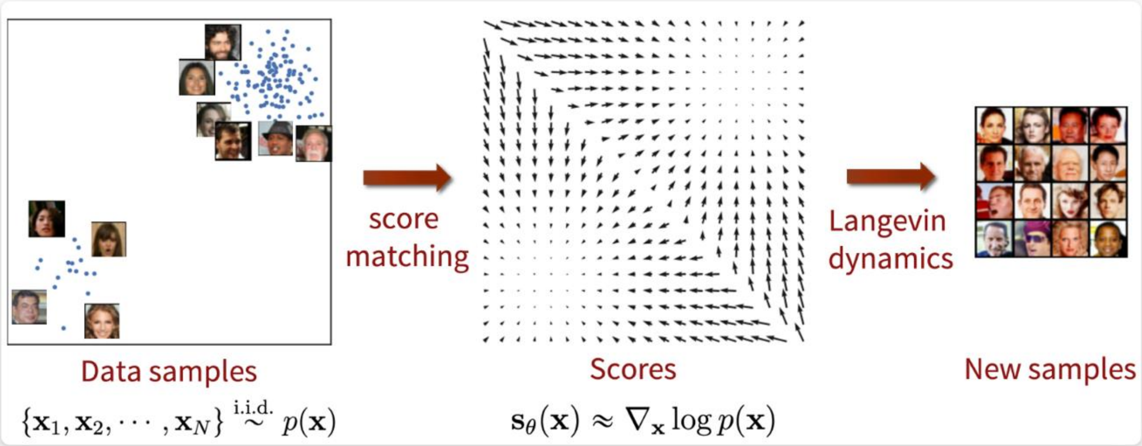
# Training & Inference for Score-Based Models

- **Visual example**
  - Distribution: mixture of two Gaussians
  - Arrows: given by our score function, point to high density regions
  - Source: https://yang-song.net/blog/2021/score/

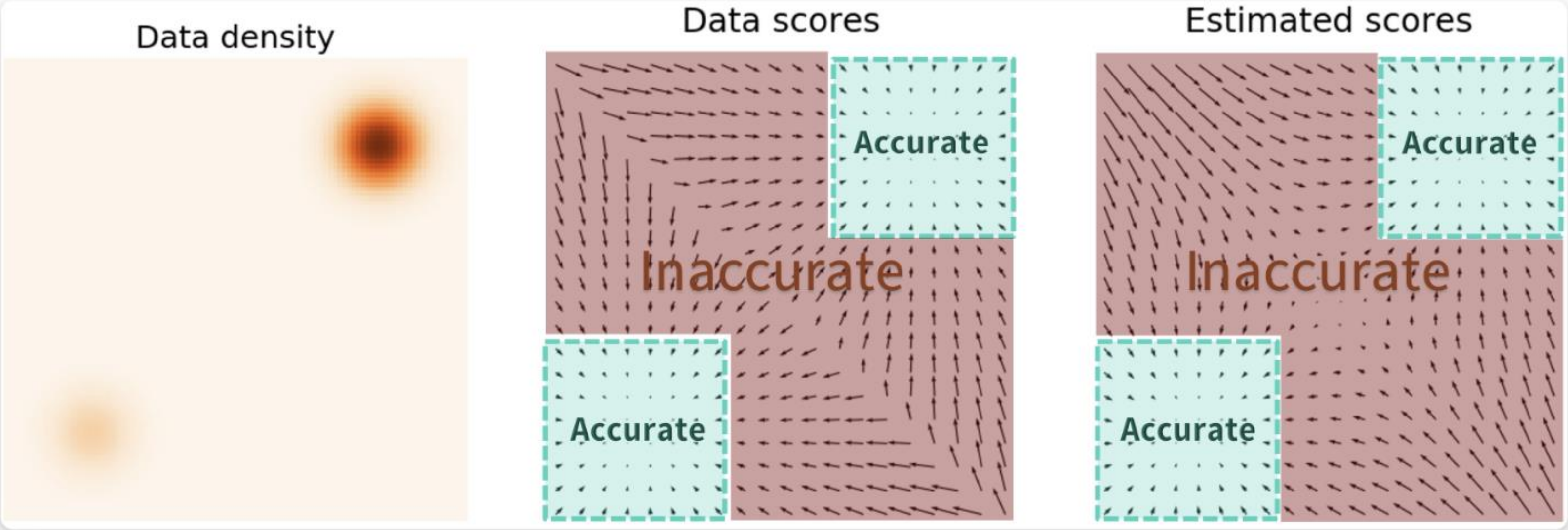


# Score-Based → Denoising Diffusion Models

- Our story so far is

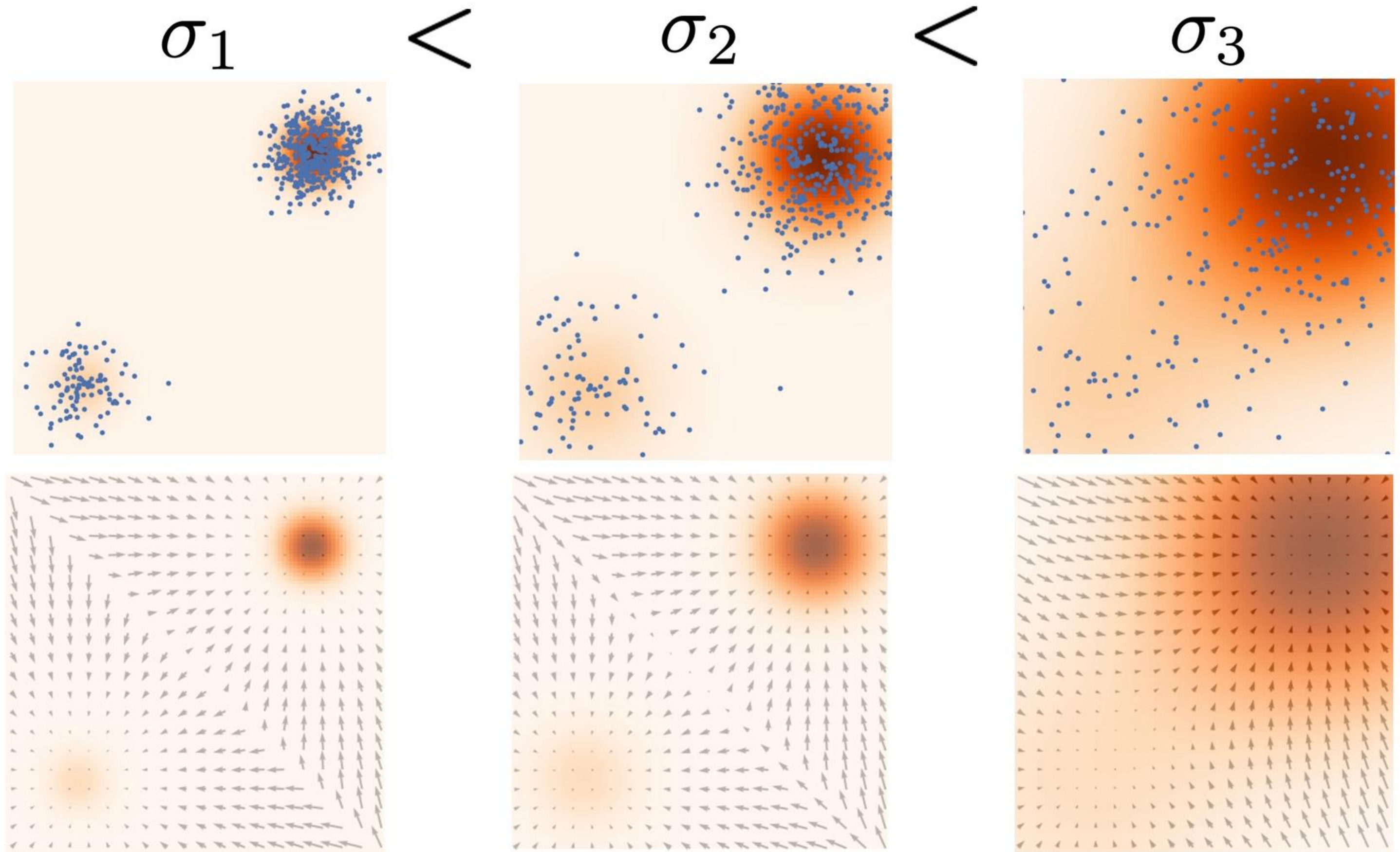



- But, this leads to inaccurate modeling in low-prob regions:



# Score-Based → Denoising Diffusion Models

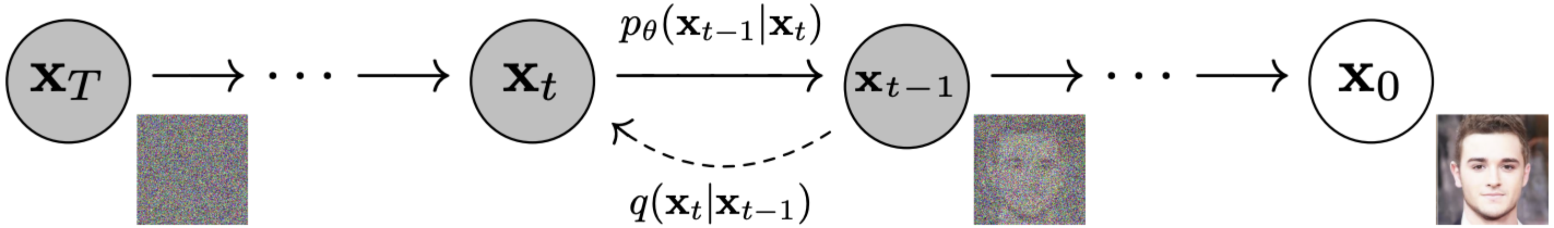
- Solution: perturb the density with noise
  - To ensure accurate modeling in more regions
  - In particular, noise at multiple scales



# Score-Based → Denoising Diffusion Models

- So far, "noise" showed up in a few places, but not in a strictly connected way
  - Train model with score matching
  - Sample with Lagenvin dynamics (which includes noise)
  - Use noise perturbation to train better

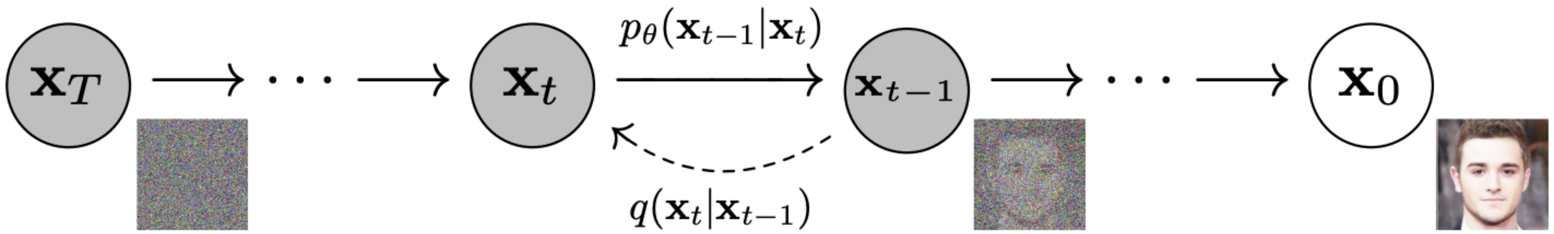
- Denoising diffusion models **directly** use noise in both training and inference




Ho et al '20

# Diffusion Models

- Basic graphical model




Ho et al ‘20

- Can easily set up the noising process,

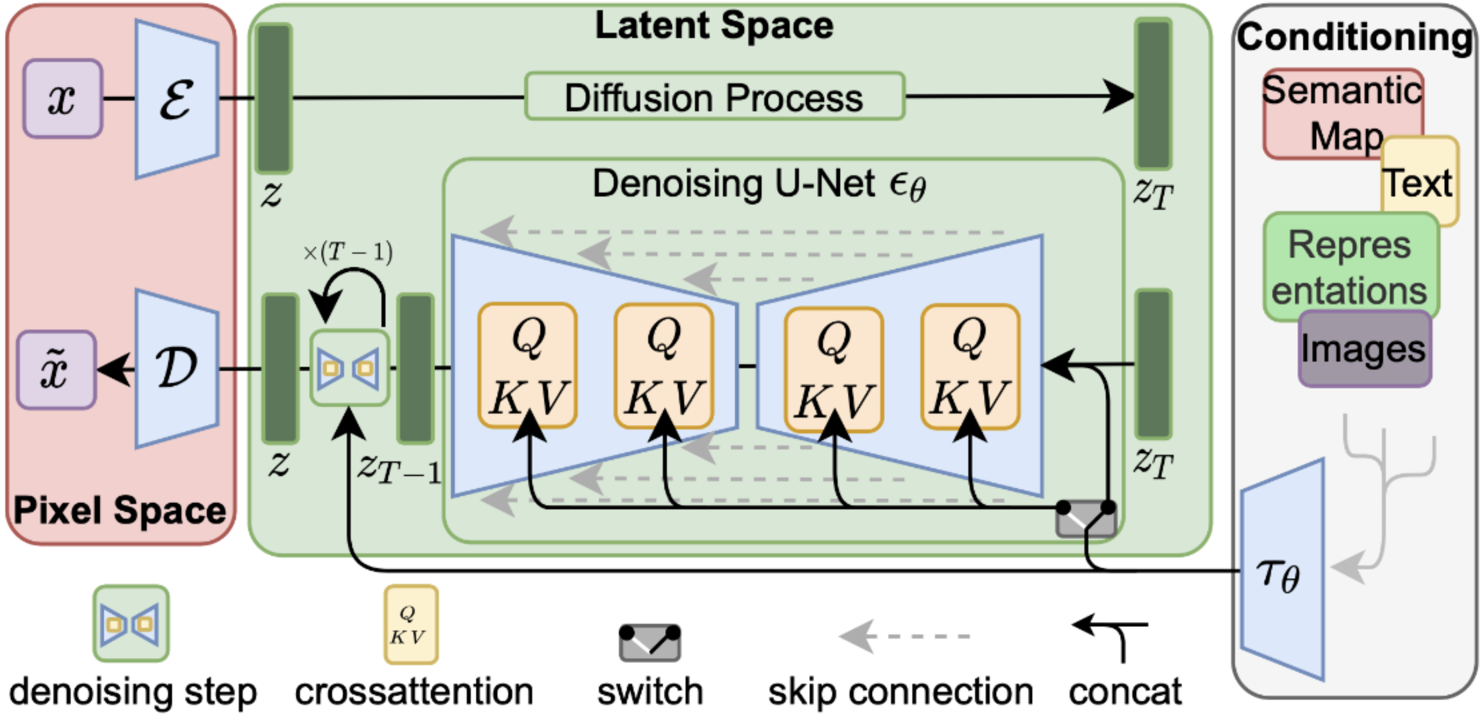
$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}) \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

- To sample, directly compute from reverse, i.e., $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$
  - Simple, nice parametrizations in Ho et al ‘20.

# Latent Diffusion Models

Latents are really just the noised images in pixel space

- No ”latent space” so far at least
- But, can add by using an autoencoder




Rombach et al ‘22

# Text-to-Image Generation + Conditional DMs

Lots of approaches! In particular, for text-to-image generation

- All based on similar principles from multimodal training

- Example: for latent diffusion (Rombach et al '22)
  - "Process y from various modalities (such as language prompts) we introduce a domain specific encoder … that projects y to an intermediate representation … which is then mapped to the intermediate layers of the UNet via a cross-attention layer "

# Bibliography


- https://lilianweng.github.io/tags/generative-model/
- https://lilianweng.github.io/posts/2018-10-13-flow-models/
- https://lilianweng.github.io/posts/2021-07-11-diffusion-models
- https://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture11.pdf
- Radford et al '16: Alec Radford, Luke Metz, Soumith Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks" (https://arxiv.org/abs/1511.06434)
- https://yang-song.net/blog/2021/score/
- Song et al '20: Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, Ben Poole , "Score-Based Generative Modeling through Stochastic Differential Equations" (https://arxiv.org/abs/2011.13456)
- Ho et al '20: Jonathan Ho, Ajay Jain, Pieter Abbeel, "Denoising Diffusion Probabilistic Models", (https://arxiv.org/abs/2006.11239)
- Rombach et al '22: Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, Björn Ommer "High-Resolution Image Synthesis with Latent Diffusion Models" (https://arxiv.org/abs/2112.10752)

# Thank You!