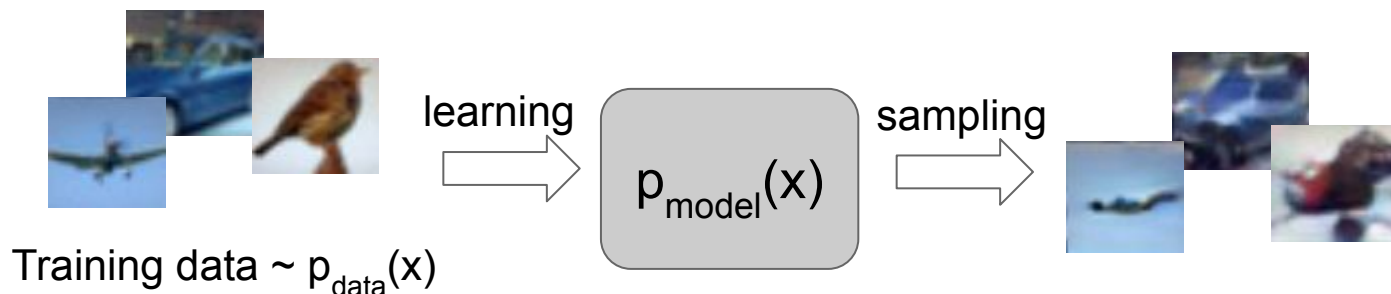


# Generative Modeling

Given training data, generate new samples from same distribution

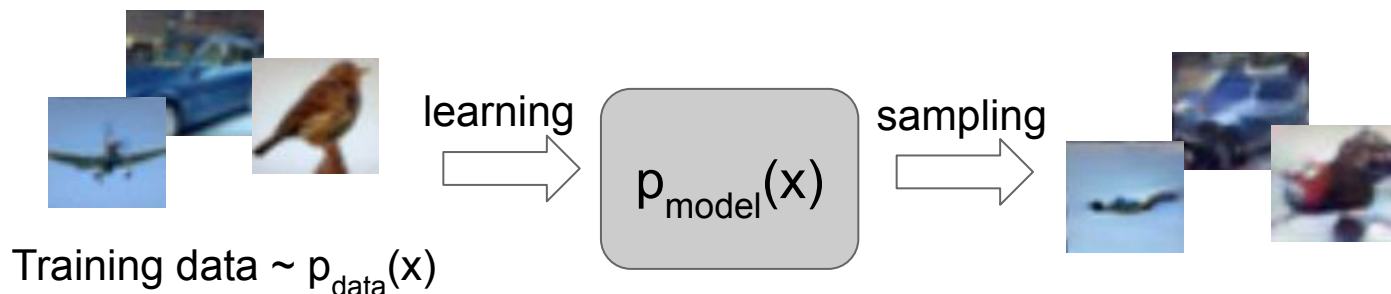


Objectives:

1. Learn  $p_{\text{model}}(x)$  that approximates  $p_{\text{data}}(x)$
2. **Sampling new  $x$  from  $p_{\text{model}}(x)$**

# Generative Modeling

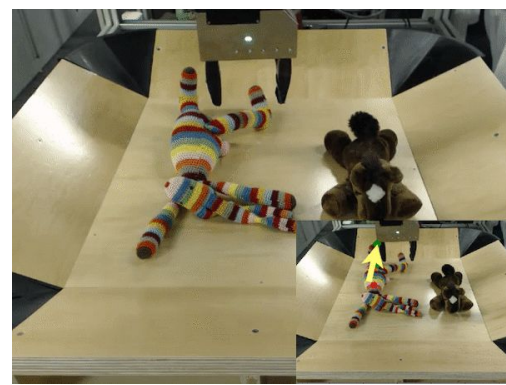
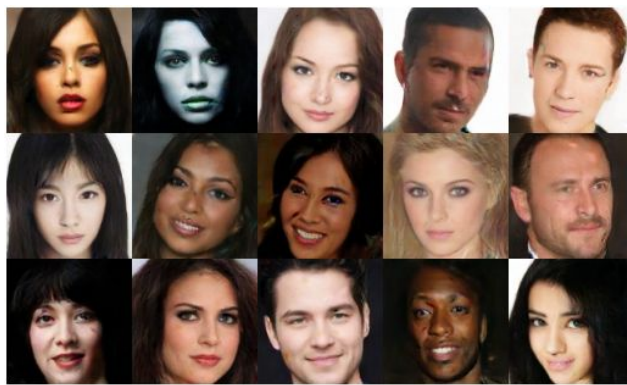
Given training data, generate new samples from same distribution



Formulate as density estimation problems:

- **Explicit density estimation:** explicitly define and solve for  $p_{\text{model}}(x)$
- **Implicit density estimation:** learn model that can sample from  $p_{\text{model}}(x)$  **without explicitly defining it.**

# Why Generative Models?



- Realistic samples for artwork, super-resolution, colorization, etc.
- Learn useful features for downstream tasks such as classification.
- Getting insights from high-dimensional data (physics, medical imaging, etc.)
- Modeling physical world for simulation and planning (robotics and reinforcement learning applications)
- Many more ...

Figures from L-R are copyright: (1) [Alec Radford et al. 2016](#); (2) [Phillip Isola et al. 2017](#). Reproduced with authors permission (3) [BAIR Blog](#).

# Taxonomy of Generative Models

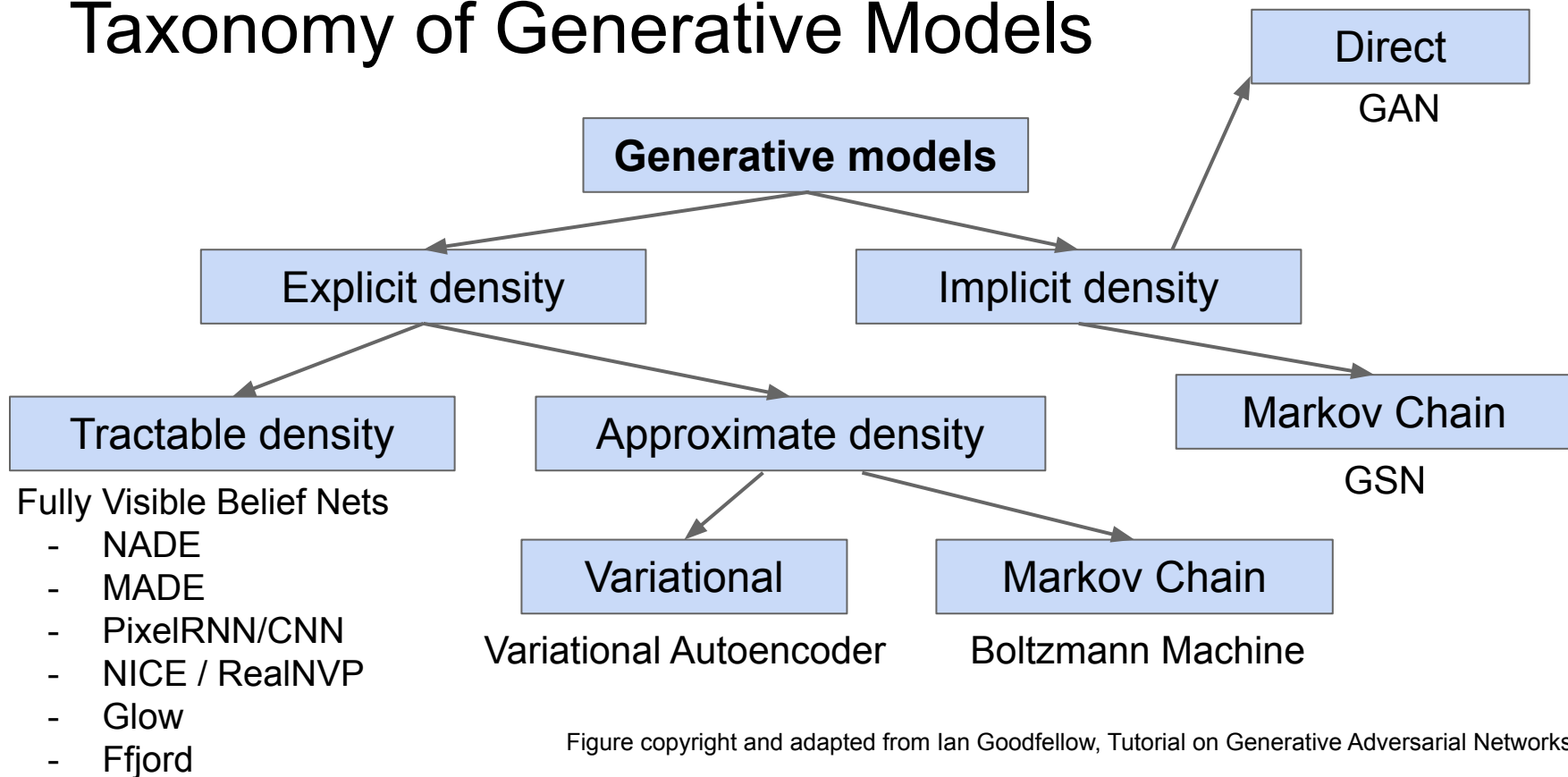


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

# Taxonomy of Generative Models

Today: discuss 3 most popular types of generative models today

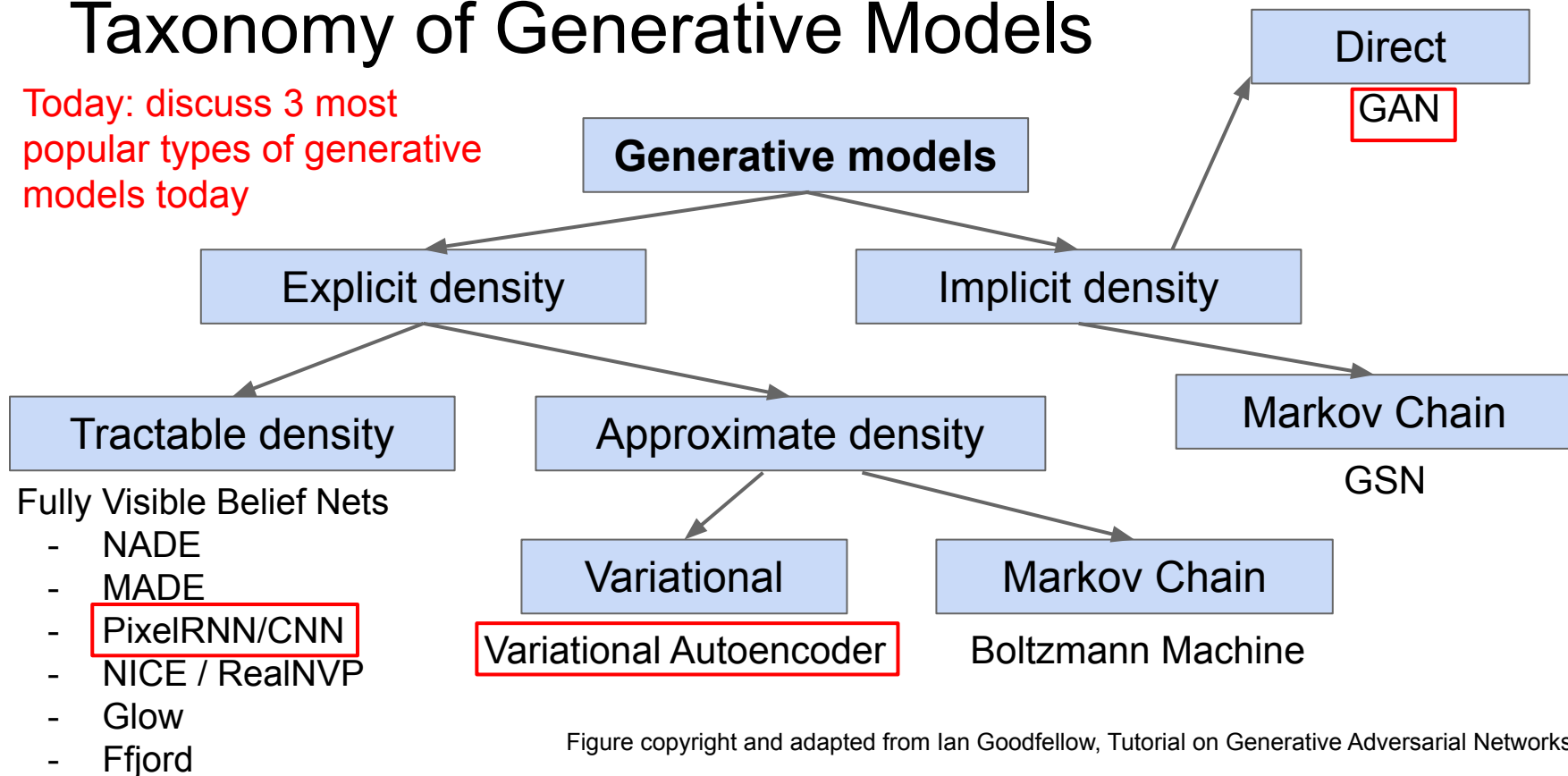


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

# PixelRNN and PixelCNN

(A very brief overview)

# Fully visible belief network (FVBN)

Explicit density model

$$p(x) = p(x_1, x_2, \dots, x_n)$$

↑  
Likelihood of  
image  $x$

↑  
Joint likelihood of each  
pixel in the image





# Fully visible belief network (FVBN)

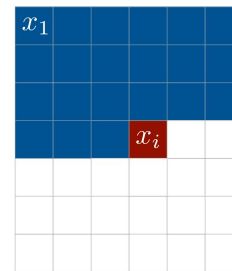
Explicit density model

Use chain rule to decompose likelihood of an image  $x$  into product of 1-d distributions:

$$p(x) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1})$$

↑  
Likelihood of  
image  $x$

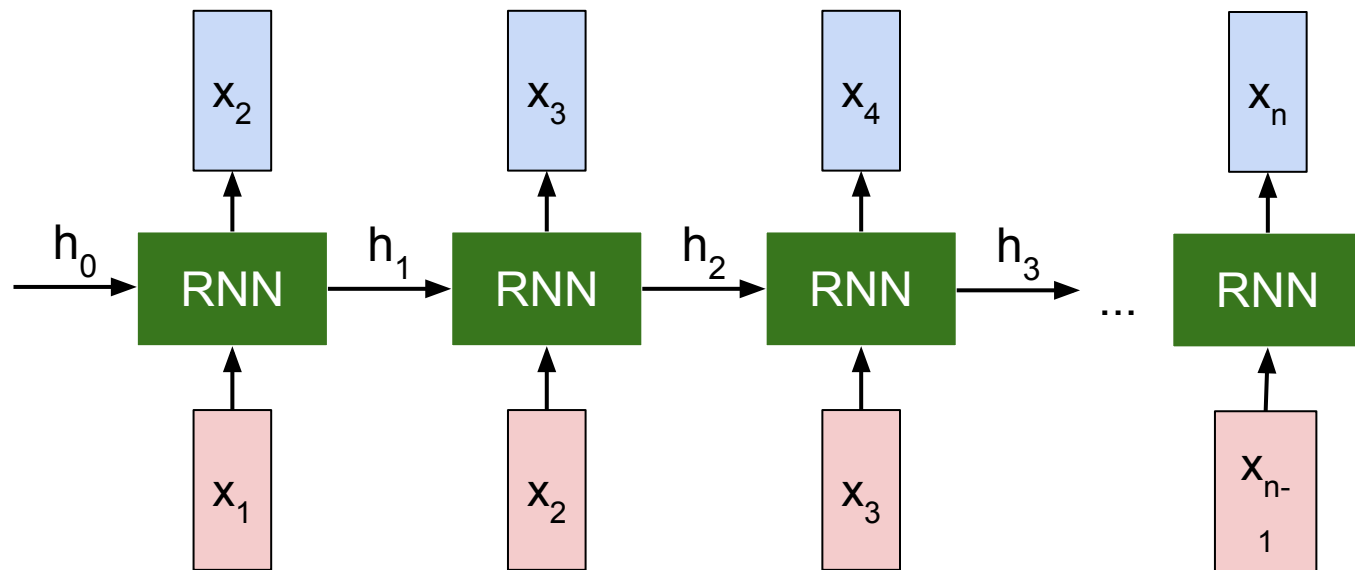
↑  
Probability of  $i$ 'th pixel value  
given all previous pixels



Complex distribution over pixel  
values => Express using a neural  
network!

Then maximize likelihood of training data

# Recurrent Neural Network

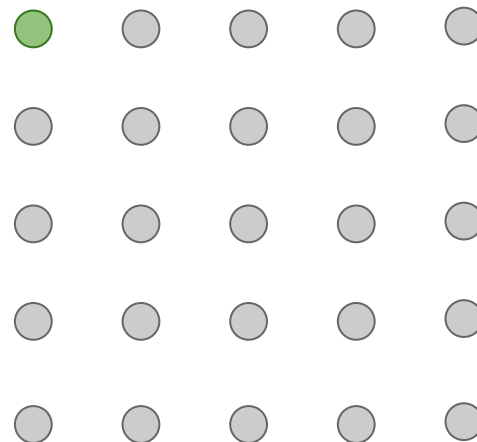


$$p(x_i | x_1, \dots, x_{i-1})$$

# PixelRNN *[van der Oord et al. 2016]*

Generate image pixels starting from corner

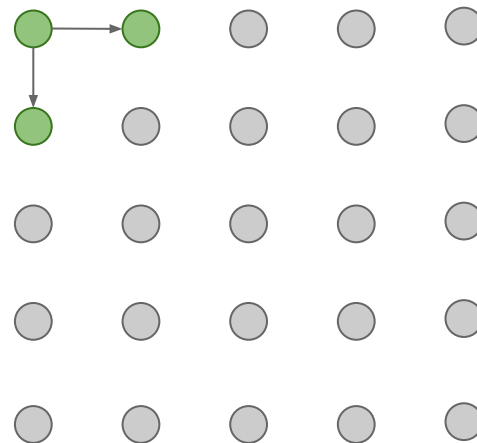
Dependency on previous pixels modeled  
using an RNN (LSTM)



# PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

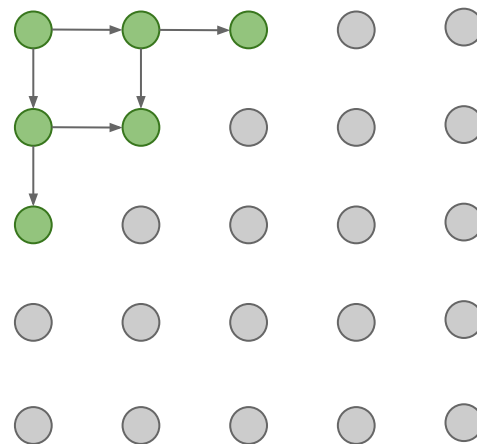
Dependency on previous pixels modeled using an RNN (LSTM)



# PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

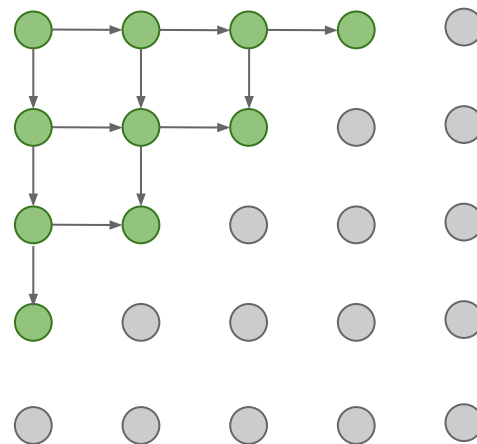


# PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow in both training and inference!



# PixelCNN *[van der Oord et al. 2016]*

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region (**masked convolution**)

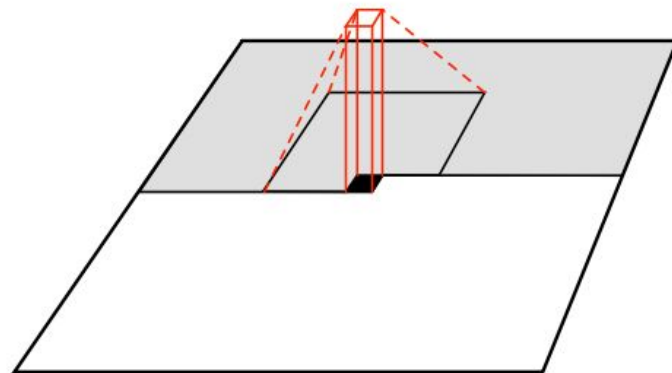


Figure copyright van der Oord et al., 2016. Reproduced with permission.

# PixelCNN *[van der Oord et al. 2016]*

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region (masked convolution)

Training is faster than PixelRNN  
(can parallelize convolutions since context region values known from training images)

Generation is still slow:

For a 32x32 image, we need to do forward passes of the network 1024 times for a single image

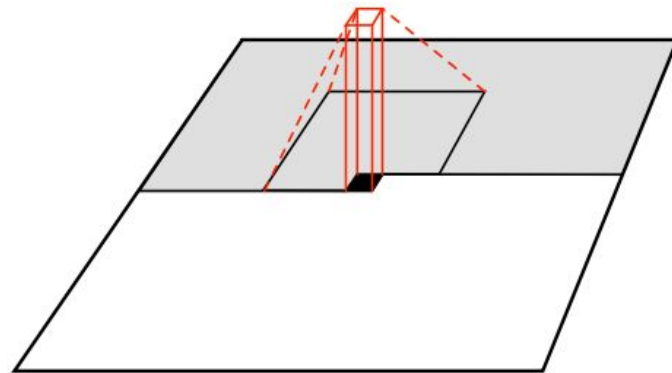
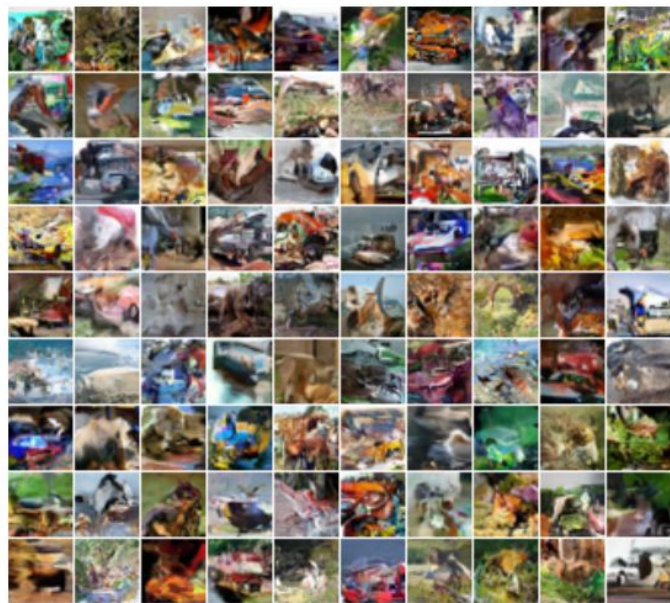


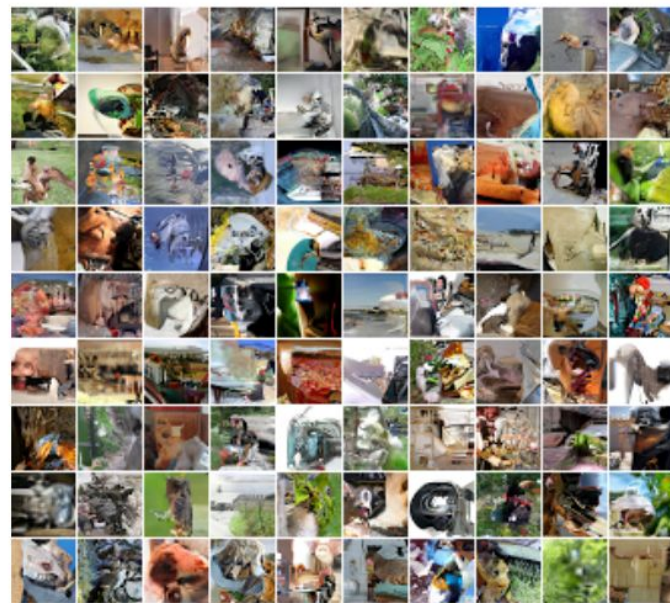
Figure copyright van der Oord et al., 2016. Reproduced with permission.



# Generation Samples



32x32 CIFAR-10



32x32 ImageNet

Figures copyright Aaron van der Oord et al., 2016. Reproduced with permission.

# PixelRNN and PixelCNN

## Pros:

- Can explicitly compute likelihood  $p(x)$
- Easy to optimize
- Good samples

## Con:

- Sequential generation => slow

## Improving PixelCNN performance

- Gated convolutional layers
- Short-cut connections
- Discretized logistic loss
- Multi-scale
- Training tricks
- Etc...

## See

- Van der Oord et al. NIPS 2016
- Salimans et al. 2017 (PixelCNN++)

# Taxonomy of Generative Models

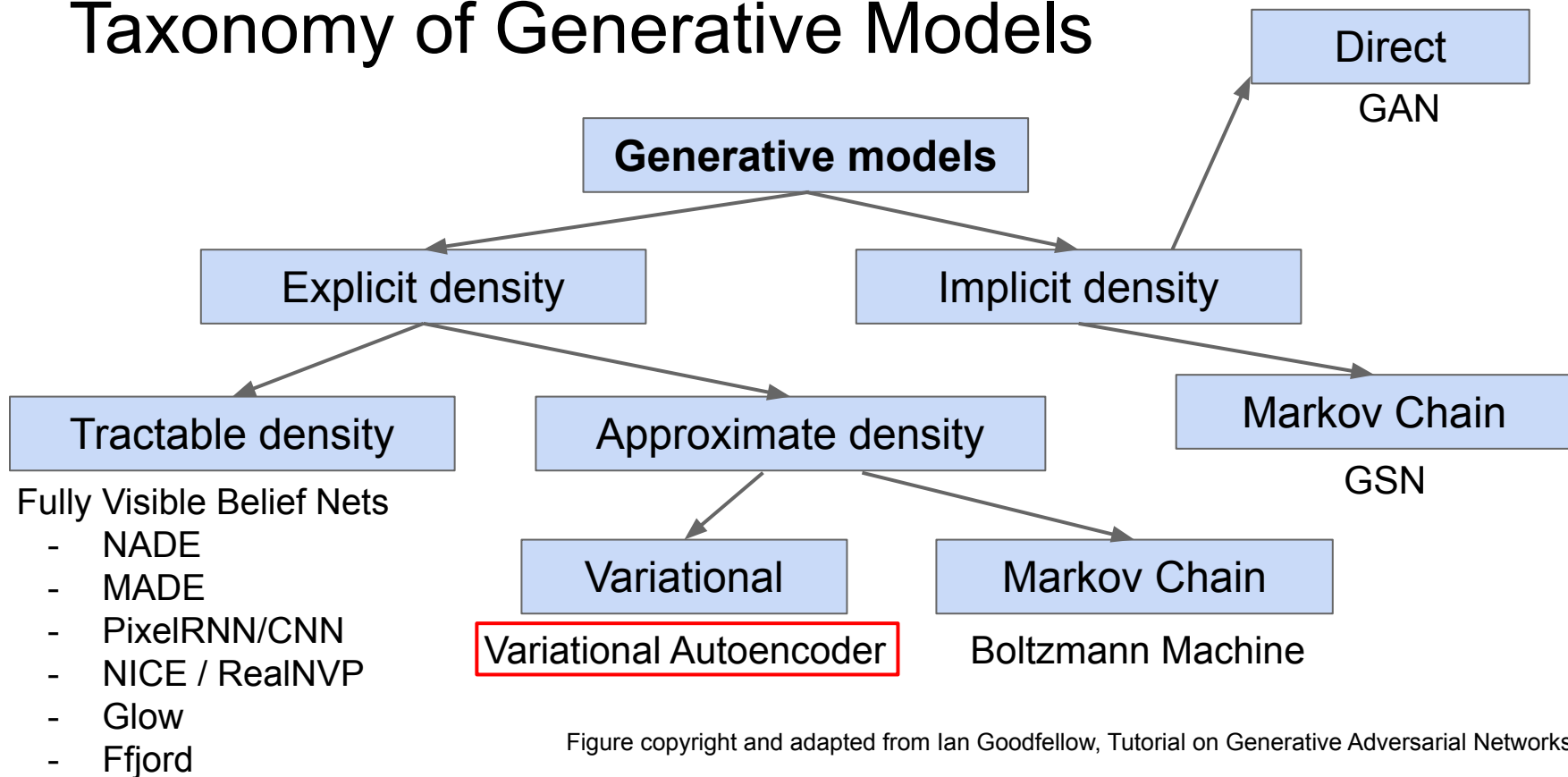


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

# Variational Autoencoders (VAE)

# So far...

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i | x_1, \dots, x_{i-1})$$

# So far...

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i | x_1, \dots, x_{i-1})$$

Variational Autoencoders (VAEs) define intractable density function with latent  $\mathbf{z}$ :

$$p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$$

No dependencies among pixels, can generate all pixels at the same time!

Cannot optimize directly, derive and optimize lower bound on likelihood instead

# So far...

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i | x_1, \dots, x_{i-1})$$

Variational Autoencoders (VAEs) define intractable density function with latent  $\mathbf{z}$ :

$$p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$$

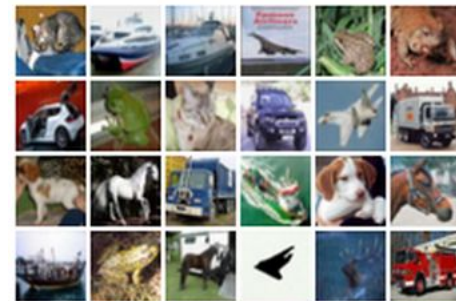
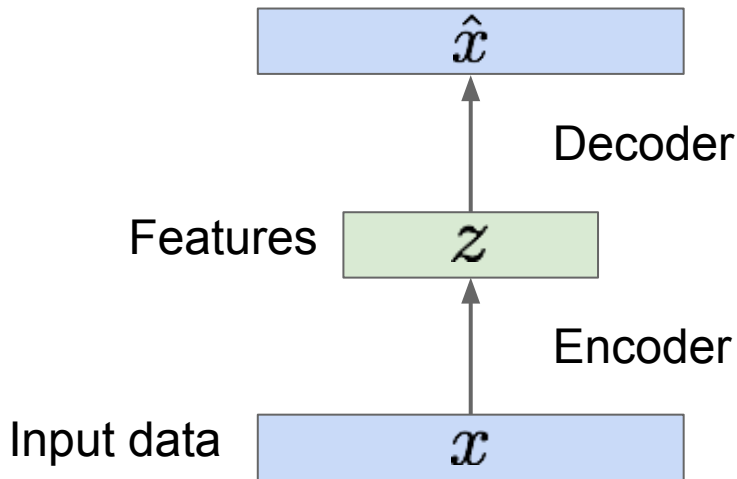
No dependencies among pixels, can generate all pixels at the same time!

Cannot optimize directly, derive and optimize lower bound on likelihood instead

Why latent  $\mathbf{z}$ ?

# Some background first: Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data



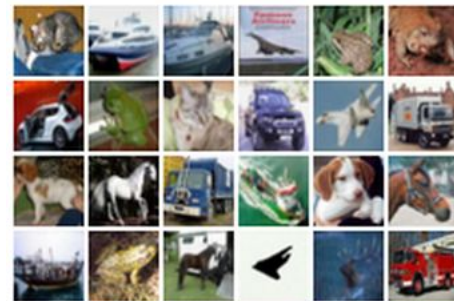
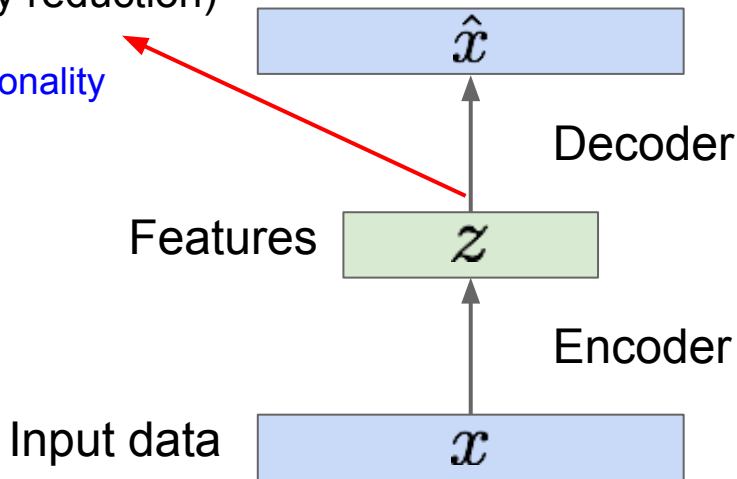


# Some background first: Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

$\mathbf{z}$  usually smaller than  $\mathbf{x}$   
(dimensionality reduction)

Q: Why dimensionality reduction?



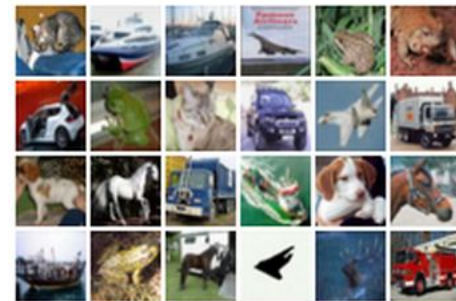
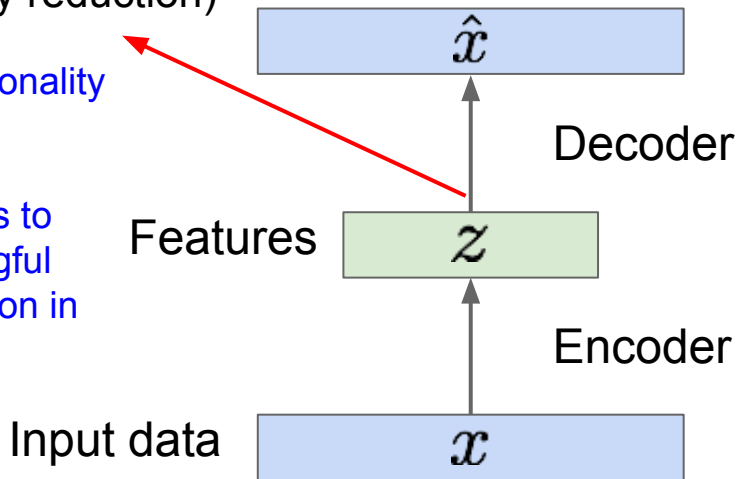
# Some background first: Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

$z$  usually smaller than  $x$   
(dimensionality reduction)

Q: Why dimensionality reduction?

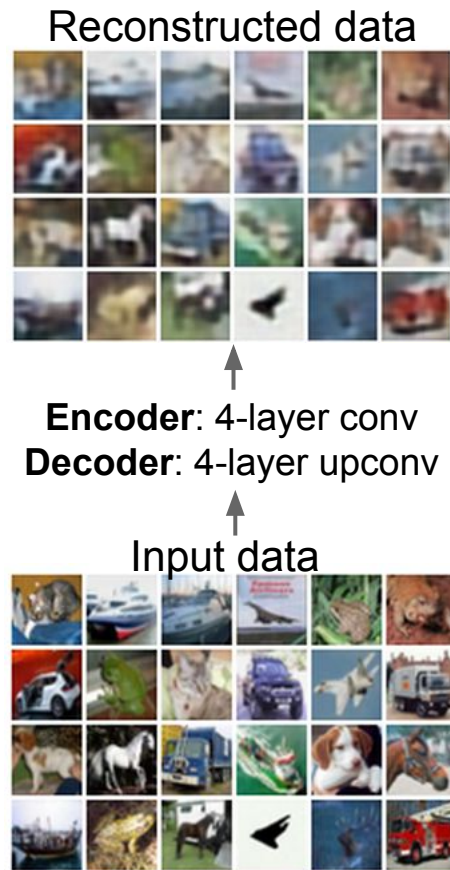
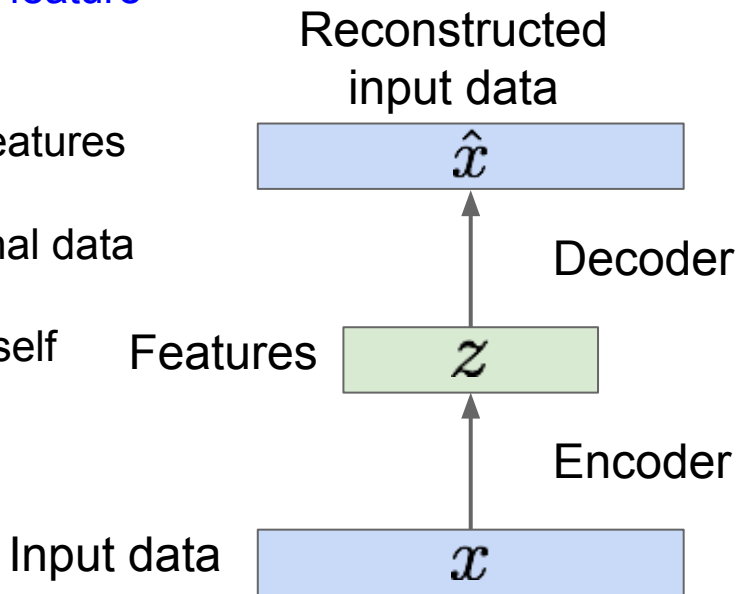
A: Want features to capture meaningful factors of variation in data



# Some background first: Autoencoders

How to learn this feature representation?

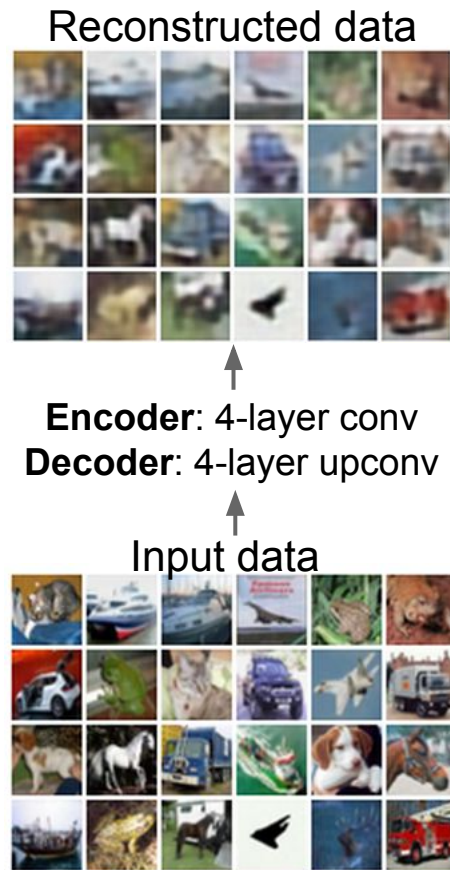
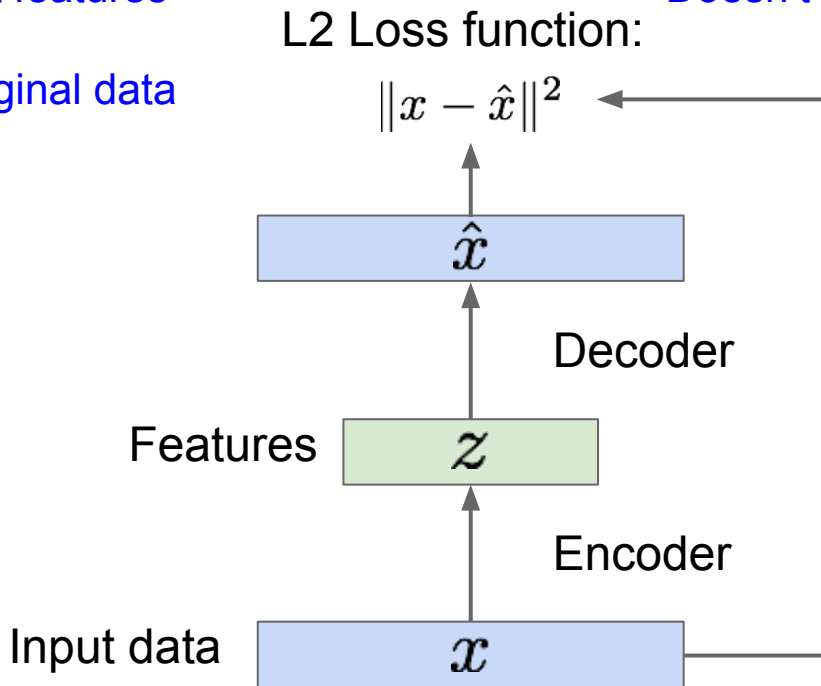
Train such that features can be used to reconstruct original data  
“Autoencoding” - encoding input itself



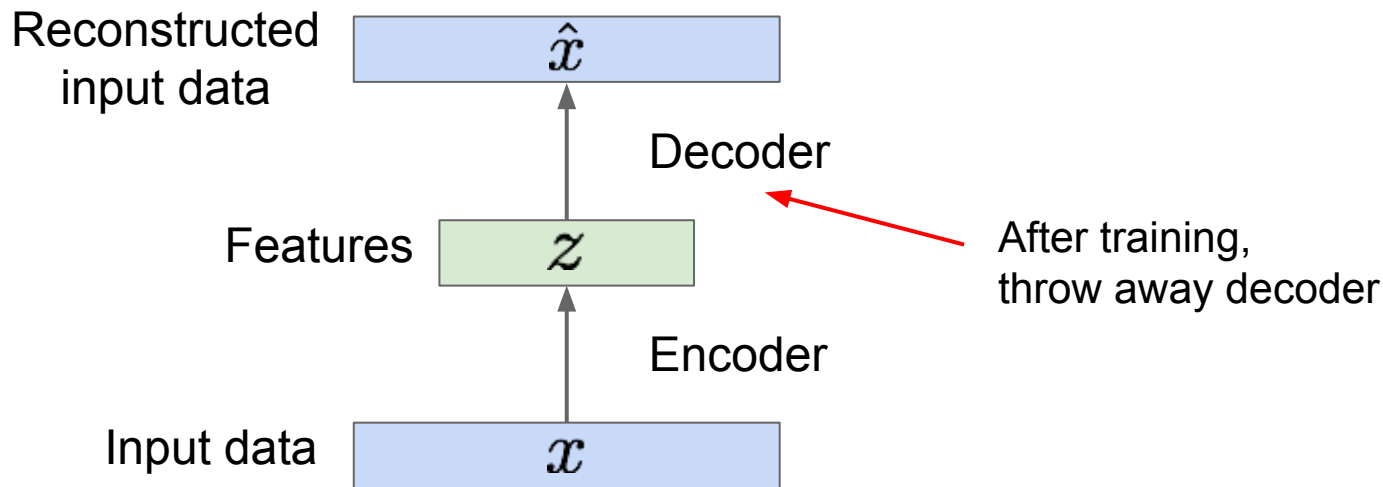
# Some background first: Autoencoders

Train such that features can be used to reconstruct original data

Doesn't use labels!

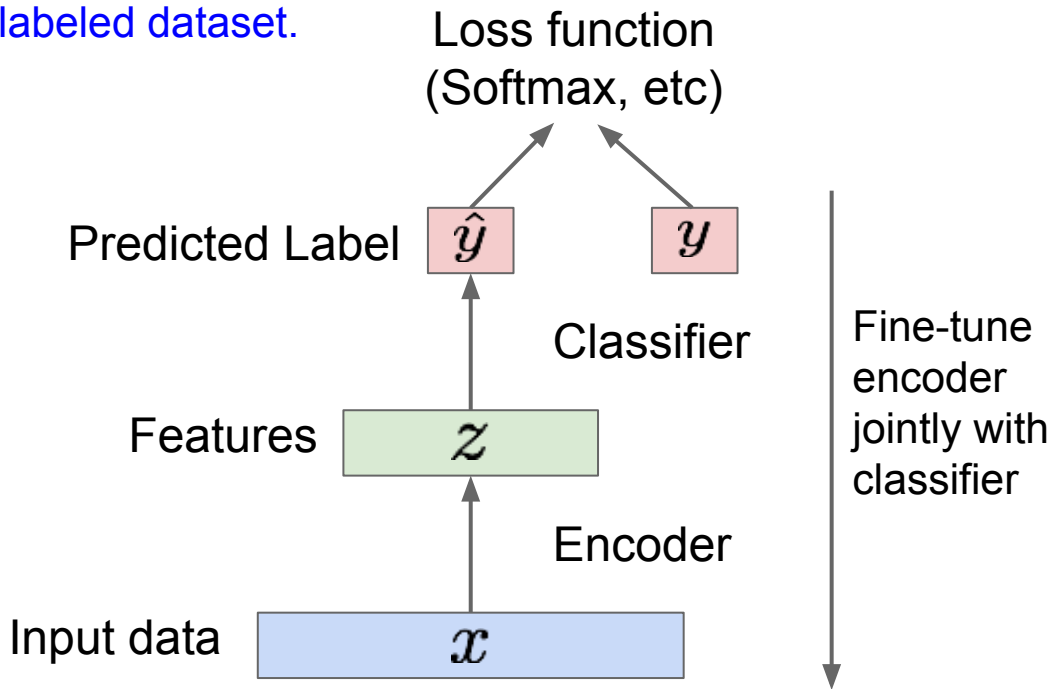


# Some background first: Autoencoders



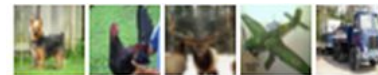
# Some background first: Autoencoders

Transfer from large, unlabeled dataset to small, labeled dataset.



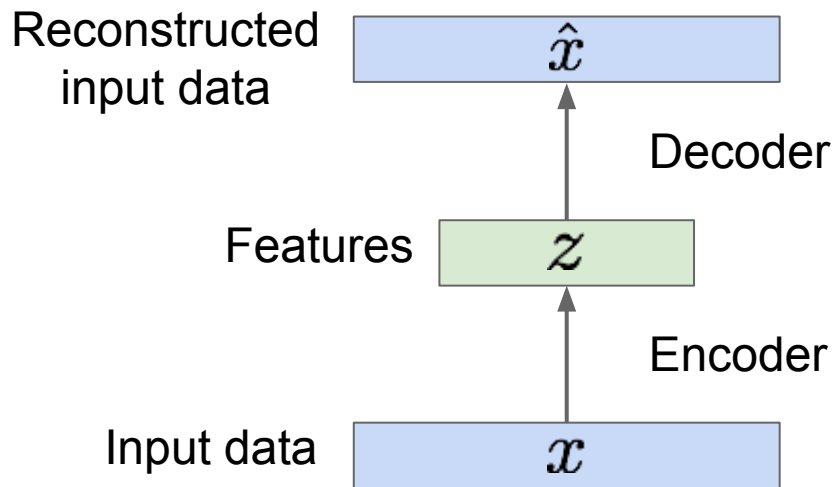
bird plane  
dog deer truck

Train for final task (sometimes with small data)



Encoder can be used to initialize a **supervised** model

# Some background first: Autoencoders



Autoencoders can reconstruct data, and can learn features to initialize a supervised model

Features capture factors of variation in training data.

But we can't generate new images from an autoencoder because we don't know the space of  $z$ .

How do we make autoencoder a **generative model**?

# Variational Autoencoders

Probabilistic spin on autoencoders - will let us sample from the model to generate data!



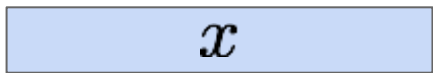
# Variational Autoencoders

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

Assume training data  $\{x^{(i)}\}_{i=1}^N$  is generated from the distribution of unobserved (latent) representation  $\mathbf{z}$

Sample from  
true conditional

$$p_{\theta^*}(x | z^{(i)})$$



Sample from  
true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

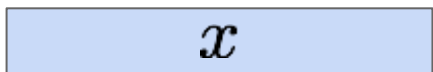
# Variational Autoencoders

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

Assume training data  $\{x^{(i)}\}_{i=1}^N$  is generated from the distribution of unobserved (latent) representation  $\mathbf{z}$

Sample from  
true conditional

$$p_{\theta^*}(x | z^{(i)})$$



Sample from  
true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$

**Intuition** (remember from autoencoders!):  
**x** is an image, **z** is latent factors used to generate **x**: attributes, orientation, etc.

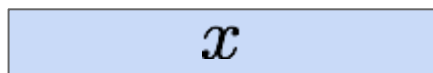
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders

We want to estimate the true parameters  $\theta^*$  of this generative model given training data  $x$ .

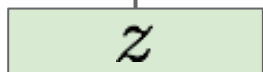
Sample from  
true conditional

$$p_{\theta^*}(x | z^{(i)})$$



Sample from  
true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



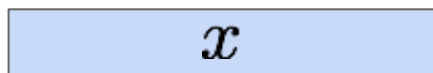
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders

We want to estimate the true parameters  $\theta^*$  of this generative model given training data  $x$ .

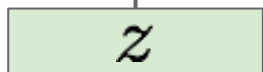
Sample from  
true conditional

$$p_{\theta^*}(x | z^{(i)})$$



Sample from  
true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



How should we represent this model?

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

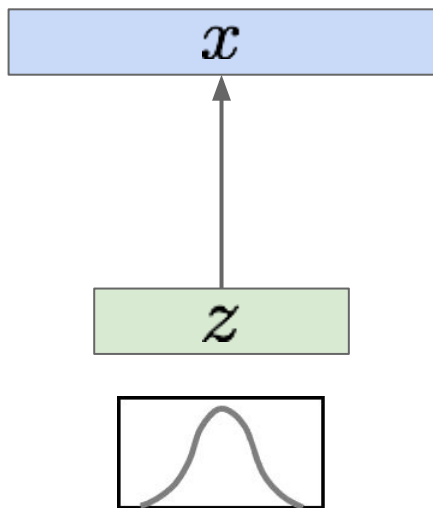
# Variational Autoencoders

Sample from  
true conditional

$$p_{\theta^*}(x | z^{(i)})$$

Sample from  
true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



We want to estimate the true parameters  $\theta^*$  of this generative model given training data  $x$ .

How should we represent this model?

Choose prior  $p(z)$  to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

# Variational Autoencoders

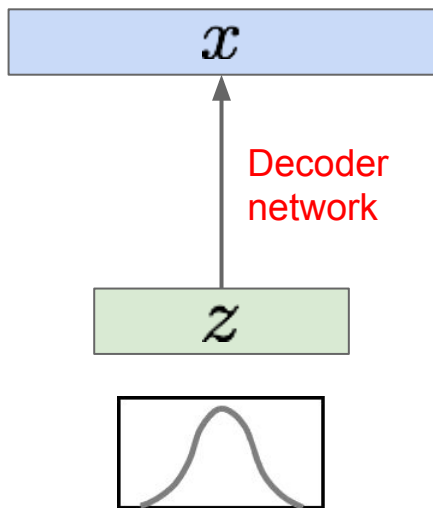


Sample from  
true conditional

$$p_{\theta^*}(x | z^{(i)})$$

Sample from  
true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



We want to estimate the true parameters  $\theta^*$  of this generative model given training data  $x$ .

How should we represent this model?

Choose prior  $p(z)$  to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.

Conditional  $p(x|z)$  is complex (generates image) => represent with neural network

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders

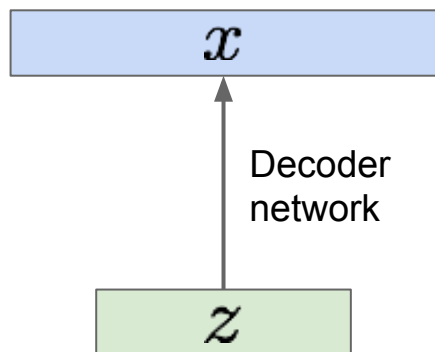
We want to estimate the true parameters  $\theta^*$  of this generative model given training data  $x$ .

Sample from true conditional

$$p_{\theta^*}(x | z^{(i)})$$

Sample from true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



How to train the model?

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

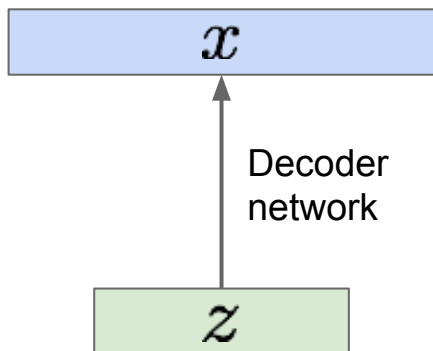
# Variational Autoencoders

Sample from  
true conditional

$$p_{\theta^*}(x | z^{(i)})$$

Sample from  
true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



We want to estimate the true parameters  $\theta^*$  of this generative model given training data  $x$ .

How to train the model?

Learn model parameters to maximize likelihood of training data

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014



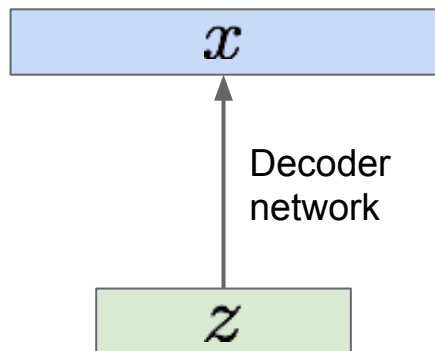
# Variational Autoencoders

Sample from  
true conditional

$$p_{\theta^*}(x | z^{(i)})$$

Sample from  
true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



We want to estimate the true parameters  $\theta^*$  of this generative model given training data  $x$ .

How to train the model?

Learn model parameters to maximize likelihood of training data

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

Q: What is the problem with this?

Intractable!

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders: Intractability

Data likelihood:  $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

# Variational Autoencoders: Intractability


Data likelihood:  $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

 Simple Gaussian prior

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

# Variational Autoencoders: Intractability

Data likelihood:  $p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$



  
Decoder neural network

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

# Variational Autoencoders: Intractability

Data likelihood:  $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

 Intractable to compute  $p(x|z)$  for every  $z$ !

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders: Intractability

Data likelihood:  $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

↑  
Intractable to compute  $p(x|z)$  for every  $z$ !

$$\log p(x) \approx \log \frac{1}{k} \sum_{i=1}^k p(x|z^{(i)}), \text{ where } z^{(i)} \sim p(z)$$

Monte Carlo estimation is too high variance

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders: Intractability

Data likelihood:  $p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$

Posterior density:  $p_{\theta}(z|x) = p_{\theta}(x|z) p_{\theta}(z) / p_{\theta}(x)$

Intractable data likelihood

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders: Intractability

Data likelihood:  $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

Posterior density also intractable:  $p_{\theta}(z|x) = p_{\theta}(x|z)p_{\theta}(z)/p_{\theta}(x)$

**Solution:** In addition to modeling  $p_{\theta}(x|z)$ , learn  $q_{\phi}(z|x)$  that approximates the true posterior  $p_{\theta}(z|x)$ .

Will see that the approximate posterior allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize.

**Variational inference** is to approximate the unknown posterior distribution from only the observed data  $x$

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014




# Variational Autoencoders

$$\log p_{\theta}(x^{(i)}) = \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)})) \text{ Does not depend on } z$$

# Variational Autoencoders

$$\log p_{\theta}(x^{(i)}) = \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z)$$

 Taking expectation wrt.  $z$   
(using encoder network) will  
come in handy later

# Variational Autoencoders

$$\begin{aligned}\log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] && (p_{\theta}(x^{(i)})) \text{ Does not depend on } z \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] && (\text{Bayes' Rule})\end{aligned}$$

# Variational Autoencoders

$$\begin{aligned}\log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] && (p_{\theta}(x^{(i)})) \text{ Does not depend on } z \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] && (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \frac{q_{\phi}(z | x^{(i)})}{q_{\phi}(z | x^{(i)})} \right] && (\text{Multiply by constant})\end{aligned}$$

# Variational Autoencoders

$$\begin{aligned}\log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] && (p_{\theta}(x^{(i)})) \text{ Does not depend on } z \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] && (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \frac{q_{\phi}(z | x^{(i)})}{q_{\phi}(z | x^{(i)})} \right] && (\text{Multiply by constant}) \\ &= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] && (\text{Logarithms})\end{aligned}$$

# Variational Autoencoders

$$\begin{aligned}\log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z) \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant}) \\ &= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Logarithms}) \\ &= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z)) + D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))\end{aligned}$$

←      ↗  
The expectation wrt.  $z$  (using  
encoder network) let us write  
nice KL terms

# Variational Autoencoders

$$\begin{aligned}\log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z) \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant}) \\ &= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Logarithms}) \\ &= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z)) + D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))\end{aligned}$$

↑  
Decoder network gives  $p_{\theta}(x|z)$ , can compute estimate of this term through sampling (need some trick to differentiate through sampling).

↑  
This KL term (between Gaussians for encoder and  $z$  prior) has nice closed-form solution!

↑  
 $p_{\theta}(z|x)$  intractable (saw earlier), can't compute this KL term :( But we know KL divergence always  $\geq 0$ .

# Variational Autoencoders

$$\log p_{\theta}(x^{(i)}) = \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z)$$

We want to maximize the data likelihood

$$= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes' Rule})$$

$$= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant})$$

$$= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Logarithms})$$

$$= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z)) + D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))$$

Decoder network gives  $p_{\theta}(x|z)$ , can compute estimate of this term through sampling.

This KL term (between Gaussians for encoder and  $z$  prior) has nice closed-form solution!

$p_{\theta}(z|x)$  intractable (saw earlier), can't compute this KL term :( But we know KL divergence always  $\geq 0$ .



# Variational Autoencoders

We want to maximize the data likelihood

$$\begin{aligned}\log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z) \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant}) \\ &= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Logarithms}) \\ &= \underbrace{\mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))}_{\geq 0}\end{aligned}$$

**Tractable lower bound** which we can take gradient of and optimize! ( $p_{\theta}(x|z)$  differentiable, KL term differentiable)

# Variational Autoencoders

$$\log p_{\theta}(x^{(i)}) = \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes' Rule})$$

Decoder:  
reconstruct  
the input data

$$= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant})$$

Encoder:  
make approximate  
posterior distribution  
close to prior

$$= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Logarithms})$$

$$= \underbrace{\mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))}_{\geq 0}$$

**Tractable lower bound** which we can take  
gradient of and optimize! ( $p_{\theta}(x|z)$  differentiable,  
KL term differentiable)

# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[ \log p_\theta(x^{(i)} | z) \right]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))$$

Let's look at computing the KL divergence between the estimated posterior and the prior given some data

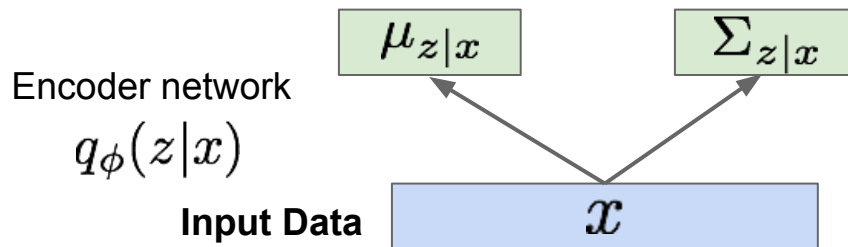
Input Data

$\mathcal{X}$

# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$



# Variational Autoencoders

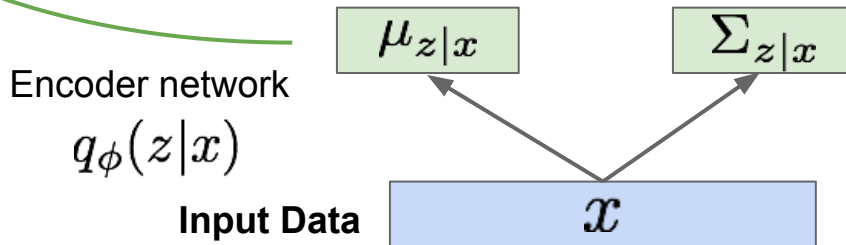
Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[ \log p_\theta(x^{(i)} | z) \right]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - \boxed{D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}$$

Make approximate posterior distribution close to prior

$$D_{KL}(\mathcal{N}(\mu_{z|x}, \Sigma_{z|x}) || \mathcal{N}(0, I))$$

Have analytical solution



# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

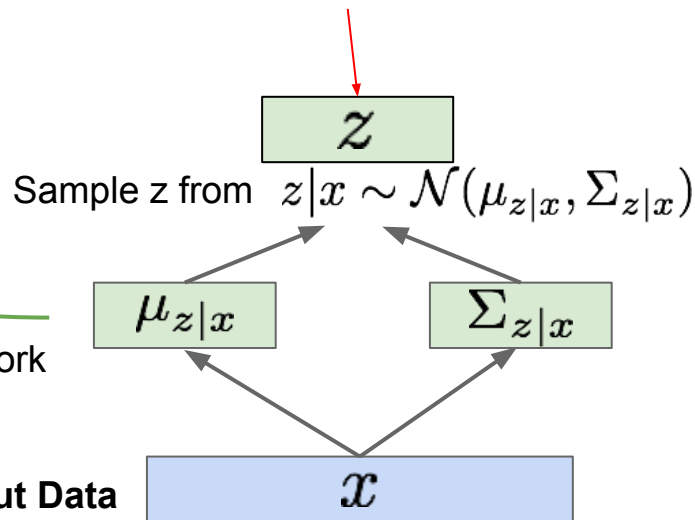
Make approximate posterior distribution close to prior

Encoder network

$$q_\phi(z|x)$$

Input Data

Not part of the computation graph!



# Variational Autoencoders

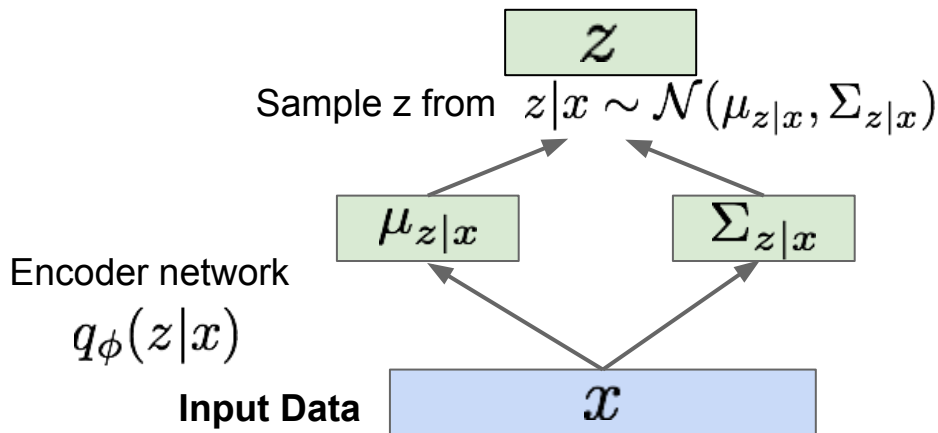
Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Reparameterization trick to make sampling differentiable:

$$\text{Sample } \epsilon \sim \mathcal{N}(0, I)$$

$$z = \mu_{z|x} + \epsilon \sigma_{z|x}$$





# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

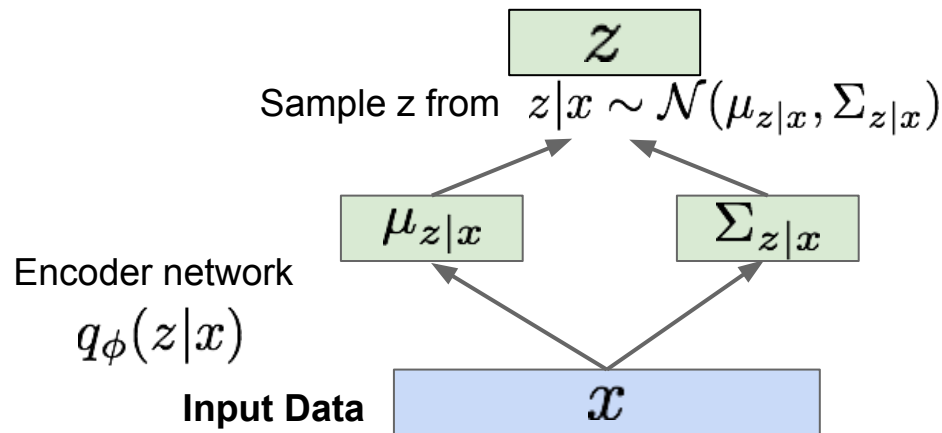
Reparameterization trick to make sampling differentiable:

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

$$z = \mu_{z|x} + \epsilon \sigma_{z|x}$$

Input to the graph

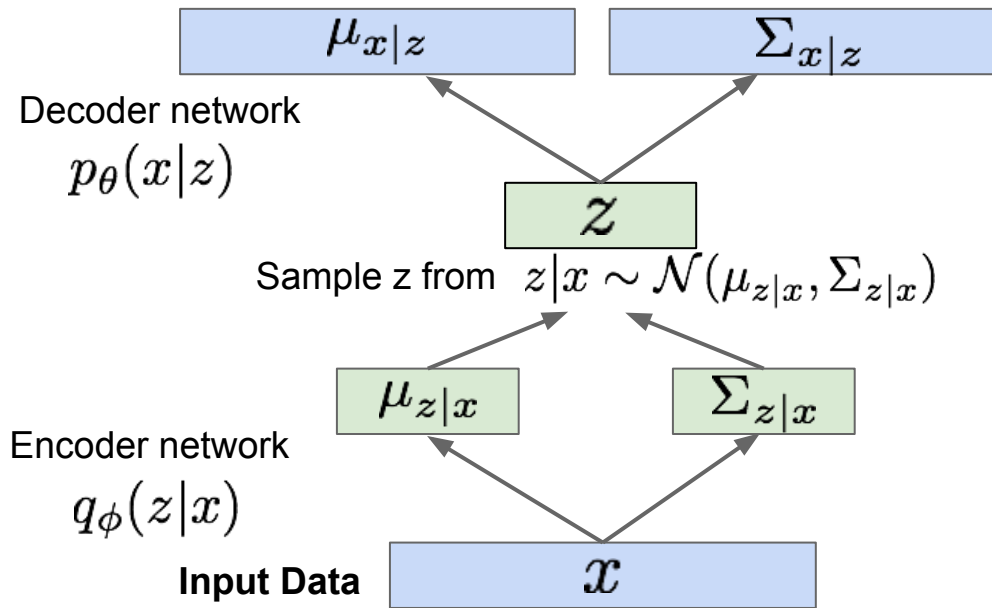
Part of computation graph



# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

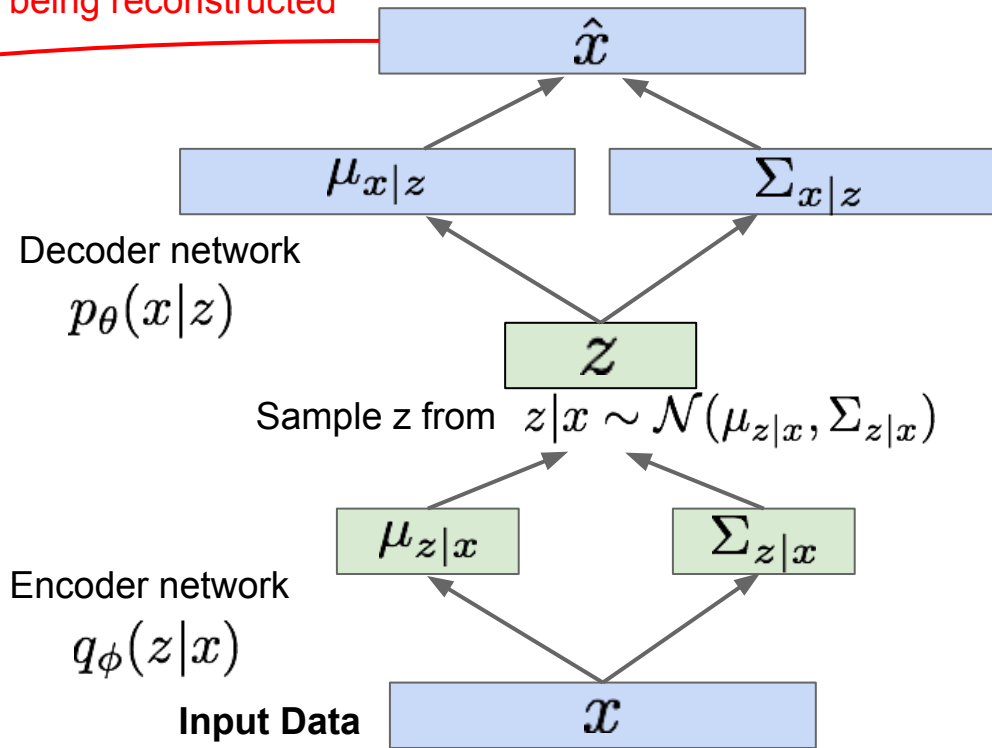


# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbb{E}_z \left[ \log p_\theta(x^{(i)} | z) \right]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))$$

Maximize likelihood of original input being reconstructed

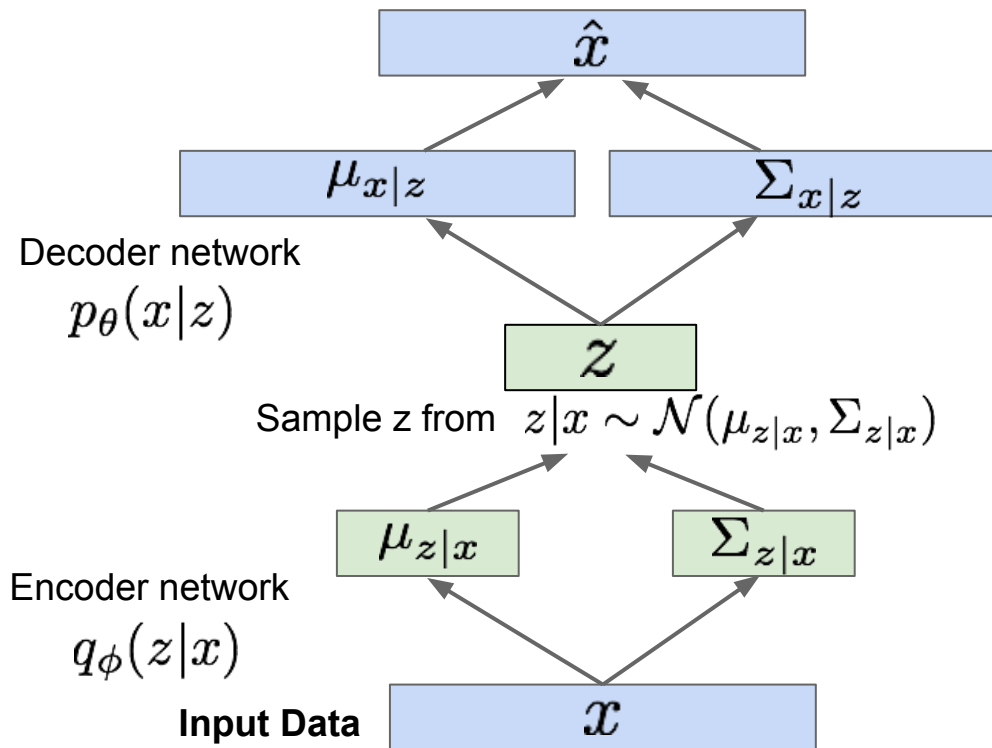


# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

For every minibatch of input data: compute this forward pass, and then backprop!

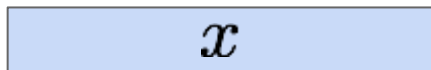


# Variational Autoencoders: Generating Data!

Our assumption about data generation process

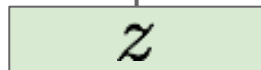
Sample from true conditional

$$p_{\theta^*}(x | z^{(i)})$$



Sample from true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



Decoder network



# Variational Autoencoders: Generating Data!

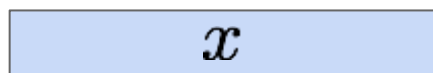
Our assumption about data generation process

Sample from true conditional

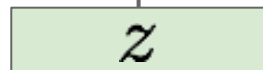
$$p_{\theta^*}(x | z^{(i)})$$

Sample from true prior

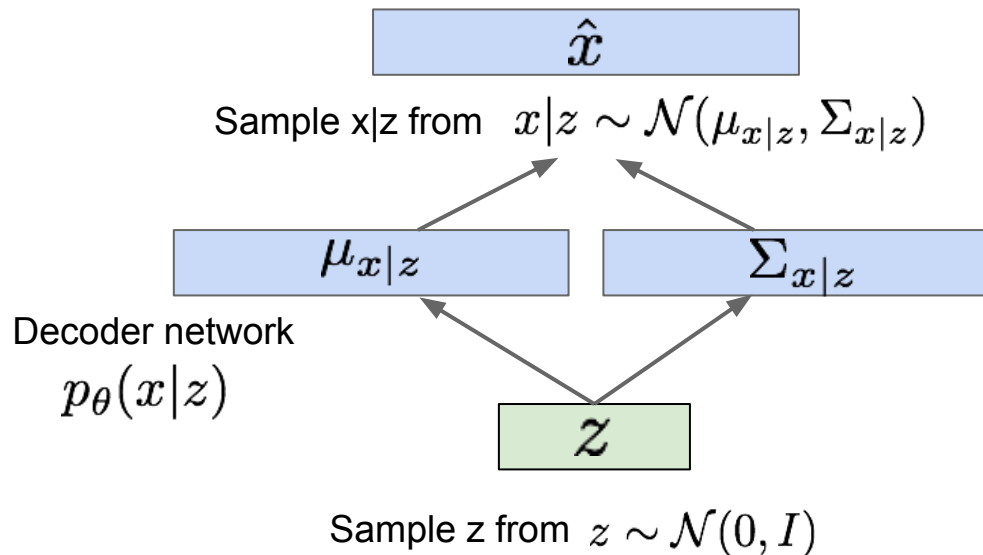
$$z^{(i)} \sim p_{\theta^*}(z)$$



Decoder network



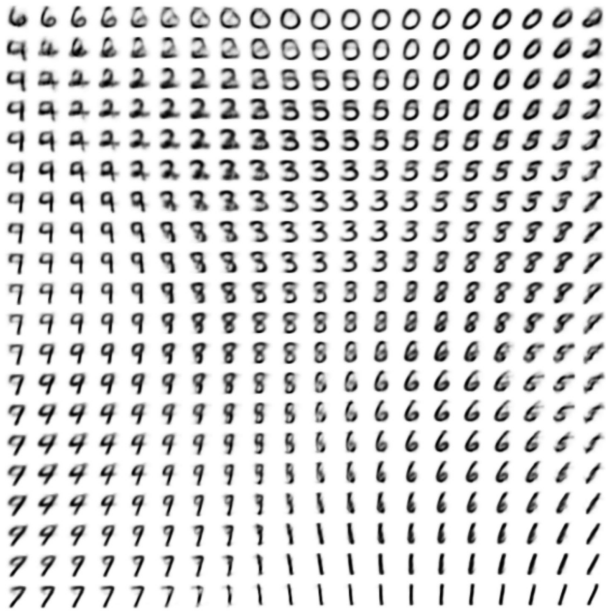
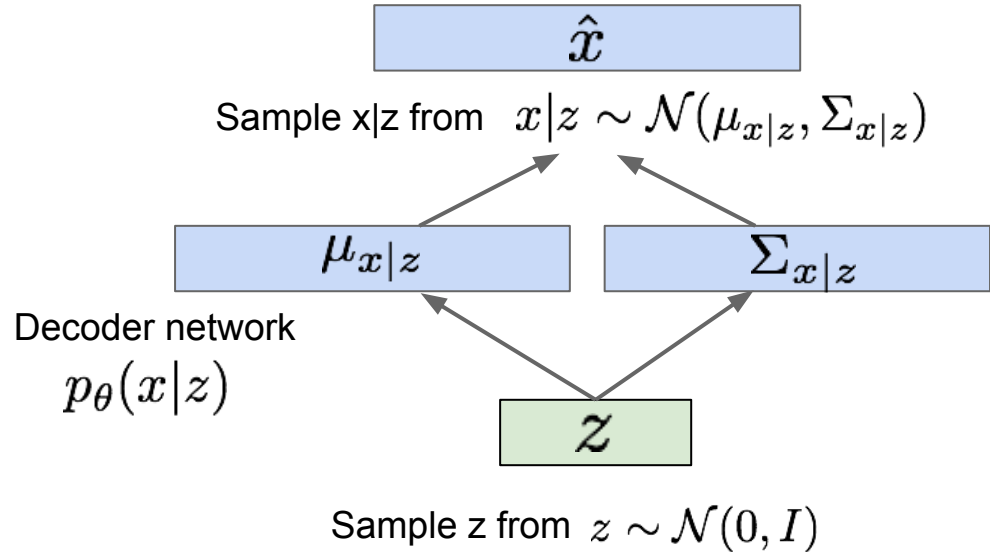
Now given a trained VAE:  
use decoder network & sample  $z$  from prior!



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders: Generating Data!

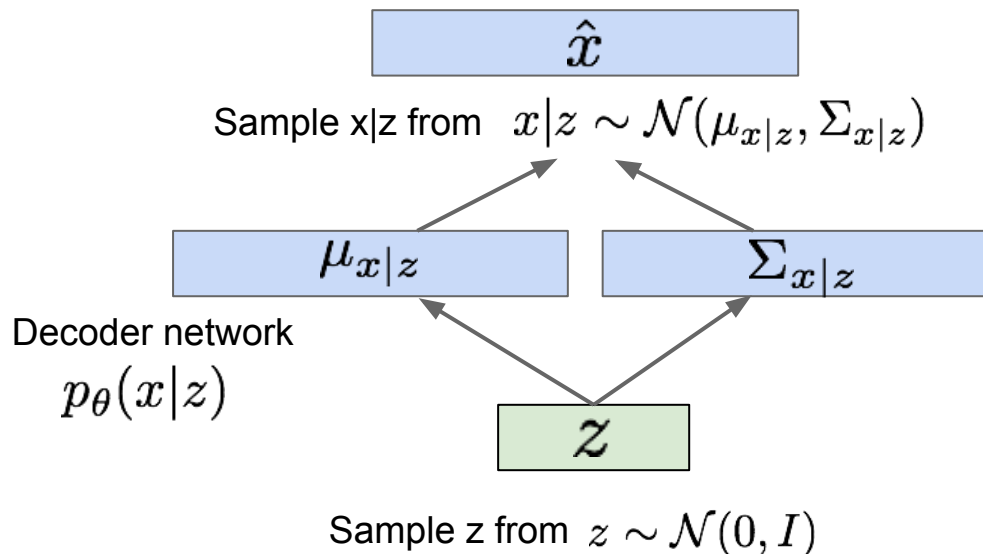
Use decoder network. Now sample  $z$  from prior!



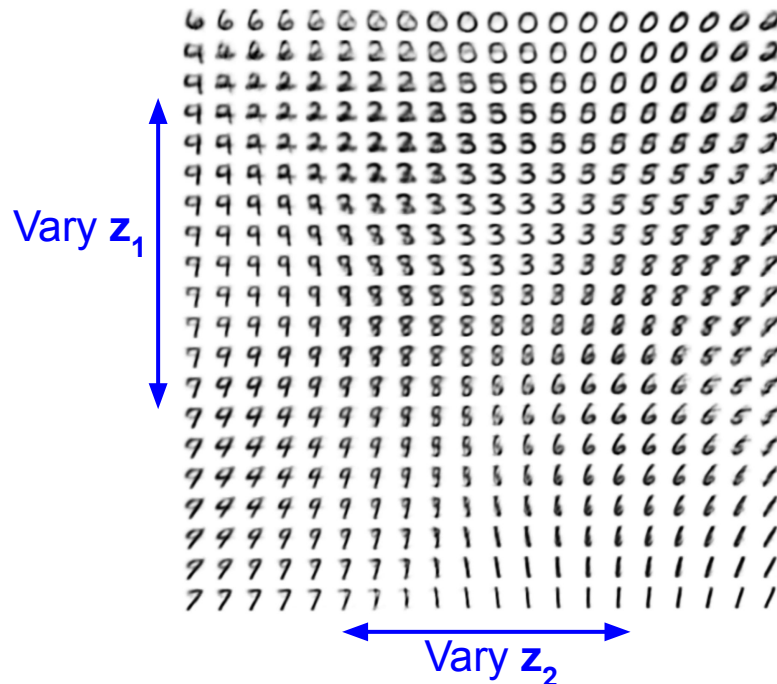
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders: Generating Data!

Use decoder network. Now sample  $z$  from prior!



Data manifold for 2-d  $z$



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014



# Variational Autoencoders: Generating Data!

Diagonal prior on  $\mathbf{z}$   
=> independent  
latent variables

Different  
dimensions of  $\mathbf{z}$   
encode  
interpretable factors  
of variation

Degree of smile

Vary  $z_1$



Vary  $z_2$

Head pose

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders: Generating Data!

Diagonal prior on  $\mathbf{z}$   
=> independent  
latent variables

Different  
dimensions of  $\mathbf{z}$   
encode  
interpretable factors  
of variation

Also good feature representation that  
can be computed using  $q_{\phi}(z|x)$ !

Degree of smile

Vary  $z_1$



Vary  $z_2$

Head pose

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders: Generating Data!



32x32 CIFAR-10



Labeled Faces in the Wild

Figures copyright (L) Dirk Kingma et al. 2016; (R) Anders Larsen et al. 2017. Reproduced with permission.

# Variational Autoencoders

Probabilistic spin to traditional autoencoders => allows generating data

Defines an intractable density => derive and optimize a (variational) lower bound

## Pros:

- Principled approach to generative models
- Interpretable latent space.
- Allows inference of  $q(z|x)$ , can be useful feature representation for other tasks

## Cons:

- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

## Active areas of research:

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, e.g., Gaussian Mixture Models (GMMs), Categorical Distributions.
- Learning disentangled representations.

# Taxonomy of Generative Models

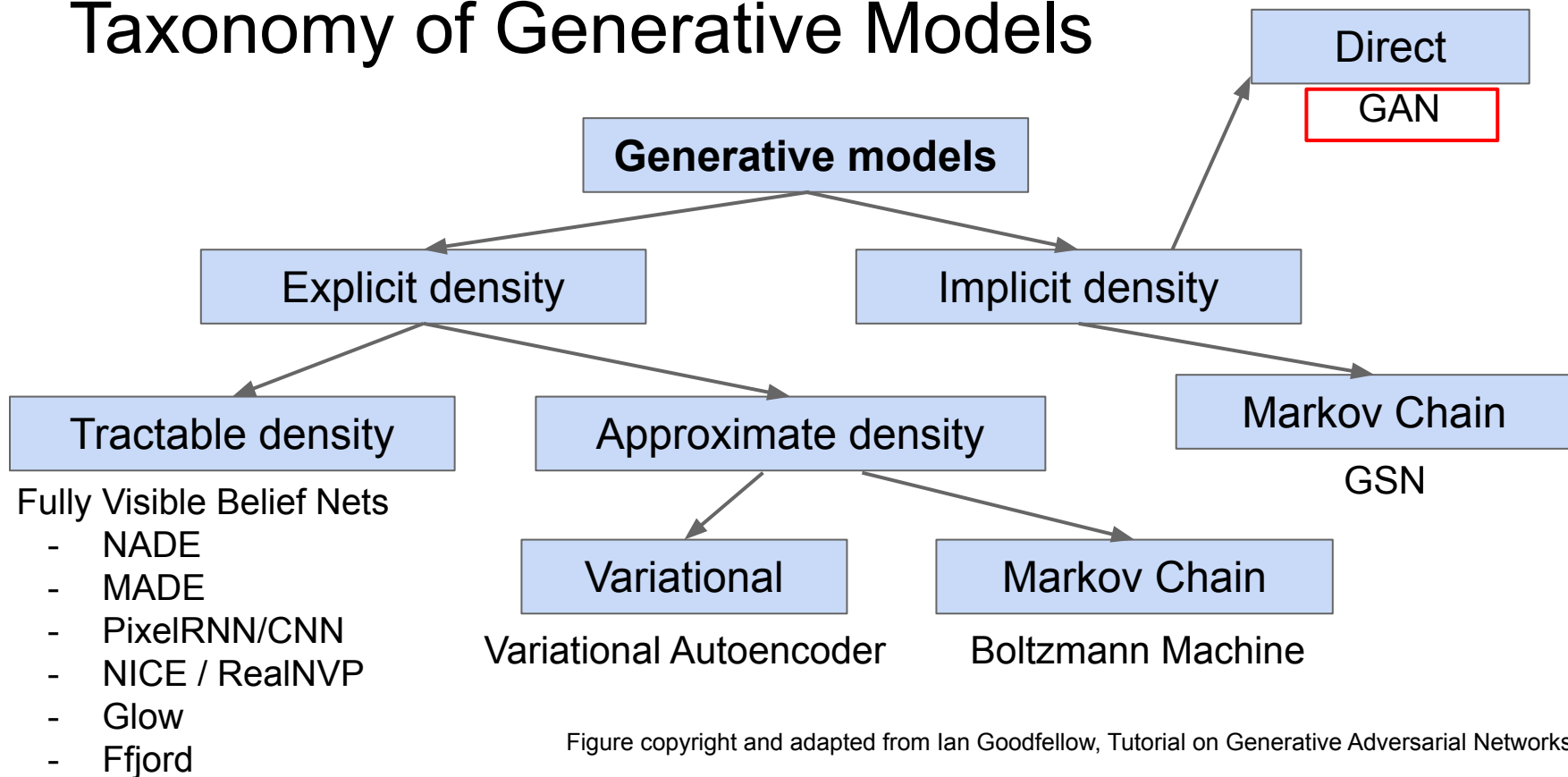


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

# Generative Adversarial Networks (GANs)

# So far...

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i | x_1, \dots, x_{i-1})$$

VAEs define intractable density function with latent  $\mathbf{z}$ :

$$p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$$

Cannot optimize directly, derive and optimize lower bound on likelihood instead

# So far...

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i | x_1, \dots, x_{i-1})$$

VAEs define intractable density function with latent  $\mathbf{z}$ :

$$p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$$

Cannot optimize directly, derive and optimize lower bound on likelihood instead

What if we give up on explicitly modeling density, and just want ability to sample?



# So far...

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i | x_1, \dots, x_{i-1})$$

VAEs define intractable density function with latent  $\mathbf{z}$ :

$$p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$$

Cannot optimize directly, derive and optimize lower bound on likelihood instead

What if we give up on explicitly modeling density, and just want ability to sample?

GANs: not modeling any explicit density function!

# Generative Adversarial Networks

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

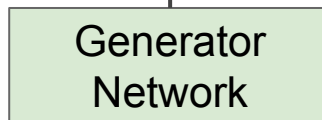
# Generative Adversarial Networks

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

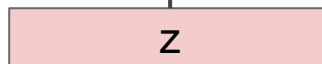
Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

Output: Sample from training distribution



Input: Random noise



# Generative Adversarial Networks

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

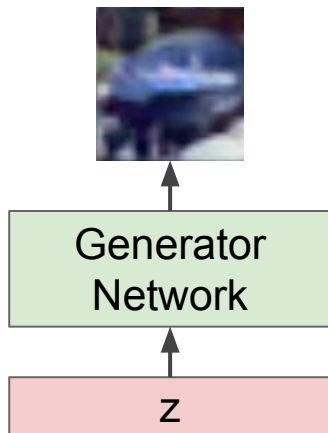
Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample  $z$  maps to which training image -> can't learn by reconstructing training images

Output: Sample from training distribution

Input: Random noise



# Generative Adversarial Networks

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

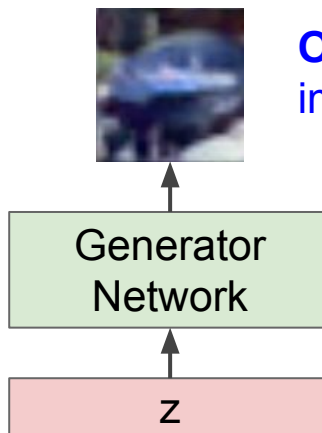
But we don't know which sample  $z$  maps to which training image -> can't learn by reconstructing training images

Output: Sample from training distribution



**Objective:** generated images should look "real"

Input: Random noise



# Generative Adversarial Networks

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

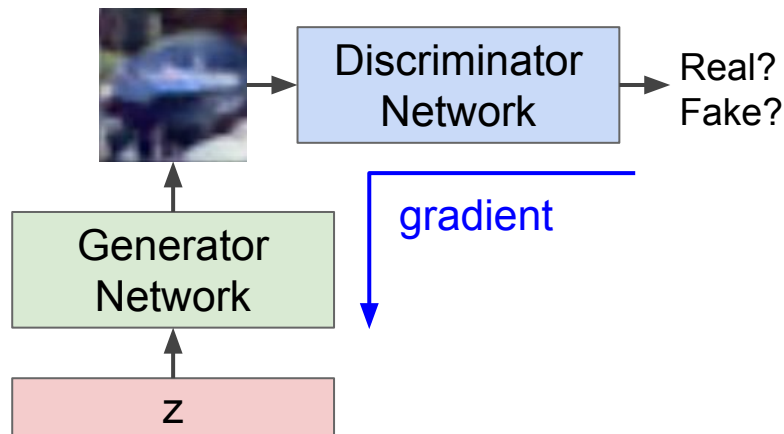
Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample  $z$  maps to which training image -> can't learn by reconstructing training images

Solution: Use a discriminator network to tell whether the generate image is within data distribution ("real") or not

Output: Sample from training distribution

Input: Random noise



# Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

**Discriminator network:** try to distinguish between real and fake images

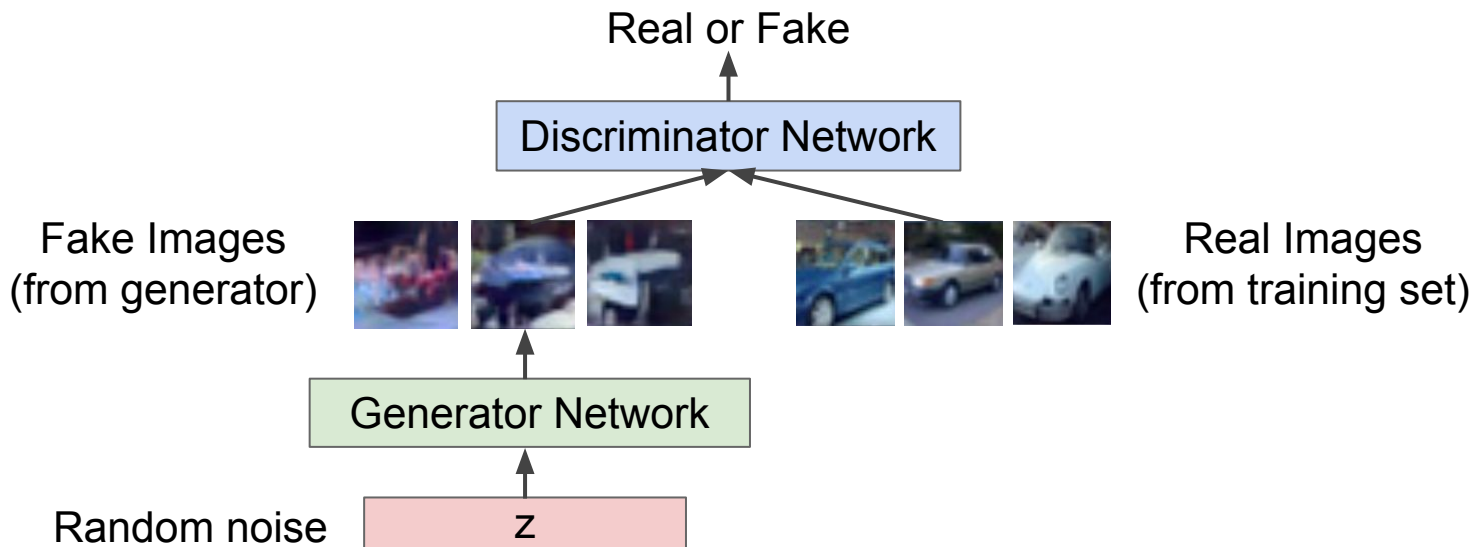
**Generator network:** try to fool the discriminator by generating real-looking images

# Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

**Discriminator network:** try to distinguish between real and fake images

**Generator network:** try to fool the discriminator by generating real-looking images



Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

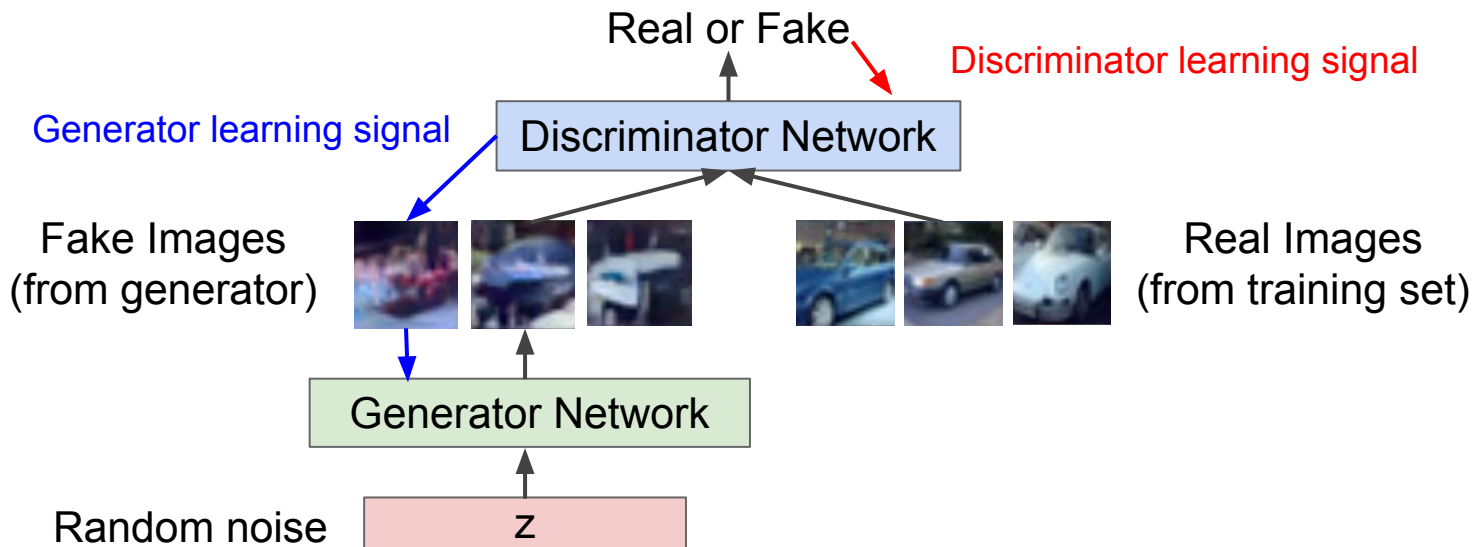


# Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

**Discriminator network:** try to distinguish between real and fake images

**Generator network:** try to fool the discriminator by generating real-looking images



Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

# Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

**Discriminator network:** try to distinguish between real and fake images

**Generator network:** try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

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

Generator objective      Discriminator objective

# Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

**Discriminator network:** try to distinguish between real and fake images

**Generator network:** try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\substack{\text{Discriminator output} \\ \text{for real data } x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\substack{\text{Discriminator output for} \\ \text{generated fake data } G(z)}}) \right]$$

Discriminator outputs likelihood in (0,1) of real image

# Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

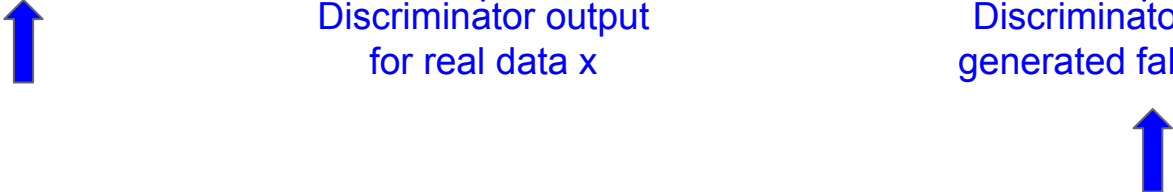
**Discriminator network:** try to distinguish between real and fake images

**Generator network:** try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\substack{\text{Discriminator output} \\ \text{for real data } x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\substack{\text{Discriminator output for} \\ \text{generated fake data } G(z)}}) \right]$$


# Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

**Discriminator network:** try to distinguish between real and fake images

**Generator network:** try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\substack{\text{Discriminator output} \\ \text{for real data } x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\substack{\text{Discriminator output for} \\ \text{generated fake data } G(z)}}) \right]$$

# Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

**Discriminator network:** try to distinguish between real and fake images

**Generator network:** try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\substack{\text{Discriminator output} \\ \text{for real data } x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\substack{\text{Discriminator output for} \\ \text{generated fake data } G(z)}}) \right]$$

- Discriminator ( $\theta_d$ ) wants to **maximize objective** such that  $D(x)$  is close to 1 (real) and  $D(G(z))$  is close to 0 (fake)
- Generator ( $\theta_g$ ) wants to **minimize objective** such that  $D(G(z))$  is close to 1 (discriminator is fooled into thinking generated  $G(z)$  is real)

# Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

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

2. **Gradient descent** on generator

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

# Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

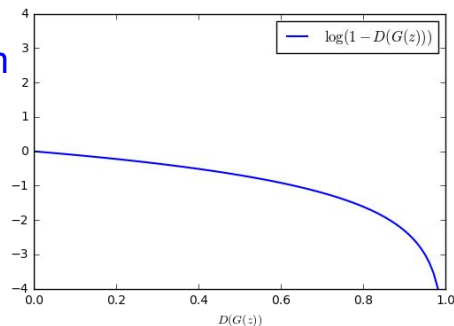
$$\max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

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

In practice, optimizing this generator objective does not work well!

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).





# Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

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

2. **Gradient descent** on generator

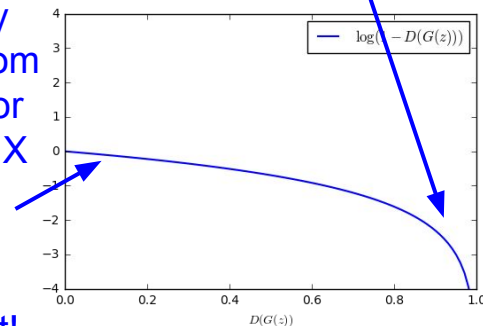
$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).

But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good



# Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

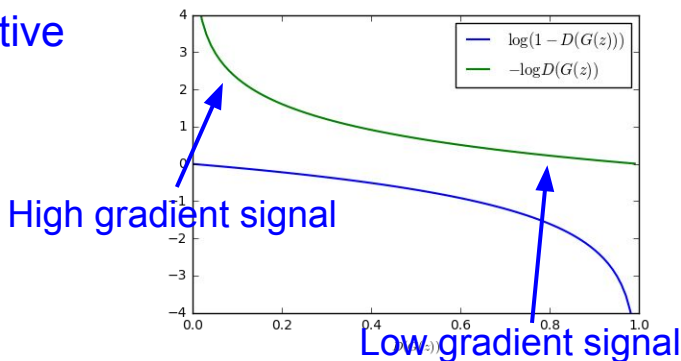
$$\max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Instead: Gradient ascent** on generator, **different objective**

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

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.

Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.



# Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

## Putting it together: GAN training algorithm

**for** number of training iterations **do**

**for**  $k$  steps **do**

- Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
- Sample minibatch of  $m$  examples  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  from data generating distribution  $p_{\text{data}}(\mathbf{x})$ .
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

**end for**

- Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

**end for**

# Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

## Putting it together: GAN training algorithm

**for** number of training iterations **do**

**for**  $k$  steps **do**

- Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
- Sample minibatch of  $m$  examples  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  from data generating distribution  $p_{\text{data}}(\mathbf{x})$ .
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

**end for**

- Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

**end for**

Some find  $k=1$   
more stable,  
others use  $k > 1$ ,  
no best rule.

Followup work  
(e.g. Wasserstein  
GAN, BEGAN)  
alleviates this  
problem, better  
stability!

Arjovsky et al. "Wasserstein gan." arXiv preprint arXiv:1701.07875 (2017)

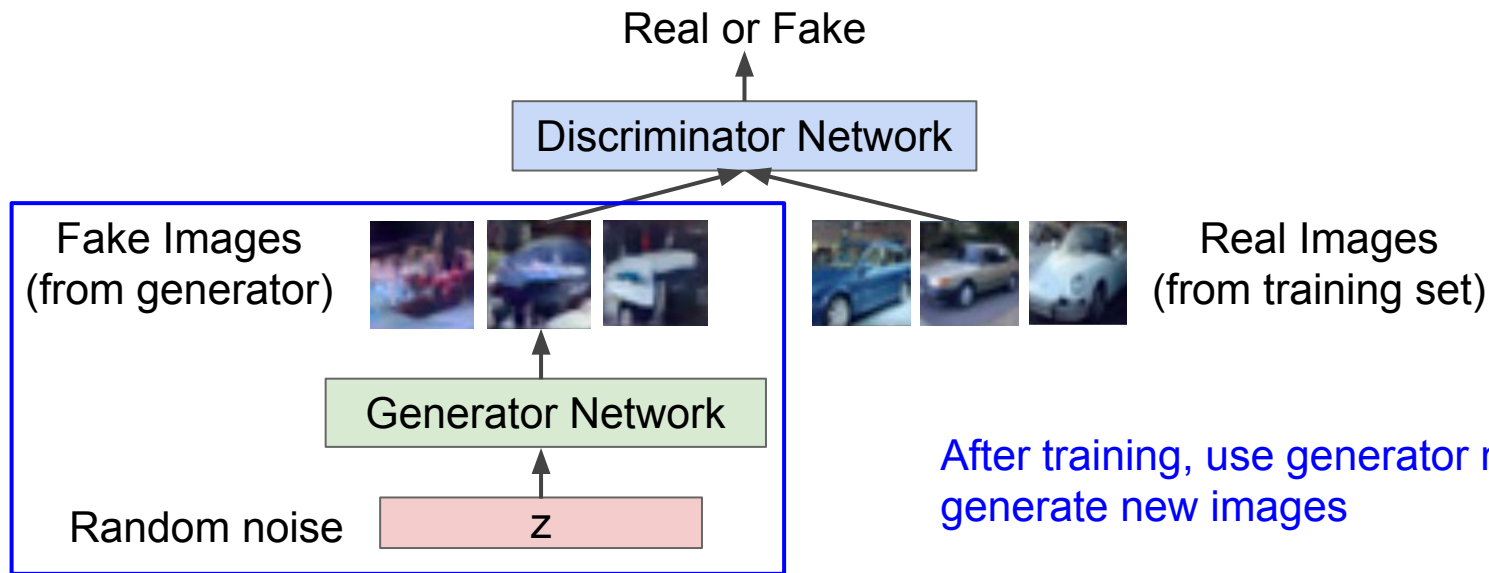
Berthelot, et al. "Began: Boundary equilibrium generative adversarial networks." arXiv preprint arXiv:1703.10717 (2017)

# Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

**Generator network:** try to fool the discriminator by generating real-looking images

**Discriminator network:** try to distinguish between real and fake images



Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

# Generative Adversarial Nets

Generated samples

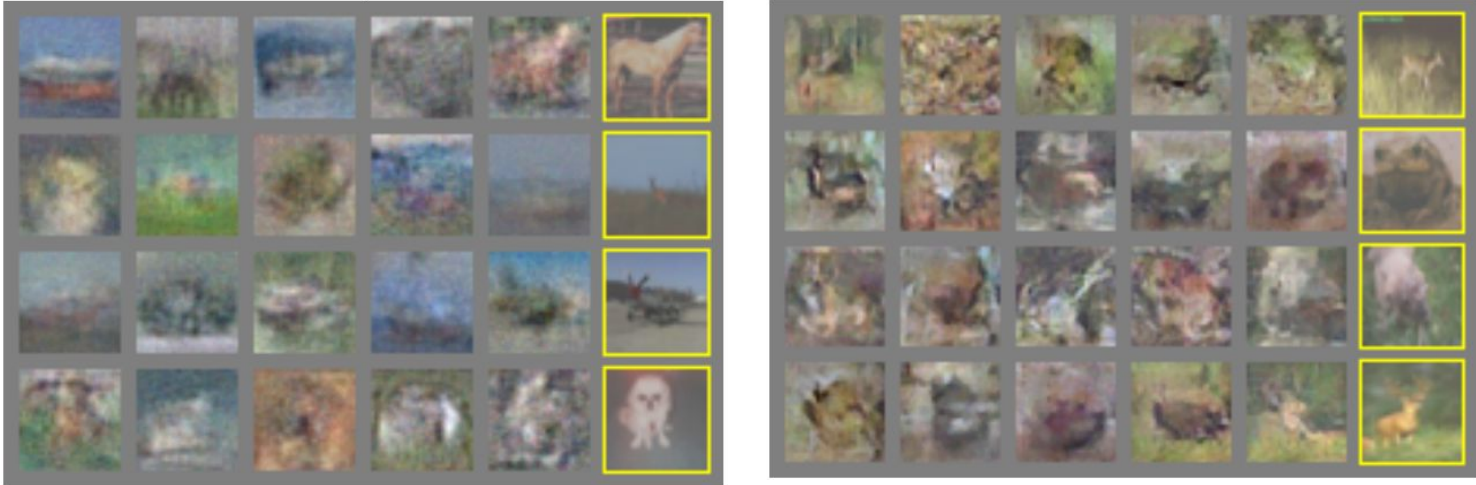


Nearest neighbor from training set

Figures copyright Ian Goodfellow et al., 2014. Reproduced with permission.

# Generative Adversarial Nets

Generated samples (CIFAR-10)



Nearest neighbor from training set

Figures copyright Ian Goodfellow et al., 2014. Reproduced with permission.

# Generative Adversarial Nets: Convolutional Architectures

Generator is an upsampling network with fractionally-strided convolutions  
Discriminator is a convolutional network

## Architecture guidelines for stable Deep Convolutional GANs

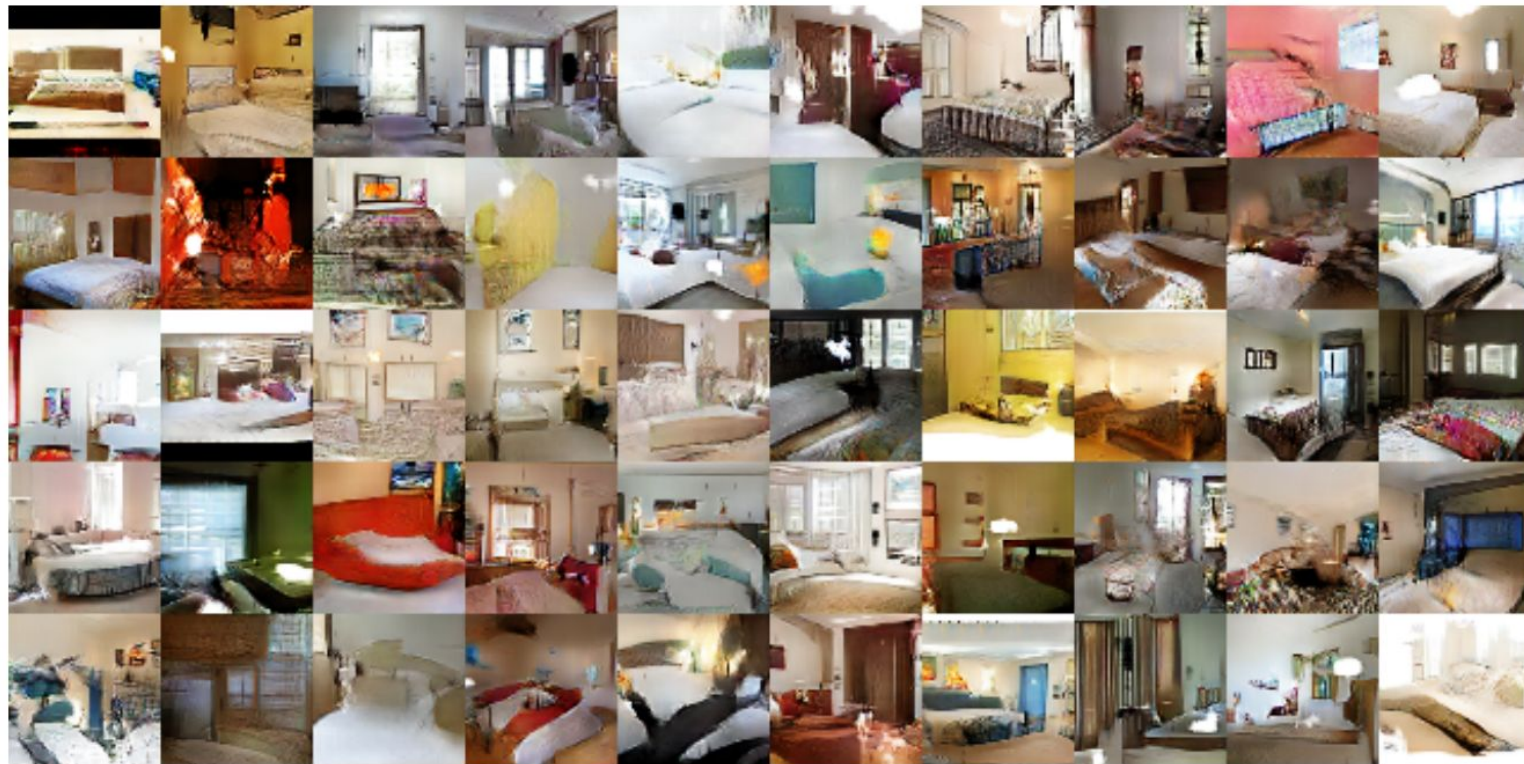
- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016



# Generative Adversarial Nets: Convolutional Architectures

Samples from the model look much better!



Radford et al,  
ICLR 2016

# Generative Adversarial Nets: Convolutional Architectures

Interpolating  
between  
random  
points in latent  
space



Radford et al,  
ICLR 2016

# Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016

Smiling woman

Neutral woman

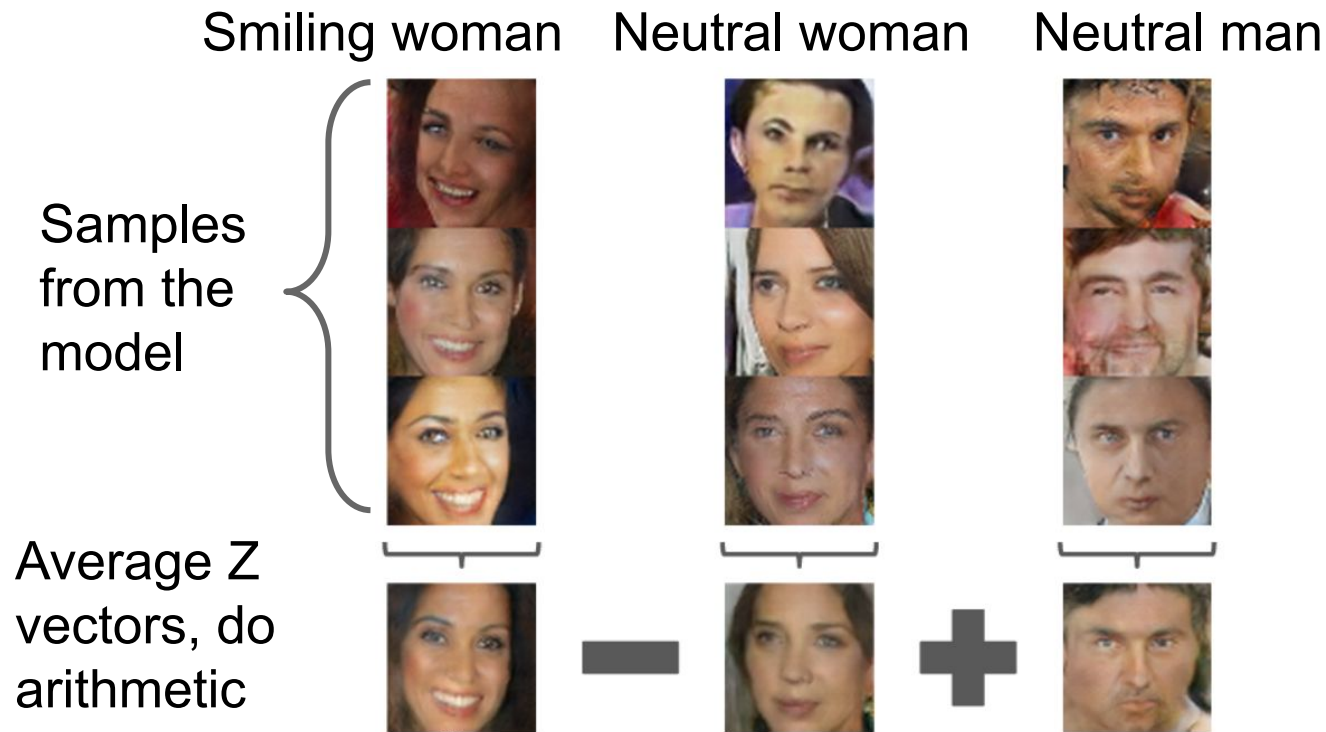
Neutral man

Samples  
from the  
model



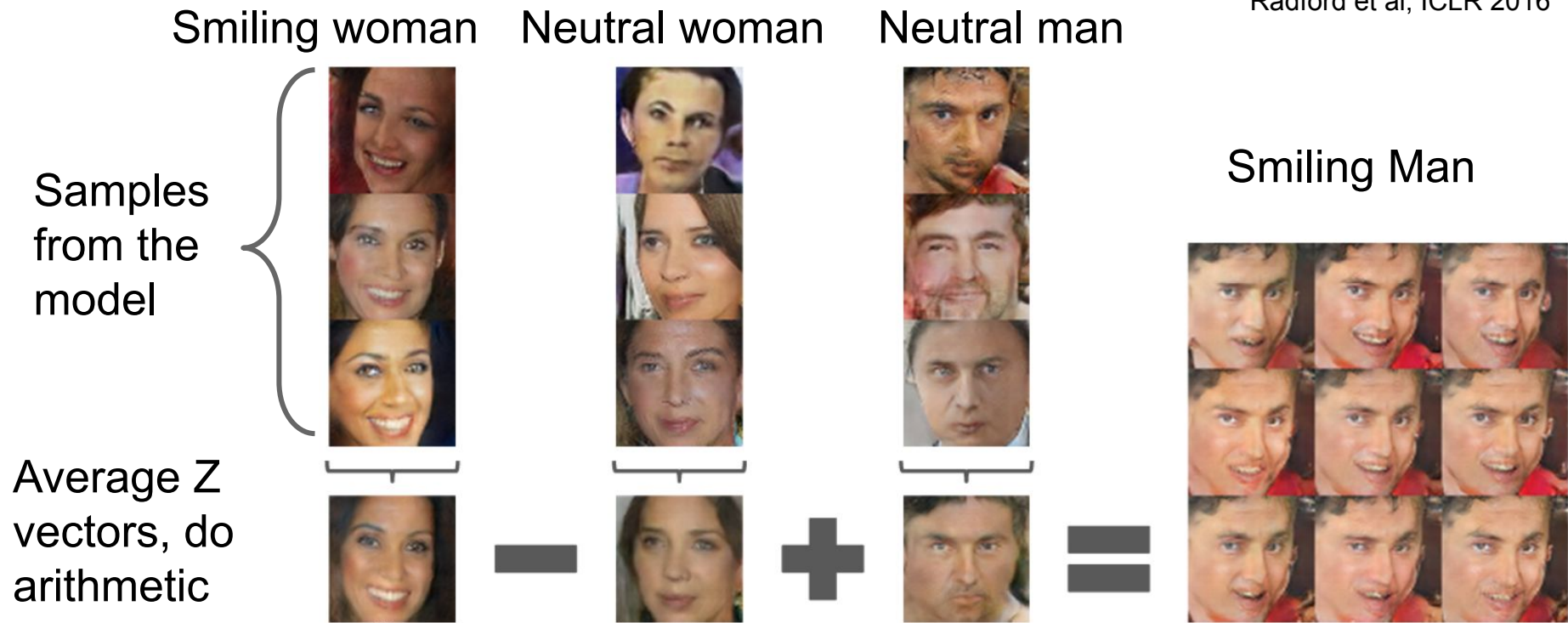
# Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016



# Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016



# Generative Adversarial Nets: Interpretable Vector Math

Glasses man



No glasses man



No glasses woman



-

+

=

Radford et al,  
ICLR 2016

Woman with glasses



# 2017: Explosion of GANs

## “The GAN Zoo”

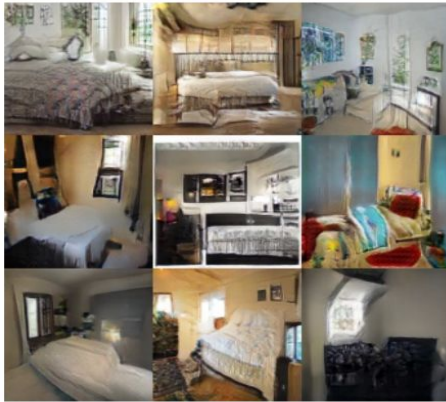
See also: <https://github.com/soumith/ganhacks> for tips and tricks for trainings GANs

- GAN - Generative Adversarial Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- AdaGAN - AdaGAN: Boosting Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference
- AM-GAN - Generative Adversarial Nets with Labeled Data by Activation Maximization
- AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
- ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorical GANs
- b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
- Bayesian GAN - Deep and Hierarchical Implicit Models
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BiGAN - Adversarial Feature Learning
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- CGAN - Conditional Generative Adversarial Nets
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
- CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CoGAN - Coupled Generative Adversarial Networks
- Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
- C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training
- CS-GAN - Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
- CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
- CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
- DTN - Unsupervised Cross-Domain Image Generation
- DCGAN - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- DiscoGAN - Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
- DR-GAN - Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
- DualGAN - DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
- EBGAN - Energy-based Generative Adversarial Network
- f-GAN - f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
- FF-GAN - Towards Large-Pose Face Frontalization in the Wild
- GAWWN - Learning What and Where to Draw
- GeneGAN - GeneGAN: Learning Object Transfiguration and Attribute Subspace from Unpaired Data
- Geometric GAN - Geometric GAN
- GoGAN - Gang of GANs: Generative Adversarial Networks with Maximum Margin Ranking
- GP-GAN - GP-GAN: Towards Realistic High-Resolution Image Blending
- IAN - Neural Photo Editing with Introspective Adversarial Networks
- iGAN - Generative Visual Manipulation on the Natural Image Manifold
- IcGAN - Invertible Conditional GANs for image editing
- ID-CGAN - Image De-raining Using a Conditional Generative Adversarial Network
- Improved GAN - Improved Techniques for Training GANs
- InfoGAN - InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- LAGAN - Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis
- LAPGAN - Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

<https://github.com/hindupuravinash/the-gan-zoo>

# 2017: Explosion of GANs

## Better training and generation



LSGAN, Zhu 2017.



Wasserstein GAN,  
Arjovsky 2017.  
Improved Wasserstein  
GAN, Gulrajani 2017.

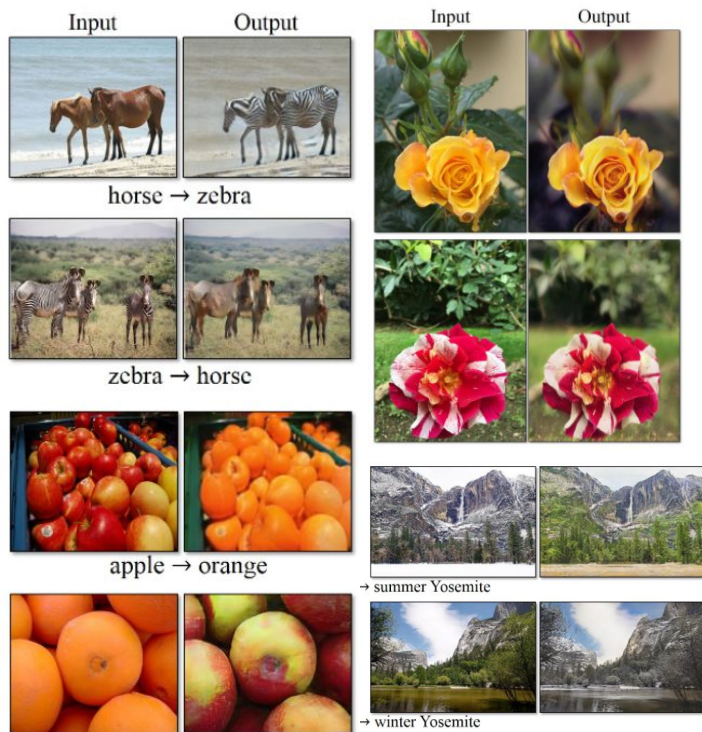


Progressive GAN, Karras 2018.



# 2017: Explosion of GANs

## Source->Target domain transfer



CycleGAN. Zhu et al. 2017.

## Text -> Image Synthesis

this small bird has a pink breast and crown, and black primaries and secondaries.

this magnificent fellow is almost all black with a red crest, and white cheek patch.



Reed et al. 2017.

## Many GAN applications



Pix2pix. Isola 2017. Many examples at <https://phillipi.github.io/pix2pix/>

# 2019: BigGAN



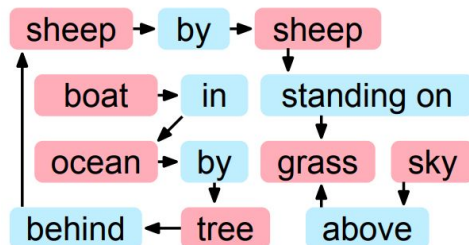
Brock et al., 2019

# Scene graphs to GANs

Specifying exactly what kind of image you want to generate.

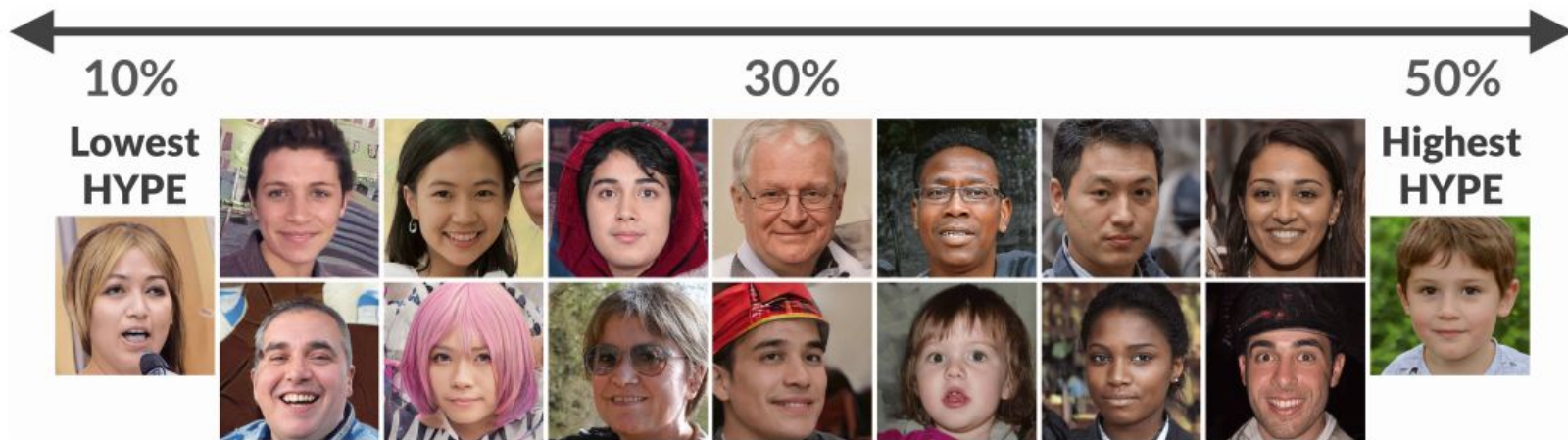
The explicit structure in scene graphs provides better image generation for complex scenes.

## Scene Graph



# HYPE: Human eYe Perceptual Evaluations

[hype.stanford.edu](http://hype.stanford.edu)



Zhou, Gordon, Krishna et al. HYPE: Human eYe Perceptual Evaluations, NeurIPS 2019

Figures copyright 2019. Reproduced with permission.

# Summary: GANs

Don't work with an explicit density function

Take game-theoretic approach: learn to generate from training distribution through 2-player game

Pros:

- Beautiful, state-of-the-art samples!

Cons:

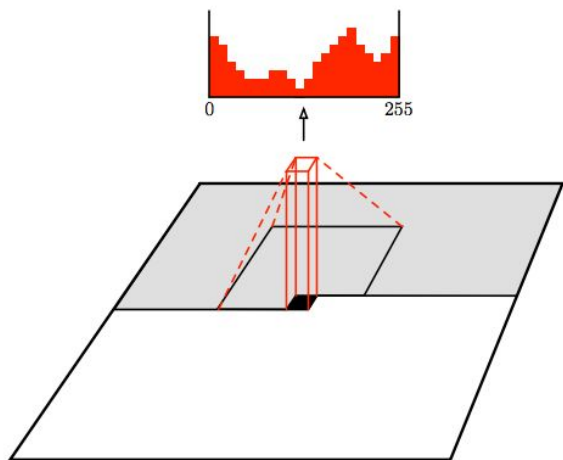
- Trickier / more unstable to train
- Can't solve inference queries such as  $p(x)$ ,  $p(z|x)$

Active areas of research:

- Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
- Conditional GANs, GANs for all kinds of applications

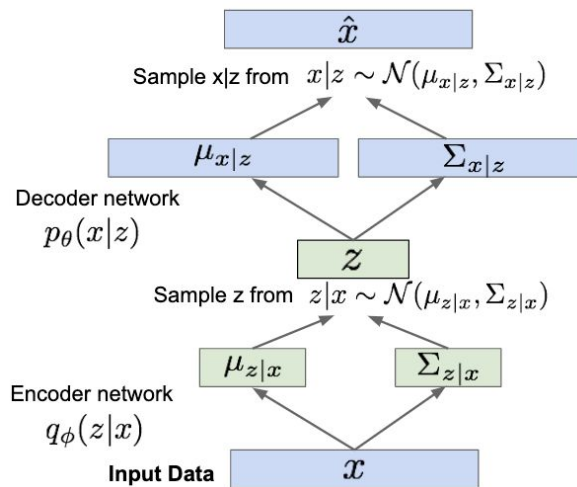
# Summary

## Autoregressive models: PixelRNN, PixelCNN



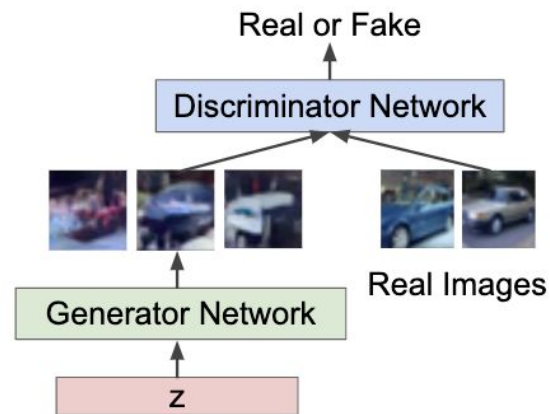
Van der Oord et al, "Conditional image generation with pixelCNN decoders", NIPS 2016

## Variational Autoencoders



Kingma and Welling, "Auto-encoding variational bayes", ICLR 2013

## Generative Adversarial Networks (GANs)



Goodfellow et al, "Generative Adversarial Nets", NIPS 2014

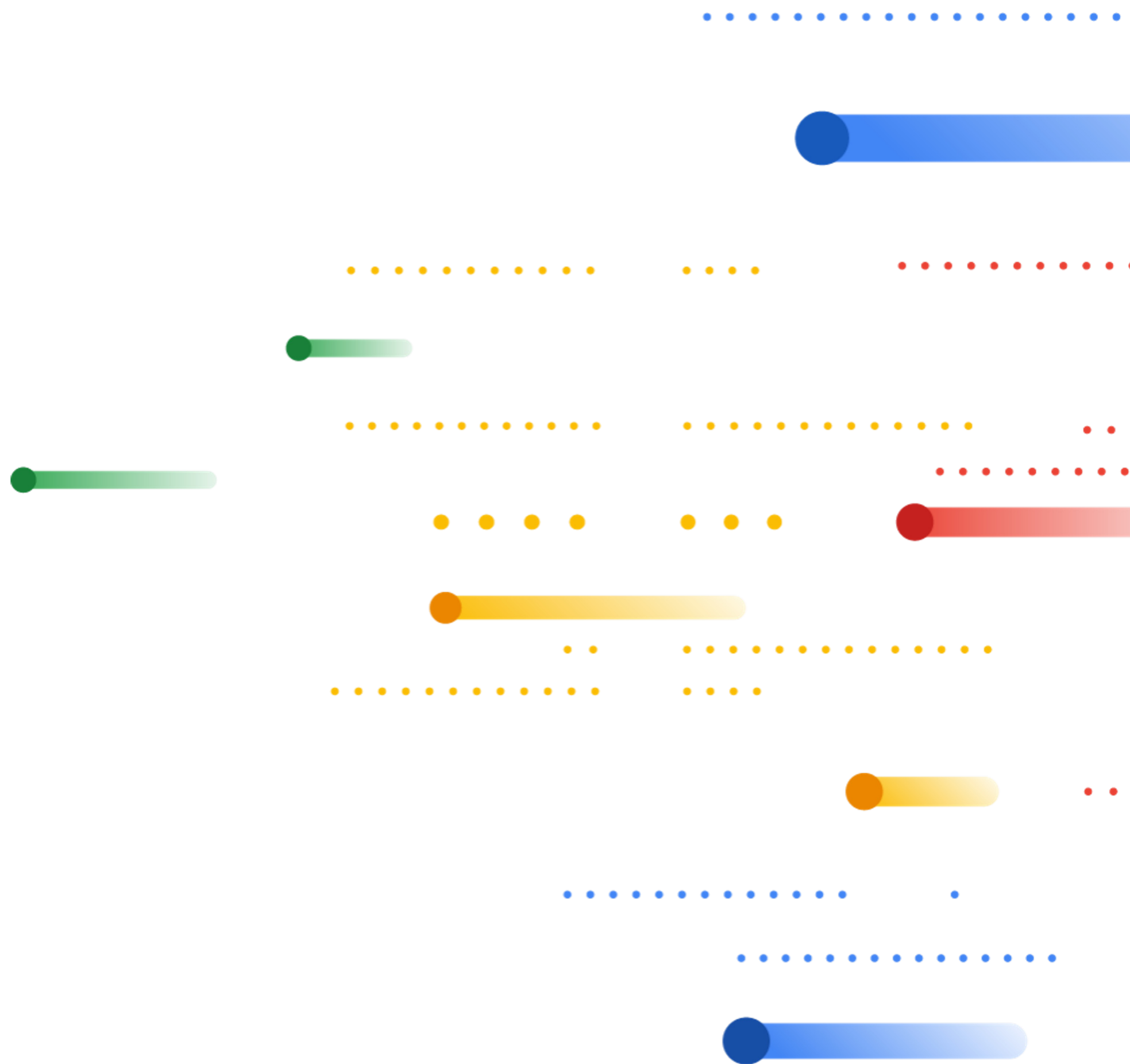
# Useful Resources on Generative Models

CS 236: [Deep Generative Models](#) (Stanford)

CS 294-158 [Deep Unsupervised Learning](#) (Berkeley)

# Variational Autoencoders and Diffusion Models

Ruiqi Gao @Stanford cs231n  
May 25, 2023





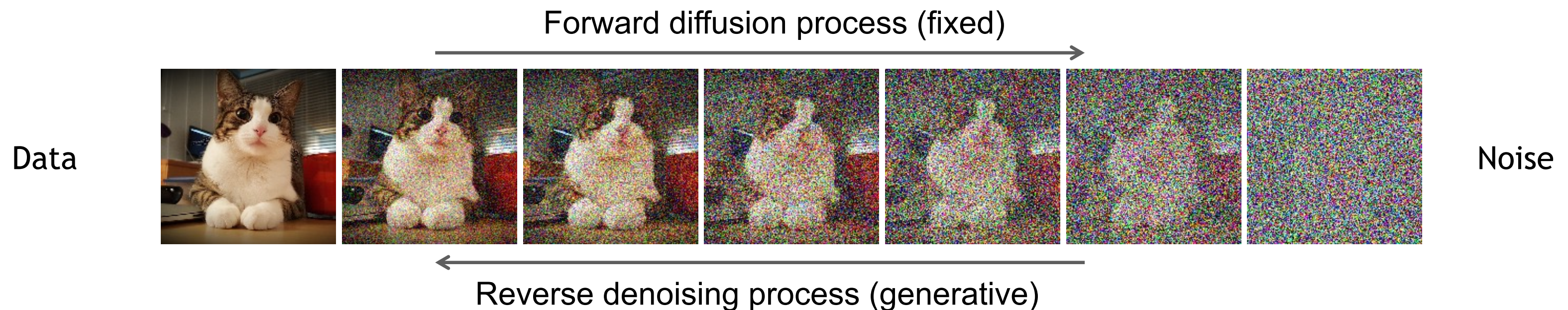
# Diffusion Models

# Denoising Diffusion Models

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



[Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015](#)

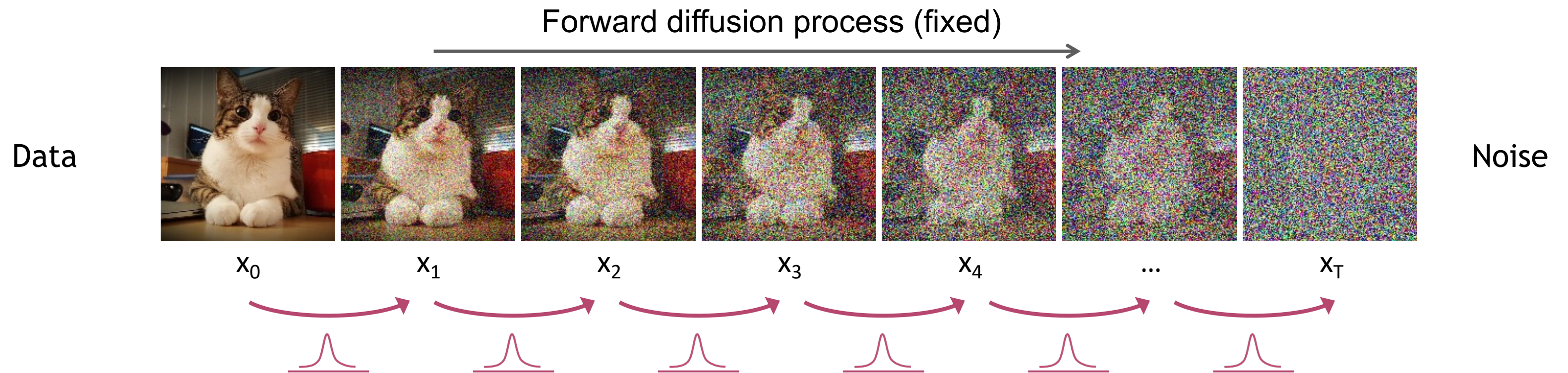
[Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020](#)

[Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021](#)

slide from <https://cvpr2022-tutorial-diffusion-models.github.io/>

# Forward Diffusion Process

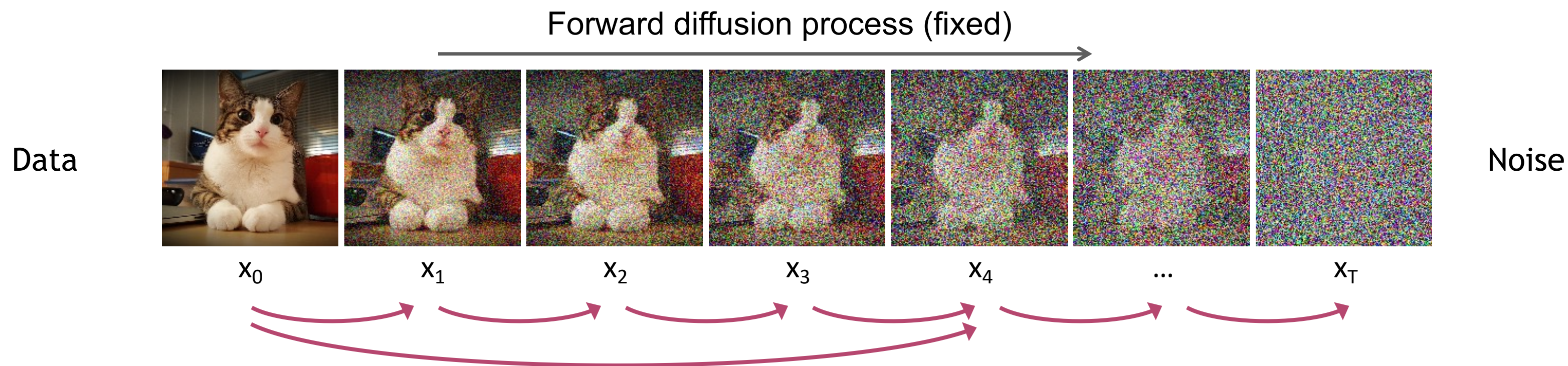
The formal definition of the forward process in T steps:



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

Similar to the inference model in hierarchical VAEs.

# Diffusion Kernel



Define  $\bar{\alpha}_t = \prod_{s=1}^t (1 - \beta_s)$   $\rightarrow$   $q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I})$  (Diffusion Kernel)

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

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

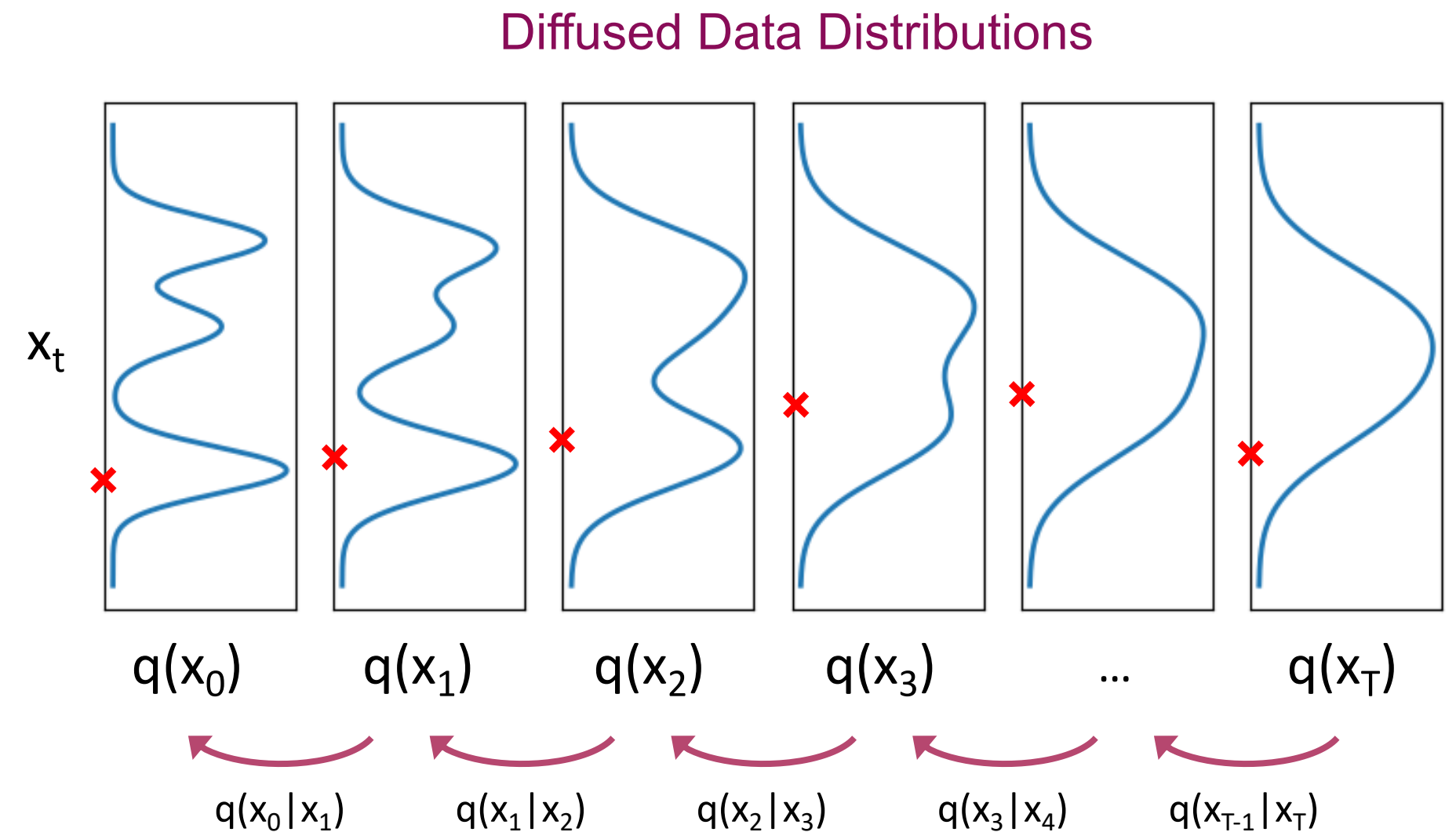
# Generative Learning by Denoising

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

**Generation:**

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

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

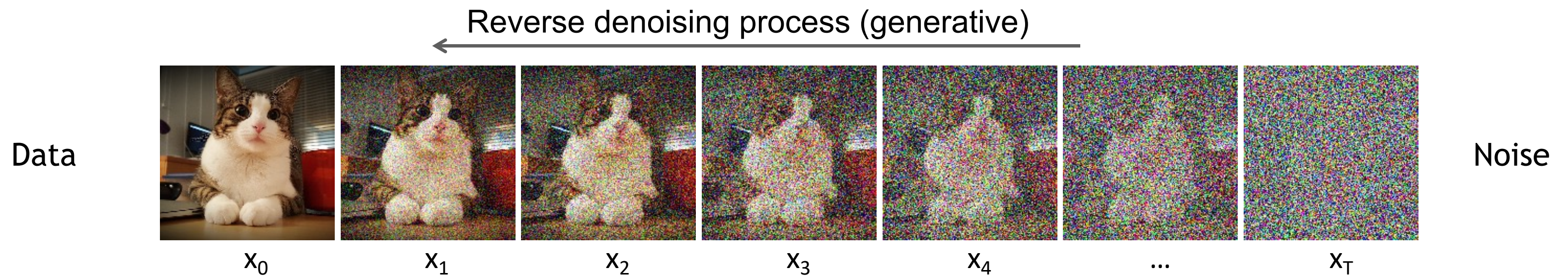


In general,  $q(\mathbf{x}_{t-1}|\mathbf{x}_t) \propto q(\mathbf{x}_{t-1})q(\mathbf{x}_t|\mathbf{x}_{t-1})$  is intractable.

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

# Reverse Denoising Process

Formal definition of forward and reverse processes in T steps:



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

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \underbrace{\mu_\theta(\mathbf{x}_t, t)}_{\text{Trainable network}}, \sigma_t^2 \mathbf{I}) \quad \rightarrow \quad p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$

Trainable network  
(U-net, Denoising Autoencoder)

Similar to the generative model in hierarchical VAEs.

# Learning Denoising Model

## Variational upper bound

For training, we can form variational upper bound (negative ELBO) that is commonly used for training variational autoencoders:

$$\mathbb{E}_{q(\mathbf{x}_0)} [-\log p_\theta(\mathbf{x}_0)] \leq \mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[ -\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] =: L$$

[Sohl-Dickstein et al. ICML 2015](#) and [Ho et al. NeurIPS 2020](#) show that:

$$L = \mathbb{E}_q \left[ \underbrace{D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T))}_{L_T} + \sum_{t>1} \underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0} \right]$$

where  $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$  is the tractable posterior distribution:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}),$$

$$\text{where } \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t} \mathbf{x}_0 + \frac{\sqrt{1 - \beta_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \mathbf{x}_t \text{ and } \tilde{\beta}_t := \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t$$

# Parameterizing the Denoising Model

Since both  $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$  and  $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$  are Normal distributions, the KL divergence has a simple form:

$$L_{t-1} = D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) = \mathbb{E}_q \left[ \frac{1}{2\sigma_t^2} \|\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)\|^2 \right] + C$$

Recall that  $\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{(1 - \bar{\alpha}_t)} \epsilon$ . [Ho et al. NeurIPS 2020](#) observe that:

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{1 - \beta_t}} \left( \mathbf{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon \right)$$

They propose to represent the mean of the denoising model using a *noise-prediction* network:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{1 - \beta_t}} \left( \mathbf{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right)$$

With this parameterization

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[ \frac{\beta_t^2}{2\sigma_t^2(1 - \beta_t)(1 - \bar{\alpha}_t)} \|\epsilon - \underbrace{\epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t)}_{\mathbf{x}_t})}_{\mathbf{x}_t}\|^2 \right] + C$$



# Training Objective Weighting

Trading likelihood for perceptual quality

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[ \underbrace{\frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\bar{\alpha}_t)}}_{\lambda_t} \|\epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t} \epsilon, t)}_{\mathbf{x}_t}\| \|^2 \right]$$

The time dependent  $\lambda_t$  ensures that the training objective is weighted properly for the maximum data likelihood training.

However, this weight is often very large for small  $t$ 's.

[Ho et al. NeurIPS 2020](#) observe that simply setting  $\lambda_t = 1$  improves sample quality. So, they propose to use:

$$L_{\text{simple}} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), t \sim \mathcal{U}(1, T)} \left[ \|\epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t} \epsilon}_{\mathbf{x}_t}, t)\| \|^2 \right]$$

# Summary

## Training and Sample Generation

---

### Algorithm 1 Training

---

- 1: **repeat**
  - 2:  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  - 3:  $t \sim \text{Uniform}(\{1, \dots, T\})$
  - 4:  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  - 5: Take gradient descent step on  
$$\nabla_{\theta} \left\| \epsilon - \epsilon_{\theta} \left( \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t \right) \right\|^2$$
  - 6: **until** converged
- 

---

### Algorithm 2 Sampling

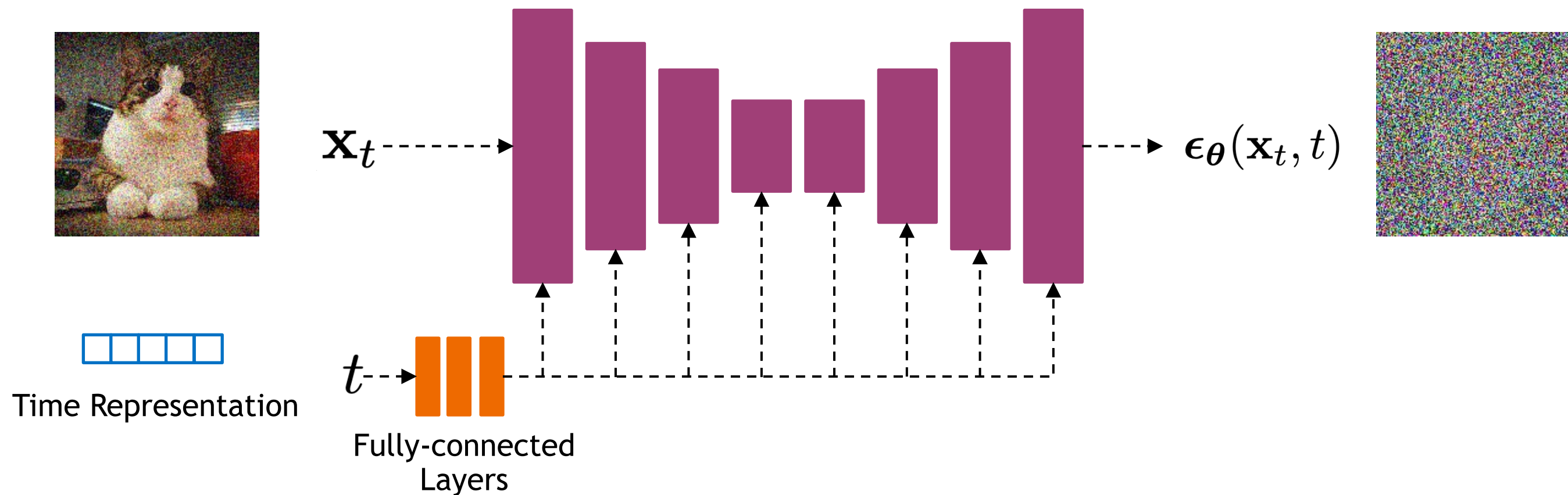
---

- 1:  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  - 2: **for**  $t = T, \dots, 1$  **do**
  - 3:  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  - 4:  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_{\theta}(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
  - 5: **end for**
  - 6: **return**  $\mathbf{x}_0$
-

# Implementation Considerations

## Network Architectures

Diffusion models often use U-Net architectures with ResNet blocks and self-attention layers to represent  $\epsilon_{\theta}(\mathbf{x}_t, t)$



Time representation: sinusoidal positional embeddings or random Fourier features.

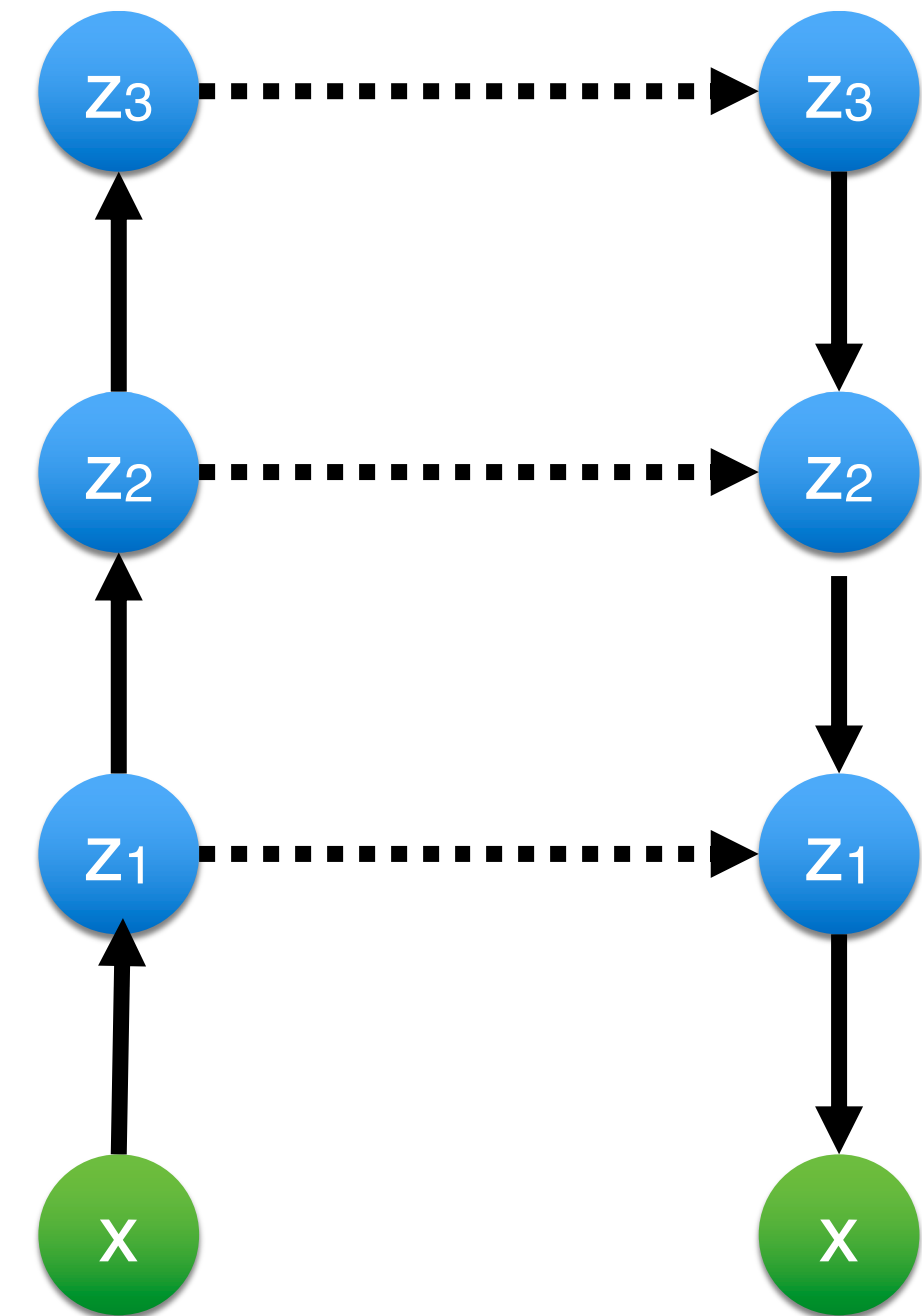
Time features are fed to the residual blocks using either simple spatial addition or using adaptive group normalization layers. (see [Dhariwal and Nichol NeurIPS 2021](#))

# Connection to VAEs

Diffusion models can be considered as a special form of hierarchical VAEs.

However, in diffusion models:

- The inference model is fixed: easier to optimize
- The latent variables have the same dimension as the data.
- The ELBO is decomposed to each time step: fast to train
  - Can be made extremely deep (even infinitely deep)
- The model is trained with some reweighting of the ELBO.



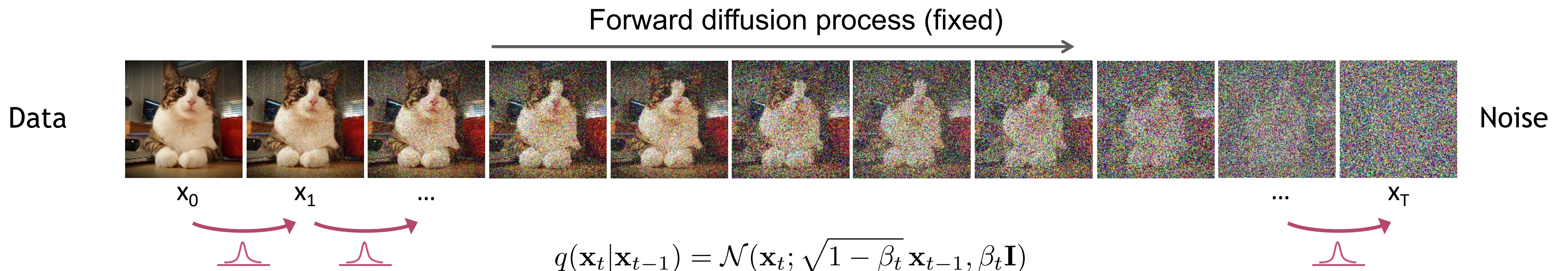
Inference model  
 $q(z|x)$

Generative model  
 $p(x,z)$

# Continuous-time diffusion models

## Stochastic differential equation framework

Consider the limit of many small steps:



$$\mathbf{x}_t = \sqrt{1 - \beta_t} \mathbf{x}_{t-1} + \sqrt{\beta_t} \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$= \sqrt{1 - \beta(t)\Delta t} \mathbf{x}_{t-1} + \sqrt{\beta(t)\Delta t} \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$(\beta_t := \beta(t)\Delta t)$$

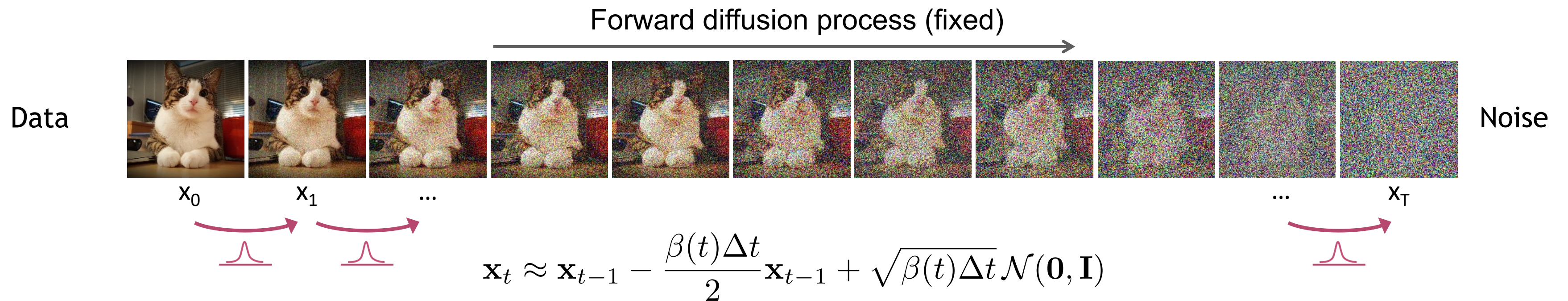


$$\approx \mathbf{x}_{t-1} - \frac{\beta(t)\Delta t}{2} \mathbf{x}_{t-1} + \sqrt{\beta(t)\Delta t} \mathcal{N}(\mathbf{0}, \mathbf{I})$$

(Taylor expansion)

# Forward Diffusion Process as Stochastic Differential Equation

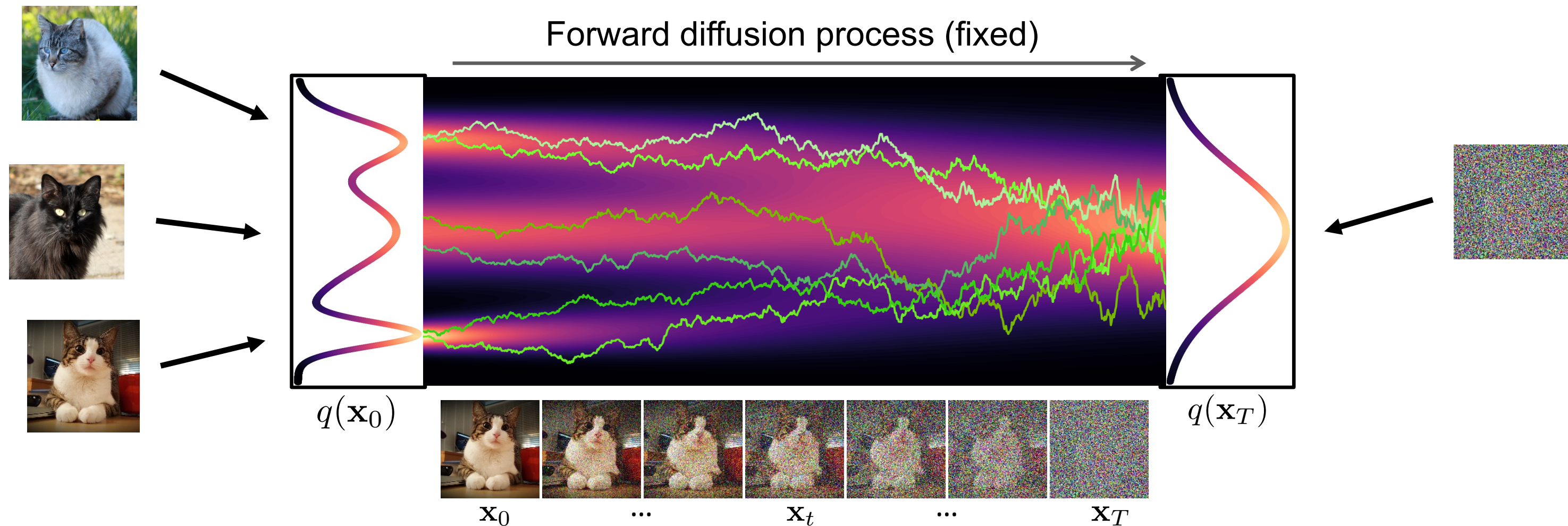
Consider the limit of many small steps:



$$d\mathbf{x}_t = -\frac{1}{2}\beta(t)\mathbf{x}_t dt + \sqrt{\beta(t)} d\omega_t$$

**Stochastic Differential Equation (SDE)**  
describing the diffusion in infinitesimal limit

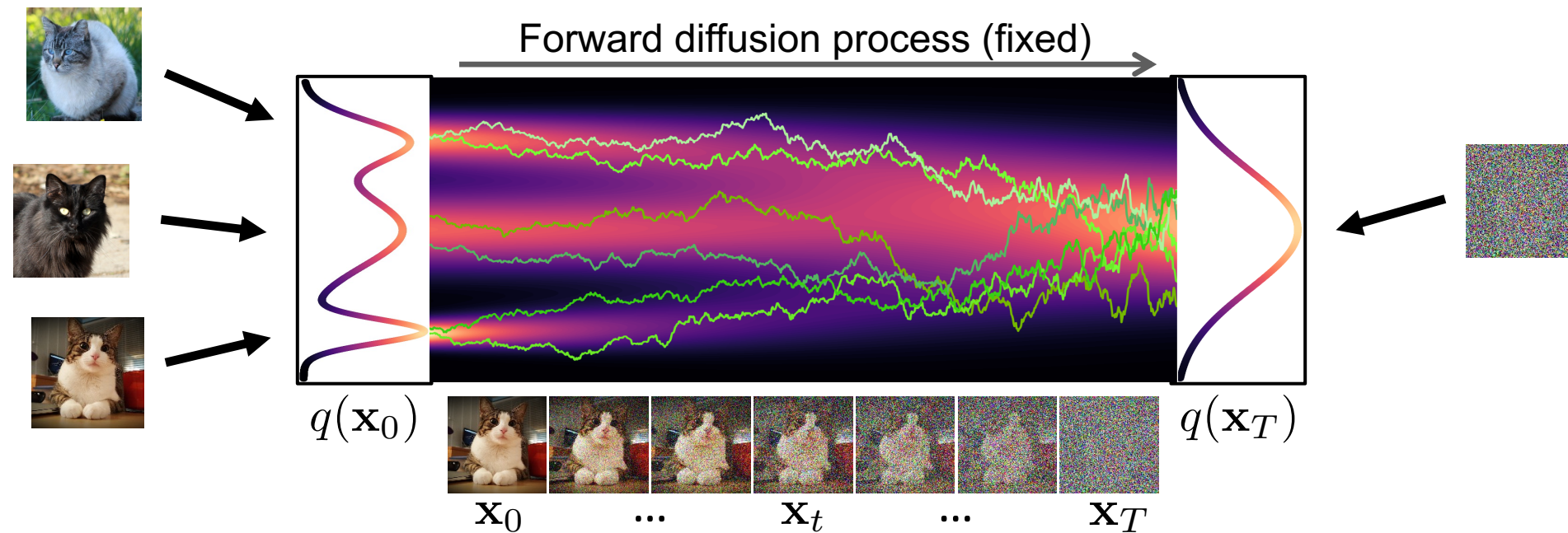
# Forward Diffusion Process as Stochastic Differential Equation



**Forward Diffusion SDE:**

$$d\mathbf{x}_t = \underbrace{-\frac{1}{2}\beta(t)\mathbf{x}_t dt}_{\text{drift term (pulls towards mode)}} + \underbrace{\sqrt{\beta(t)} d\omega_t}_{\text{diffusion term (injects noise)}}$$

# The Generative Reverse Stochastic Differential Equation



**Forward Diffusion SDE:**

$$d\mathbf{x}_t = \underbrace{-\frac{1}{2}\beta(t)\mathbf{x}_t}_{\text{drift term}} dt + \underbrace{\sqrt{\beta(t)} d\omega_t}_{\text{diffusion term}}$$

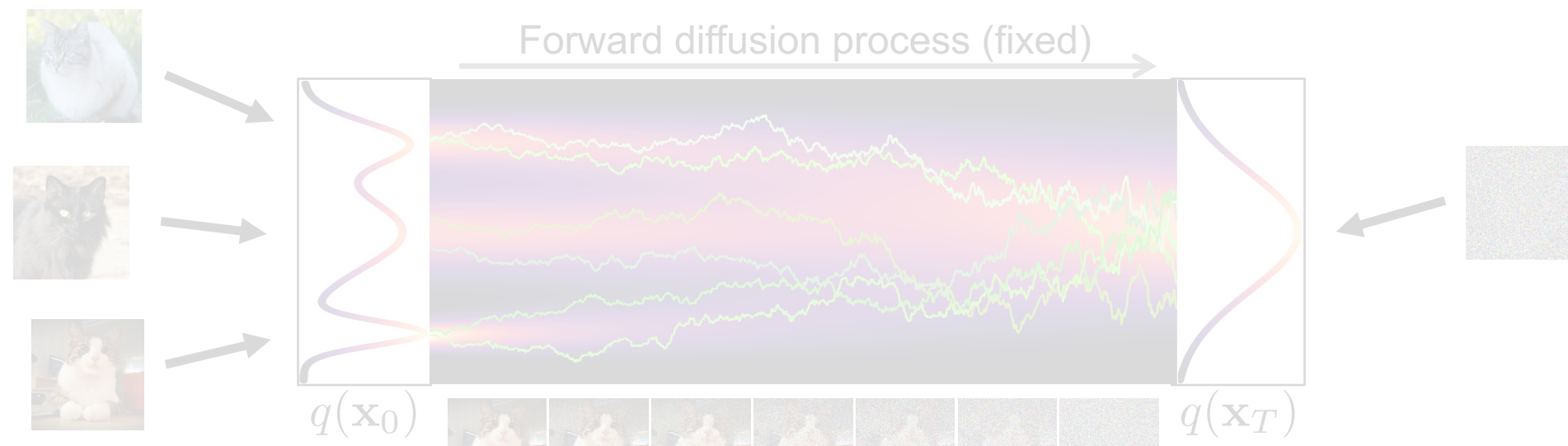
**Reverse Generative Diffusion SDE:**

$$d\mathbf{x}_t = \left[ \underbrace{-\frac{1}{2}\beta(t)\mathbf{x}_t}_{\text{drift term}} - \underbrace{\beta(t) \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)}_{\text{“Score Function”}} \right] dt + \underbrace{\sqrt{\beta(t)} d\bar{\omega}_t}_{\text{diffusion term}}$$

➔ **Simulate reverse diffusion process: Data generation from random noise!**



# The Generative Reverse Stochastic Differential Equation



**But how to get the score function  $\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)$ ?**

Forward Diffusion SDE:

$$d\mathbf{x}_t = -\frac{1}{2}\beta(t)\mathbf{x}_t dt + \sqrt{\beta(t)} d\omega_t$$

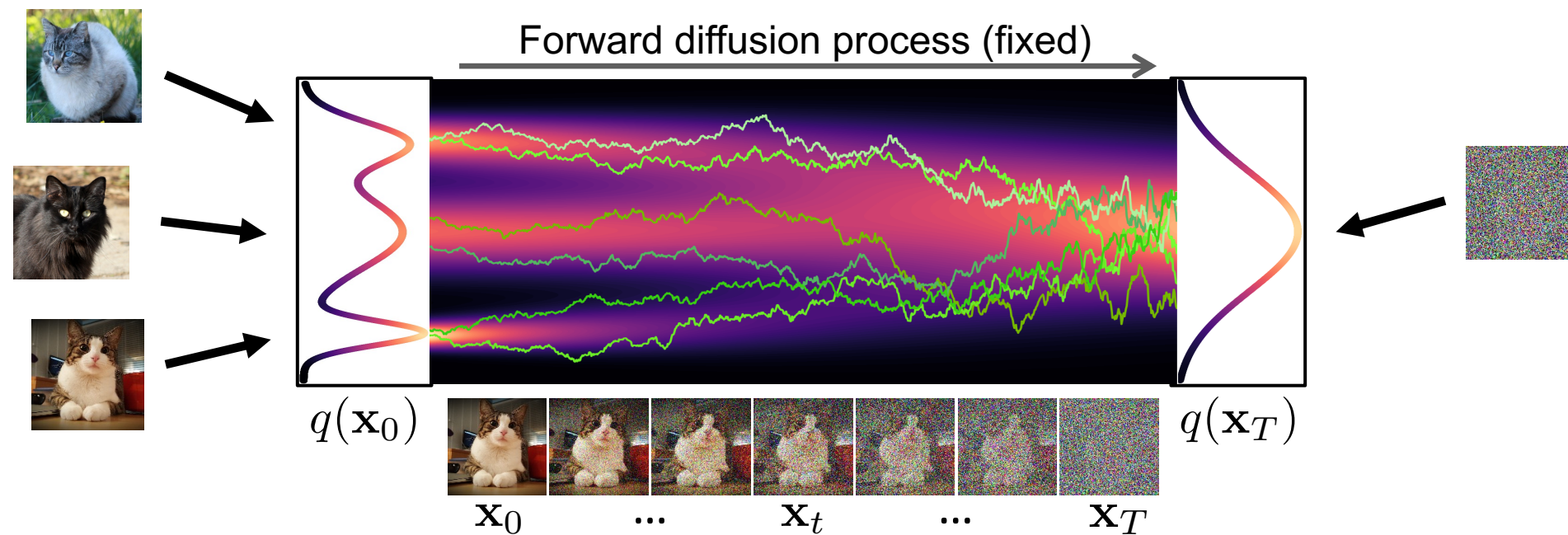
Reverse Generative Diffusion SDE:

$$d\mathbf{x}_t = \underbrace{\left[ -\frac{1}{2}\beta(t)\mathbf{x}_t - \beta(t) \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t) \right]}_{\text{drift term}} dt + \underbrace{\sqrt{\beta(t)} d\bar{\omega}_t}_{\text{diffusion term}}$$

"Score Function"

➔ Simulate reverse diffusion process: Data generation from random noise!

# Score Matching

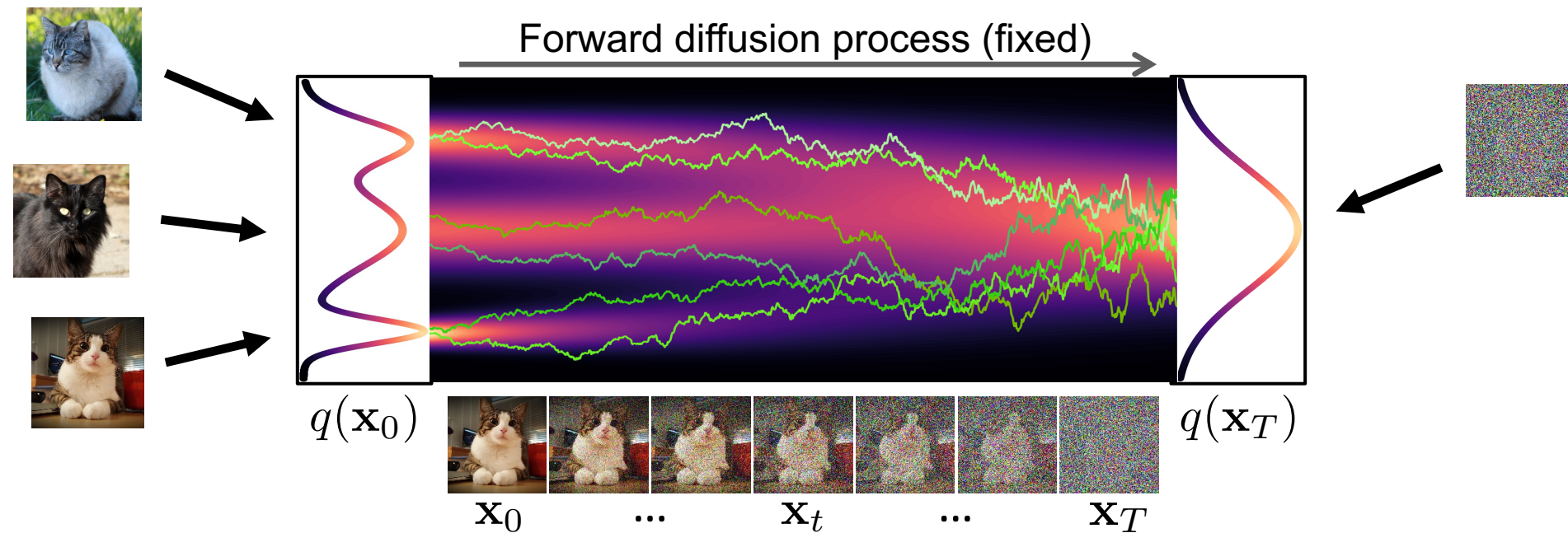


- Naïve idea, learn model for the score function by direct regression?

$$\min_{\theta} \underbrace{\mathbb{E}_{t \sim \mathcal{U}(0, T)}}_{\text{diffusion time } t} \underbrace{\mathbb{E}_{\mathbf{x}_t \sim q_t(\mathbf{x}_t)}}_{\text{diffused data } \mathbf{x}_t} \underbrace{\tilde{w}(t)}_{\text{weighting function}} \cdot \underbrace{\|\mathbf{s}_{\theta}(\mathbf{x}_t, t)\|_2}_{\text{neural network}} - \underbrace{\|\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)\|_2^2}_{\text{score of diffused data (marginal)}}$$

➔ **But  $\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)$  (score of the *marginal diffused density*  $q_t(\mathbf{x}_t)$ ) is not tractable!**

# Denoising Score Matching



- Instead, diffuse individual data points  $\mathbf{x}_0$ . Diffused  $q_t(\mathbf{x}_t|\mathbf{x}_0)$  *is* tractable!
- **Denoising Score Matching:**

$$\min_{\theta} \underbrace{\mathbb{E}_{t \sim \mathcal{U}(0, T)}}_{\text{diffusion time } t} \underbrace{\mathbb{E}_{\mathbf{x}_0 \sim q_0(\mathbf{x}_0)}}_{\text{data sample } \mathbf{x}_0} \underbrace{\mathbb{E}_{\mathbf{x}_t \sim q_t(\mathbf{x}_t|\mathbf{x}_0)}}_{\text{diffused data sample } \mathbf{x}_t} \underbrace{\tilde{w}(t)}_{\text{weighting function}} \cdot \underbrace{\|s_{\theta}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t|\mathbf{x}_0)\|_2^2}_{\text{neural network score of diffused data sample}}$$

➔ **After expectations,  $s_{\theta}(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)$ !**

[Vincent, in \*Neural Computation\*, 2011](#)

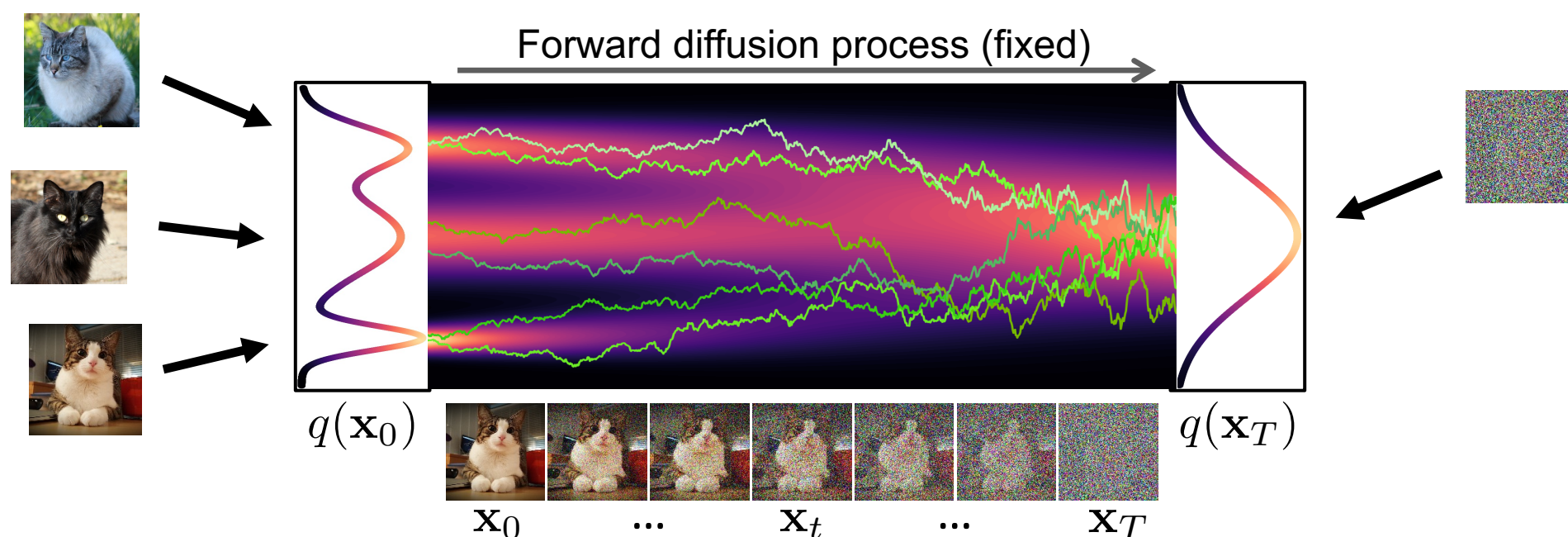
[Song and Ermon, \*NeurIPS\*, 2019](#)

[Song et al. \*ICLR\*, 2021](#)

slide from <https://cvpr2022-tutorial-diffusion-models.github.io/>

# Denoising Score Matching

## Epsilon-prediction parametrization



- Denoising Score Matching:**

$$\min_{\theta} \mathbb{E}_{t \sim \mathcal{U}(0, T)} \mathbb{E}_{\mathbf{x}_0 \sim q_0(\mathbf{x}_0)} \mathbb{E}_{\mathbf{x}_t \sim q_t(\mathbf{x}_t | \mathbf{x}_0)} \tilde{w}(t) \cdot \|\mathbf{s}_{\theta}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t | \mathbf{x}_0)\|_2^2$$

- Re-parametrized sampling:  $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \epsilon \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

- Score function:  $\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t | \mathbf{x}_0) = -\nabla_{\mathbf{x}_t} \frac{(\mathbf{x}_t - \alpha_t \mathbf{x}_0)^2}{2\sigma_t^2} = -\frac{\mathbf{x}_t - \alpha_t \mathbf{x}_0}{\sigma_t^2} = -\frac{\alpha_t \mathbf{x}_0 + \sigma_t \epsilon - \alpha_t \mathbf{x}_0}{\sigma_t^2} = -\frac{\epsilon}{\sigma_t}$

- Neural network model:  $\mathbf{s}_{\theta}(\mathbf{x}_t, t) := -\frac{\epsilon_{\theta}(\mathbf{x}_t, t)}{\sigma_t}$

$$\rightarrow \min_{\theta} \mathbb{E}_{t \sim \mathcal{U}(0, T)} \mathbb{E}_{\mathbf{x}_0 \sim q_0(\mathbf{x}_0)} \mathbb{E}_{\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \hat{w}(t) \cdot \|\epsilon - \epsilon_{\theta}(\mathbf{x}_t, t)\|_2^2 \quad \hat{w}(t) = \frac{\tilde{w}(t)}{\sigma_t}$$

[Vincent, in \*Neural Computation\*, 2011](#)

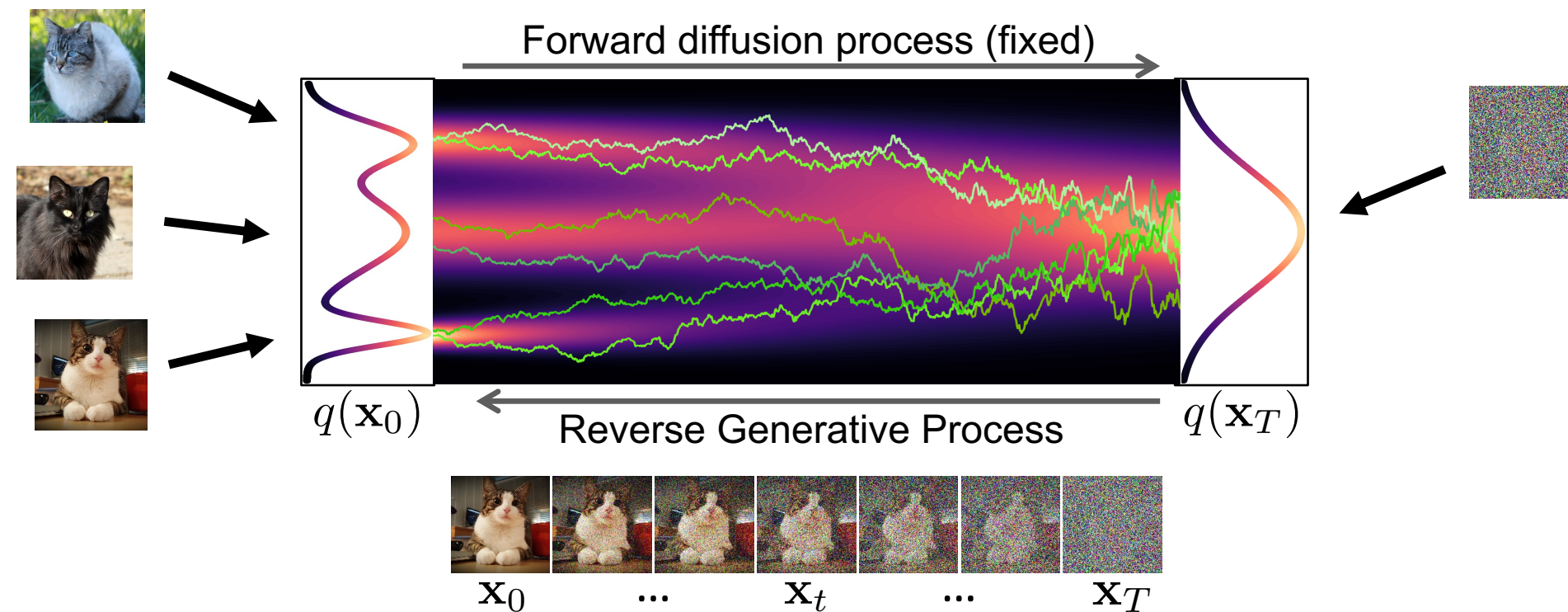
[Song and Ermon, \*NeurIPS\*, 2019](#)

[Song et al. \*ICLR\*, 2021](#)

slide from <https://cvpr2022-tutorial-diffusion-models.github.io/>

# Probability Flow ODE

## Alternative reverse process

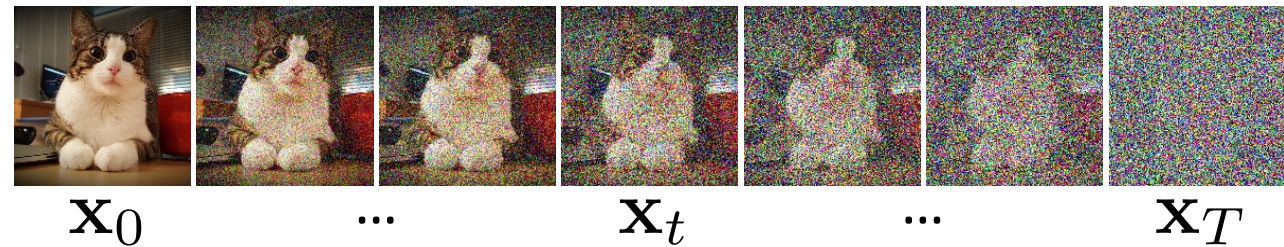
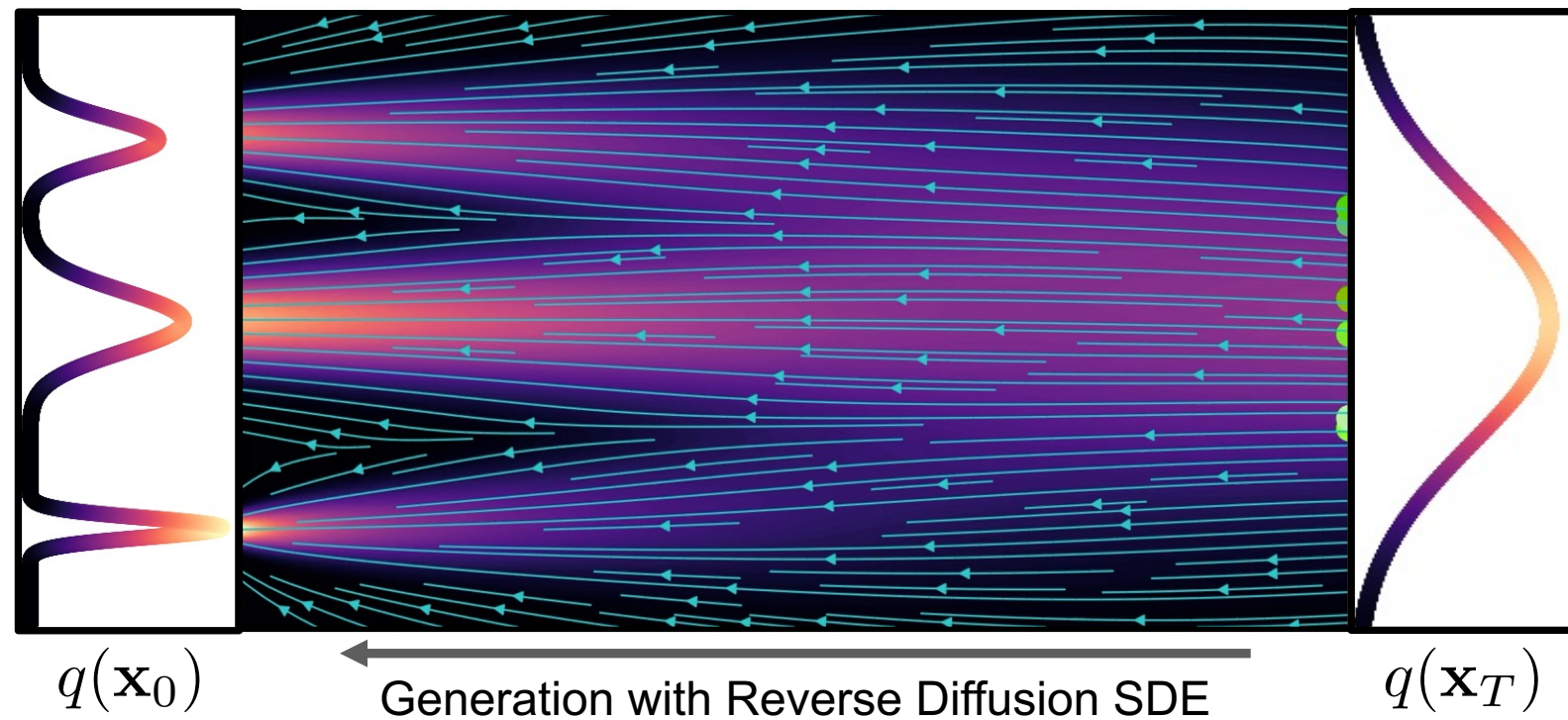


- Consider reverse generative diffusion SDE:
 
$$d\mathbf{x}_t = -\frac{1}{2}\beta(t) [\mathbf{x}_t + 2\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)] dt + \sqrt{\beta(t)} d\bar{\omega}_t$$
- In distribution equivalent to "Probability Flow ODE":
 
$$d\mathbf{x}_t = -\frac{1}{2}\beta(t) [\mathbf{x}_t + \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)] dt$$

(solving this ODE results in the same  $q_t(\mathbf{x}_t)$  when initializing  $q_T(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$ )

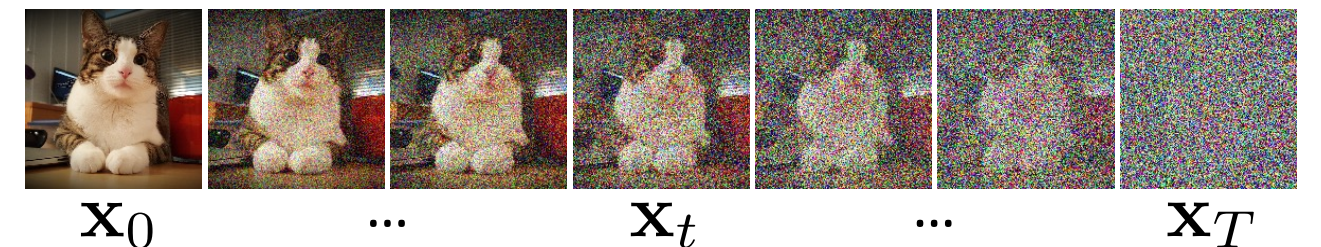
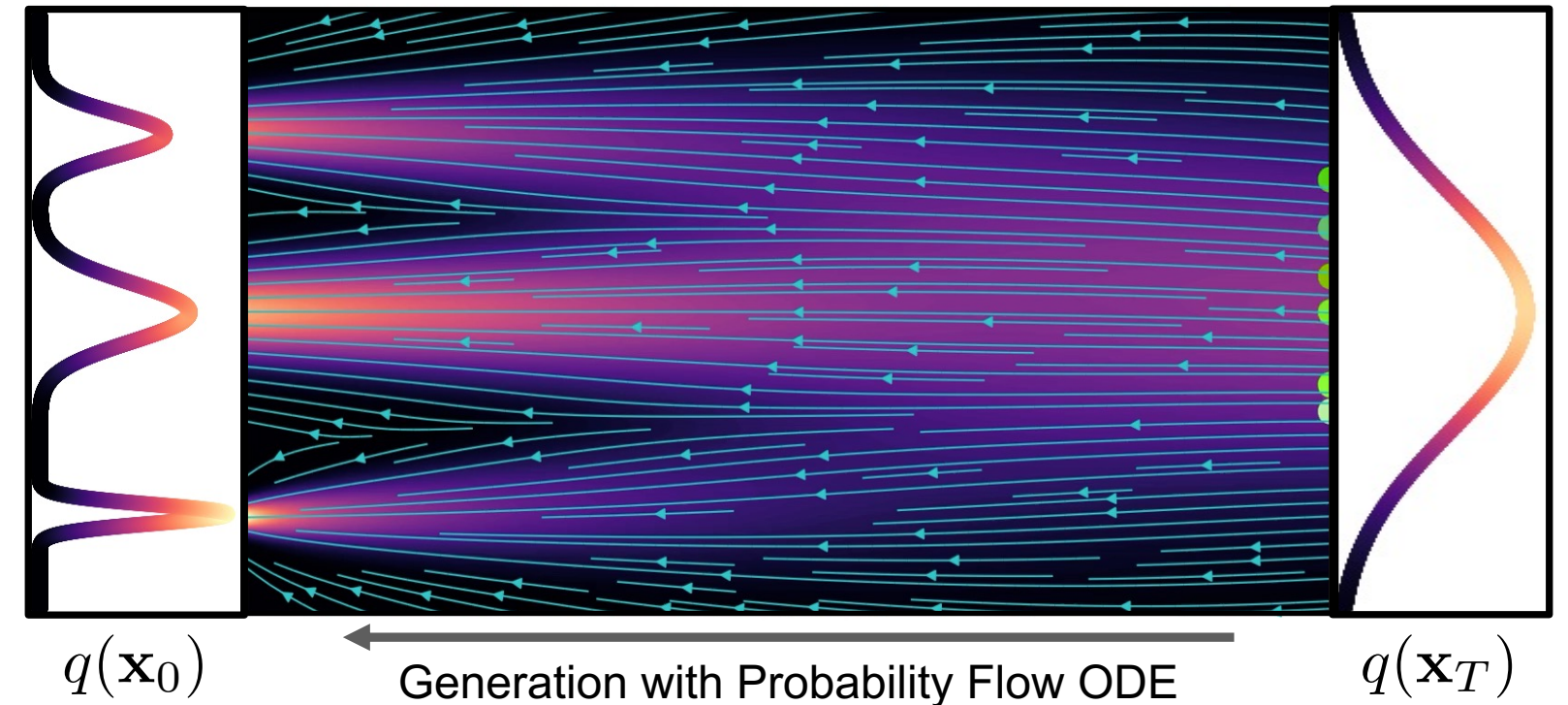
Deterministic mapping from  $\mathbf{x}_T$  to  $\mathbf{x}_0$

# Synthesis with SDE vs. ODE



- **Generative Reverse Diffusion SDE (stochastic):**

$$d\mathbf{x}_t = -\frac{1}{2}\beta(t) [\mathbf{x}_t + 2\mathbf{s}_\theta(\mathbf{x}_t, t)] dt + \sqrt{\beta(t)} d\bar{\omega}_t$$

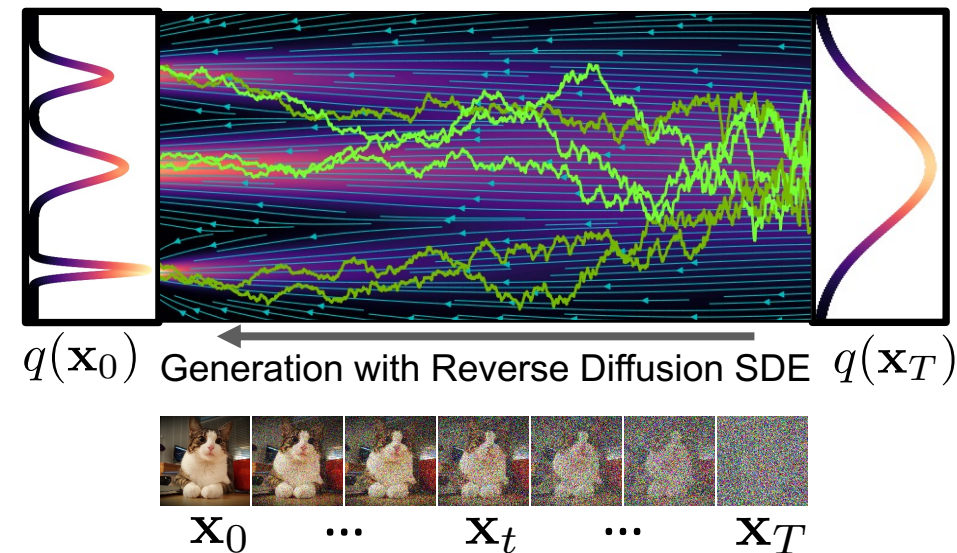


- **Generative Probability Flow ODE (deterministic):**

$$d\mathbf{x}_t = -\frac{1}{2}\beta(t) [\mathbf{x}_t + \mathbf{s}_\theta(\mathbf{x}_t, t)] dt$$

# Sampling from “Continuous-Time” Diffusion Models

## SDE vs. ODE Sampling: Pro’s and Con’s

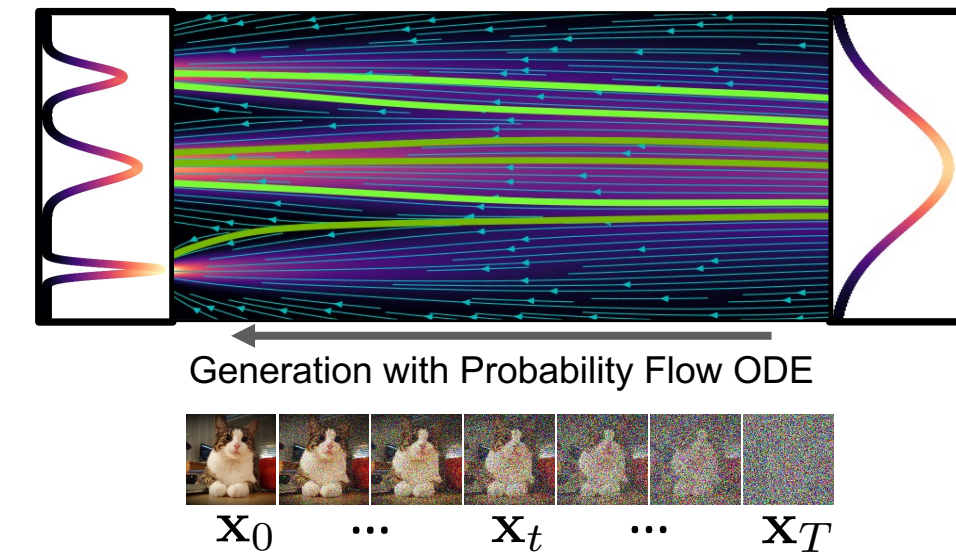


### Generative Diffusion SDE:

$$d\mathbf{x}_t = -\frac{1}{2}\beta(t) [\mathbf{x}_t + 2\mathbf{s}_\theta(\mathbf{x}_t, t)] dt + \sqrt{\beta(t)} d\bar{\omega}_t$$

$$d\mathbf{x}_t = \underbrace{-\frac{1}{2}\beta(t) [\mathbf{x}_t + \mathbf{s}_\theta(\mathbf{x}_t, t)] dt}_{\text{Probability Flow ODE}} + \underbrace{-\frac{1}{2}\beta(t)\mathbf{s}_\theta(\mathbf{x}_t, t)dt + \sqrt{\beta(t)} d\bar{\omega}_t}_{\text{Langevin dynamics}}$$

- ➔ **Pro:** Continuous noise injection can help to compensate errors during diffusion process (Langevin sampling actively pushes towards correct distribution).
- ➔ **Con:** Often slower, because the stochastic terms themselves require fine discretization during solve.



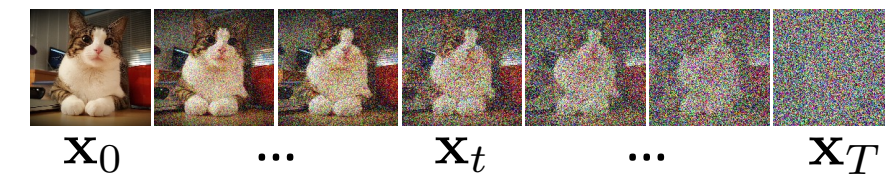
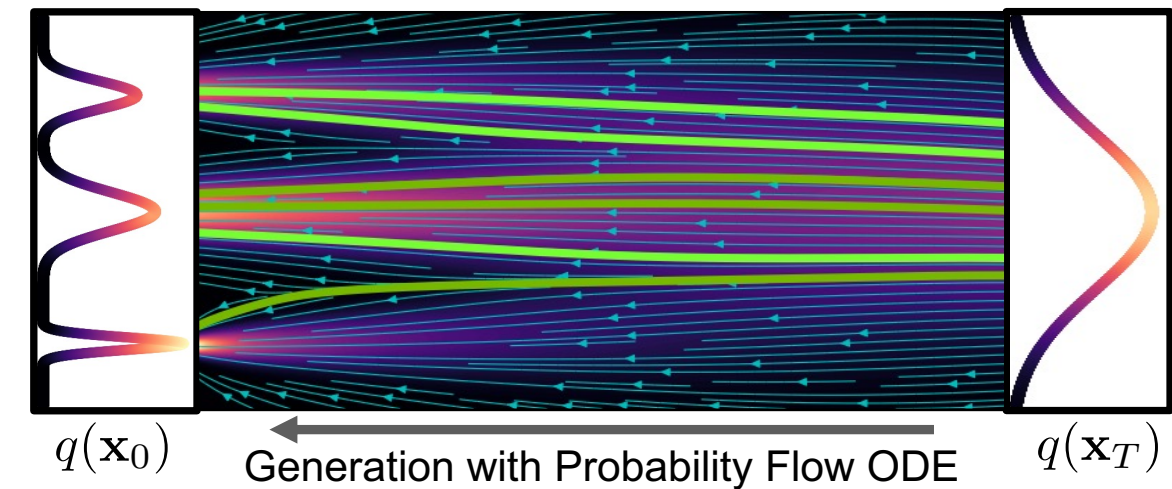
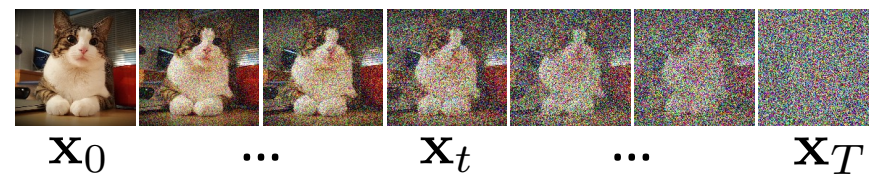
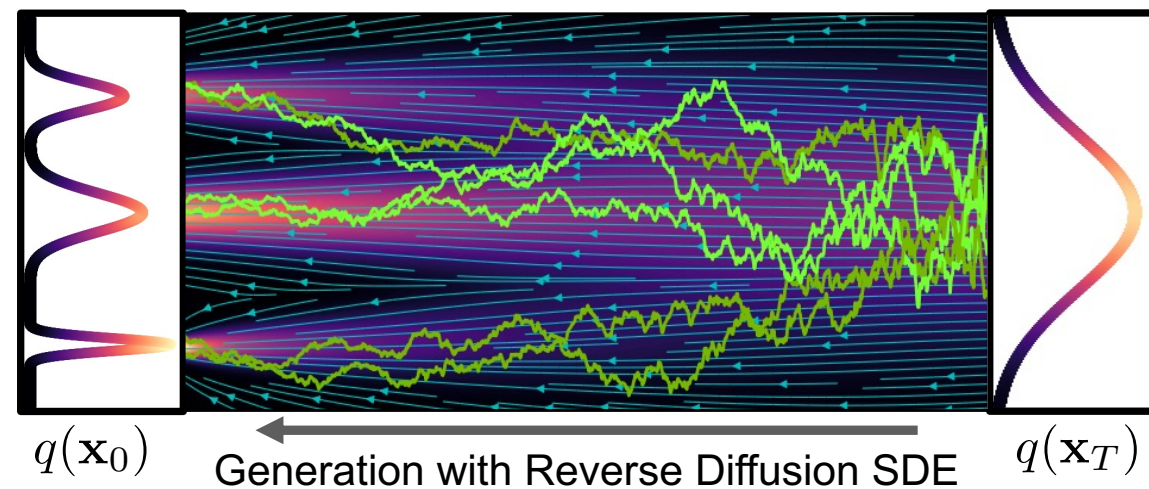
### Probability Flow ODE:

$$d\mathbf{x}_t = -\frac{1}{2}\beta(t) [\mathbf{x}_t + \mathbf{s}_\theta(\mathbf{x}_t, t)] dt$$

- ➔ **Pro:** Can leverage fast ODE solvers. Best when targeting very fast sampling.
- ➔ **Con:** No “stochastic” error correction, often slightly lower performance than stochastic sampling.

# Sampling from “Continuous-Time” Diffusion Models

How to solve the generative SDE or ODE in practice?



**Generative Diffusion SDE:**

$$d\mathbf{x}_t = -\frac{1}{2}\beta(t) [\mathbf{x}_t + 2\mathbf{s}_\theta(\mathbf{x}_t, t)] dt + \sqrt{\beta(t)} d\bar{\omega}_t$$

➔ *Euler-Maruyama:*

$$\mathbf{x}_{t-1} = \mathbf{x}_t + \frac{1}{2}\beta(t) [\mathbf{x}_t + 2\mathbf{s}_\theta(\mathbf{x}_t, t)] \Delta t + \sqrt{\beta(t)\Delta t} \mathcal{N}(\mathbf{0}, \mathbf{I})$$

➔ *Ancestral Sampler* (discrete-time) is also a generative SDE sampler!

**Probability Flow ODE:**

$$d\mathbf{x}_t = -\frac{1}{2}\beta(t) [\mathbf{x}_t + \mathbf{s}_\theta(\mathbf{x}_t, t)] dt$$

➔ *Euler’s Method:*

$$\mathbf{x}_{t-1} = \mathbf{x}_t + \frac{1}{2}\beta(t) [\mathbf{x}_t + \mathbf{s}_\theta(\mathbf{x}_t, t)] \Delta t$$

➔ In practice: DDIM sampler, another solver of the ODE.



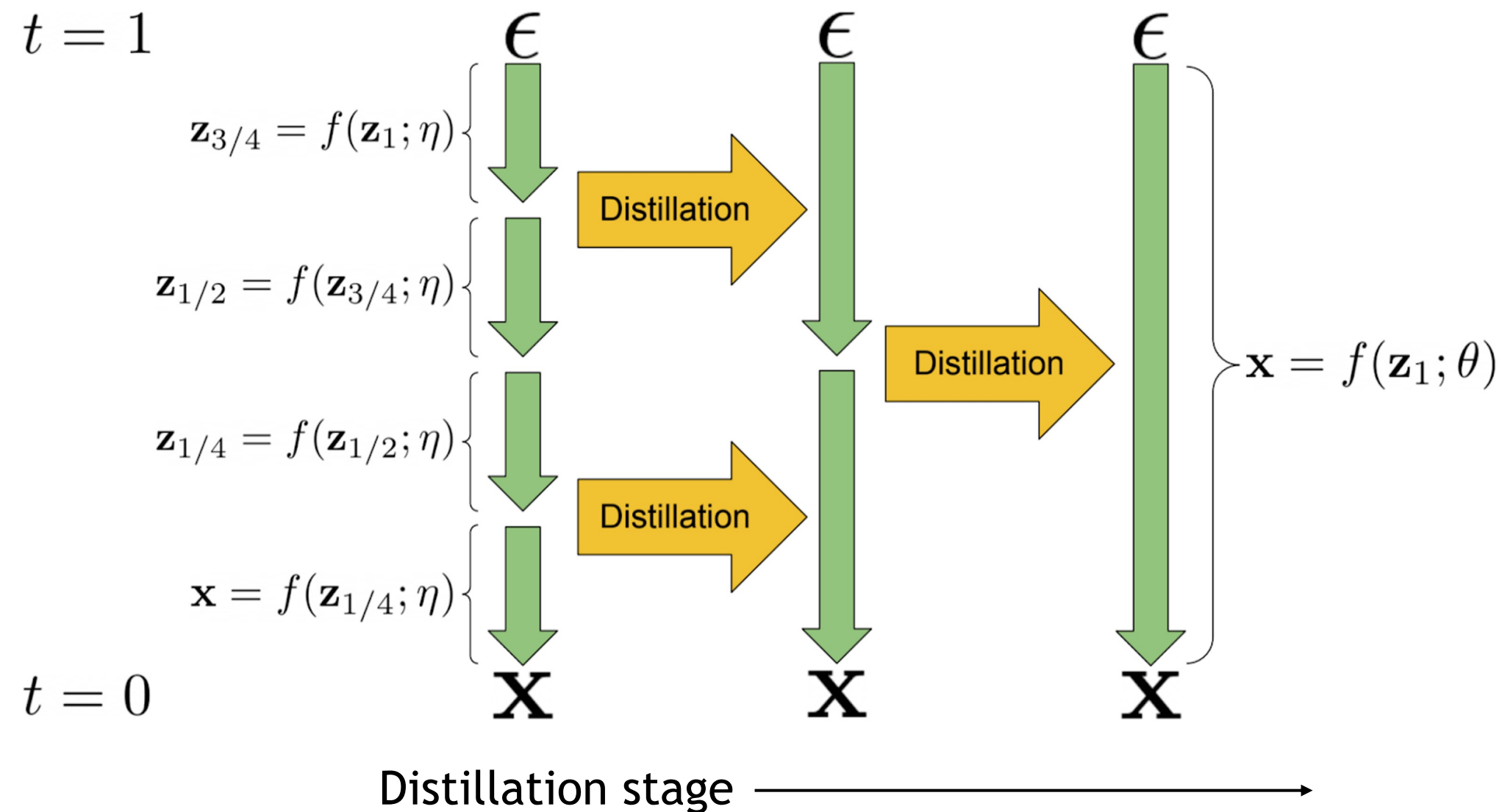
# How to make sampling faster?

- One bottleneck of diffusion models is its slowness in sampling: need 10-1000+ steps to generate high quality samples
- Generative models need to be fast for practical use.
- One solution: distill diffusion models into models using just 4-8 sampling steps!
  - *Progressive distillation for fast sampling of diffusion models, Salimans & Ho, ICLR 2022*
  - *On Distillation of Guided Diffusion Models, Meng et al., CVPR 2023*

# Progressive distillation

## How to make sampling faster?

- Distill a deterministic ODE sampler (i.e. DDIM sampler) to the same model architecture.
- At each stage, a “student” model is learned to distill two adjacent sampling steps of the “teacher” model to one sampling step.
- At next stage, the “student” model from previous stage will serve as the new “teacher” model.



**Algorithm 1** Standard diffusion training**Require:** Model  $\hat{\mathbf{x}}_{\theta}(\mathbf{z}_t)$  to be trained**Require:** Data set  $\mathcal{D}$ **Require:** Loss weight function  $w()$ **while** not converged **do** $\mathbf{x} \sim \mathcal{D}$   $\triangleright$  Sample data $t \sim U[0, 1]$   $\triangleright$  Sample time $\epsilon \sim N(0, I)$   $\triangleright$  Sample noise $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$   $\triangleright$  Add noise to data $\tilde{\mathbf{x}} = \mathbf{x}$   $\triangleright$  Clean data is target for  $\hat{\mathbf{x}}$  $\lambda_t = \log[\alpha_t^2 / \sigma_t^2]$   $\triangleright$  log-SNR $L_{\theta} = w(\lambda_t) \|\tilde{\mathbf{x}} - \hat{\mathbf{x}}_{\theta}(\mathbf{z}_t)\|_2^2$   $\triangleright$  Loss $\theta \leftarrow \theta - \gamma \nabla_{\theta} L_{\theta}$   $\triangleright$  Optimization**end while****Algorithm 2** Progressive distillation**Require:** Trained teacher model  $\hat{\mathbf{x}}_{\eta}(\mathbf{z}_t)$ **Require:** Data set  $\mathcal{D}$ **Require:** Loss weight function  $w()$ **Require:** Student sampling steps  $N$ **for**  $K$  iterations **do** $\theta \leftarrow \eta$   $\triangleright$  Init student from teacher**while** not converged **do** $\mathbf{x} \sim \mathcal{D}$  $t = i/N, i \sim \text{Cat}[1, 2, \dots, N]$  $\epsilon \sim N(0, I)$  $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$ 

# 2 steps of DDIM with teacher

 $t' = t - 0.5/N, t'' = t - 1/N$  $\mathbf{z}_{t'} = \alpha_{t'} \hat{\mathbf{x}}_{\eta}(\mathbf{z}_t) + \frac{\sigma_{t'}}{\sigma_t} (\mathbf{z}_t - \alpha_t \hat{\mathbf{x}}_{\eta}(\mathbf{z}_t))$  $\mathbf{z}_{t''} = \alpha_{t''} \hat{\mathbf{x}}_{\eta}(\mathbf{z}_{t'}) + \frac{\sigma_{t''}}{\sigma_{t'}} (\mathbf{z}_{t'} - \alpha_{t'} \hat{\mathbf{x}}_{\eta}(\mathbf{z}_{t'}))$  $\tilde{\mathbf{x}} = \frac{\mathbf{z}_{t''} - (\sigma_{t''}/\sigma_t)\mathbf{z}_t}{\alpha_{t''} - (\sigma_{t''}/\sigma_t)\alpha_t}$   $\triangleright$  Teacher  $\hat{\mathbf{x}}$  target $\lambda_t = \log[\alpha_t^2 / \sigma_t^2]$  $L_{\theta} = w(\lambda_t) \|\tilde{\mathbf{x}} - \hat{\mathbf{x}}_{\theta}(\mathbf{z}_t)\|_2^2$  $\theta \leftarrow \theta - \gamma \nabla_{\theta} L_{\theta}$ **end while** $\eta \leftarrow \theta$   $\triangleright$  Student becomes next teacher $N \leftarrow N/2$   $\triangleright$  Halve number of sampling steps**end for**

# On Distillation of Guided Diffusion Models

Meng et al., CVPR 2023 award nominated

Now also works with

- CF-Guidance
- Stochastic sampling
- Text-to-image/video
- Image-to-image
- Inpainting
- Latent Diffusion



Text-guided generation (1 step)



Text-guided generation (4 steps)



Text-guided generation (2 steps)



Class-conditional generation (1 step)



Input Mask Result 1 Result 2

Image inpainting (2 steps)



Input Output (different styles)

Image to image translation (3 steps)