

EECS 545: Machine Learning

Lecture 18. Deep Generative Models

Honglak Lee

03/19/2024



Outline

- **Generative Models Basics**
- Autoregressive Models
- Autoencoder and Variational Autoencoder
- Generative Adversarial Network
- Diffusion Models

Generative Models

Given training data, generate new samples from same distribution



Training data $\sim p_{\text{data}}(x)$



Generated samples $\sim p_{\text{model}}(x)$

Want to learn $p_{\text{model}}(x)$ similar to $p_{\text{data}}(x)$

Generative Models

Given training data, generate new samples from same distribution



Training data $\sim p_{\text{data}}(x)$



Generated samples $\sim p_{\text{model}}(x)$

Want to learn $p_{\text{model}}(x)$ similar to $p_{\text{data}}(x)$

Addresses density estimation, a core problem in unsupervised learning

Several flavors:

- Explicit density estimation: explicitly define and solve for $p_{\text{model}}(x)$
- Implicit density estimation: learn model that can sample from $p_{\text{model}}(x)$ w/o explicitly defining it

Why Generative Models?

- Realistic samples for artwork, super-resolution, colorization, etc.



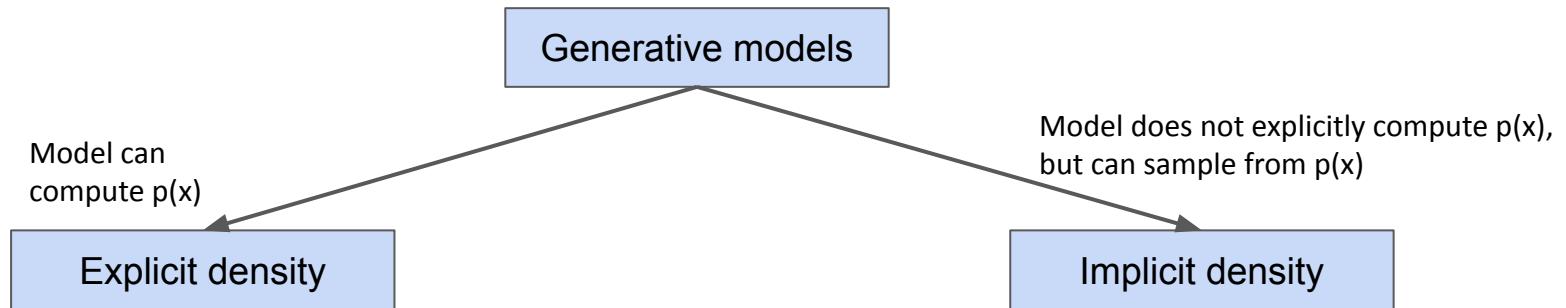
- Generative models of time-series data can be used for simulation and planning (reinforcement learning applications!)
- Training generative models can also enable inference of latent representations that can be useful as general features

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

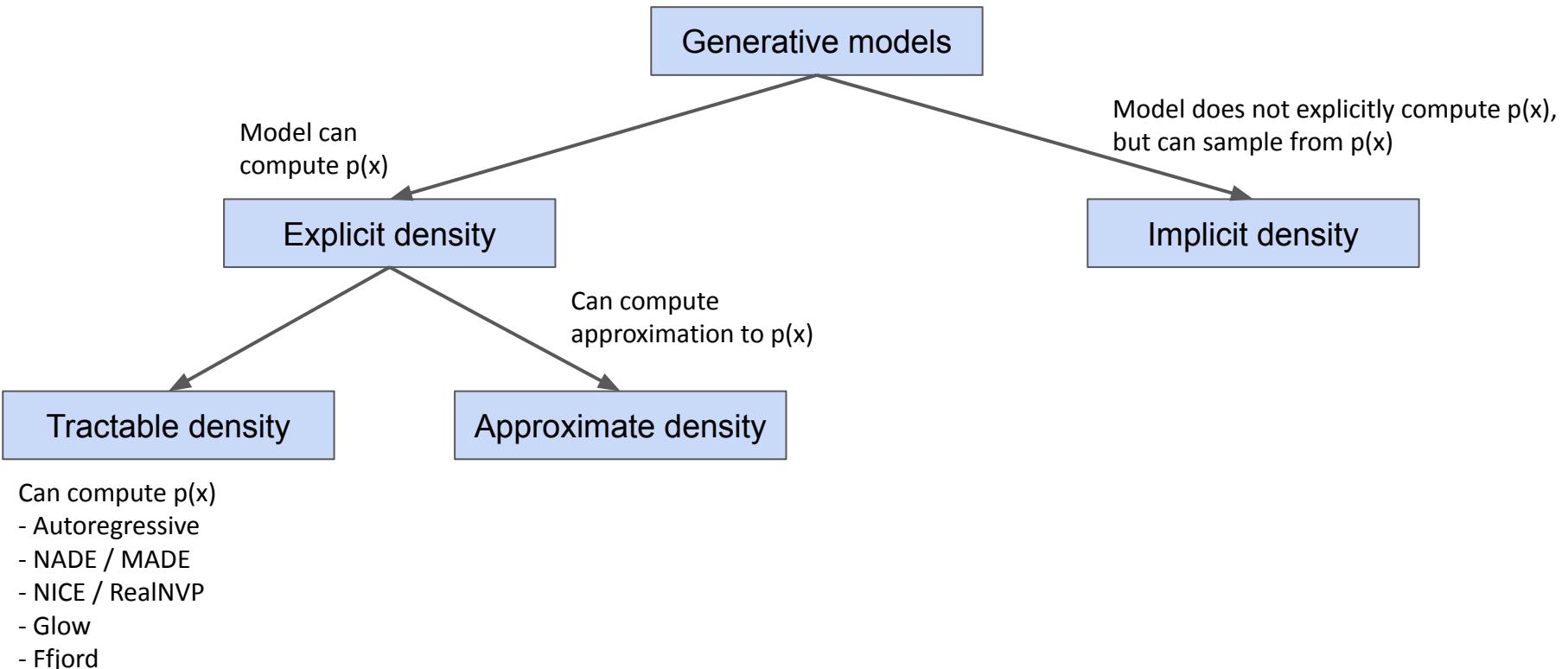
Taxonomy of Generative Models

Generative models

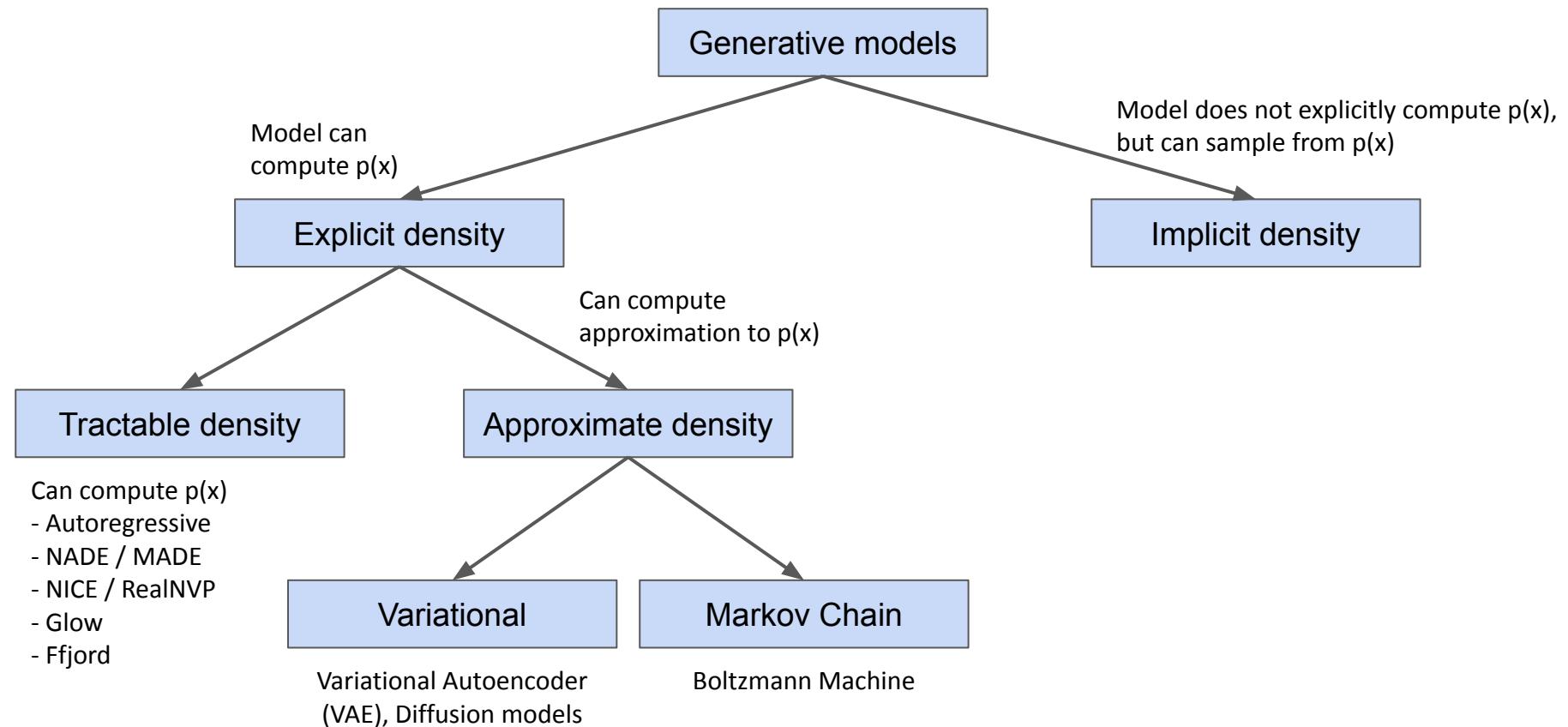
Taxonomy of Generative Models



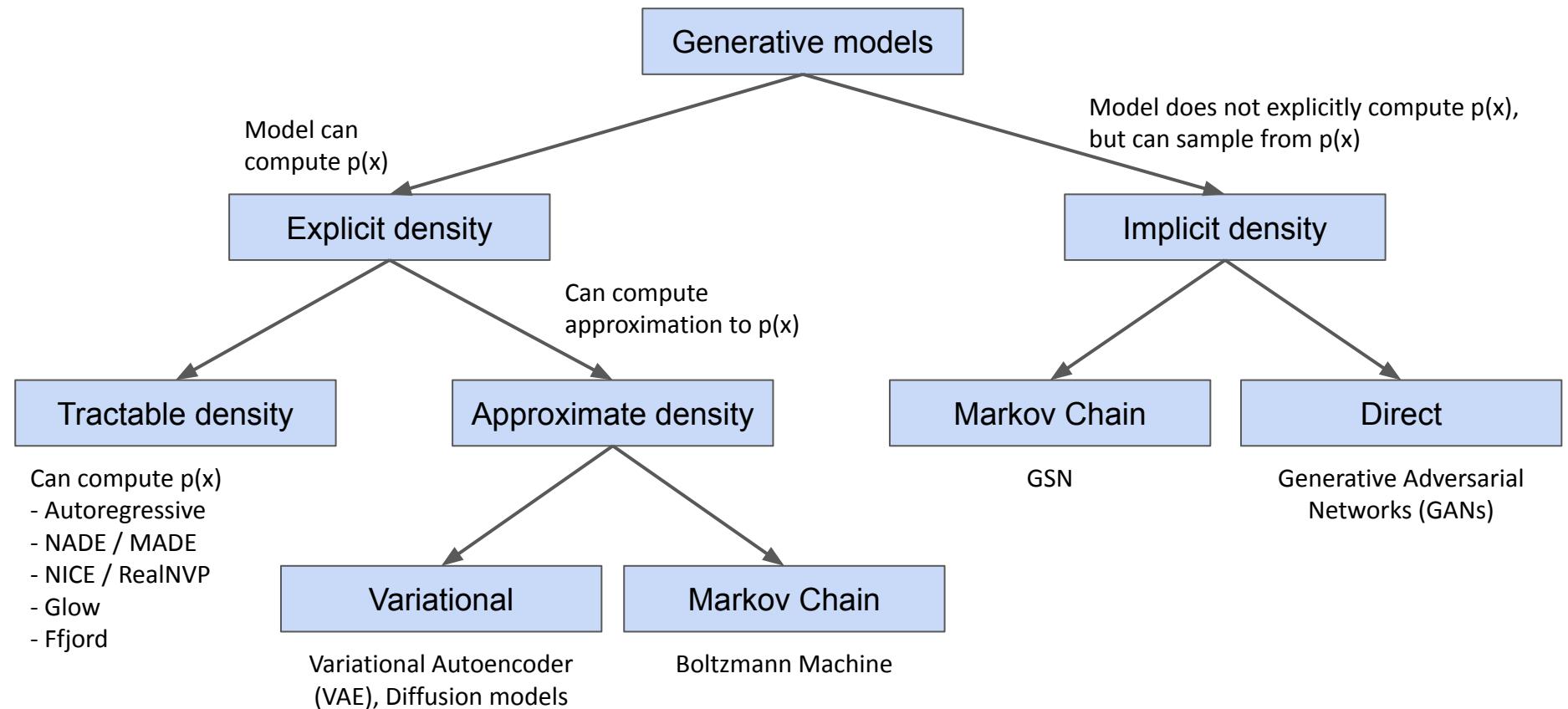
Taxonomy of Generative Models



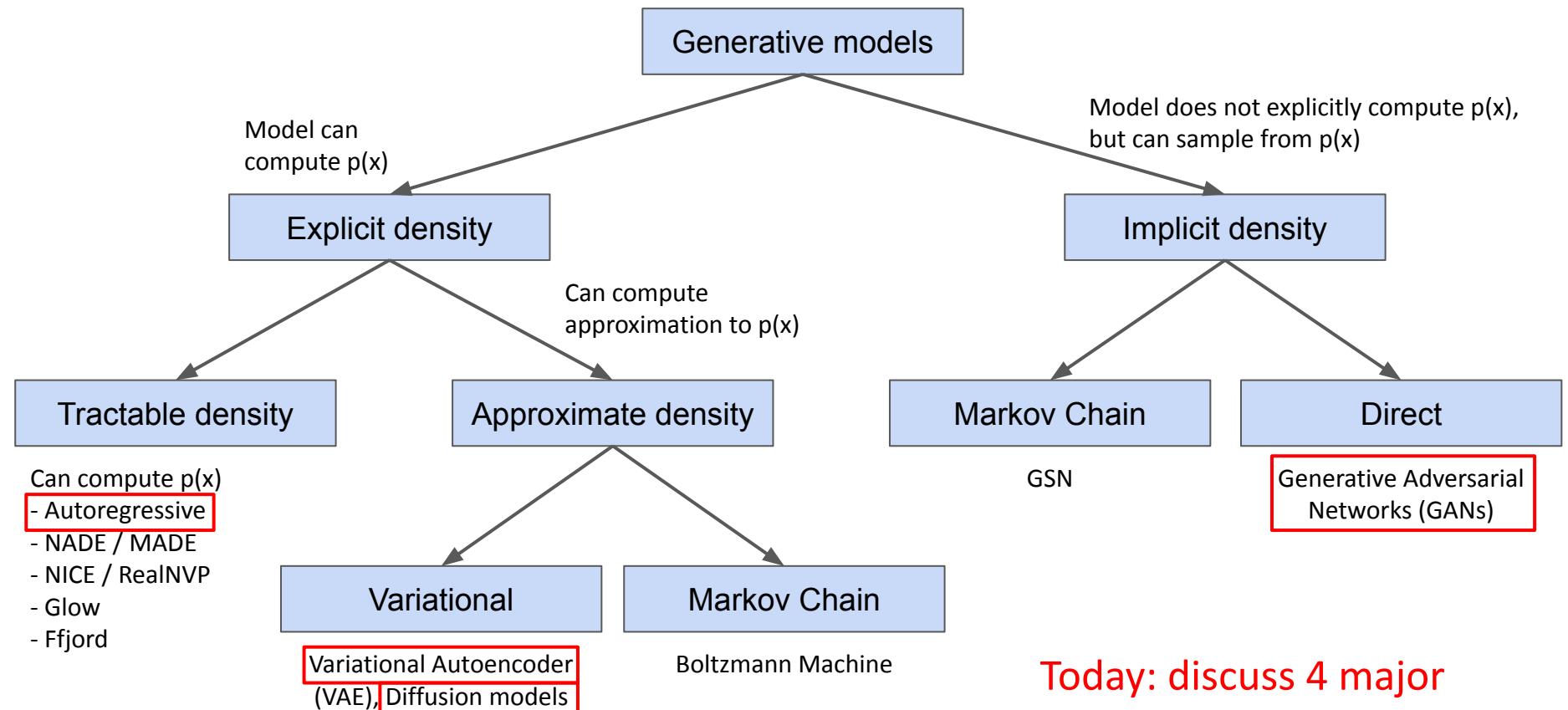
Taxonomy of Generative Models



Taxonomy of Generative Models



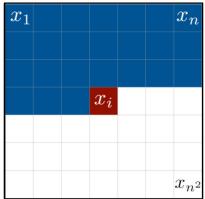
Taxonomy of Generative Models



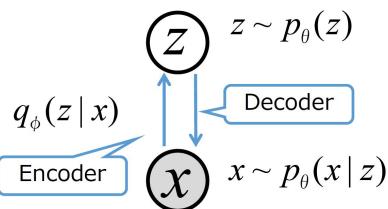
Today: discuss 4 major types of generative models

Image Generation Models

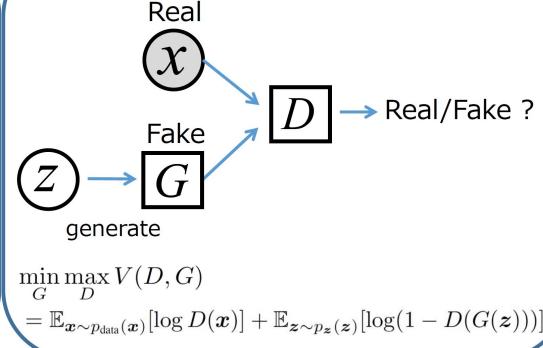
Autoregressive Models



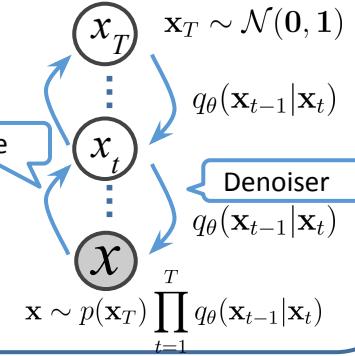
Variational AutoEncoders (VAE)



Generative Adversarial Networks (GAN)



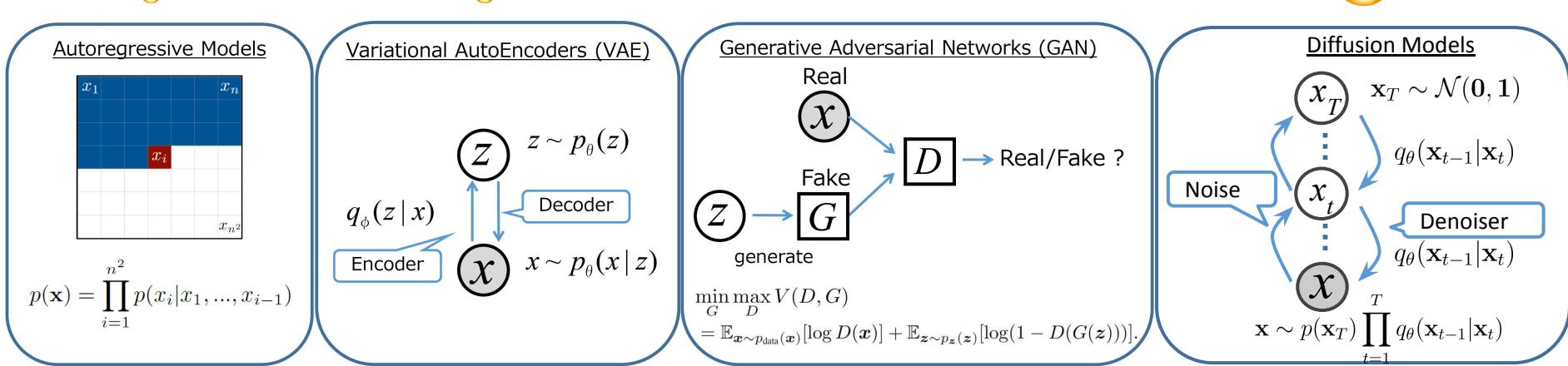
Diffusion Models



| | Autoregressive Models | VAE | GAN | Diffusion Models |
|------|---|---|---|---|
| Pros | simple and stable training process Currently gives the best log likelihood Tractable likelihood | Efficient inference with approximate latent variables | Generate sharp image No need for any Markov Chain or approx networks during sampling | Simple and stable training process Best samples! |
| Cons | Relatively inefficient during sampling | Generated samples tend to be blurry | Difficult to optimize due to unstable training dynamics | Relatively inefficient during sampling |

Image Generation Models

Current research interest in
image generation:



| | Autoregressive Models | VAE | GAN | Diffusion Models |
|------|---|---|---|---|
| Pros | simple and stable training process Currently gives the best log likelihood Tractable likelihood | Efficient inference with approximate latent variables | Generate sharp image No need for any Markov Chain or approx networks during sampling | Simple and stable training process Best samples! |
| Cons | Relatively inefficient during sampling | Generated samples tend to be blurry | Difficult to optimize due to unstable training dynamics | Relatively inefficient during sampling |

Outline

- Generative Models Basics
- **Autoregressive Models**
- Autoencoder and Variational Autoencoder
- Generative Adversarial Network
- Diffusion Models

Fully visible belief network

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 Probability of i'th pixel value
image x given all previous pixels

Then maximize likelihood of training data

Fully visible belief network

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

Fully visible belief network

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

Will need to define ordering of “previous pixels”

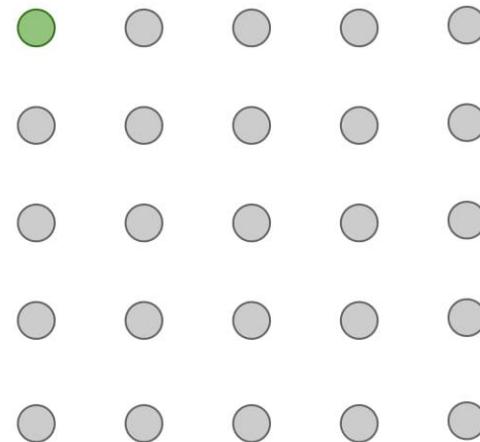
Complex distribution over pixel values => Express using a neural network!

Then maximize likelihood of training data

PixelRNN

[van der Oord et al. 2016]

Generate image pixels starting from corner



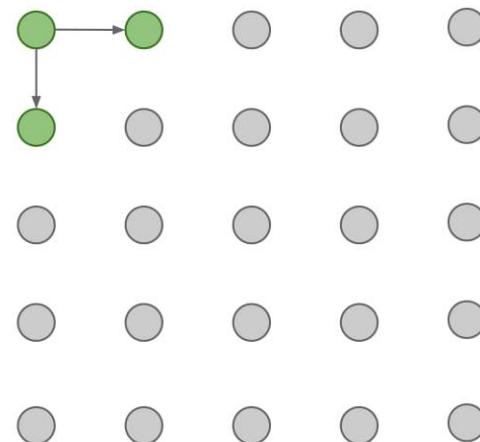
Dependency on previous pixels modeled
using an RNN (LSTM)

PixelRNN

[van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled
using an RNN (LSTM)

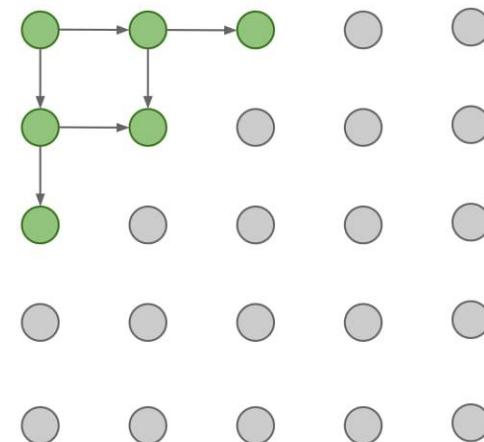


PixelRNN

[van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)



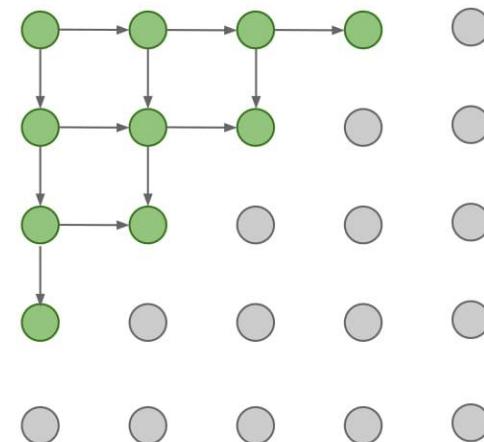
PixelRNN

[van der Oord et al. 2016]

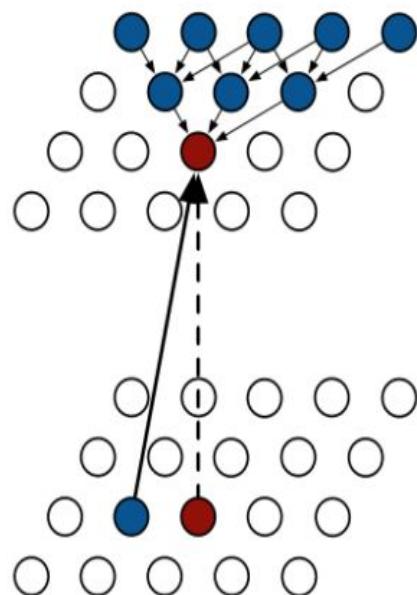
Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

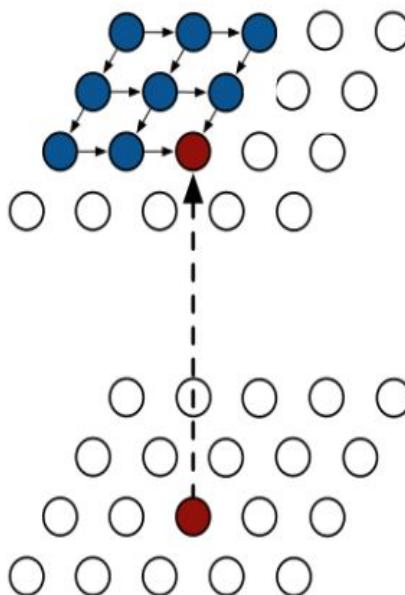
Drawback: sequential generation is slow!



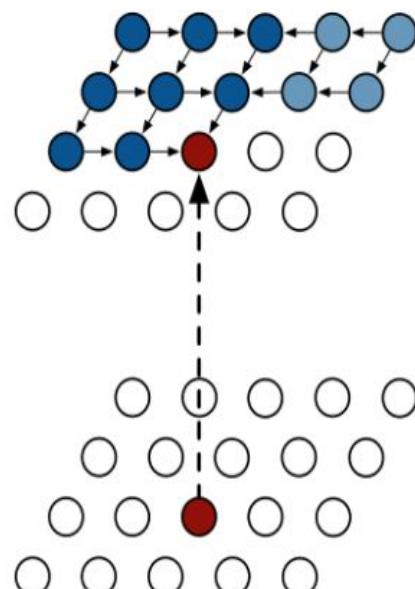
Variants of Pixel RNN (LSTM)



Row LSTM



Diagonal LSTM



Diagonal BiLSTM

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

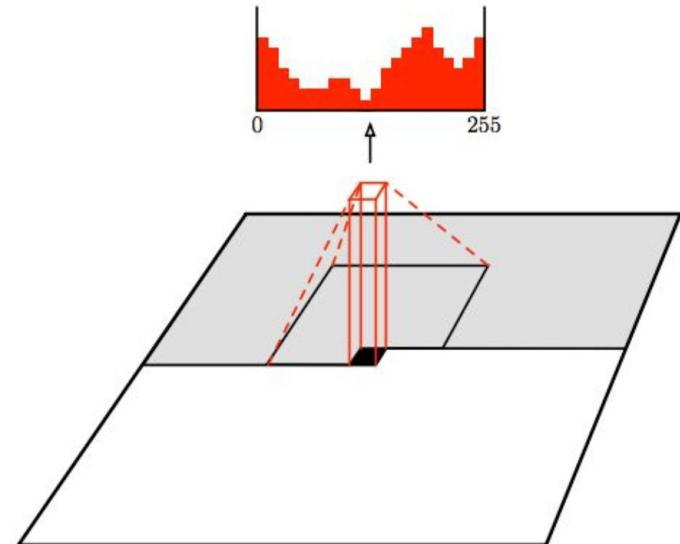


Figure copyright van der Oord et al., 2016. Reproduced with permission.

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

Training: maximize likelihood of training images

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

Softmax loss at each pixel

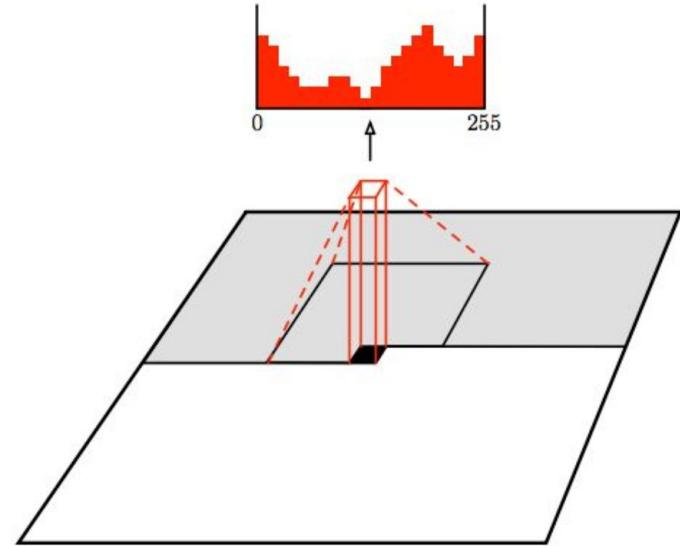


Figure copyright van der Oord et al., 2016. Reproduced with permission.

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

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

Generation must still proceed sequentially
=> still slow

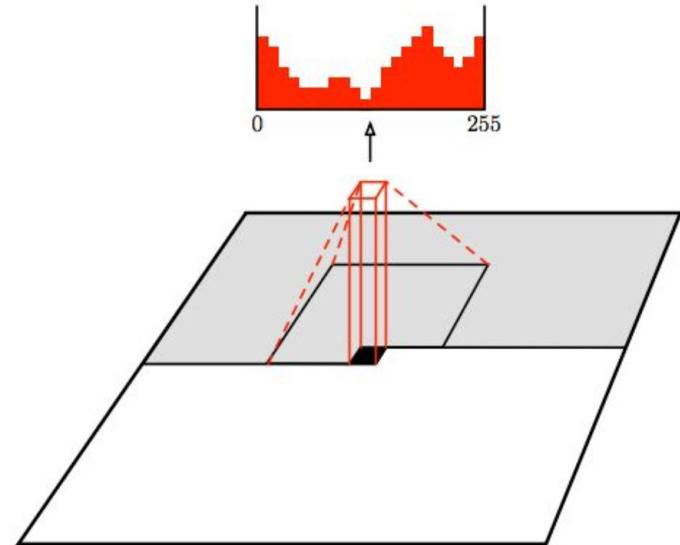
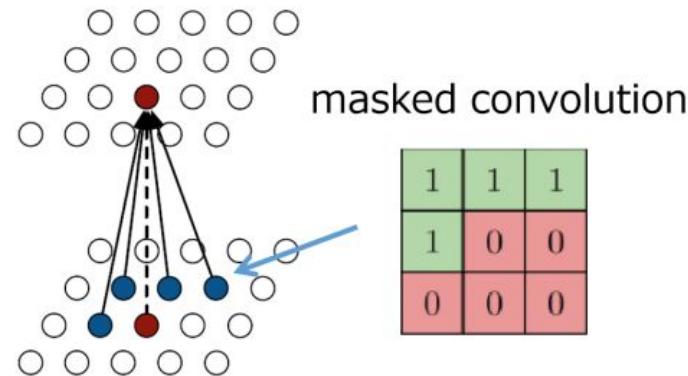


Figure copyright van der Oord et al., 2016. Reproduced with permission.

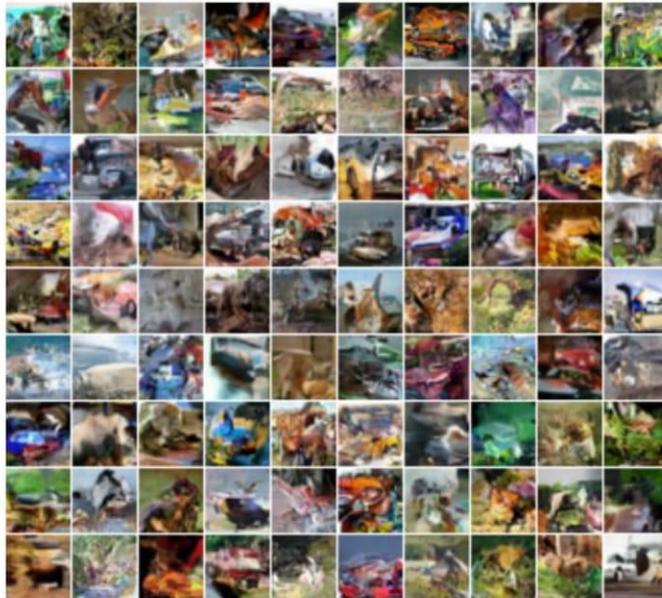
PixelCNN

- 2D convolution on previous layer
- Apply masks so a pixel does not see future pixels (in sequential order)

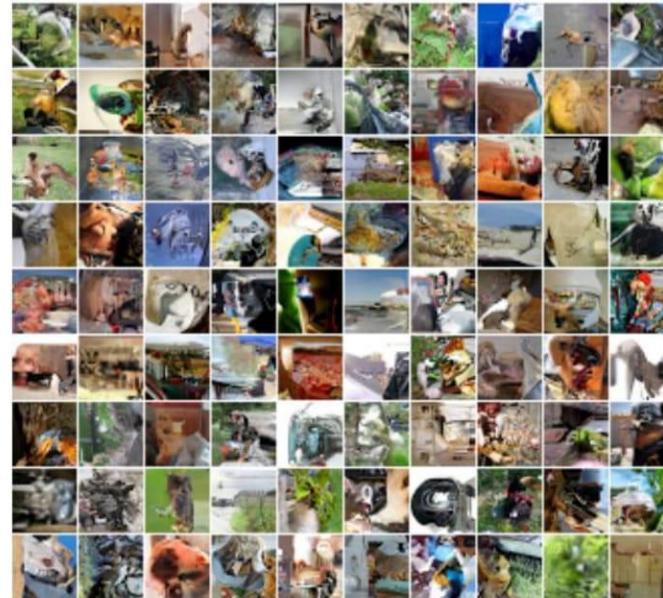


[Pixel recurrent neural networks](#), ICML 2016

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 $p(x)$
- Explicit likelihood of training data gives good evaluation metric
- 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++)

Outline

- Generative Models Basics
- Autoregressive Models
- **Autoencoder and Variational Autoencoder**
- Generative Adversarial Network
- Diffusion Models

So far...

PixelCNNs 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...

PixelCNNs define tractable density function, optimize likelihood of training data:

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

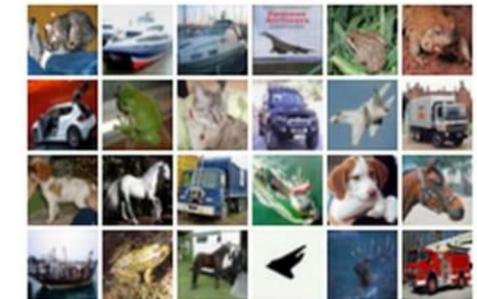
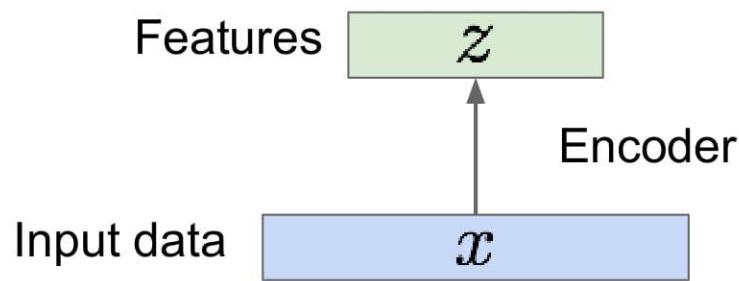
VAEs define intractable density function with latent \mathbf{z} :

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

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

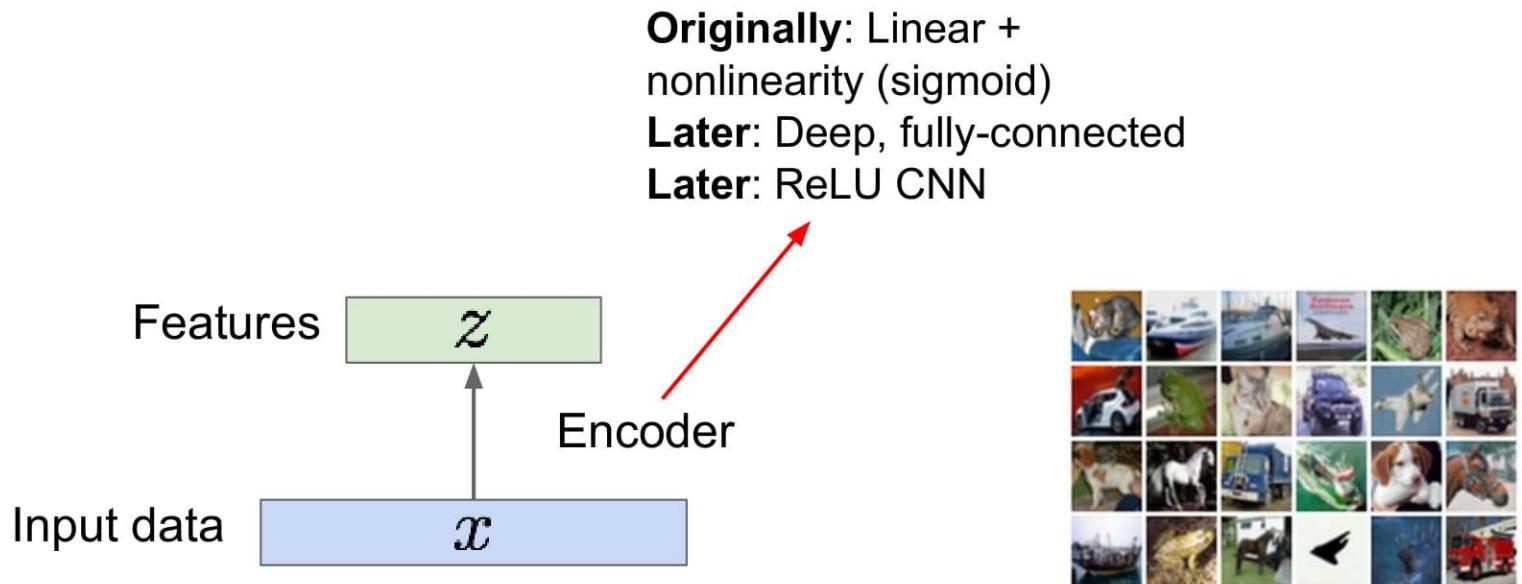
Some background first: Autoencoders

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

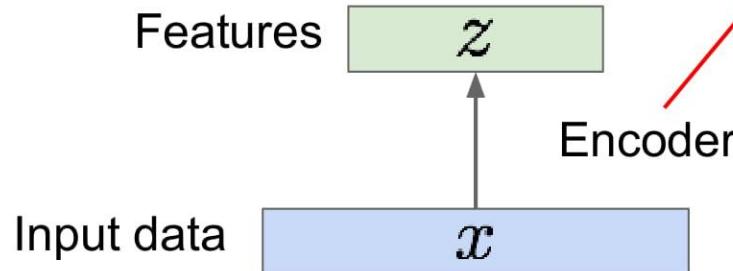


Some background first: Autoencoders

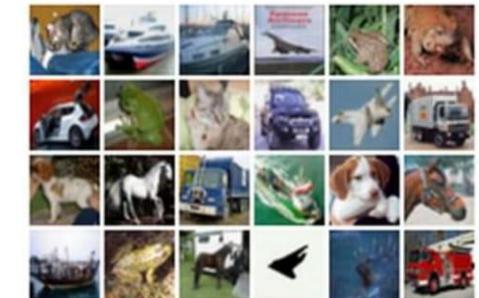
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than x
(dimensionality reduction)

Q: Why dimensionality reduction?



Originally: Linear +
nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN



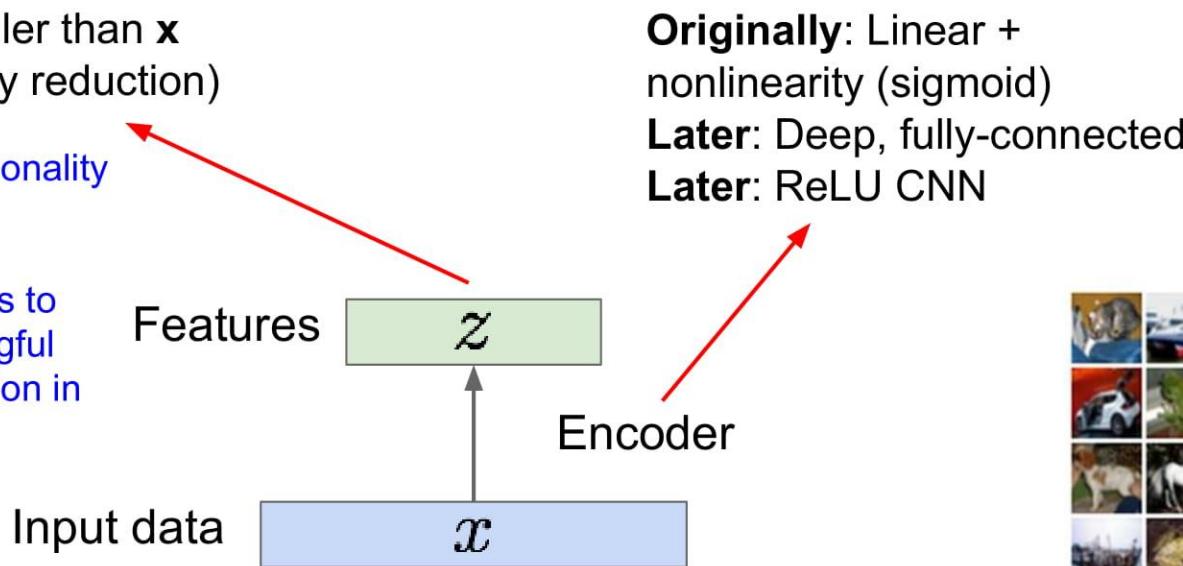
Some background first: Autoencoders

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

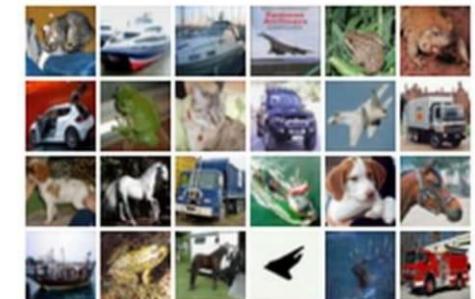
z usually smaller than x
(dimensionality reduction)

Q: Why dimensionality reduction?

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

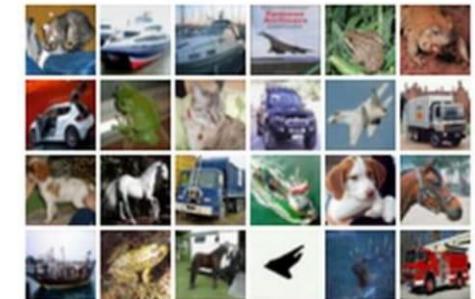
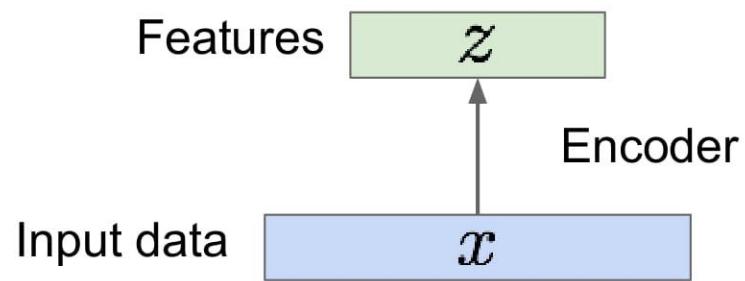


Originally: Linear +
nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN



Some background first: Autoencoders

How to learn this feature representation?

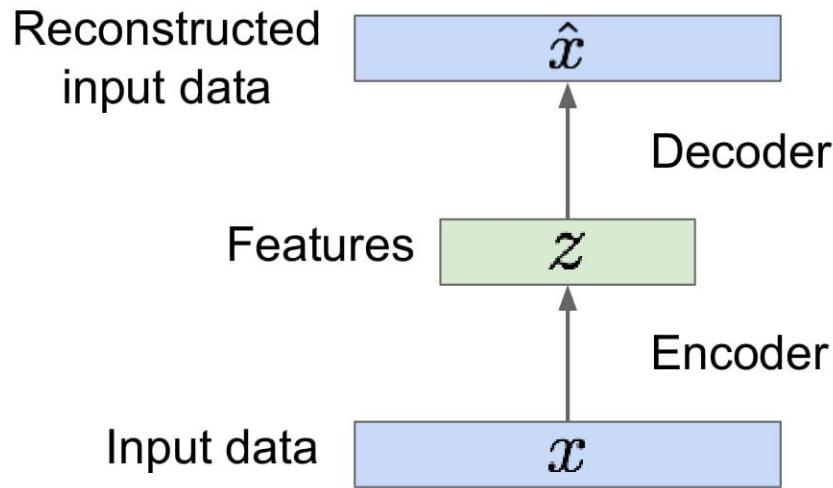


Some background first: Autoencoders

How to learn this feature representation?

Train such that features can be used to reconstruct original data

“Autoencoding” - encoding itself

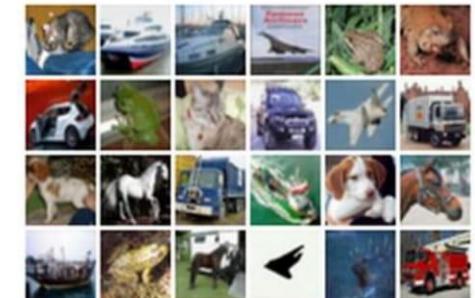
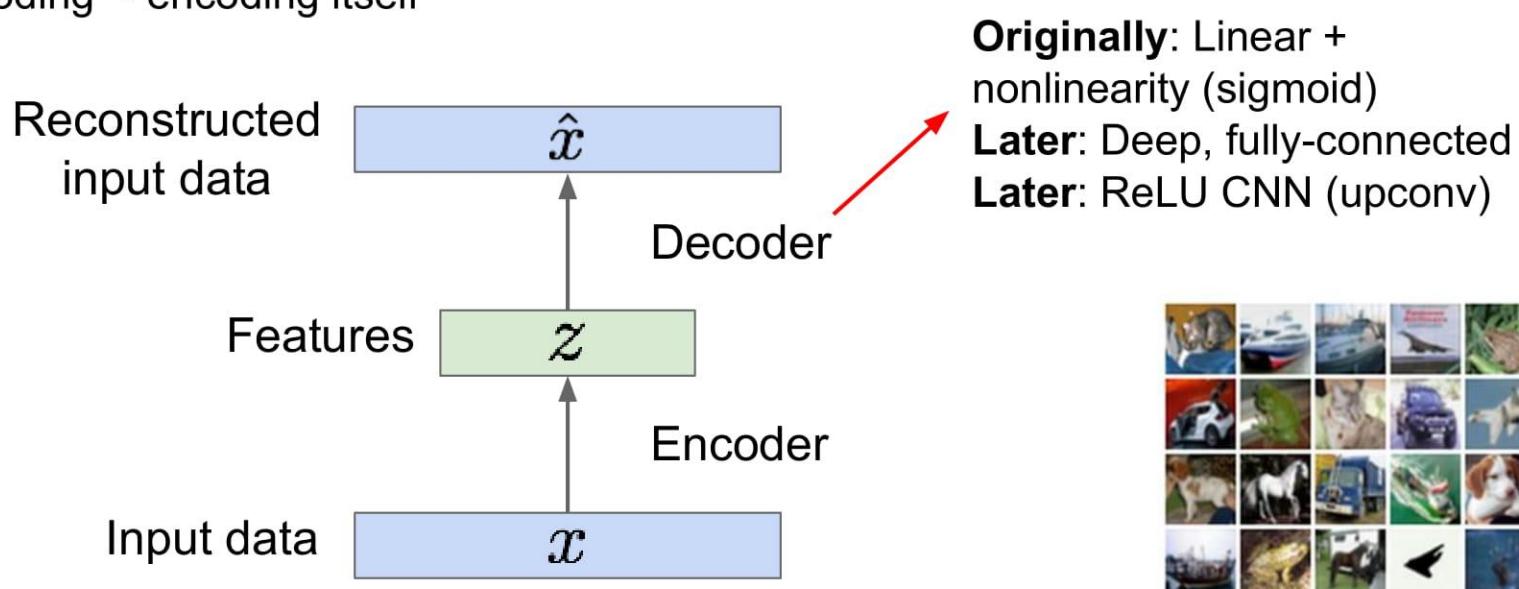


Some background first: Autoencoders

How to learn this feature representation?

Train such that features can be used to reconstruct original data

“Autoencoding” - encoding itself

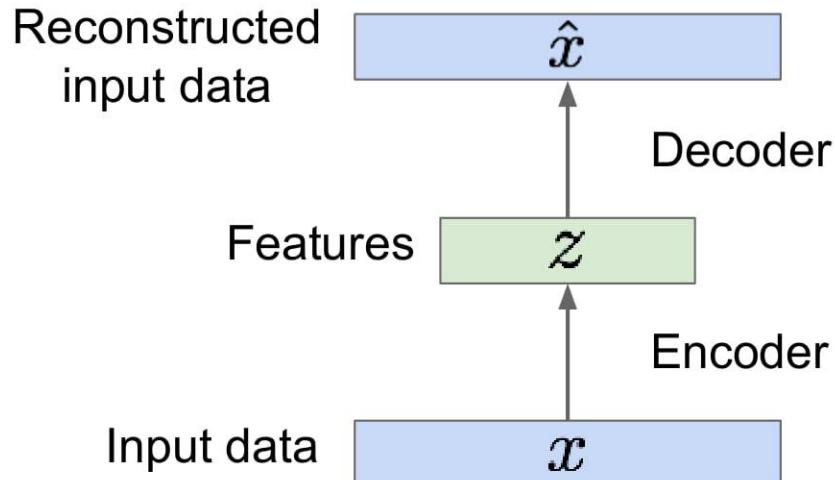


Some background first: Autoencoders

How to learn this feature representation?

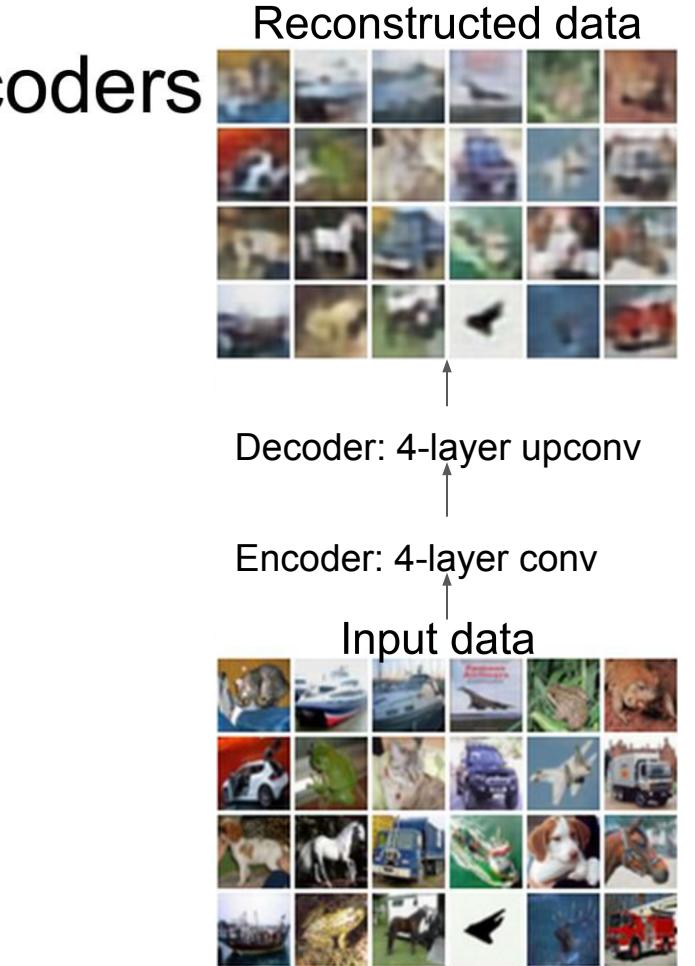
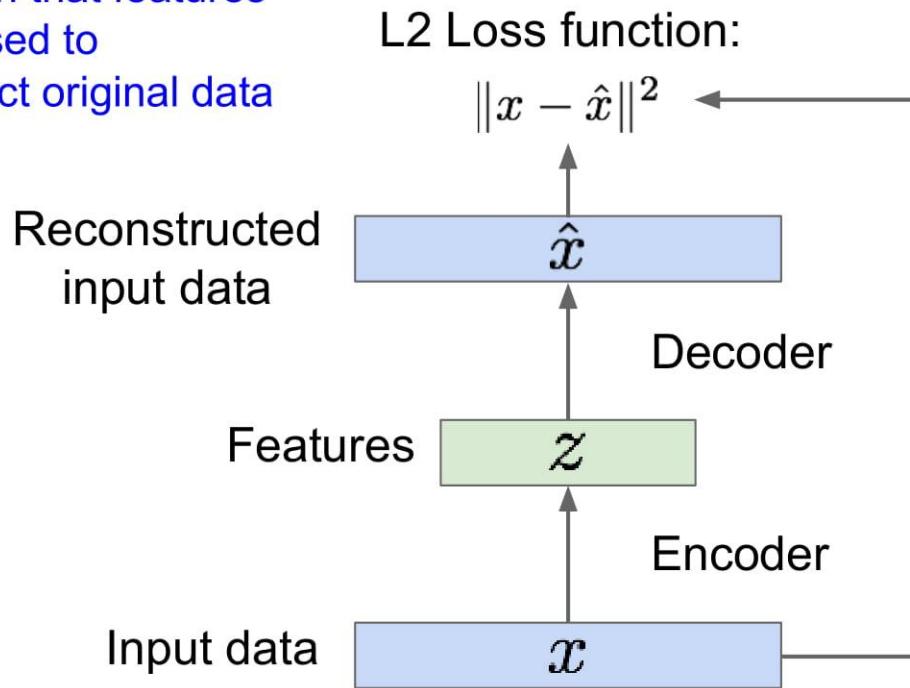
Train such that features can be used to reconstruct original data

“Autoencoding” - encoding itself



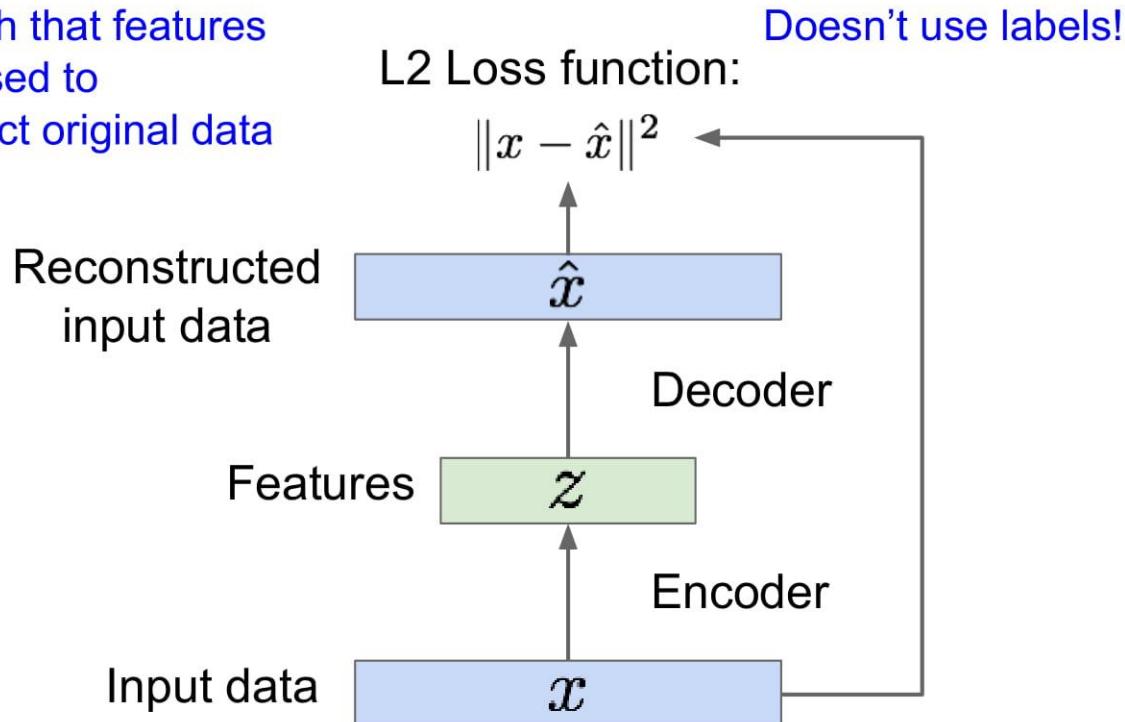
Some background first: Autoencoders

Train such that features
can be used to
reconstruct original data



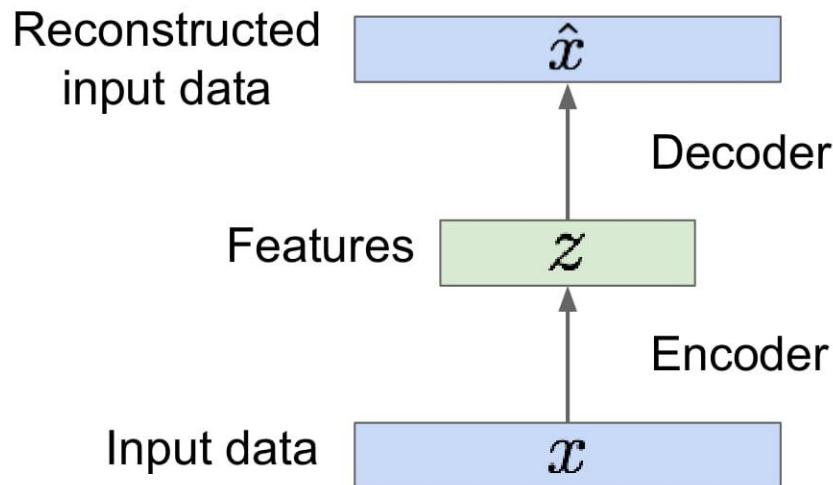
Some background first: Autoencoders

Train such that features
can be used to
reconstruct original data

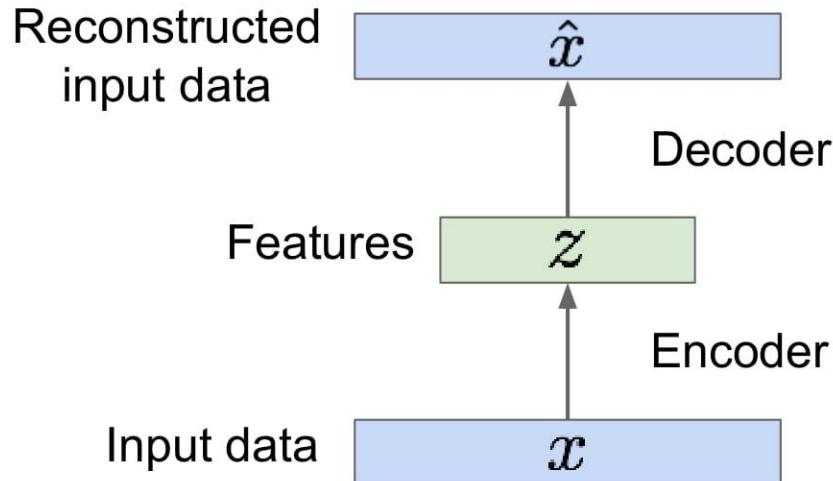


Some backgrounds first: Autoencoders

Autoencoders can reconstruct data, and can learn features 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. Can we generate new images from an autoencoder?

Variational Autoencoders

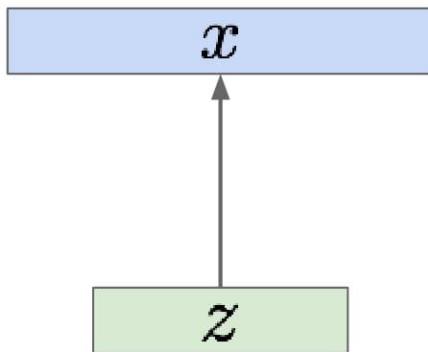
Probabilistic spin on autoencoders - will let us sample from the model to generate data!

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 underlying unobserved (latent) representation z

Sample from
true conditional
 $p_{\theta^*}(x \mid z^{(i)})$



Sample from
true prior
 $p_{\theta^*}(z)$

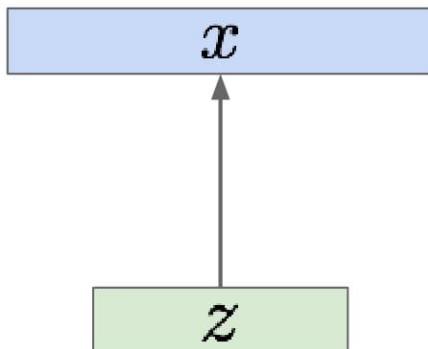
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

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 underlying unobserved (latent) representation z

Sample from
true conditional
 $p_{\theta^*}(x \mid z^{(i)})$



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

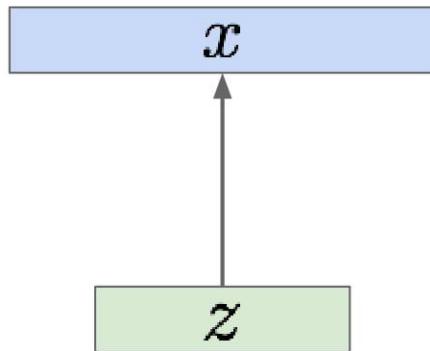
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders

We want to estimate the true parameters θ^* of this generative model.

Sample from
true conditional
 $p_{\theta^*}(x | z^{(i)})$

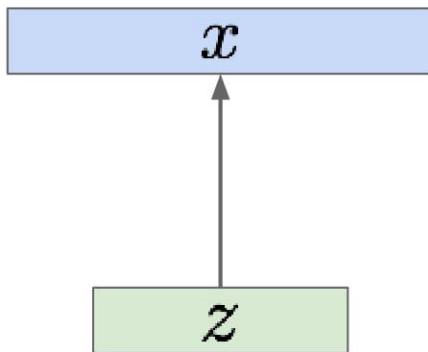
Sample from
true prior
 $p_{\theta^*}(z)$



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

Variational Autoencoders

Sample from
true conditional
 $p_{\theta^*}(x \mid z^{(i)})$



Sample from
true prior
 $p_{\theta^*}(z)$

We want to estimate the true parameters θ^* of this generative model.

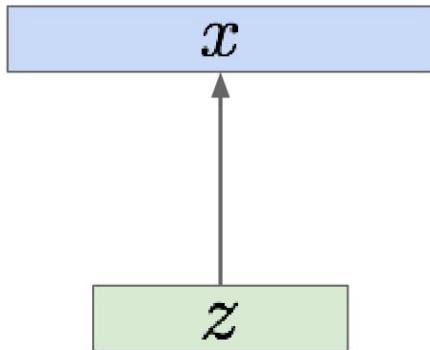
How should we represent this model?

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

Variational Autoencoders

Sample from
true conditional
 $p_{\theta^*}(x \mid z^{(i)})$

Sample from
true prior
 $p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model.

How should we represent this model?

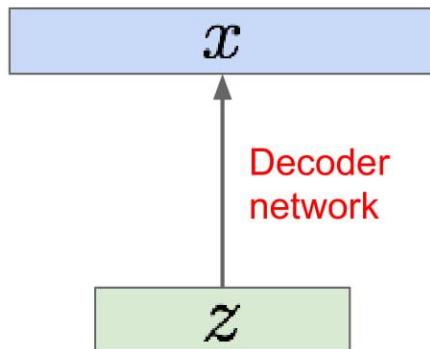
Choose prior $p(z)$ to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.

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

Variational Autoencoders

Sample from
true conditional
 $p_{\theta^*}(x | z^{(i)})$

Sample from
true prior
 $p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model.

How should we represent this model?

Choose prior $p(z)$ to be simple, e.g.
Gaussian.

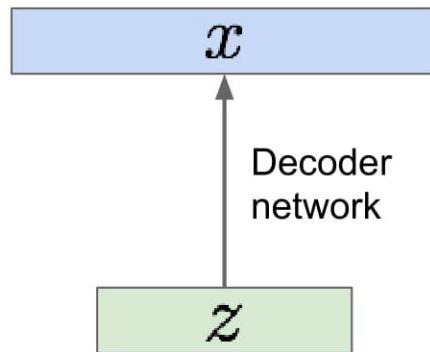
Conditional $p(x|z)$ is complex (generates
image) => represent with neural network

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

Variational Autoencoders

Sample from
true conditional
 $p_{\theta^*}(x | z^{(i)})$

Sample from
true prior
 $p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model.

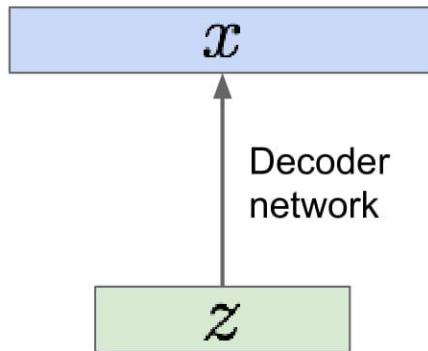
How to train the model?

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

Variational Autoencoders

Sample from
true conditional
 $p_{\theta^*}(x | z^{(i)})$

Sample from
true prior
 $p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model.

How to train the model?

Learn model parameters to maximize likelihood of training data

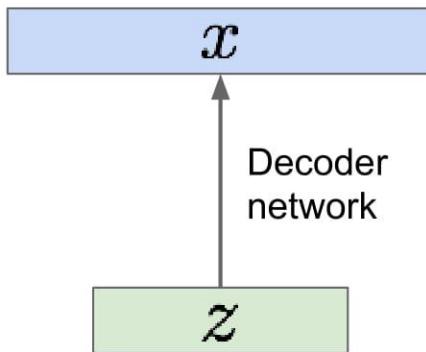
$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

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

Variational Autoencoders

Sample from
true conditional
 $p_{\theta^*}(x | z^{(i)})$

Sample from
true prior
 $p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model.

How to train the model?

Learn model parameters to maximize likelihood of training data

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

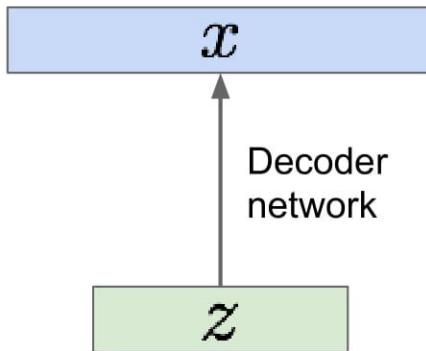
Now with latent z

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

Variational Autoencoders

Sample from
true conditional
 $p_{\theta^*}(x | z^{(i)})$

Sample from
true prior
 $p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model.

How to train the model?

Learn model parameters to maximize likelihood of training data

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

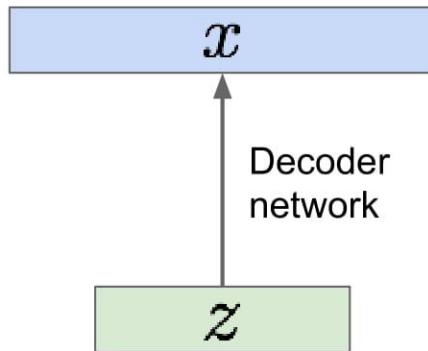
Q: What is the problem with this?

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

Variational Autoencoders

Sample from
true conditional
 $p_{\theta^*}(x | z^{(i)})$

Sample from
true prior
 $p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model.

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

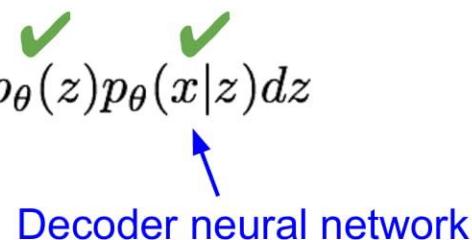


Simple Gaussian prior

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

Variational Autoencoders: Intractability

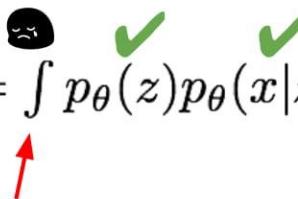
Data likelihood: $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$



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

Variational Autoencoders: Intractability

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



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

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

Variational Autoencoders: Intractability



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

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

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

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 decoder network modeling $p_\theta(x|z)$, define additional encoder network $q_\phi(z|x)$ that approximates $p_\theta(z|x)$

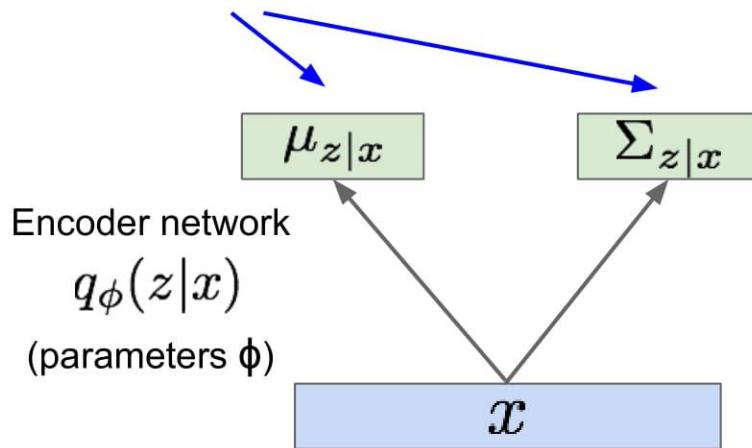
Will see that this allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize

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

Variational Autoencoders

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic

Mean and (diagonal) covariance of $z | x$

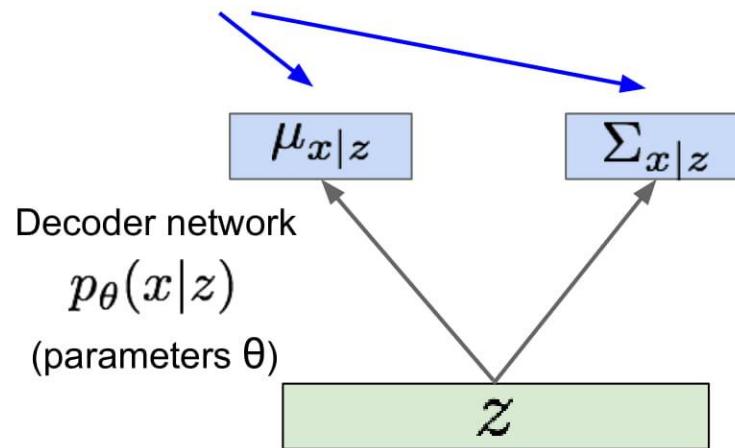


Encoder network

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

(parameters ϕ)

Mean and (diagonal) covariance of $x | z$



Decoder network

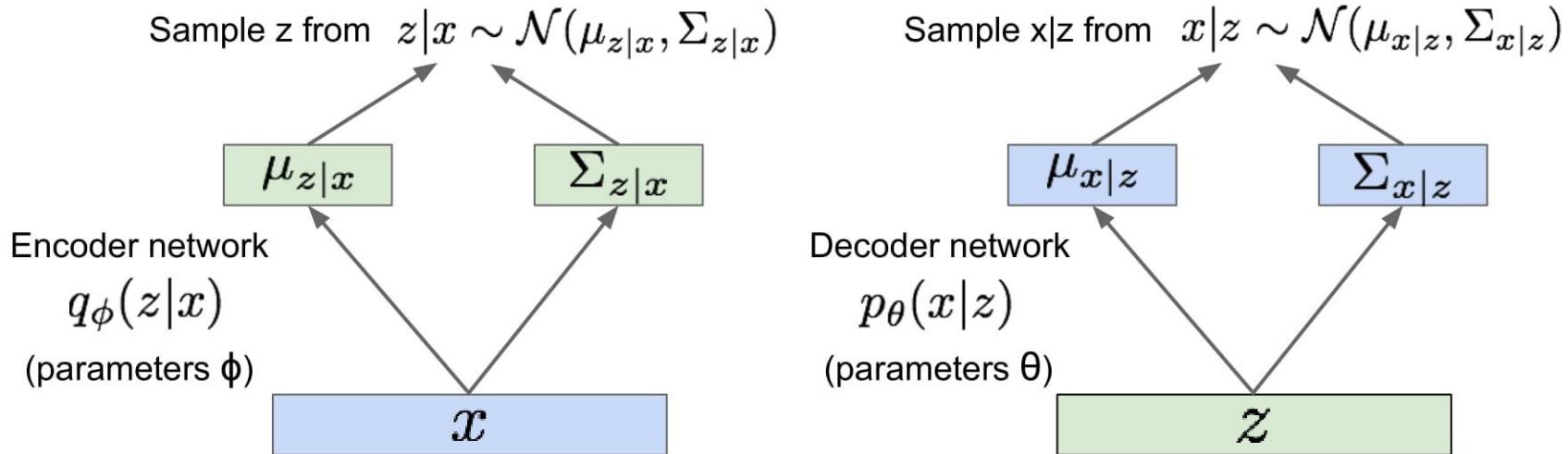
$$p_\theta(x|z)$$

(parameters θ)

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

Variational Autoencoders

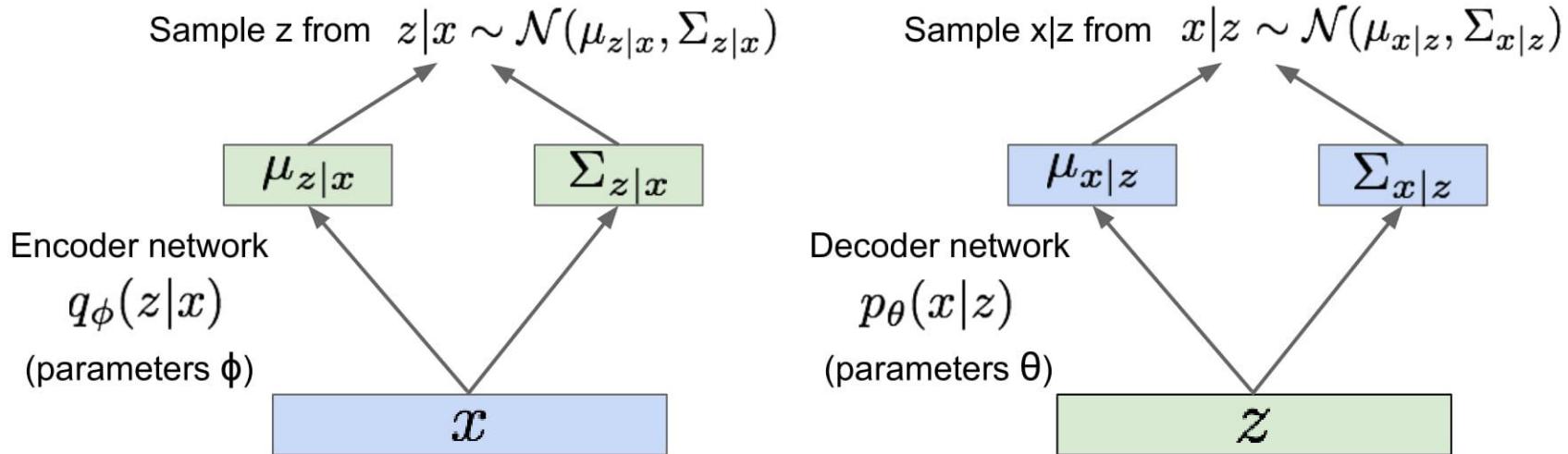
Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic



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

Variational Autoencoders

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic



Encoder and decoder networks also called
“recognition”/“inference” and “generation” networks

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

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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



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

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

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

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

↑
Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling. (Sampling differentiable through reparam. trick, see paper.)

↑
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 ≥ 0 .

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

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

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)$$

Variational lower bound ("ELBO")

$$\theta^*, \phi^* = \arg \max_{\theta, \phi} \sum_{i=1}^N \mathcal{L}(x^{(i)}, \theta, \phi)$$

Training: Maximize lower bound

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

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

Reconstruct
the input data

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

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

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

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)$$

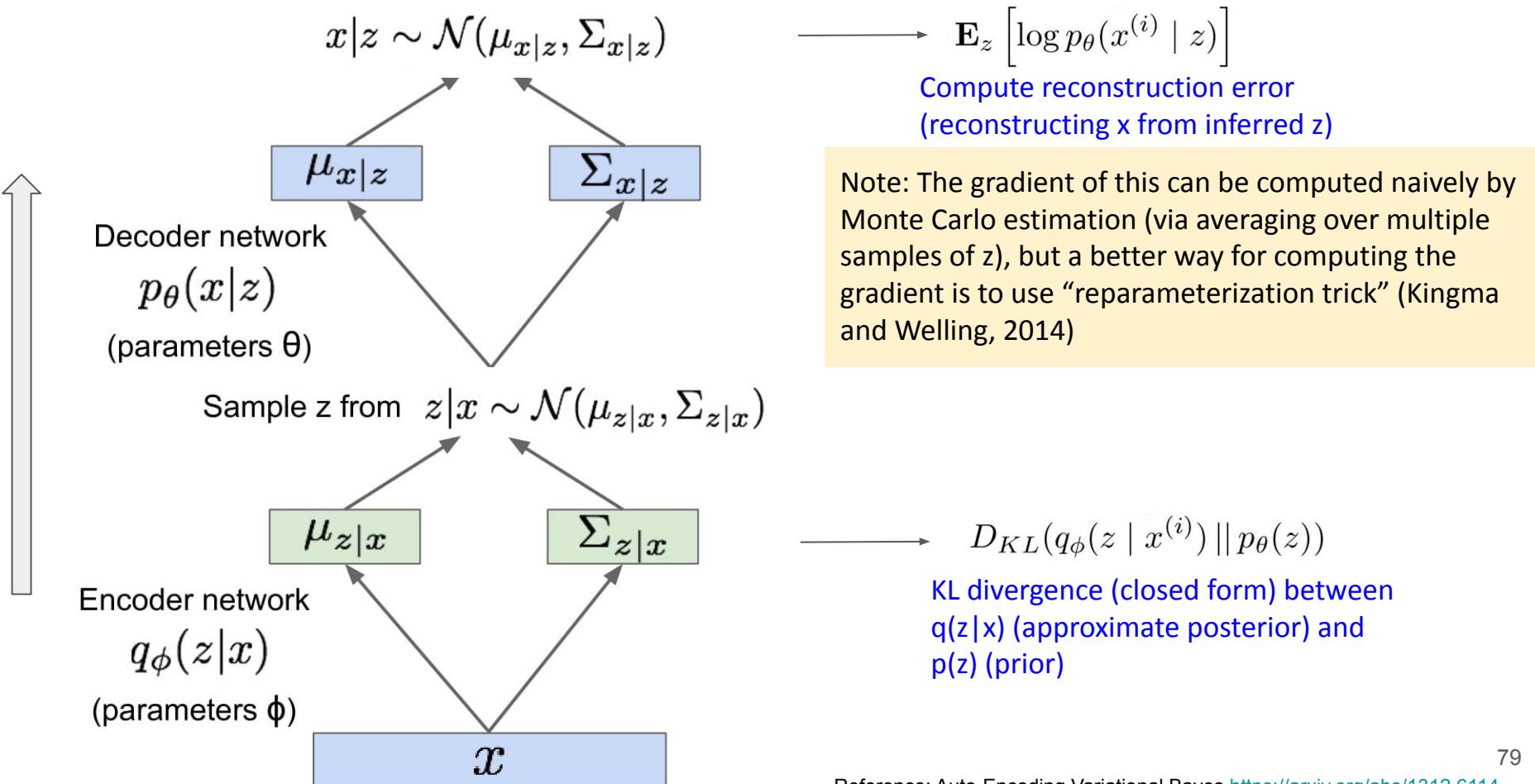
Variational lower bound ("ELBO")

Make approximate posterior distribution close to prior

$$\theta^*, \phi^* = \arg \max_{\theta, \phi} \sum_{i=1}^N \mathcal{L}(x^{(i)}, \theta, \phi)$$

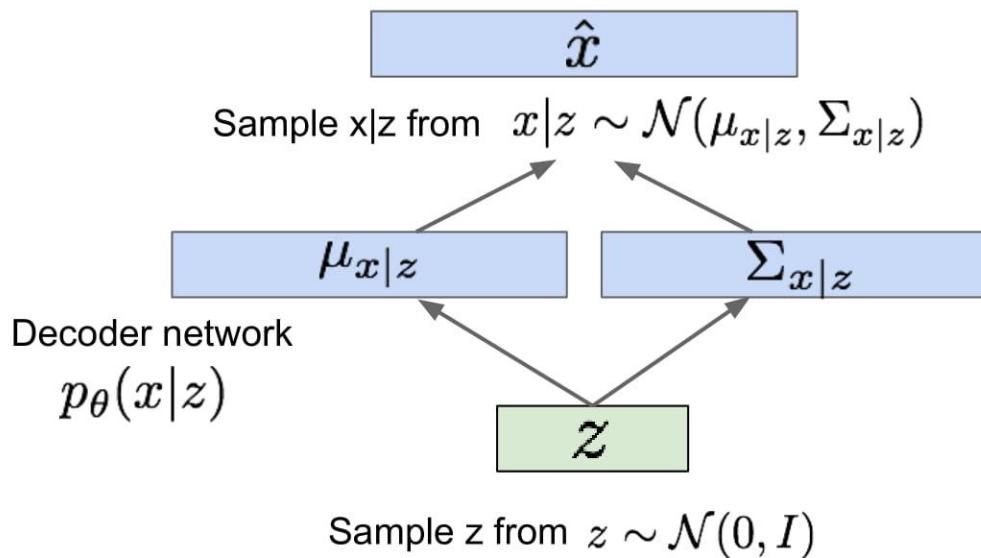
Training: Maximize lower bound

Training objective of VAE: ELBO

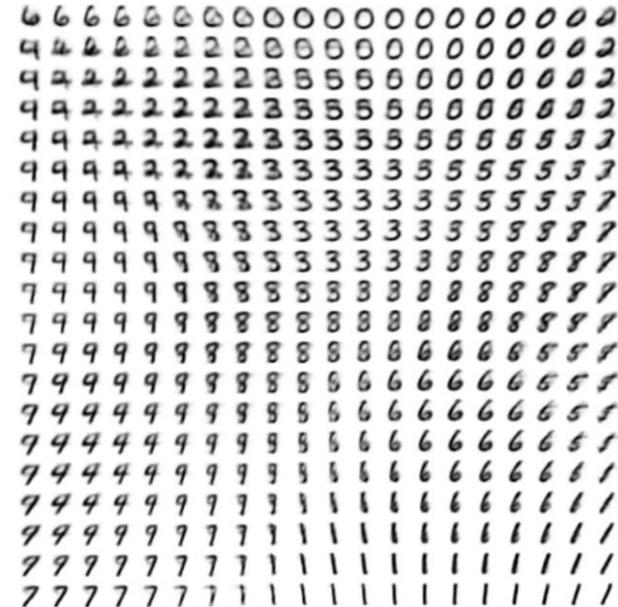


Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!

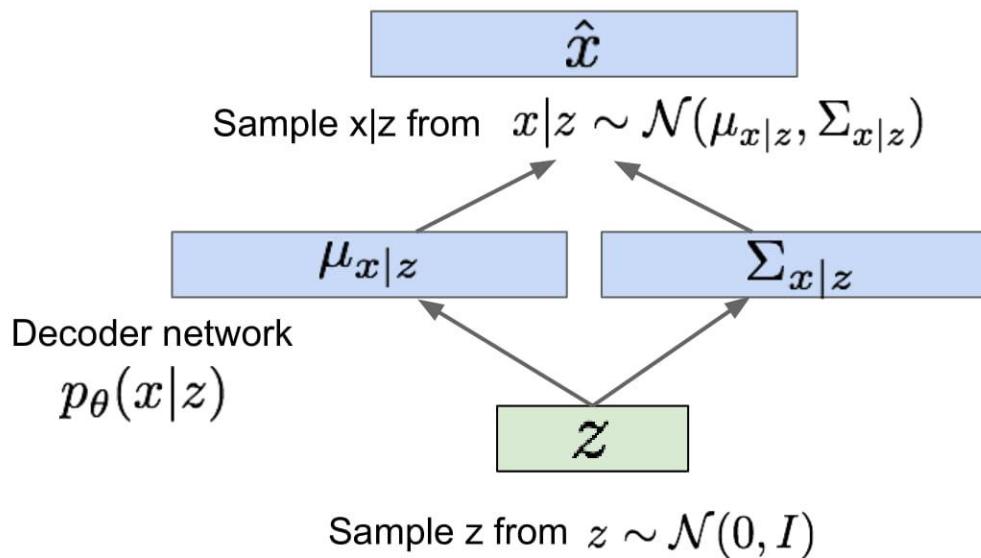


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

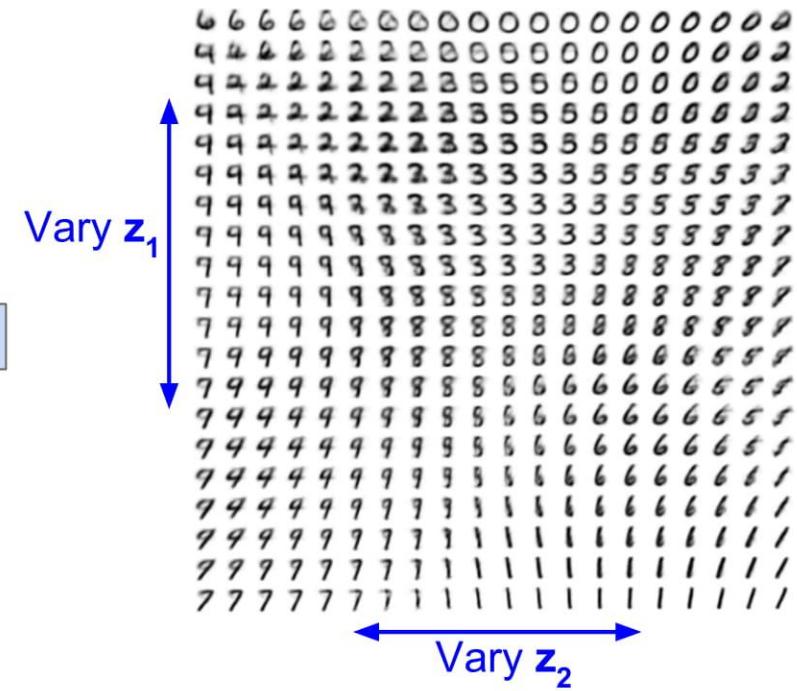


Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!



Data manifold for 2-d z

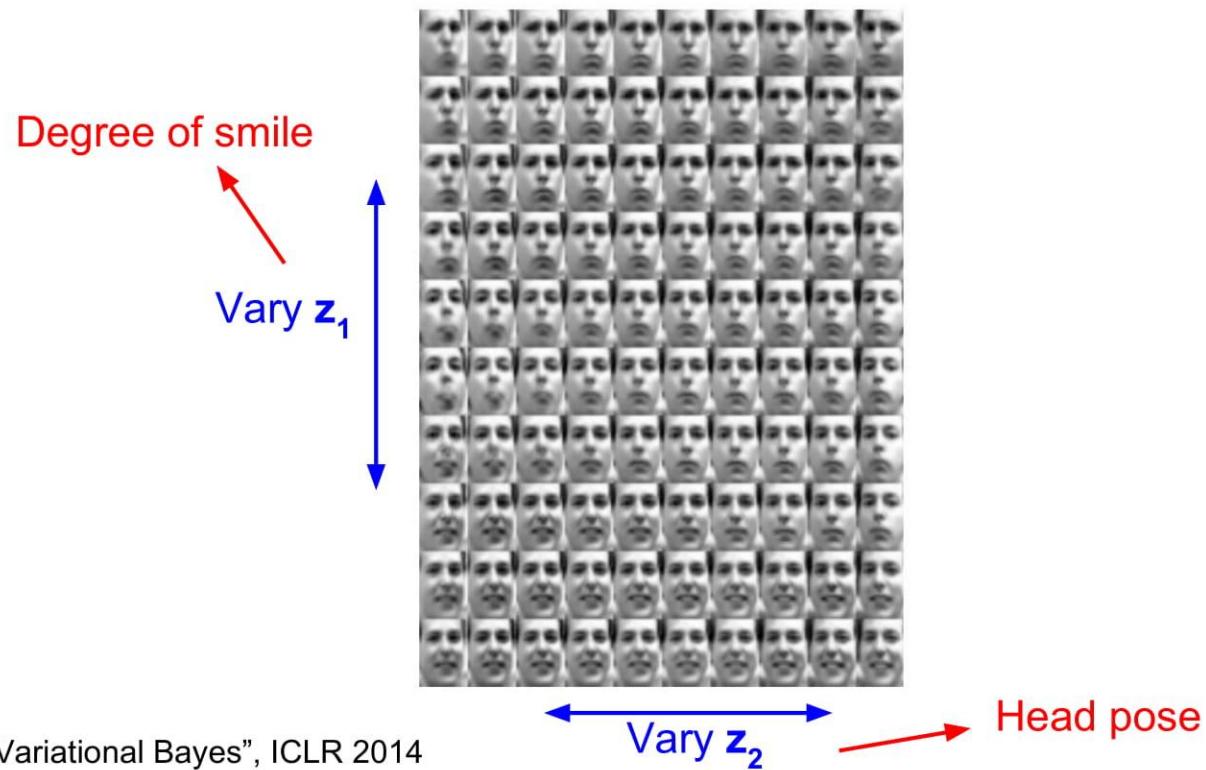


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders: Generating Data!

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

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



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 Autoencoder: Summary

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
- 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)

Further Extensions:

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, e.g., Gaussian Mixture Models (GMMs)
- Incorporating structure in latent variables, e.g., Categorical Distributions

Outline

- Generative Models Basics
- Autoregressive Models
- Autoencoder and Variational Autoencoder
- **Generative Adversarial Network**
- Diffusion Models

So far...

PixelCNNs 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...

PixelCNNs 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...

PixelCNNs 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: don't work with any explicit density function!

Instead, take game-theoretic approach: learn to generate from training distribution through 2-player game

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, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

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!

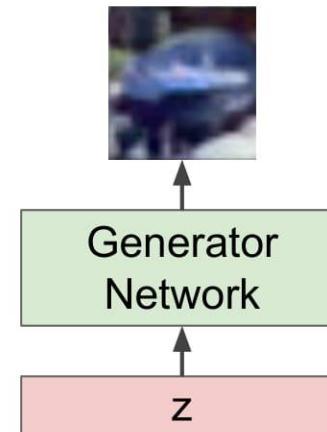
Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

A: A neural network!

Output: Sample from training distribution

Input: Random noise



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

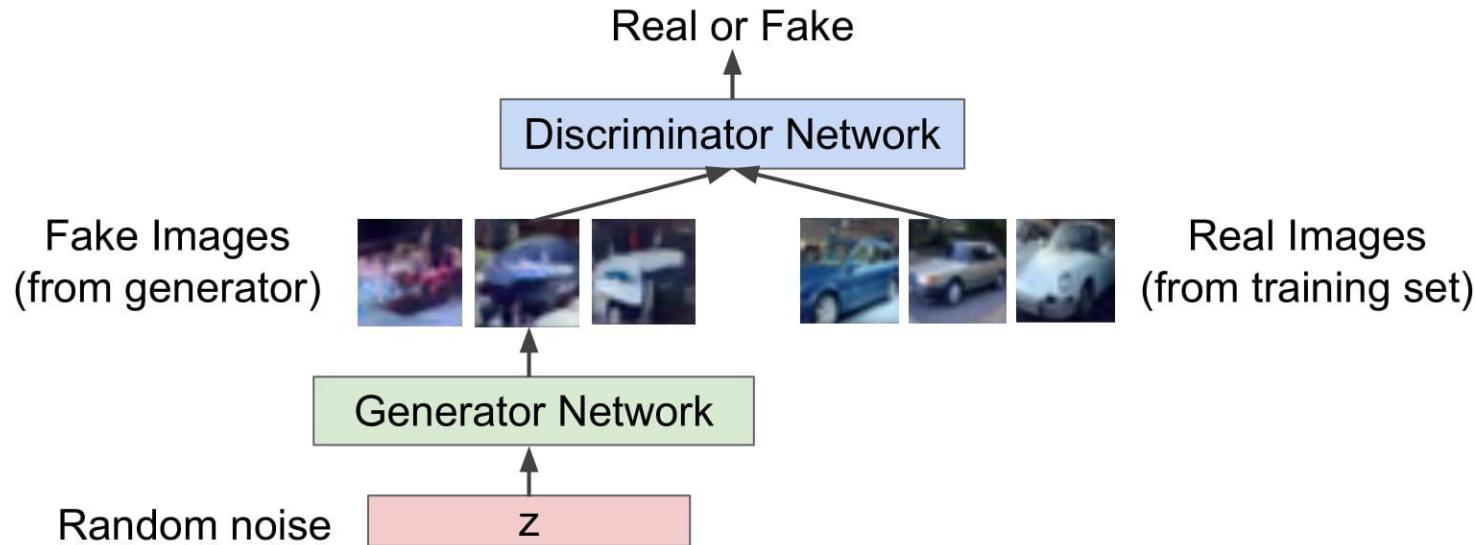
Discriminator network: try to distinguish between real and fake images

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.

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

Train jointly in **minimax game**

Minimax objective function:

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

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

Train jointly in **minimax game**

Minimax objective function:

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

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

Discriminator output for generated fake data $G(z)$

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

Train jointly in **minimax game**

Minimax objective function:

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

- Discriminator (θ_d) wants to **maximize objective** such that $D(x)$ is close to 1 (real) and $D(G(z))$ is close to 0 (fake)
- Generator (θ_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

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:

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

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):

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

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:

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

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

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

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):

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

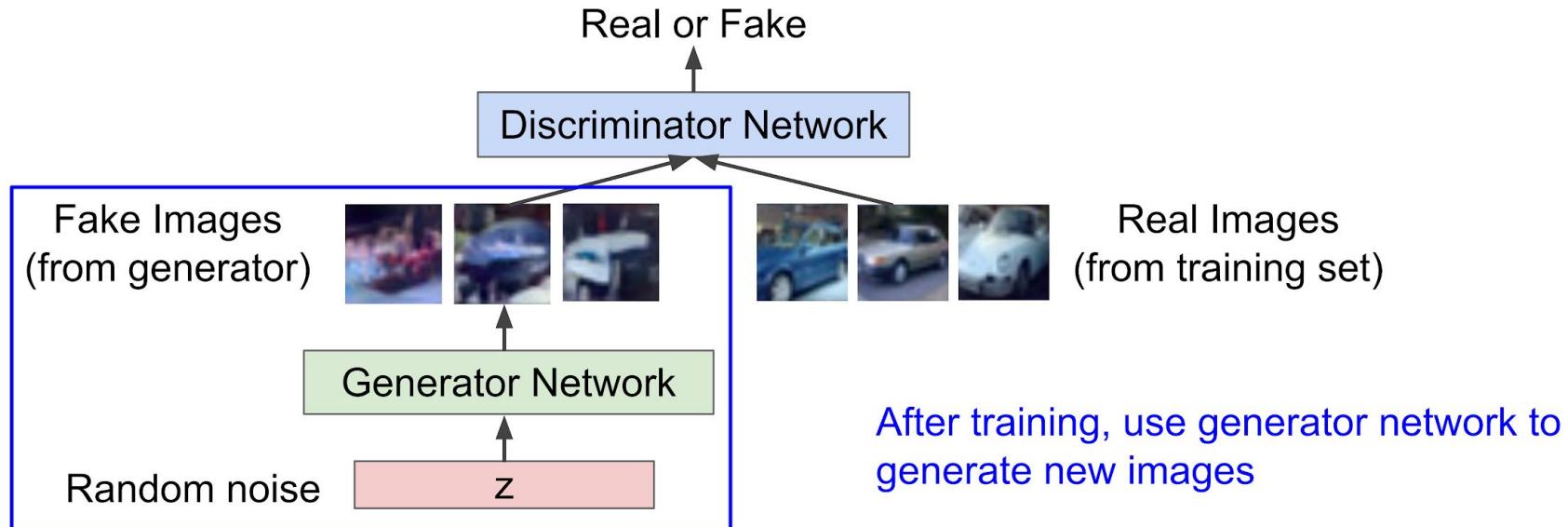
end for

Training GANs: Two-player game

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

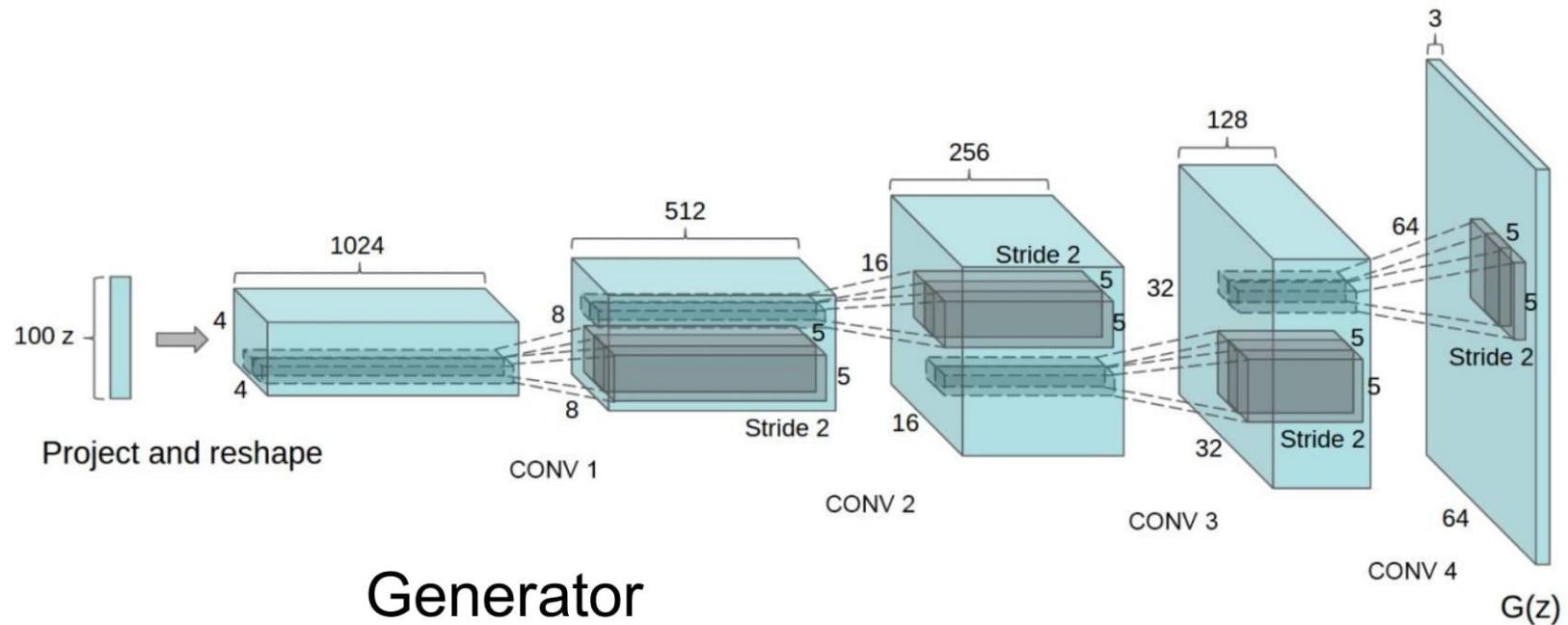
Generator network: try to fool the discriminator by generating real-looking images

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: Convolutional Architectures



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

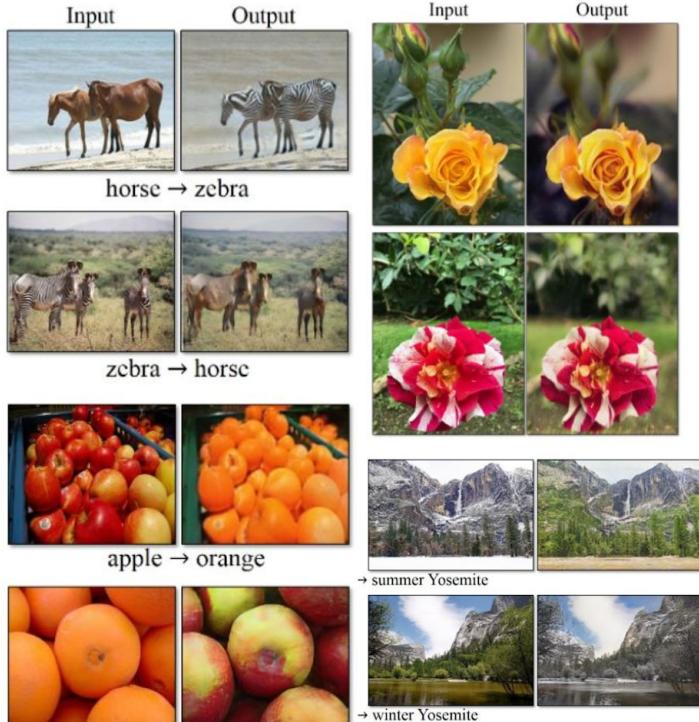
Interpolating
between
random
points in
latent space



Radford et al,
ICLR 2016

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.

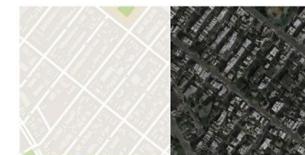
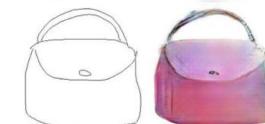


this magnificent fellow is almost all black with a red crest, and white cheek patch.



Reed et al. 2017.

Many GAN applications



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

GAN Improvements: Higher Resolution

256 x 256 bedrooms



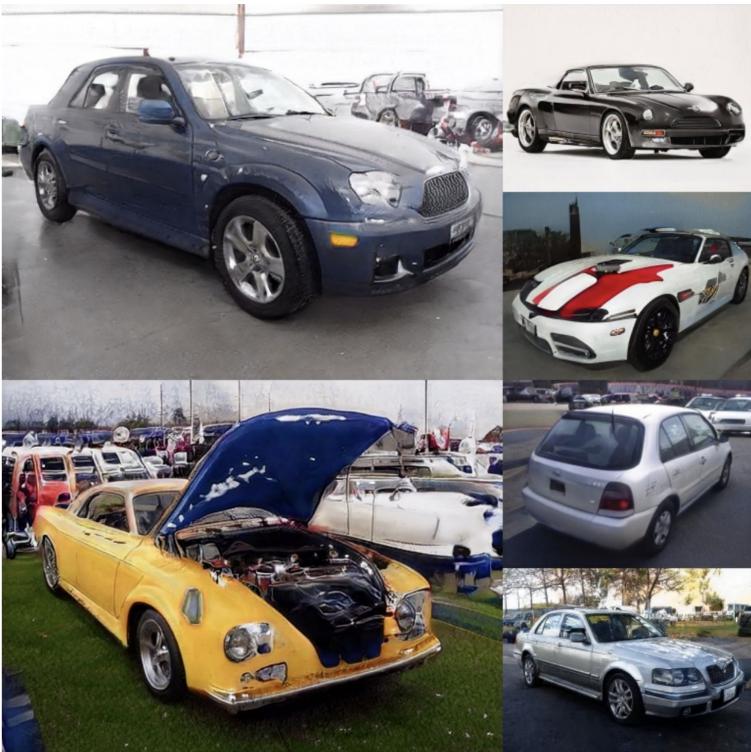
1024 x 1024 faces



Karras et al, "Progressive Growing of GANs for Improved Quality, Stability, and Variation", ICLR 2018

GAN Improvements: Higher Resolution

512 x 384 cars



1024 x 1024 faces



GANs: Summary

Don't work with an explicit density function

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

Pros:

- Beautiful samples!

Cons:

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

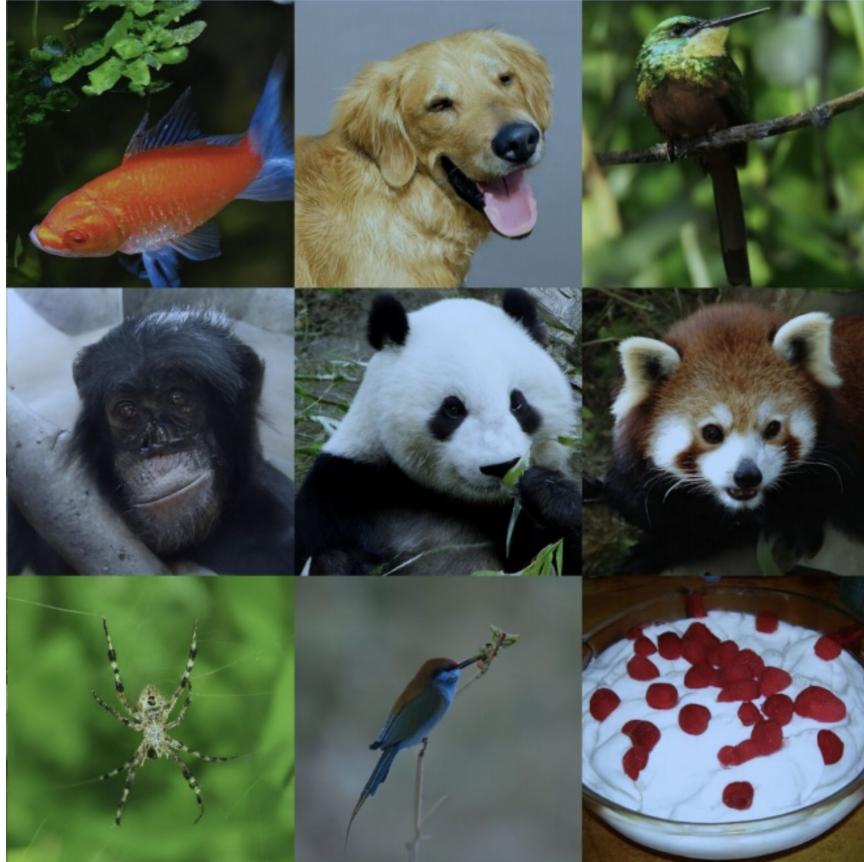
Further Extensions:

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

Outline

- Generative Models Basics
- Autoregressive Models
- Autoencoder and Variational Autoencoder
- Generative Adversarial Network
- **Diffusion Models**

Denoising Diffusion Models

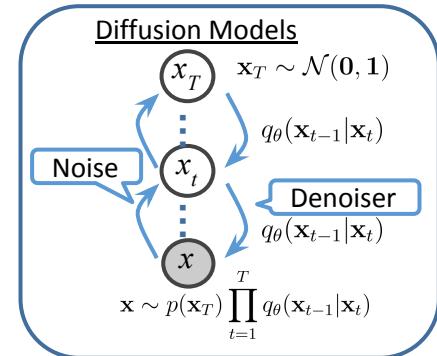


"Diffusion Models Beat GANs on Image Synthesis"
Dhariwal & Nichol, OpenAI, 2021



"Cascaded Diffusion Models for High Fidelity Image Generation" Ho et al., Google, 2021

Denoising Diffusion Models



Forward diffusion process (fixed)

Data



Noise

Reverse denoising process (generative)

Denoising Diffusion Models

Forward Diffusion Process

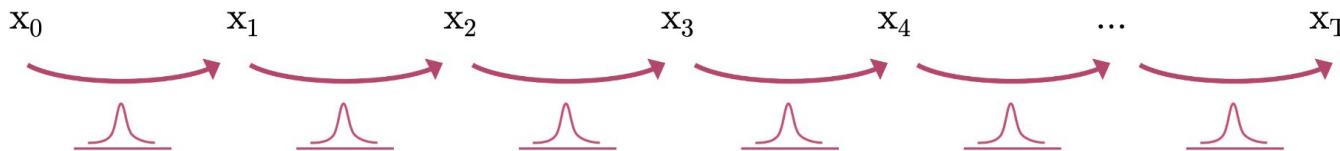
The formal definition of the forward process in T steps:

Forward diffusion process (fixed)

Data

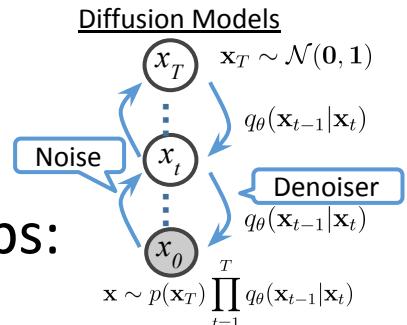


Noise

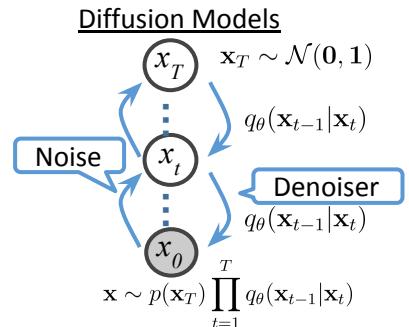


$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I}) \rightarrow q(\mathbf{x}_{1:T} | \mathbf{x}_0) = \prod_{t=1}^T q(\mathbf{x}_t | \mathbf{x}_{t-1}) \quad (\text{joint})$$

β_s controls the *noise schedule* such that \mathbf{x}_T is complete noise.



Diffusion Kernel



Forward diffusion process (fixed)

Data



Noise

$x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow \dots \rightarrow x_T$

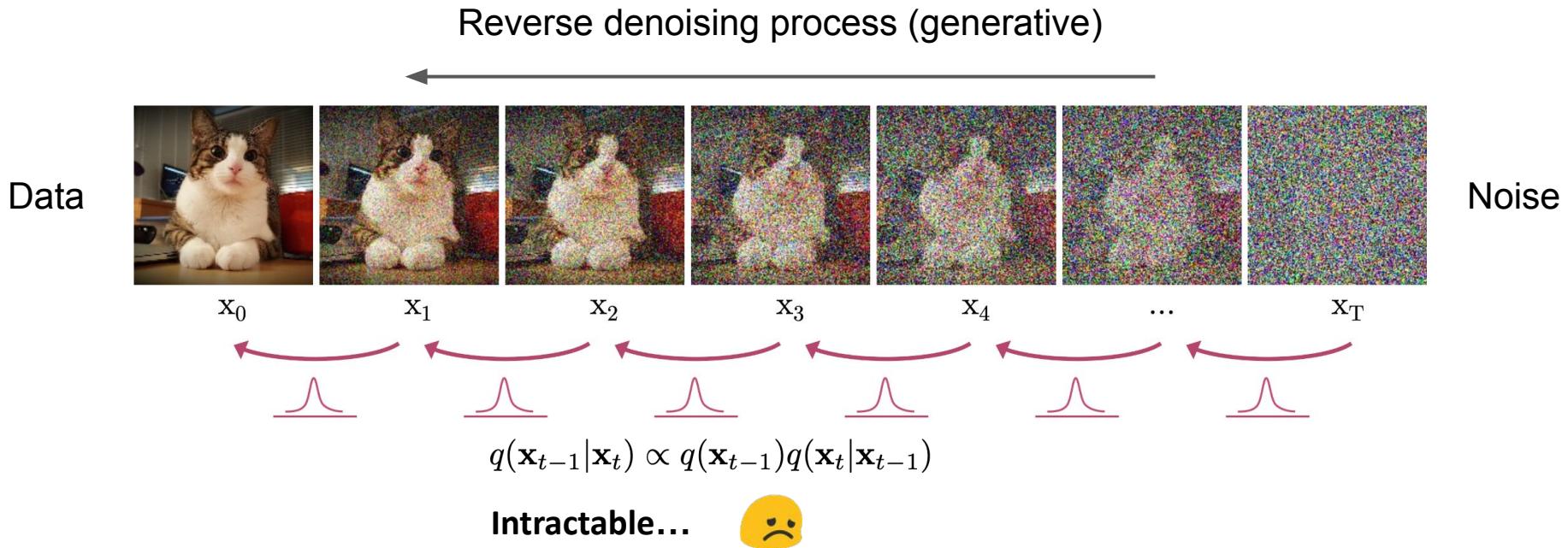


“Skip” t steps (**Diffusion Kernel**)

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I})$$

$$\text{Define } \bar{\alpha}_t = \prod_{s=1}^t (1 - \beta_s)$$

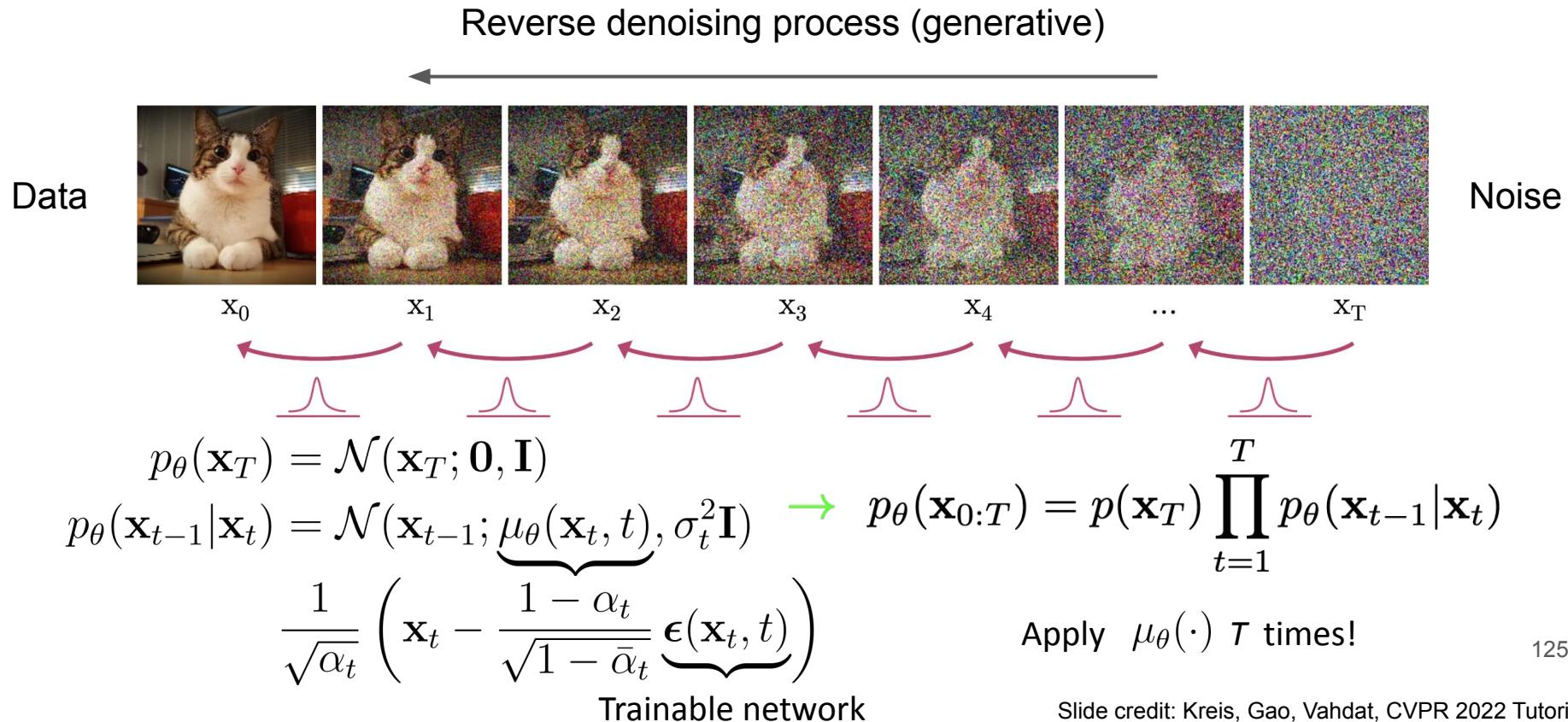
Reverse Denoising Process



Idea: Approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. Use a **Normal distribution** if β_t is small in each forward diffusion step.

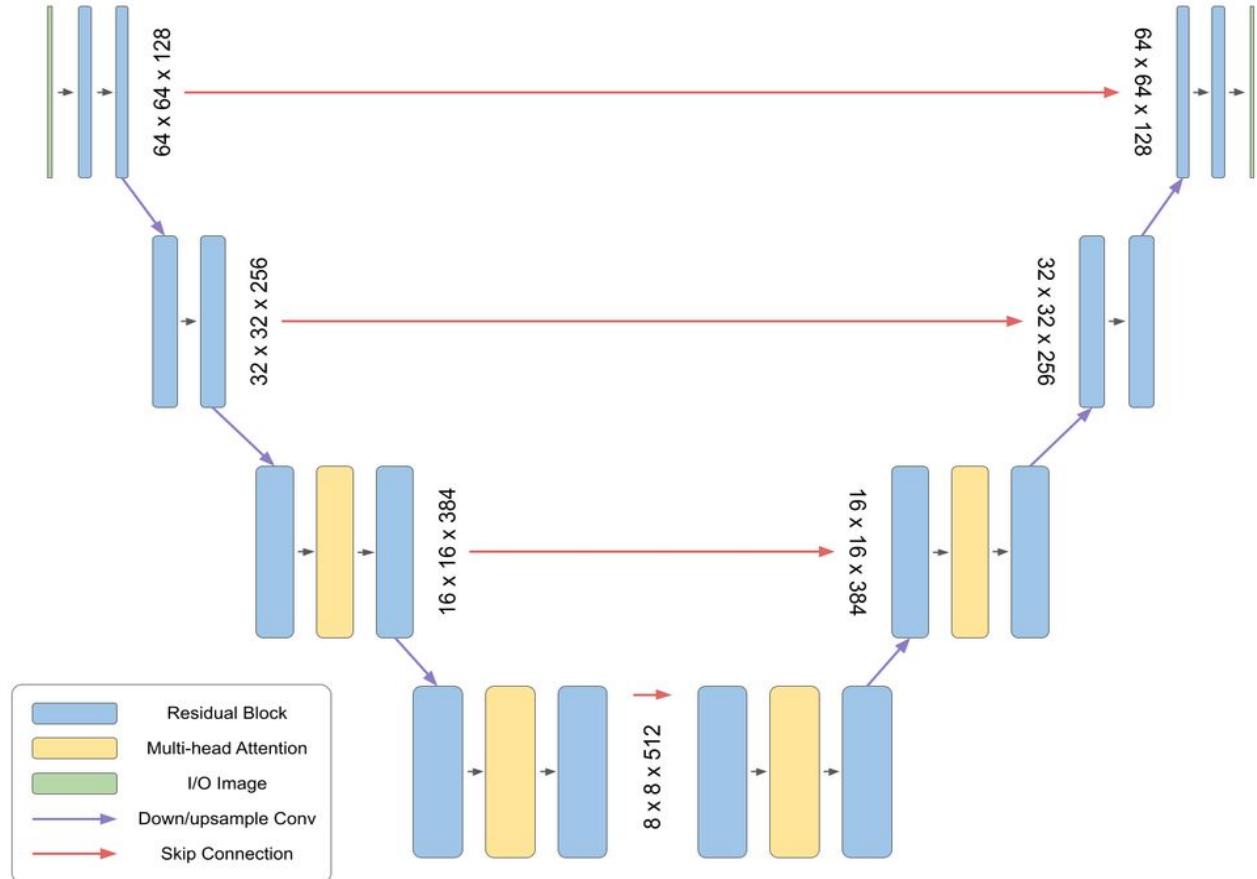
Reverse Denoising Process

Formal definition of forward and reverse processes in T steps:



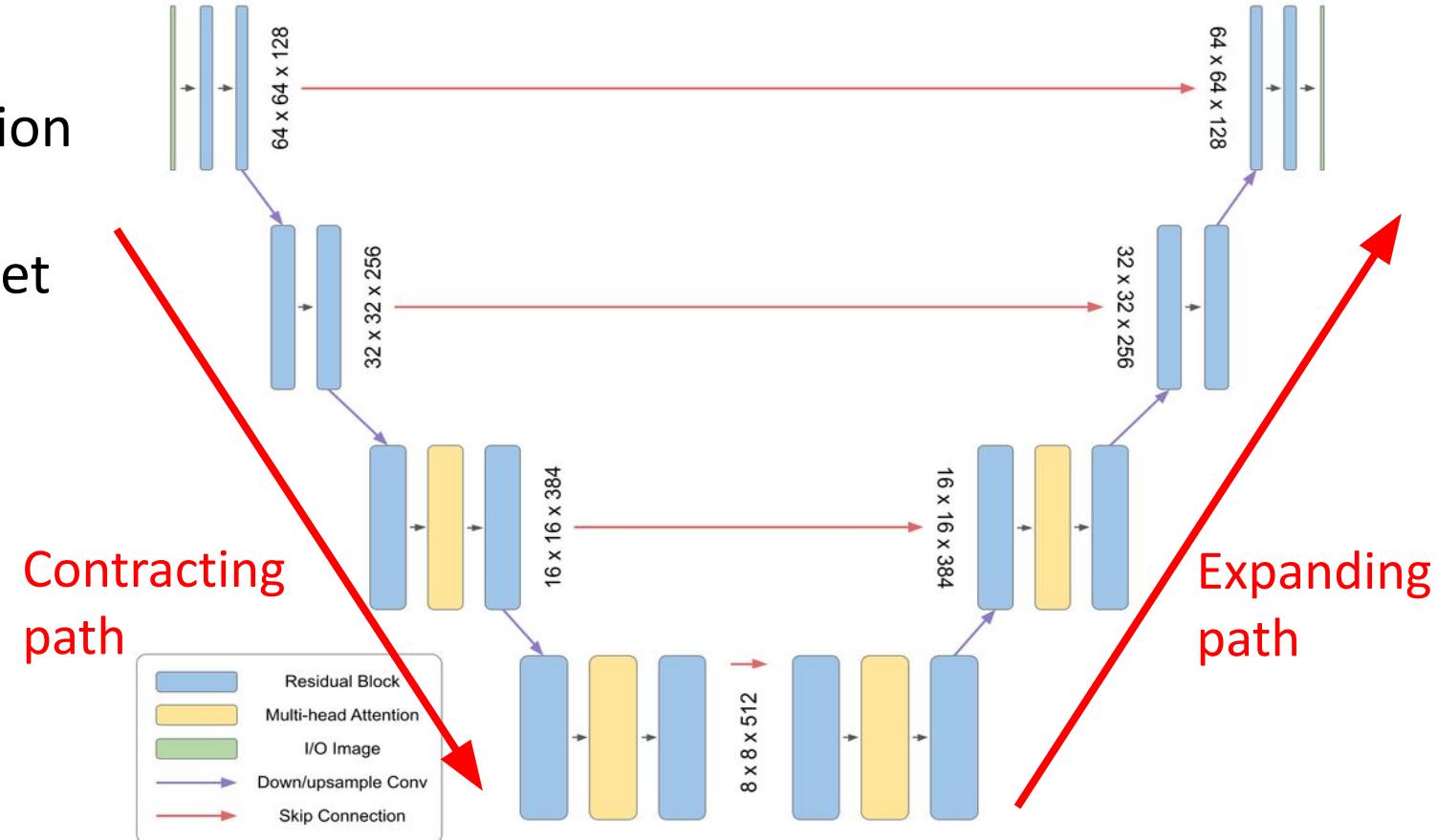
U-Net: Image-to-Image CNN network

The trainable
noise prediction
network is
usually a U-Net



U-Net: Image-to-Image CNN network

The trainable noise prediction network is usually a U-Net



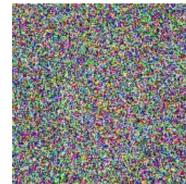
Training

Repeat:

Sample:



\mathbf{x}_0



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

$$t \sim \text{Uniform}(\{1, \dots, T\})$$

Diffusion kernel:



$$\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}$$

Add noise to clean image (with variance equivalent to t steps of noise)

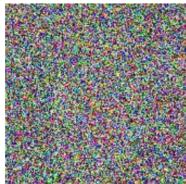
Gradient update:

$$\nabla_{\theta} \left\| \mathbf{x}_0 - \boldsymbol{\epsilon}_{\theta} \left(\begin{array}{c} \text{noisy image} \\ , t \end{array} \right) \right\|^2$$
$$\nabla_{\theta} \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\theta} \left(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, t \right) \right\|^2$$

Give model noisy image, ask it to predict the added noise. Train with L2 loss between predicted noise and original noise

Sampling

Sample:

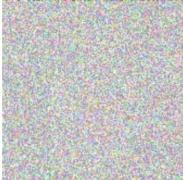


$$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$



for $t = 1 \dots T$ **do:**

Sample noise:



$$\mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \sigma_t \mathbf{I})$$

Predict next step:



$$\begin{aligned} &= \frac{1}{\sqrt{\alpha_t}} \left(\text{[Noisy Image]} - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_{\theta} \left(\text{[Noisy Image]}, t \right) \right) + \text{[Smoothed Image]} \\ x_{t-1} &= \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_{\theta}(\mathbf{x}_t, t) \right) + \mathbf{z}_t \end{aligned}$$

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:    $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
5:   Take gradient descent step on
      
$$\nabla_{\theta} \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\theta}(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, t)\|^2$$

6: until converged
```

Algorithm 2 Sampling

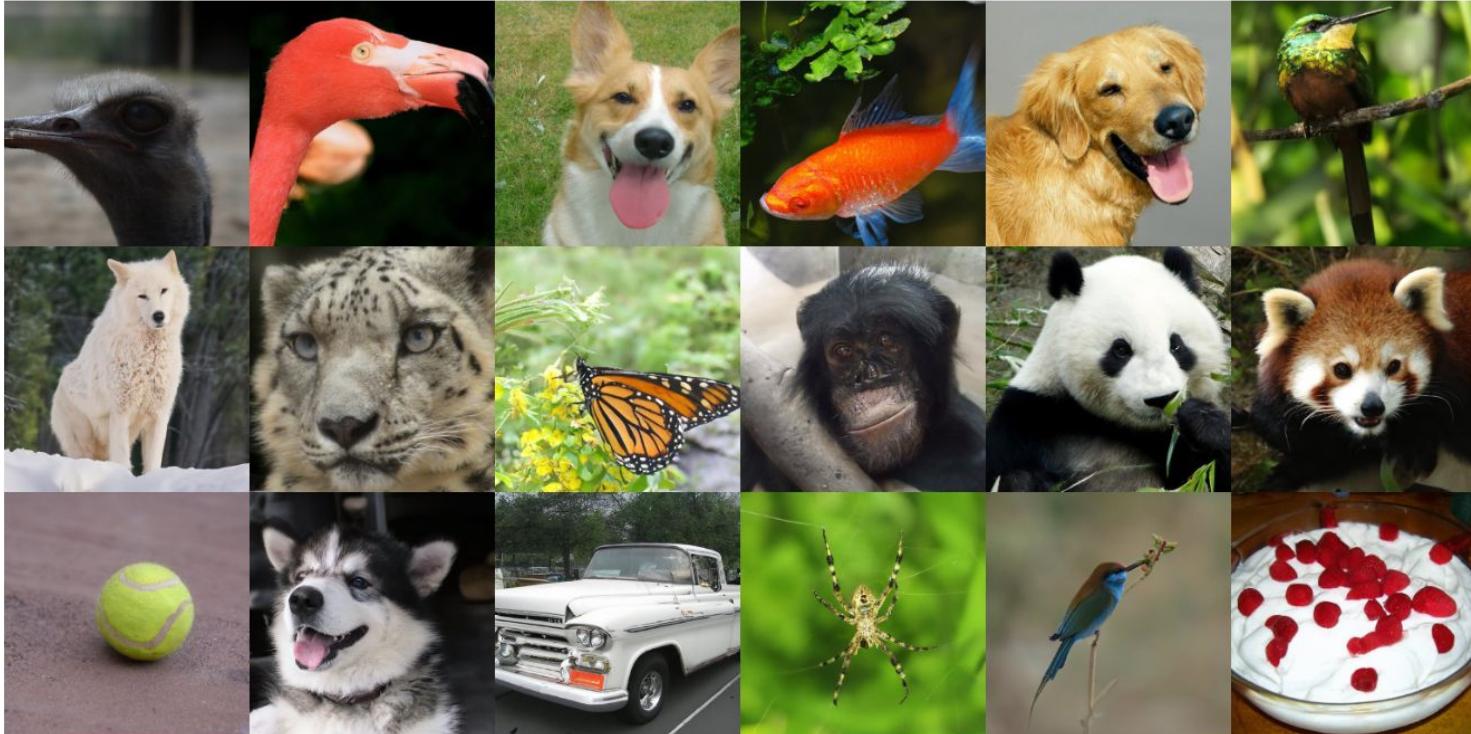
```
1:  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
2: for  $t = T, \dots, 1$  do
3:    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
4:   
$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_{\theta}(\mathbf{x}_t,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

5: end for
6: return  $\mathbf{x}_0$ 
```

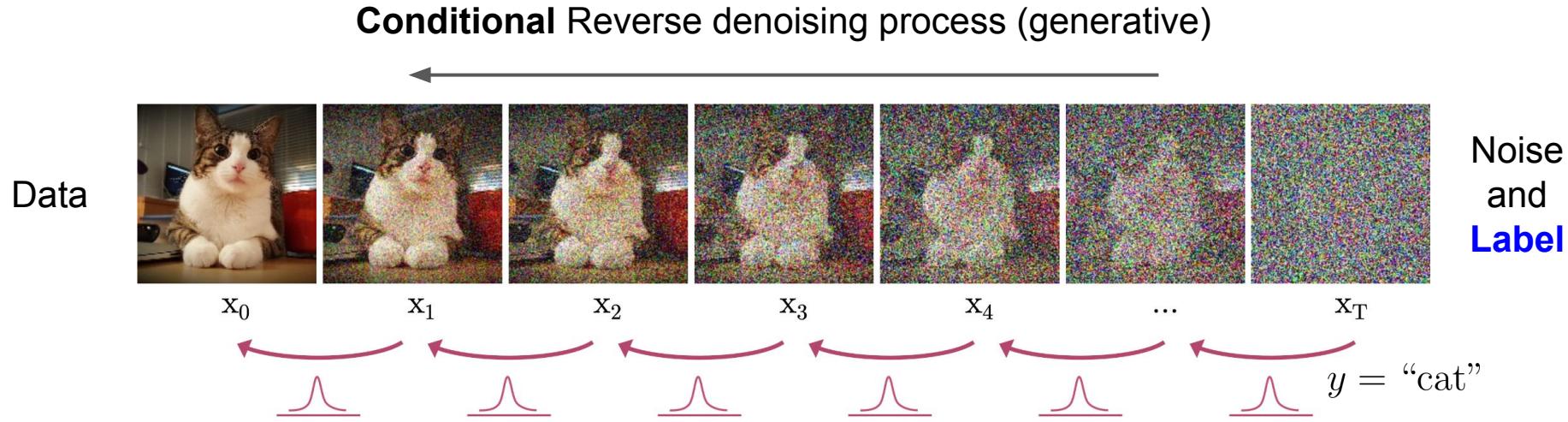
Improvements: Conditional Generation

Guided Diffusion

512 x 512 ImageNet Conditioned Samples



Improvements: Conditional Generation



We pass the condition ($y=\text{cat}$) to the noise-predicting model in addition to the noisy image and step number to “control” the generation (i.e. force the generated image to be a cat instead of a dog)

Text-to-image generation: train the diffusion model to condition on text!

Improvements: Latent Diffusion Models

(Combined with text-image models such as [CLIP](#))

Stable Diffusion

Text-to-Image Synthesis on LAION. 1.45B Model.

'A street sign that reads
"Latent Diffusion"'

'A zombie in the
style of Picasso'

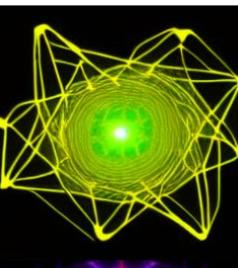
'An image of an animal
half mouse half octopus'

'An illustration of a slightly
conscious neural network'

'A painting of a
squirrel eating a burger'

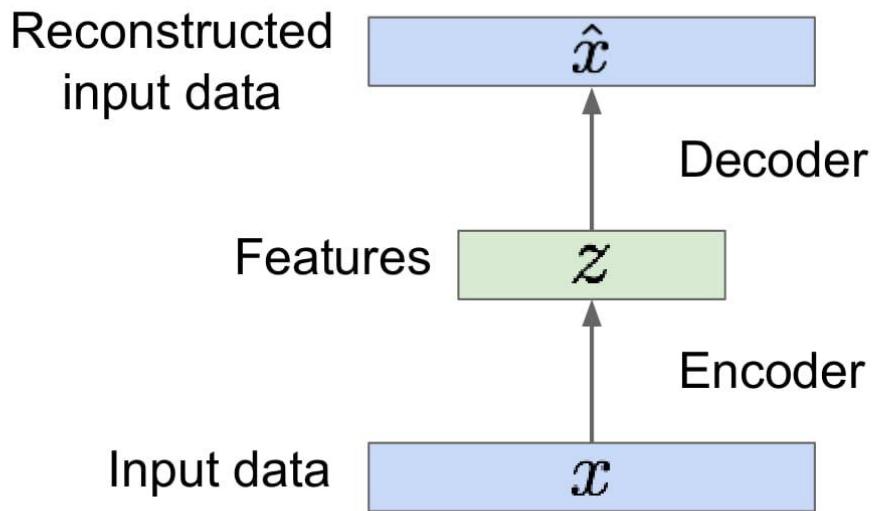
'A watercolor painting of a
chair that looks like an octopus'

'A shirt with the inscription:
"I love generative models!"'



Improvements: Latent Diffusion Models

Use an **autoencoder** to compress the data!

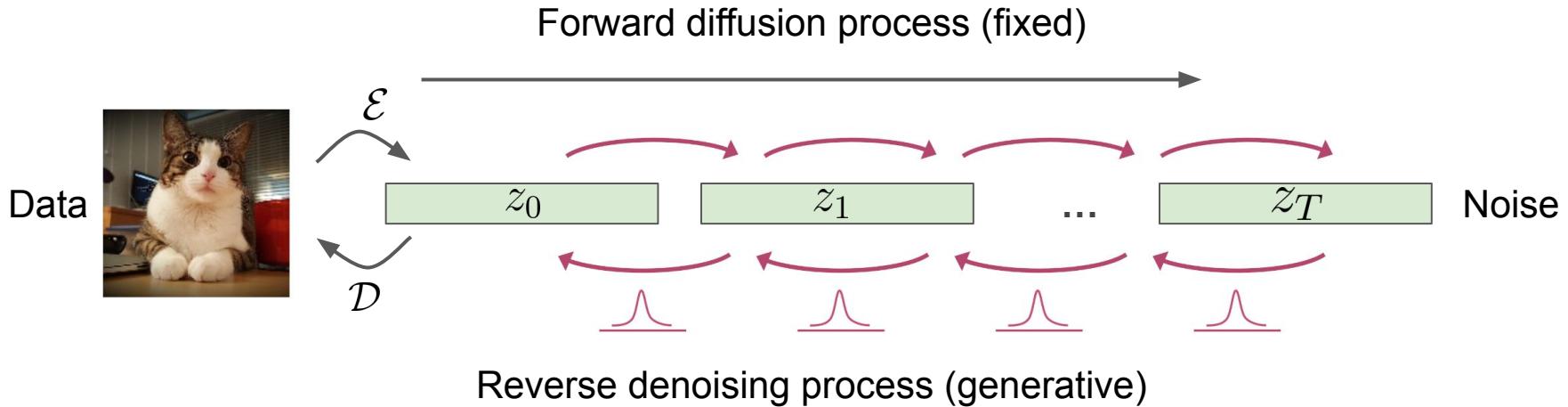
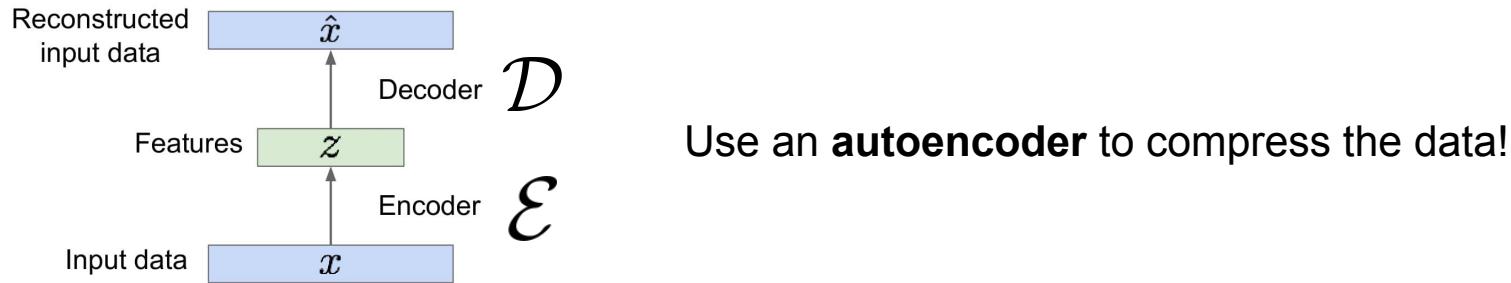


Original Image: 256x256x3

Latent Representation: 32x32x4

Reconstructed Image: 256x256x3

Improvements: Latent Diffusion Models



Most popular text-to-image generation models nowadays are Latent Diffusion Models

For example,

- OpenAI DALLE-2 and DALLE-3, and likely all images generated from text within ChatGPT
- Midjourney
- Imagen (by Google)

Diffusion models can also be used for text-to-video generation.

- OpenAI Sora is a latent diffusion model with transformer architecture

Recap

| | Pros | Cons |
|--|--|--|
| Autoregressive (PixelRNN, PixelCNN) | Explicit density model Optimizes exact likelihood | Inefficient sequential generation, sample quality not the best |

Recap

| | | Pros | Cons |
|--|--|---|--|
| Autoregressive (PixelRNN, PixelCNN) | Explicit density model | Optimizes exact likelihood | Inefficient sequential generation, sample quality not the best |
| Variational Autoencoders (VAE) | Optimize variational lower bound on likelihood | Useful latent representation, inference queries | Sample quality not the best |

Recap

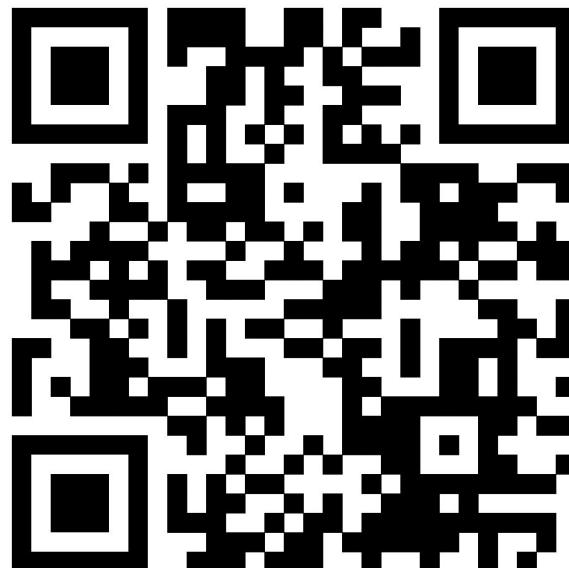
| | | Pros | Cons |
|--|--|---|--|
| Autoregressive (PixelRNN, PixelCNN) | Explicit density model | Optimizes exact likelihood | Inefficient sequential generation, sample quality not the best |
| Variational Autoencoders (VAE) | Optimize variational lower bound on likelihood | Useful latent representation, inference queries | Sample quality not the best |
| Generative Adversarial Networks (GANs) | Game-theoretic approach | Great samples | Tricky and unstable to train, no inference queries |

Recap

| | | Pros | Cons |
|--|--|---|--|
| Autoregressive (PixelRNN, PixelCNN) | Explicit density model | Optimizes exact likelihood | Inefficient sequential generation, sample quality not the best |
| Variational Autoencoders (VAE) | Optimize variational lower bound on likelihood | Useful latent representation, inference queries | Sample quality not the best |
| Generative Adversarial Networks (GANs) | Game-theoretic approach | Great samples | Tricky and unstable to train, no inference queries |
| Diffusion Models | Optimize variational lower bound for denoising | Best samples! Easy to train | Inefficient sequential generation |

Any feedback (about lecture, slide, homework, project, etc.)?

(via anonymous google form: <https://forms.gle/fpYmiBtG9Me5qbP37>)



Change Log of lecture slides:

<https://docs.google.com/document/d/e/2PACX-1vSSIHjklypK7rKFSR1-5GYXyBCEW8UPtpSfCR9AR6M1l7K9ZQEmxfFwaWaW7kLDxusthsF8WICyZJ-/pub>

Mathematical details for diffusion models

Learning Denoising Model: Variational upper bound

(Training) Variational upper bound 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

As $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ 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:

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

With this parameterization

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

Training Objective Weighting

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

The time dependent λ_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:

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

For more advanced weighting, see “Choi et al., Perception Prioritized Training of Diffusion Models, CVPR 2022.”