

EECS 498/598: Deep Learning

Lecture 8. Generative Models

Honglak Lee

03/01/2019



Outline

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

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

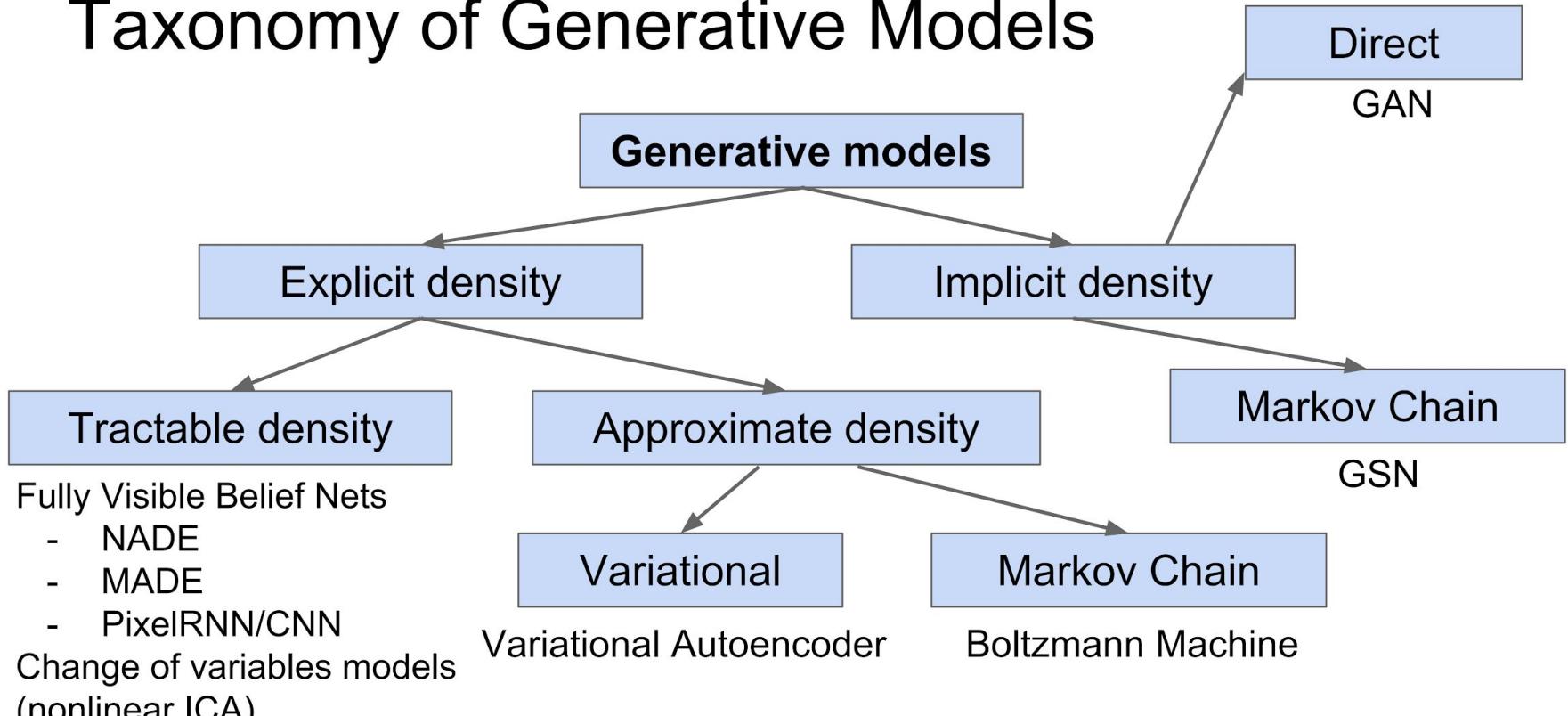


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

Today: discuss 3 most popular types of generative models today

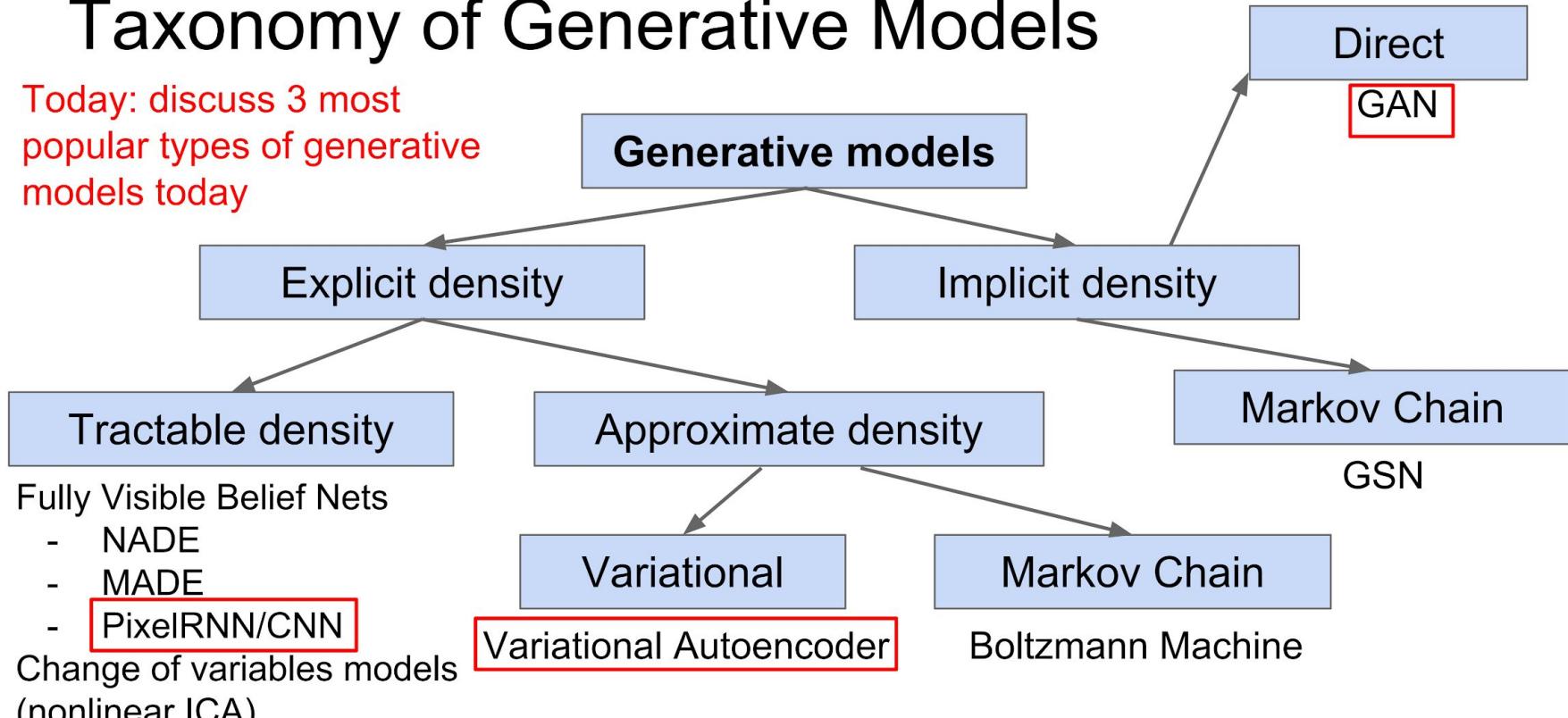
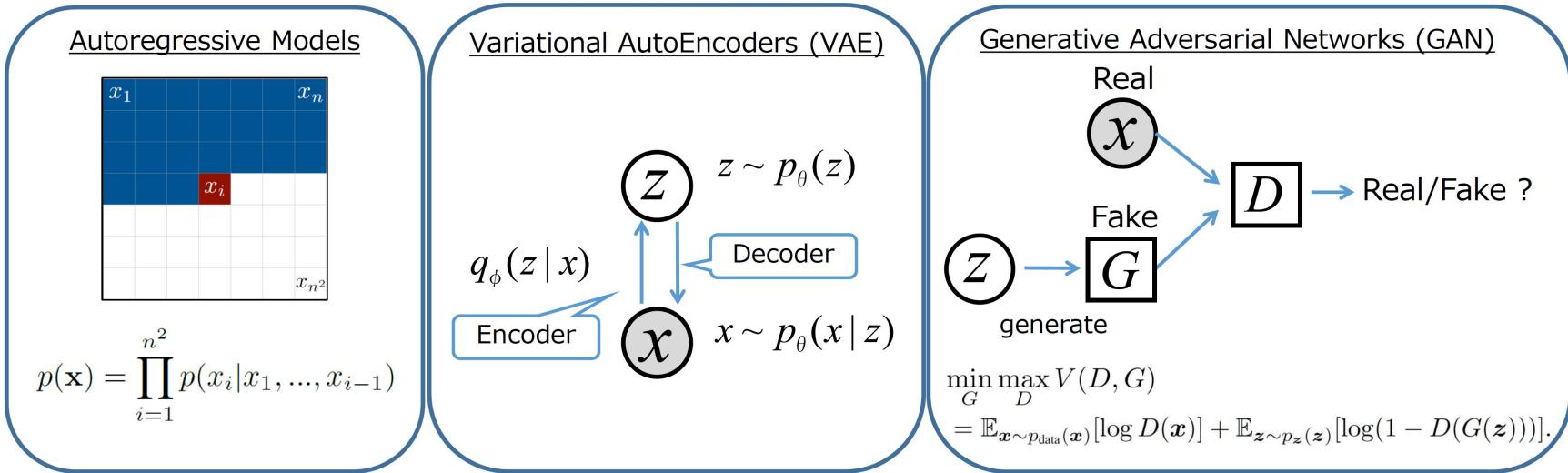


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Image Generation Models

Three image generation approaches are dominating this field:



	Autoregressive Models	VAE	GAN
Pros	Very 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
Cons	Relatively inefficient during sampling	Generated samples tend to be blurry	Difficult to optimize due to unstable training dynamics

Outline

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

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

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”

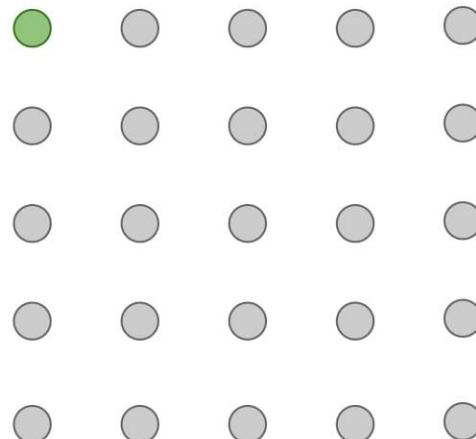
Then maximize likelihood of training data

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

PixelRNN

Generate image pixels starting from corner

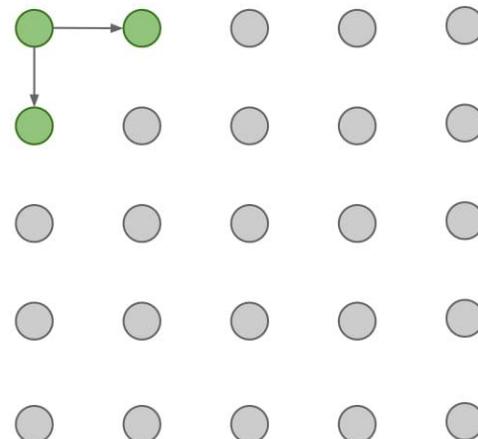
Dependency on previous pixels modeled
using an RNN (LSTM)



PixelRNN

Generate image pixels starting from corner

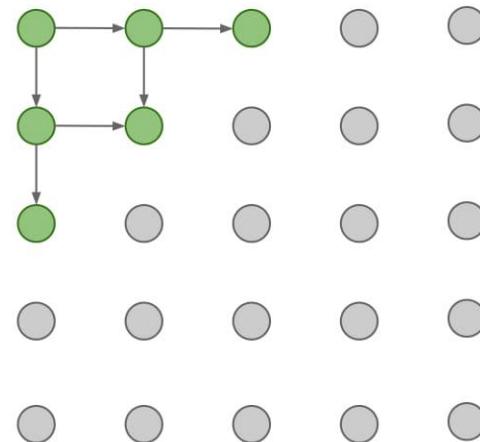
Dependency on previous pixels modeled
using an RNN (LSTM)



PixelRNN

Generate image pixels starting from corner

Dependency on previous pixels modeled
using an RNN (LSTM)

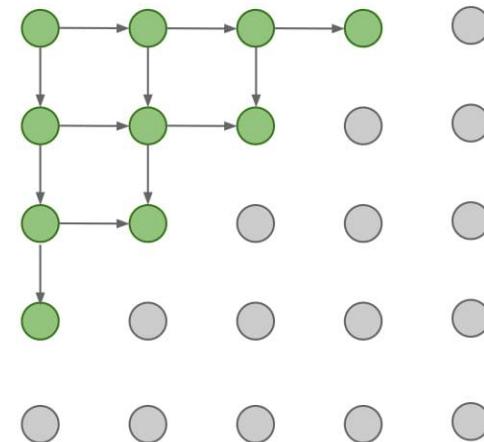


PixelRNN

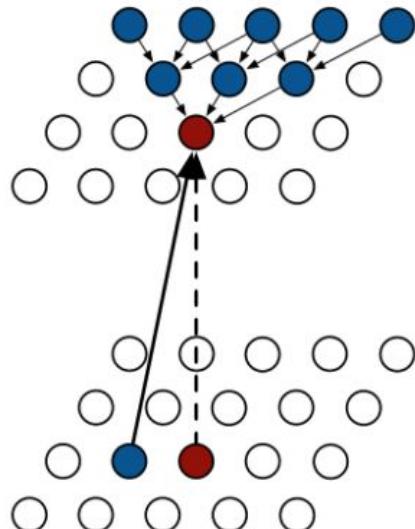
Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

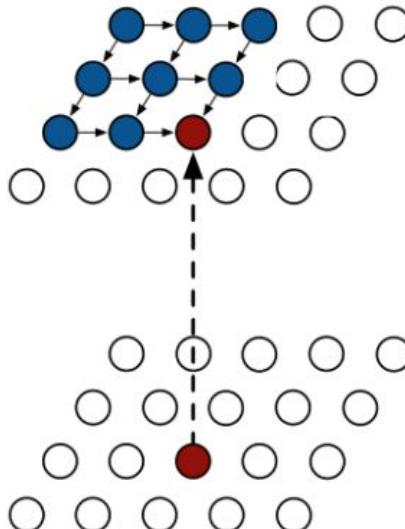
Drawback: sequential generation is slow!



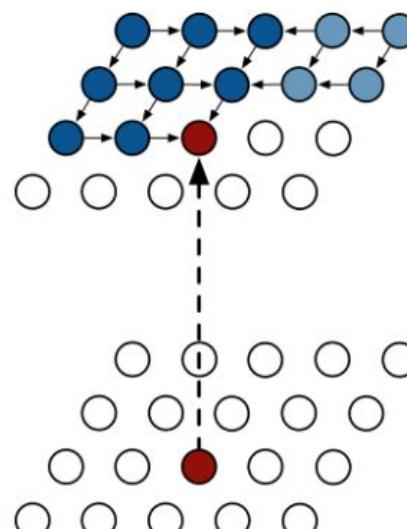
Variants of Pixel RNN (LSTM)



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

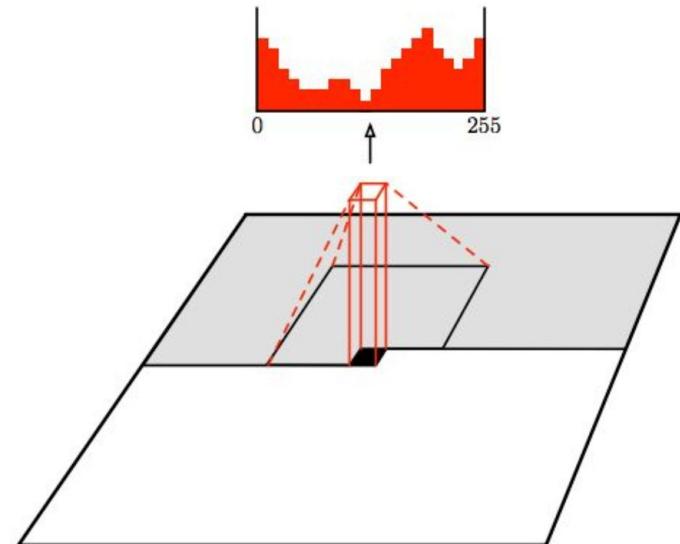


Figure copyright van der Oord et al., 2016. Reproduced with permission.

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

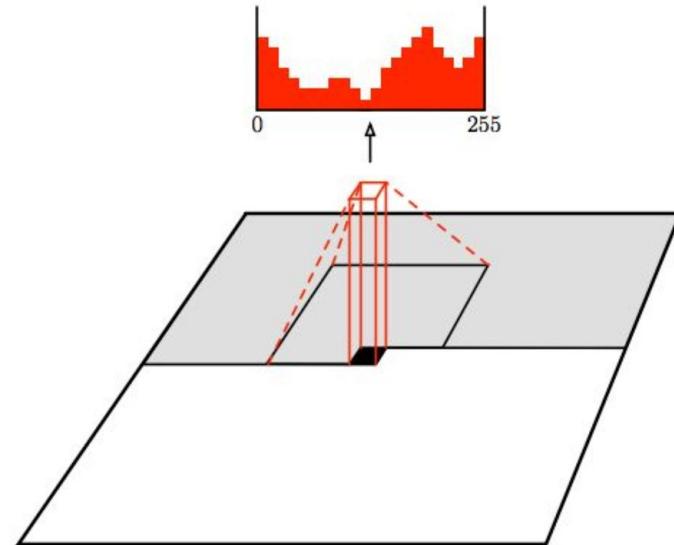


Figure copyright van der Oord et al., 2016. Reproduced with permission.

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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

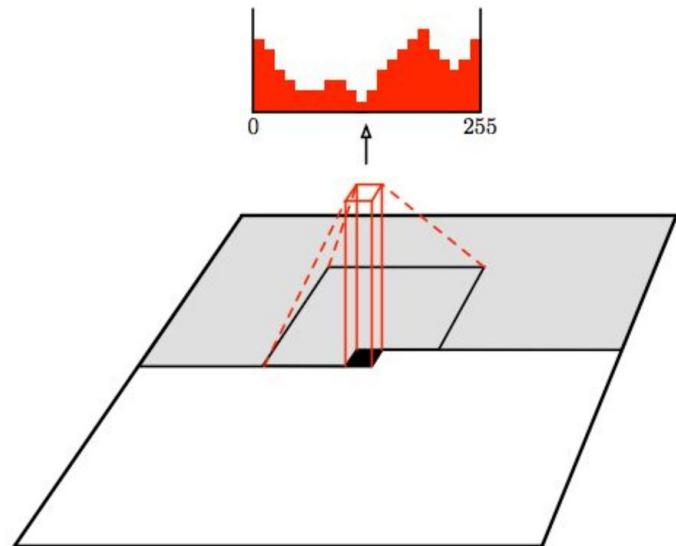
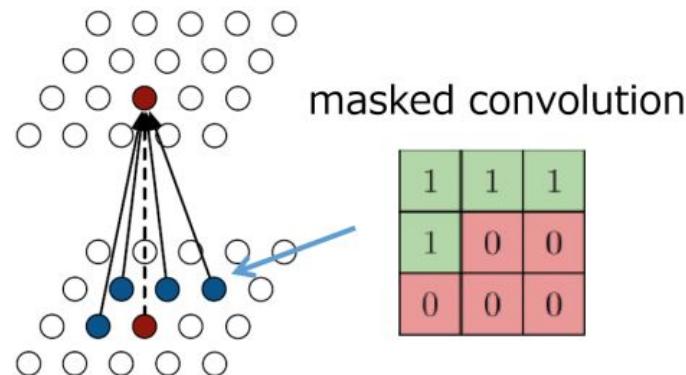


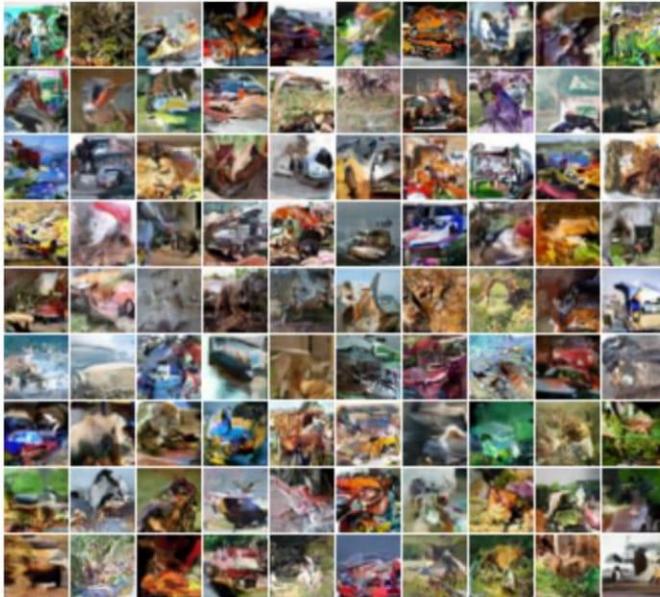
Figure copyright van der Oord et al., 2016. Reproduced with permission.

PixelCNN

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



Generation Samples



32x32 CIFAR-10



32x32 ImageNet

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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Quantitative Evaluation

Model	NLL Test
DBM 2hl [1]:	≈ 84.62
DBN 2hl [2]:	≈ 84.55
NADE [3]:	88.33
EoNADE 2hl (128 orderings) [3]:	85.10
EoNADE-5 2hl (128 orderings) [4]:	84.68
DLGM [5]:	≈ 86.60
DLGM 8 leapfrog steps [6]:	≈ 85.51
DARN 1hl [7]:	≈ 84.13
MADE 2hl (32 masks) [8]:	86.64
DRAW [9]:	≤ 80.97
PixelCNN:	81.30
Row LSTM:	80.54
Diagonal BiLSTM (1 layer, $h = 32$):	80.75
Diagonal BiLSTM (7 layers, $h = 16$):	79.20

Table 4. Test set performance of different models on MNIST in *nats* (negative log-likelihood). Prior results taken from [1] (Salakhutdinov & Hinton, 2009), [2] (Murray & Salakhutdinov, 2009), [3] (Uria et al., 2014), [4] (Raiko et al., 2014), [5] (Rezende et al., 2014), [6] (Salimans et al., 2015), [7] (Gregor et al., 2014), [8] (Germain et al., 2015), [9] (Gregor et al., 2015).

Model	NLL Test (Train)
Uniform Distribution:	8.00
Multivariate Gaussian:	4.70
NICE [1]:	4.48
Deep Diffusion [2]:	4.20
Deep GMMs [3]:	4.00
RIDE [4]:	3.47
PixelCNN:	3.14 (3.08)
Row LSTM:	3.07 (3.00)
Diagonal BiLSTM:	3.00 (2.93)

Table 5. Test set performance of different models on CIFAR-10 in *bits/dim*. For our models we give training performance in brackets. [1] (Dinh et al., 2014), [2] (Sohl-Dickstein et al., 2015), [3] (van den Oord & Schrauwen, 2014a), [4] personal communication (Theis & Bethge, 2015).

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 den Oord et al., NIPS 2016
- Salimans et al. 2017
(PixelCNN++)

Outline

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

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:

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

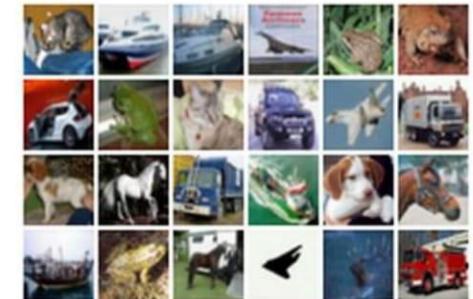
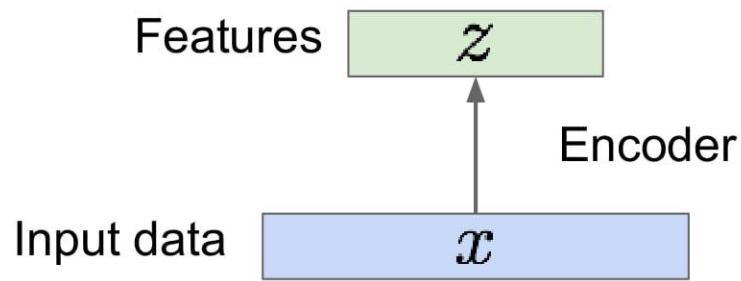
VAEs define intractable density function with latent \mathbf{z} :

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

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

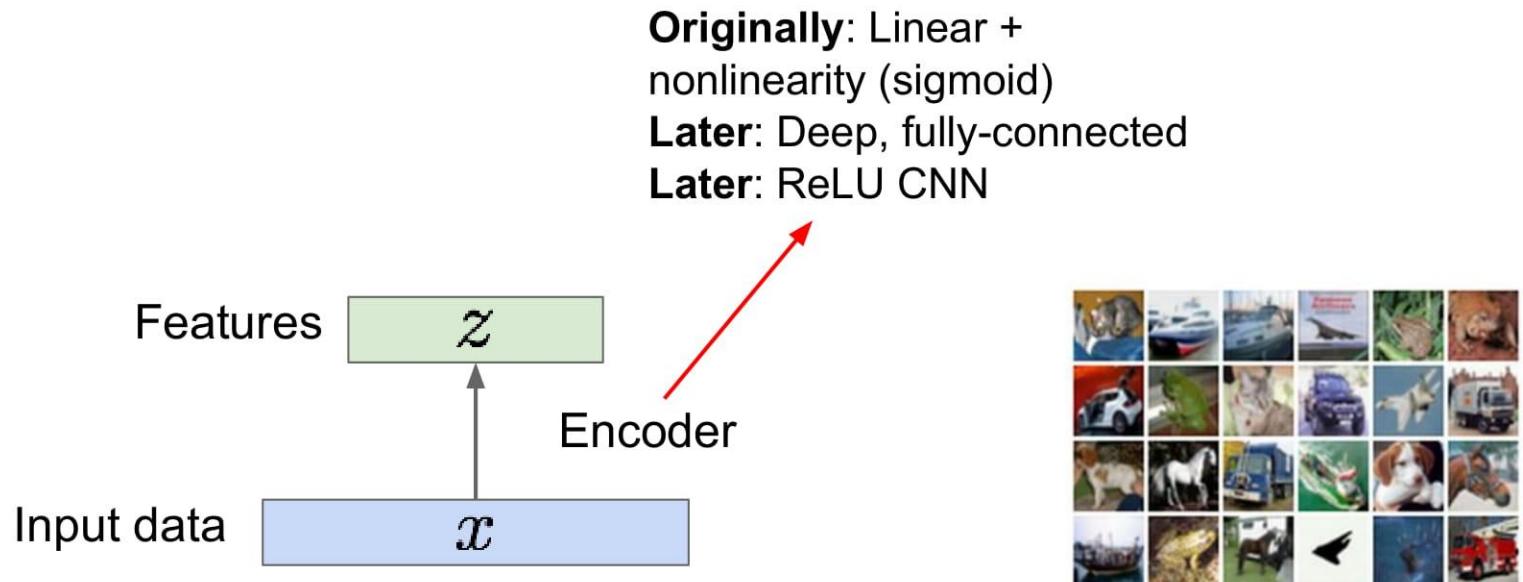
Some background first: Autoencoders

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

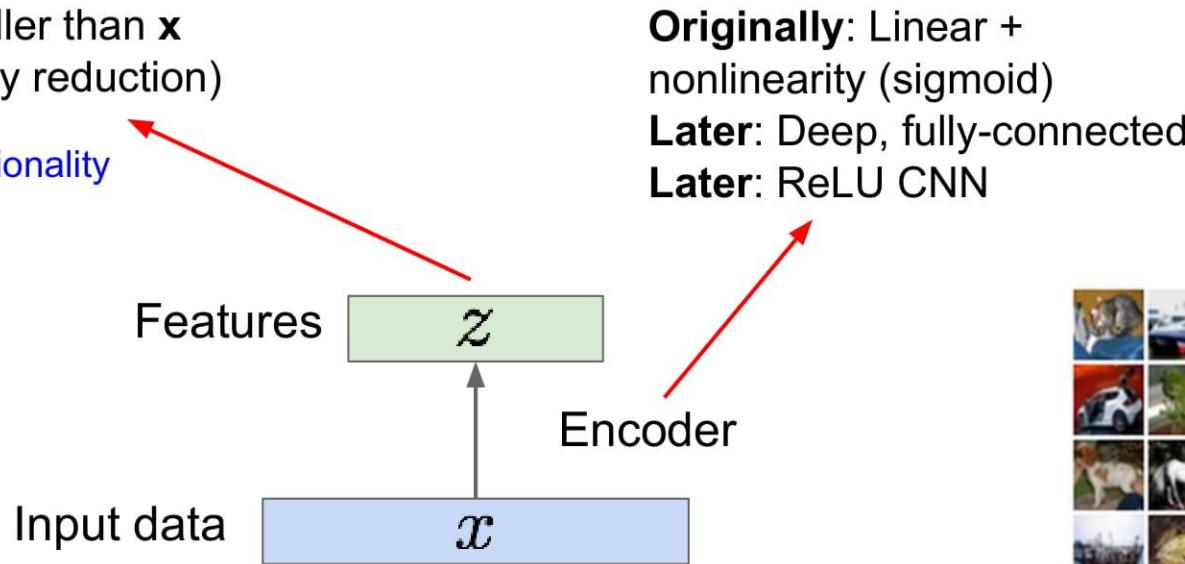


Some background first: Autoencoders

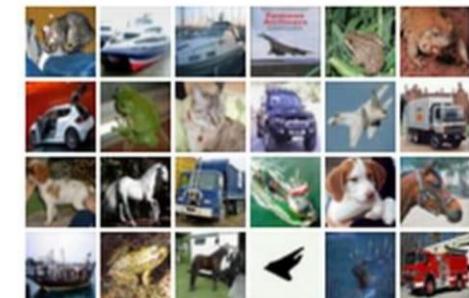
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than x
(dimensionality reduction)

Q: Why dimensionality reduction?



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



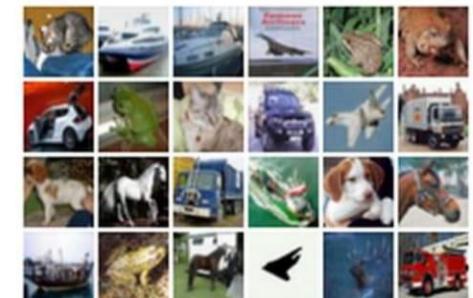
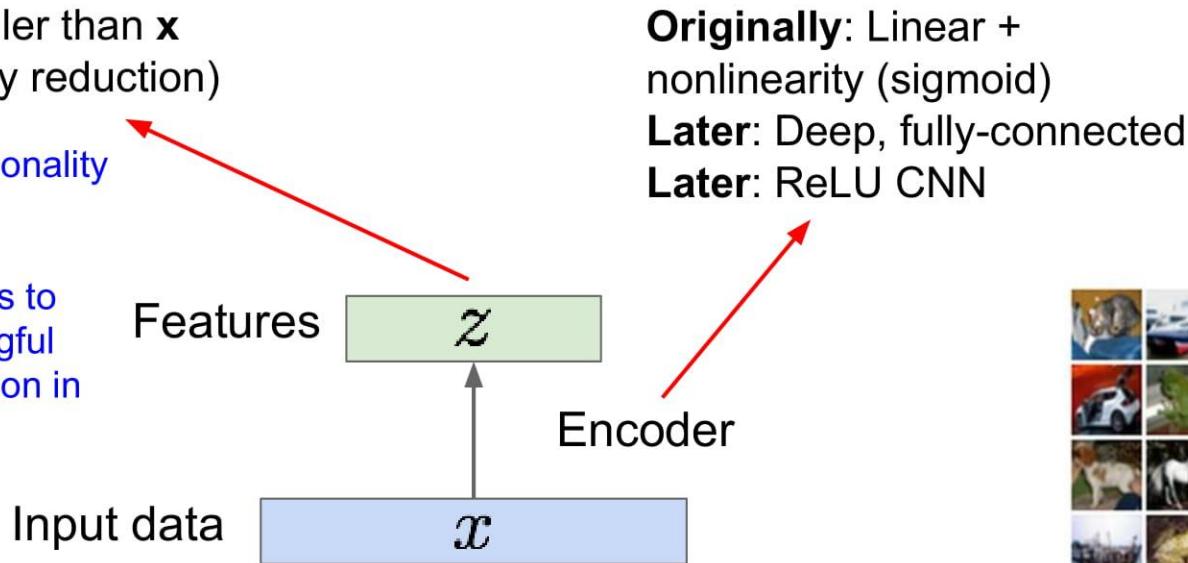
Some background first: Autoencoders

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

z usually smaller than x
(dimensionality reduction)

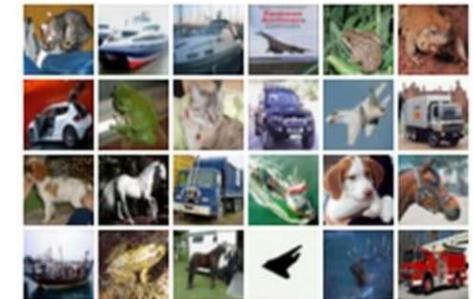
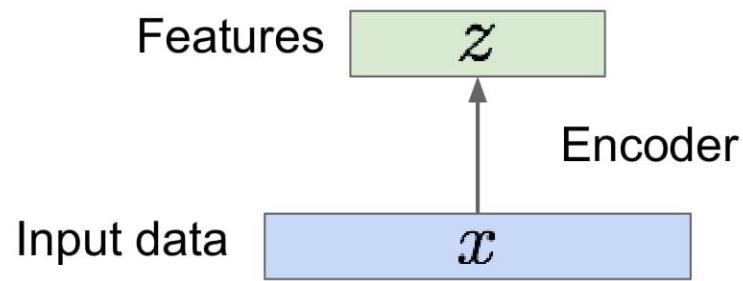
Q: Why dimensionality reduction?

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



Some background first: Autoencoders

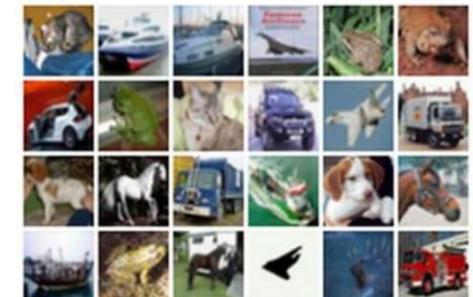
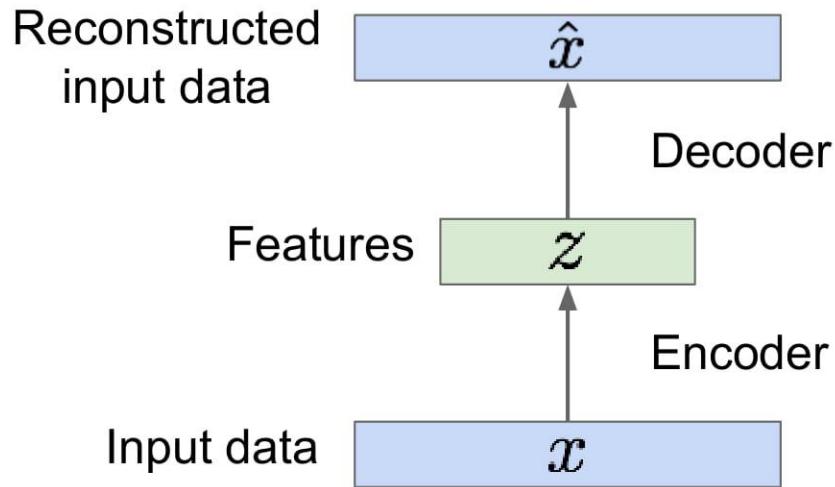
How to learn this feature representation?



Some background first: Autoencoders

How to learn this feature representation?

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

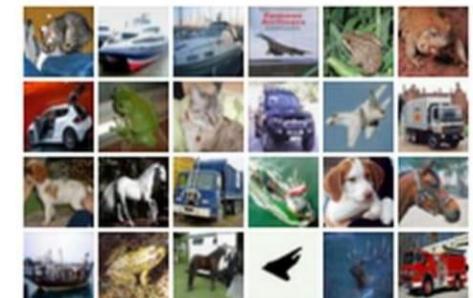
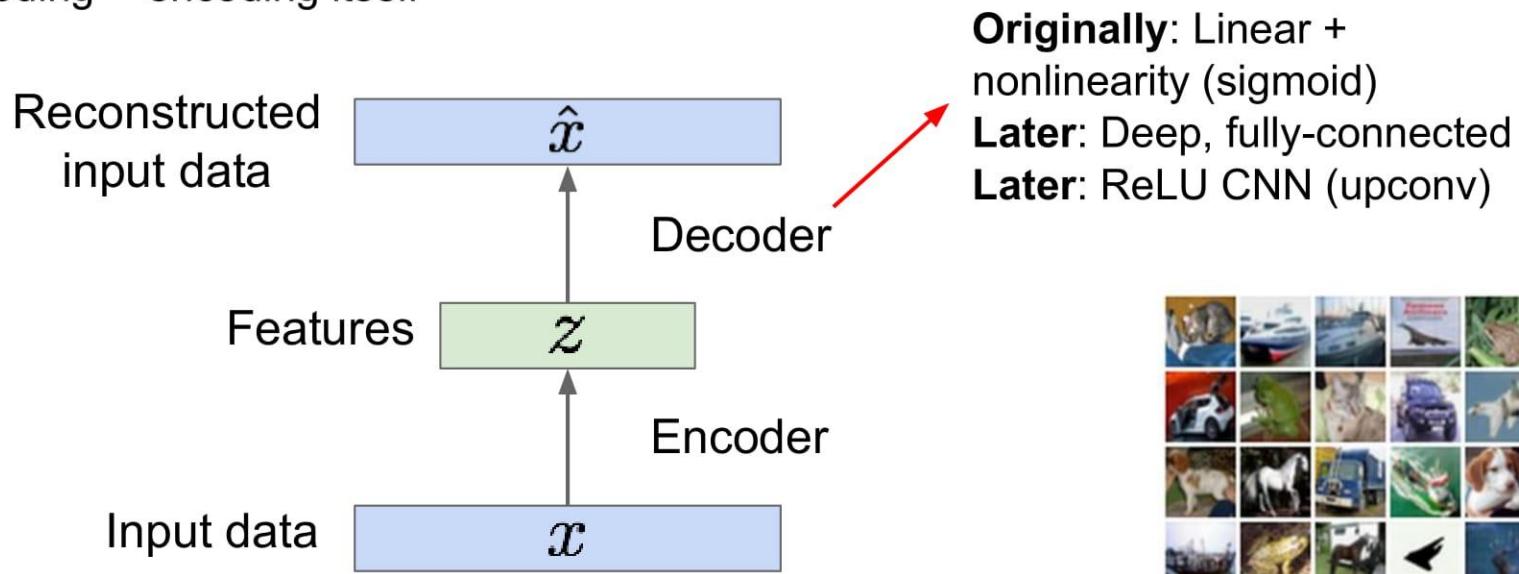


Some background first: Autoencoders

How to learn this feature representation?

Train such that features can be used to reconstruct original data

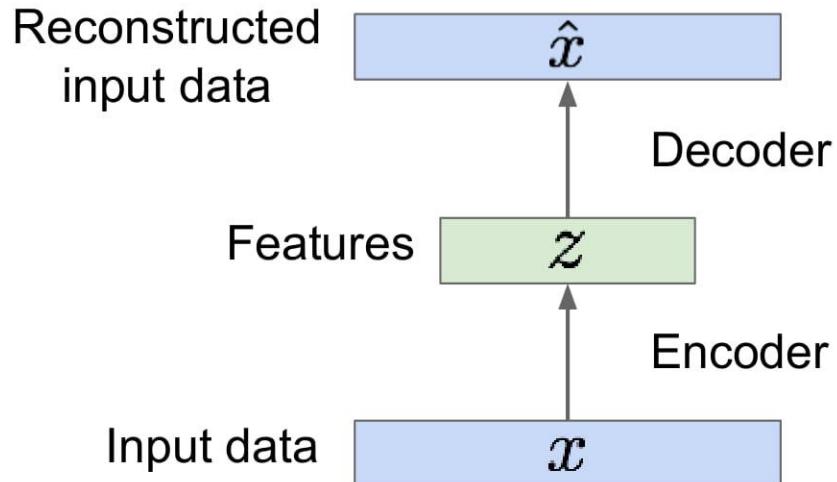
“Autoencoding” - encoding itself



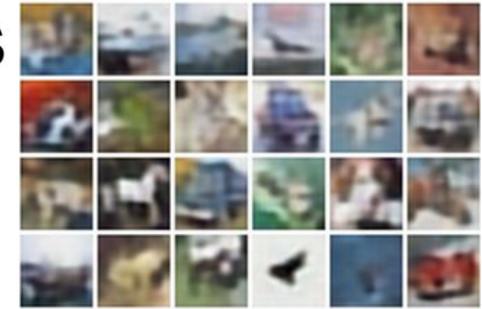
Some background first: Autoencoders

How to learn this feature representation?

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

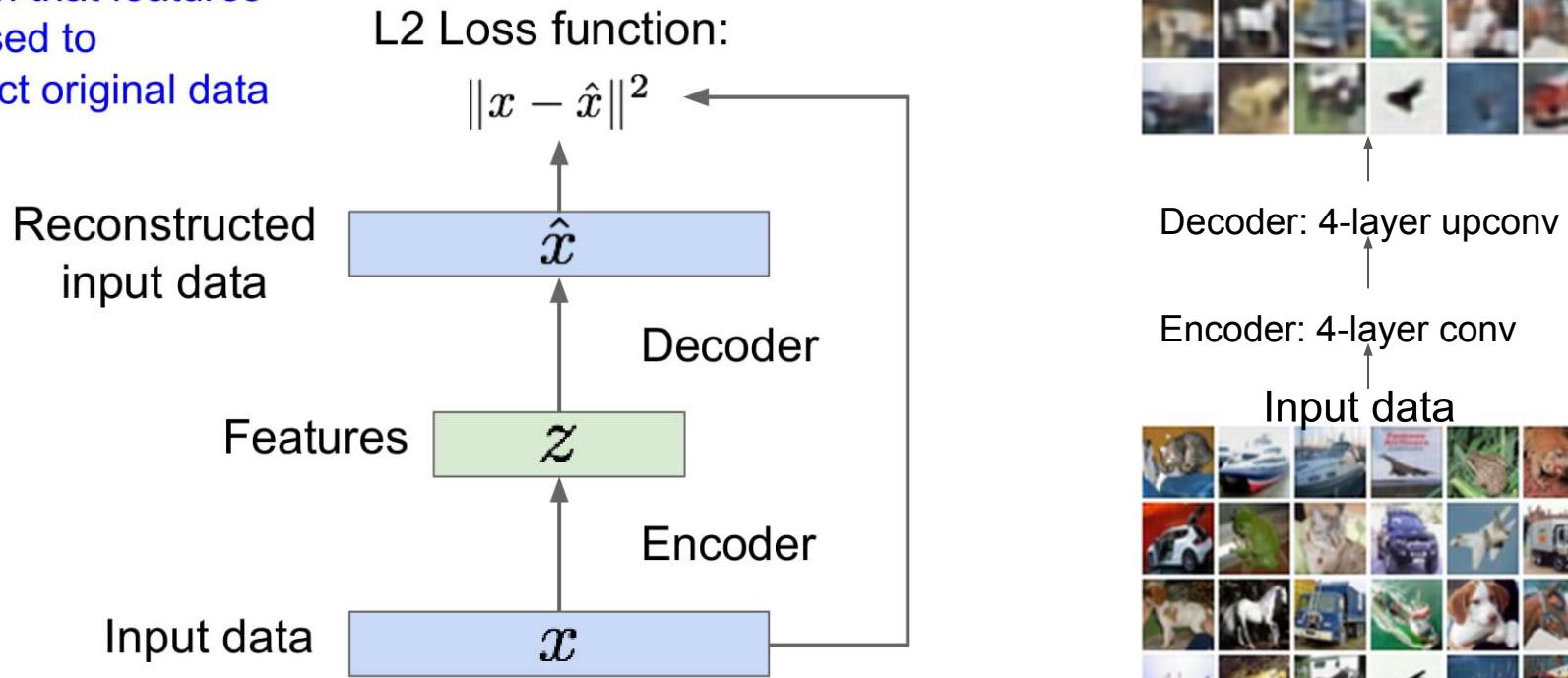


Reconstructed data



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

Doesn't use labels!

L2 Loss function:

$$\|x - \hat{x}\|^2$$

Reconstructed input data

Features

Input data

Decoder

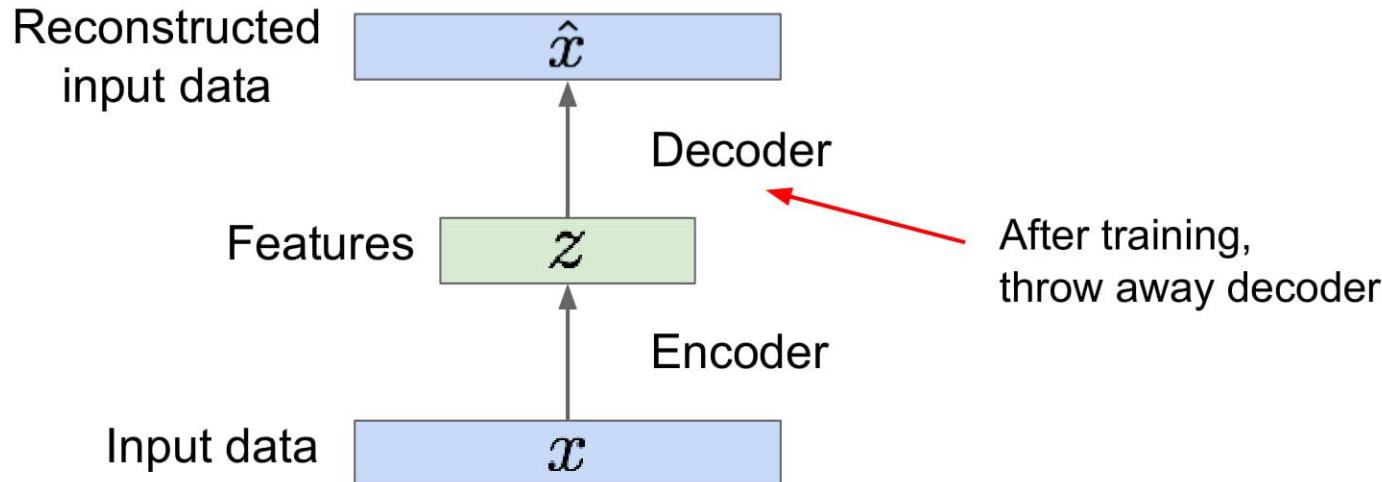
Encoder

z

x

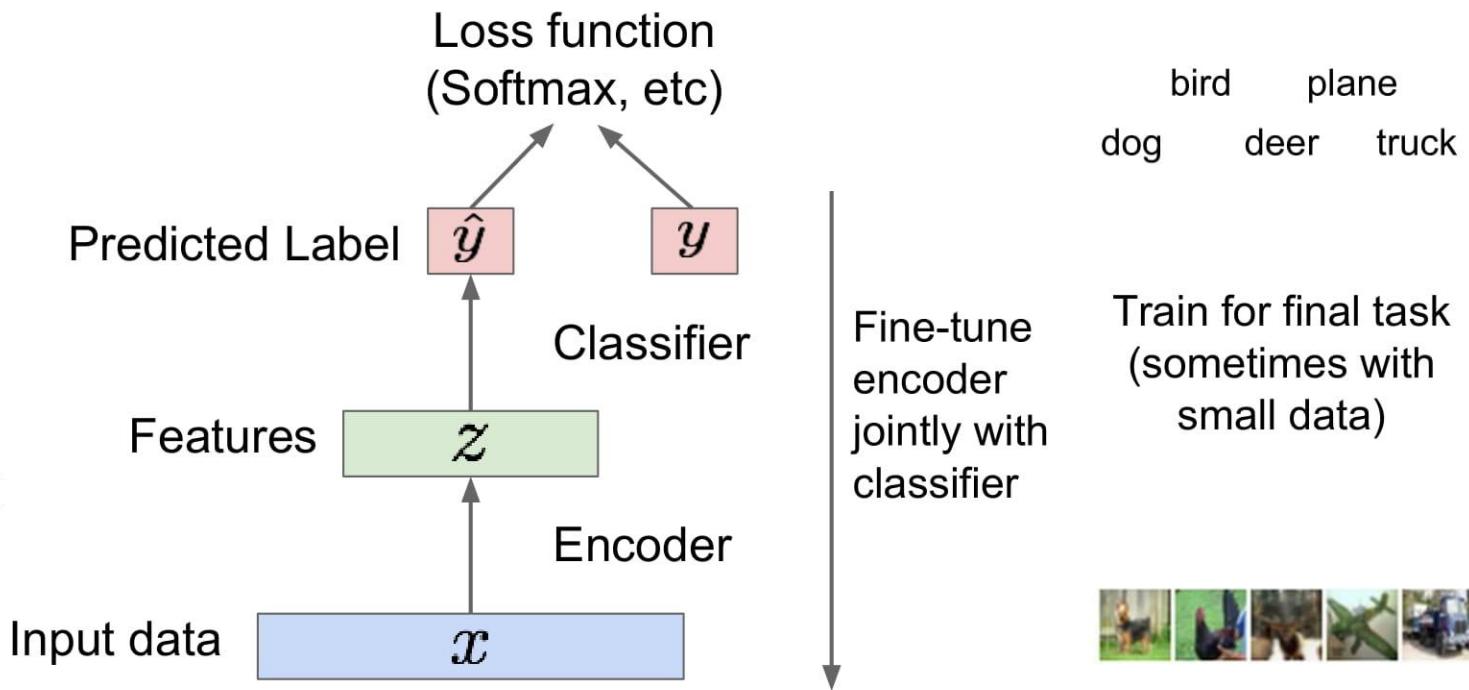


Some background first: Autoencoders

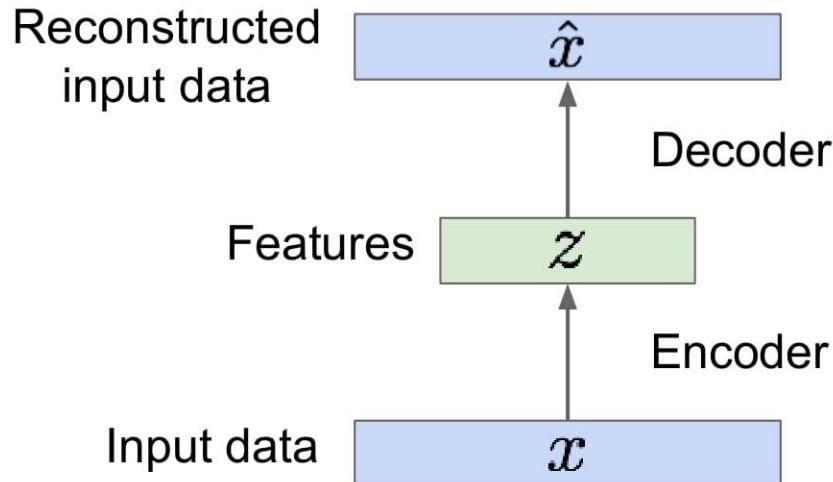


Some background first: Autoencoders

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. Can we generate new images from an autoencoder?

Variational Autoencoders

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

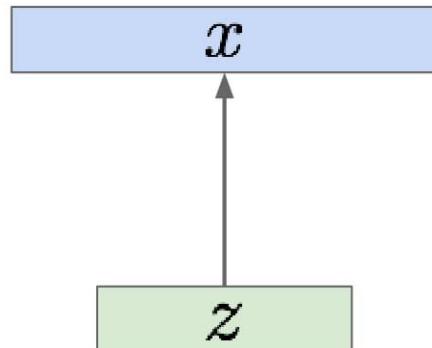
Variational Autoencoders

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

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



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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Variational Autoencoders

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

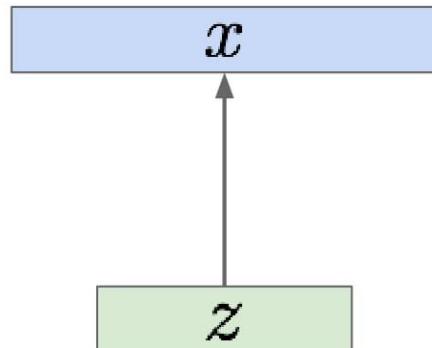
Assume training data $\{x^{(i)}\}_{i=1}^N$ is generated from underlying unobserved (latent) representation z

Sample from
true conditional

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

Sample from
true prior

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



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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Variational Autoencoders

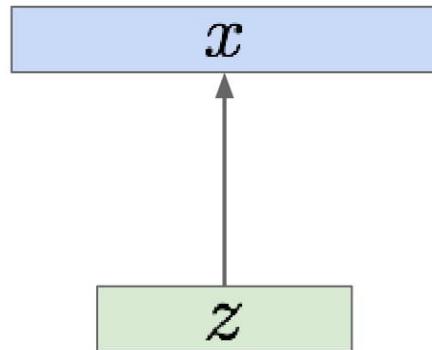
We want to estimate the true parameters θ^* of this generative model.

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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

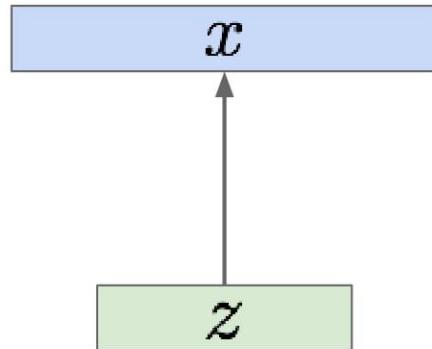
Variational Autoencoders

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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

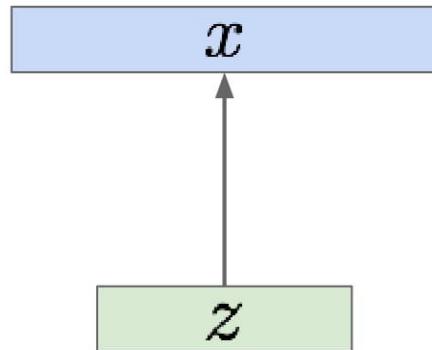
Variational Autoencoders

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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

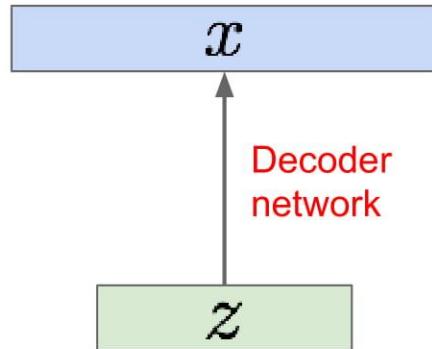
Variational Autoencoders

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.

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

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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

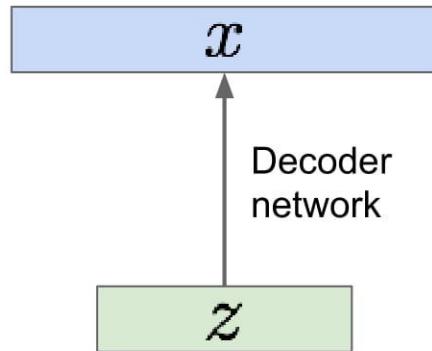
Variational Autoencoders

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 to train the model?

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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

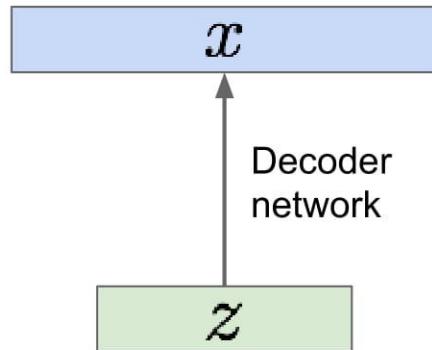
Variational Autoencoders

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 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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

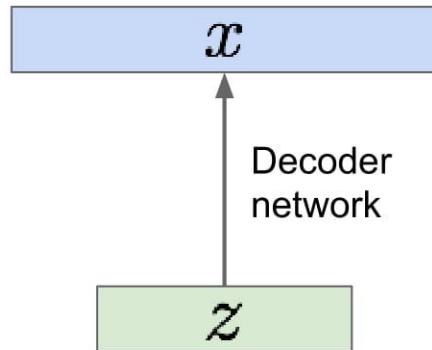
Variational Autoencoders

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 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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

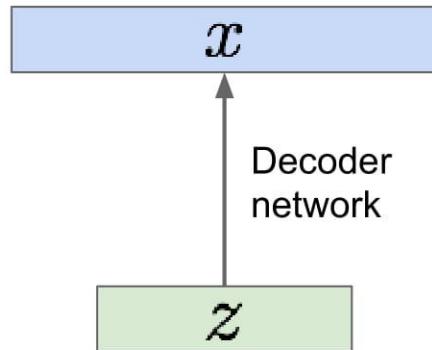
Variational Autoencoders

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 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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

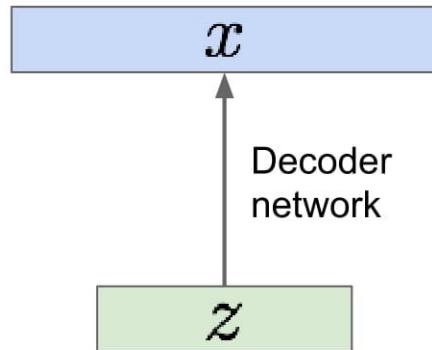
Variational Autoencoders

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 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

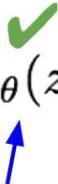
Data likelihood: $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Variational Autoencoders: Intractability

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



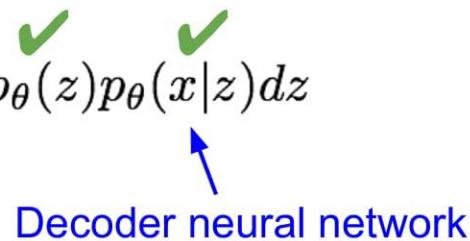
Simple Gaussian prior

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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Variational Autoencoders: Intractability

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

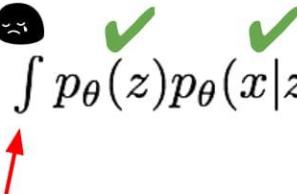


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Variational Autoencoders: Intractability



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

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

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

Variational Autoencoders: Intractability

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

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

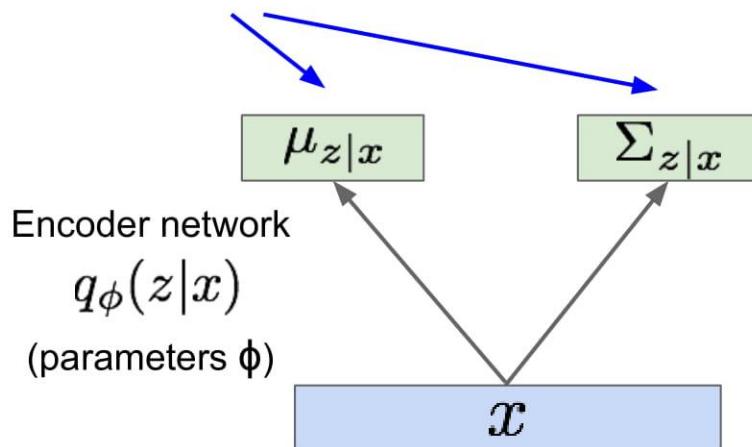
Solution: In addition to decoder network modeling $p_{\theta}(x|z)$, define additional encoder network $q_{\phi}(z|x)$ that approximates $p_{\theta}(z|x)$

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

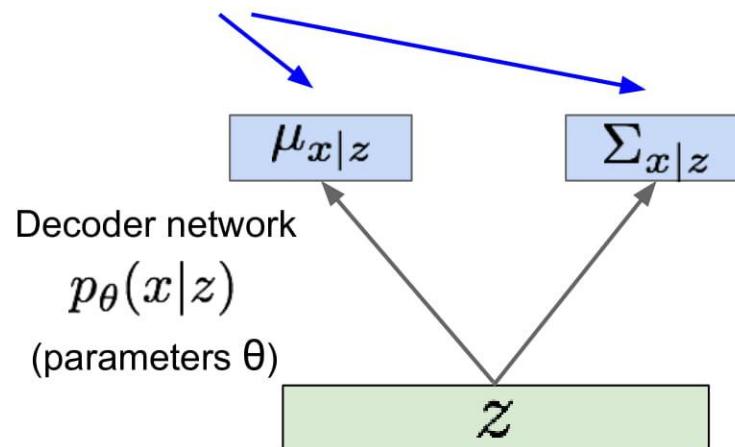
Variational Autoencoders

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

Mean and (diagonal) covariance of $\mathbf{z} | \mathbf{x}$



Mean and (diagonal) covariance of $\mathbf{x} | \mathbf{z}$

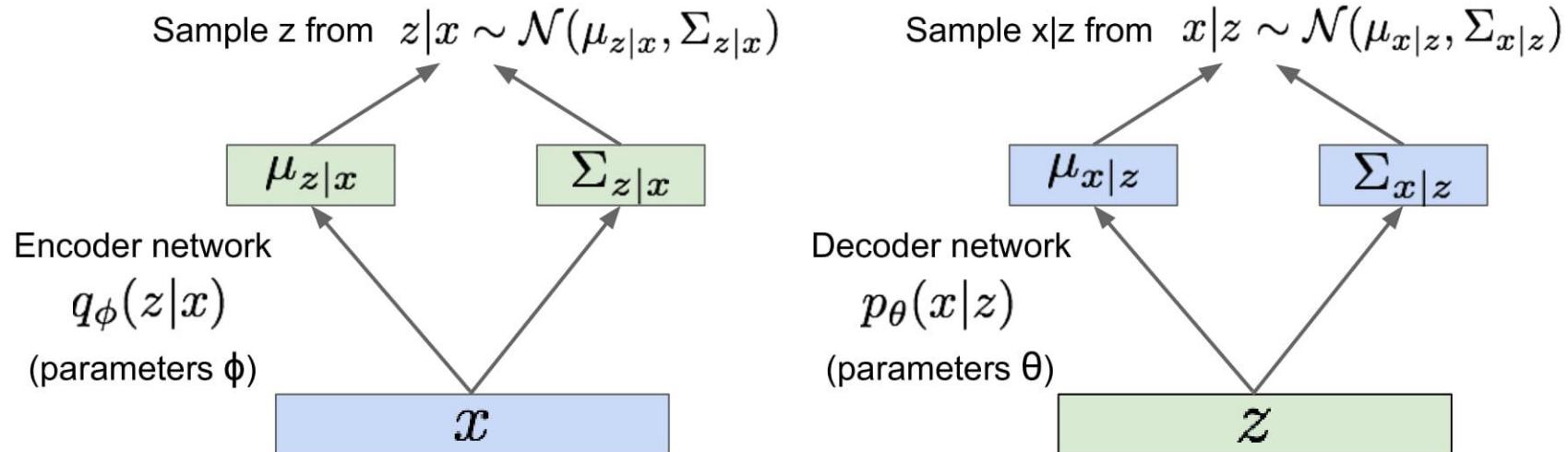


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Variational Autoencoders

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

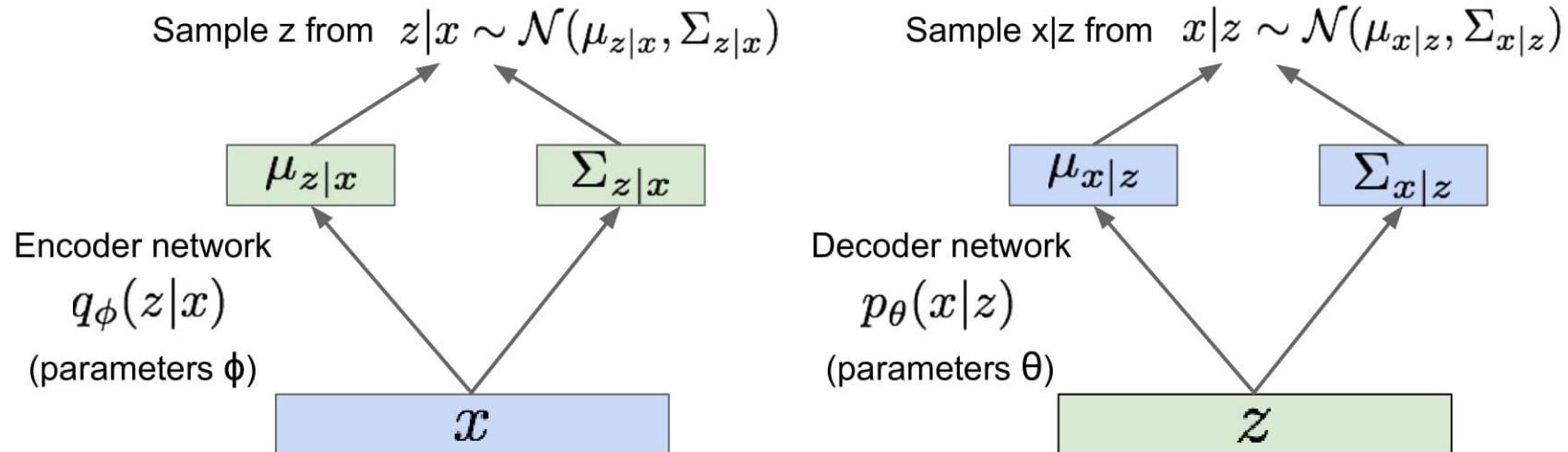


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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:

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

Variational Autoencoders

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

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

Variational Autoencoders

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

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

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

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

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

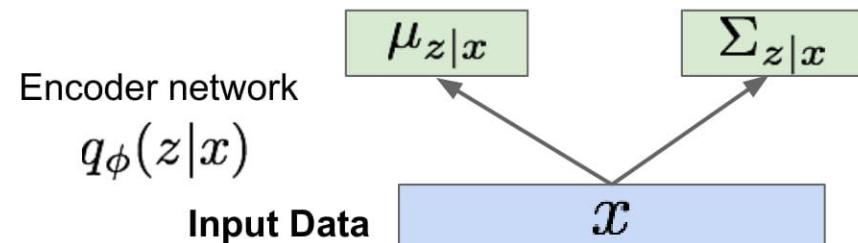
Let's look at computing the bound (forward pass) for a given minibatch of input data

Input Data 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

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

Make approximate posterior distribution close to prior

Encoder network

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

Input Data

$$\mu_{z|x}$$

$$\Sigma_{z|x}$$

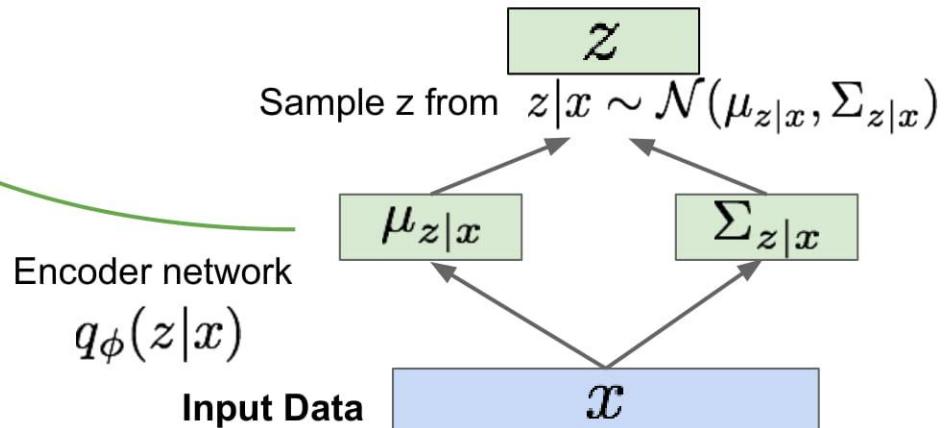
x

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

Make approximate posterior distribution close to prior

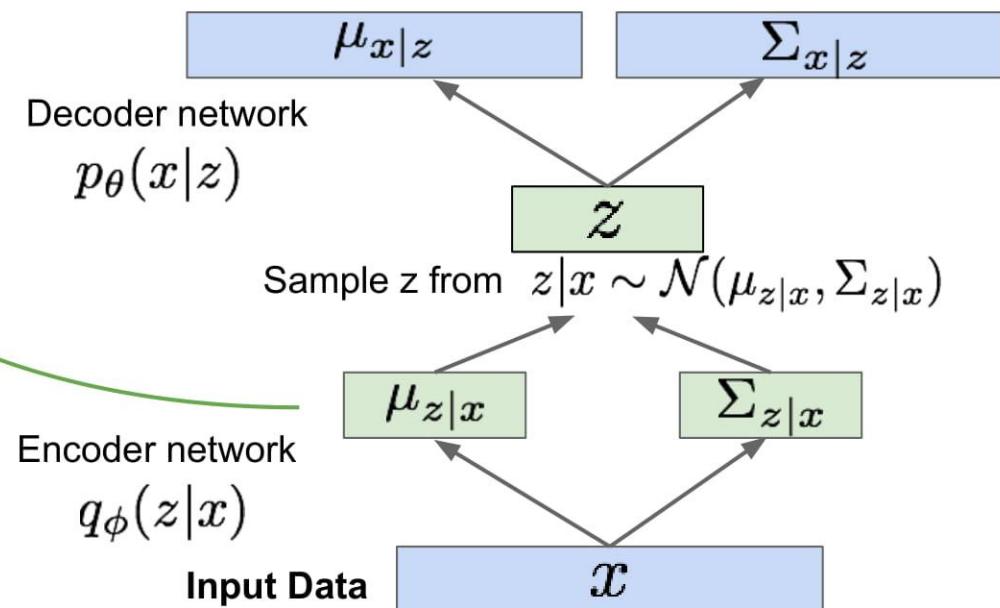


Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

Make approximate posterior distribution close to prior

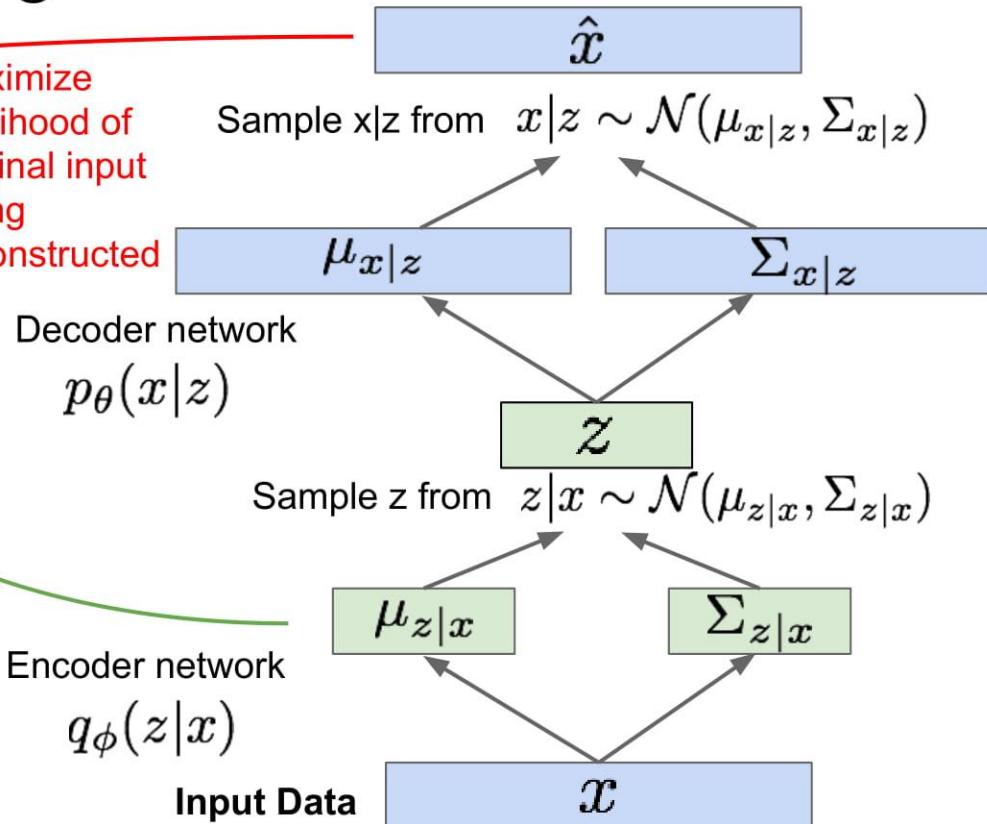


Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

Maximize likelihood of original input being reconstructed



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

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

Maximize likelihood of original input being reconstructed

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

Encoder network

$q_\phi(z|x)$

Input Data
 x

Sample $x|z$ from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$

$\mu_{x|z}$

\hat{x}

$\Sigma_{x|z}$

Sample $z|x$ from $z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$

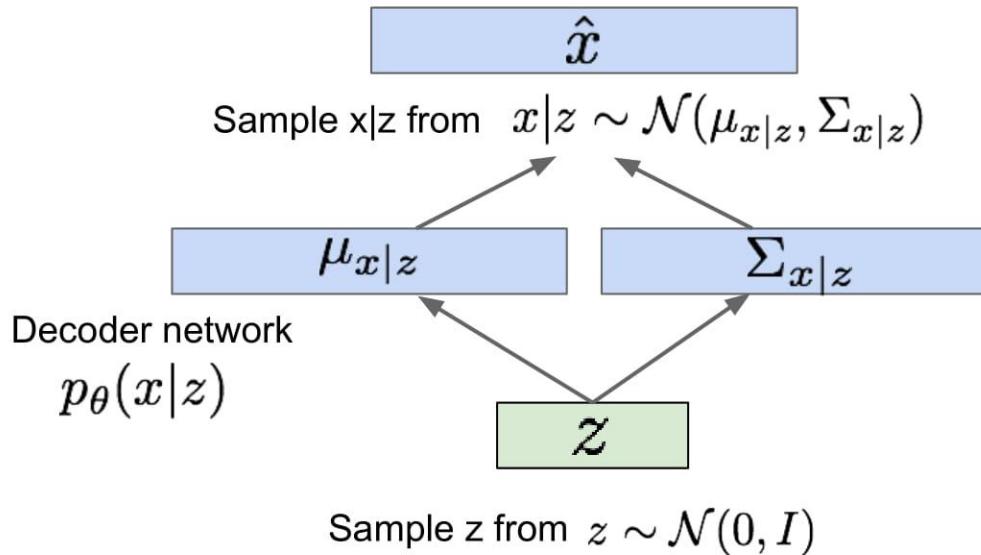
$\mu_{z|x}$

$\Sigma_{z|x}$

x

Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!

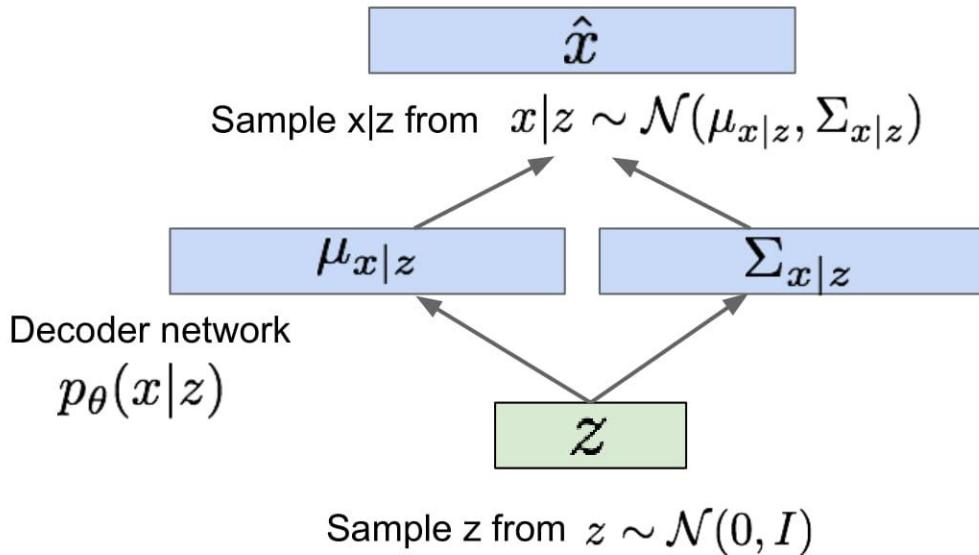


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

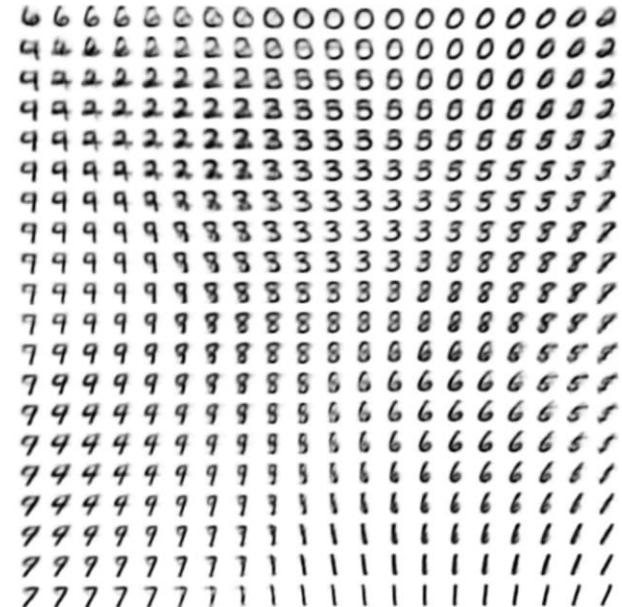
Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!



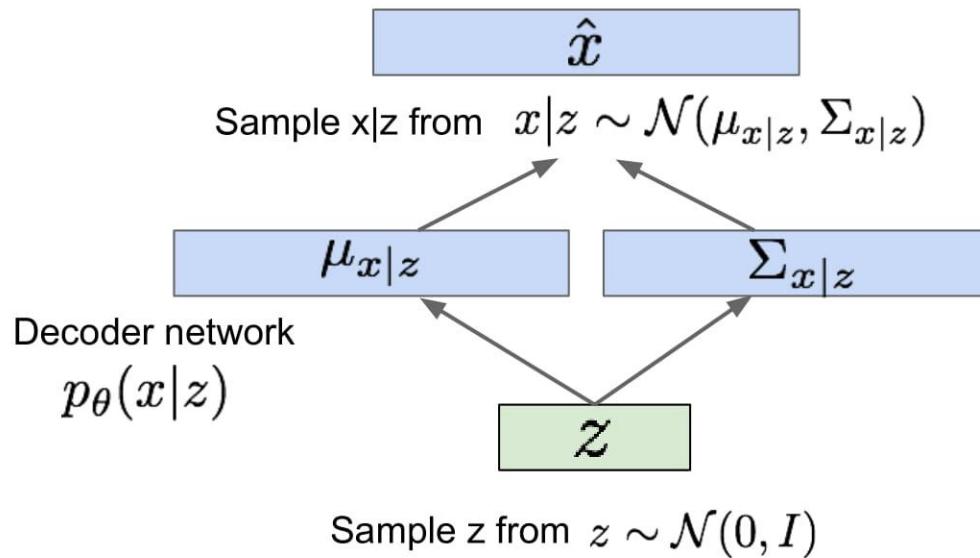
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014



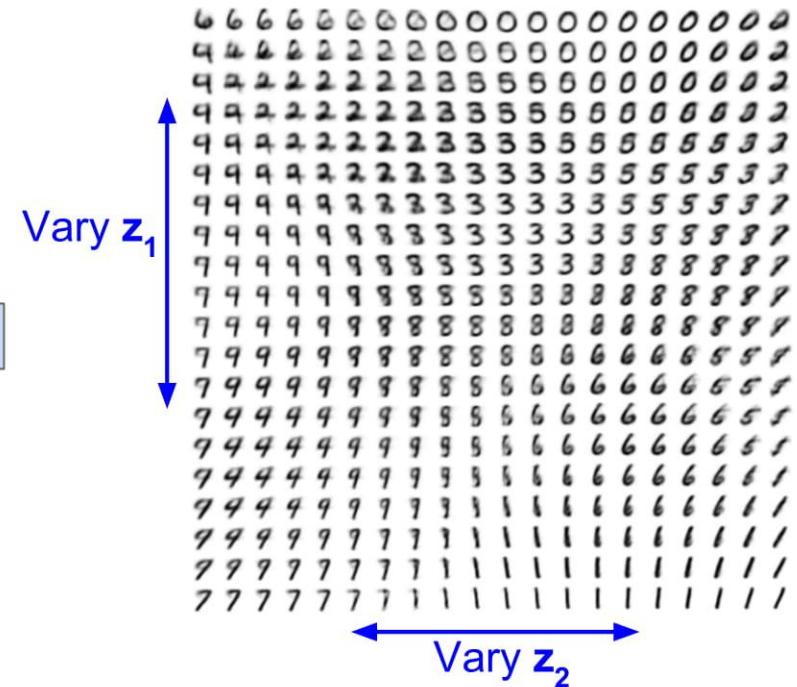
Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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
 \uparrow
Vary \mathbf{z}_1
 \downarrow



Head pose
 \longleftrightarrow
Vary \mathbf{z}_2

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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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 $q_\phi(\mathbf{z}|\mathbf{x})$!

Degree of smile
Vary \mathbf{z}_1

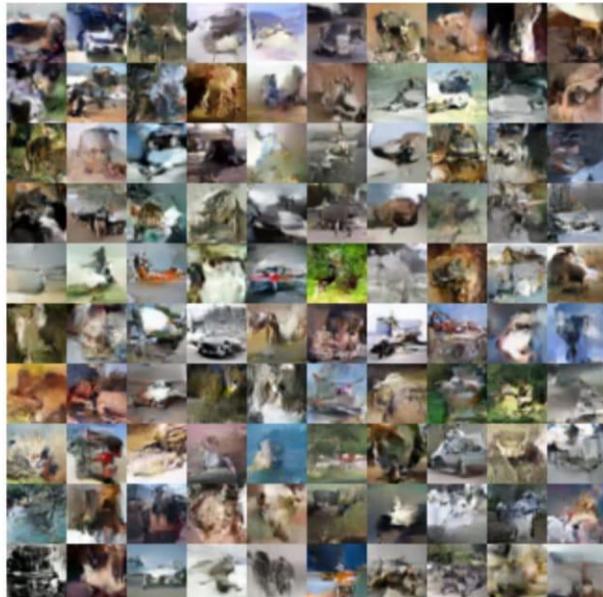


Vary \mathbf{z}_2 Head pose

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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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.

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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
- 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
- Incorporating structure in latent variables

Outline

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

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!

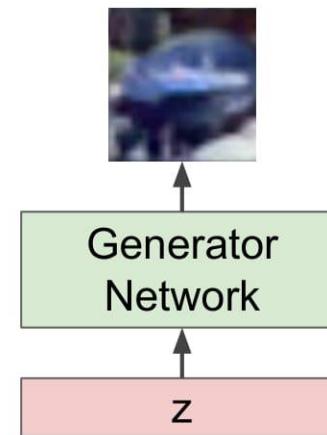
Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

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

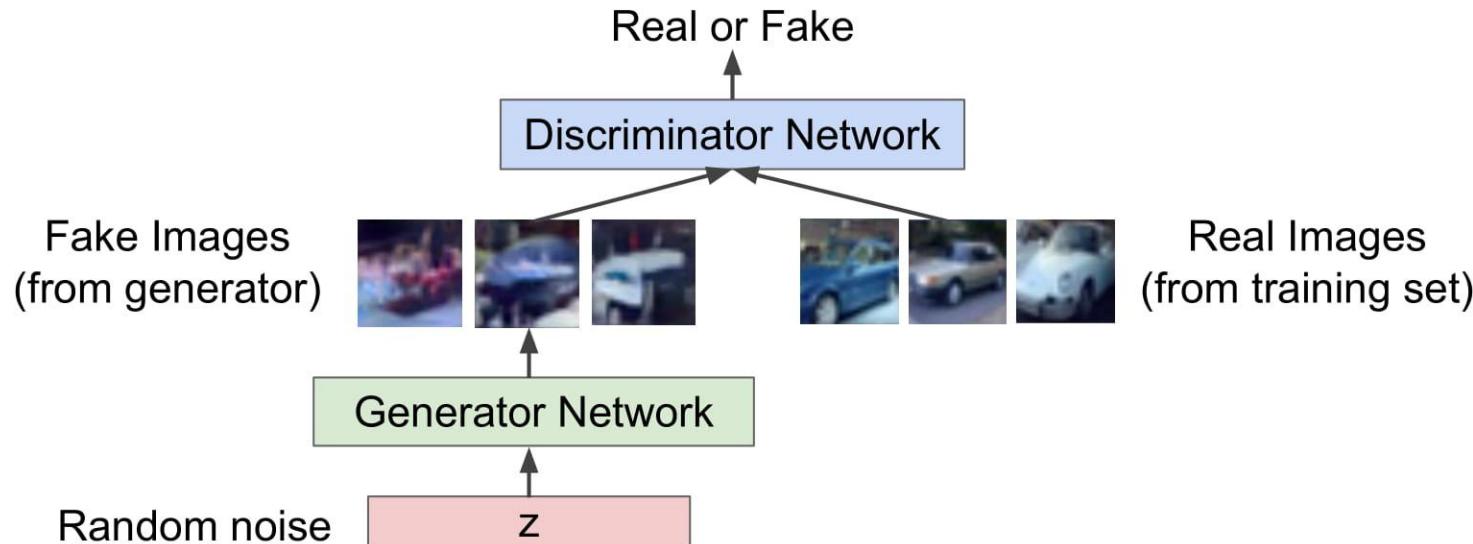
Discriminator network: try to distinguish between real and fake images

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.

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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:

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

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

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

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

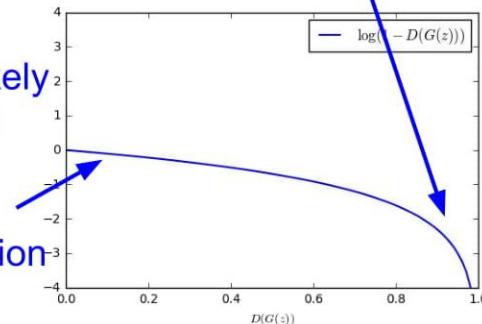
2. **Gradient descent** on generator

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

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

When sample is likely fake, want to learn from it to improve generator. 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

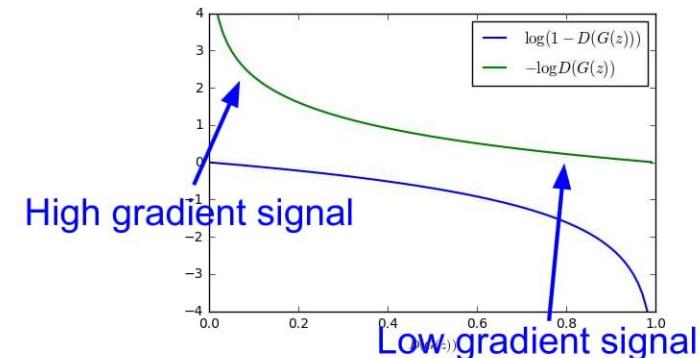
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

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

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

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

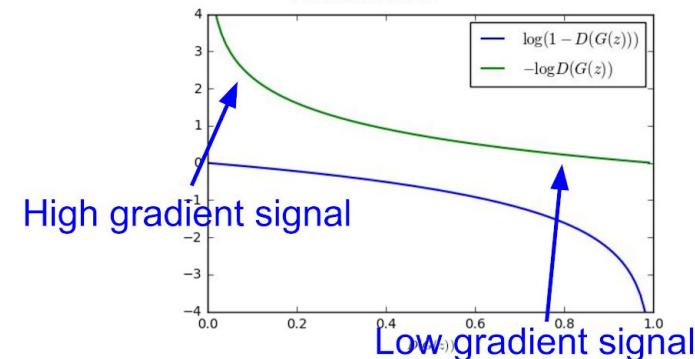
2. Instead: **Gradient ascent** on generator, different objective

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

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.

Aside: Jointly training two networks is challenging, can be unstable. Choosing objectives with better loss landscapes helps training, is an active area of research.



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:

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

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

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

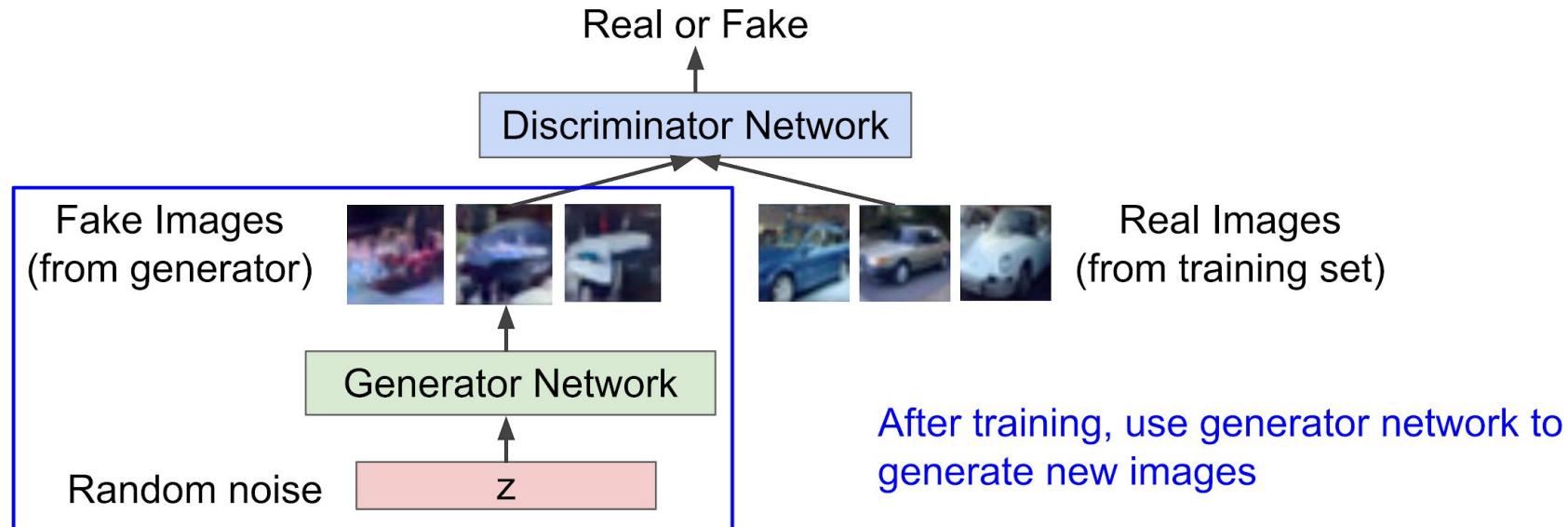
Recent work (e.g.
Wasserstein GAN)
alleviates this
problem, better
stability!

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

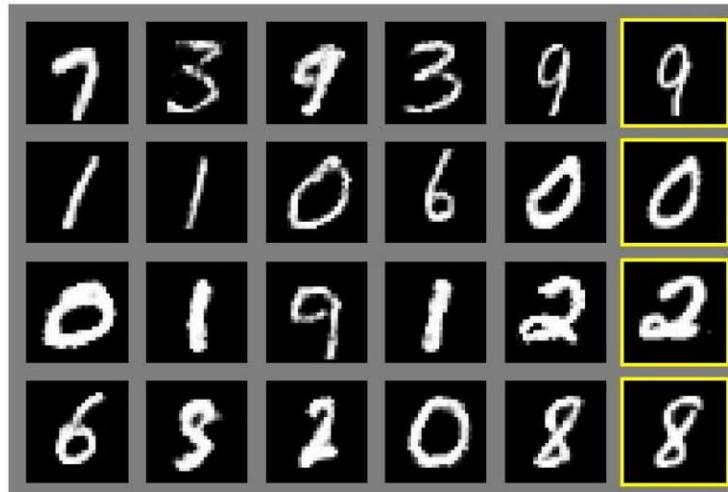


Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Generative Adversarial Nets

Generated samples



Nearest neighbor from training set

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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Generative Adversarial Nets

Generated samples (CIFAR-10)



Nearest neighbor from training set

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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Generative Adversarial Nets: Convolutional Architectures

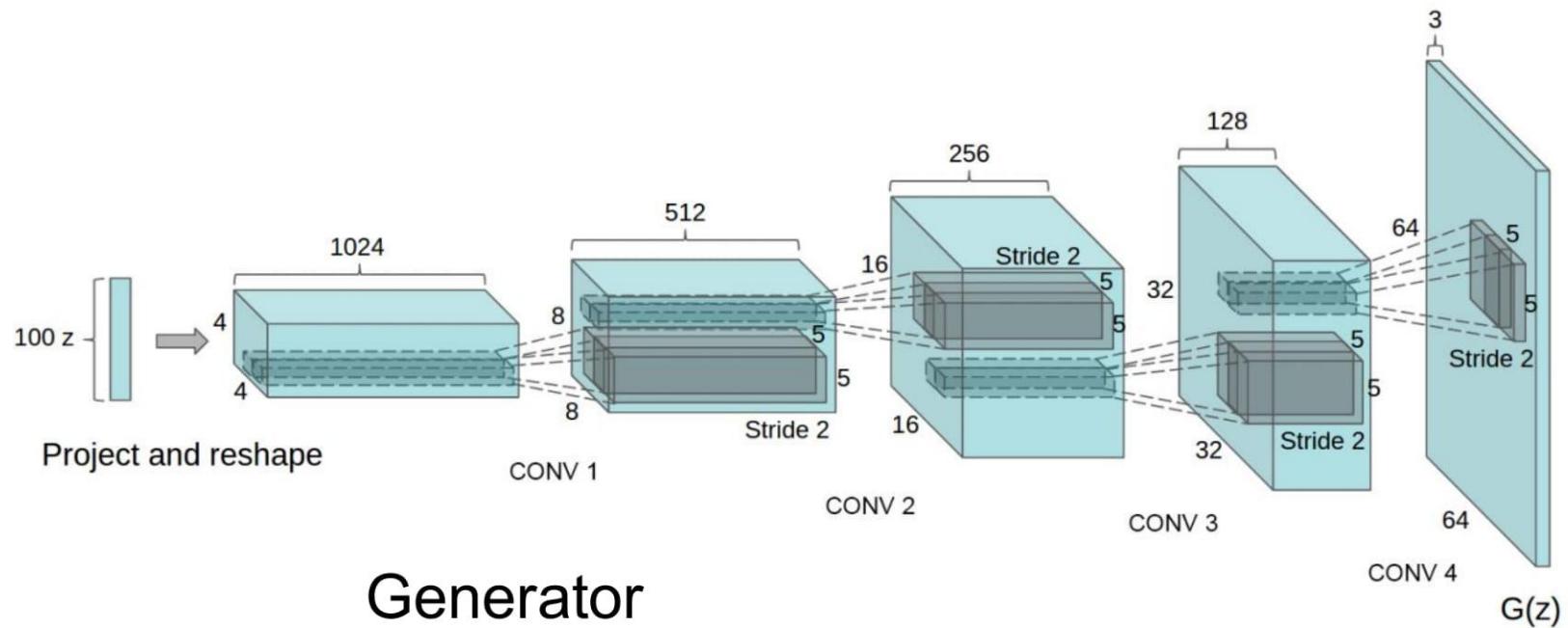
Generator is an upsampling network with fractionally-strided convolutions

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.

Generative Adversarial Nets: Convolutional Architectures



Generator

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

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Generative Adversarial Nets: Convolutional Architectures

Samples
from the
model look
much
better!



Radford et al,
ICLR 2016

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Generative Adversarial Nets: Convolutional Architectures

Interpolating
between
random
points in latent
space



Radford et al,
ICLR 2016

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

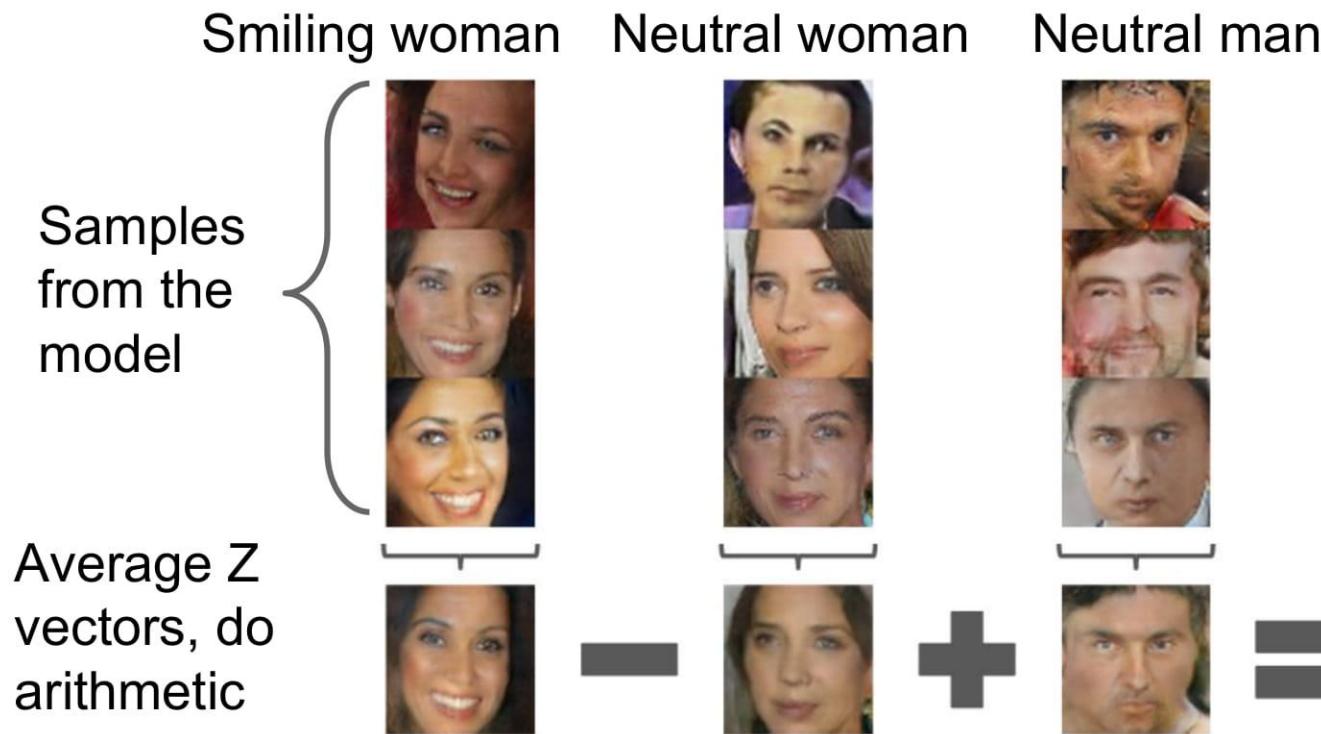


Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung



Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Generative Adversarial Nets: Interpretable Vector Math



Radford et al, ICLR 2016

Smiling Man





Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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”

- 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

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

- 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

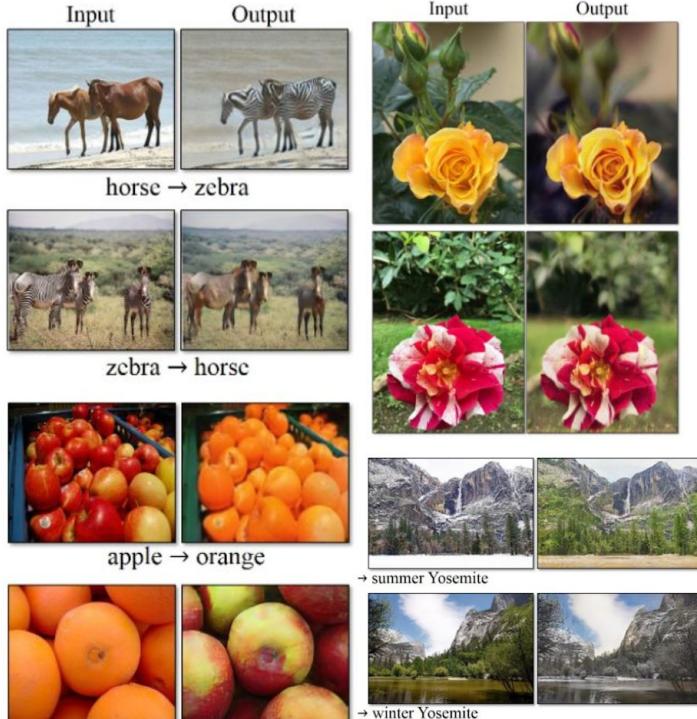
<https://github.com/hindupuravinash/the-gan-zoo>

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung



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/>

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

Recap

Generative Models

- PixelRNN and PixelCNN Explicit density model, optimizes exact likelihood, good samples. But inefficient sequential generation.
- Variational Autoencoders (VAE) Optimize variational lower bound on likelihood. Useful latent representation, inference queries. But current sample quality not the best.
- Generative Adversarial Networks (GANs) Game-theoretic approach, best samples! But can be tricky and unstable to train, no inference queries.

Recap

Generative Models

- PixelRNN and PixelCNN Explicit density model, optimizes exact likelihood, good samples. But inefficient sequential generation.
- Variational Autoencoders (VAE) Optimize variational lower bound on likelihood. Useful latent representation, inference queries. But current sample quality not the best.
- Generative Adversarial Networks (GANs) Game-theoretic approach, best samples! But can be tricky and unstable to train, no inference queries.

Also recent work in combinations of these types of models! E.g. Adversarial Autoencoders (Makhzani 2015) and PixelVAE (Gulrajani 2016)