

Lecture 13: Generative Models

Administrative

- A3 is out. Due May 25.
- Milestone was due May 10th
 - Read website page for milestone requirements.
 - Need to Finish data preprocessing and initial results by then.
- Midterm and A2 grades will be out this week

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

Examples: Classification,
regression, object detection,
semantic segmentation, image
captioning, etc.

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

Examples: Classification,
regression, object detection,
semantic segmentation, image
captioning, etc.



→ Cat

Classification

[This image](#) is CC0 public domain

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

Examples: Classification,
regression, object detection,
semantic segmentation, image
captioning, etc.



A cat sitting on a suitcase on the floor

Image captioning

Caption generated using [neuraltalk2](#)
Image is [CC0 Public domain](#)

Supervised vs Unsupervised Learning

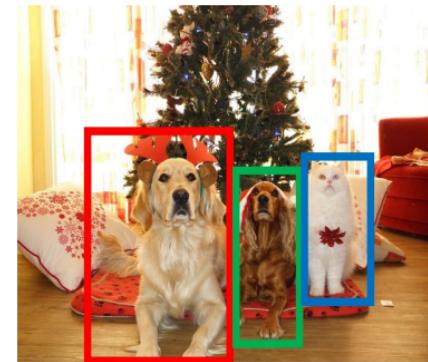
Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

Examples: Classification,
regression, object detection,
semantic segmentation, image
captioning, etc.



DOG, DOG, CAT

Object Detection

This image is CC0 public domain

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

Examples: Classification,
regression, object detection,
semantic segmentation, image
captioning, etc.



GRASS, CAT,
TREE, SKY

Semantic Segmentation

Supervised vs Unsupervised Learning

Unsupervised Learning

Data: x

Just data, **no labels!**

Goal: Learn some underlying
hidden *structure* of the data

Examples: Clustering,
dimensionality reduction, feature
learning, density estimation, etc.

Supervised vs Unsupervised Learning

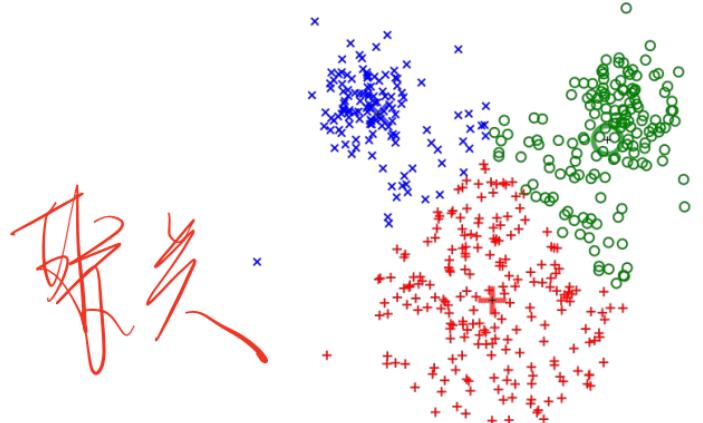
Unsupervised Learning

Data: x

Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, density estimation, etc.



K-means clustering

This image is CC0 public domain

Supervised vs Unsupervised Learning

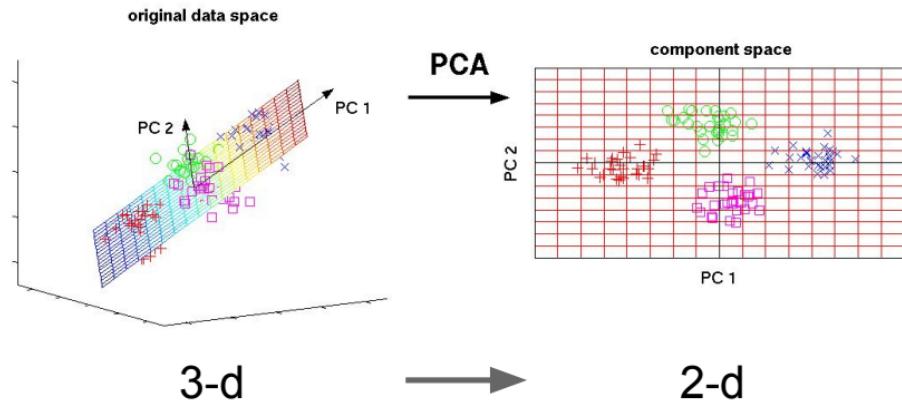
Unsupervised Learning

Data: x

Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, density estimation, etc.



Principal Component Analysis
(Dimensionality reduction)



This image from Matthias Scholz is CC0 public domain

Supervised vs Unsupervised Learning

~~Supervised~~ Unsupervised

Unsupervised Learning

Data: x

Just data, no labels!

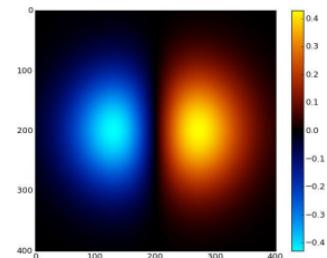
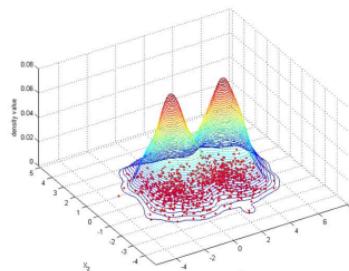
Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, density estimation, etc.



Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

Modeling $p(x)$

2-d density images [left](#) and [right](#)
are [CC0 public domain](#)

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

Examples: Classification,
regression, object detection,
semantic segmentation, image
captioning, etc.

Unsupervised Learning

Data: x

Just data, no labels!

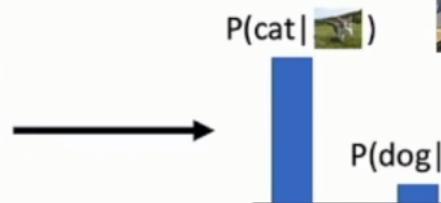
Goal: Learn some underlying
hidden *structure* of the data

Examples: Clustering,
dimensionality reduction, density
estimation, etc.

Discriminative vs Generative Models

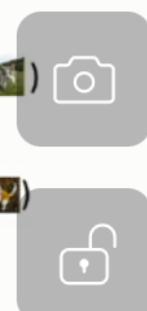
半监督式
模型
学习
输入
输出
分布
不同
类别的
概率
分布

Discriminative Model:
Learn a probability distribution $p(y|x)$



监督式
模型
输入
输出
分布
相同
类别的
概率
分布

Generative Model:
Learn a probability distribution $p(x)$



条件生成
模型
输入
输出
分布

Conditional Generative Model: Learn $p(x|y)$

Discriminative model: the possible labels for each input "compete" for probability mass. But no competition between images

Justin Johnson

Lecture 19 - 17

November 20, 2019

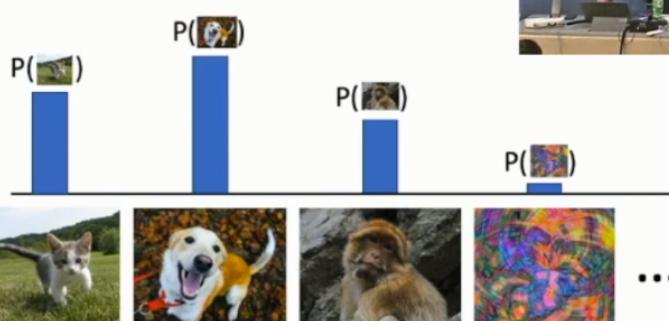
Just Johnson CC-BY Public Domain

半监督模型即使对于不合理输入（数据不存在）
也会输出结果，不能反映不合理输入。不对

Discriminative vs Generative Models

识别
绝不含
任何的
输入

Discriminative Model:
Learn a probability distribution $p(y|x)$



识别
任何的
输入

Generative Model:
Learn a probability distribution $p(x)$

Generative model: All possible images compete with each other for probability mass

识别
任何的
输入

Conditional Generative Model: Learn $p(x|y)$

Requires deep image understanding! Is a dog more likely to sit or stand? How about 3-legged dog vs 3-armed monkey?

Justin Johnson

Lecture 19 - 21

November 20, 2019

好的识别方法：用复杂的但训练数据集中不有的图像去测试，也能输出较高的概率

Discriminative vs Generative Models

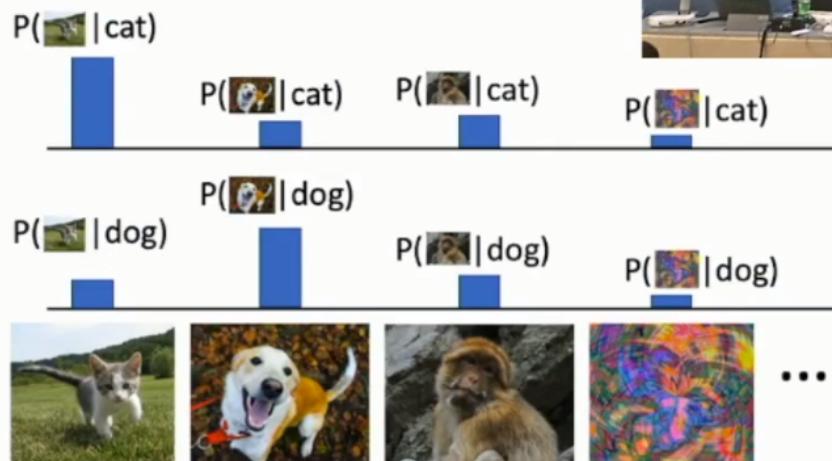


Discriminative Model:

Learn a probability distribution $p(y|x)$

Generative Model:
Learn a probability distribution $p(x)$

Conditional Generative Model: Learn $p(x|y)$



~~Conditional Generative Model: Each possible label induces a competition among all images~~

Justin Johnson

Lecture 19 - 23

November 20, 2019

的能力去拒绝不合理的需求，这样团队的
label下的程序
都很低

Discriminative vs Generative Models



Discriminative Model:

Learn a probability distribution $p(y|x)$

Generative Model:

Learn a probability distribution $p(x)$

Recall Bayes' Rule:

$$P(x | y) = \frac{P(y | x)}{P(y)} P(x)$$

Conditional
 Generative Model Discriminative Model
 (Unconditional)
 Generative Model

We can build a conditional generative model from other components!

Justin Johnson

Lecture 19 - 25

November 20, 2019

What can we do with a generative model?



Discriminative Model:

Learn a probability distribution $p(y|x)$

→ Assign labels to data

Feature learning (supervised)

Generative Model:

Learn a probability distribution $p(x)$

→ Detect outliers

拉近距离图

Feature learning (unsupervised)

Sample to **generate** new data

Conditional Generative Model:

Learn $p(x|y)$

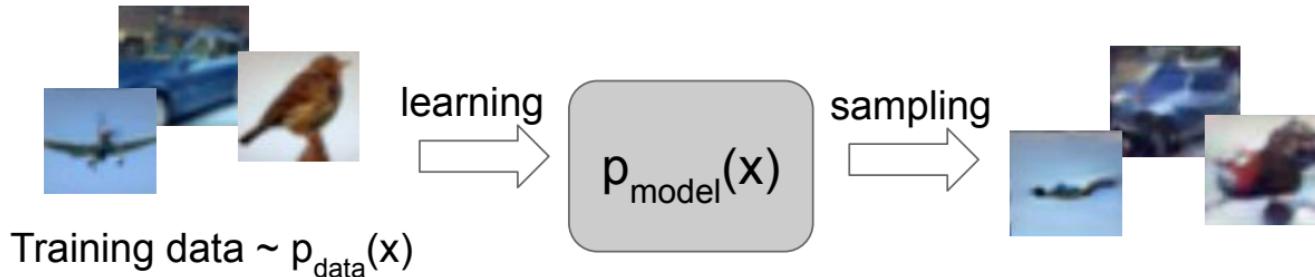
→ Assign labels, while rejecting outliers!

Generate new data conditioned on input labels

标注文字, 让输出更自然

Generative Modeling

Given training data, generate new samples from same distribution

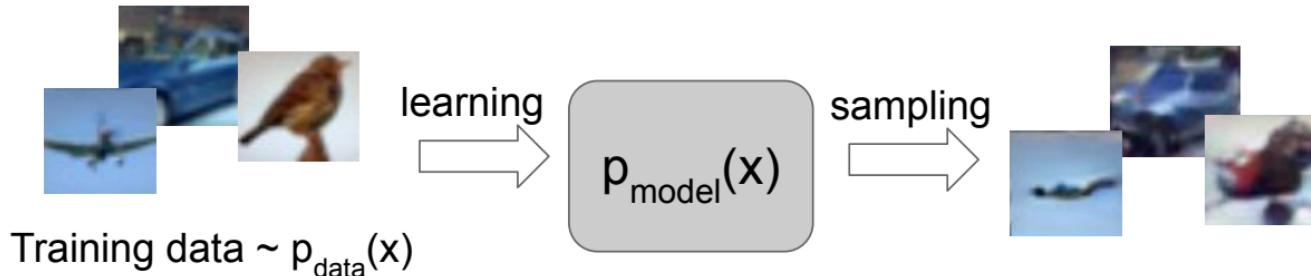


Objectives:

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

Generative Modeling

Given training data, generate new samples from same distribution



Formulate as density estimation problems:

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

Why Generative Models?



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

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

Taxonomy of Generative Models

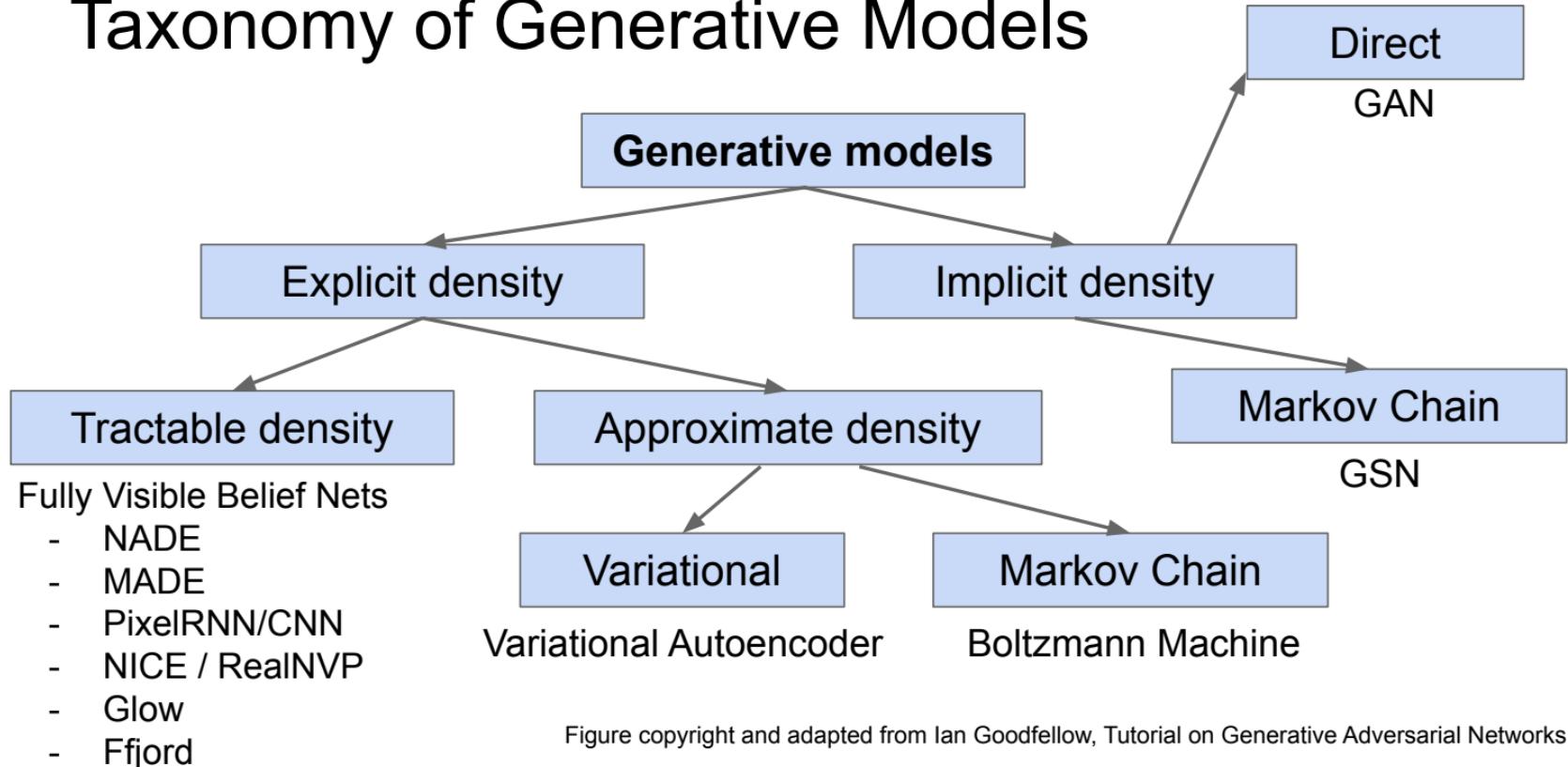


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

Today: discuss 3 most popular types of generative models today

Explicit density
Implicit density

Tractable density

Fully Visible Belief Nets

- NADE
- MADE
- PixelRNN/CNN
- NICE / RealNVP
- Glow
- Ffjord

Generative models

Implicit density

Markov Chain

Approximate density

Variational

Variational Autoencoder

Markov Chain

Boltzmann Machine

Direct

GAN

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

PixelRNN and PixelCNN

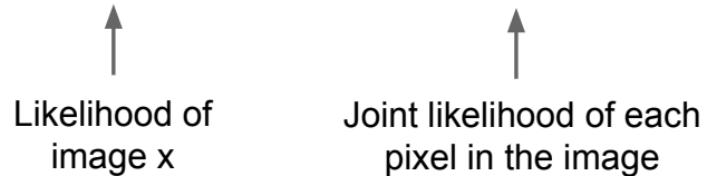
(A very brief overview)

Batch regressive mode |

Fully visible belief network (FVBN)

Explicit density model

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



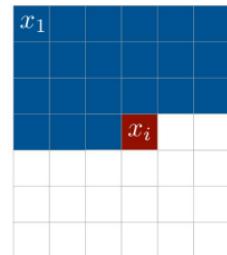
Fully visible belief network (FVBN)

Explicit density model

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

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

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



Then maximize likelihood of training data

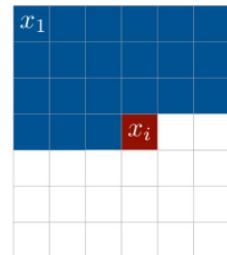
Fully visible belief network (FVBN)

Explicit density model

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

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

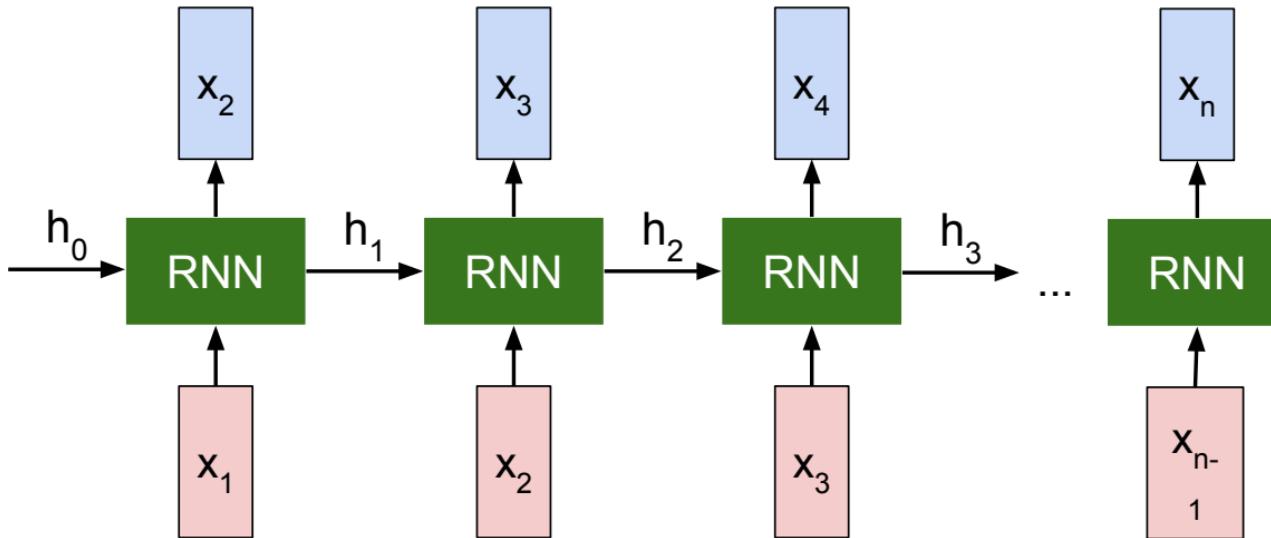
↑ ↑
Likelihood of image x Probability of i 'th pixel value
given all previous pixels



Then maximize likelihood of training data

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

Recurrent Neural Network

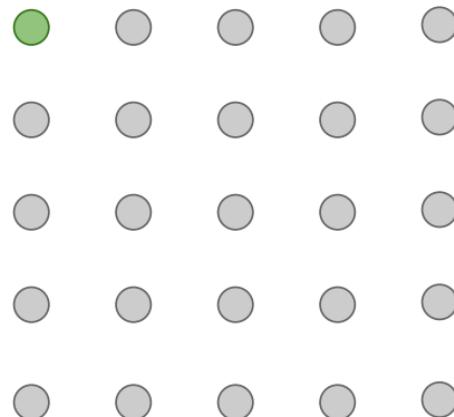


$$p(x_i|x_1, \dots, x_{i-1})$$

PixelRNN

[van der Oord et al. 2016]

Generate image pixels starting from corner



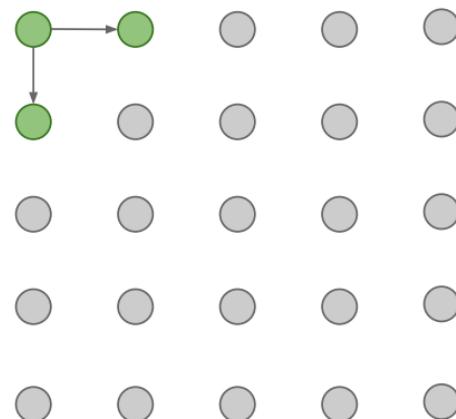
Dependency on previous pixels modeled
using an RNN (LSTM)

PixelRNN

[van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

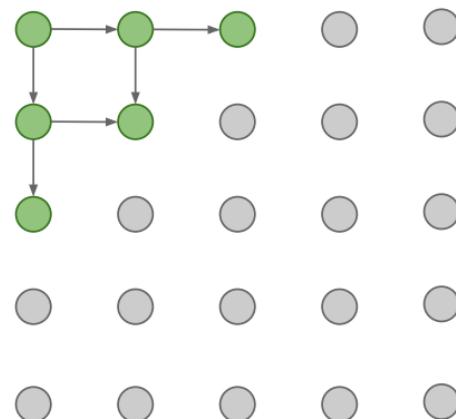


PixelRNN

[van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)



PixelRNN

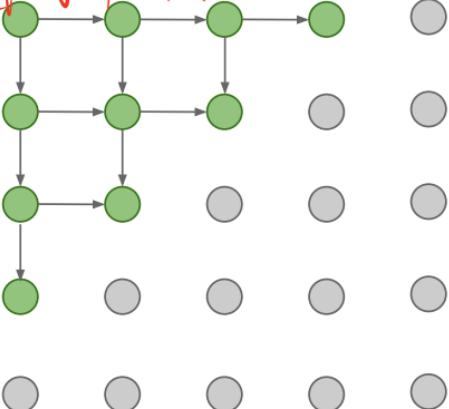
[van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

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

逐个生成像素，通过上一个的值
预测下一个像素的值。
从左下到右上。



逐个生成
需要 $2n - 1$ 步

PixelCNN

[van der Oord et al. 2016]

Still generate image pixels starting from corner

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

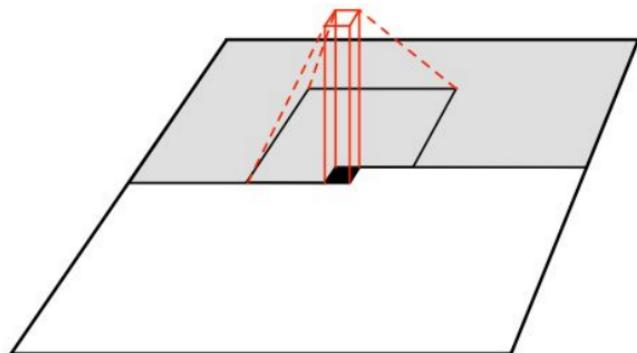


Figure copyright van der Oord et al., 2016. Reproduced with permission.

PixelCNN

[van der Oord et al. 2016]

Still generate image pixels starting from corner

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

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

Generation is still slow:

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

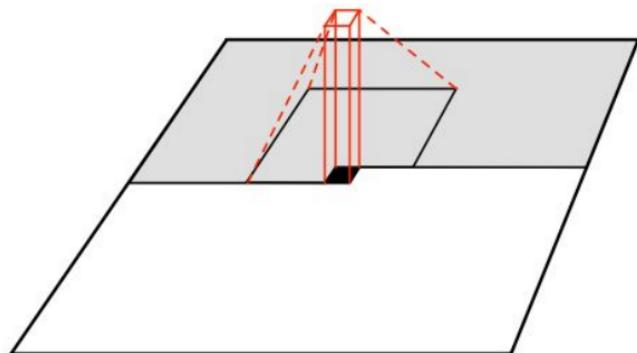
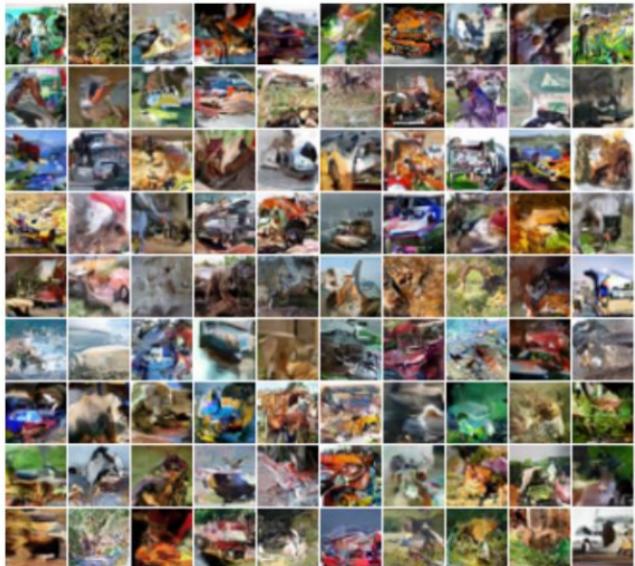
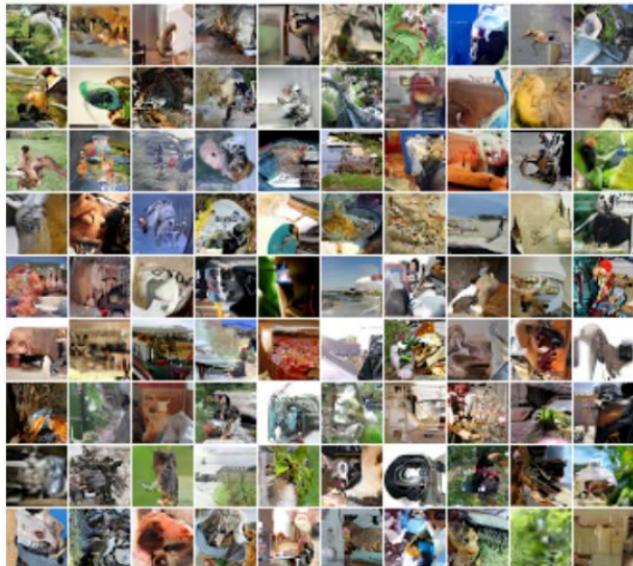


Figure copyright van der Oord et al., 2016. Reproduced with permission.

Generation Samples



32x32 CIFAR-10



32x32 ImageNet

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

PixelRNN and PixelCNN

Autoregressive Model
自回归模型

Pros:

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

Con:

- Sequential generation => slow

Improving PixelCNN performance

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

See

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

Taxonomy of Generative Models

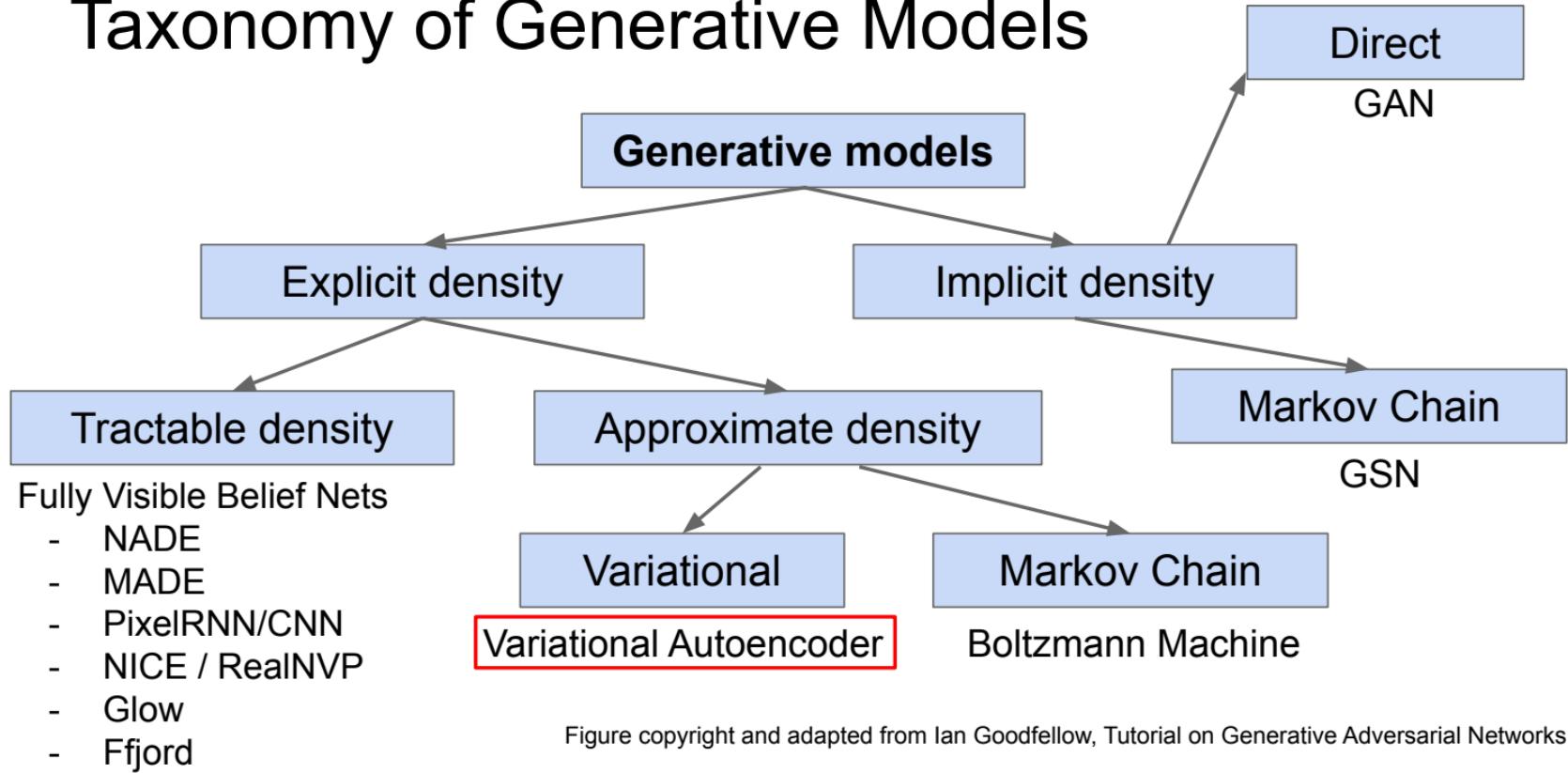


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Variational Autoencoders (VAE)

变分自动编码器

变分

So far...

易处理的

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

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

So far...

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

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

拜拜

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

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

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

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

So far...

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

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

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

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

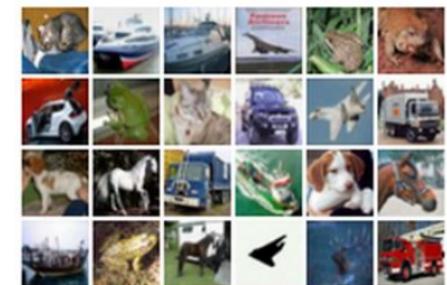
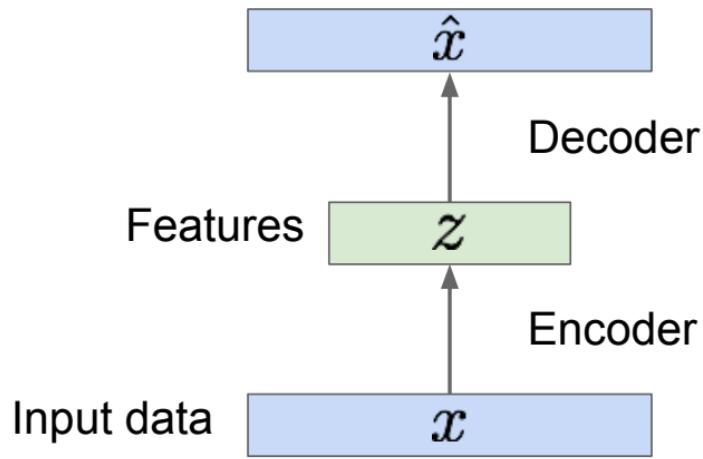
No dependencies among pixels, can generate all pixels at the same time!

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

Why latent \mathbf{z} ?

Some background first: Autoencoders

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

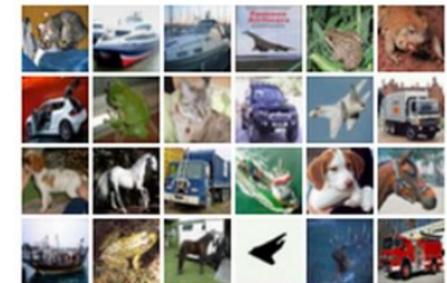
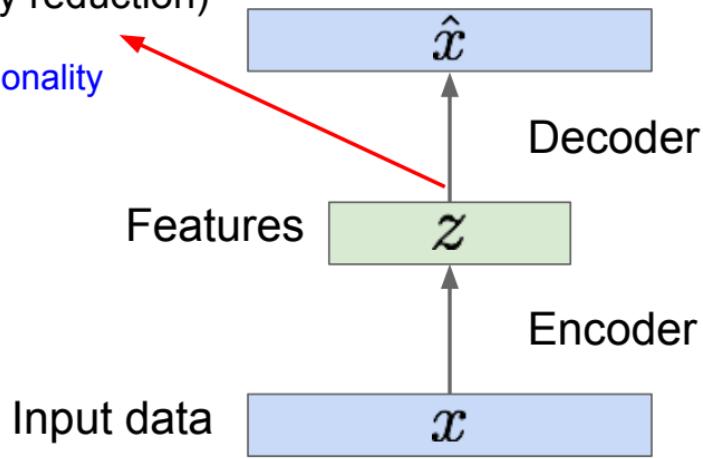


Some background first: Autoencoders

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

z usually smaller than x
(dimensionality reduction)

Q: Why dimensionality reduction?



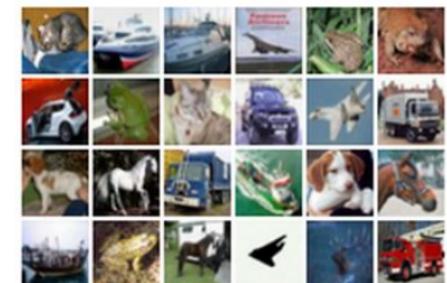
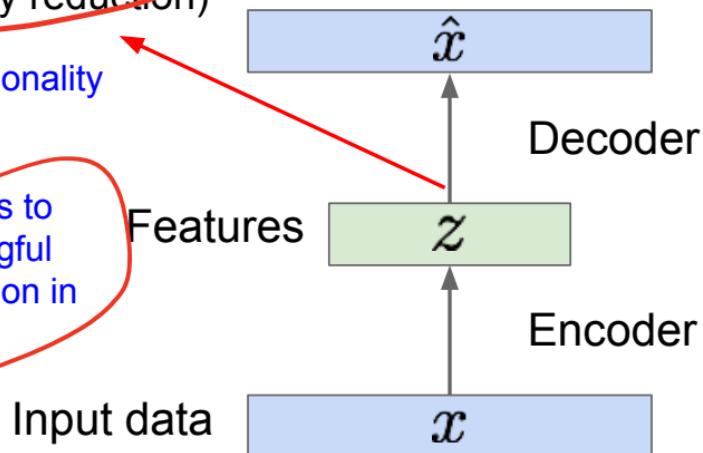
Some background first: Autoencoders

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

z usually smaller than x
(dimensionality reduction)

Q: Why dimensionality reduction?

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

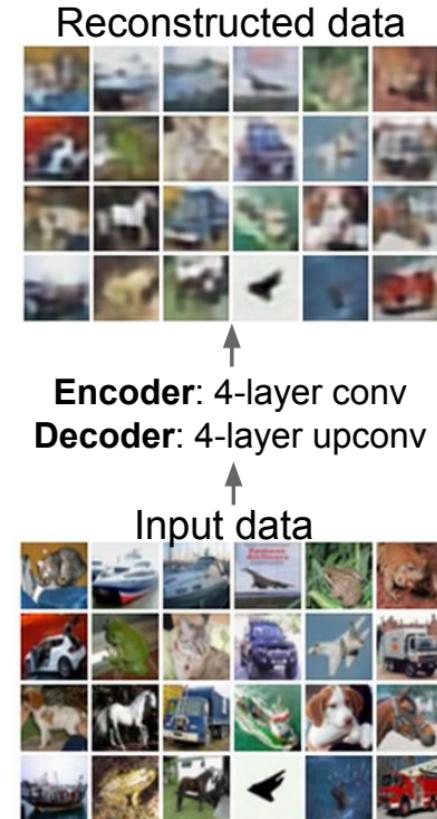
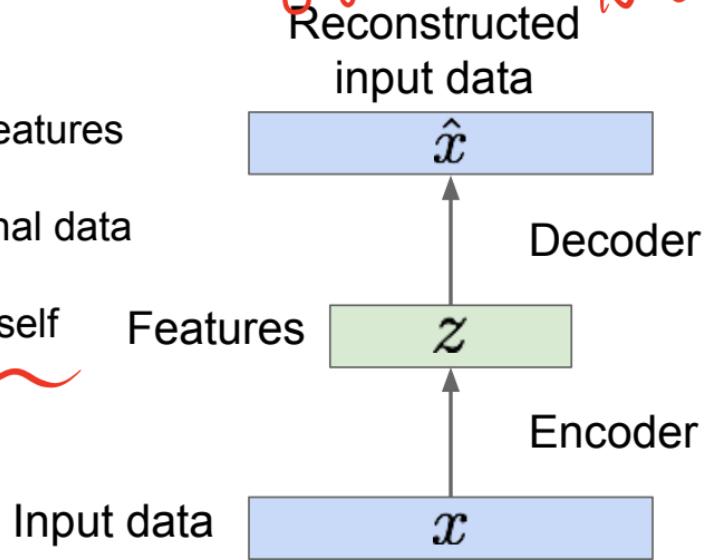


Some background first: Autoencoders

从语义分割
到语义，即用卷积和反卷积

How to learn this feature representation?

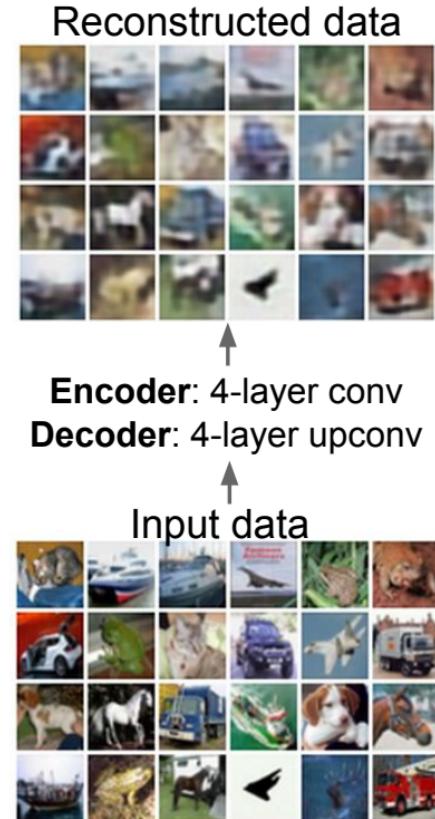
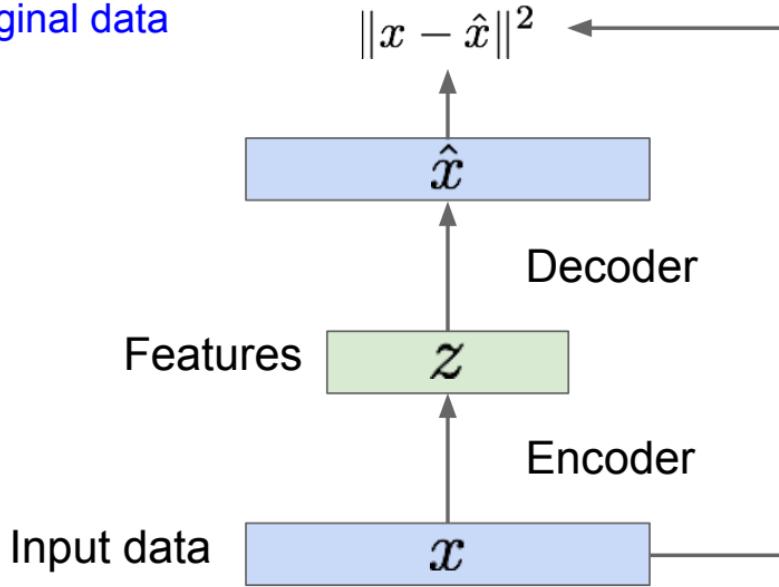
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding input itself



Some background first: Autoencoders

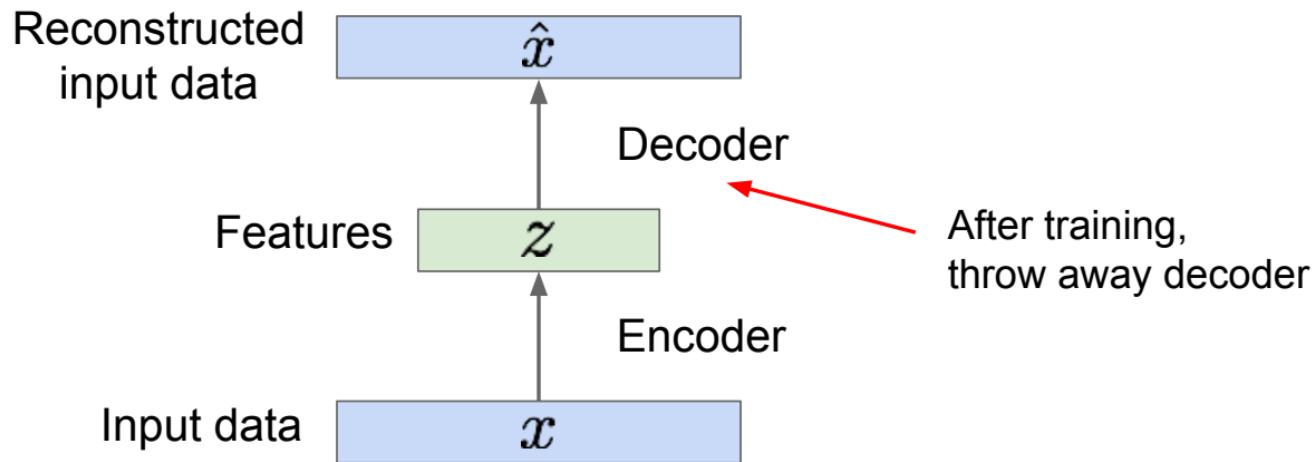
Train such that features
can be used to
reconstruct original data

L2 Loss function:
Doesn't use labels!



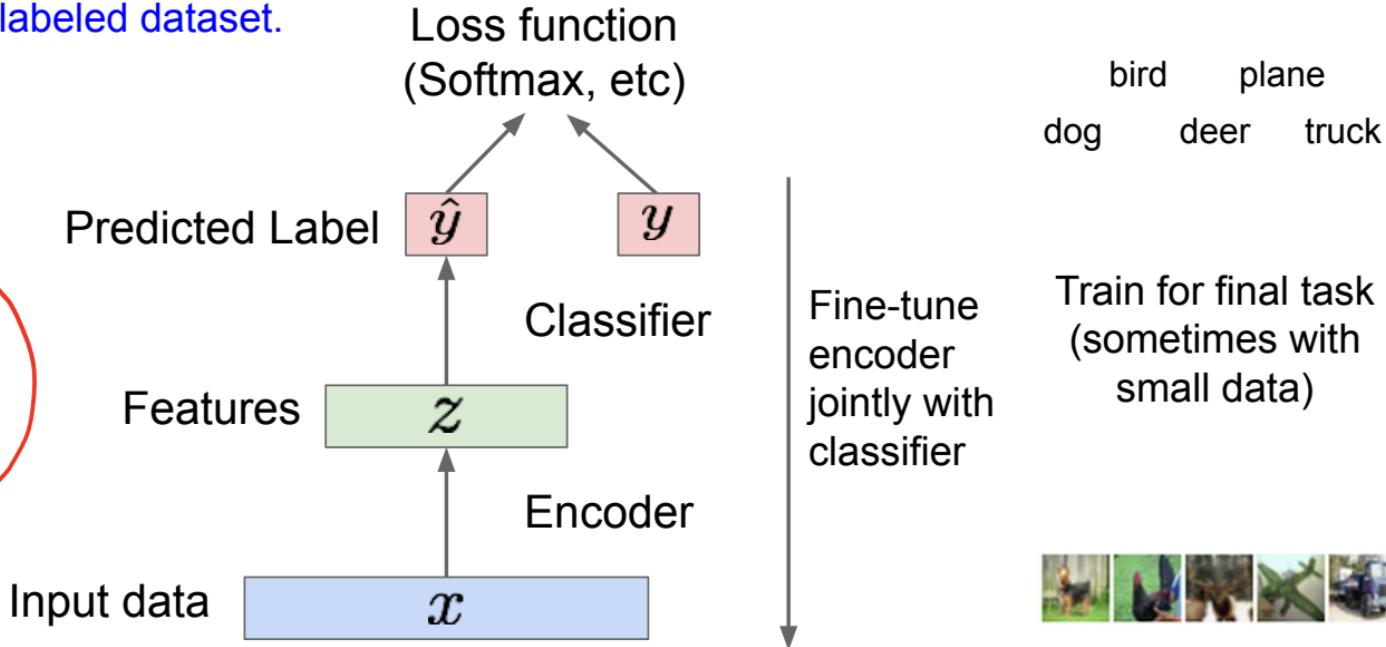
Some background first: Autoencoders

不直接 reconstruct (直接),
是通过降维得到 z .

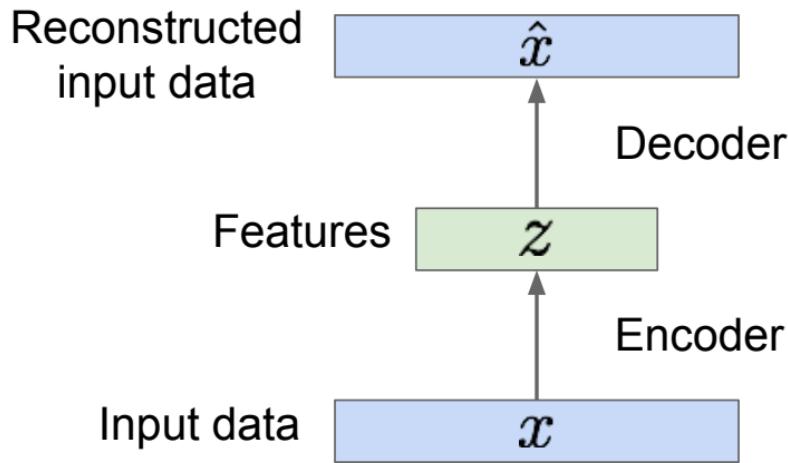


Some background first: Autoencoders

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



Some background first: Autoencoders



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

Features capture factors of variation in training data.

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

How do we make autoencoder a generative model?

Variational Autoencoders

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

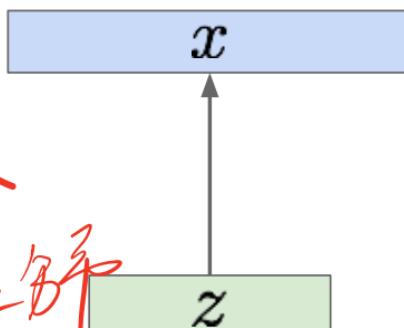
Variational Autoencoders

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

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

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

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



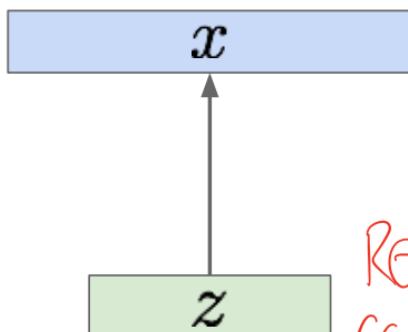
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders

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

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

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



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



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

ASSUME simple prior $P(z)$
Represent $P(x|z)$ with a neural network
(Similar to decoder from autoencoder)

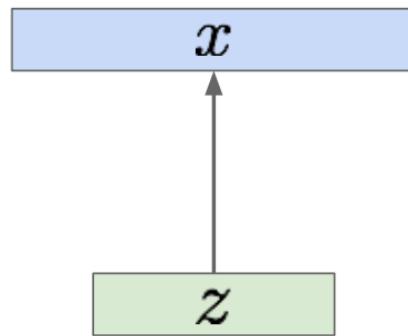
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders

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

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

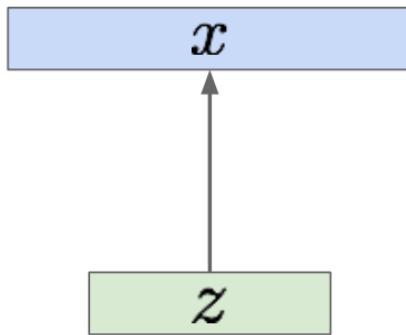
Sample from
true prior
 $z^{(i)} \sim p_{\theta^*}(z)$



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

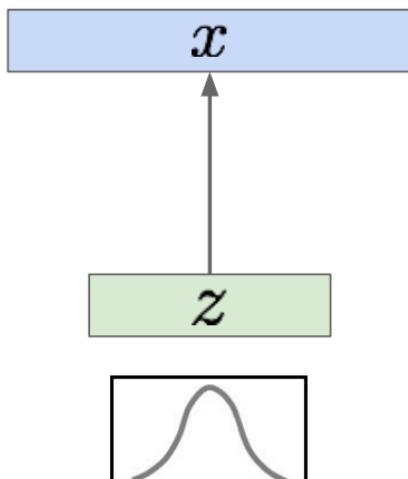
We want to estimate the true parameters θ^* of this generative model given training data x .

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

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

How should we represent this model?

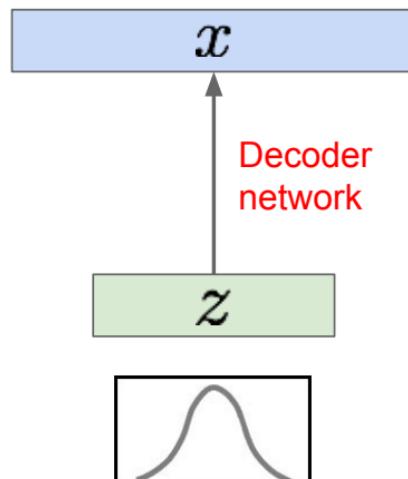
Choose prior $p(z)$ to be simple, e.g.
Gaussian. Reasonable for latent attributes,
e.g. pose, how much smile.

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

Variational Autoencoders



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



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

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

How should we represent this model?

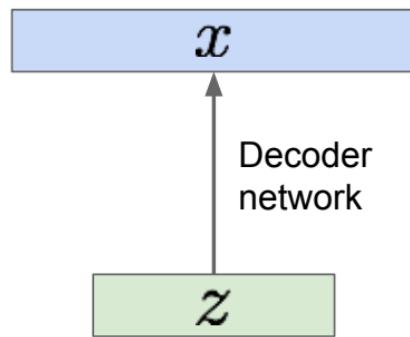
Choose prior $p(z)$ to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.

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

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

Variational Autoencoders

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



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

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

How to train the model?

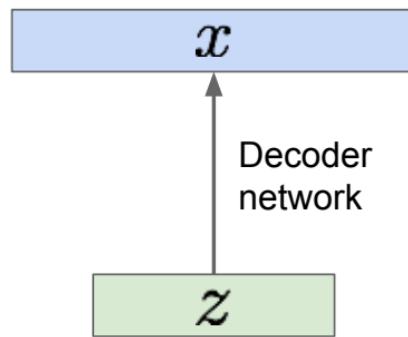
Variational Autoencoders

Sample from
true conditional

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

Sample from
true prior

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



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

How to train the model?

Learn model parameters to maximize likelihood
of training data

BKLKBSJF

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

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

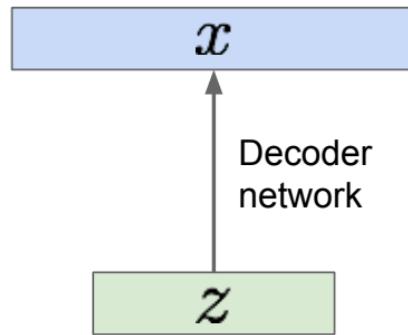
Variational Autoencoders

Sample from
true conditional

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

Sample from
true prior

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



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

How to train the model?

Learn model parameters to maximize likelihood of training data

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

Q: What is the problem with this?

Intractable!

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

Variational Autoencoders: Intractability

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

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

Variational Autoencoders: Intractability

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

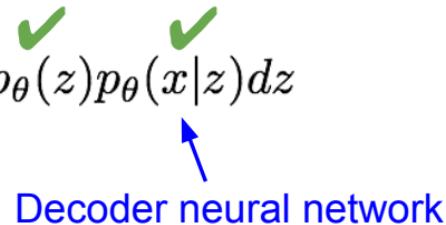
Simple Gaussian prior



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

Variational Autoencoders: Intractability

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



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

Variational Autoencoders: Intractability

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



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

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

Variational Autoencoders: Intractability

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



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

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

Monte Carlo estimation is too high variance

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

Variational Autoencoders: Intractability

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

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

Posterior:

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

Intractable data likelihood

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

Variational Autoencoders: Intractability

输入 X , 找 $p_{\theta}(x)$
解

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

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

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

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

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

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

Variational Autoencoders

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

Variational Autoencoders

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



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

Variational Autoencoders

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

Not true

Variational Autoencoders

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

Variational Autoencoders

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

Variational Autoencoders

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

Wrt. 經子編-陳

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

Variational Autoencoders

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



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



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



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

Variational Autoencoders

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

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

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

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

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

We want to
maximize the
data
likelihood

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

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

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

Variational Autoencoders

We want to
maximize the
data
likelihood

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

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

Variational Autoencoders

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

Decoder:
reconstruct
the input data

Encoder:
make approximate
posterior distribution
close to prior

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

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

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

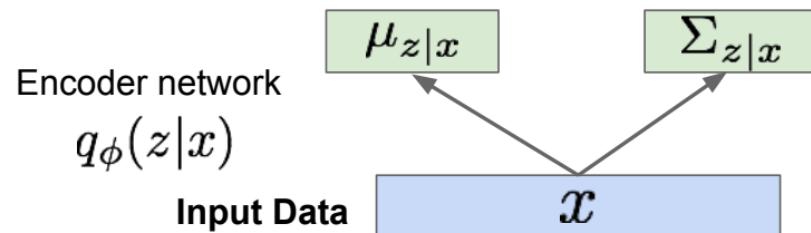
Input Data

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

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

Have analytical solution

Encoder network

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

Input Data

$$\mu_{z|x}$$

$$\Sigma_{z|x}$$

对称的协方差矩阵
只用上半部分

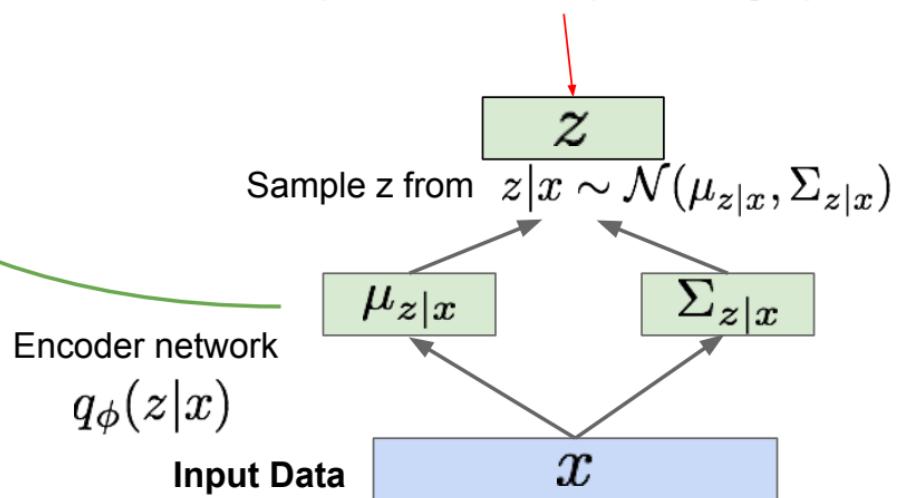
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

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

Make approximate posterior distribution close to prior



Variational Autoencoders

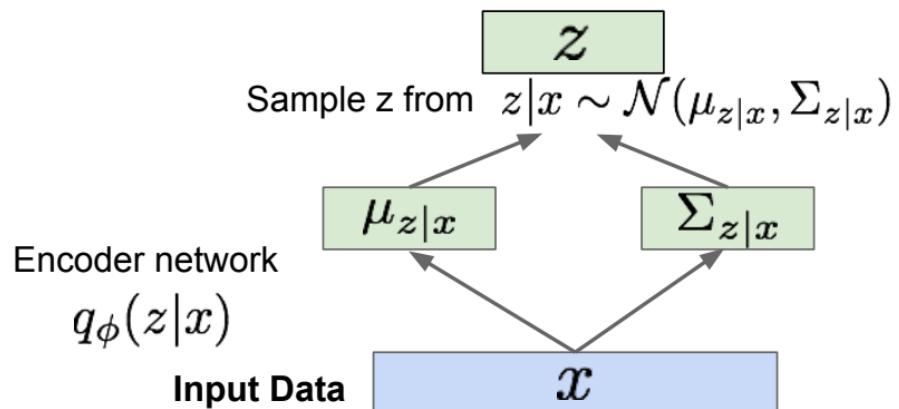
Putting it all together: maximizing the likelihood lower bound

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

Reparameterization trick to make sampling differentiable:

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

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



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

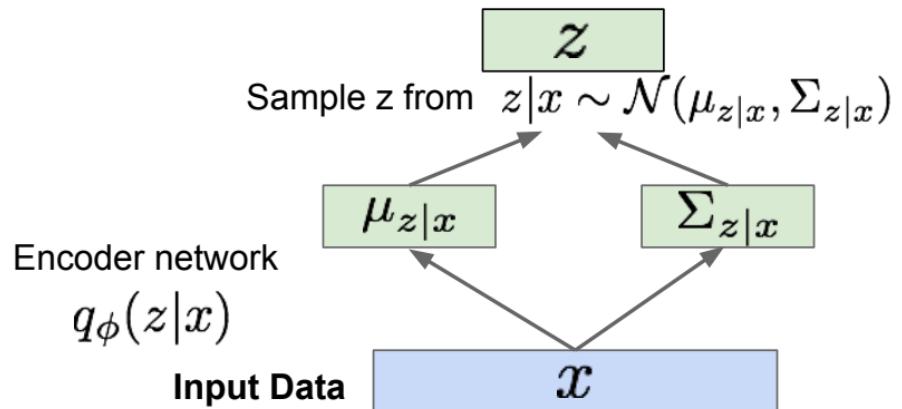
$$\underbrace{\mathbf{E}_z \left[\log p_\theta(x^{(i)} | z) \right]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))$$

Reparameterization trick to make sampling differentiable:

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

Part of computation graph

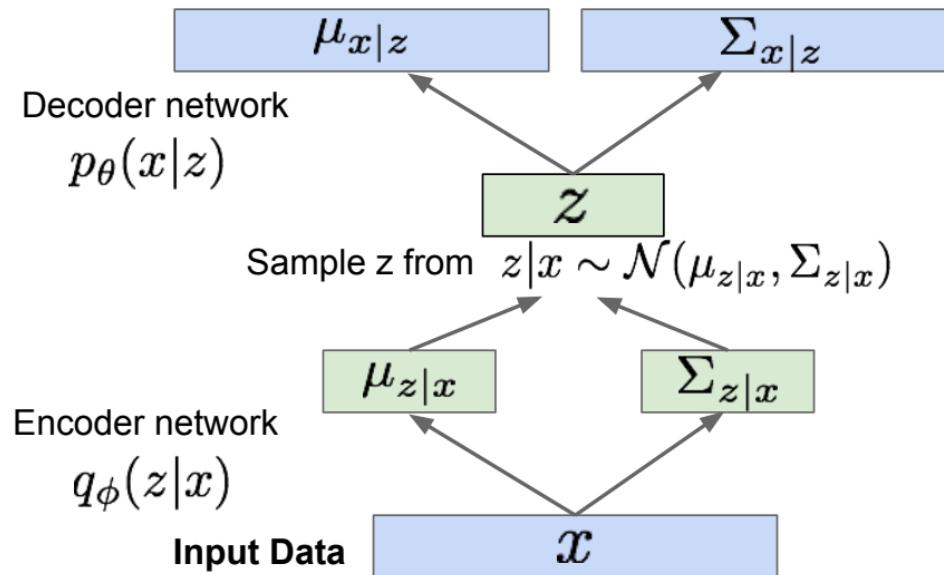
Input to the graph



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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



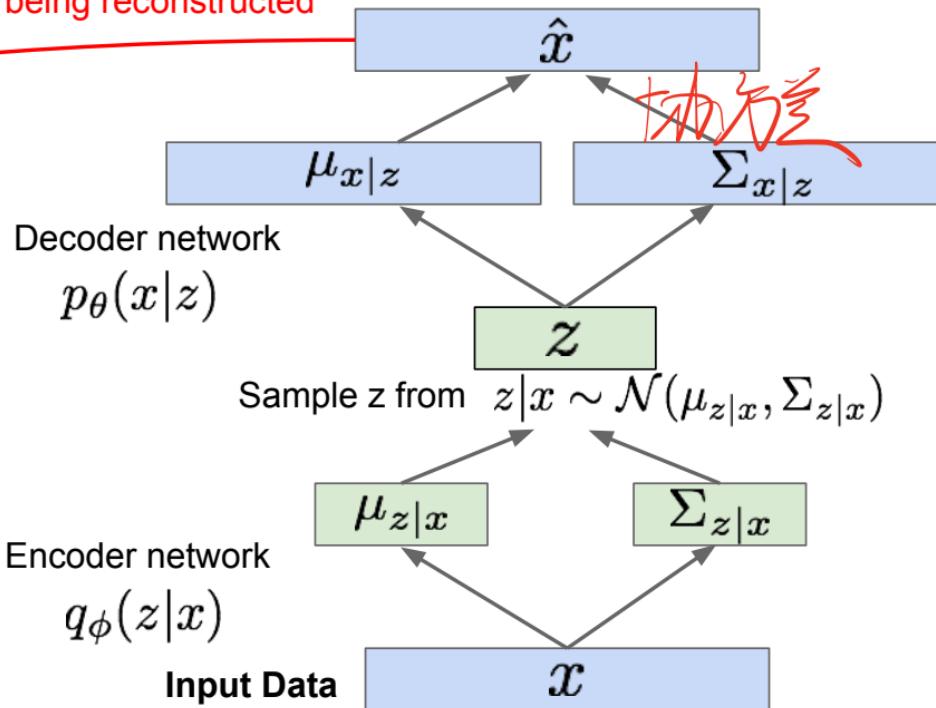
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

$\underbrace{\quad}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$

Maximize likelihood of original input being reconstructed



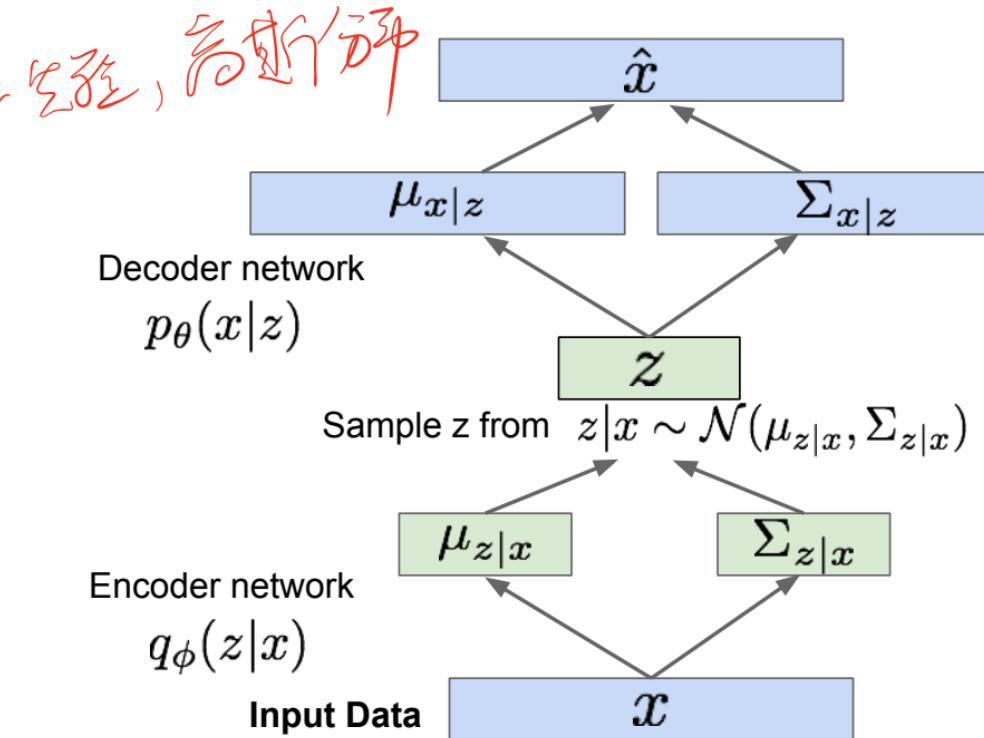
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

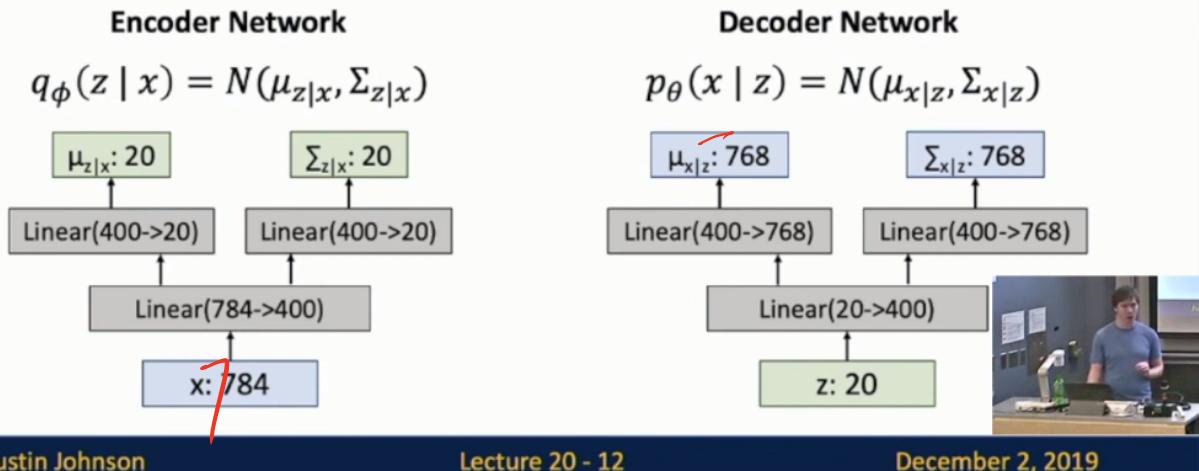
$\mathcal{L}(x^{(i)}, \theta, \phi)$

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



Example: Fully-Connected VAE

x : 28x28 image, flattened to 784-dim vector
 z : 20-dim vector



Variational Autoencoders

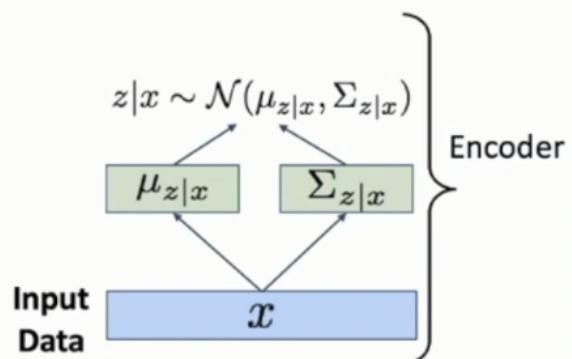
Train by maximizing the
variational lower bound



Closed form solution when
 q_ϕ is diagonal Gaussian and
 p is unit Gaussian!
(Assume z has dimension J)

1. Run input data through **encoder** to get a distribution over latent codes
2. **Encoder output should match the prior $p(z)$!**

$$\begin{aligned} -D_{KL}(q_\phi(z|x), p(z)) &= \int_z q_\phi(z|x) \log \frac{p(z)}{q_\phi(z|x)} dz \\ &= \int_z N(z; \mu_{z|x}, \Sigma_{z|x}) \log \frac{N(z; 0, I)}{N(z; \mu_{z|x}, \Sigma_{z|x})} dz \\ &= \sum_{j=1}^J \left(1 + \log \left((\Sigma_{z|x})_j^2 \right) - (\mu_{z|x})_j^2 - (\Sigma_{z|x})_j^2 \right) \end{aligned}$$



Justin Johnson

Lecture 20 - 16

December 2, 2019

Latent variable 潛在變數

長短句變換

→ 对于进行的及
它应该多了解
该函数的特性和

Variational Autoencoders

Train by maximizing the
variational lower bound

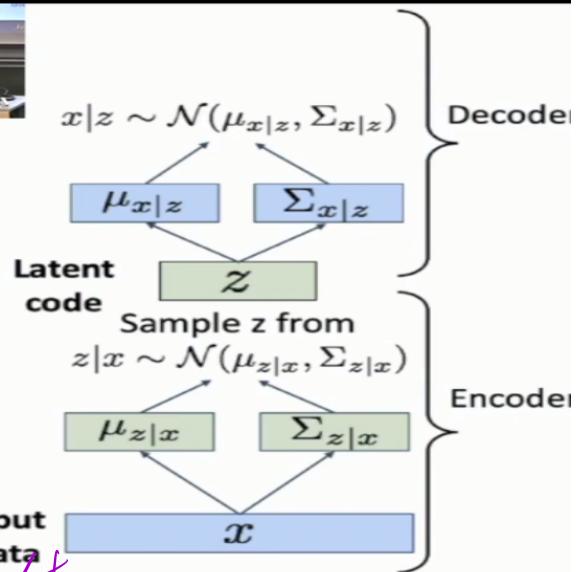
$$E_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z))$$

1. Run input data through **encoder** to get a distribution over latent codes
2. **Encoder output should match the prior $p(z)$!**
3. Sample code z from encoder output
4. Run sampled code through **decoder** to get a distribution over data samples
5. **Original input data should be likely under the distribution output from (4)!**
6. Can sample a reconstruction from 4.

Justin Johnson

Lecture 20 - 19

December 2, 2019



PRB 8, 从 z 的分布，运行 z 后进入 Decoder.
再通过 Decoder 产生 PRB

Variational Autoencoders: Generating Data!

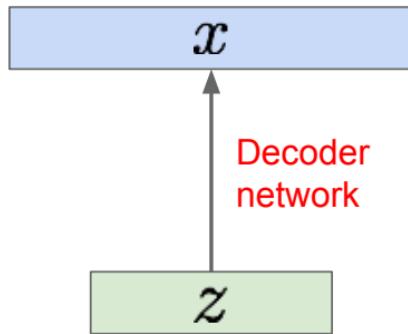
Our assumption about data generation process

Sample from true conditional

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

Sample from true prior

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



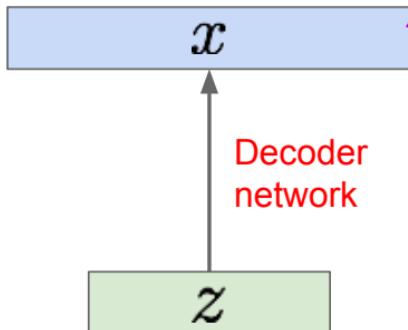
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders: Generating Data!

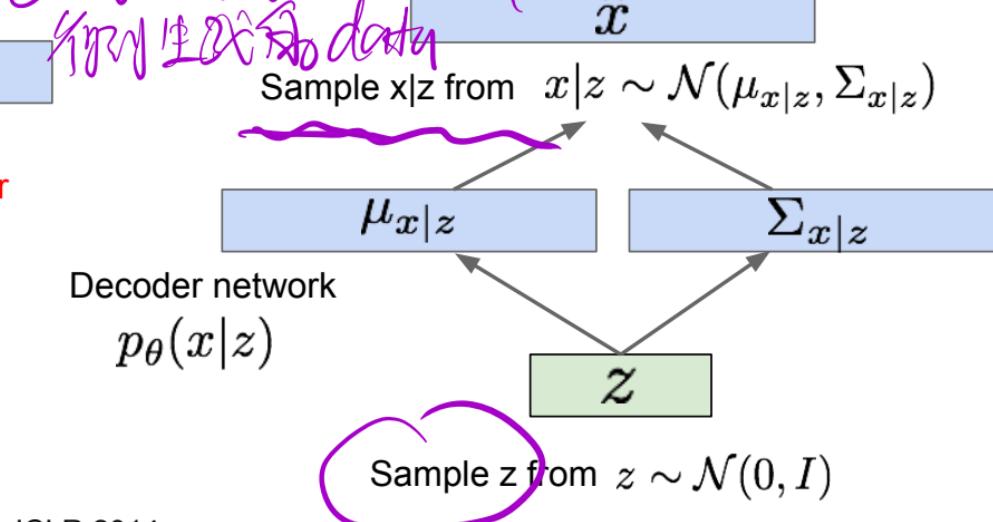
Our assumption about data generation process

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

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



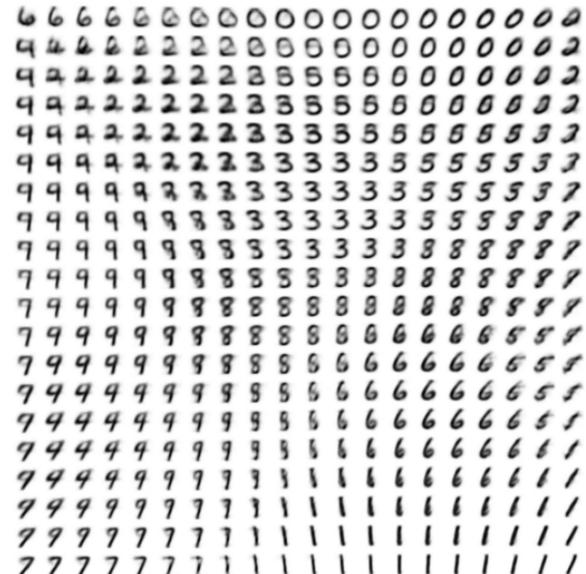
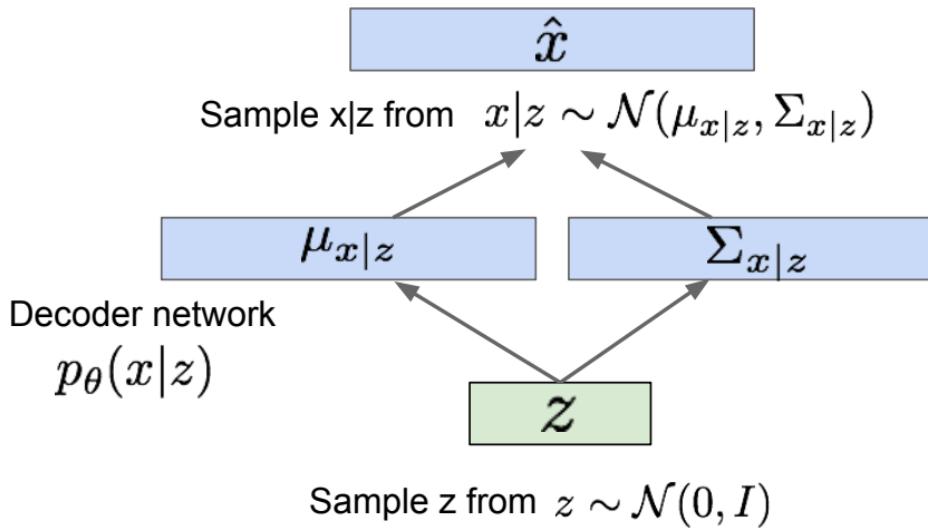
Now given a trained VAE:
use decoder network & sample z from prior!



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

Variational Autoencoders: Generating Data!

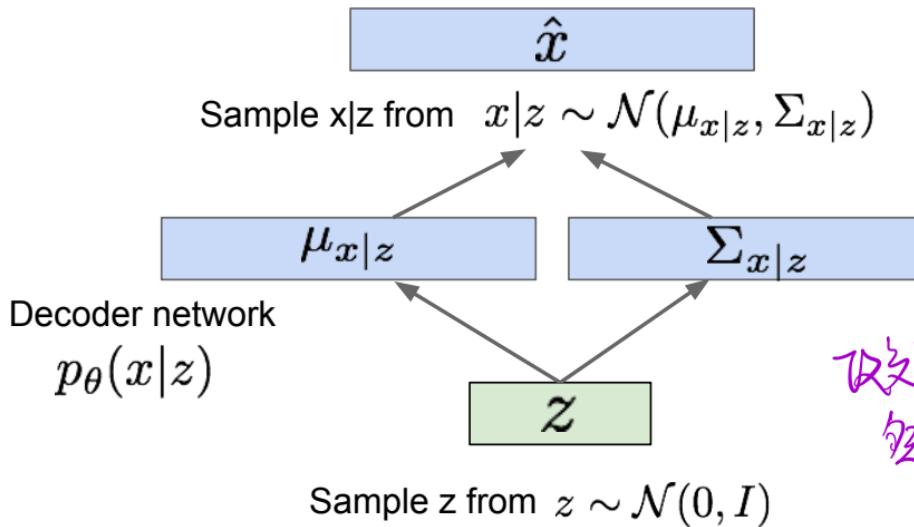
Use decoder network. Now sample z from prior!



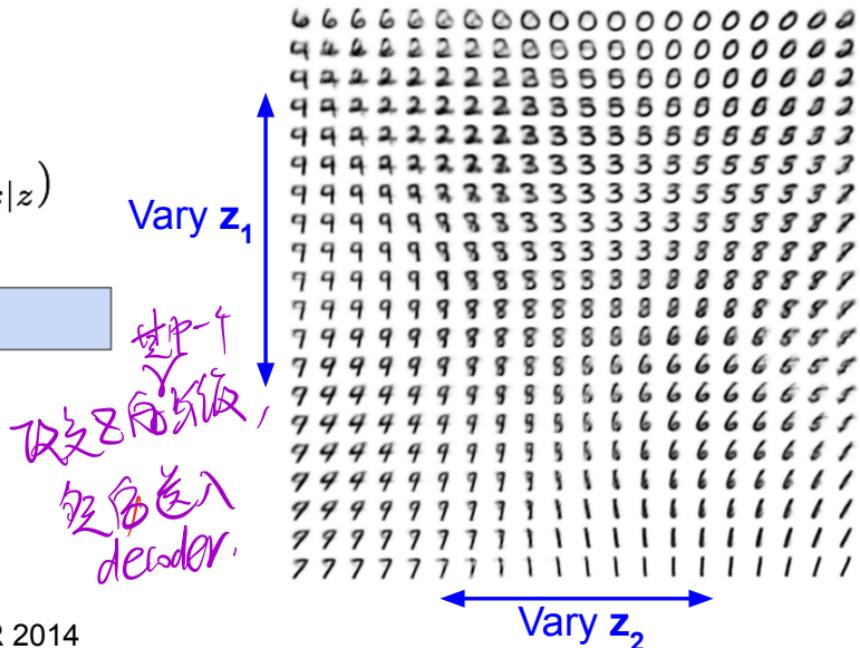
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!



Data manifold for 2-d z



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

Variational Autoencoders

After training we can **edit images**

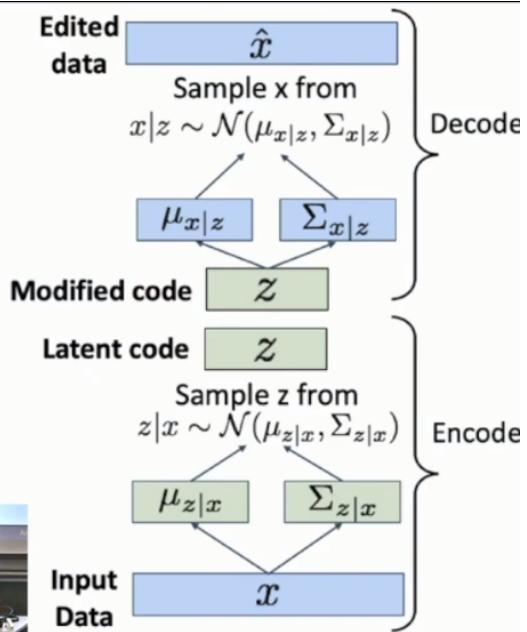
1. Run input data through **encoder** to get a distribution over latent codes
2. Sample code z from encoder output
3. Modify some dimensions of sampled code
4. Run modified z through **decoder** to get a distribution over data samples
5. Sample new data from (4)



Justin Johnson

Lecture 20 - 30

December 2, 2019



Variational Autoencoders: Generating Data!

Z服从正态分布,

Diagonal prior on z

\Rightarrow independent
latent variables

Different
dimensions of z
encode
interpretable factors
of variation

多样性
多样性

Degree of smile

Vary z_1



Vary z_2 Head pose

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

Variational Autoencoders: Generating Data!

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

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

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

Degree of smile

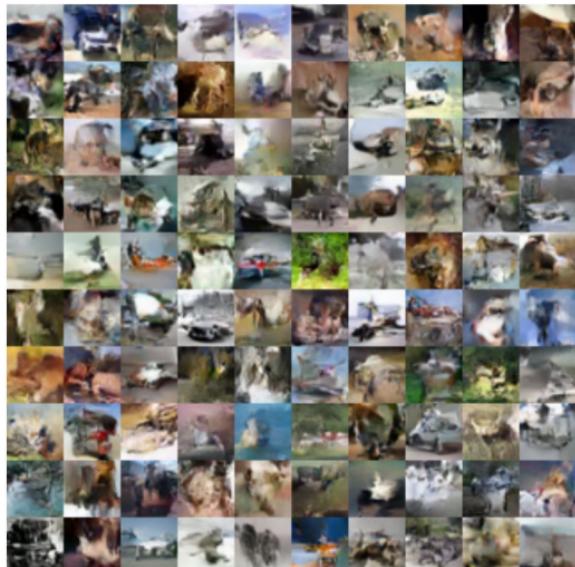
Vary z_1



Vary z_2 Head pose

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

Variational Autoencoders: Generating Data!



32x32 CIFAR-10



Labeled Faces in the Wild

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

Variational Autoencoders

(变分自编码器)

Probabilistic spin to traditional autoencoders => allows generating data

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

Pros:

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

Cons:

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

Active areas of research:

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

Taxonomy of Generative Models

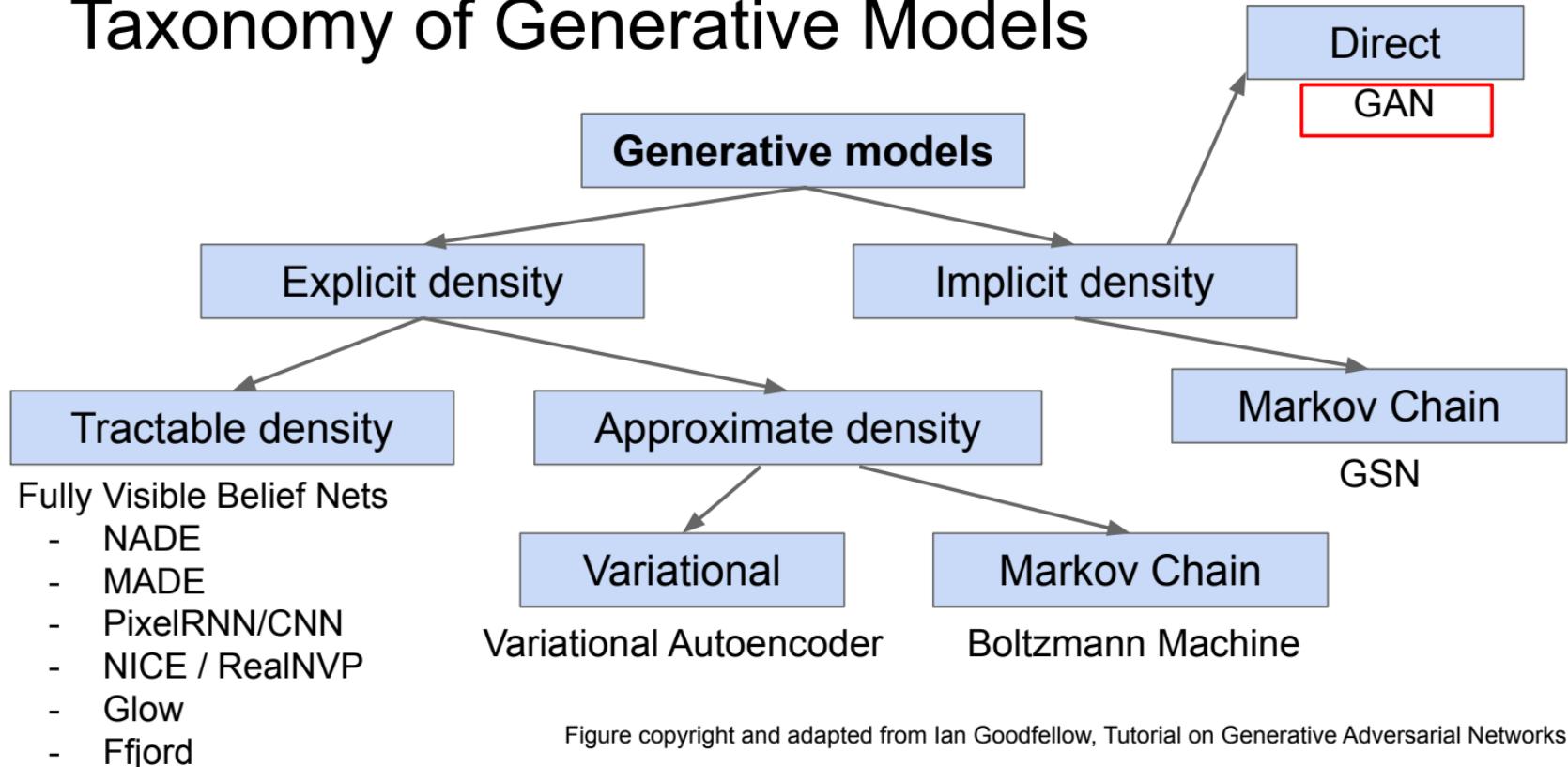


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Generative Adversarial Networks (GANs)

So far...

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

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

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

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

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

So far...

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

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

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

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

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

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

So far...

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

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

→ 太慢
→ 快，但是生成图像很模糊 -

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

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

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

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

GANs: not modeling any explicit density function!

Generative Adversarial Networks

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

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

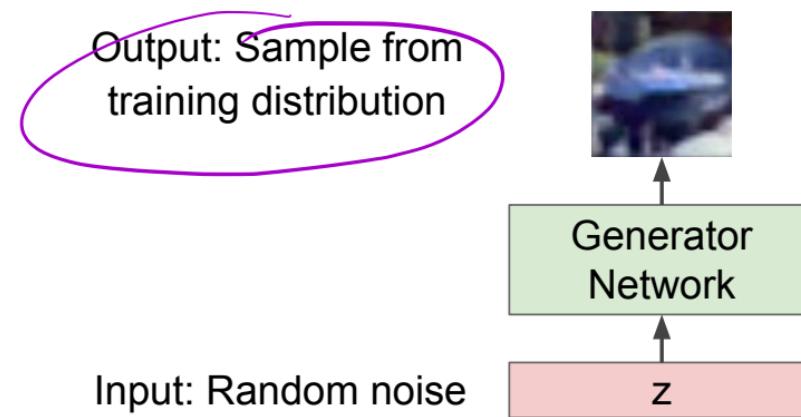
Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

Generative Adversarial Networks

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

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

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



Generative Adversarial Networks

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

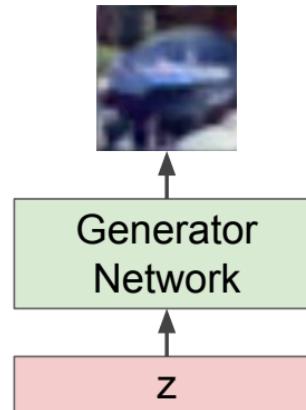
Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

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

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

Output: Sample from training distribution

Input: Random noise



Generative Adversarial Networks

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

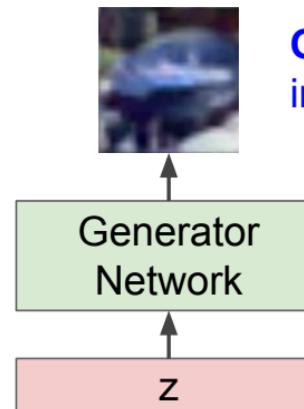
Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

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

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

Output: Sample from training distribution

Input: Random noise



Objective: generated images should look "real"



$\mathcal{P}_G = \mathcal{P}_{\text{data}}$

Generative Adversarial Networks

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

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

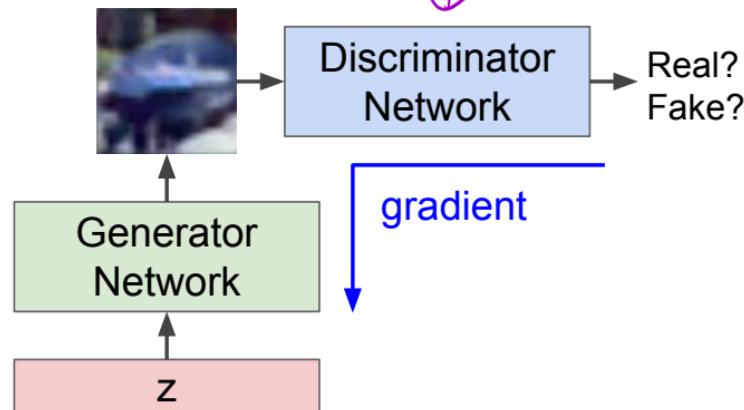
Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

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

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

Output: Sample from training distribution

Input: Random noise



Training GANs: Two-player game

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

Discriminator network: try to distinguish between real and fake images

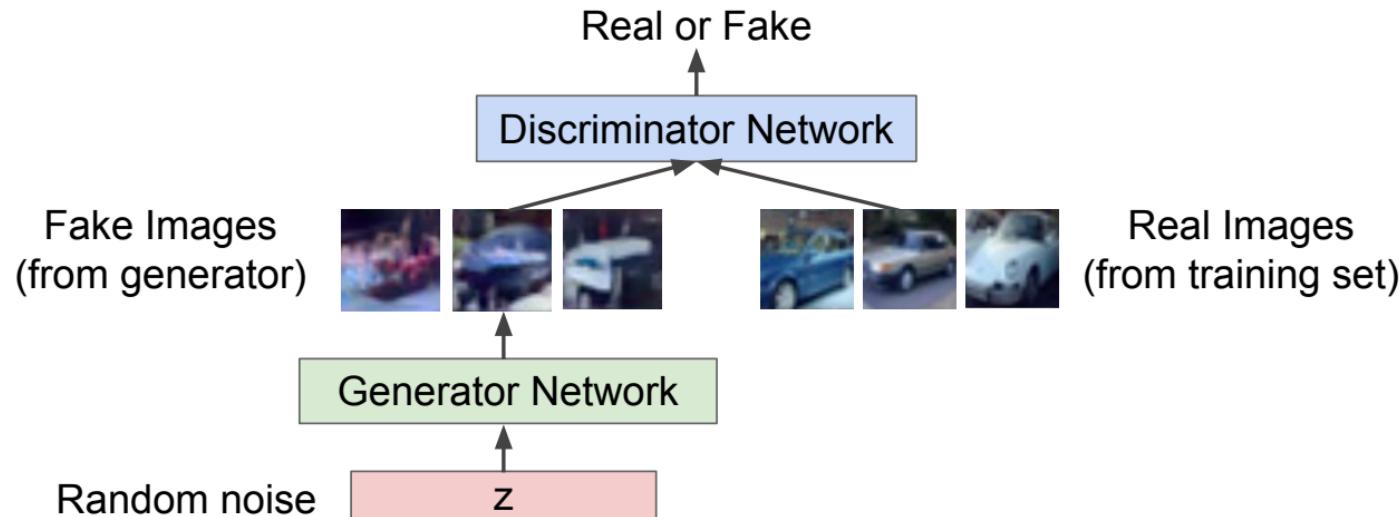
Generator network: try to fool the discriminator by generating real-looking images

Training GANs: Two-player game

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

Discriminator network: try to distinguish between real and fake images

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



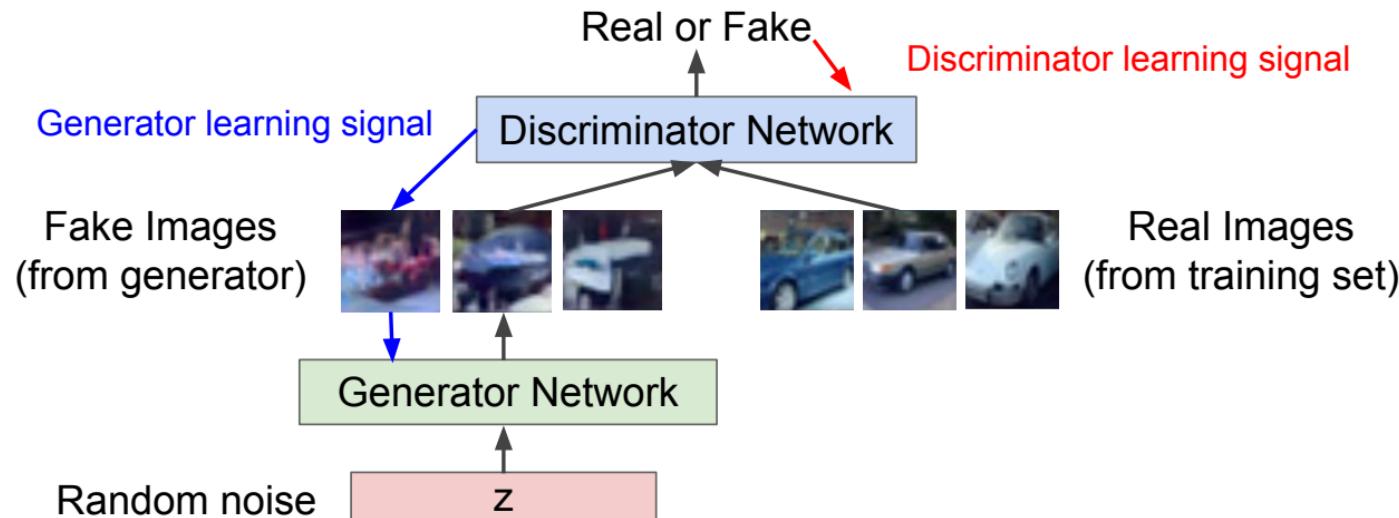
Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

Training GANs: Two-player game

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

Discriminator network: try to distinguish between real and fake images

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



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

Training GANs: Two-player game

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

Discriminator network: try to distinguish between real and fake images

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

Train jointly in **minimax game**

Minimax objective function:

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

Generator objective
Discriminator objective

Training GANs: Two-player game

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

Discriminator network: try to distinguish between real and fake images

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

Train jointly in **minimax game**

Minimax objective function:

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

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

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

Discriminator output for real data x

\uparrow

\uparrow

\uparrow

$x: \text{real data}$

$G(z): \text{fake data}$

discriminator

让假的数据出错

Dod 假的错

Gog 好的对

Training GANs: Two-player game

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

Discriminator network: try to distinguish between real and fake images

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

Train jointly in **minimax game**

Minimax objective function:

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

Training GANs: Two-player game

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

Discriminator network: try to distinguish between real and fake images

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

Train jointly in **minimax game**

Minimax objective function:

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

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

Training GANs: Two-player game

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

Discriminator network: try to distinguish between real and fake images

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

Train jointly in **minimax game**

Minimax objective function:

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

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

- 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

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



Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Train G and D using alternating gradient updates

$$\begin{aligned} & \min_{\mathbf{G}} \max_{\mathbf{D}} \left(E_{x \sim p_{data}} [\log \mathbf{D}(x)] + E_{\mathbf{z} \sim p(\mathbf{z})} \left[\log \left(1 - \mathbf{D}(\mathbf{G}(\mathbf{z})) \right) \right] \right) \\ &= \min_{\mathbf{G}} \max_{\mathbf{D}} \mathbf{V}(\mathbf{G}, \mathbf{D}) \quad \text{For t in 1, ... T:} \\ &\quad 1. \text{ (Update } \mathbf{D} \text{)} \quad \mathbf{D} = \mathbf{D} + \alpha_{\mathbf{D}} \frac{\partial \mathbf{V}}{\partial \mathbf{D}} \\ &\quad 2. \text{ (Update } \mathbf{G} \text{)} \quad \mathbf{G} = \mathbf{G} - \alpha_{\mathbf{G}} \frac{\partial \mathbf{V}}{\partial \mathbf{G}} \end{aligned}$$

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



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

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

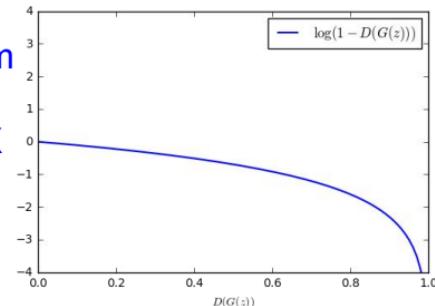
2. Gradient descent on generator

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

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

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

不能只优化 loss 来获得有效的训练
一个必须看，另一个必须看



Training GANs: Two-player game

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

Minimax objective function:

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

Generator 亂序
反梯度法

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

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

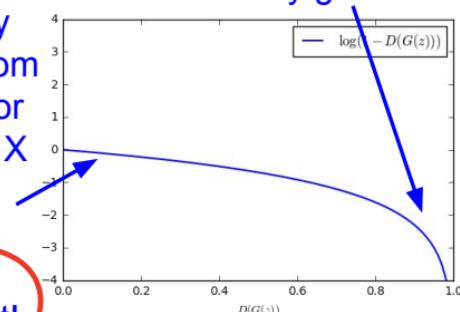
In practice, optimizing this generator objective does not work well!

梯度上升法

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

But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good



Training GANs: Two-player game

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

Minimax objective function:

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

Alternate between:

1. Gradient ascent on discriminator

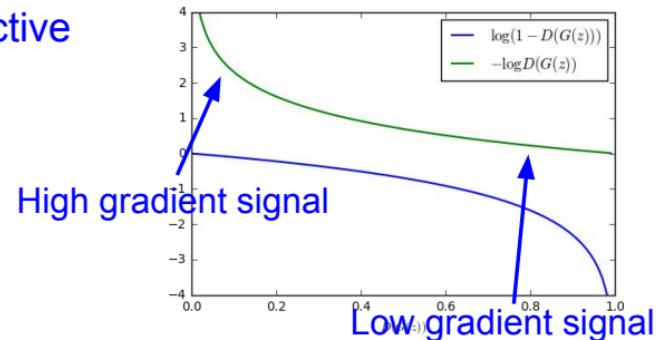
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. Instead: Gradient ascent on generator, different objective

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

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

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



Generative Adversarial Networks: Optimality



$$\min_G \max_D \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} [\log (1 - D(G(z)))] \right)$$

$$= \min_G \max_D (E_{x \sim p_{data}} [\log D(x)] + E_{x \sim p_G} [\log (1 - D(x))])$$

$$= \min_G \int_X \max_D (p_{data}(x) \log D(x) + p_G(x) \log (1 - D(x))) dx$$

$$f(y) = a \log y + b \log(1 - y) \quad f'(y) = 0 \Leftrightarrow y = \frac{a}{a+b} \text{ (local max)}$$

$$f'(y) = \frac{a}{y} - \frac{b}{1-y} \quad \text{Optimal Discriminator: } D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

Justin Johnson

Lecture 20 - 69

December 2, 2019

Generative Adversarial Networks: Optimality



$$\min_G \max_D \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} [\log (1 - D(G(z)))] \right)$$

$$= \min_G \left(E_{x \sim p_{data}} \left[\log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[\log \frac{2 * p_G(x)}{p_{data}(x) + p_G(x)} \right] - \log 4 \right)$$

$$= \min_G \left(KL \left(p_{data}, \frac{p_{data} + p_G}{2} \right) + KL \left(p_G, \frac{p_{data} + p_G}{2} \right) - \log 4 \right)$$

$$= \min_G (2 * JSD(p_{data}, p_G) - \log 4)$$

JSD is always nonnegative, and zero only when the two distributions are equal!

Thus $p_{data} = p_G$ is the global min, QED

Jensen-Shannon Divergence:

$$JSD(p, q) = \frac{1}{2} KL \left(p, \frac{p+q}{2} \right) + \frac{1}{2} KL \left(q, \frac{p+q}{2} \right)$$

Justin Johnson

Lecture 20 - 85

December 2, 2019

Generative Adversarial Networks: Optimality



$$\min_{\textcolor{brown}{G}} \max_{\textcolor{blue}{D}} \left(E_{x \sim p_{data}} [\log \textcolor{blue}{D}(x)] + E_{\textcolor{violet}{z} \sim p(\textcolor{violet}{z})} \left[\log \left(1 - \textcolor{blue}{D}(\textcolor{brown}{G}(\textcolor{violet}{z})) \right) \right] \right)$$

$$= \min_{\textcolor{brown}{G}} (2 * JSD(p_{data}, p_{\textcolor{brown}{G}}) - \log 4)$$

Summary: The global minimum of the minimax game happens when:

- $D_G^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}$ (Optimal discriminator for any G)
- $p_G(x) = p_{\text{data}}(x)$ (Optimal generator for optimal D)

Caveats:

1. G and D are neural nets with fixed architecture. We don't know whether they can actually represent the optimal D and G.
 2. This tells us nothing about convergence to the optimal solution

Justin Johnson

Lecture 20 - 88

December 2, 2019

↓
不建议
要领的
是否在
能辨别
和空间

Training GANs: Two-player game

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

Putting it together: GAN training algorithm

for number of training iterations **do**

for k steps **do**

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

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

end for

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

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

end for

Training GANs: Two-player game

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

Putting it together: GAN training algorithm

for number of training iterations **do**

for k steps **do**

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

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

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.

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

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

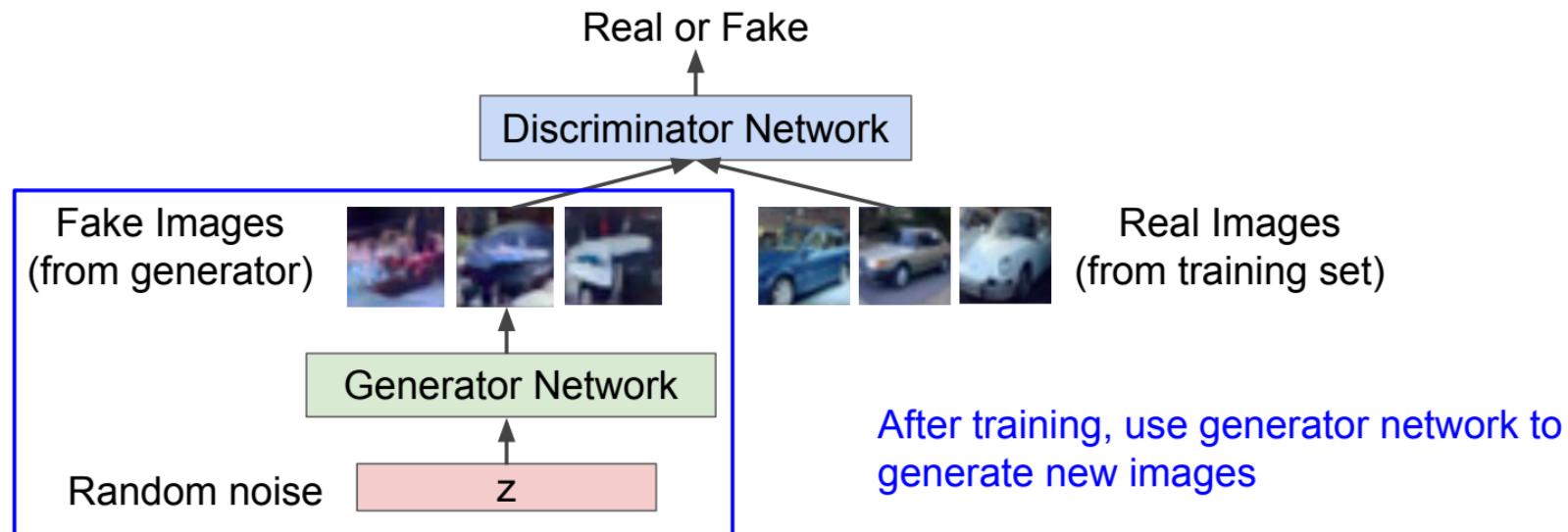
Berthelot, et al. "Began: Boundary equilibrium generative adversarial networks." arXiv preprint arXiv:1703.10717 (2017)

Training GANs: Two-player game

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

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

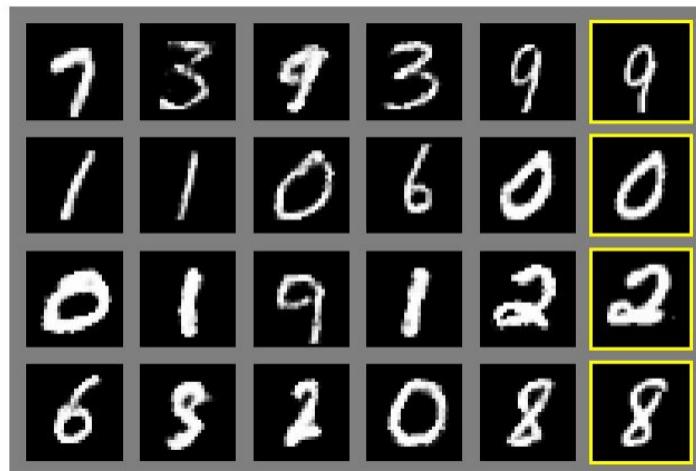
Discriminator network: try to distinguish between real and fake images



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

Generative Adversarial Nets

Generated samples

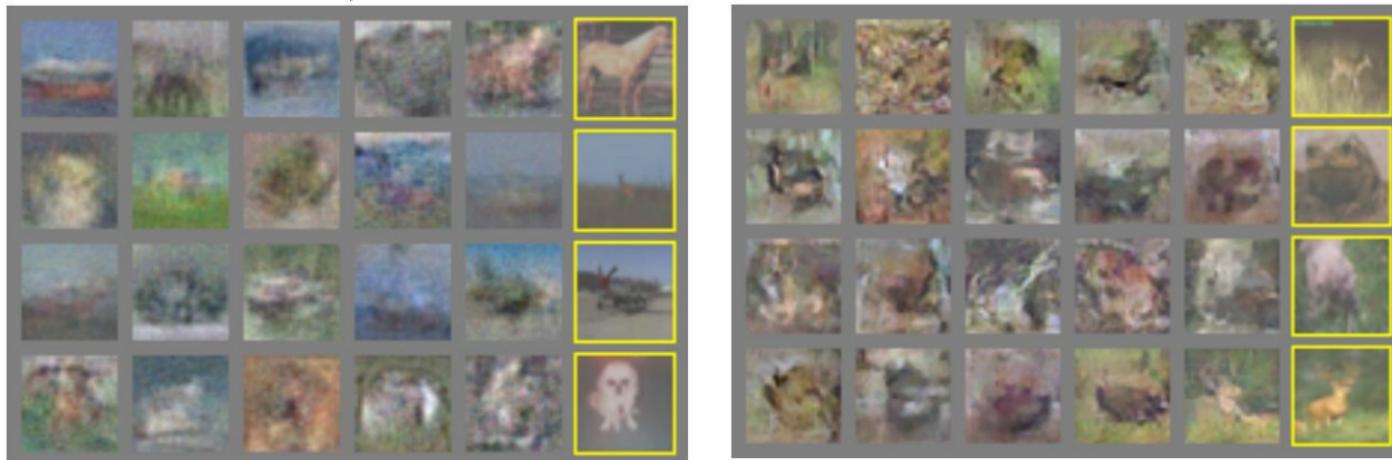


Nearest neighbor from training set

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

Generative Adversarial Nets

Generated samples (CIFAR-10)



Nearest neighbor from training set

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

Generative Adversarial Nets: Convolutional Architectures

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

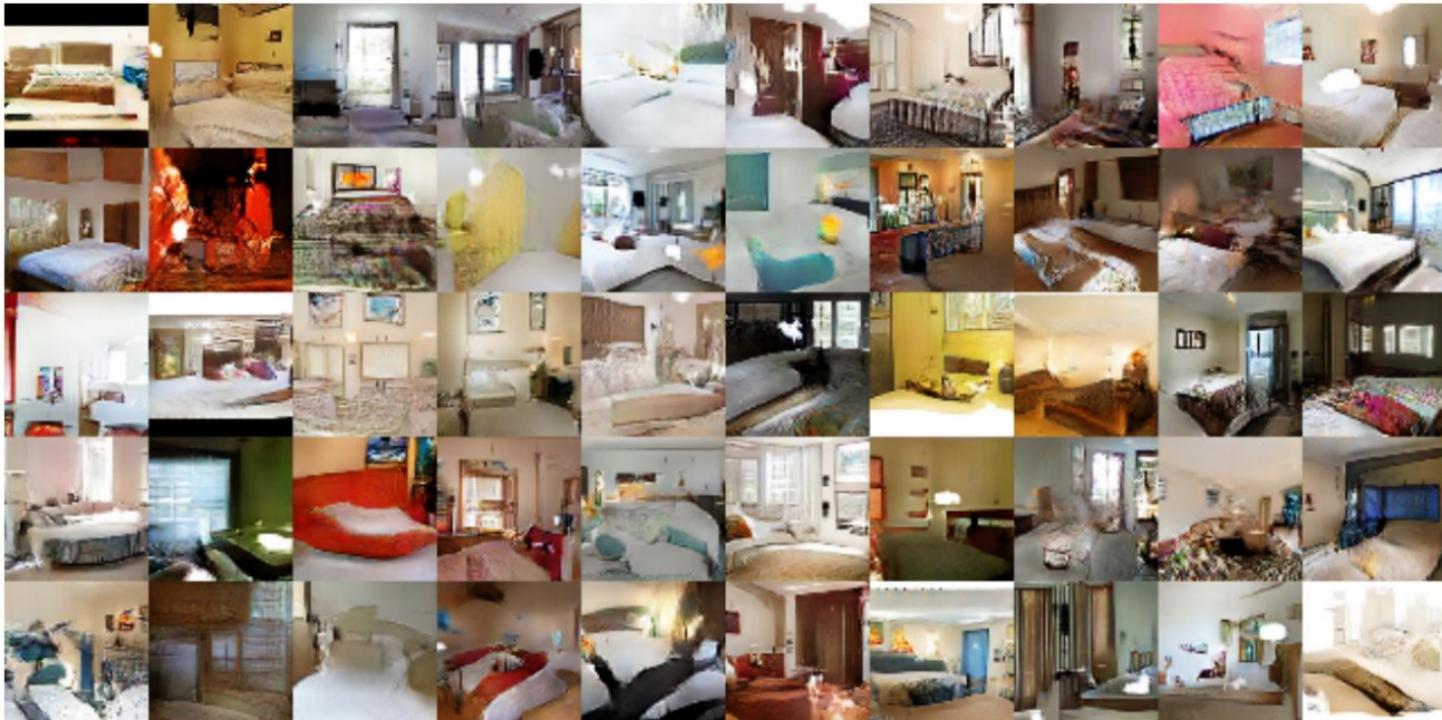
Architecture guidelines for stable Deep Convolutional GANs

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

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

Generative Adversarial Nets: Convolutional Architectures

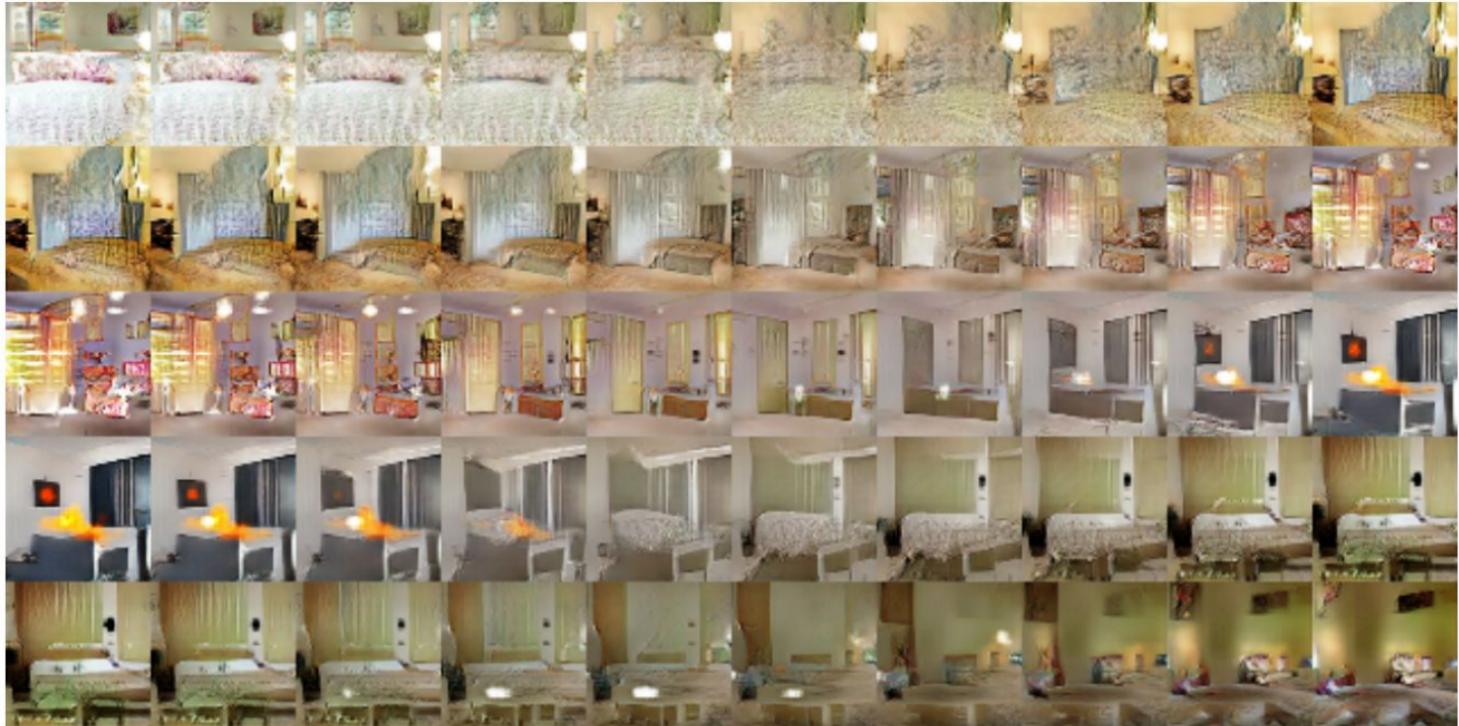
Samples
from the
model look
much
better!



Radford et al,
ICLR 2016

Generative Adversarial Nets: Convolutional Architectures

Interpolating
between
random
points in latent
space



Radford et al,
ICLR 2016

Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016

Smiling woman



Neutral woman



Neutral man



Samples
from the
model



Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016

Smiling woman Neutral woman Neutral man

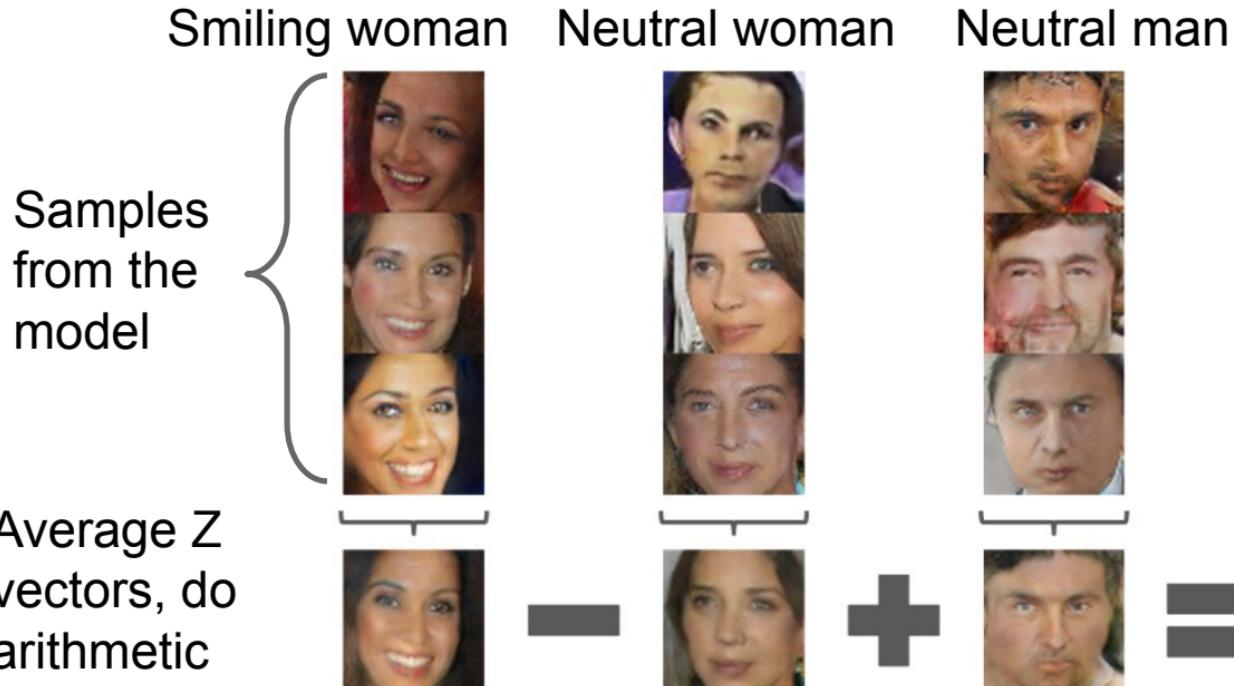
Samples
from the
model



Average Z
vectors, do
arithmetic



Generative Adversarial Nets: Interpretable Vector Math



Radford et al, ICLR 2016

Smiling Man



Generative Adversarial Nets: Interpretable Vector Math

Glasses man



No glasses man



No glasses woman



Radford et al,
ICLR 2016

Woman with glasses



2017: Explosion of GANs

“The GAN Zoo”

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

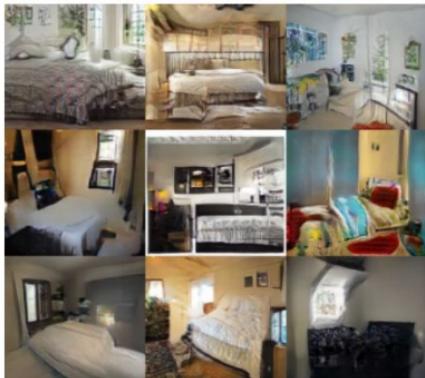
- GAN - Generative Adversarial Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- AdaGAN - AdaGAN: Boosting Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference
- AM-GAN - Generative Adversarial Nets with Labeled Data by Activation Maximization
- AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
- ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorical GANs
- b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
- Bayesian GAN - Deep and Hierarchical Implicit Models
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BiGAN - Adversarial Feature Learning
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- CGAN - Conditional Generative Adversarial Nets
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
- CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CoGAN - Coupled Generative Adversarial Networks

- Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
- C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training
- CS-GAN - Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
- CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
- CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
- DTN - Unsupervised Cross-Domain Image Generation
- DCGAN - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- DiscoGAN - Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
- DR-GAN - Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
- DualGAN - DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
- EBGAN - Energy-based Generative Adversarial Network
- f-GAN - f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
- FF-GAN - Towards Large-Pose Face Frontalization in the Wild
- GAWWN - Learning What and Where to Draw
- GeneGAN - GeneGAN: Learning Object Transfiguration and Attribute Subspace from Unpaired Data
- Geometric GAN - Geometric GAN
- GoGAN - Gang of GANs: Generative Adversarial Networks with Maximum Margin Ranking
- GP-GAN - GP-GAN: Towards Realistic High-Resolution Image Blending
- IAN - Neural Photo Editing with Introspective Adversarial Networks
- iGAN - Generative Visual Manipulation on the Natural Image Manifold
- IcGAN - Invertible Conditional GANs for image editing
- ID-CGAN - Image De-raining Using a Conditional Generative Adversarial Network
- Improved GAN - Improved Techniques for Training GANs
- InfoGAN - InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- LAGAN - Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis
- LAPGAN - Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

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

2017: Explosion of GANs

Better training and generation



LSGAN, Zhu 2017.



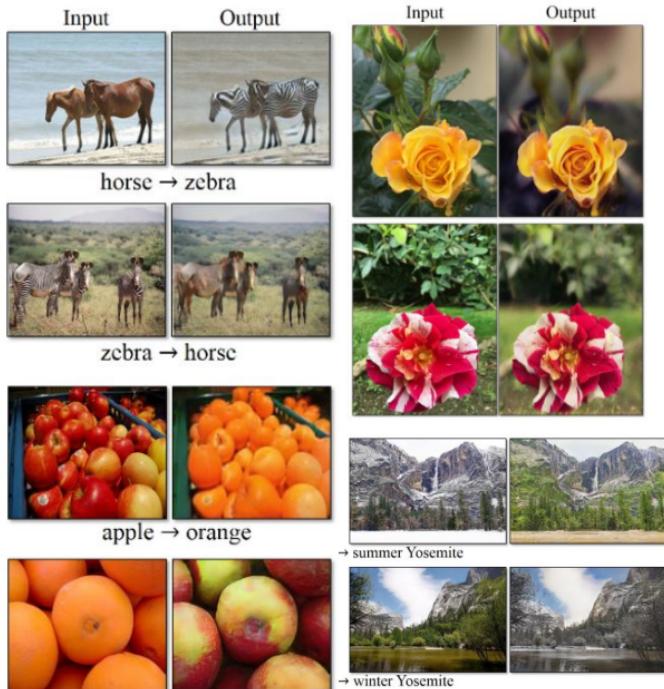
Wasserstein GAN,
Arjovsky 2017.
Improved Wasserstein
GAN, Gulrajani 2017.



Progressive GAN, Karras 2018.

2017: Explosion of GANs

Source->Target domain transfer



Text -> Image Synthesis

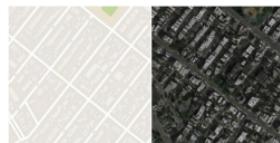
this small bird has a pink breast and crown, and black primaries and secondaries.

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



Reed et al. 2017.

Many GAN applications



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

2019: BigGAN

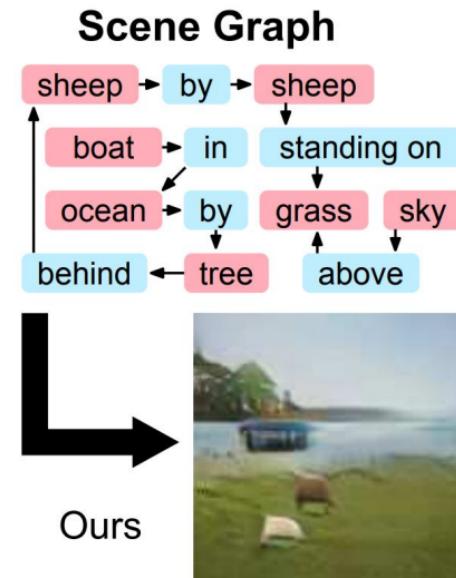


Brock et al., 2019

Scene graphs to GANs

Specifying exactly what kind of image you want to generate.

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

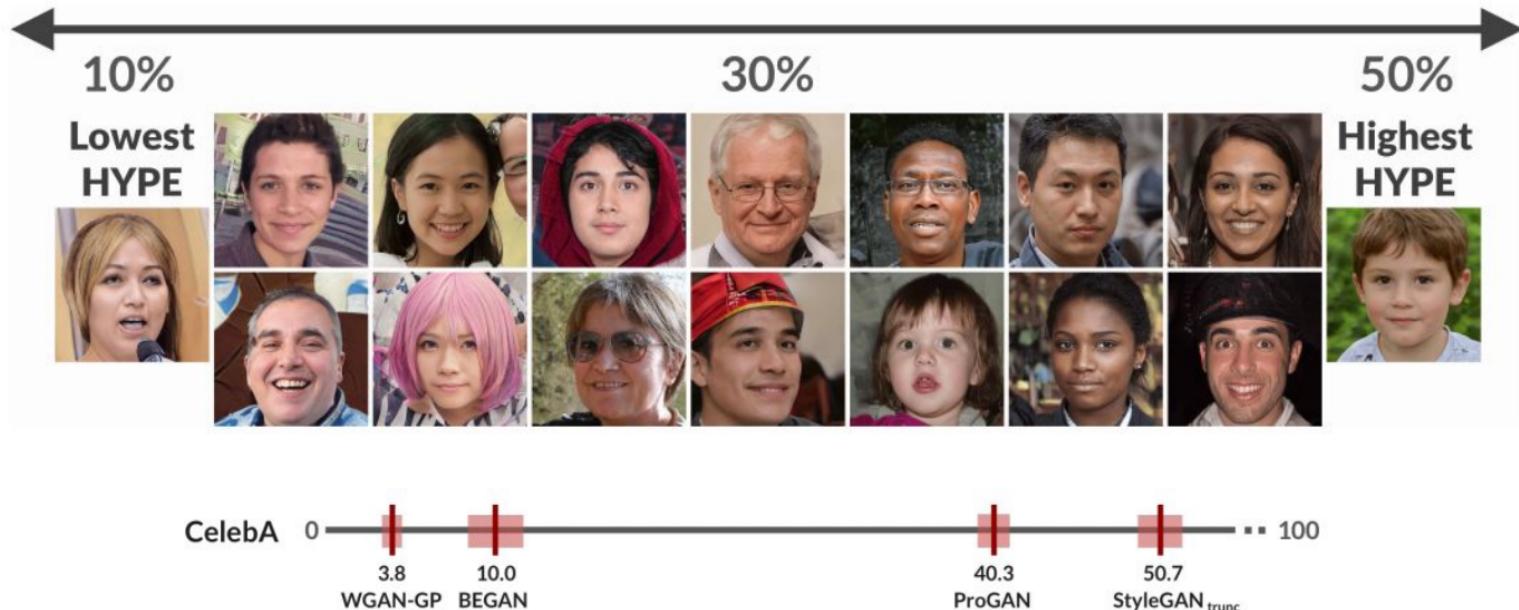


Figures copyright 2019. Reproduced with permission.

Johnson et al. Image Generation from Scene Graphs, CVPR 2019

HYPE: Human eYe Perceptual Evaluations

hype.stanford.edu



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

Figures copyright 2019. Reproduced with permission.

Summary: GANs

Don't work with an explicit density function

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

Pros:

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

Cons:

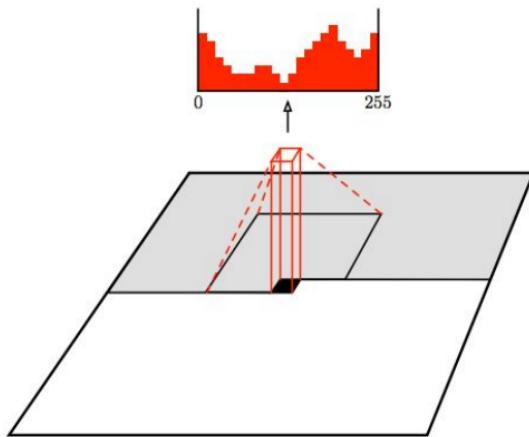
- Trickier / more unstable to train
- Can't solve inference queries such as $p(x)$, $p(z|x)$

Active areas of research:

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

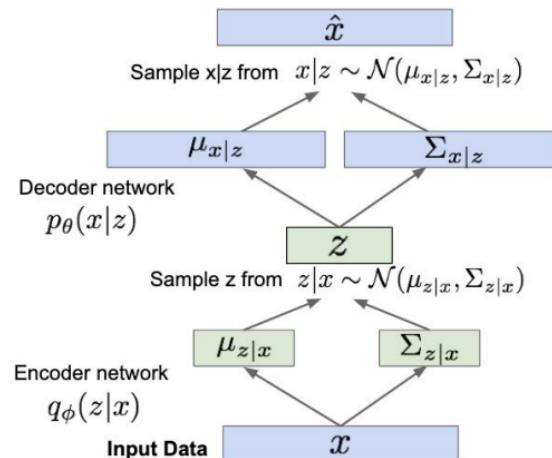
Summary

Autoregressive models:
PixelRNN, PixelCNN



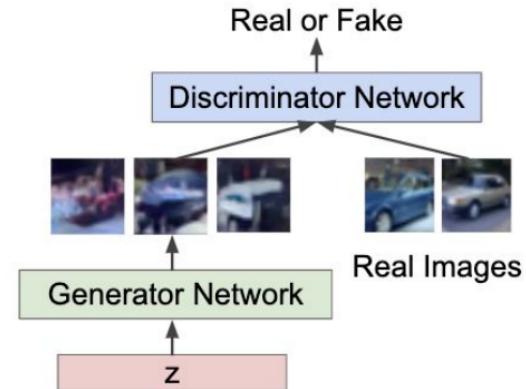
Van der Oord et al, "Conditional image generation with pixelCNN decoders", NIPS 2016

Variational Autoencoders



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

Generative Adversarial Networks (GANs)



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

Useful Resources on Generative Models

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

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

Next: Self-Supervised Learning