# Lecture 15:
# Generative Models

# Administrative

- A3 is due 11/29
- Quiz 3 on 11/30

# Supervised vs Unsupervised Learning

**Supervised Learning**

**Data**: (x, y)
x is data, y is label

**Goal**: Learn a *function* to map x -> 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 -> y

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



Cat

Classification

# Supervised vs Unsupervised Learning

**Supervised Learning**

**Data**: (x, y)
x is data, y is label

**Goal**: Learn a *function* to map x -> y

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



*A cat sitting on a suitcase on the floor*

Image captioning

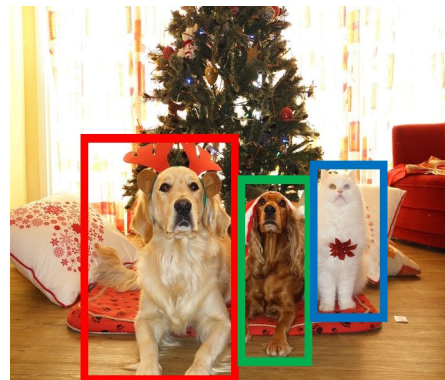# Supervised vs Unsupervised Learning

**Supervised Learning**

**Data**: (x, y)
x is data, y is label

**Goal**: Learn a *function* to map x -> y

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



**DOG**, **DOG**, **CAT**

Object Detection
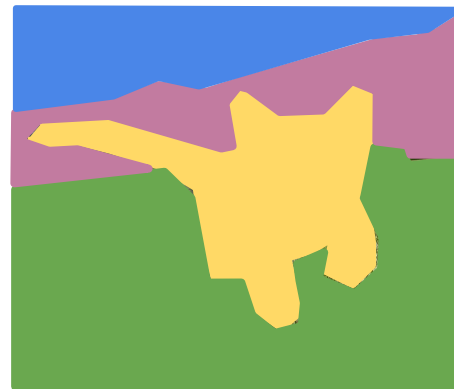
# Supervised vs Unsupervised Learning

**Supervised Learning**

**Data**: (x, y)
x is data, y is label

**Goal**: Learn a *function* to map x -> y

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



GRASS, CAT, TREE, SKY

Semantic Segmentation
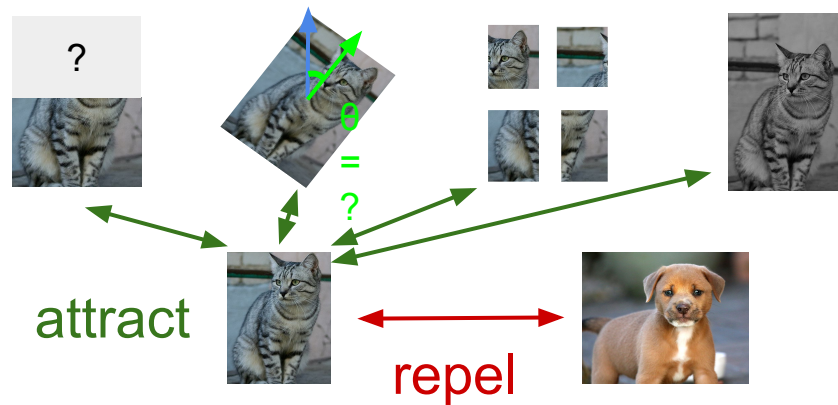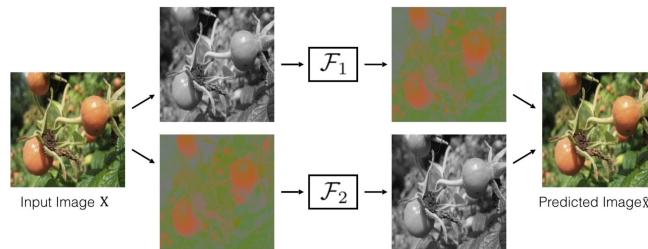
# Supervised vs Unsupervised Learning

**Self-Supervised Learning**

**Data**: (x, y)
x is data, y is a proxy label

**Goal**: Learn a *function* to map x -> y

**Examples**: Inpainting, colorization, contrastive learning.



attract

repel

# 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.

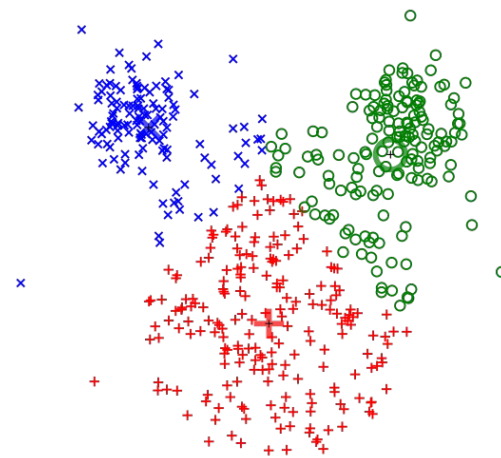# Supervised vs Unsupervised Learning

**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

# 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)
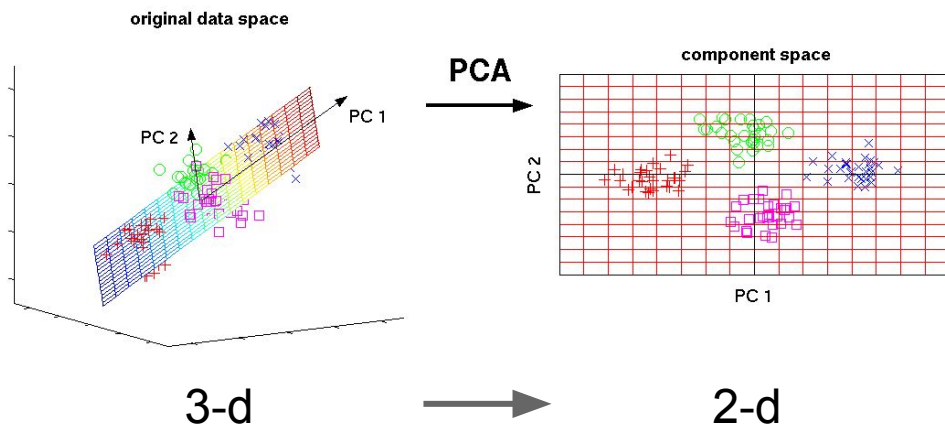
# Supervised vs Unsupervised Learning

**Unsupervised Learning**

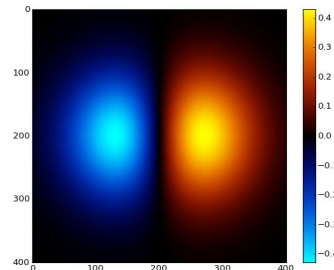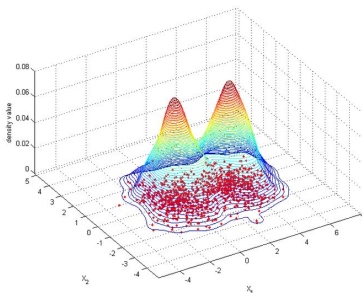**Data**: x
Just data, no labels!

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

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



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 -> y

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

**Unsupervised Learning**

**Data**: x
Just data, no labels!

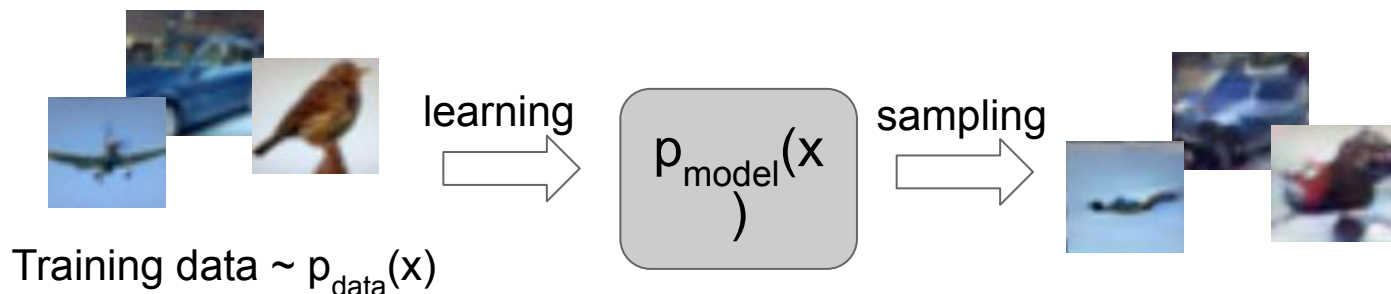**Goal**: Learn some underlying hidden *structure* of the data

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

# Generative Modeling

Given training data, generate new samples from same distribution



learning

$p_{model}(x)$

sampling

Training data ~ $p_{data}(x)$

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

# Generative Modeling

Given training data, generate new samples from same distribution



Training data ~ $p_{data}(x)$

learning → $p_{model}(x)$ → sampling
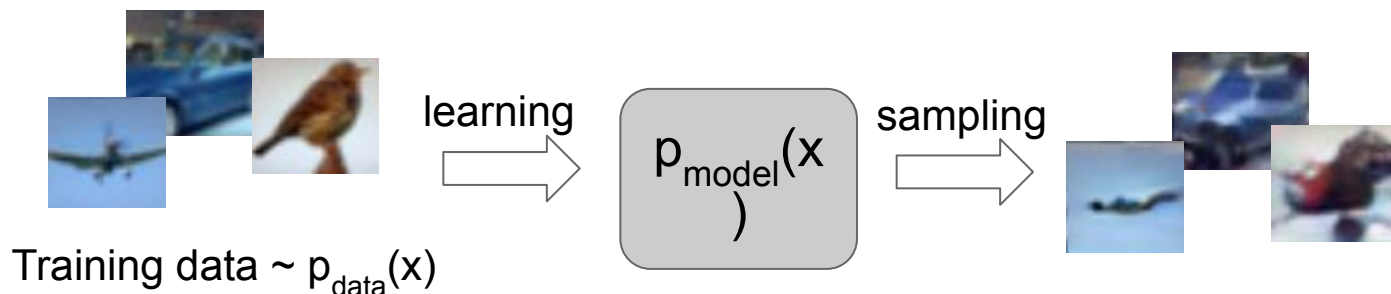
Formulate as density estimation problems**:**
- **Explicit density estimation**: explicitly define and solve for $p_{model}(x)$
- **Implicit density estimation**: learn model that can sample from $p_{model}(x)$ **without explicitly defining it.**

# Why Generative Models?
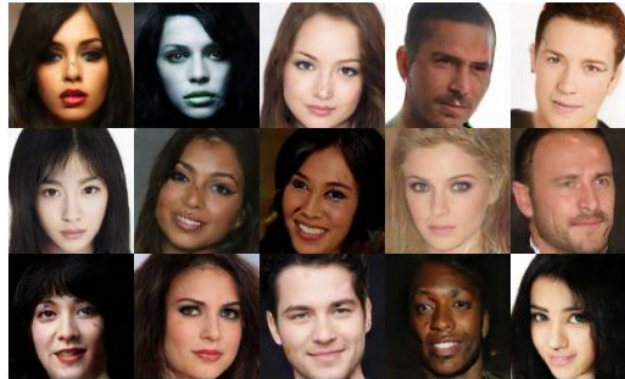


- Realistic samples for artwork, super-resolution, colorization, etc.
- Learn useful features for downstream tasks such as classification.
- Getting insights from high-dimensional data (physics, medical imaging, etc.)
- Modeling physical world for simulation and planning (robotics and reinforcement learning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

# Taxonomy of Generative Models

**Generative models**

Model can compute p(x)

Model does not compute p(x)
But can sample from p(x)

Explicit density

Implicit density



p(x) measures the likelihood of an image

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

# Taxonomy of Generative Models

**Generative models**

Explicit density

Implicit density

Model exactly calculates p(x)

Model approximates p(x)

Tractable density

Approximate density

Fully Visible Belief Nets
- Autoregressive
- NADE
- MADE
- NICE / RealNVP
- Glow
- Ffjord

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

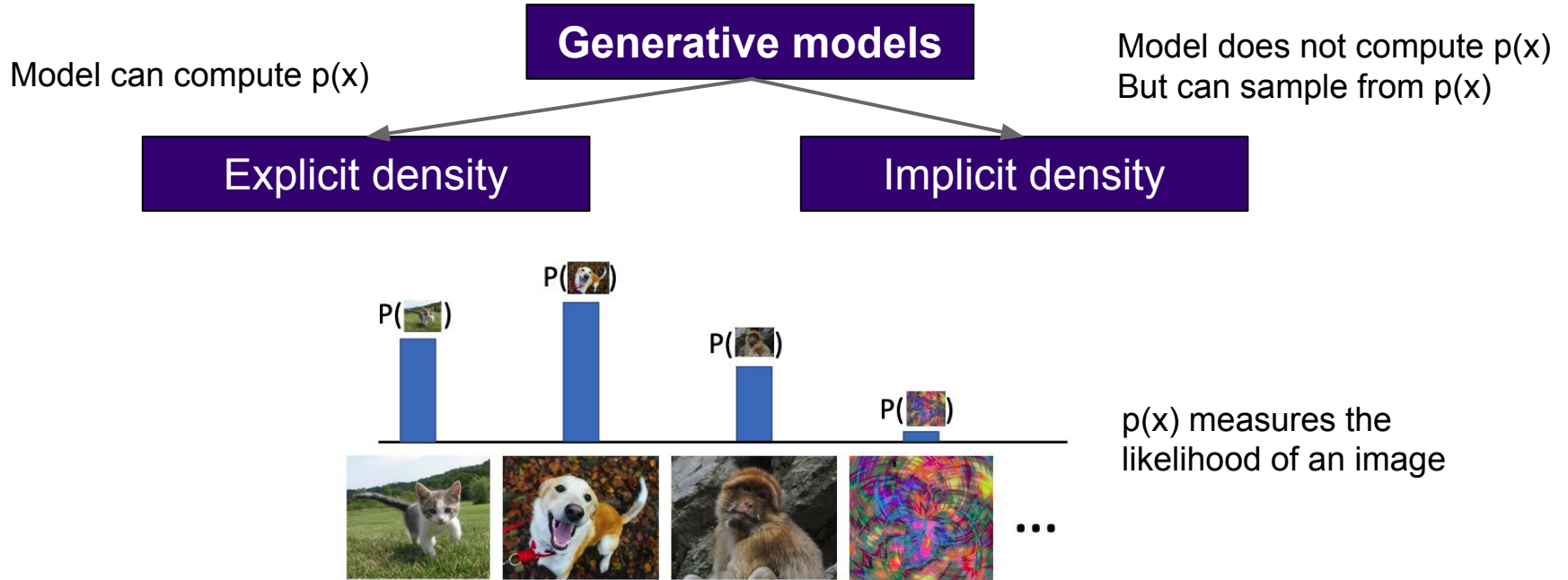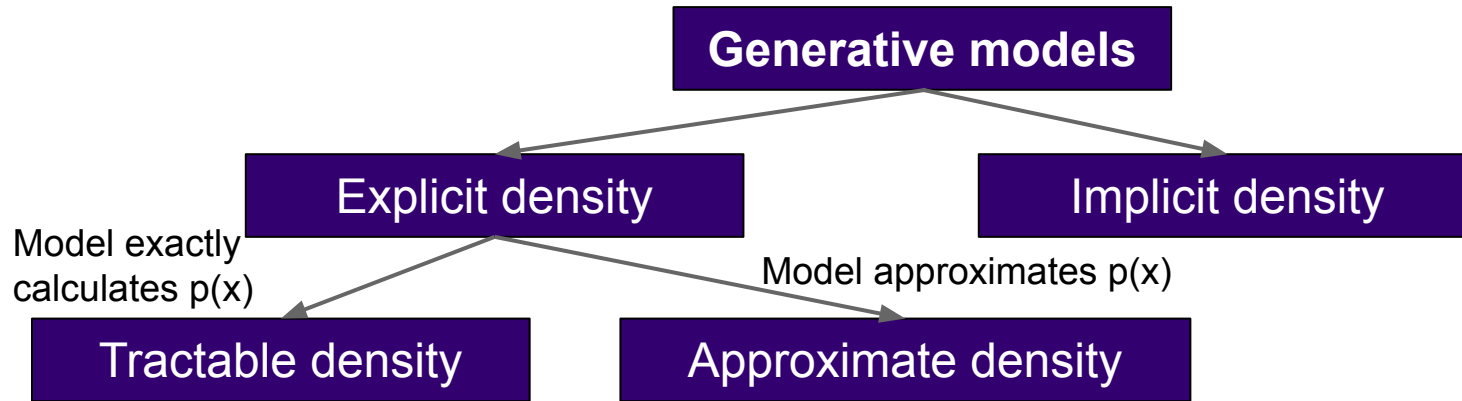# Taxonomy of Generative Models



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

# Taxonomy of Generative Models



Fully Visible Belief Nets
- NADE
- MADE
- PixelRNN/CNN
- NICE / RealNVP
- Glow
- Ffjord

Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.
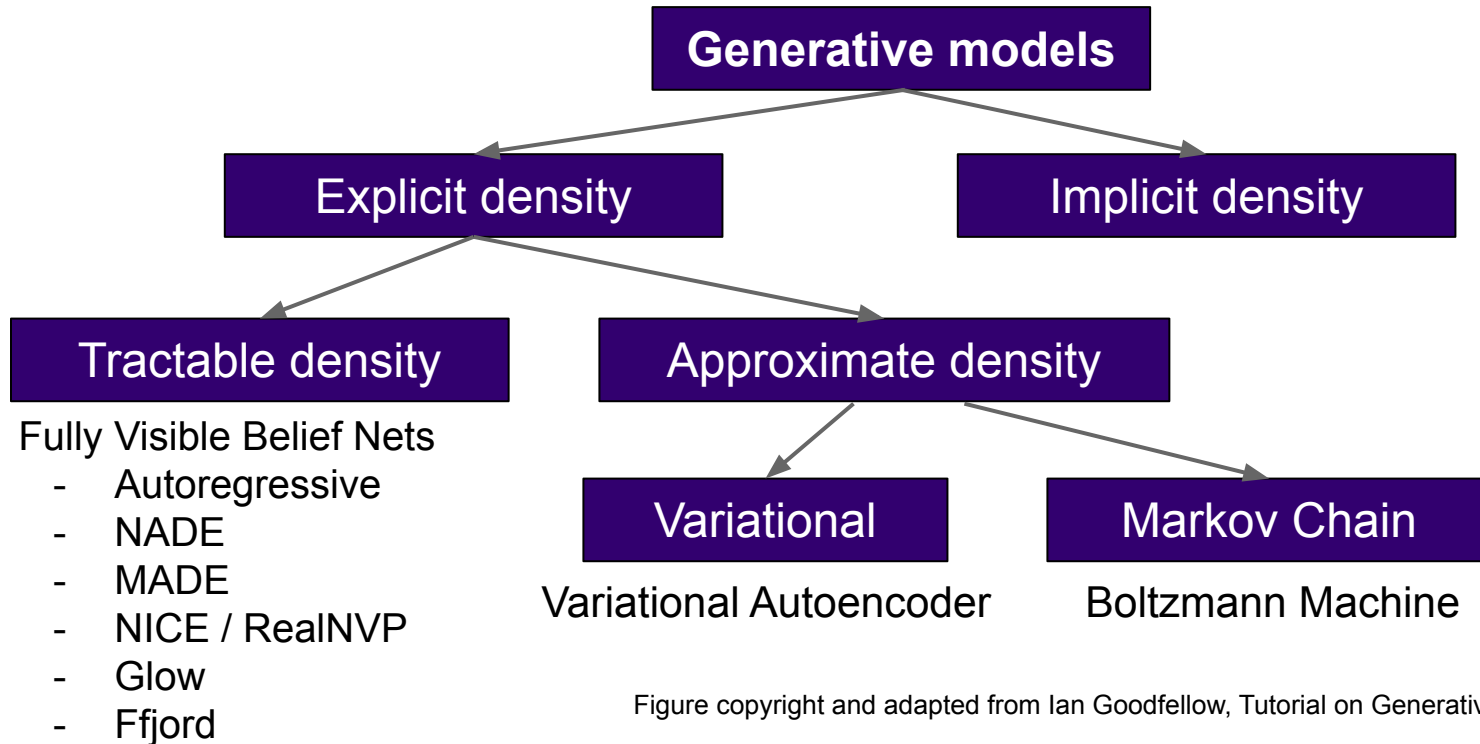
# Taxonomy of Generative Models

Today: discuss 3 types of generative models today
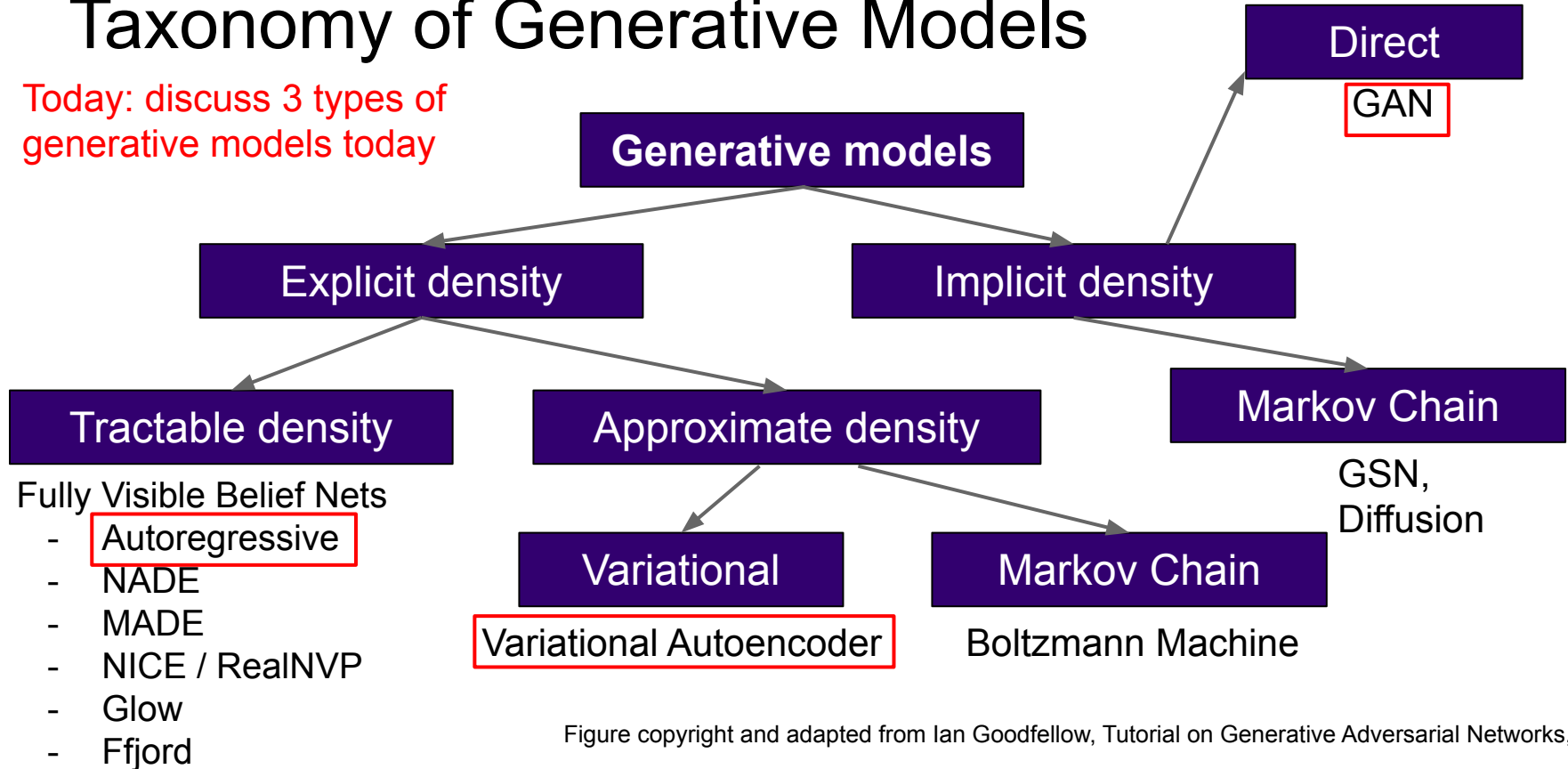
**Generative models**

Explicit density

Implicit density

**Direct**

GAN

Tractable density

Approximate density

Markov Chain

GSN, Diffusion

Fully Visible Belief Nets
- Autoregressive
- NADE
- MADE
- NICE / RealNVP
- Glow
- Ffjord

Variational

Markov Chain

Variational Autoencoder

Boltzmann Machine

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

# Autorgressive models

## (PixelRNN and PixelCNN)
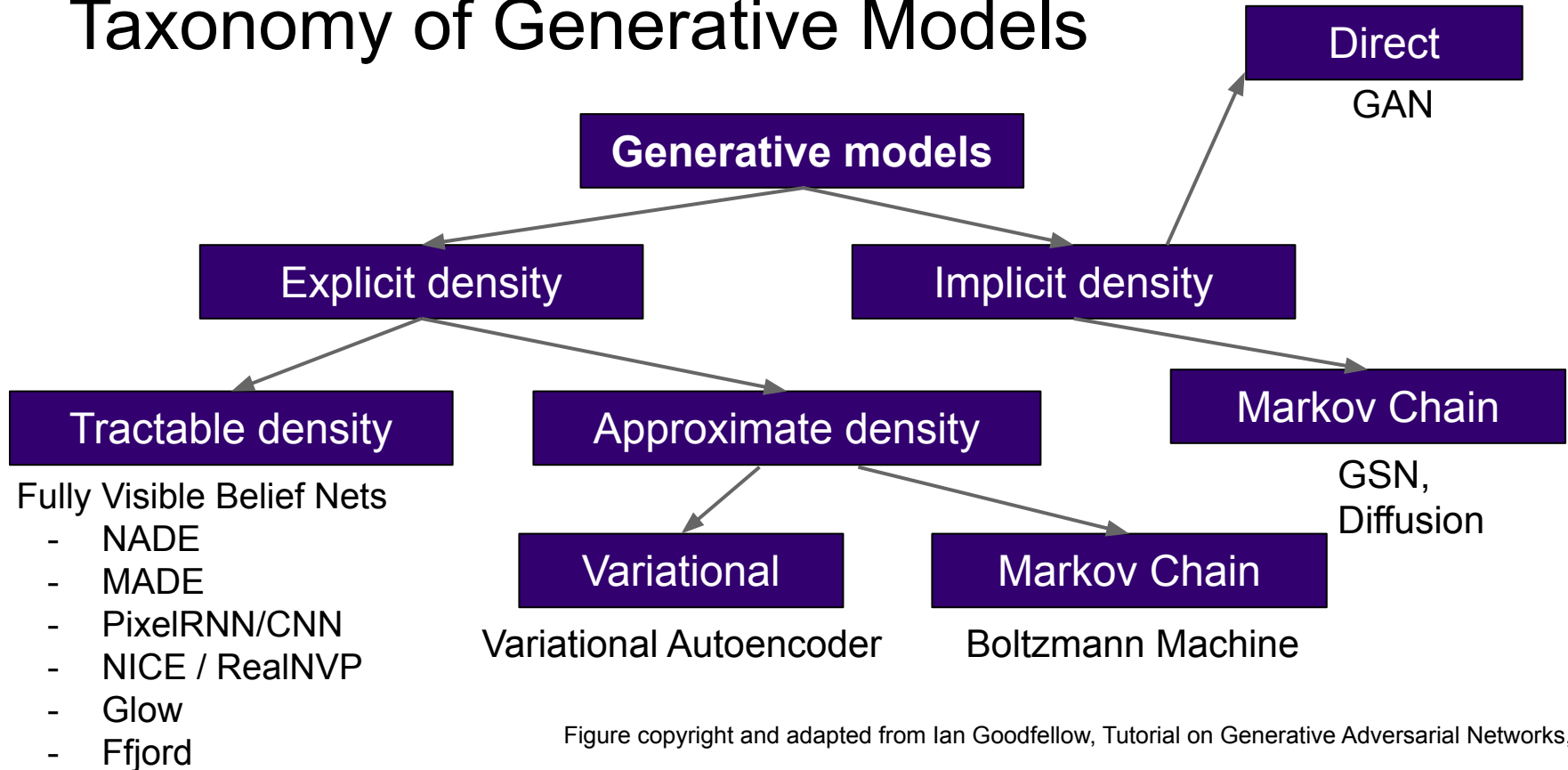
# Fully visible belief network (FVBN)

Explicit density model

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

Likelihood of
image x

Joint likelihood of each
pixel in the image

# Fully visible belief network (FVBN)

Explicit density model

Remember the probability chain rule:

$$p(x) = p(x_n | x_1, x_2, \ldots, x_{n-1}) p(x_1, x_2, \ldots, x_{n-1})$$

# Fully visible belief network (FVBN)

Explicit density model

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

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

Likelihood of image x

Probability of i'th pixel value given all previous pixels

Then maximize likelihood of training data
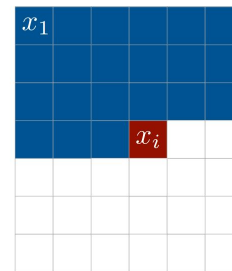
# Fully visible belief network (FVBN)

Explicit density model

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

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

Likelihood of image x

Probability of i'th pixel value given all previous pixels
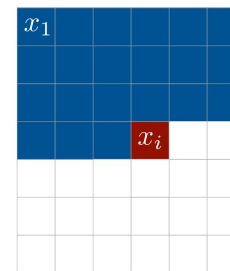
Complex distribution over pixel values => Express using a neural network!

Then maximize likelihood of training data

# Recurrent Neural Network
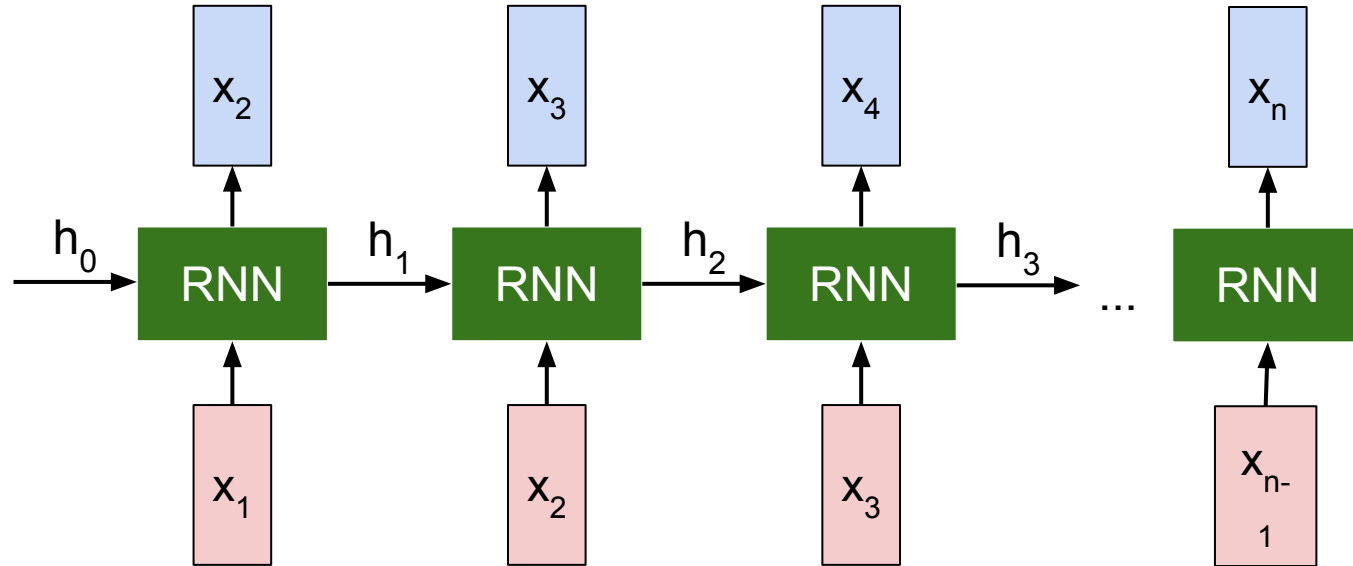


$$p(x_i | x_1, ..., x_{i-1})$$

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

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

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

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)
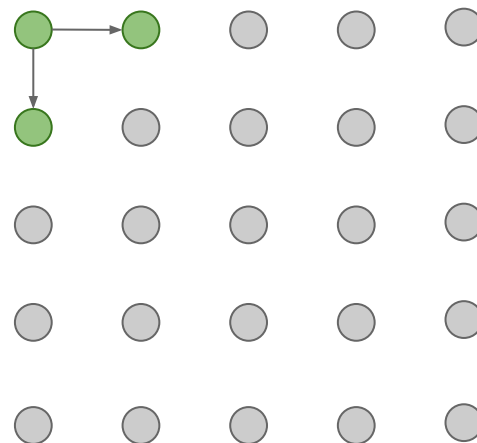
# PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled
using an RNN (LSTM)

# PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

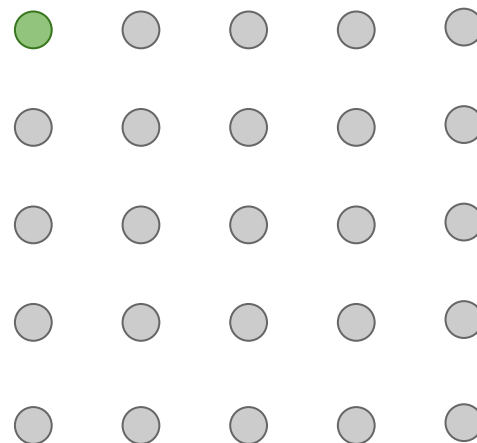Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow in both training and inference!
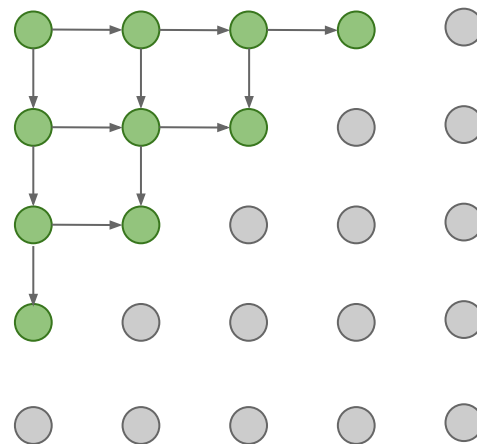
# PixelCNN [van der Oord et al. 2016]

Still generate image pixels starting from corner

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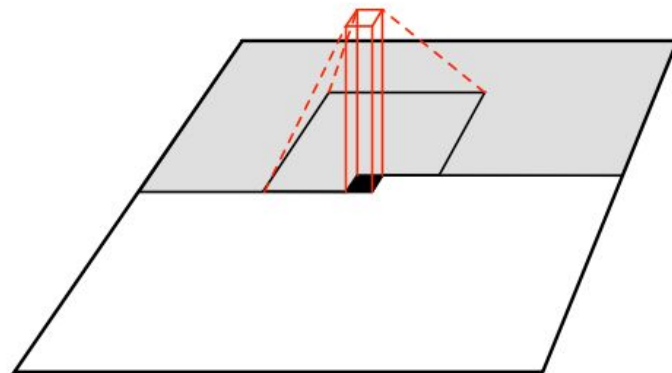


Figure copyright van der Oord et al., 2016. Reproduced with permission.

# PixelCNN [van der Oord et al. 2016]

Still generate image pixels starting from corner

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

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

Generation is still slow:
For a 32x32 image, we need to do forward passes of the network 1024 times for a single image

Softmax loss over pixel values at every location
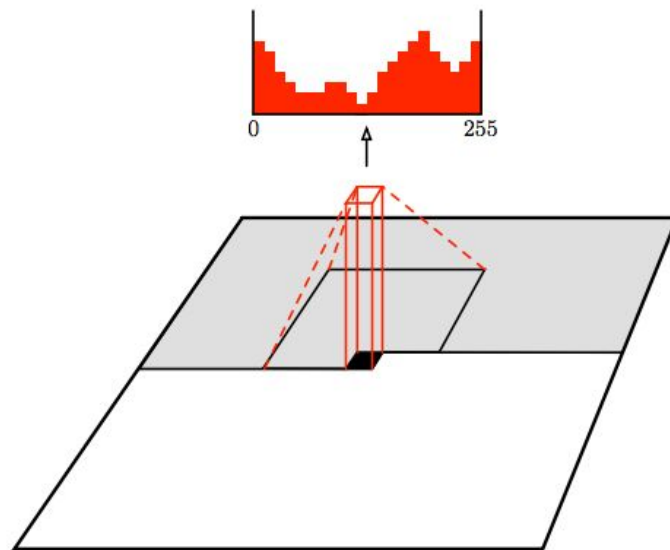


Figure copyright van der Oord et al., 2016. Reproduced with permission.

# Generation Samples
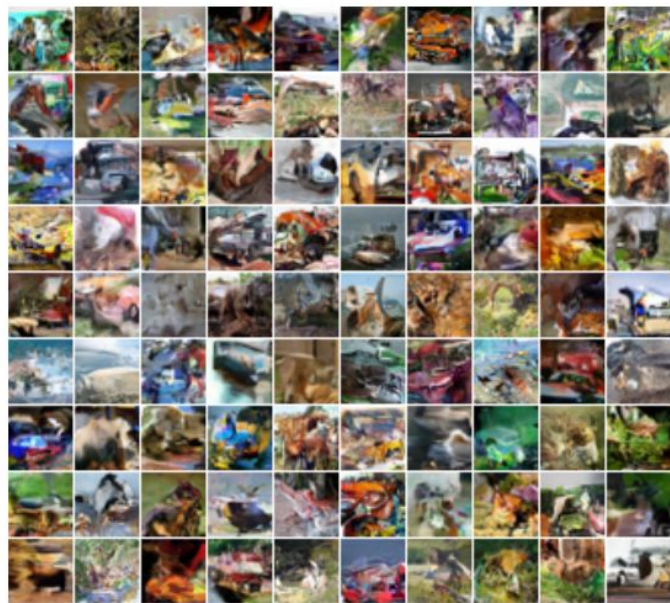


32x32 CIFAR-10
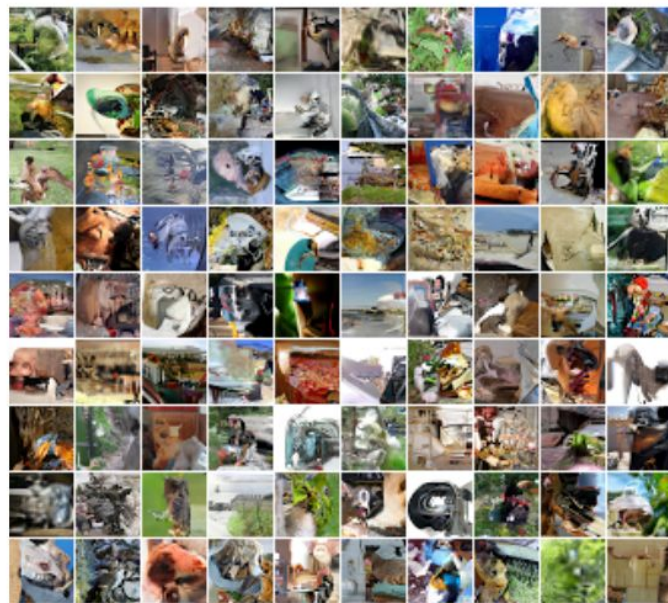


32x32 ImageNet

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

# PixelRNN and PixelCNN

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

Con:
- Sequential generation => slow

Improving PixelCNN performance
- Gated convolutional layers
- Short-cut connections
- Discretized logistic loss
- Multi-scale
- Training tricks
- Etc…

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

# Taxonomy of Generative Models
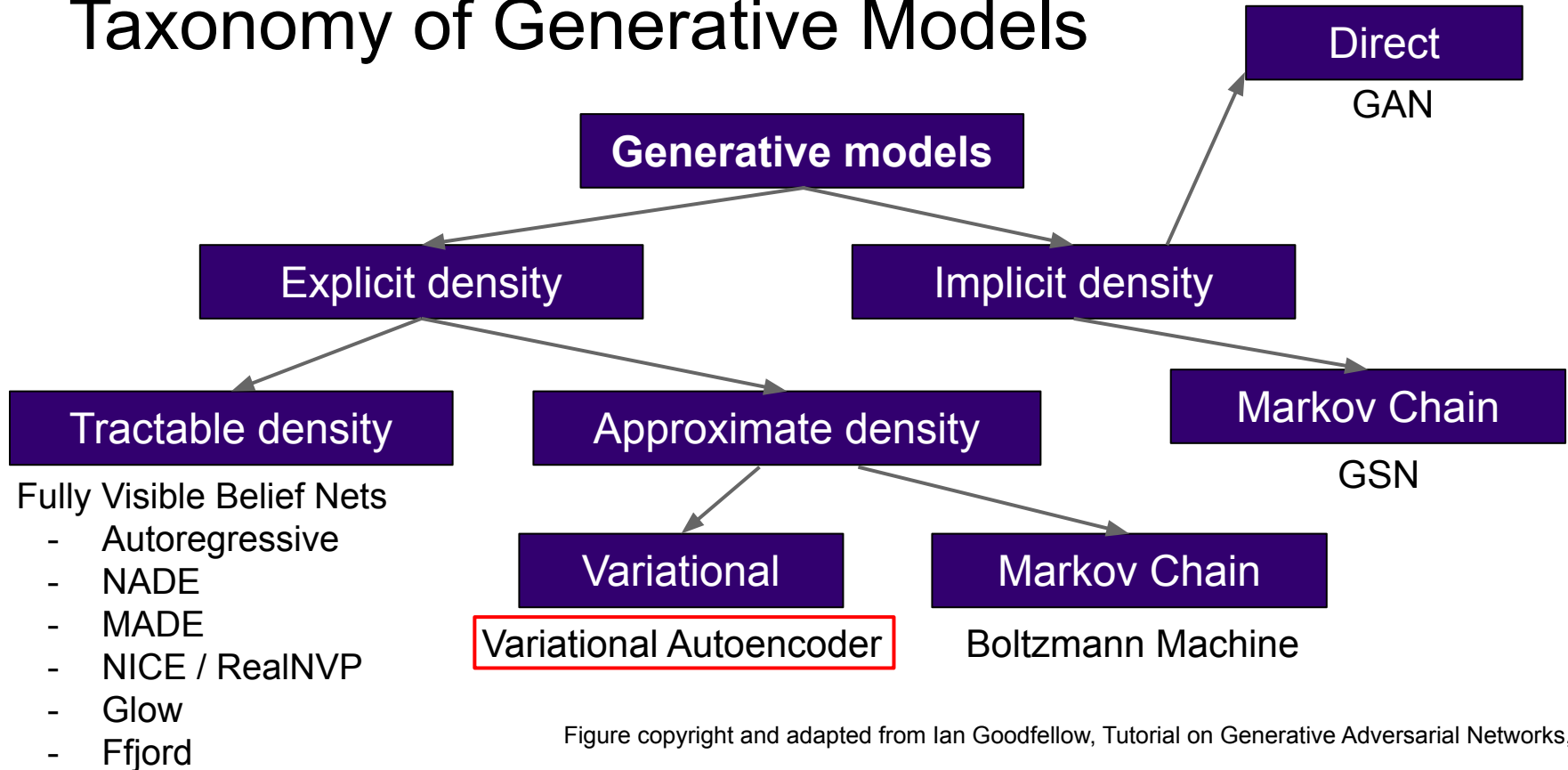


Fully Visible Belief Nets
- Autoregressive
- NADE
- MADE
- NICE / RealNVP
- Glow
- Ffjord

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

# Variational Autoencoders (VAE)

# So far...

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

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

# So far...

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

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

Variational Autoencoders (VAEs) define intractable density function with latent **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...

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

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

Variational Autoencoders (VAEs) define intractable density function with latent **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 z?

# ~~Variational~~ Autoencoders

# Some background first: Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data
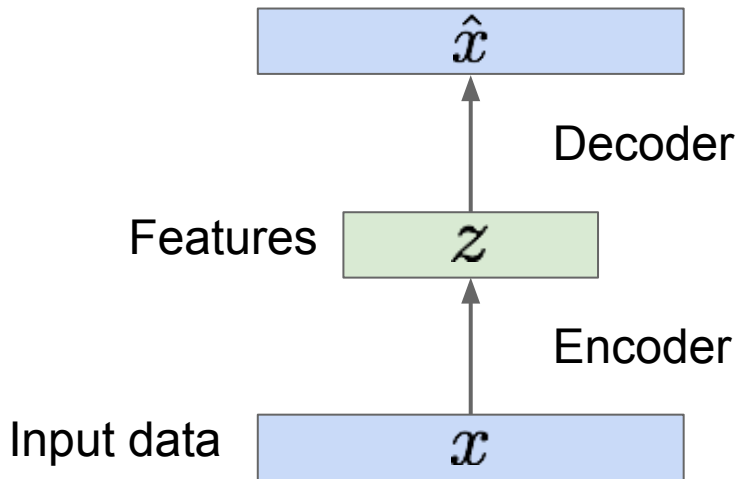
**Z** should extract useful information (maybe object identities, properties, scene type, etc) that we can use for downstream tasks
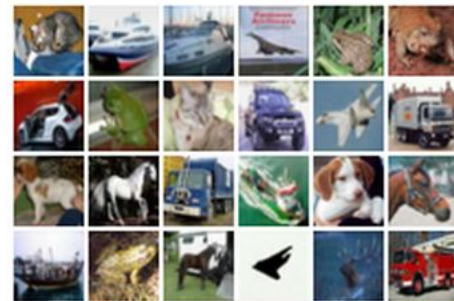
$\hat{x}$

Decoder

Features $z$

Encoder

Input data $x$

# 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?

$\hat{x}$

Decoder

Features $z$

Encoder

Input data $x$

# 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?

$\hat{x}$

Decoder
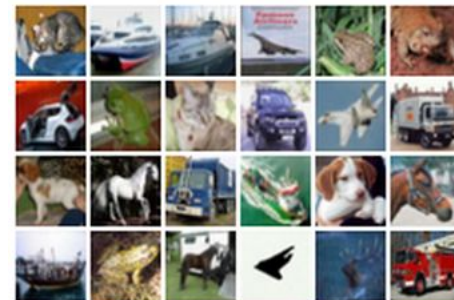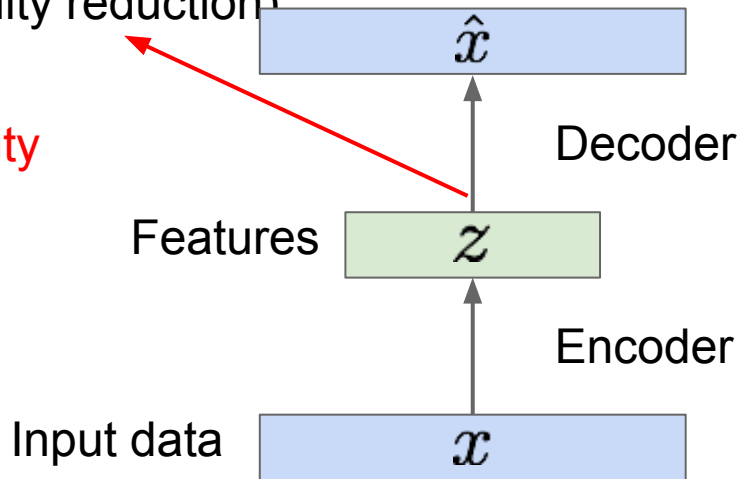
Features $z$

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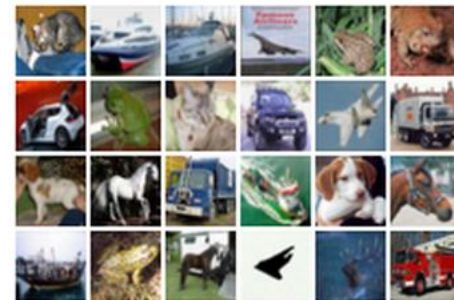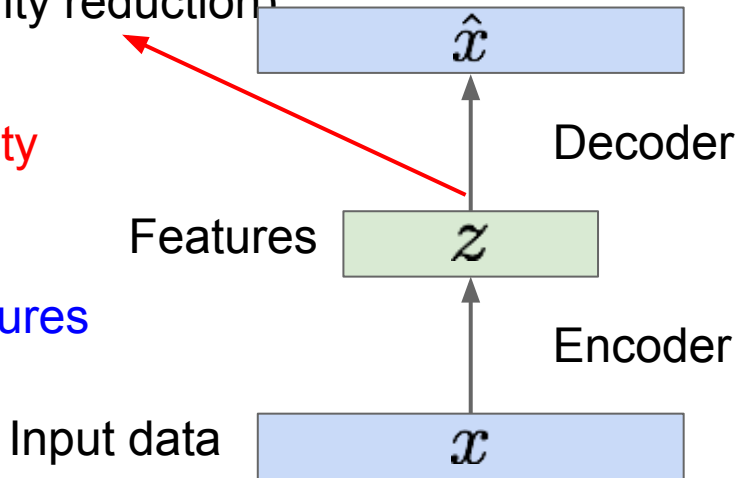
Encoder

Input data $x$

# Some background first: Autoencoders

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

**How do we learn this z?**

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

$$\|\hat{x} - x\|_2^2$$

$\hat{x}$

Decoder

Features $z$

Encoder

Input data $x$
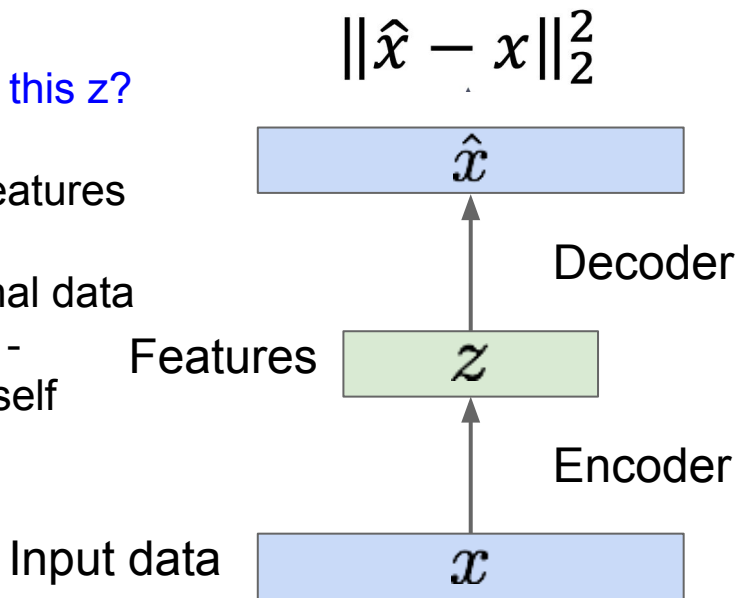
**Learning objective:** reconstruct the image and use l2 loss.

No labels are necessary!!
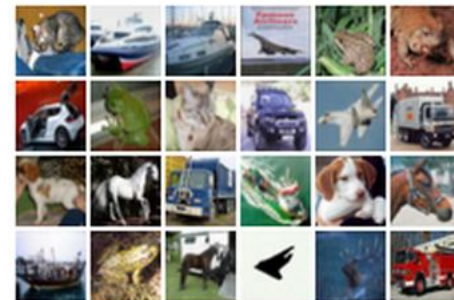
# Some background first: Autoencoders

Images reconstructed are blurry because they don't contain pixel-perfect information

Reconstructed input data

$\hat{x}$

Decoder

Features  $z$

Encoder

Input data  $x$

**Encoder**: 4-layer conv
**Decoder**: 4-layer upconv

Input data

# Some background first: Autoencoders



Reconstructed input data
$\hat{x}$

Decoder

After training, throw away decoder

Features
$z$

Encoder

Input data
$x$
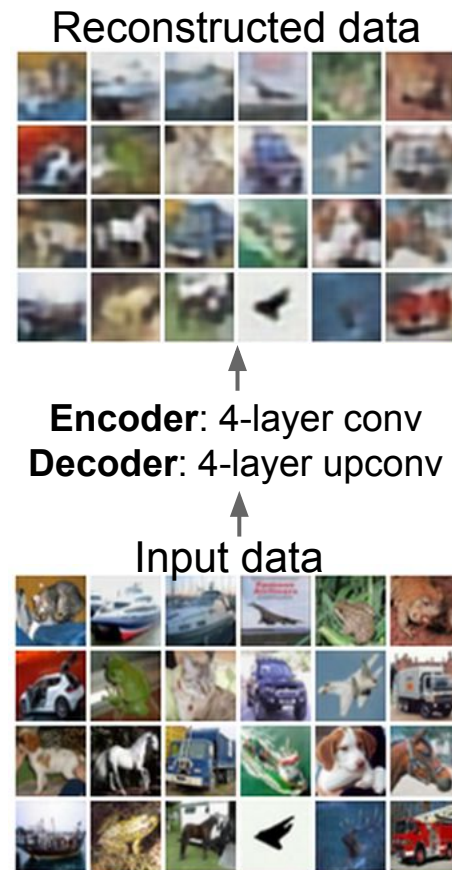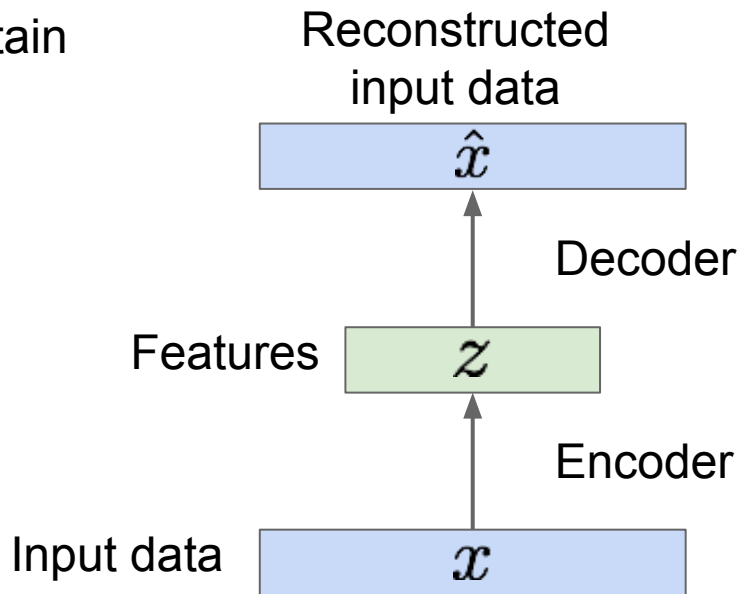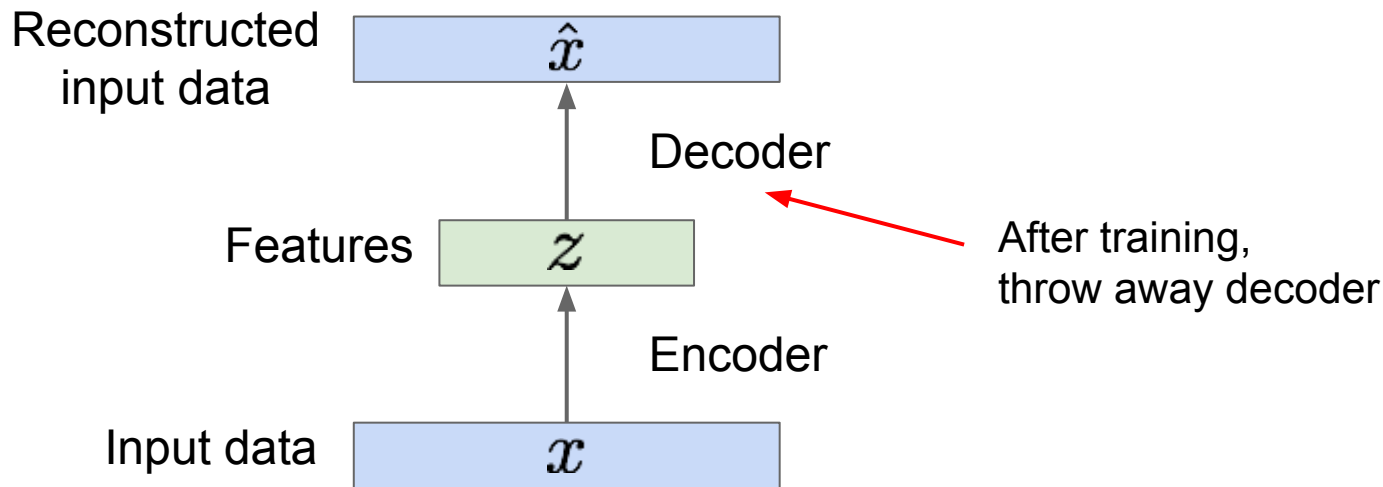
# Some background first: Autoencoders

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

Loss function (Softmax, etc)

bird        plane

dog      deer      truck

Predicted Label    $\hat{y}$      $y$

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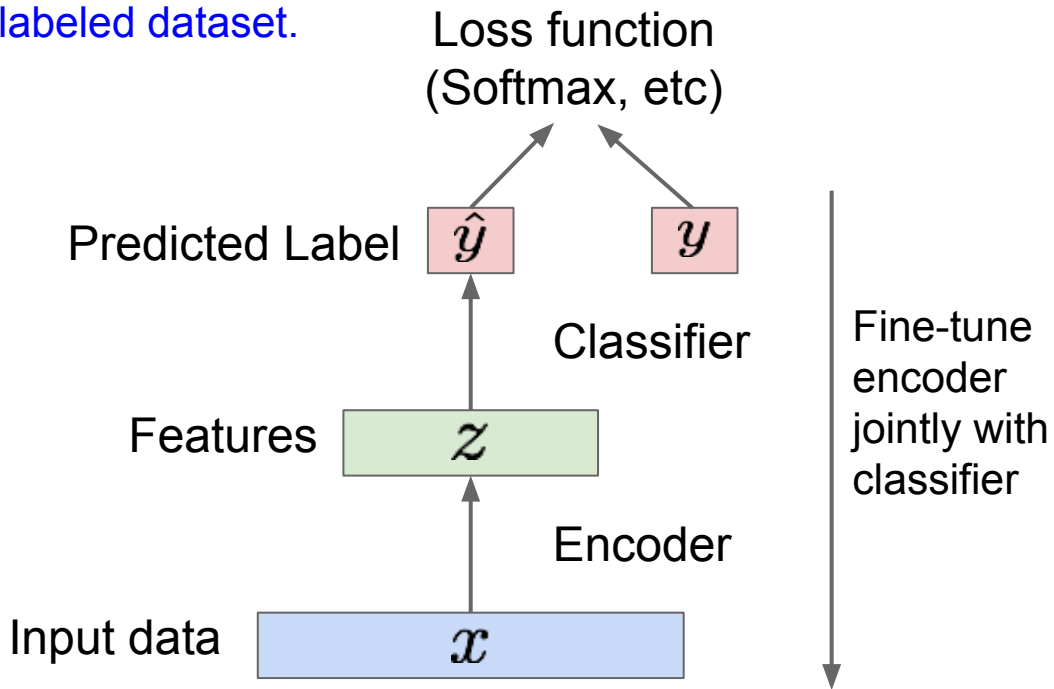
Encoder can be used to initialize a **supervised** model

Features    $z$

Fine-tune encoder jointly with classifier

Train for final task (sometimes with small data)

Encoder

Input data    $x$
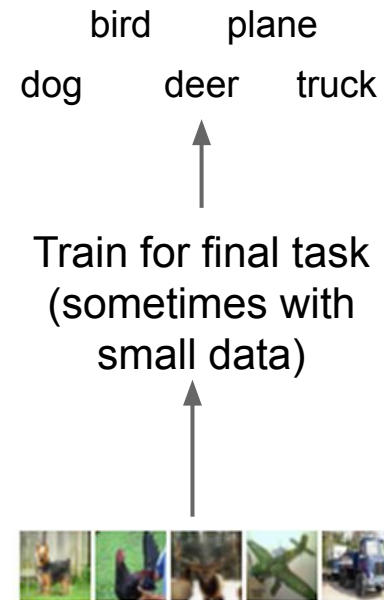
# Some background first: Autoencoders
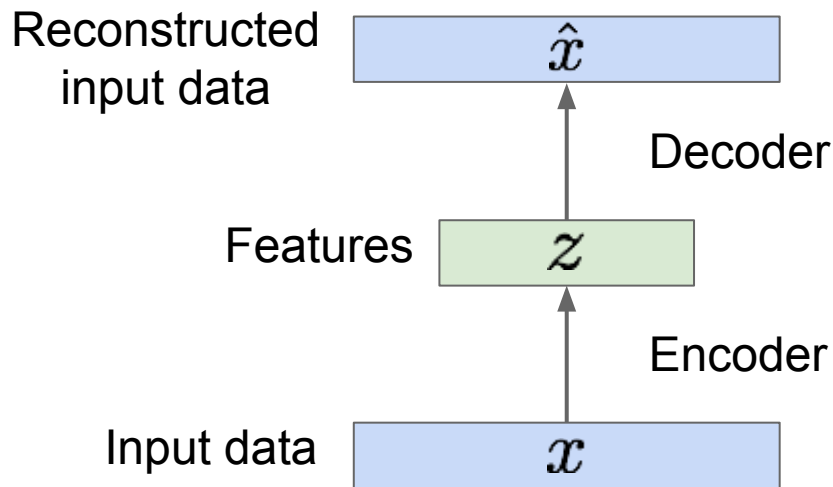
Autoencoders can reconstruct data, and can learn features to initialize a supervised model

Features capture factors of variation in training data.

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

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

Reconstructed input data

$\hat{x}$

Decoder

Features

$z$

Encoder

Input data

$x$

# 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**

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

$x$
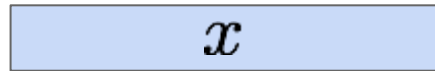
Decoder

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

$z$

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

$$x$$

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

$$z$$

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

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

# Variational Autoencoders

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

Sample from true conditional

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

$$x$$

Sample from true prior

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

$$z$$

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

# Variational Autoencoders

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

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

How should we represent this model?

$\boxed{x}$

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

$\boxed{z}$

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 $\theta*$
of this generative model given training data x.

How should we represent this model?

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

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

# Variational Autoencoders



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

Sample from true conditional

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

Sample from true prior

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

Decoder network

$x$

$z$

How should we represent this model?

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

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

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

# Variational Autoencoders

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

How to train the model?

Sample from true conditional

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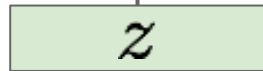


$x$

Decoder network

$z$

Sample from true prior

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

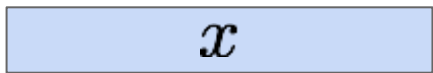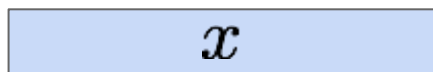Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders

Sample from true conditional

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

Sample from true prior

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

$x$

Decoder network

$z$

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

How to train the model?

Learn model parameters to maximize likelihood of training data

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

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

# Variational Autoencoders
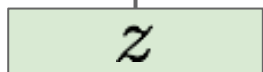
Sample from
true conditional

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

Sample from
true prior

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



$x$

Decoder
network

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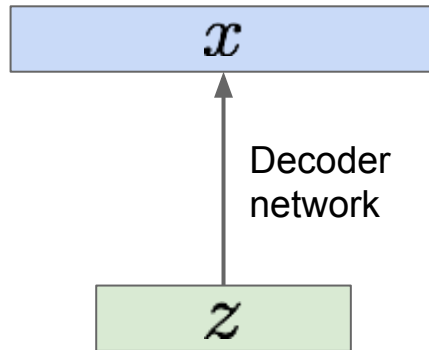
We want to estimate the true parameters $\theta*$ of this generative model given training data x.

How to train the model?

Learn model parameters to maximize likelihood of training data

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

Q: What is the problem with this?

Intractable!

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

# Variational Autoencoders: Intractability

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

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

# Variational Autoencoders: Intractability

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

✔

↑ Simple Gaussian prior

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

# Variational Autoencoders: Intractability

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

✔ ✔

Decoder neural network

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

# Variational Autoencoders: Intractability

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

✔ ✔

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

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

# Variational Autoencoders: Intractability

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

✔ ✔

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

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

Monte Carlo estimation is too high variance

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

# Variational Autoencoders: Intractability

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

Another idea: $p_\theta(x) = \dfrac{p_\theta(x \mid z) p_\theta(z)}{p_\theta(z \mid x)}$ ⟵ Use Bayes rule

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

# Variational Autoencoders: Intractability

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

Another idea: $p_\theta(x) = \dfrac{\boxed{p_\theta(x \mid z) p_\theta(z)}}{p_\theta(z \mid x)}$ ← We know how to calculate these

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

# Variational Autoencoders: Intractability

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

Another idea: $p_\theta(x) = \dfrac{p_\theta(x \mid z) p_\theta(z)}{\boxed{p_\theta(z \mid x)}}$    <span style="color:red">But how do you calculate this?</span>

<span style="color:blue">**Solution**: In addition to modeling $p_\theta(x|z)$, learn $q_\phi(z|x)$ that approximates the true posterior $p_\theta(z|x)$.</span>

<span style="color:blue">Will see that the approximate posterior allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize.</span>

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

# Variational Autoencoders

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

# Variational Autoencoders

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

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

# Variational Autoencoders

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

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

# Variational Autoencoders

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

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

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

# Variational Autoencoders

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

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

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

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

# Variational Autoencoders

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

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

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

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

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

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

# Variational Autoencoders

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

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

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

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

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

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

# Variational Autoencoders

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

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

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

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

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

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!

# Variational Autoencoders

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

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

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

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

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

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

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

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

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

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

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

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

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

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

We want to maximize the data likelihood

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

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

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

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

# Variational Autoencoders

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

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

Decoder:
reconstruct
the input data

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

Encoder:
make approximate
posterior distribution
close to prior

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

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

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

# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

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

**Input Data** $\boxed{x}$

# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

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

$\mu_{z|x}$    $\Sigma_{z|x}$

Input Data    $x$

# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

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

Have analytical solution

Make approximate posterior distribution close to prior

Encoder network
$q_\phi(z|x)$
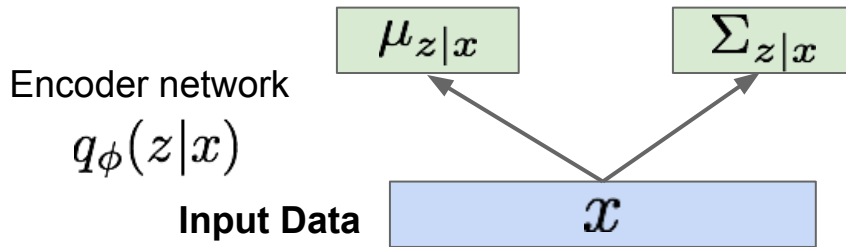
$\mu_{z|x}$    $\Sigma_{z|x}$

Input Data    $x$

# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

Not part of the computation graph!

$z$

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

Make approximate posterior distribution close to prior

$\mu_{z|x}$  $\Sigma_{z|x}$

Encoder network

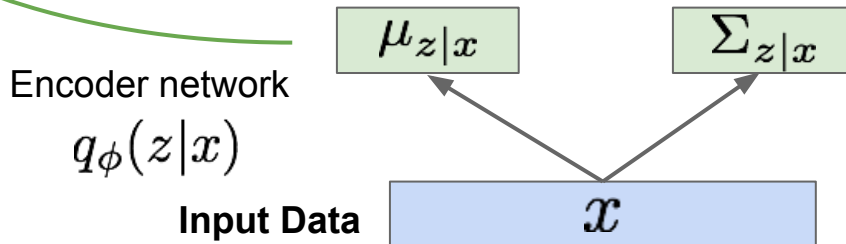$q_\phi(z|x)$

**Input Data**  $x$

# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

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

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

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

$z$

$\mu_{z|x}$ $\Sigma_{z|x}$

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

**Input Data** $x$

# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[ \log p_\theta(x^{(i)} \mid z) \right] - D_{KL}(q_\phi(z \mid x^{(i)}) \| p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$
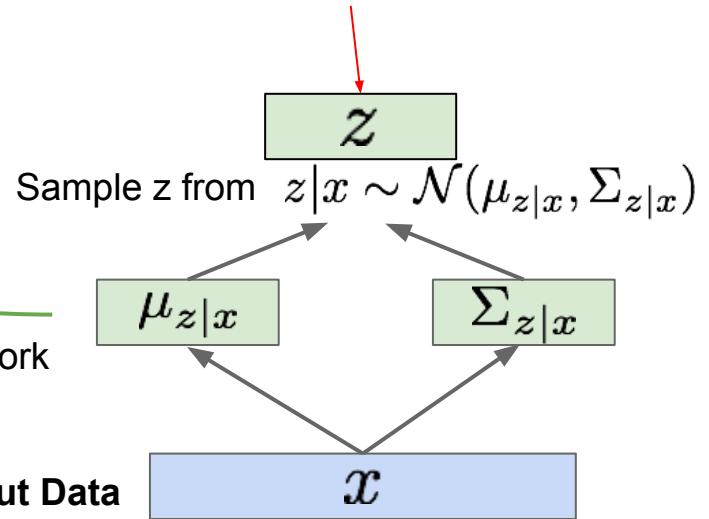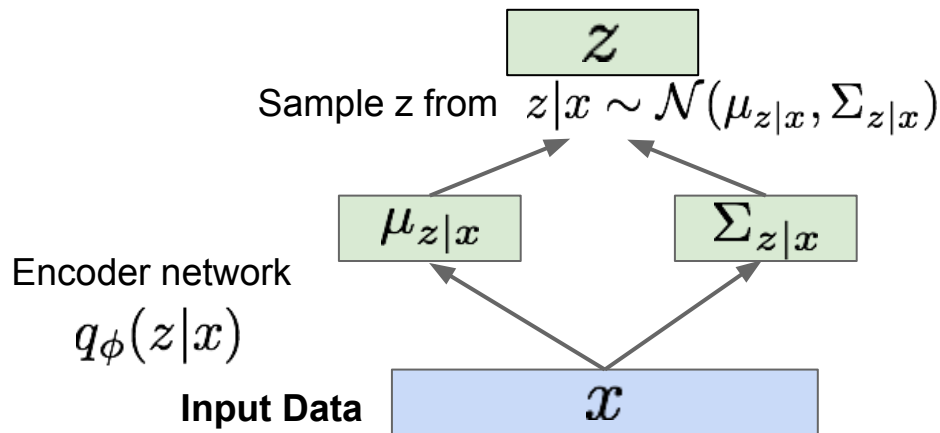
Sample $\epsilon \sim \mathcal{N}(0, I)$

Input to the graph

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

Part of computation graph

$z$

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

$\mu_{z|x}$          $\Sigma_{z|x}$

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

**Input Data**          $x$

# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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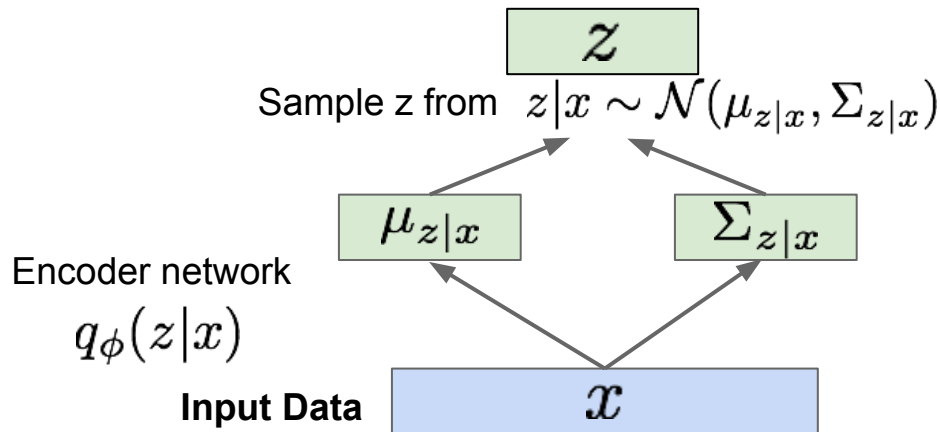
$\mu_{x|z}$ $\quad$ $\Sigma_{x|z}$

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

$z$

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

$\mu_{z|x}$ $\quad$ $\Sigma_{z|x}$

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

**Input Data** $\quad x$

# Variational Autoencoders

Maximize likelihood of original input being reconstructed

Putting it all together: maximizing the likelihood lower bound

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

$\hat{x}$

$\mu_{x|z}$     $\Sigma_{x|z}$

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

$z$

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

$\mu_{z|x}$     $\Sigma_{z|x}$

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

**Input Data**     $x$
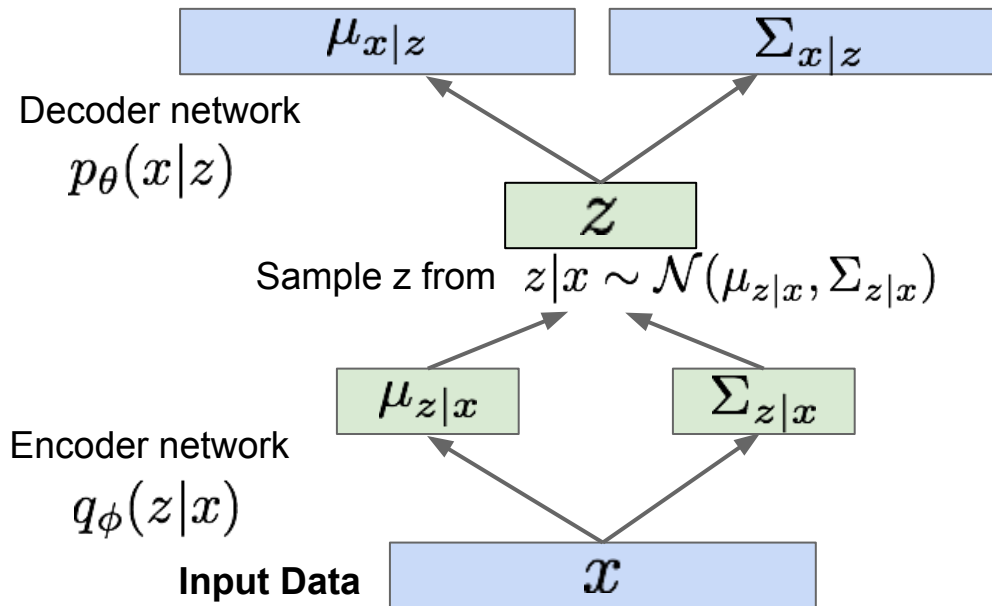
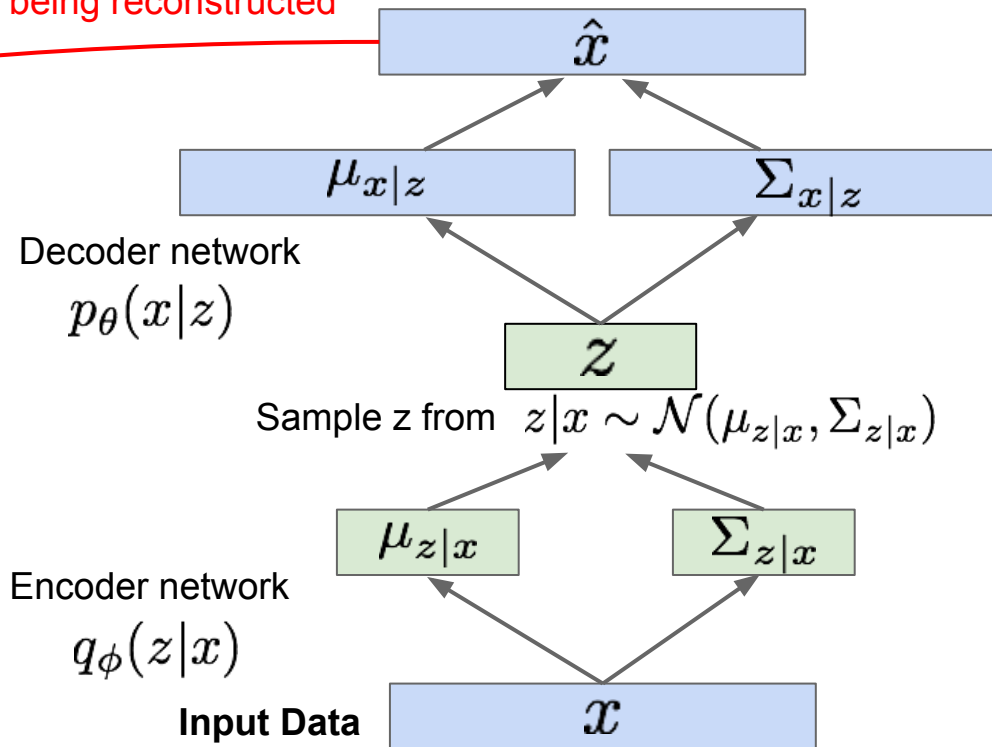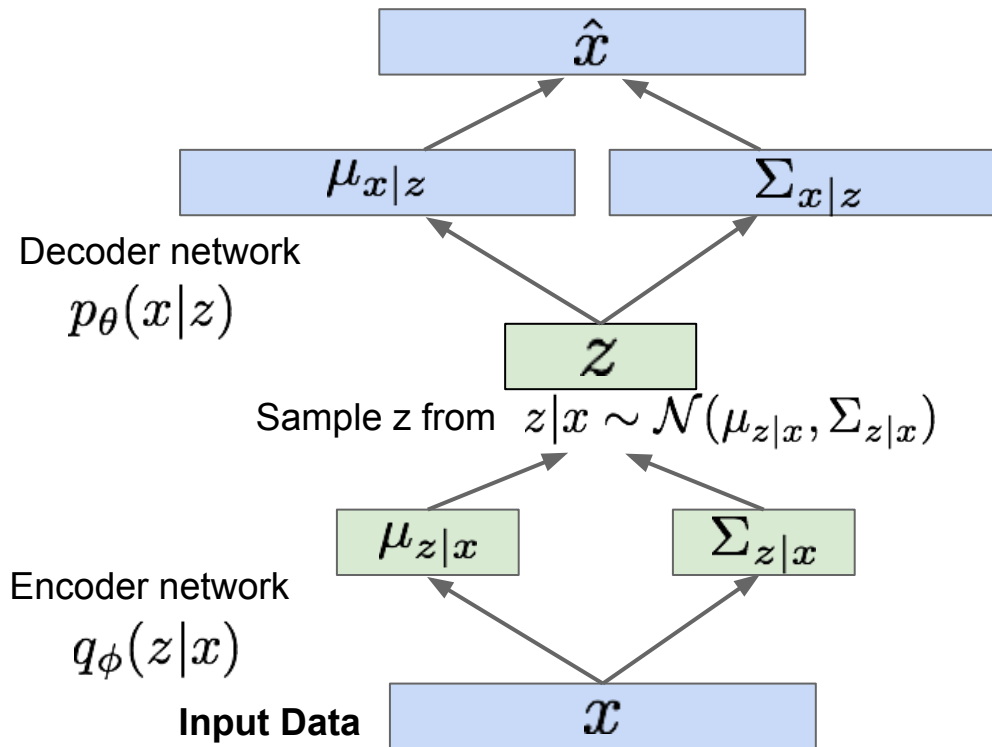# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

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

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

$\hat{x}$

$\mu_{x|z}$     $\Sigma_{x|z}$

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

$z$

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

$\mu_{z|x}$     $\Sigma_{z|x}$

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

**Input Data**     $x$
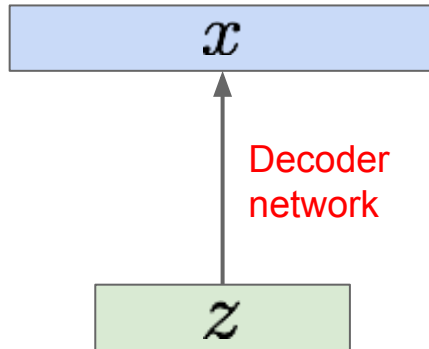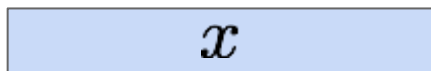
# Variational Autoencoders: Generating Data!

Our assumption about data generation process
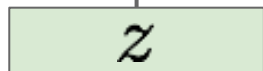
Sample from
true conditional
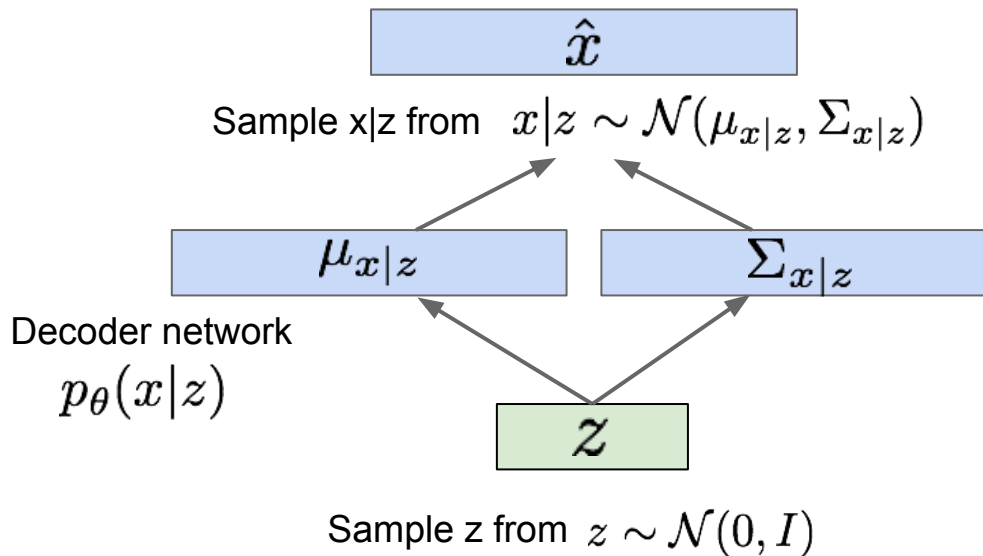
$p_{\theta^*}(x \mid z^{(i)})$

Sample from
true prior

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



$x$

Decoder
network

$z$

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

# Variational Autoencoders: Generating Data!

Our assumption about data generation process

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

Sample from true conditional

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

Decoder network

Sample from true prior

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

$\hat{x}$

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

$\mu_{x|z}$ $\qquad \Sigma_{x|z}$

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

$z$

Sample z from $\quad z \sim \mathcal{N}(0, I)$

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

# Variational Autoencoders: Generating Data!

Use decoder network.  Now sample z from prior!

$\hat{x}$

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

$\mu_{x|z}$ $\Sigma_{x|z}$

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

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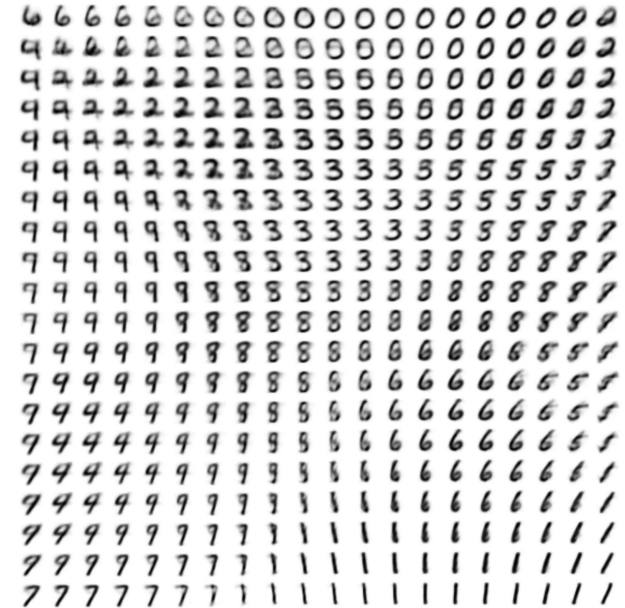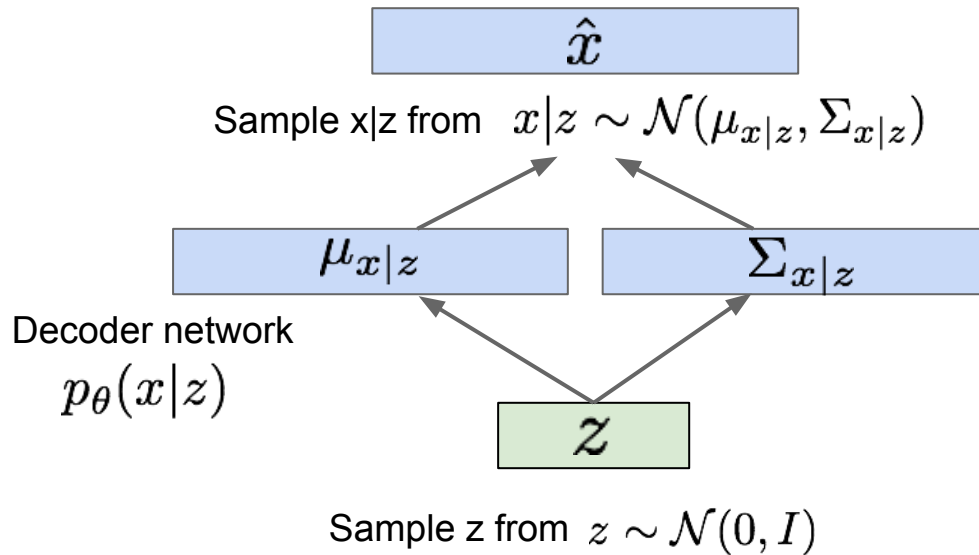
Sample z from $z \sim \mathcal{N}(0, I)$

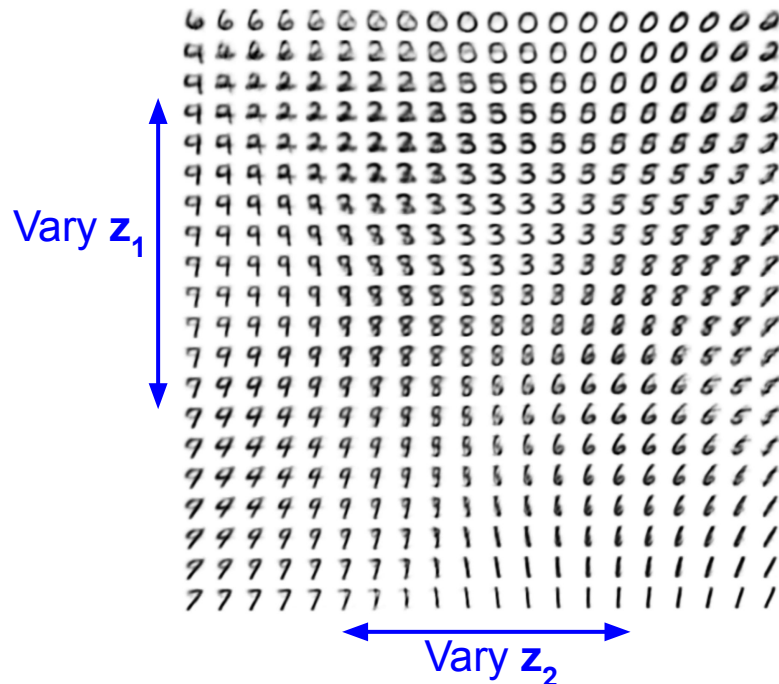Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders: Generating Data!

Use decoder network.  Now sample z from prior!

Data manifold for 2-d **z**



$\hat{x}$

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

Vary **z**$_1$

$\mu_{x|z}$   $\Sigma_{x|z}$

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

$z$

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

Vary **z**$_2$

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

# Variational Autoencoders: Generating Data!

Diagonal prior on **z** => independent latent variables

Different dimensions of **z** encode interpretable factors of variation

Degree of smile

Vary $z_1$

Vary $z_2$

Head pose

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

# Variational Autoencoders: Generating Data!

Diagonal prior on **z** => independent latent variables

Different dimensions of **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

# Variational Autoencoders

Probabilistic spin to traditional autoencoders => allows generating data
Defines an intractable density => derive and optimize a (variational) lower bound

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

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

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

# Comparing the two methods so far

Autoregressive model
- Directly maximize p(data)
- High-quality generated images
- Slow to generate images
- No explicit latent codes

Variational model
- Maximize lower bound on p(data)
- Generated images often blurry
- Very fast to generate images
- Learn rich latent codes

# Taxonomy of Generative Models

**Generative models**

Direct

GAN

Explicit density

Implicit density

Tractable density

Approximate density

Markov Chain

GSN

Fully Visible Belief Nets
- Autoregressive
- NADE
- MADE
- NICE / RealNVP
- Glow
- Ffjord

Variational

Markov Chain

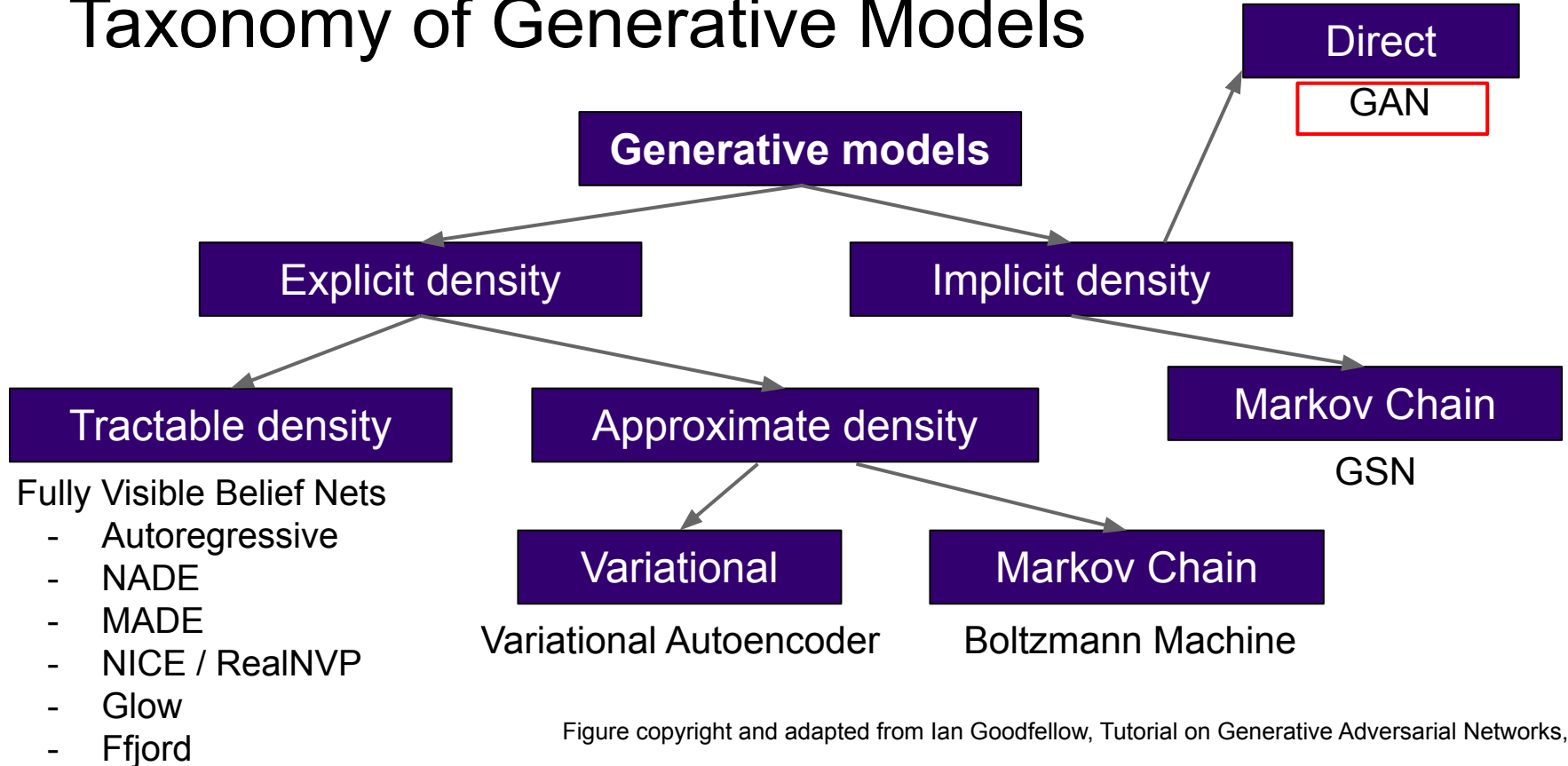Variational Autoencoder

Boltzmann Machine

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

# Generative Adversarial Networks (GANs)

# So far...

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

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

VAEs define intractable density function with latent **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:

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

VAEs define intractable density function with latent **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:

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

VAEs define intractable density function with latent **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

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.
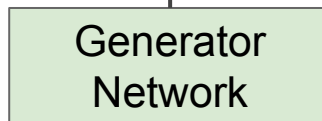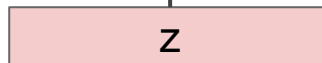
# Generative Adversarial Networks

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

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

Output: Sample from
training distribution

Generator
Network

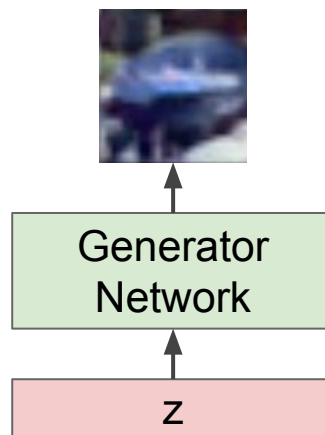Input: Random noise — z

# Generative Adversarial Networks

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



Generator Network

Input: Random noise

z

# Generative Adversarial Networks

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

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.
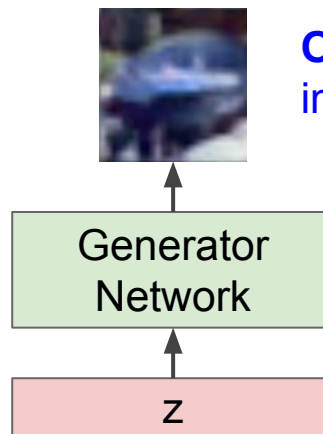
But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Output: Sample from training distribution

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

Generator Network

Input: Random noise

z

# Generative Adversarial Networks

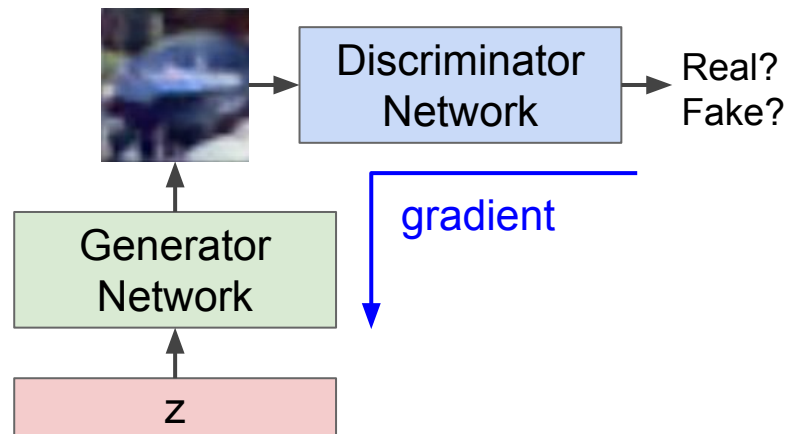Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

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

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

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

Output: Sample from training distribution



Discriminator Network

Real? Fake?

gradient

Generator Network

Input: Random noise

z

# Training GANs: Two-player game
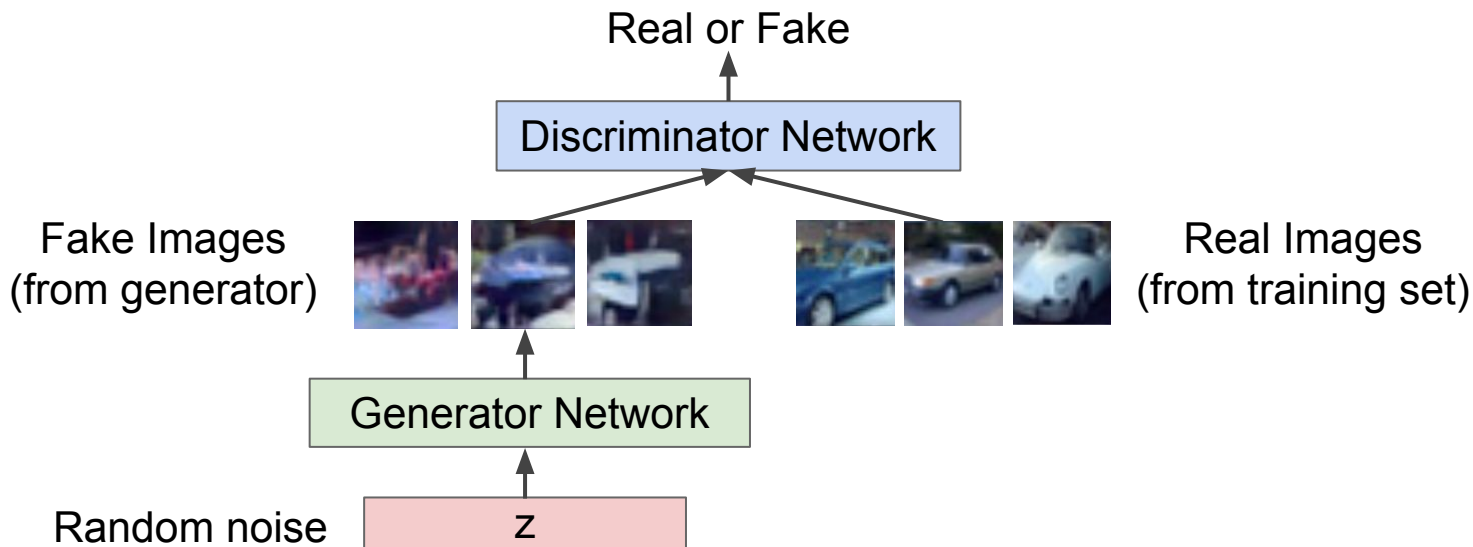
**Discriminator network**: try to distinguish between real and fake images
**Generator network**: try to fool the discriminator by generating real-looking images

# Training GANs: Two-player game

**Discriminator network**: try to distinguish between real and fake images
**Generator network**: try to fool the discriminator by generating real-looking images

Real or Fake

Discriminator Network

Fake Images
(from generator)

Real Images
(from training set)

Generator Network

Random noise          z

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

# Training GANs: Two-player game

**Discriminator network**: try to distinguish between real and fake images
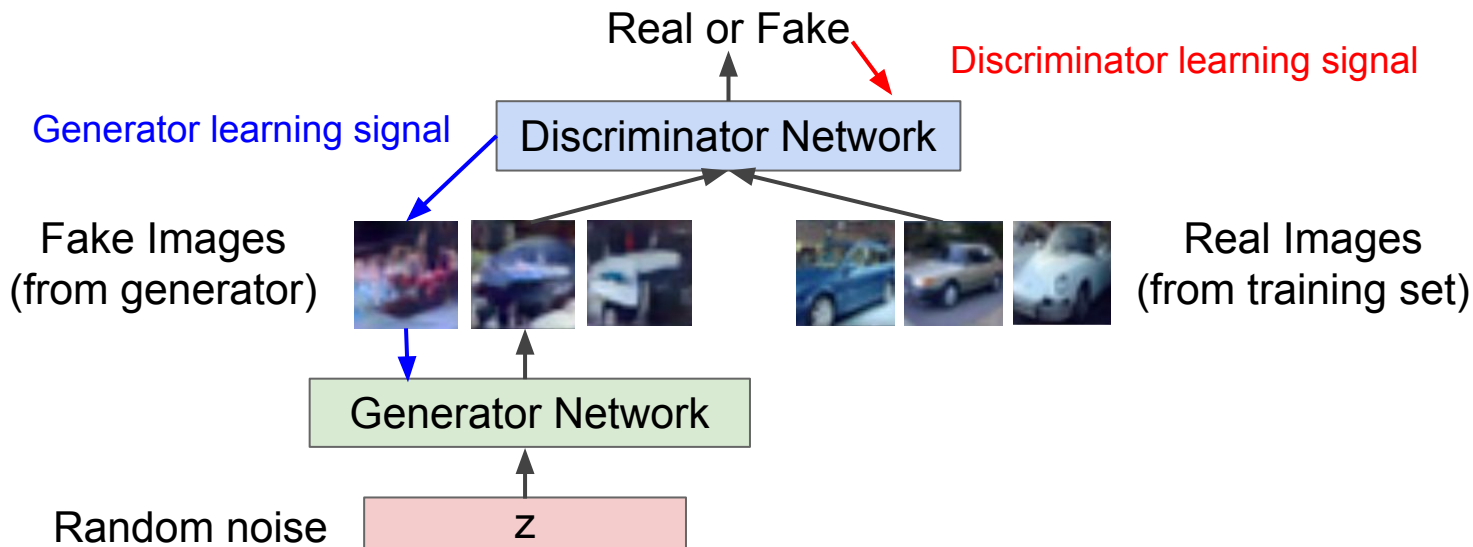**Generator network**: try to fool the discriminator by generating real-looking images



Real or Fake

Discriminator learning signal

Generator learning signal

Discriminator Network

Fake Images
(from generator)

Real Images
(from training set)

Generator Network

Random noise     z

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

# Training GANs: Two-player game

**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

**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

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

Discriminator output
for real data x

Discriminator output for
generated fake data G(z)
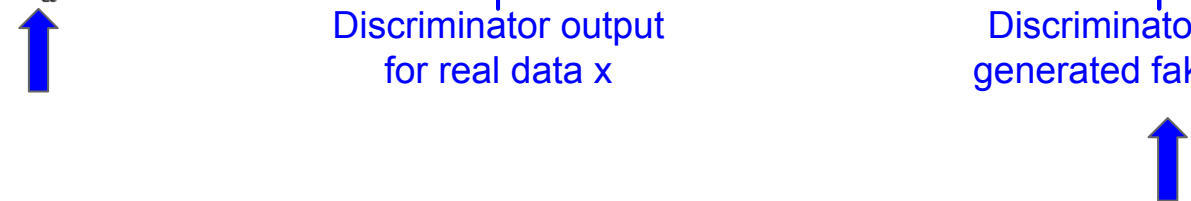
# Training GANs: Two-player game

**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**

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

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 output
for real data x

Discriminator output for
generated fake data G(z)

# Training GANs: Two-player game

**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**

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

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 output
for real data x

Discriminator output for
generated fake data G(z)

# Training GANs: Two-player game

**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**

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

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 output
for real data x

Discriminator output for
generated fake data G(z)

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

# Training GANs: Two-player game

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

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:

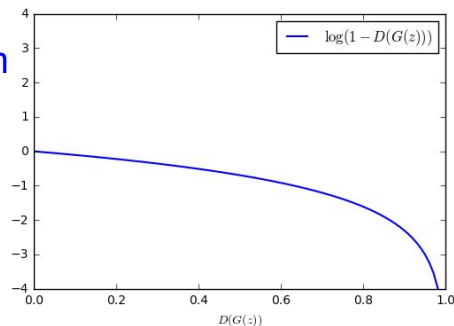1. **Gradient ascent** on discriminator

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

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!

# Training GANs: Two-player game

Minimax objective function:

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

Alternate between:

1.  **Gradient ascent** on discriminator

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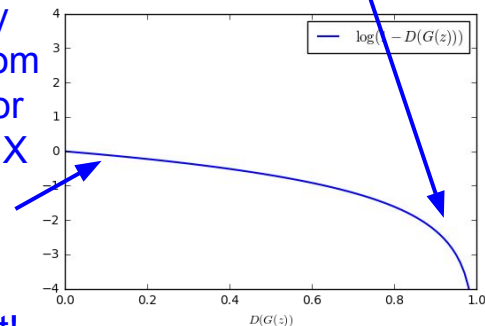
2.  **Gradient descent** on generator

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

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

Gradient signal dominated by region where sample is already good

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!

# Training GANs: Two-player game

Minimax objective function:

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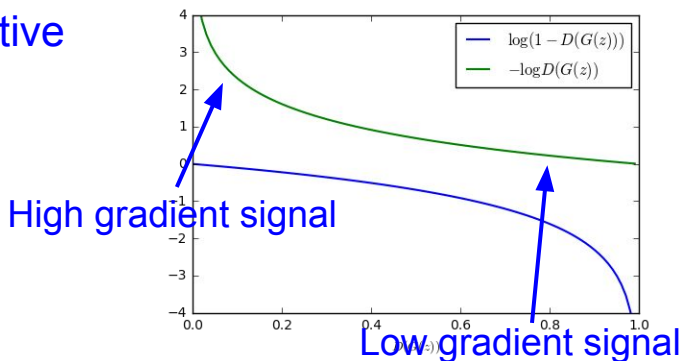
Alternate between:

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.



High gradient signal

Low gradient signal

# Training GANs: Two-player game

Putting it together: GAN training algorithm

**for** number of training iterations **do**
   **for** $k$ steps **do**
      • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
      • Sample minibatch of $m$ examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
      • 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 $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
• 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

Putting it together: GAN training algorithm

**for** number of training iterations **do**
  **for** $k$ steps **do**
  - Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  - Sample minibatch of $m$ examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
  - 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 $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Update the generator by ascending its stochastic gradient (improved objective):

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

**end for**

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

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

Arjovsky et al. "Wasserstein gan." arXiv preprint arXiv:1701.07875 (2017)
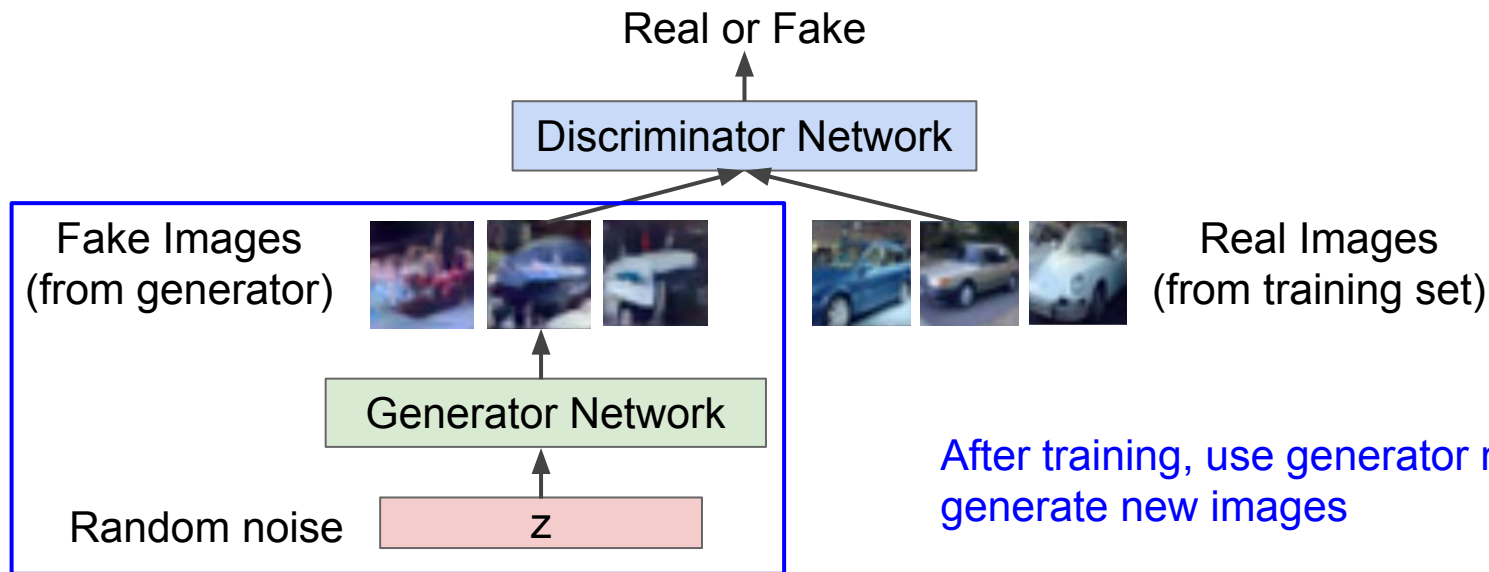Berthelot, et al. "Began: Boundary equilibrium generative adversarial networks." arXiv preprint arXiv:1703.10717 (2017)

# Training GANs: Two-player game

**Generator network**: try to fool the discriminator by generating real-looking images
**Discriminator network**: try to distinguish between real and fake images



After training, use generator network to generate new images

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

# Generative Adversarial Nets

## Generated samples



Nearest neighbor from training set

# Generative Adversarial Nets

## Generated samples (CIFAR-10)



Nearest neighbor from training set

# 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!
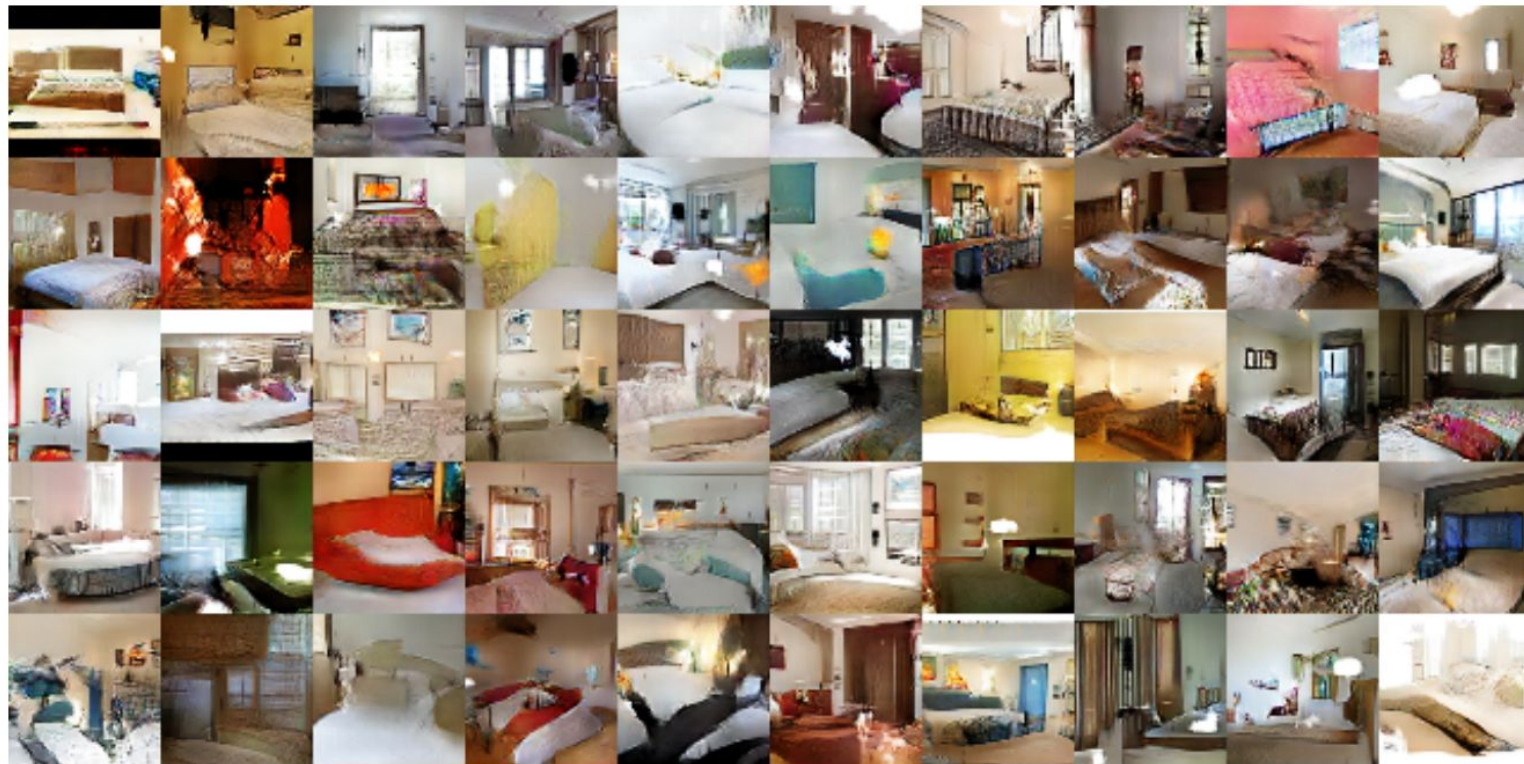


Radford et al, ICLR 2016

# Generative Adversarial Nets: Convolutional Architectures

Interpolating
between
random
points in latent
space

Radford et al,
 ICLR 2016

# Generative Adversarial Nets: Interpretable Vector Math

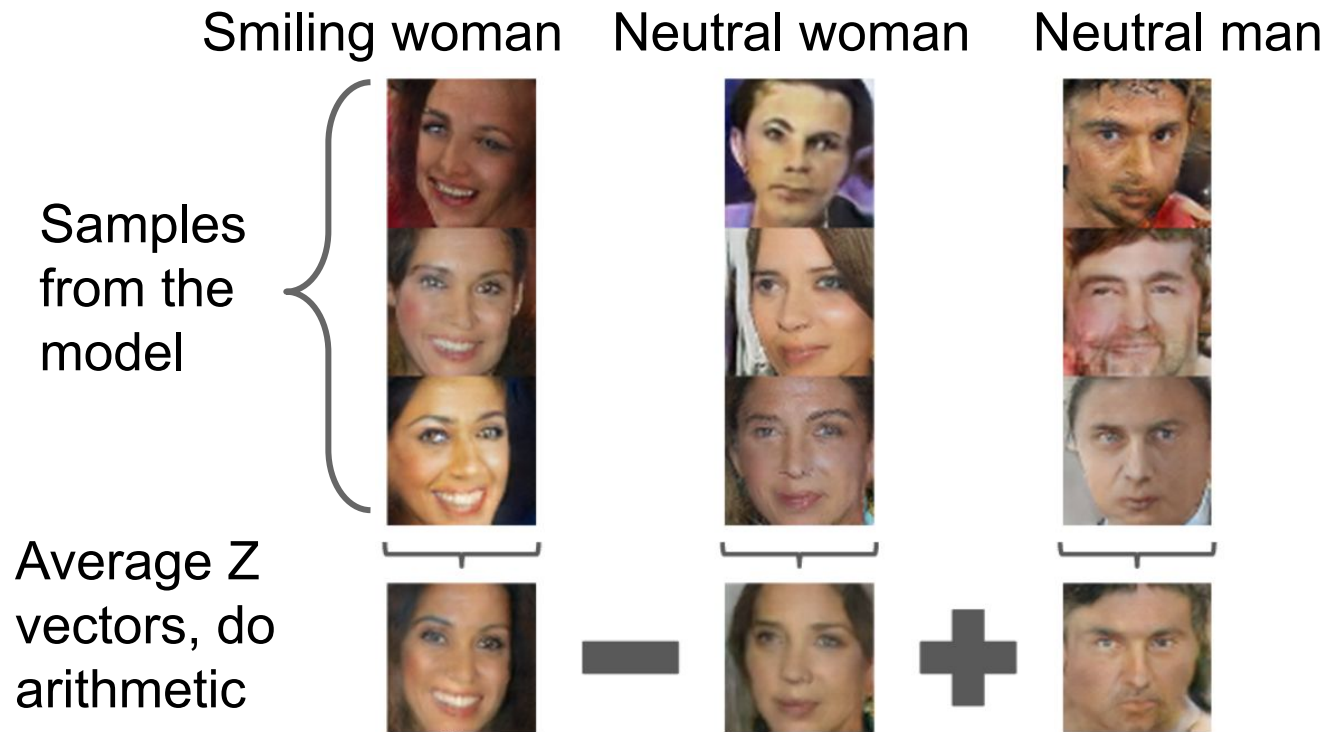Radford et al, ICLR 2016

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

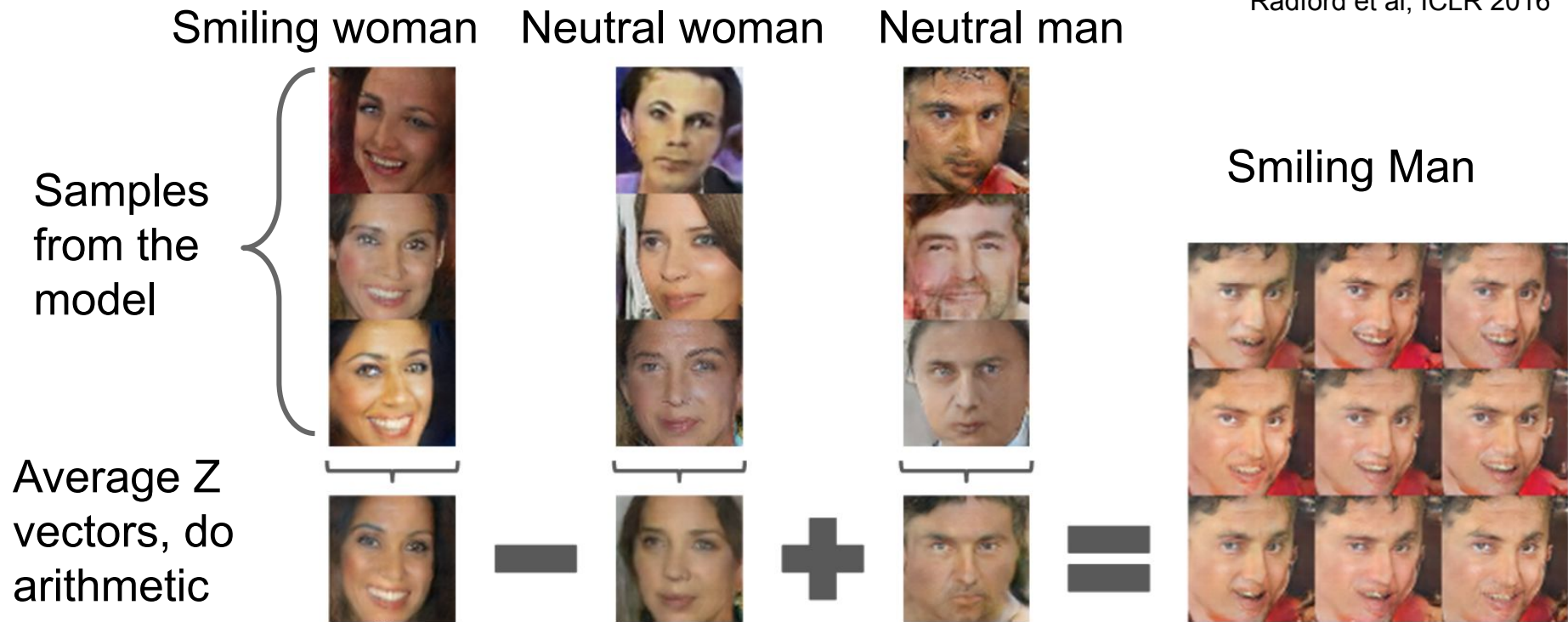Smiling woman     Neutral woman     Neutral man

Samples from the model

Smiling Man

Average Z vectors, do arithmetic

# Generative Adversarial Nets: Interpretable Vector Math

Glasses man          No glasses man          No glasses woman          Radford et al, ICLR 2016

Woman with glasses

# Since then: 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 Categorial GANs
- b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
- Bayesian GAN - Deep and Hierarchical Implicit Models
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BiGAN - Adversarial Feature Learning
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- CGAN - Conditional Generative Adversarial Nets
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
- CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CoGAN - Coupled Generative Adversarial Networks

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

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

# 2017: Explosion of GANs

Better training and generation
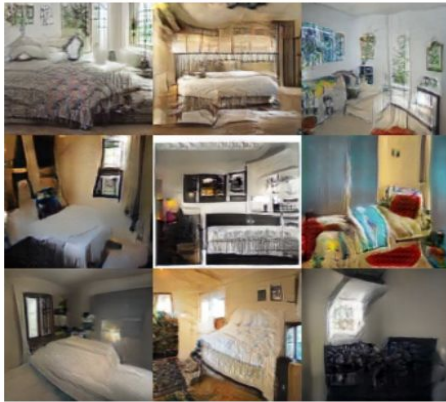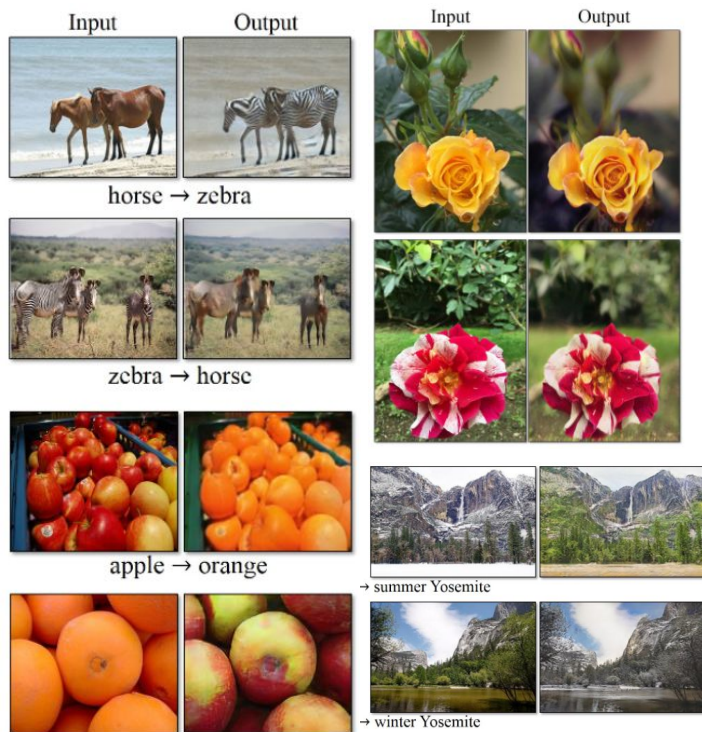


LSGAN, Zhu 2017.



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





Progressive GAN, Karras 2018.

# 2017: Explosion of GANs

Source->Target domain transfer



CycleGAN. Zhu et al. 2017.

Text -> Image Synthesis



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

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

Reed et al. 2017.

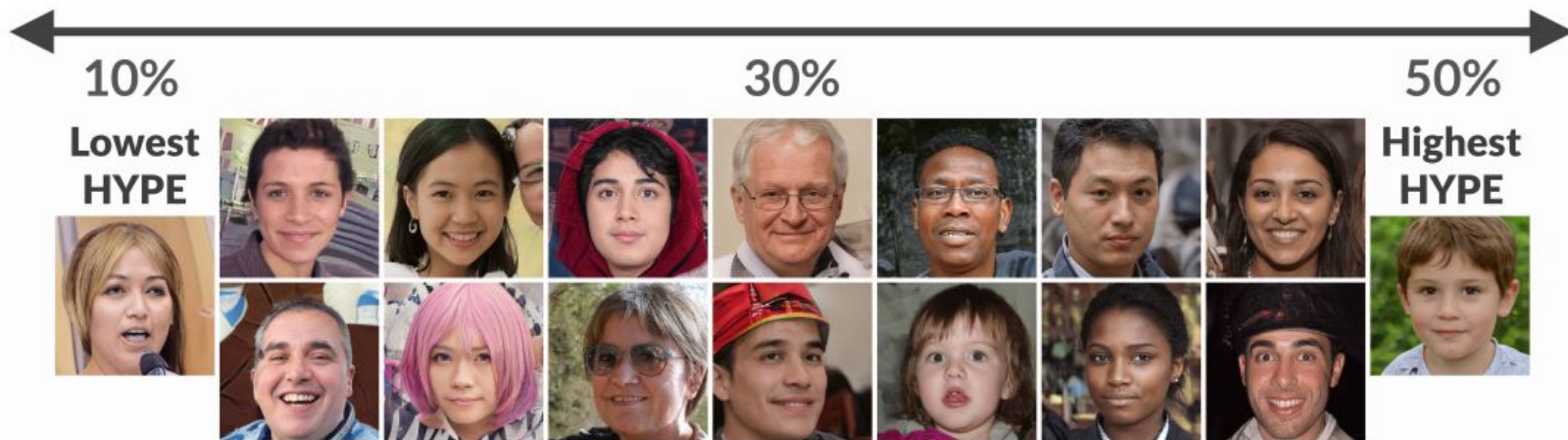Many GAN applications



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

# 2019: BigGAN



Brock et al., 2019

# 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
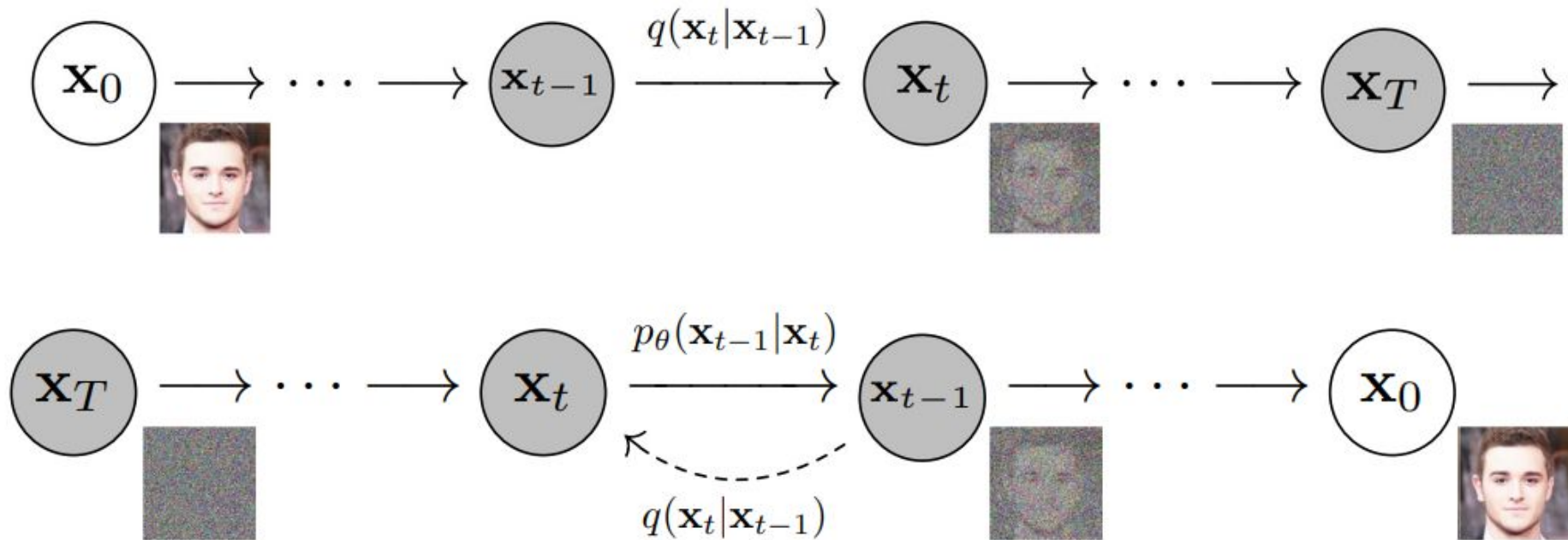
Pros:
- Beautiful, state-of-the-art samples!

Cons:
- Trickier / more unstable to train
- Can't solve inference queries such as $p(x)$, $p(z|x)$
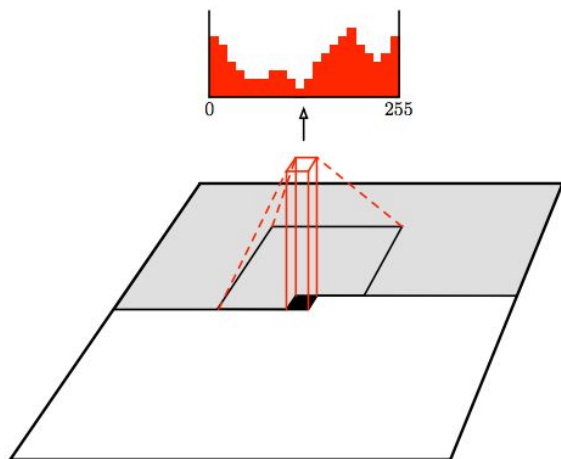
Active areas of research:
- Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
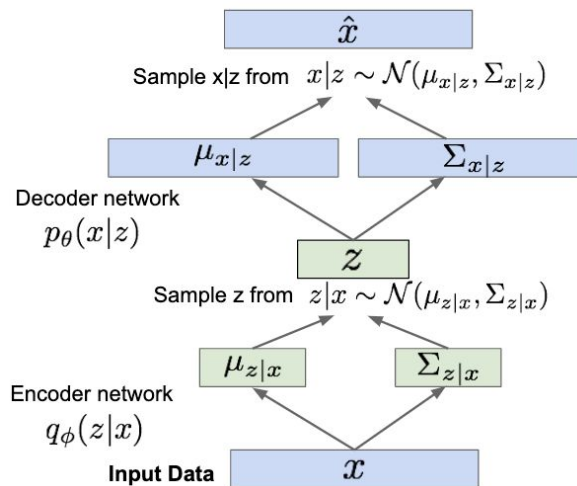- Conditional GANs, GANs for all kinds of applications

# Diffusion Models

# Summary

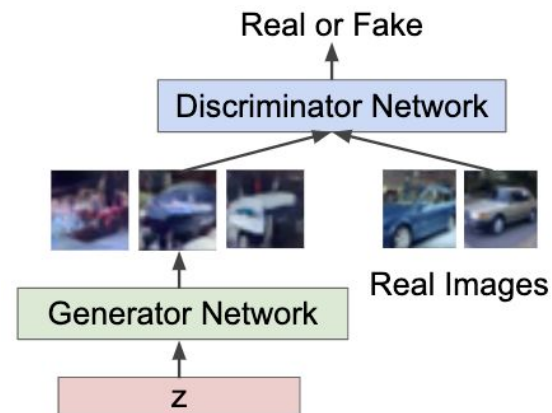**Autoregressive models:**
PixelRNN, PixelCNN



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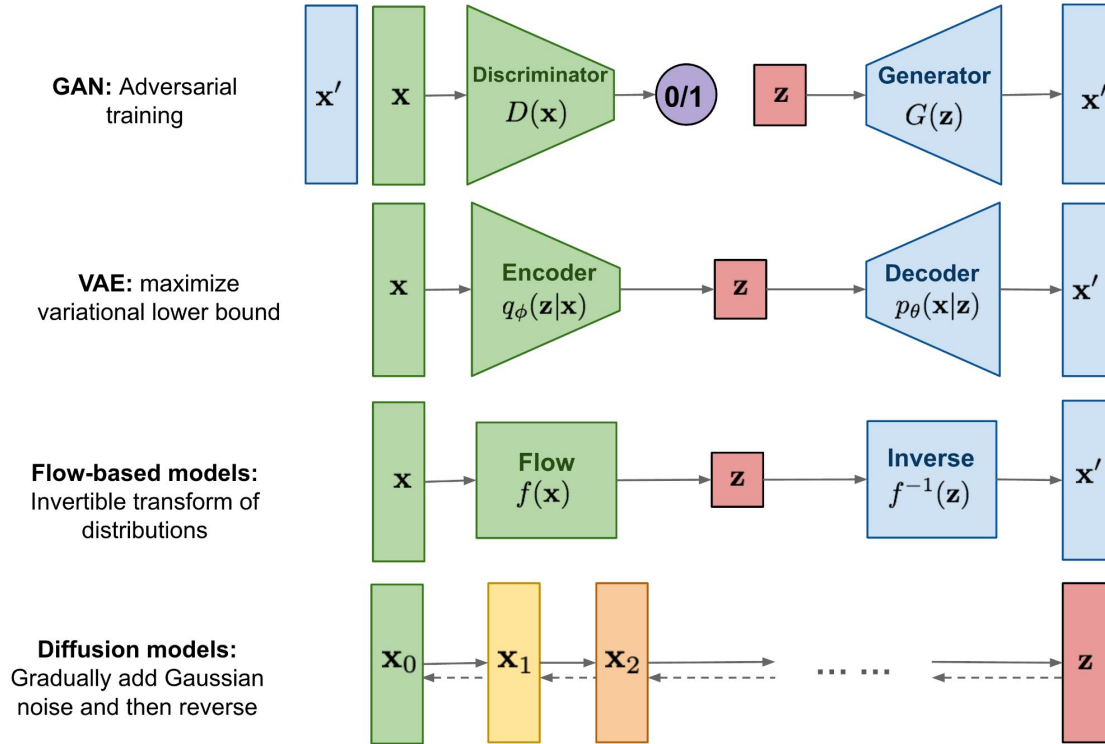
**Variational Autoencoders**



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

**Generative Adversarial Networks (GANs)**



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

# Summary



**GAN:** Adversarial training

**VAE:** maximize variational lower bound

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

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

Next: Reinforcement Learning