Lecture 12: Generative Models
Administrative

Project Milestone due tomorrow (Wed 5/16)

A3 due next Wed, 5/23
Overview

- Unsupervised Learning
- Generative Models
  - PixelRNN and PixelCNN
  - Variational Autoencoders (VAE)
  - Generative Adversarial Networks (GAN)
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.
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.

Object Detection

DOG, DOG, CAT
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 \(x \rightarrow y\)

**Examples:** Classification, regression, object detection, semantic segmentation, image captioning, 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, feature learning, density estimation, etc.
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, feature learning, density estimation, etc.

Principal Component Analysis (Dimensionality reduction)

This image from Matthias Scholz 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, feature learning, density estimation, etc.

Autoencoders (Feature learning)

L2 Loss function:
$$||x - \hat{x}||^2$$
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.

1-d density estimation

2-d density estimation

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.
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, feature learning, density estimation, 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.

**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.

Training data is cheap

Holy grail: Solve unsupervised learning => understand structure of visual world
Generative Models

Given training data, generate new samples from same distribution

Training data \sim p_{data}(x) \quad \text{Generated samples} \sim p_{model}(x)

Want to learn \( p_{model}(x) \) similar to \( p_{data}(x) \)
Generative Models

Given training data, generate new samples from same distribution

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

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

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

Addresses density estimation, a core problem in unsupervised learning

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

- Realistic samples for artwork, super-resolution, colorization, etc.
- Generative models of time-series data can be used for simulation and planning (reinforcement learning applications!)
- Training generative models can also enable inference of latent representations that can be useful as general features
Taxonomy of Generative Models

Generative models

- Explicit density
  - Tractable density
    - Fully Visible Belief Nets
      - NADE
      - MADE
      - PixelRNN/CNN
    - Change of variables models (nonlinear ICA)
- Implicit density
  - Approximate density
    - Variational Autoencoder
  - Markov Chain
    - Variational
    - Markov Chain
      - Boltzmann Machine
      - GSN
      - GAN

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

Generative models

 Explicit density

 Tractable density

 Fully Visible Belief Nets
 - NADE
 - MADE
 - PixelRNN/CNN

 Approximate density

 Variational

 Variational Autoencoder

 Implicit density

 Markov Chain

 Markov Chain

 Direct

 GAN

 Change of variables models (nonlinear ICA)

Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.
PixelRNN and PixelCNN
Fully visible belief network

Explicit density model

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

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

Then maximize likelihood of training data
**Fully visible belief network**

Explicit density model

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

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

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

Then maximize likelihood of training data
Fully visible belief network

Explicit density model

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

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

Likelihood of image $x$  
Probability of $i$'th pixel value given all previous pixels  
Complex distribution over pixel values $\Rightarrow$ Express using a neural network!

Will need to define ordering of “previous pixels”

Then maximize likelihood of training data
PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

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

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)
PixelRNN \cite{van_der_oord_2016}

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)
PixelRNN \cite{van_der_Oord_etal_2016}

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow!
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
PixelCNN [van der Oord et al. 2016]

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region

Training: maximize likelihood of training images

$$p(x) = \prod_{i=1}^{n} p(x_i|x_1, \ldots, x_{i-1})$$
PixelCNN [van der Oord et al. 2016]

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region

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

Generation must still proceed sequentially => still slow

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

32x32 CIFAR-10

32x32 ImageNet

Figures copyright Aaron van der Oord et al., 2016. Reproduced with permission.
**PixelRNN and PixelCNN**

**Pros:**
- Can explicitly compute likelihood $p(x)$
- Explicit likelihood of training data gives good evaluation metric
- Good samples

**Con:**
- Sequential generation => slow

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

See
- Van der Oord et al. NIPS 2016
- Salimans et al. 2017 (PixelCNN++)
Variational Autoencoders (VAE)
So far...

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

\[ p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, \ldots, x_{i-1}) \]
So far...

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

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

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
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

**Originally**: Linear + nonlinearity (sigmoid)
**Later**: Deep, fully-connected
**Later**: ReLU CNN
Some background first: Autoencoders

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

- \( z \) usually smaller than \( x \) (dimensionality reduction)

Q: Why dimensionality reduction?

**Originally:** Linear + nonlinearity (sigmoid)

**Later:** Deep, fully-connected

**Later:** ReLU CNN

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?

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

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

How to learn this feature representation?
Some background first: Autoencoders

How to learn this feature representation?
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding itself

![Autoencoder Diagram]

- Input data: $x$
- Features: $z$
- Reconstructed input data: $\hat{x}$
- Encoder
- Decoder
Some background first: Autoencoders

How to learn this feature representation?
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding itself

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN (upconv)
Some background first: Autoencoders

How to learn this feature representation?
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding itself

Encoder
Input data $x$
Features $z$
Decoder
Reconstructed input data $\hat{x}$

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

Reconstructed data

Input data

Fei-Fei Li & Justin Johnson & Serena Yeung
Lecture 12 - 44 May 15, 2018
Some background first: Autoencoders

Train such that features can be used to reconstruct original data

L2 Loss function:

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

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

Input data

Features

Reconstructed input data

Reconstructed data

Encoder

Decoder
Some background first: Autoencoders

Train such that features can be used to reconstruct original data

L2 Loss function:
$$\| x - \hat{x} \|^2$$

Doesn’t use labels!

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

Reconstructed data
Some background first: Autoencoders

![Autoencoder Diagram]

Reconstructed input data

Decoder

Features

Encoder

Input data

After training, throw away decoder

Encoder (\( \mathbf{x} \)) → Features (\( \mathbf{z} \)) → Decoder (\( \hat{\mathbf{x}} \))
Some background first: Autoencoders

Encoder can be used to initialize a supervised model.

Input data $x$ → Features $z$ → Encoder → Predicted Label $\hat{y}$ → Loss function (Softmax, etc) → Classifier $y$ → Fine-tune encoder jointly with classifier → Train for final task (sometimes with small data).
Some background first: Autoencoders

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

Features capture factors of variation in training data. Can we generate new images from an autoencoder?
Variational Autoencoders

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

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

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

Sample from true conditional

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

Sample from true prior

\[
p_{\theta^*}(z)
\]

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

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

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

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

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

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

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

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

Sample from true conditional

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

Sample from true prior

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

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

We want to estimate the true parameters $\theta^*$ of this generative model. How should we represent this model?

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

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

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

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

How should we represent this model?

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

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

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

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

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

How should we represent this model?

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

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

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

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

Decoder network

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

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

How to train the model?

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

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

Decoder network

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

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

How to train the model?

Remember strategy for training generative models from FVBNs. 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 $p_{\theta^*}(z)$

Decoder network

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

How to train the model?

Remember strategy for training generative models from FVBNs. Learn model parameters to maximize likelihood of training data

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

Now with latent $z$

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

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

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

We want to estimate the true parameters \( \theta^* \) of this generative model.

How to train the model?

Remember strategy for training generative models from FVBNs. Learn model parameters to maximize likelihood of training data

\[ p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x \mid z)dz \]

Q: What is the problem with this?

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

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

How to train the model?

Remember strategy for training generative models from FVBNs. 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: \( p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz \)
Variational Autoencoders: Intractability

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

Simple Gaussian prior
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 \( z \)!
Variational Autoencoders: Intractability

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

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

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

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

Posterior density also intractable: \[ p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)} \]

Intractable data likelihood

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

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

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

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

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

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

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

Mean and (diagonal) covariance of $z \mid x$

Encoder network
$q_{\phi}(z \mid x)$
(parameters $\phi$)

$x$

Mean and (diagonal) covariance of $x \mid z$

Decoder network
$p_{\theta}(x \mid z)$
(parameters $\theta$)

$z$

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

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

Sample \( z \) from \( z|\mathbf{x} \sim \mathcal{N}(\mu_z|\mathbf{x}, \Sigma_z|\mathbf{x}) \)

Encoder network

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

(parameters \( \phi \))

\( \mathcal{X} \)

Decoder network

\( p_\theta(\mathbf{x}|z) \)

(parameters \( \theta \))

\( \mathcal{Z} \)

Sample \( \mathbf{x} \) from \( \mathbf{x}|z \sim \mathcal{N}(\mu_x|z, \Sigma_x|z) \)

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

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

Sample $z$ from $z | x \sim \mathcal{N}(\mu_z | x, \Sigma_z | x)$

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

$x$ (data)

Sample $x$|$z$ from $x | z \sim \mathcal{N}(\mu_x | z, \Sigma_x | z)$

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

$z$ (latent variables)

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

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

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

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

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

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

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

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

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

\[
= \mathbb{E}_z \left[ \log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad (\text{Bayes’ Rule})
\]
Variational Autoencoders

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\[
= E_z \left[ \log \frac{p_\theta(x^{(i)} | z) p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad \text{(Bayes’ Rule)}
\]

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

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

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

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

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

\[
\log p_θ(x^{(i)}) = E_{z \sim q_ϕ(z|x^{(i)})} \left[ \log p_θ(x^{(i)}) \right] \quad (p_θ(x^{(i)}) \text{ Does not depend on } z)
\]

\[
= E_z \left[ \log \frac{p_θ(x^{(i)} | z)p_θ(z)}{p_θ(z | x^{(i)})} \right] \quad (\text{Bayes’ Rule})
\]

\[
= E_z \left[ \log \frac{p_θ(x^{(i)} | z)p_θ(z)}{p_θ(z | x^{(i)})} \frac{q_ϕ(z | x^{(i)})}{q_ϕ(z | x^{(i)})} \right] \quad (\text{Multiply by constant})
\]

\[
= E_z \left[ \log p_θ(x^{(i)} | z) \right] - E_z \left[ \log \frac{q_ϕ(z | x^{(i)})}{p_θ(z)} \right] + E_z \left[ \log \frac{q_ϕ(z | x^{(i)})}{p_θ(z | x^{(i)})} \right] \quad (\text{Logarithms})
\]

\[
= E_z \left[ \log p_θ(x^{(i)} | z) \right] - D_{KL}(q_ϕ(z | x^{(i)}) \parallel p_θ(z)) + D_{KL}(q_ϕ(z | x^{(i)}) \parallel p_θ(z | x^{(i)}))
\]

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

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

\( p_θ(z|x) \) intractable (saw earlier), can’t compute this KL term :(

But we know KL divergence always \( \geq 0 \).
Variational Autoencoders

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

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

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

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

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

\[
= \mathbb{E}_z \left[ \log p_\theta(x^{(i)} \mid z) \right] - D_{KL}(q_\phi(z \mid x^{(i)}) || p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) || p_\theta(z \mid x^{(i)})) + \mathbb{E}_z \left[ \log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})} \right] \geq 0
\]

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

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

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

\[
= \mathbb{E}_{z} \left[ \log \frac{p_{\theta}(x^{(i)} \mid z)p_{\theta}(z)}{p_{\theta}(z \mid x^{(i)})} \right] \quad \text{(Bayes’ Rule)}
\]

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

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

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

\[
\mathcal{L}(x^{(i)}, \theta, \phi) > 0
\]

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

Variational lower bound (“ELBO”)

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

Training: Maximize lower bound
Variational Autoencoders

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

\[
\log p_{\theta}(x^{(i)}) = \mathbb{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] \quad \text{(} p_{\theta}(x^{(i)}) \text{ Does not depend on } z) \\
= \mathbb{E}_{z} \left[ \log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad \text{(Bayes’ Rule)} \\
= \mathbb{E}_{z} \left[ \log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} q_{\phi}(z | x^{(i)}) \right] \quad \text{(Multiply by constant)} \\
= \mathbb{E}_{z} \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbb{E}_{z} \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbb{E}_{z} \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad \text{(Logarithms)} \\
= \mathbb{E}_{z} \left[ \log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) \| p_{\theta}(z)) + D_{KL}(q_{\phi}(z | x^{(i)}) \| p_{\theta}(z | x^{(i)})) \\
> 0
\]

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

Variational lower bound (“ELBO”)

Training: Maximize lower bound

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

Putting it all together: maximizing the likelihood lower bound

$$L(x^{(i)}, \theta, \phi) = \mathbb{E}_z \left[ \log p_{\theta}(x^{(i)} \mid z) \right] - D_{KL}(q_{\phi}(z \mid x^{(i)}) \mid \mid p_{\theta}(z))$$
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

\[ E_z \left[ \log p_{\theta}(x^{(i)} | z) \right] \quad \text{for a given minibatch of input data} \]

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

Putting it all together: maximizing the likelihood lower bound

\[
E_Z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) \parallel p_\theta(z))
\]

\[
\mathcal{L}(x^{(i)}, \theta, \phi)
\]
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

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

Make approximate posterior distribution close to prior

Encoder network

\[ q_{\phi}(z | x) \]

Input Data

\[ \mathcal{X} \]
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

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

Make approximate posterior distribution close to prior

Sample $z$ from $z | x \sim \mathcal{N}(\mu_z | x, \Sigma_z | x)$
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

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

- Encoder network
  - Sample $z$ from $z | x \sim \mathcal{N}(\mu_z | x, \Sigma_z | x)$
- Decoder network
  - $p_{\theta}(x | z)$
- Make approximate posterior distribution close to prior

Fei-Fei Li & Justin Johnson & Serena Yeung
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

Maximize likelihood of original input being reconstructed

\[ x | z \sim \mathcal{N}(\mu_x | z, \Sigma_x | z) \]

Sample z from \( z | x \sim \mathcal{N}(\mu_z | x, \Sigma_z | x) \)

Sample x|z from \( x | z \sim \mathcal{N}(\mu_x | z, \Sigma_x | z) \)

Make approximate posterior distribution close to prior

Encoder network
\[ q_\phi(z | x) \]

Decoder network
\[ p_\theta(x | z) \]
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

Maximize likelihood of original input being reconstructed

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

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

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

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

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!

$$\mu_{x|z}$$

Sample $x|z$ from

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

Decoder network

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

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 $z$

Sample $x|z$ from $x|z \sim \mathcal{N}(\mu_x|z, \Sigma_x|z)$

Decoder network

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

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

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 \( z_1 \)

Head pose
Vary \( z_2 \)

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014
Variational Autoencoders: Generating Data!

Diagonal prior on $z$
$\Rightarrow$ independent latent variables

Different dimensions of $z$
encode interpretable factors of variation

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

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
- Allows inference of $q(z|x)$, can be useful feature representation for other tasks

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

Active areas of research:
- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian
- Incorporating structure in latent variables
Generative Adversarial Networks (GAN)
So far...

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

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

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:

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

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: don’t work with any explicit density function!
Instead, take game-theoretic approach: learn to generate from training distribution through 2-player game
Generative Adversarial Networks

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

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

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

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

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

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

A: A neural network!

Output: Sample from training distribution

Input: Random noise

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

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

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

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

Train jointly in **minimax game**

Minimax objective function:

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

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

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

Train jointly in **minimax game**

Minimax objective function:

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

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

- Discriminator output for real data \( x \)
- Discriminator output for generated fake data \( G(z) \)

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

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 \((\theta_d)\) 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)

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014
Training GANs: Two-player game

Minimax objective function:

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

Alternate between:

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

2. **Gradient descent** on generator
   \[
   \min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z)))
   \]

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

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

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good
Training GANs: Two-player game

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

Alternate between:

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

2. **Instead**: **Gradient ascent** on generator, different objective
   \[
   \max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))
   \]

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.
Training GANs: Two-player game

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 \(\Rightarrow\) works much better! Standard in practice.

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

Putting it together: GAN training algorithm

\begin{equation}
\text{for number of training iterations do}
\begin{align*}
\quad & \text{for } k \text{ steps do} \\
\quad & \quad \bullet \text{ Sample minibatch of } m \text{ noise samples } \{z^{(1)}, \ldots, z^{(m)}\} \text{ from noise prior } p_g(z). \\
\quad & \quad \bullet \text{ Sample minibatch of } m \text{ examples } \{x^{(1)}, \ldots, x^{(m)}\} \text{ from data generating distribution } p_{data}(x). \\
\quad & \quad \bullet \text{ Update the discriminator by ascending its stochastic gradient:} \\
\quad & \quad \qquad \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] \\
\quad & \text{end for} \\
\quad & \bullet \text{ Sample minibatch of } m \text{ noise samples } \{z^{(1)}, \ldots, z^{(m)}\} \text{ from noise prior } p_g(z). \\
\quad & \bullet \text{ Update the generator by ascending its stochastic gradient (improved objective):} \\
\quad & \qquad \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \\
\text{end for}
\end{align*}
\end{equation}
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 $p_{data}(x)$.
        • 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.

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

---

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

Generated samples

Nearest neighbor from training set

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

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.

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 12 - May 15, 2018

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

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

Architecture guidelines for stable Deep Convolutional GANs

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

Generative Adversarial Nets: Convolutional Architectures

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

Smiling woman  Neutral woman  Neutral man

Samples from the model

Radford et al, ICLR 2016
Generative Adversarial Nets: Interpretable Vector Math

Samples from the model

Smiling woman
Neutral woman
Neutral man

Average Z vectors, do arithmetic

Radford et al, ICLR 2016
Generative Adversarial Nets: Interpretable Vector Math

Samples from the model

Average Z vectors, do arithmetic

Smiling woman
Neutral woman
Neutral man

Radford et al, ICLR 2016
Generative Adversarial Nets: Interpretable Vector Math

Glasses man  No glasses man  No glasses woman

Radford et al, ICLR 2016
Generative Adversarial Nets: Interpretable Vector Math

Glasses man  No glasses man  No glasses woman

Woman with glasses

Radford et al, ICLR 2016
2017: Explosion of GANs

“The GAN Zoo”

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

https://github.com/hindupuravinash/the-gan-zoo
2017: Explosion of GANs

"The GAN Zoo"

| GAN - Generative Adversarial Networks |
| 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling |
| acGAN - Face Aging With Conditional Generative Adversarial Networks |
| AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs |
| AdaGAN - AdaGAN: Boosting Generative Models |
| AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets |
| AffGAN - Amortised MAP Inference for Image Super-resolution |
| AL-GAN - 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 Generative 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 |
| CCoGAN - 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 |
| GAWIN - 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 Intraspective Adversarial Networks |
| iGAN - Generative Visual Manipulation on the Natural Image Manifold |
| IgGAN - 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 |

See also: [https://github.com/soumith/ganhacks](https://github.com/soumith/ganhacks) for tips and tricks for trainings GANs

https://github.com/hindupuravinash/the-gan-zoo
2017: Explosion of GANs

Better training and generation


Improved Wasserstein GAN, Gulrajani 2017.

Progressive GAN, Karras 2018.
2017: Explosion of GANs

Source→Target domain transfer


Text → Image Synthesis

Reed et al. 2017.

Many GAN applications

GANs

Don’t work with an explicit density function
Take game-theoretic approach: learn to generate from training distribution through 2-player game

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

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

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

Generative Models

- **PixelRNN and PixelCNN**
  
  Explicit density model, optimizes exact likelihood, good samples. But inefficient sequential generation.

- **Variational Autoencoders (VAE)**


- **Generative Adversarial Networks (GANs)**

  Game-theoretic approach, best samples! But can be tricky and unstable to train, no inference queries.
Recap

Generative Models

- **PixelRNN and PixelCNN**
  
  Explicit density model, optimizes exact likelihood, good samples. But inefficient sequential generation.

- **Variational Autoencoders (VAE)**
  

- **Generative Adversarial Networks (GANs)**

  Game-theoretic approach, best samples!

  But can be tricky and unstable to train, no inference queries.

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

Generative Models

- **PixelRNN and PixelCNN** Explicit density model, optimizes exact likelihood, good samples. But inefficient sequential generation.


- **Generative Adversarial Networks (GANs)** Game-theoretic approach, best samples! But can be tricky and unstable to train, no inference queries.

**Next time: Visualizing and understanding networks**