Overview

• Unsupervised Learning

• Generative Models
  • Variational Autoencoders (VAE)
  • Generative Adversarial Networks
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

• 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.
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, 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.
Supervised vs Unsupervised Learning

- **Unsupervised learning**
  - **Data:** $x$
  - **Just data, no labels**

- **Goal:** Learn some underlying hidden \textit{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 \( x \to y \)

- **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.
Generative Models

• Given training data, generate new samples from the same distribution

• Addresses density estimation, a core problem in unsupervised learning

• Explicit density estimation: explicitly define and solve for $p_{model}(x)$

• Implicit density estimation: learn model that can sample from $p_{model}(x)$ w/o explicitly defining it
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
  - Variational Autoencoder
- Markov Chain
  - Boltzmann Machine

Markov Chain
- GSN

Direct
- GAN

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

- Attempt to learn an identity function
- Latent vector constrained in some way (low dimensional)
- Generate new samples by giving different latent vectors to trained decoder
- Variational: use probabilistic latent encoding
Background: Autoencoders

• Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data
Background: Autoencoders

- Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data.
Background: Autoencoders

• Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data
Background: Autoencoders

• Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data
Background: Autoencoders

• How to learn this feature representation? Train such that features can be used to reconstruct original data
Background: Autoencoders

• How to learn this feature representation? Train such that features can be used to reconstruct original data
Background: Autoencoders

Train such that features can be used to reconstruct original data.

L2 Loss function:

\[ \|x - \hat{x}\|^2 \]

Reconstructed input data

Features

Input data

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

Reconstructed data

Encoder input data
Background: Autoencoders
Background: Autoencoders
Background: 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!

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.
Variational Autoencoders

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

How should we represent this model?
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) $\Rightarrow$ represent with neural network.
Variational Autoencoders

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

How to train the model?

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

Sample from true prior $p_{\theta^*}(z)$
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 BNs. Learn model parameters to maximize likelihood of training data

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

Now with latent $z$
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
Variational Autoencoders: Intractability

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

Intractible to compute \( p(x|z) \) for every \( z \)!
Variational Autoencoders: Intractability

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

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

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

Intractable data likelihood
Variational Autoencoders: Intractability

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

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

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

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

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

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

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

$\mu_{z \mid x}$

$\Sigma_{z \mid x}$

$\mathcal{X}$

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

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

$\mu_{x \mid z}$

$\Sigma_{x \mid z}$

$\mathcal{Z}$
Variational Autoencoders

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

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

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

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

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

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

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

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

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

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

Taking expectation wrt. $z$

(Using encoder network) will come in handy later
Variational Autoencoders

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

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

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

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

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

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

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

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

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

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

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

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

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:

\[
\log p_\theta(x^{(i)}) = \mathbb{E}_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right]
\]

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

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

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

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

- Decoder network gives \( p_\theta(x|z) \), can compute estimate of this term through sampling. (Sampling differentiable through reparam. trick, see paper.)
- This KL term (between Gaussians for encoder and \( z \) prior) has nice closed-form solution!
- \( p_\theta(z|x) \) intractable (saw earlier), can’t compute this KL term :( But we know KL divergence always \( \geq 0 \).
Variational Autoencoders: Reparametrization Trick

We want to use gradient descent to learn the model’s parameters.

Given $z$ drawn from $q_\theta(z|x)$, how do we take derivatives of (a function of) $z$ w.r.t. $\theta$?

We can reparameterize: $z = \mu + \sigma \odot \epsilon$

$\epsilon \sim \mathcal{N}(0, 1)$, and $\odot$ is element-wise product.

Can take derivatives of (functions of) $z$ w.r.t. $\mu$ and $\sigma$.

Output of $q_\theta(z|x)$ is vector of $\mu$'s and vector of $\sigma$'s.
Reparametrization Trick

- The reparametrization trick allows us to push the randomness of a normally-distributed random variable $z$ into epsilon, sampled from a standard normal. Diamonds are deterministic dependencies, circles are random variables.
Variational Autoencoders

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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:

\[
\log p_\theta(x^{(i)}) = \mathbb{E}_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right]
\]

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

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

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

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

\[
\geq 0
\]

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

Variational lower bound (“ELBO”)

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

Training: Maximize lower bound
Putting it all together: maximizing the likelihood lower bound

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

Putting it all together: maximizing the likelihood lower bound

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

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

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

Putting it all together: maximizing the likelihood lower bound

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

Putting it all together: maximizing the likelihood lower bound

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

Make approximate posterior distribution close to prior
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

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

Make approximate posterior distribution close to prior

Sample \( z \) from

\[ z \mid x \sim \mathcal{N}(\mu_z \mid x, \Sigma_z \mid x) \]

Decoder network

\( p_\theta(x \mid z) \)

Encoder network

\( q_\phi(z \mid x) \)

Input Data

\( \mathcal{X} \)
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

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

Maximize likelihood of original input being reconstructed

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

\( \mu_{x|z} \)

\( \Sigma_{x|z} \)

Decoder network

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

Make approximate posterior distribution close to prior

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

\( \mu_{z|x} \)

\( \Sigma_{z|x} \)

Encoder network

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

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

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

$\hat{x}$

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

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

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

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

Data manifold for 2-d $z$

Vary $z_1$

Vary $z_2$
Variational Autoencoders

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
Variational Autoencoders

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

Different dimensions of $z$ encode interpretable factors of variation

Also a good feature representation that can be computed using $q_\phi(z|x)$!
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
  - 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
CS839: Probabilistic Graphical Models

Lecture 20: Generative Adversarial Networks

Theo Rekatsinas
So far...

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

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

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

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

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

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

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

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

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

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!

Input: Random noise
Output: Sample from training distribution
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

![Diagram of GAN architecture](image)
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 output for real data x
- Discriminator output for generated fake data G(z)
- Discriminator outputs likelihood in (0,1) of real image
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)
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
\[
\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z)))
\]

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. 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.
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.

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

\[
\text{for number of training iterations do} \\
\quad \text{for } k \text{ steps do} \\
\quad \quad \cdot \text{Sample minibatch of } m \text{ noise samples } \{z^{(1)}, \ldots, z^{(m)}\} \text{ from noise prior } p_g(z). \\
\quad \quad \cdot \text{Sample minibatch of } m \text{ examples } \{x^{(1)}, \ldots, x^{(m)}\} \text{ from data generating distribution } p_{\text{data}}(x). \\
\quad \quad \cdot \text{Update the discriminator by ascending its stochastic gradient:} \\
\quad \quad \quad \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right] \\
\quad \text{end for} \\
\quad \cdot \text{Sample minibatch of } m \text{ noise samples } \{z^{(1)}, \ldots, z^{(m)}\} \text{ from noise prior } p_g(z). \\
\quad \cdot \text{Update the generator by ascending its stochastic gradient (improved objective):} \\
\quad \quad \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \\
\text{end for}
\]
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 \( p_{data}(x) \).
    - 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

After training, use generator network to generate new images
GANs: 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.
GANs: Convolutional Architectures

Generator
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
- 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 In-trospective Adversarial Networks
- iGAN - Generative Visual Manipulation on the Natural Image Manifold
- icGAN - Invertible Generative 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 for tips and tricks for trainings GANs
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