Lecture 12: Deep Generative Models

André Martins

Unbabel instituto de telecomunicações TÉCNICO LISBOA

Deep Structured Learning Course, Fall 2018

Announcements

- Homework 3 has been graded.
- The deadline for turning in Homework 4 is next week.
- The deadline for the final report is due January 2.
- The class presentations will be in January 9 and 16.

Today's Roadmap

Most of the course was about supervised learning. Today we'll talk about deep generative models for unsupervised learning.

- Deep auto-regressive models
- Boltzmann machines
- Deep belief networks
- Evidence lower bound (ELBO) and variational inference
- Wake-sleep algorithm
- Variational auto-encoders
- Generative adversarial networks
- Energy networks

Generative Modeling

Modeling complex high-dimensional data is an open problem

Deep generative models are currently making progress on this.

Goal: model $\mathbb{P}(\boldsymbol{x})$ (unsupervised learning) or $\mathbb{P}(\boldsymbol{x}, \boldsymbol{y})$ (supervised learning)

Often, deep generative models also use latent variables $\boldsymbol{h}$, in which case they may model $\mathbb{P}(\boldsymbol{x}, \boldsymbol{h})$ or $\mathbb{P}(\boldsymbol{x}, \boldsymbol{h}, \boldsymbol{y})$.

Examples of Deep Generative Models

- Auto-Regressive Networks
- Restricted Boltzmann Machines
- Deep Belief Networks
- Deep Boltzmann Machines
- Gaussian-Bernoulli RBMs
- Convolutional Boltzmann Machines
- Sigmoid Belief Nets
- Variational Auto-Encoders
- Generative Adversarial Networks
- Convolutional Generative Networks
- Generative Stochastic Networks

Examples of Deep Generative Models

- Auto-Regressive Networks
- Restricted Boltzmann Machines
- Deep Belief Networks
- Deep Boltzmann Machines
- Gaussian-Bernoulli RBMs
- Convolutional Boltzmann Machines
- Sigmoid Belief Nets
- Variational Auto-Encoders
- Generative Adversarial Networks
- Convolutional Generative Networks
- Generative Stochastic Networks

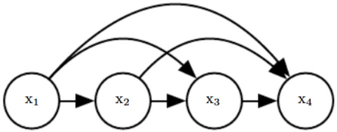
Outline

Deep Auto-Regressive Models

Deep auto-regressive models have no latent variables.

Instead, they use the chain rule of probabilities to decompose:

$$\mathbb{P}(\boldsymbol{x}) = \mathbb{P}(x_1) \times \mathbb{P}(x_2 \mid x_1) \times \ldots \times \mathbb{P}(x_D \mid x_1, \ldots, x_{D-1})$$



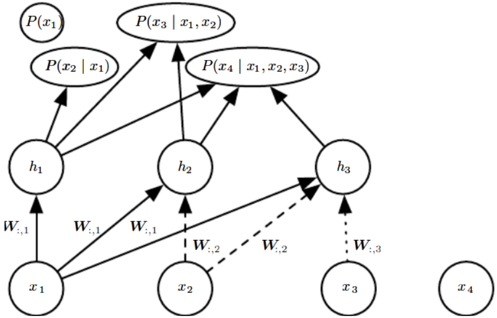
Also called fully-visible Bayes networks.

We saw examples already: RNNs, Pixel RNNs, Pixel CNNs, ...

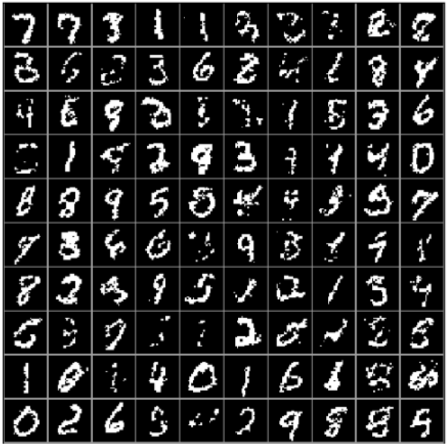
Neural Auto-Regressive Density Estimator (NADE)

Proposed by Larochelle and Murray (2011).

Similar to fully visible Bayes networks, but with some parameter sharing.



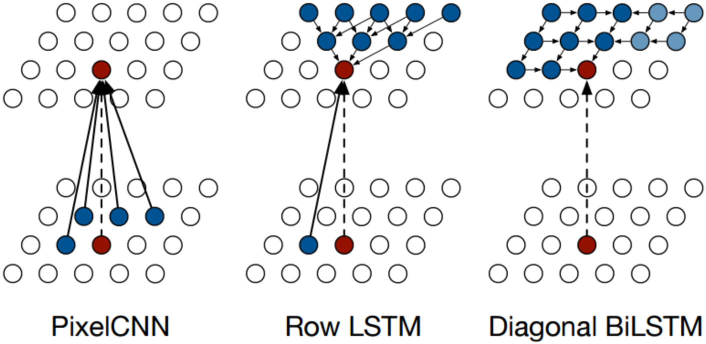
Neural Auto-Regressive Density Estimator (NADE)



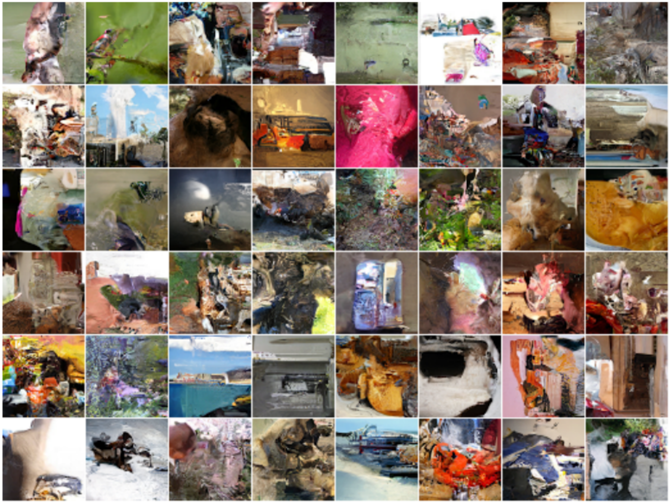
(Larochelle and Murray, 2011)

Examples: PixelCNNs and PixelRNNs

- Input-to-state and state-to-state mappings for PixelCNN and two PixelRNN models (Oord et al., 2016):



RNNs for Generating Images



(Oord et al., 2016)

Summary

Despite their simplicity, deep auto-regressive models can be very powerful.

However, for some problems they may require too many parameters/complex functions due to the assumption all variables are observed.

Models with latent variables are an appealing alternative, since they can represent "clusters" and explain the data on a simpler representation space.

Outline

Energy Based Models

Idea: define a probability distribution (mixing observed and latent variables) via an energy function $E(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{\theta})$:

$$\mathbb{P}(\boldsymbol{x}, \boldsymbol{h}) = \frac{\exp(-E(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{\theta}))}{Z(\boldsymbol{\theta})}$$

Maximizing probability corresponds to minimize the energy.

Challenges:

- Compute the partition function $Z(\boldsymbol{\theta})$
- Compute the evidence $\mathbb{P}(\boldsymbol{x})$
- Compute the posterior $\mathbb{P}(\boldsymbol{h} \mid \boldsymbol{x})$
- Sample?

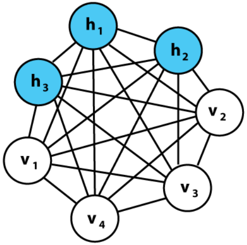
Boltzmann

"The maximum entropy S of a gas relates to the number of microstates W via $S = k \log W$ ($k = 1.38065 \times 10^{-23} J/K$)."

Boltzmann Machine (Ackley et al., 1985)

- Energy-based model to learn arbitrary probability distributions over binary vectors
- Defined over a binary random vector $\boldsymbol{x} = (\boldsymbol{v}, \boldsymbol{h}) \in \{0,1\}^{N \times M}$:

$$\mathbb{P}(\boldsymbol{v}, \boldsymbol{h}) = \frac{\exp(-E(\boldsymbol{v}, \boldsymbol{h}))}{Z}$$



- Some variables are observed ($\boldsymbol{v}$), others are latent ($\boldsymbol{h}$)
- Energy function:

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{v}^\top \mathbf{R}\boldsymbol{v} - \boldsymbol{v}^\top \mathbf{W}\boldsymbol{h} - \boldsymbol{h}^\top \mathbf{S}\boldsymbol{h} - \boldsymbol{b}^\top \boldsymbol{v} - \boldsymbol{c}^\top \boldsymbol{h}$$

Boltzmann Machine

The Boltzmann machine is a universal approximator of probability mass functions over discrete variables (Le Roux and Bengio, 2008)

Emulates the idea in Hebbian learning that “neurons that fire together wire together.”

However, in general:

- Sampling is hard
- Inference is hard
- Learning is hard.

How to Learn a Boltzmann Machine?

Learning is usually based on maximum likelihood.

All Boltzmann machines have an intractable partition function Z, so for learning the gradient must be approximated:

- contrastive divergence
- pseudo-likelihood
- noise-contrastive estimation
- annealed importance sampling

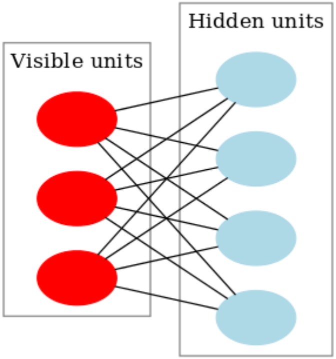
We won't cover this today (but check Goodfellow et al. (2016, Chapter 18)).

In a nutshell: learning a fully general Boltzmann machine is usually very challenging, so we typically resort to some particular cases.

Some Particular Cases

- Restricted Boltzmann machine
- Deep belief networks
- Deep Boltzmann machines
- . . .

Restricted Boltzmann Machines



Key idea: assumes that visible and hidden units are arranged as a bipartite graph.

Restricted Boltzmann Machines

Also called harmonium (Smolensky, 1986)

RBMs are undirected probabilistic graphical models containing

- a layer of observable variables
- a single layer of latent variables.

In other words: a bipartite graph, without intra-layer connections

The energy function becomes:

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{v}^\top \mathbf{W} \boldsymbol{h} - \boldsymbol{b}^\top \boldsymbol{v} - \boldsymbol{c}^\top \boldsymbol{h}$$

What is this buying us?

Restricted Boltzmann Machines

Unfortunately, the partition function Z is still intractable :(

... however, the conditional distributions $\mathbb{P}(\boldsymbol{h} \mid \boldsymbol{v})$ and $\mathbb{P}(\boldsymbol{v} \mid \boldsymbol{h})$ are now tractable! (next slide)

- easy to compute!
- easy to sample!
- can do MCMC with Gibbs sampling.

Restricted Boltzmann Machines

Why are the conditionals tractable?

- Because without intra-layer connections, $h_1, \ldots, h_N$ are conditionally independent given $\boldsymbol{v}$, hence $\mathbb{P}(\boldsymbol{h} \mid \boldsymbol{v})$ factors:

$$\mathbb{P}(h_j = 1 \mid \boldsymbol{v}) \quad = \quad \sigma(c_j + \mathbf{W}_{:,j}{}^\top \boldsymbol{v}), \quad \forall j = 1, \ldots, M.$$

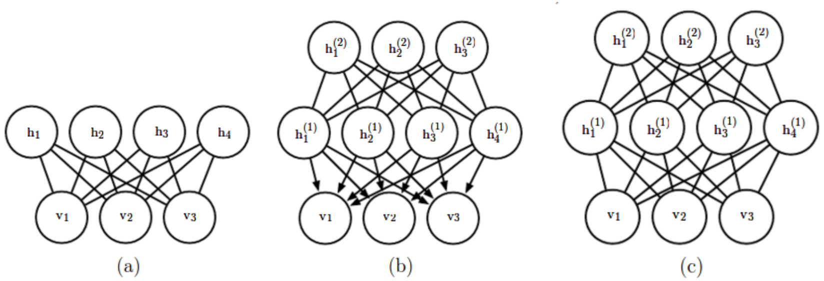
- Similarly for $\mathbb{P}(\boldsymbol{v} \mid \boldsymbol{h})$.

RBMs are relatively straightforward to train (by approximating Z).

See Goodfellow et al. (2016, Chapter 18) for more details.

RBMs may be stacked (one on top of the other) to form deeper models.

Some RBM's Friends



h_1 h_2 h_3 h_4
v_1 v_2 v_3
(a)
$h_1^{(2)}$ $h_2^{(2)}$ $h_3^{(2)}$
$h_1^{(1)}$ $h_2^{(1)}$ $h_3^{(1)}$ $h_4^{(1)}$
v_1 v_2 v_3
(b)
$h_1^{(2)}$ $h_2^{(2)}$ $h_3^{(2)}$
$h_1^{(1)}$ $h_2^{(1)}$ $h_3^{(1)}$ $h_4^{(1)}$
v_1 v_2 v_3
(c)

(Image from Goodfellow et al. (2016))

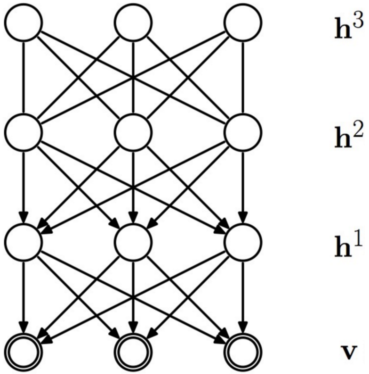
(a) Restricted Boltzmann machine (RBM)

(b) Deep belief network (DBN): hybrid directed/undirected GM with multiple latent layers

(c) Deep Boltzmann machine (DBM): undirected GM with several layers of latent variables.

Deep Belief Networks (Hinton et al., 2006)

- Began the deep learning renaissance!
- Before DBNs: deep models were considered too difficult to optimize
- Today, DBNs mostly fell out of favor
- **Idea:** several layers of latent variables, again no intra-layer connections



Deep Belief Networks (Hinton et al., 2006)

- The connections between the top two layers are undirected:

$$\mathbb{P}(\boldsymbol{h}^{(\ell)}, \boldsymbol{h}^{(\ell-1)}) \propto \exp(-\boldsymbol{h}^{(\ell-1)}{}^\top \mathbf{W}^{(\ell)} \boldsymbol{h}^{(\ell)} - (\boldsymbol{b}^{(\ell-1)})^\top \boldsymbol{h}^{(\ell-1)} - \boldsymbol{b}^{(\ell)}{}^\top \boldsymbol{h}^{(\ell)})$$

- The connections between all other layers are directed:

$$\begin{aligned} \mathbb{P}(h_i^k = 1 \mid \boldsymbol{h}^{(k+1)}) &= \sigma(b_i^{(k)} + \mathbf{W}_{:,i}^{(k+1)}{}^\top \boldsymbol{h}^{(k+1)}) \\ \mathbb{P}(v_i = 1 \mid \boldsymbol{h}^{(1)}) &= \sigma(b_i^{(0)} + \mathbf{W}_{:,i}^{(1)}{}^\top \boldsymbol{h}^{(1)}) \end{aligned}$$

Deep Belief Networks

Inference in a deep belief network is intractable:

- "explaining away" effect within each directed layer
- interaction between the two hidden layers with undirected connections

Evaluating or maximizing the standard evidence lower bound on the log-likelihood is also intractable

How to train a DBN?

Deep Belief Networks

Inference in a deep belief network is intractable:

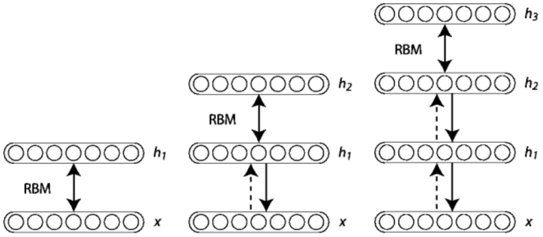
- "explaining away" effect within each directed layer
- interaction between the two hidden layers with undirected connections

Evaluating or maximizing the standard evidence lower bound on the log-likelihood is also intractable

How to train a DBN? Layerwise training.

Layerwise Training

- Begin by training a RBM for the first layer; then train a second RBM to model the distribution defined by sampling the hidden units of the first RBM, etc.



Most interest in DBNs arose from their ability to improve classification:

- take DBN's weights and define a MLP (discriminative fine-tuning)

Examples of Deep Generative Models

- Auto-Regressive Networks ✓
- Restricted Boltzmann Machines ✓
- Deep Belief Networks ✓
- Deep Boltzmann Machines ✓
- Gaussian-Bernoulli RBMs
- Convolutional Boltzmann Machines
- Sigmoid Belief Nets
- Variational Auto-Encoders
- Generative Adversarial Networks
- Convolutional Generative Networks
- Generative Stochastic Networks

Examples of Deep Generative Models

- Auto-Regressive Networks ✓
- Restricted Boltzmann Machines ✓
- Deep Belief Networks ✓
- Deep Boltzmann Machines ✓
- Gaussian-Bernoulli RBMs
- Convolutional Boltzmann Machines
- Sigmoid Belief Nets
- Variational Auto-Encoders
- Generative Adversarial Networks
- Convolutional Generative Networks
- Generative Stochastic Networks

Next: Differentiable Generator Networks

Several recent models are based on the idea of using a differentiable generator network.

This is a differentiable function $G(\boldsymbol{h}; \boldsymbol{\theta})$ that transforms latent variables $\boldsymbol{h}$ into sample reconstructions $\boldsymbol{x}$ (or distributions $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{h})$).

This idea underlies models such as:

- Variational auto-encoders
- Generative adversarial networks.

We'll cover those next.

Outline

Variational Auto-Encoders

Many latent variable models have:

- intractable evidence $\mathbb{P}(\boldsymbol{x})$
- intractable posterior $\mathbb{P}(\boldsymbol{h} \mid \boldsymbol{x})$.

Variational inference (e.g. mean field algorithms) is a technique used to approximate these quantities.

- Widely used in Bayesian inference, topic models, etc.

Auto-encoders are effective to learn data representations or codes (i.e. mapping $\boldsymbol{x} \longrightarrow \boldsymbol{h} \longrightarrow \boldsymbol{x}$)

Key idea: combine auto-encoders with variational inference.

Recap: HMMs

HMMs are defined by:

- a sequence of latent states $\boldsymbol{h} = h_1, \ldots, h_L$
- a sequence of observations $\boldsymbol{x} = x_1, \ldots, x_L$
- emissions $\mathbb{P}_{\boldsymbol{\theta}}(x_i \mid h_i)$
- transitions $\mathbb{P}_{\boldsymbol{\theta}}(h_{i+1} \mid h_i)$.

Is computing the evidence $\mathbb{P}(\boldsymbol{x})$ tractable?

Recap: HMMs

HMMs are defined by:

- a sequence of latent states $\boldsymbol{h} = h_1, \ldots, h_L$
- a sequence of observations $\boldsymbol{x} = x_1, \ldots, x_L$
- emissions $\mathbb{P}_{\boldsymbol{\theta}}(x_i \mid h_i)$
- transitions $\mathbb{P}_{\boldsymbol{\theta}}(h_{i+1} \mid h_i)$.

Is computing the evidence $\mathbb{P}(\boldsymbol{x})$ tractable? Yes. Forward and backward both return this.

Recap: HMMs

HMMs are defined by:

- a sequence of latent states $\boldsymbol{h} = h_1, \ldots, h_L$
- a sequence of observations $\boldsymbol{x} = x_1, \ldots, x_L$
- emissions $\mathbb{P}_{\boldsymbol{\theta}}(x_i \mid h_i)$
- transitions $\mathbb{P}_{\boldsymbol{\theta}}(h_{i+1} \mid h_i)$.

Is computing the evidence $\mathbb{P}(\boldsymbol{x})$ tractable? Yes. Forward and backward both return this.

Is computing the posteriors $\mathbb{P}(h_i \mid \boldsymbol{x})$ tractable?

Recap: HMMs

HMMs are defined by:

- a sequence of latent states $\boldsymbol{h} = h_1, \ldots, h_L$
- a sequence of observations $\boldsymbol{x} = x_1, \ldots, x_L$
- emissions $\mathbb{P}_{\boldsymbol{\theta}}(x_i \mid h_i)$
- transitions $\mathbb{P}_{\boldsymbol{\theta}}(h_{i+1} \mid h_i)$.

Is computing the evidence $\mathbb{P}(\boldsymbol{x})$ tractable? Yes. Forward and backward both return this.

Is computing the posteriors $\mathbb{P}(h_i \mid \boldsymbol{x})$ tractable? Yes. Forward-backward returns this.

Recap: HMMs

HMMs are defined by:

- a sequence of latent states $\boldsymbol{h} = h_1, \ldots, h_L$
- a sequence of observations $\boldsymbol{x} = x_1, \ldots, x_L$
- emissions $\mathbb{P}_{\boldsymbol{\theta}}(x_i \mid h_i)$
- transitions $\mathbb{P}_{\boldsymbol{\theta}}(h_{i+1} \mid h_i)$.

Is computing the evidence $\mathbb{P}(\boldsymbol{x})$ tractable? Yes. Forward and backward both return this.

Is computing the posteriors $\mathbb{P}(h_i \mid \boldsymbol{x})$ tractable? Yes. Forward-backward returns this.

Can we expect this to happen for every model?

Recap: HMMs

HMMs are defined by:

- a sequence of latent states $\boldsymbol{h} = h_1, \ldots, h_L$
- a sequence of observations $\boldsymbol{x} = x_1, \ldots, x_L$
- emissions $\mathbb{P}_{\boldsymbol{\theta}}(x_i \mid h_i)$
- transitions $\mathbb{P}_{\boldsymbol{\theta}}(h_{i+1} \mid h_i)$.

Is computing the evidence $\mathbb{P}(\boldsymbol{x})$ tractable? Yes. Forward and backward both return this.

Is computing the posteriors $\mathbb{P}(h_i \mid \boldsymbol{x})$ tractable? Yes. Forward-backward returns this.

Can we expect this to happen for every model? No.

Intractable Evidence and Posterior

In HMMs, the evidence $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x})$ and posterior $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h} \mid \boldsymbol{x})$ are tractable, via the forward-backward algorithm.

- This makes it possible to define an EM algorithm to estimate the model parameters $\boldsymbol{\theta}$

Unfortunately, for many other models both are intractable:

- Topic models like LDA (latent Dirichlet allocation)
- Mixtures of RNNs
- For probabilistic reconstruction models defined by neural networks, e.g. a Gaussian prior $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h})$ followed by a feedforward layer to define $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{h})$.

What to do in these cases?

Our Assumptions

Henceforth, we assume that:

- The prior $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h})$ is tractable (e.g. zero-mean, unit-variance Gaussian)
- The conditional $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{h})$ is tractable (e.g. a feed-forward neural network or an RNN).

But:

- Evidence $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x})$ (i.e. marginalizing out $\boldsymbol{h}$) is intractable
- Computing the posterior $\mathbb{P}(\boldsymbol{h} \mid \boldsymbol{x})$ in intractable.

We'll use variational inference to approximate the evidence and the posterior.

Outline

Basic Recap

Before proceeding, we need to recap what is:

- Shannon's entropy
- Kullback-Leibler divergence.

Recap: Shannon's Entropy

Let $\mathbb{P}$ be a distribution over $\boldsymbol{x} \in \mathcal{X}$.

$$H(\mathbb{P}) = -\sum_{\boldsymbol{x}} \mathbb{P}(\boldsymbol{x}) \log \mathbb{P}(\boldsymbol{x}).$$

Always non-negative, and zero iff $\boldsymbol{x}$ is deterministic (i.e. $\mathbb{P}(\boldsymbol{x})$ is a delta distribution).

Intuitively: how many bits on average do we need to encode an object $\boldsymbol{x} \sim \mathbb{P}(\boldsymbol{x})$ with an optimal code?

Recap: Kullback-Leibler Divergence

Let $\mathbb{P}$ and $\mathbb{Q}$ be two distributions over $\boldsymbol{x} \in \mathcal{X}$.

$$\begin{aligned} KL(\mathbb{P}\|\mathbb{Q}) &= \sum_{\boldsymbol{x}} \mathbb{P}(\boldsymbol{x}) \log \frac{\mathbb{P}(\boldsymbol{x})}{\mathbb{Q}(\boldsymbol{x})} \\ &= -\sum_{\boldsymbol{x}} \mathbb{P}(\boldsymbol{x}) \log \mathbb{Q}(\boldsymbol{x}) - H(\mathbb{P}) \end{aligned}$$

Always non-negative, and zero iff $\mathbb{P}(\boldsymbol{x}) = \mathbb{Q}(\boldsymbol{x})$.

Not symmetric!

Intuitively: how many extra bits on average do we need to encode an object $\boldsymbol{x} \sim \mathbb{P}(\boldsymbol{x})$ if our code is optimal for $\mathbb{Q}(\boldsymbol{x})$?

Evidence Lower Bound (ELBO)

This is a central concept in variational inference.

Evidence Lower Bound (ELBO)

Let $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h} \mid \boldsymbol{x})$ be the true posterior, and $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x})$ be the true evidence.

For any distribution $\mathbb{Q}(\boldsymbol{h})$, we have:

$$\begin{aligned} 0 &\geq -KL(\mathbb{Q}(\boldsymbol{h}) \| \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h} \mid \boldsymbol{x})) \\ &= \mathbb{E}_{\mathbb{Q}(\boldsymbol{h})}[\log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h} \mid \boldsymbol{x})] - \mathbb{E}_{\mathbb{Q}(\boldsymbol{h})}[\log \mathbb{Q}(\boldsymbol{h})] \\ &= \underbrace{\mathbb{E}_{\mathbb{Q}(\boldsymbol{h})}[\log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{h})] - \mathbb{E}_{\mathbb{Q}(\boldsymbol{h})}[\log \mathbb{Q}(\boldsymbol{h})]}_{\text{ELBO}(\mathbb{Q})} - \log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}). \end{aligned}$$

Therefore, we have the following **lower bound on the evidence**:

$$\begin{aligned} \log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \text{ELBO}(\mathbb{Q}) + KL(\mathbb{Q}(\boldsymbol{h}) \| \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h} \mid \boldsymbol{x})) \\ &\geq \text{ELBO}(\mathbb{Q}). \end{aligned}$$

Variational Inference

Evidence lower bound:

$$\begin{aligned}\log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \mathrm{ELBO}(\mathbb{Q}) + KL(\mathbb{Q}(\boldsymbol{h}) \| \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h} \mid \boldsymbol{x})) \\ &\geq \mathrm{ELBO}(\mathbb{Q}).\end{aligned}$$

Equality is achieved when $\mathbb{Q}(\boldsymbol{h}) = \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h} \mid \boldsymbol{x})$, but the latter is intractable.

Key idea: define a tractable family and look for the $\mathbb{Q}(\boldsymbol{h})$ in this family that maximize ELBO.

Since $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x})$ fixed, maximize $\mathrm{ELBO}(\mathbb{Q}) \Leftrightarrow$ minimize $KL(\mathbb{Q}(\boldsymbol{h}) \| \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h} \mid \boldsymbol{x}))$.

Rooted in old ideas from "calculus of variations" (Newton, Bernoulli, Euler, Lagrange, ...)

Evidence Lower Bound

ELBO can be written in multiple ways:

$$\begin{aligned}\mathsf{ELBO}(\mathbb{Q}) &= \mathbb{E}_{\mathbb{Q}(\boldsymbol{h})}[\log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{h})] - \mathbb{E}_{\mathbb{Q}(\boldsymbol{h})}[\log \mathbb{Q}(\boldsymbol{h})] \\ &= \mathbb{E}_{\mathbb{Q}(\boldsymbol{h})}[\log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{h})] - \mathbb{E}_{\mathbb{Q}(\boldsymbol{h})}[\log \frac{\mathbb{Q}(\boldsymbol{h})}{\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h})}] \\ &= \mathbb{E}_{\mathbb{Q}(\boldsymbol{h})}[\log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{h})] - KL(\mathbb{Q}(\boldsymbol{h}) \| \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h})).\end{aligned}$$

Which values of $\boldsymbol{h}$ is $\mathbb{Q}(\boldsymbol{h})$ encouraged to place its mass on?

- The first term is an **expected likelihood**: encourages placing mass on latent variables $\boldsymbol{h}$ that explain the observed data $\boldsymbol{x}$.
- The second term is the **negative divergence between $\mathbb{Q}(\boldsymbol{h})$ and the prior**: encourages staying close to the prior.

Thus, ELBO mirrors the usual balance between **likelihood** and **prior**.

Mean Field Approximation

Which tractable family to use for $\mathbb{Q}(\boldsymbol{h})$?

Mean field approximation: assume $\mathbb{Q}(\boldsymbol{h})$ is a factorial distribution,

$$\mathbb{Q}(\boldsymbol{h}) = \prod_i \mathbb{Q}(h_i).$$

More sophisticated: **structured mean field** (imposes a graphical model structure on $\mathbb{Q}$ that captures interactions among the latent variables and is still tractable)

Lots of literature on this topic; see Wainwright and Jordan (2008) for details.

Example: Bayesian Inference

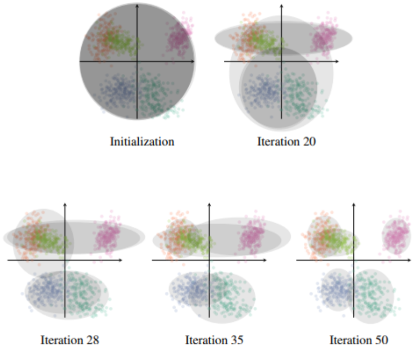
We saw before that the EM algorithm seeks maximum-likelihood estimates in models with latent variables (e.g., HMMs, mixtures of Gaussians)

In Bayesian inference, **model parameters are treated as latent variables**

- This leads to a coupling between *global* latent variables (corresponding to the model parameters $\boldsymbol{\mu}$) and *local* latent variables $\boldsymbol{z}$ (e.g. states in HMMs, clusters in mixtures of Gaussians), making inference intractable
- Mean field inference uses a variational approximation $\mathbb{Q}(\boldsymbol{z}, \boldsymbol{\mu}) = \mathbb{Q}(\boldsymbol{z})\mathbb{Q}(\boldsymbol{\mu})$ and then minimizes ELBO, leading to alternating optimization algorithms similar to EM

Doing these updates at a per-instance level leads to stochastic variational inference, which scales to large datasets.

Stochastic Variational Inference



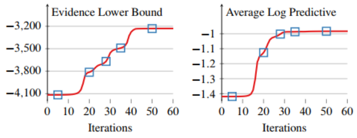


Figure 3: A simulation study of a two dimensional Gaussian mixture model. The ellipses are 2σ contours of the variational approximating factors.

(From Blei et al. (2017).)

Amortized Variational Inference

What we described so far requires optimizing the variational distribution $\mathbb{Q}(\boldsymbol{h})$ for every example $\boldsymbol{x}$ in the training set.

- This can be expensive: requires several gradient/coordinate ascent iterations per example.

An alternative is to use amortized variational inference!

Key idea: instead of optimizing $\mathbb{Q}(\boldsymbol{h})$ for every example, use an encoder with shared parameters $\boldsymbol{\phi}$ and define $\mathbb{Q}_{\boldsymbol{\phi}}(\boldsymbol{h} \mid \boldsymbol{x})$.

For each example:

- make a forward pass on the encoder to obtain $\mathbb{Q}_{\boldsymbol{\phi}}(\boldsymbol{h} \mid \boldsymbol{x})$
- backpropagate through the encoder to update $\boldsymbol{\phi}$.

Example: Multivariate Bernoulli with Continuous Latent Variables (Kingma and Welling, 2013)

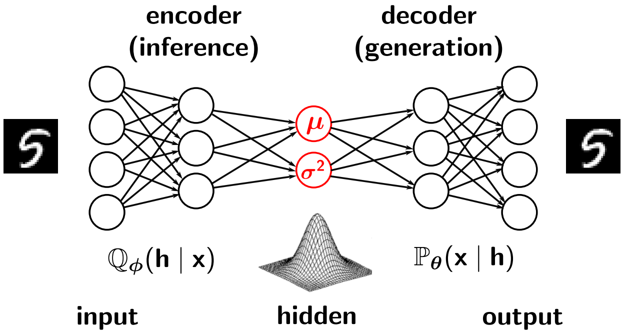
- Prior is a multivariate isotropic Gaussian $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h}) = \mathcal{N}(\boldsymbol{h}; \mathbf{0}, \mathbf{I})$
- Conditional is a multivariate Bernoulli

$$\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{h}) = \prod_{i=1}^{D} \sigma(f_i(\boldsymbol{h}; \boldsymbol{\theta}))^{x_i}(1 - \sigma(f_i(\boldsymbol{h}; \boldsymbol{\theta})))^{1-x_i},$$

  where $\boldsymbol{f}(\boldsymbol{h}; \boldsymbol{\theta})$ is computed by a MLP with parameters $\boldsymbol{\theta}$ when $\boldsymbol{h}$ is given as input
- The true posterior $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h} \mid \boldsymbol{x})$ is intractable
- Approximate the posterior with a variational distribution $\mathbb{Q}_{\boldsymbol{\phi}}(\boldsymbol{h} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{h}; \boldsymbol{\mu}(\boldsymbol{x}; \boldsymbol{\phi}), \boldsymbol{\sigma}^2(\boldsymbol{x}; \boldsymbol{\phi}))$

Variational Auto-Encoders (Kingma and Welling, 2013)

This leads to variational auto-encoders:



encoder (inference)
decoder (generation)
μ
σ^2
$\mathbb{Q}_\phi(\mathbf{h} \mid \mathbf{x})$
$\mathbb{P}_\theta(\mathbf{x} \mid \mathbf{h})$
input
hidden
output

... we'll come back to this!

Related: Helmholtz Machines and Wake-Sleep

This model is related to Helmholtz machines and the Wake-Sleep algorithm, which we next describe.

Helmholtz Machines

Trained to create a generative model of the data

Key idea: by learning a compact representation of the data, the underlying structure of the generative model should reasonably approximate the hidden structure of the data set.

Two components:

- A inference network to represent $\mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x})$ (also called **recognition network**)
- A generation network to represent $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{h})$ (also called **reconstruction network**)

ELBO is also called the (negative) variational Helmholtz free energy.

Wake-Sleep Algorithm (Hinton et al., 1995)

Works in two phases:

- a wake phase: observe external data and update generation model to increase likelihood
- a sleep phase: keep eyes shut, and use the model to generate new data, updating the inference model

Inspired by biological theories of how mammals dream during sleep.

Wake-Sleep Algorithm (Hinton et al., 1995)

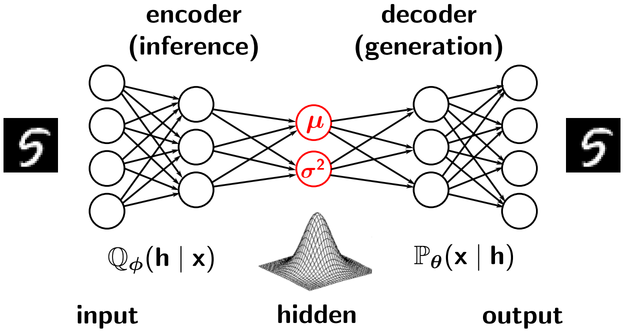
- **Wake phase:** use the inference model ϕ to draw samples $\boldsymbol{h}^{(1)}, \ldots, \boldsymbol{h}^{(N)}$ according to $\mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x})$ and use them to update the generator model $\boldsymbol{\theta}$ by maximizing:

$$\mathbb{E}_{\mathbb{Q}_{\phi}(\boldsymbol{h}|\boldsymbol{x})}[\nabla_{\boldsymbol{\theta}} \log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{h})] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{h}^{(i)})$$

- **Sleep phase:** use the generator model $\boldsymbol{\theta}$ to sample $\boldsymbol{h}^{(1)}, \ldots, \boldsymbol{h}^{(N)} \sim \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h})$ and $\boldsymbol{x}^{(i)} \sim \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{h}^{(i)})$. Then update the inference model ϕ to maximize $\log \mathbb{Q}_{\phi}(\boldsymbol{h}^{(i)} \mid \boldsymbol{x}^{(i)})$:

$$\mathbb{E}_{\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{h})}[\nabla_{\phi} \log \mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x})] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_{\phi} \log \mathbb{Q}_{\phi}(\boldsymbol{h}^{(i)} \mid \boldsymbol{x}^{(i)})$$

Wake-Sleep and Variational Auto-Encoder



encoder (inference)
decoder (generation)
μ
σ^2
$\mathbb{Q}_\phi(\mathbf{h} \mid \mathbf{x})$
$\mathbb{P}_\theta(\mathbf{x} \mid \mathbf{h})$
input
hidden
output

- Wake phase updates $\mathbb{P}_\theta(\boldsymbol{x} \mid \boldsymbol{h})$
- Sleep phase updates $\mathbb{Q}_\phi(\boldsymbol{h} \mid \boldsymbol{x})$

Limitations of the Wake-Sleep Algorithm

Advantages:

- simple to implement
- the wake-sleep gradient for the inference network parameters ϕ is much easier to estimate than the actual variational bound gradient

Disadvantages:

- wake-sleep is not optimizing any well-defined objective function
- wake and sleep phases are optimizing separate parts of the model.

Next: a strategy to maximize the ELBO objective end-to-end.

Outline

Parameter Gradients

Recall that:

$$\mathrm{ELBO}(\boldsymbol{\phi}; \boldsymbol{\theta}) = \mathbb{E}_{\mathbb{Q}_{\boldsymbol{\phi}}(\boldsymbol{h}|\boldsymbol{x})}[\log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{h}) - \log \mathbb{Q}_{\boldsymbol{\phi}}(\boldsymbol{h} \mid \boldsymbol{x})].$$

We need to compute gradients with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$.

Gradient of the generation network $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} \mathrm{ELBO}(\boldsymbol{\phi}; \boldsymbol{\theta}) = \mathbb{E}_{\mathbb{Q}_{\boldsymbol{\phi}}(\boldsymbol{h}|\boldsymbol{x})}[\nabla_{\boldsymbol{\theta}} \log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{h})]$$

- Follows from linearity of the expectation.
- This is simple and can be well approximated with Monte Carlo samples.

Parameter Gradients

Recall that:

$$\mathrm{ELBO}(\boldsymbol{\phi};\boldsymbol{\theta}) = \mathbb{E}_{\mathbb{Q}_{\boldsymbol{\phi}}(\boldsymbol{h}|\boldsymbol{x})}[\log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{h}) - \log \mathbb{Q}_{\boldsymbol{\phi}}(\boldsymbol{h} \mid \boldsymbol{x})].$$

Gradient of the inference network:

$$\begin{aligned}&\nabla_{\boldsymbol{\phi}}\mathrm{ELBO}(\boldsymbol{\phi};\boldsymbol{\theta}) \\ &= \mathbb{E}_{\mathbb{Q}_{\boldsymbol{\phi}}(\boldsymbol{h}|\boldsymbol{x})}[\underbrace{(\log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{h}) - \log \mathbb{Q}_{\boldsymbol{\phi}}(\boldsymbol{h} \mid \boldsymbol{x}))}_{\text{“reward”} \; R_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{h})} \times \nabla_{\boldsymbol{\phi}} \log \mathbb{Q}_{\boldsymbol{\phi}}(\boldsymbol{h} \mid \boldsymbol{x})].\end{aligned}$$

- This is hard—Monte Carlo estimators have high variance due to the left part!
- As in REINFORCE (Williams, 1992), we can mitigate this by using a baseline to reduce variance.

Derivation of the Inference Network Gradient

$$\begin{aligned}
&\nabla_{\phi}\text{ELBO}(\phi; \boldsymbol{\theta}) \\
&= \nabla_{\phi}\mathbb{E}_{\mathbb{Q}_{\phi}(\boldsymbol{h}|\boldsymbol{x})}[\log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{h}) - \log \mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x})] \\
&= \nabla_{\phi} \sum_{\boldsymbol{h}} \mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x}) \log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{h}) - \nabla_{\phi} \sum_{\boldsymbol{h}} \mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x}) \log \mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x}) \\
&= \sum_{\boldsymbol{h}} \log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{h}) \nabla_{\phi}\mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x}) - \sum_{\boldsymbol{h}} (1 + \log \mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x})) \nabla_{\phi}\mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x}) \\
&= \sum_{\boldsymbol{h}} (\log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{h}) - \log \mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x})) \nabla_{\phi}\mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x}) \\
&= \mathbb{E}_{\mathbb{Q}_{\phi}(\boldsymbol{h}|\boldsymbol{x})}[(\log \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{h}) - \log \mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x})) \times \nabla_{\phi} \log \mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x})],
\end{aligned}$$

where we used the facts:

$$\sum_{\boldsymbol{h}} \nabla_{\phi}\mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x}) = \nabla_{\phi} \sum_{\boldsymbol{h}} \mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x}) = \nabla_{\phi} 1 = 0.$$

$$\nabla_{\phi}\mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x}) = \mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x}) \nabla_{\phi} \log \mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x}).$$

To sum up, the bottleneck is the gradient of the inference network ϕ, whose Monte Carlo approximation has large variance.

Is there a better strategy?

To sum up, the bottleneck is the gradient of the inference network ϕ, whose Monte Carlo approximation has large variance.

Is there a better strategy? Yes—the reparameterization trick.

Reparameterization Trick (Kingma and Welling, 2013)

How to draw samples $\boldsymbol{h} \sim \mathbb{Q}_\phi(\boldsymbol{h} \mid \boldsymbol{x})$?

Trick:

- Define an auxiliary random variable $\boldsymbol{\epsilon}$ with independent marginal $\mathbb{P}(\boldsymbol{\epsilon})$
- Sample $\boldsymbol{\epsilon} \sim \mathbb{P}(\boldsymbol{\epsilon})$, and express the random variable $\boldsymbol{h}$ as a deterministic variable $\boldsymbol{h} = \boldsymbol{g}_\phi(\boldsymbol{\epsilon}, \boldsymbol{x})$.

Then, we have:

$$\mathbb{E}_{\mathbb{Q}_\phi(\boldsymbol{h}|\boldsymbol{x})}[f(\boldsymbol{h})] \approx \frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{g}_\phi(\boldsymbol{x}, \boldsymbol{\epsilon}^{(i)}))$$

and gradients with respect to ϕ can be estimated with regular backpropagation over $\boldsymbol{g}_\phi$.

Reparameterization Trick

This construction is possible in many cases for continuous latent variables:

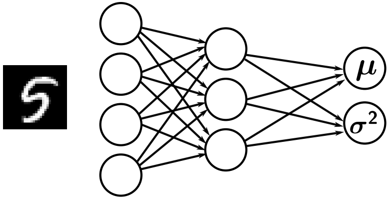
- exponential
- Gaussian
- location-scale family
- log-normal, etc.

For discrete latent variables, it is still possible via the Gumbel-softmax trick (we won't cover this today).

Example: Gaussian

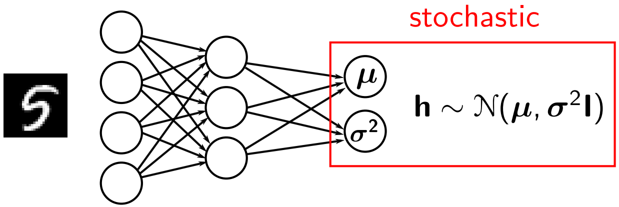
1. Sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{I})$
2. Use inference network $\boldsymbol{g}_\phi$ with input $\boldsymbol{x}$ to output mean $\boldsymbol{\mu}(\boldsymbol{x})$ and variance $\boldsymbol{\sigma}^2(\boldsymbol{x})$
3. Set $\boldsymbol{h} = \boldsymbol{\mu}(\boldsymbol{x}) + \boldsymbol{\epsilon}\boldsymbol{\sigma}(\boldsymbol{x})$.

Reparameterization Trick

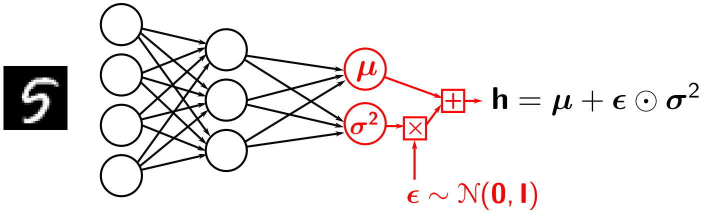


$$\mathbf{h} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2 \mathbf{I})$$

Reparameterization Trick

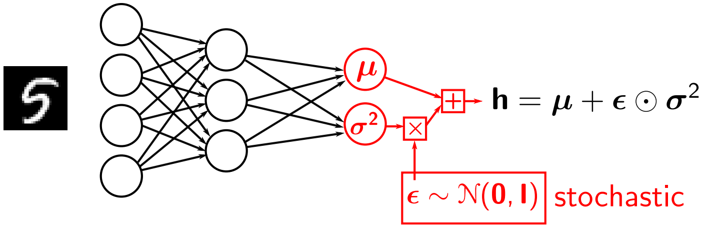

stochastic
μ
σ^2
$\mathbf{h} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I})$

Reparameterization Trick

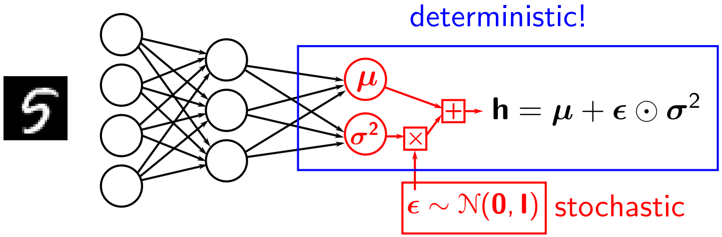


μ
σ^2
$\mathbf{h} = \boldsymbol{\mu} + \boldsymbol{\epsilon} \odot \boldsymbol{\sigma}^2$
$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

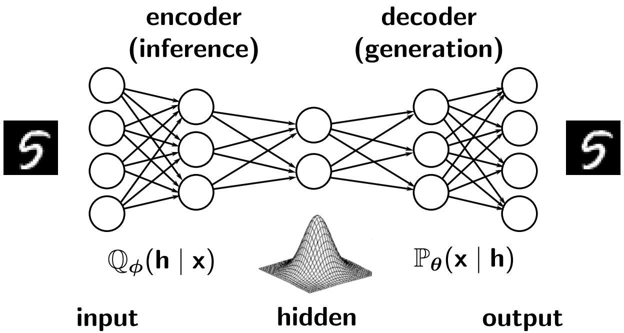
Reparameterization Trick


$\boldsymbol{\mu}$
$\boldsymbol{\sigma}^2$
$+$
$\times$
$\mathbf{h} = \boldsymbol{\mu} + \boldsymbol{\epsilon} \odot \boldsymbol{\sigma}^2$
$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
stochastic

Reparameterization Trick


deterministic!
$\boldsymbol{\mu}$
$\boldsymbol{\sigma}^2$
$\mathbf{h} = \boldsymbol{\mu} + \boldsymbol{\epsilon} \odot \boldsymbol{\sigma}^2$
$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
stochastic

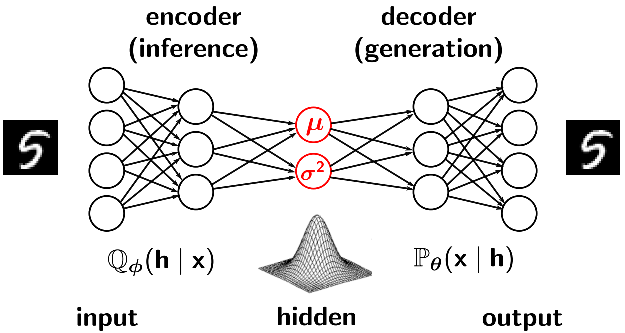
Variational Auto-Encoders (Kingma and Welling, 2013)



encoder
(inference)
decoder
(generation)
$\mathbb{Q}_\phi(\mathbf{h} \mid \mathbf{x})$
$\mathbb{P}_\theta(\mathbf{x} \mid \mathbf{h})$
input
hidden
output

- Decoder computes $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h})$ and $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{h})$
- Encoder computes $\mathbb{Q}_\phi(\boldsymbol{h} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{h}; \boldsymbol{\mu}_\phi(\boldsymbol{x}), \boldsymbol{\sigma}^2_\phi(\boldsymbol{x}))$
- Loss function: ELBO.

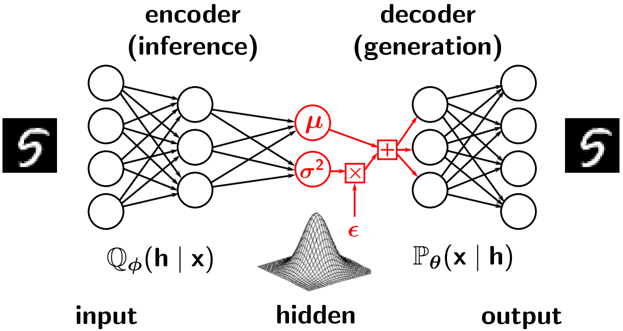
Variational Auto-Encoders (Kingma and Welling, 2013)



encoder
(inference)
decoder
(generation)
μ
σ^2
$\mathbb{Q}_\phi(\mathbf{h} \mid \mathbf{x})$
$\mathbb{P}_\theta(\mathbf{x} \mid \mathbf{h})$
input
hidden
output

- Decoder computes $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h})$ and $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{h})$
- Encoder computes $\mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{h}; \boldsymbol{\mu}_{\phi}(\boldsymbol{x}), \boldsymbol{\sigma}^2_{\phi}(\boldsymbol{x}))$
- Loss function: ELBO.

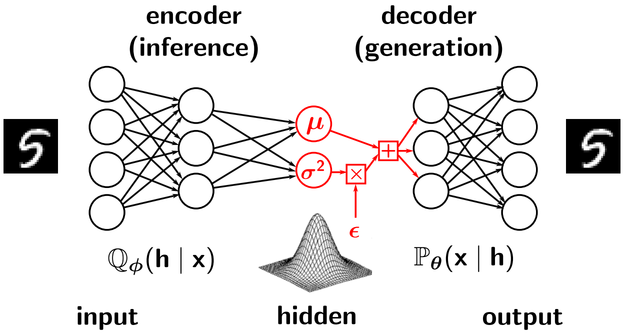
Variational Auto-Encoders (Kingma and Welling, 2013)



encoder
(inference)
decoder
(generation)
μ
σ^2
ϵ
$\mathbb{Q}_\phi(\mathbf{h} \mid \mathbf{x})$
$\mathbb{P}_\theta(\mathbf{x} \mid \mathbf{h})$
input
hidden
output

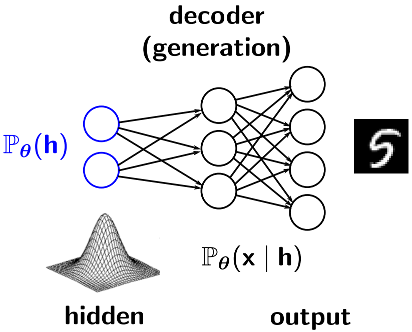
- Decoder computes $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h})$ and $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{h})$
- Encoder computes $\mathbb{Q}_\phi(\boldsymbol{h} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{h}; \boldsymbol{\mu}_\phi(\boldsymbol{x}), \boldsymbol{\sigma}_\phi^2(\boldsymbol{x}))$
- Loss function: ELBO.

Summing Up: VAEs at Training Time



encoder
(inference)
decoder
(generation)
μ
σ^2
ϵ
$\mathbb{Q}_\phi(\mathbf{h} \mid \mathbf{x})$
$\mathbb{P}_\theta(\mathbf{x} \mid \mathbf{h})$
input
hidden
output

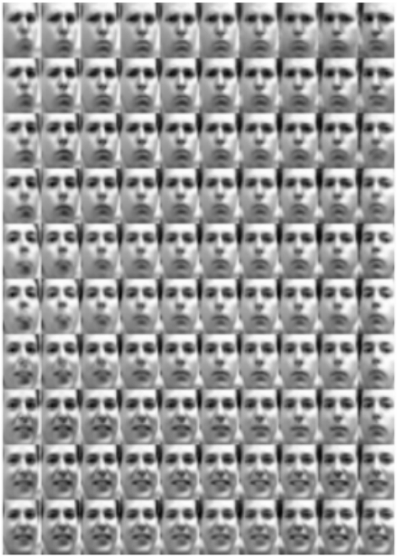
Summing Up: VAEs at Test Time


decoder
(generation)
$\mathbb{P}_\theta(\mathbf{h})$
$\mathbb{P}_\theta(\mathbf{x} \mid \mathbf{h})$
hidden
output

What is the Latent Variable Representing?

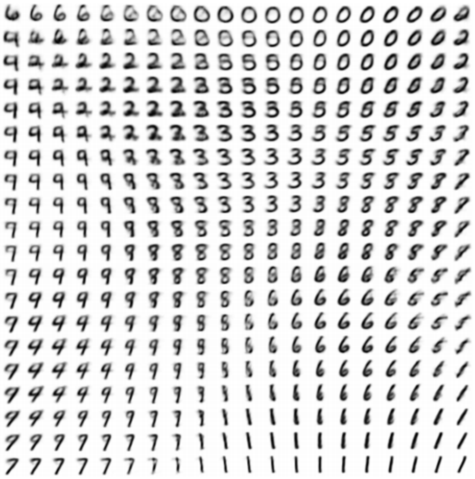
- One very nice property of the variational autoencoder is that simultaneously training a parametric encoder in combination with the generator network forces the model to learn a predictable coordinate system that the encoder can capture.
- This makes it an excellent manifold learning algorithm.
- Example: the algorithm discovered two independent factors of variation present in images of faces: angle of rotation and emotional expression.

What is the Latent Variable Representing?



From Kingma and Welling (2013).

What is the Latent Variable Representing?



(b) Learned MNIST manifold

From Kingma and Welling (2013).

Issues with VAEs

Posterior collapse: if the generative part is strong, the model learns to ignore the latent variables:

$$\begin{aligned} \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{h}) &\approx \mathbb{P}(\boldsymbol{x}) \\ \mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x}) &\approx \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h}). \end{aligned}$$

Can be mitigated with a few tricks:

- Decrease/anneal the weight of the $KL(\mathbb{Q}_{\phi}(\boldsymbol{h} \mid \boldsymbol{x}) \| \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h}))$ in the ELBO objective
- Use auxiliary losses
- Combine stochastic and amortized inference.

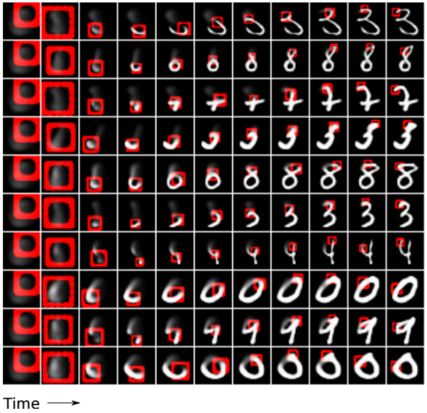
In general, reporting both reconstruction loss and the KL term is needed to be able to tell if the model makes use of the latent variables.

DRAW: Deep Recurrent Attentive Writer (Gregor et al., 2015)

DRAW uses a recurrent encoder and recurrent decoder combined with an attention mechanism.

The generation process for the DRAW model consists of sequentially visiting different small image patches and drawing the values of the pixels at those points.

DRAW: Deep Recurrent Attentive Writer (Gregor et al., 2015)



Examples of Deep Generative Models

- Auto-Regressive Networks ✓
- Restricted Boltzmann Machines ✓
- Deep Belief Networks ✓
- Deep Boltzmann Machines ✓
- Gaussian-Bernoulli RBMs
- Convolutional Boltzmann Machines
- Sigmoid Belief Nets
- Variational Auto-Encoders ✓
- Generative Adversarial Networks
- Convolutional Generative Networks
- Generative Stochastic Networks

Examples of Deep Generative Models

- Auto-Regressive Networks ✓
- Restricted Boltzmann Machines ✓
- Deep Belief Networks ✓
- Deep Boltzmann Machines ✓
- Gaussian-Bernoulli RBMs
- Convolutional Boltzmann Machines
- Sigmoid Belief Nets
- Variational Auto-Encoders ✓
- Generative Adversarial Networks
- Convolutional Generative Networks
- Generative Stochastic Networks

Outline

Why Maximum Likelihood?

All models we discussed so far attempt to maximize the likelihood (evidence) $\mathbb{P}(\boldsymbol{x})$

In fact, since this is intractable, they maximize a lower bound (ELBO)

But if we want to build a generator, is this really the best criterion?

- Maximum likelihood tends to produce fuzzy outputs (blurry images)

Generative Adversarial Networks (GANs) (Goodfellow et al., 2014)

Key idea:

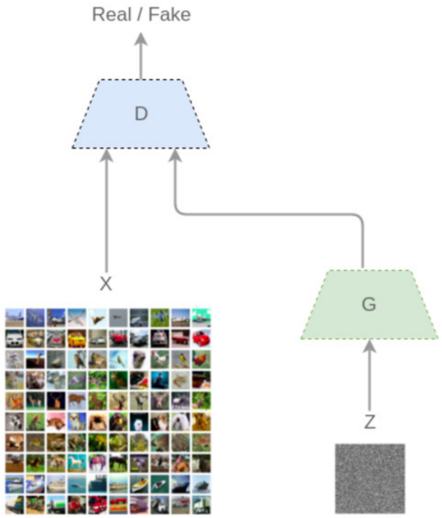
- keep the generation network $G = \{\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h}), \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{h})\}$
- drop the inference network and use instead a discriminator network $D : \mathcal{X} \to \{0, 1\}$

Formulate the learning problem as a game between two players:

- the generator's job is to generate data that looks real
- the discriminator's job is to distinguish between real data and fake data generated by the generator

This is like a Turing test!

Generative Adversarial Networks (GANs) (Goodfellow et al., 2014)



Minimax Game

We arrive at a saddle point problem:

$$\min_G \max_D V(D, G) \quad = \quad \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{h}\sim\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h})}[\log(1 - D(G(\boldsymbol{h})))].$$

The optimal discriminator (intractable to compute) is:

$$D^\star(\boldsymbol{x}) = \frac{\mathbb{P}_{\text{data}}(\boldsymbol{x})}{\mathbb{P}_{\text{data}}(\boldsymbol{x}) + \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x})}, \qquad \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{h})\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h}).$$

Given $D^\star(\boldsymbol{x})$, the optimal $G^\star(\boldsymbol{x})$ is the one minimizing the Jensen-Shannon divergence between $\mathbb{P}_{\text{data}}(\boldsymbol{x})$ and $\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x})$:

$$JS(\mathbb{P}_{\text{data}}(\boldsymbol{x}), \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x})) = \frac{1}{2}KL\left(\mathbb{P}_{\text{data}}(\boldsymbol{x}) \| \bar{\mathbb{P}}(\boldsymbol{x})\right) + \frac{1}{2}KL\left(\mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x}) \| \bar{\mathbb{P}}(\boldsymbol{x})\right),$$

where $\bar{\mathbb{P}}(\boldsymbol{x}) = \frac{\mathbb{P}_{\text{data}}(\boldsymbol{x}) + \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{x})}{2}$.

Training GANs

How to train a GAN?

Use stochastic gradient descent! Alternate between:

- Stochastic gradients updates of the generator parameters $\boldsymbol{\theta}$
- Stochastic gradients updates of the discriminator D.

Several variants and schedules have been proposed.

Caveat: no convergence guarantees; optimization in GANs is often difficult.

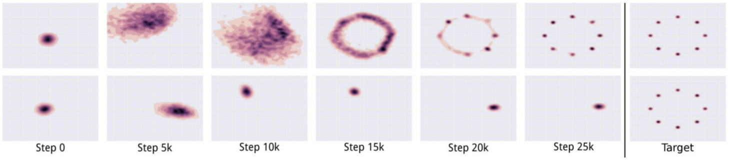
Images Generated by GANs

(https://tryolabs.com/blog/2016/12/06/major-advancements-deep-learning-2016/)

Mode Collapse

$$\min_G \max_D V(D, G) \neq \max_D \min_G V(D, G).$$

- G in inner loop: place all mass on most likely point



(From Metz et al. (2016))

What prevents the generator from picking the same example all the time?

The top row finds all the modes, the bottom finds just one mode.

Mode Collapse

GANs often seem to collapse to far fewer modes than the model can represent

This causes low output diversity.

How to mitigate mode collapse?

One strategy: minibatch features (Salimans et al., 2016)

- Let the discriminator make a decision by comparing an example to a whole minibatch of fake/real examples
- Discriminator can now consider diversity.

Wasserstein GANs (WGANs)

Instead of optimizing the Jensen-Shannon divergence, optimize instead:

$$\min_G \max_D V(D, G) \quad = \quad \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_{\mathsf{data}}(\boldsymbol{x})}[D(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{h} \sim \mathbb{P}_{\boldsymbol{\theta}}(\boldsymbol{h})}[D(G(\boldsymbol{h}))].$$

This is related to the Wasserstein distance (also called Earth mover's distance).

A technical condition is that ∇D is bounded; in practice this is ensured with gradient clipping.

This improves stability and mitigates the mode collapse problem.

Pros and Cons of GANs

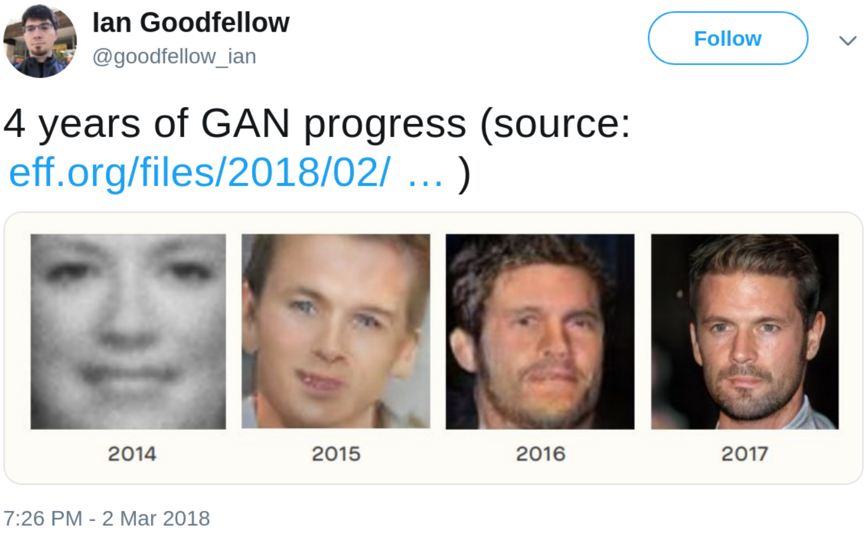
Advantages:

- They currently generate the sharpest images
- They are easy to train (since no statistical inference is required), and only back-propogation is needed to obtain gradients

Disadvantages:

- GANs are difficult to optimize due to unstable training dynamics.
- No statistical inference can be done with them.

Still Improving...


Ian Goodfellow
@goodfellow_ian

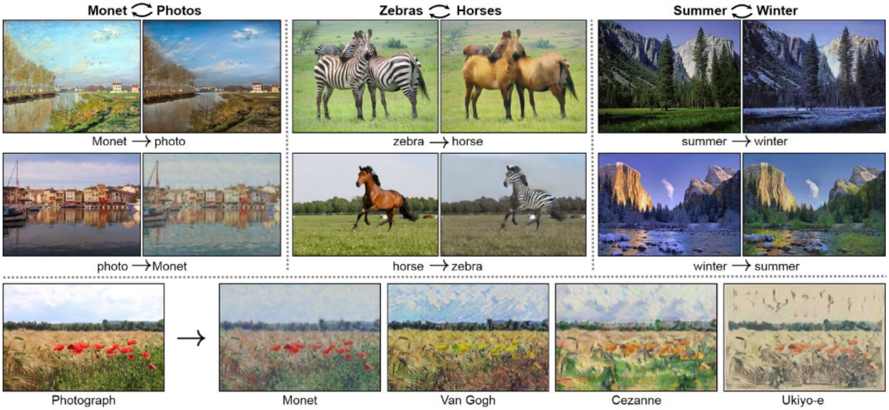
4 years of GAN progress (source: eff.org/files/2018/02/ ...)

7:26 PM - 2 Mar 2018

Some Extensions of GANs

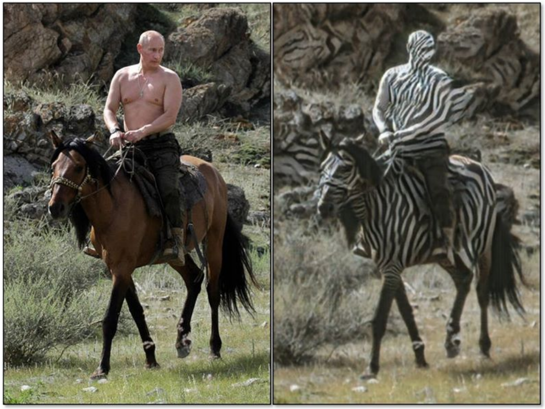
- Augmenting GANs with an inference network (Dumoulin et al., 2016; Donahue et al., 2016)
- Domain adversarial training for domain adaptation (Ganin et al., 2016)
- Conditional GANs and semi-supervised GANs (Salimans et al., 2016)
- CycleGAN (Zhu et al., 2017): "translate" images from a source domain $\mathcal{X}$ to a target domain $\mathcal{Y}$ without paired examples. Use two generators $G : \mathcal{Y} \rightarrow \mathcal{X}$ and $G : \mathcal{Y} \rightarrow \mathcal{X}$ and introduce a **cycle consistency loss** to push $F(G(\boldsymbol{y})) \approx \boldsymbol{y}$ and $G(F(\boldsymbol{x})) \approx \boldsymbol{x}$.

Image-to-Image Translation w/ CycleGAN (Zhu et al., 2017)



(`https://junyanz.github.io/CycleGAN`)

Failure Cases



(https://junyanz.github.io/CycleGAN)

Evaluation

There is not any single compelling way to evaluate a generative model.

- Models with good likelihood can produce bad samples
- Models with good samples can have bad likelihood
- There is not a good way to quantify how good samples are
- For GANs, it is also hard to even estimate the likelihood
- See "A note on the evaluation of generative models," Theis et al. (2015), for a good overview.

Discrete Outputs

To train a GAN, G must be differentiable

But G cannot be differentiable if the output is discrete.

Possible workarounds:

- REINFORCE (Williams, 1992)
- Concrete/Gumbel-softmax distribution (Maddison et al., 2016; Jang et al., 2016)
- Learn distribution over continuous embeddings, decode to discrete

How does this compare with VAEs?

- VAEs have trouble with discrete latent variables (cannot differentiate through the inference network)
- GANs have trouble with discrete output variables (cannot differentiate through the generator network).

Connections to Reinforcement Learning

We can regard the discriminator loss as a reward signal for the generator.

- GANs interpreted as actor-critic (Pfau and Vinyals, 2016)
- GANs as inverse reinforcement learning (Finn et al., 2016)

Outline

Conclusions

- Generative models are useful to model high-dimensional data
- Latent-variable generative models are appealing since they are more compact ("minimum description length" principle)
- Often, computing evidence and posterior distributions is intractable (e.g. Boltzmann machines)
- A common surrogate for maximum likelihood is the evidence lower bound (ELBO)
- Variational auto-encoders optimize the ELBO with amortized VI
- Their main drawback is posterior collapse
- Generative adversarial networks (GANs) are formulated as a game between a generator and a discriminator
- They manage to generate sharp outputs, but suffer from mode collapse and do not return a likelihood score
- Open problem (both VAEs/GANs): how to deal with discrete data?

Thank you!

Questions?

References I

Ackley, D., Hinton, G., and Sejnowski, T. (1985). A learning algorithm for Boltzmann machines. *Cognitive science*, 9(1):147–169.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877.

Donahue, J., Krähenbühl, P., and Darrell, T. (2016). Adversarial feature learning. *arXiv preprint arXiv:1605.09782*.

Dumoulin, V., Belghazi, I., Poole, B., Mastropietro, O., Lamb, A., Arjovsky, M., and Courville, A. (2016). Adversarially learned inference. *arXiv preprint arXiv:1606.00704*.

Finn, C., Christiano, P., Abbeel, P., and Levine, S. (2016). A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. (2016). Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep Learning. Book in preparation for MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680.

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., and Wierstra, D. (2015). Draw: A recurrent neural network for image generation. In *Proc. of the International Conference on Machine Learning*.

Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995). The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Larochelle, H. and Murray, I. (2011). The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 29–37.

References II

Le Roux, N. and Bengio, Y. (2008). Representational power of restricted boltzmann machines and deep belief networks. *Neural computation*, 20(6):1631–1649.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. (2016). Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*.

Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel Recurrent Neural Networks. In *Proc. of the International Conference on Machine Learning*.

Pfau, D. and Vinyals, O. (2016). Connecting generative adversarial networks and actor-critic methods. *arXiv preprint arXiv:1610.01945*.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. Technical report, DTIC Document.

Theis, L., Oord, A. v. d., and Bethge, M. (2015). A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*.

Wainwright, M. and Jordan, M. (2008). *Graphical Models, Exponential Families, and Variational Inference.* Now Publishers.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint*.