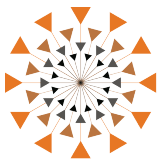
A Tutorial on Deep Probabilistic Generative Models

Ryan P. Adams
Princeton University

Machine Learning Summer School
Buenos Aires, Argentina
June 2018

lips.cs.princeton.edu
@ryan_p_adams



PRINCETON
LABORATORY FOR
INTELLIGENT
PROBABILISTIC
SYSTEMS

Tutorial Outline

What is generative modeling?

Recipes for flexible generative models

Algorithms for learning generative models from data

Variational autoencoder

Combining graphical models and neural networks

What is generative modeling?[1]

Today I will use the following definition of a generative model:

*A model is **generative** if it places a joint distribution over all observed dimensions of the data.*

[1] Generative modeling is surprisingly poorly defined in the literature!

Generative versus discriminative supervised learning

Generative models are often contrasted against *discriminative* models.

Consider a **supervised** learning task with features X and labels Y:

- Generative models want to learn $P(X, Y)$.
- Discriminative models want to learn $P(Y \mid X)$.

Philosophically, it's hard to justify learning $P(X, Y)$ if you just want $P(Y \mid X)$.[2]

> *"... one should solve the [classification] problem directly and never solve a more general problem as an intermediate step ... " Vapnik (1998)*

But there's so much more to life than supervised learning!

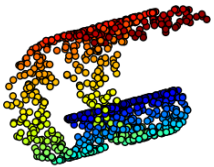
[2] See Ng and Jordan (2002) for a discussion.

Generative models: beyond $P(Y \mid X)$

What can you do with a generative model?

- Compute arbitrary conditionals and marginals.
- Compare the probabilities of different examples.
- Reduce the dimensionality of the data.
- Identify interpretable latent structure.
- Fantasize completely new data.

Dimensionality reduction



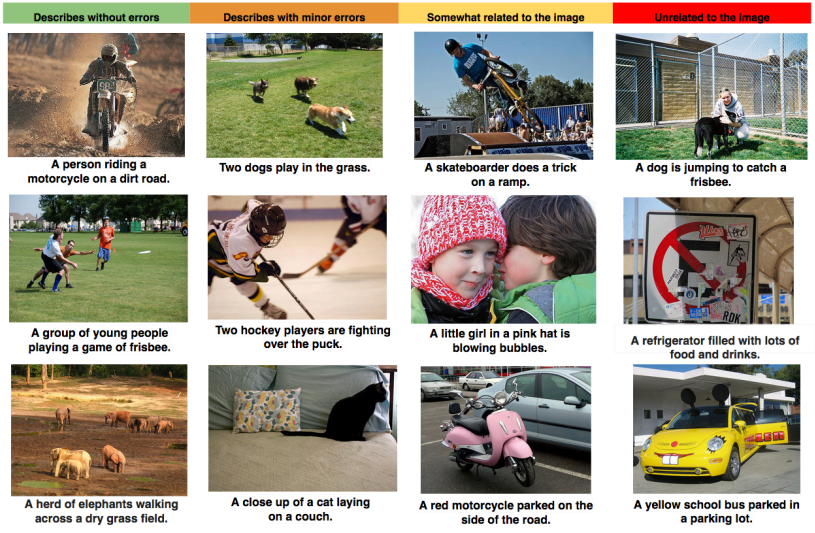
Denoising

Credit: *Wikipedia*

Synthesizing data

Credit: *Mescheder et al. (2017)*

Example: Image captioning



Credit: *Google AI Blog*, Vinyals et al. (2015)

Example: Image super-resolution

Credit: *Ledig et al. (2017)*

Example: Machine translation

Buenos Aires is beautiful this time of year.

↓

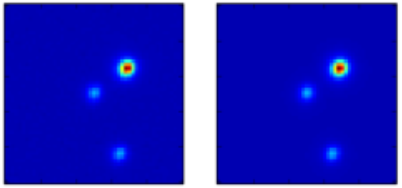
Buenos Aires es hermoso en esta época del año.

Example: Synthesizing faces

Credit: *Mescheder et al. (2017)*

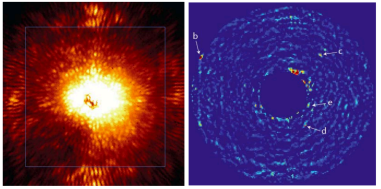
Example: Generative modeling in astronomy

Cataloging light sources



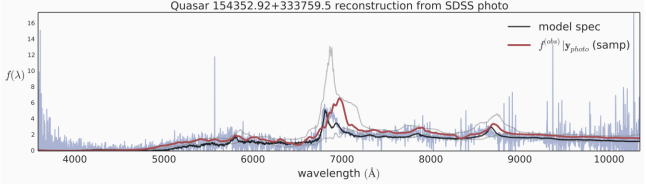
Credit: *Regier et al. (2015)*

Discovering exoplanets



Credit: *Fergus et al. (2014)*

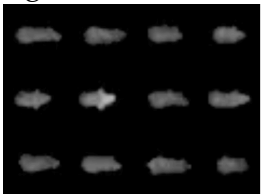
Identifying redshift in quasars



Credit: *Miller et al. (2015)*

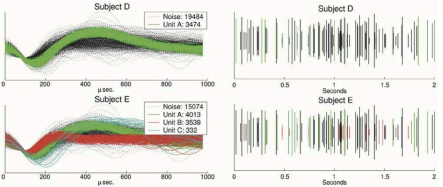
Example: Generative modeling in neuroscience

Modeling behavioral time series



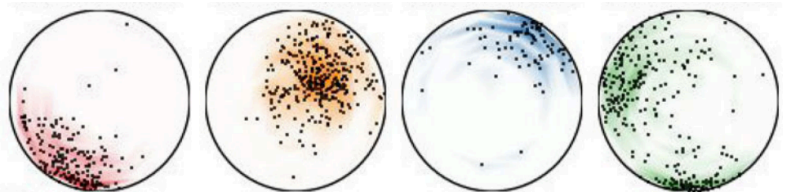
Credit: *Wiltschko et al. (2015)*

Spike sorting



Credit: *Wood and Black (2008)*

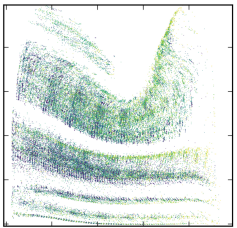
Identifying neural function

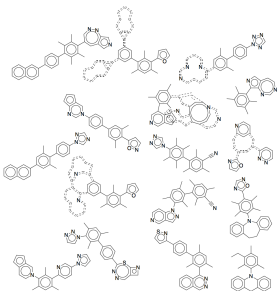


Credit: *Linderman et al. (2016)*

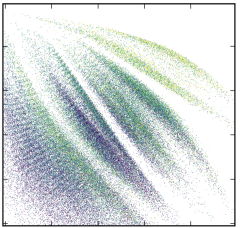
Example: Generative modeling in molecular design

Organic light-emitting diodes





Drug-like molecules



Credit: *Gómez-Bombarelli et al. (2016)*

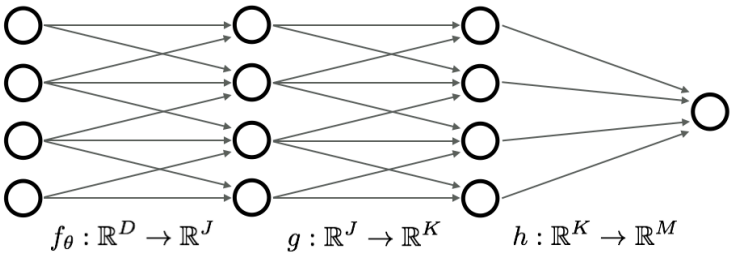
Generative modeling is density estimation

Generative modeling is the art and science of engineering a family of probability distributions that is simultaneously ***rich, parsimonious, and tractable****.*

Why *deep* generative models?

Deep neural networks are flexible function families:

- Useful for engineering highly parameterized distributions.
- Allow for "modest" nonlinearity in function approximation.
- Compositionality can lead to parsimony in latent representation.
- Structures such as convolution reflect good priors for many data.
- Extensive toolchains around optimization and automatic differentiation.
- A way to build nonparametric and semiparametric statistical models.



Tutorial Outline

What is generative modeling?

Recipes for flexible generative models

Algorithms for learning generative models from data

Variational autoencoder

Combining graphical models and neural networks

Design philosophies for flexible generative models

How to design a rich family of probability distributions?

Three basic recipes for using a flexible function $f_\theta(\cdot)$:

1. Apply a richly parameterized transformation to a simple random variable.

$$Z \sim \mathcal{N}(0, \mathbf{I}) \qquad\qquad X = f_\theta(Z)$$

2. Use a rich mixing distribution for a simple parametric family.

$$Z \sim \mathcal{N}(0, \mathbf{I}) \qquad\qquad X \sim \mathcal{N}(f_\theta(Z), \mathbf{\Sigma})$$

3. Specify a complicated distribution via its log density:

$$X \sim \frac{1}{\mathcal{Z}_\theta} \exp\{f_\theta(x)\} \qquad\qquad \mathcal{Z}_\theta = \int \exp\{f_\theta(x)\}\, dx$$

Recipe 1: Transform a simple random variable

Construct a family of densities $g_\theta(x)$ on $\mathbb{R}^K$ with parameters θ.

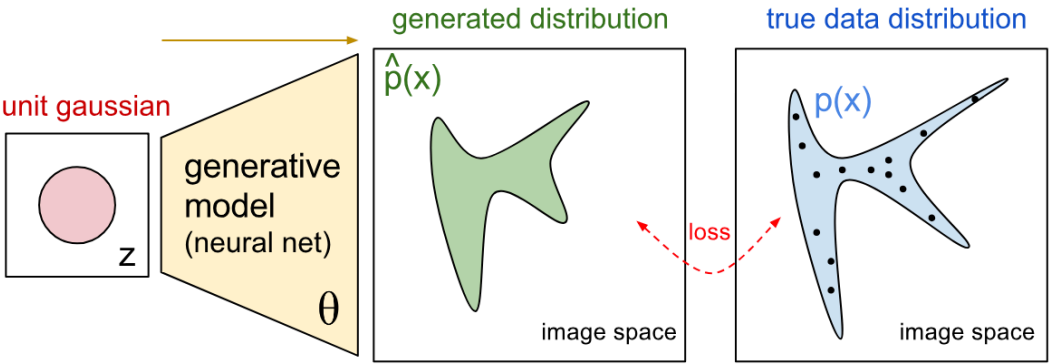
- Choose a simple continuous distribution on $\mathbb{R}^J$ with density $\pi(z)$.
- Parameterize a class of functions: $f_\theta : \mathbb{R}^J \to \mathbb{R}^K$.
- If $J = K$ and f_θ is bijective, then you get a density

$$g_\theta(x) = \pi(f_\theta^{-1}(x)\,)\,|\mathcal{J}[f_\theta^{-1}(x)\,]|$$

  where $\mathcal{J}[\cdot]$ is the Jacobian matrix.
- If f_θ is not bijective, then it may be very hard to compute the density $g_\theta(x)$.
- If $J < K$, it is probably necessary to add some noise after the transformation, but then you get potentially useful dimensionality reduction.
- Always very easy to fantasize data for given θ.

Recipe 1: Transform a simple random variable

$$g_\theta(x) = \pi(f_\theta^{-1}(x)\,)\,|\mathcal{J}[f_\theta^{-1}(x)\,]|$$



Credit: *OpenAI blog post on generative models*

Recipe 1: Transform a simple random variable

Classic Example: Factor Analysis and Principal Component Analysis

- Latent spherical Gaussian: $\pi = \mathcal{N}(0, \mathbf{I})$
- f_θ is a linear transformation with $J < K$:

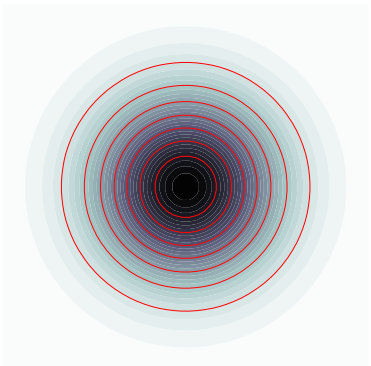
$$\theta \in \mathbb{R}^{K \times J} \qquad\qquad f_\theta(z) = \theta z$$

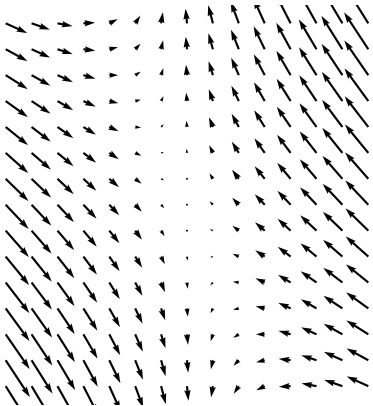
- Add diagonal noise to make covariance full rank.
- Classic dimensionality reduction technique.
- Roweis (1998), Tipping and Bishop (1999), Roweis and Ghahramani (1999).
- Many non-linear extensions to f_θ:
  - Neural networks (DeMers and Cottrell, 1993, Kramer, 1991, MacKay, 1995)
  - Gaussian processes (Lawrence, 2005)
  - Kernelization (Schölkopf et al., 1998)

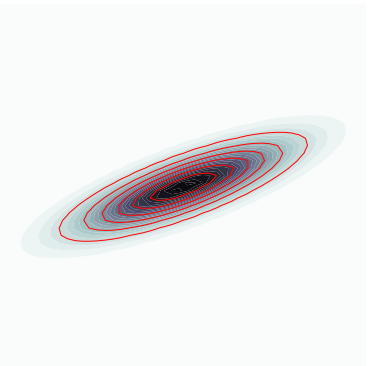
Recipe 1: Transform a simple random variable

Classic Example: Factor Analysis and Principal Component Analysis

$\pi(z) = \mathcal{N}(z \mid 0, \mathbf{I})$ $\quad f_\theta(z) = \theta z$ $\quad g_\theta(x)$







Recipe 1: Transform a simple random variable

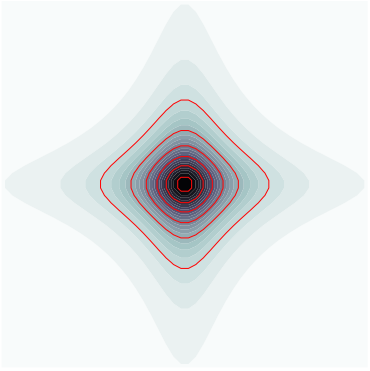
Classic Example: Independent Component Analysis (ICA)

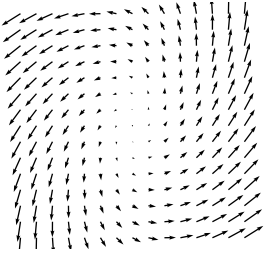
- Latent distribution continuous but non-Gaussian.
- Seeks to recover the invertible rotation that makes the data independent.
- Famous method for solving the "cocktail party problem."
- See Jutten and Herault (1991), Comon (1994), Hyvärinen and Oja (2000).
- Neural network extensions, e.g., Burel (1992), Pajunen et al. (1996)
- Kernelized version in Bach and Jordan (2002).

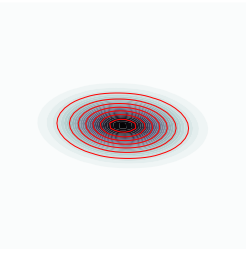
Recipe 1: Transform a simple random variable

Classic Example: Independent Component Analysis

$\pi(z) = \text{Cauchy}(z \mid 0, \mathbf{I})$ $f_\theta(z) = \theta z$ $g_\theta(x)$



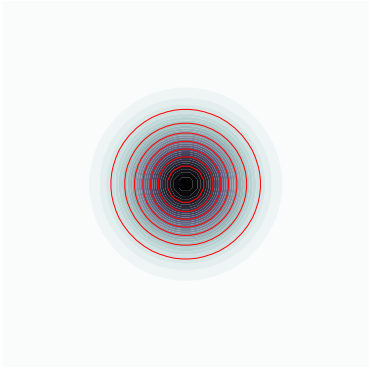


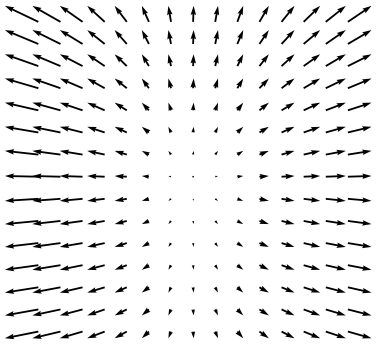


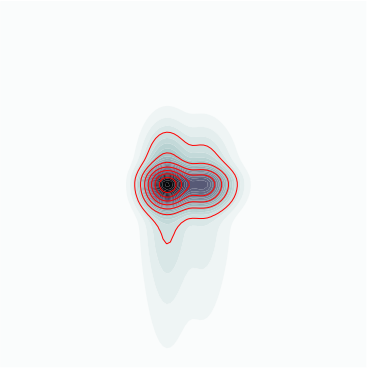
Recipe 1: Transform a simple random variable

Nonlinear transformation

$\pi(z) = \mathcal{N}(z \mid 0, \mathbf{I})$ $f_\theta^{-1}(z)$ $g_\theta(x)$



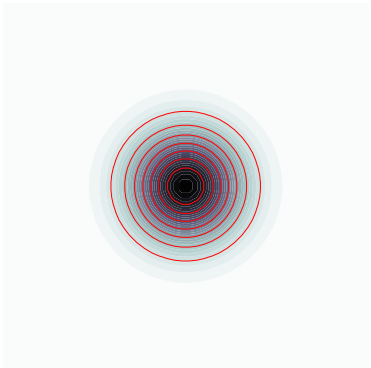


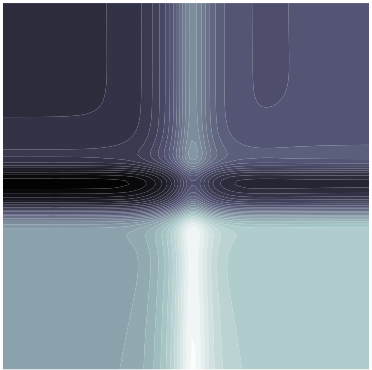


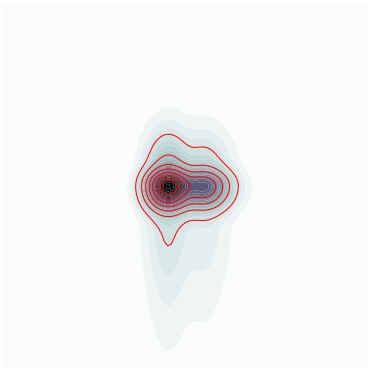
Recipe 1: Transform a simple random variable

Nonlinear transformation

$\pi(z) = \mathcal{N}(z \mid 0, \mathbf{I})$ $|\mathcal{J}[f_\theta^{-1}(z)]|$ $g_\theta(x)$

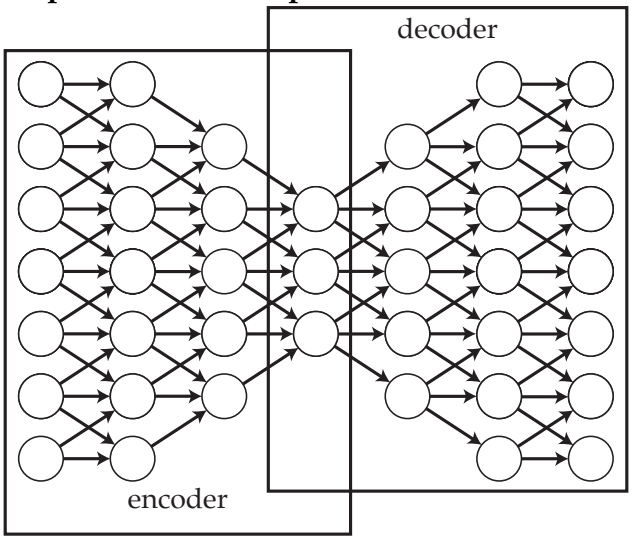






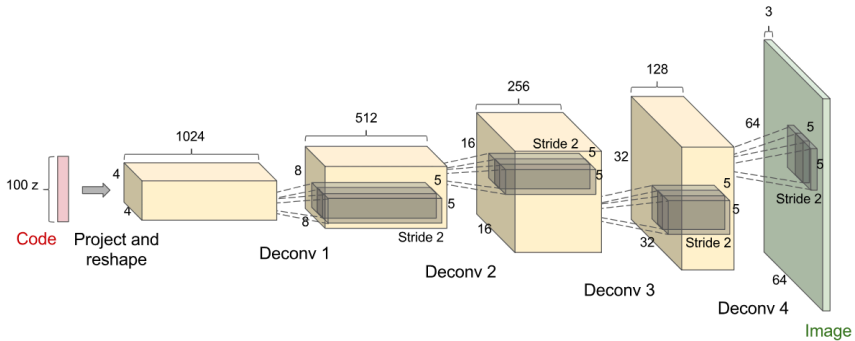
Recipe 1: Transform a simple random variable

Example: the decoder portion of an autoencoder



Recipe 1: Transform a simple random variable

Example: generative adversarial network (Goodfellow et al., 2014)
(DCGAN shown below, Radford et al. (2015))

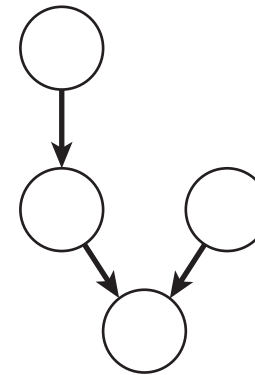


Credit: *OpenAI blog post on generative models*

Recipe 2: Mix a simple random variable

Construct a family of densities (or PMFs) $g_\theta(x)$ with parameters θ.

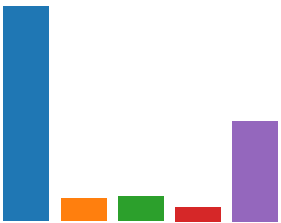
- Choose a family of simple distributions π_z, parameterized by z.
- The family π_z can be discrete, continuous, or both.
- Define a distribution $\psi_\theta(z)$ on z with parameters θ.
- Draw a z from ψ_θ, then $x \sim \pi_z$.
- Different ψ for every datum!
- Hard because we don't know z for any given example.
- Always easy to fantasize data for a given θ.



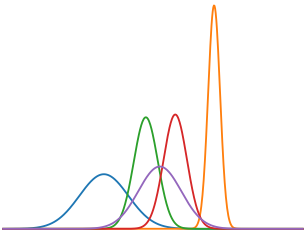
Recipe 2: Mix a simple random variable

Classic Example: Gaussian Mixture Model

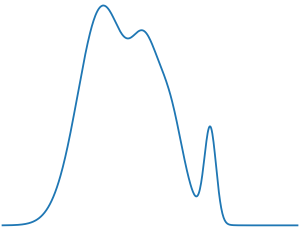
mixing distribution



components



$g_\theta(x)$



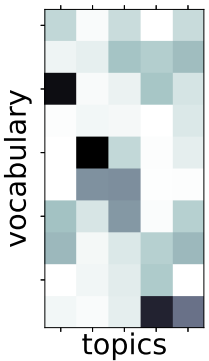
Recipe 2: Mix a simple random variable

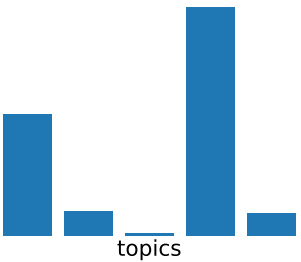
Classic Example: Latent Dirichlet Allocation (Blei et al., 2003)

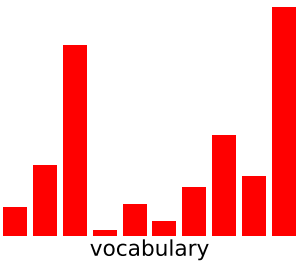
topics: distributions over vocabulary

per-document distribution over topics

$g_\theta(x)$: per-document distribution over vocabulary





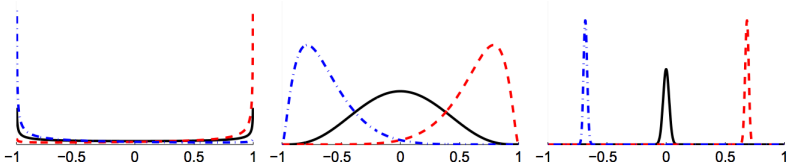


Recipe 2: Mix a simple random variable

Nonlinear Gaussian belief networks (Frey and Hinton, 1999, Neal, 1992)

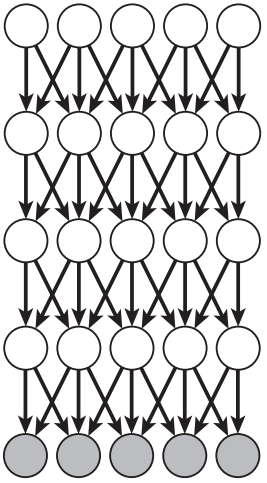
Each layer linearly transforms the previous layer, adds Gaussian noise and squashes through normal CDF.

$$\boldsymbol{z}_{t+1} = \Phi(\boldsymbol{W}\boldsymbol{z}_t + \boldsymbol{\epsilon}_t) \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{\Lambda})$$



(a) $\nu = \frac{1}{2}$ (b) $\nu = 5$ (c) $\nu = 1000$

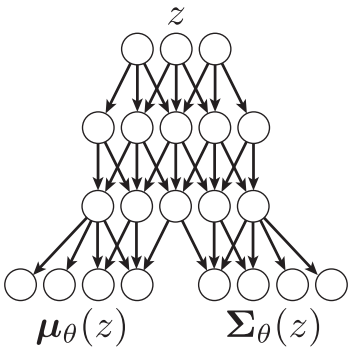
See Adams et al. (2010) for more details on the construction in this figure.



Recipe 2: Mix a simple random variable

Variational autoencoder (Kingma and Welling, 2014)
(more on this later)

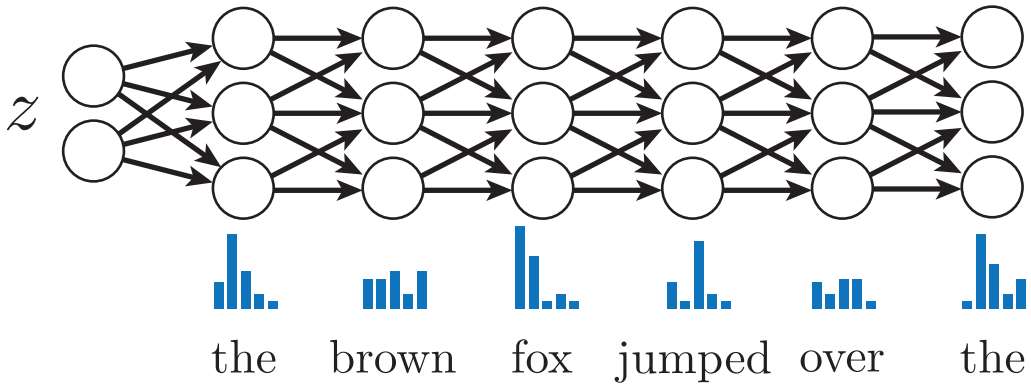
Parameterize the mean and (probably diagonal) covariance of a Gaussian via a feedforward neural network with random inputs.



Recipe 2: Mix a simple random variable

Variational autoencoder (Kingma and Welling, 2014)

Parameterize softmax logits via a recurrent neural network with random inputs.



Recipe 3: Specify a log density directly

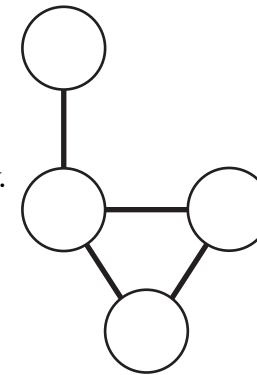
Construct a family of densities (or PMFs) $g_\theta(x)$ with parameters θ.

- Parametrize any scalar function $f_\theta(x)$.
- Exponentiate and normalize:

$$g_\theta(x) = \frac{1}{\mathcal{Z}_\theta} \exp\{f_\theta(x)\}$$

$$\mathcal{Z}_\theta = \int \exp\{f_\theta(x)\}$$

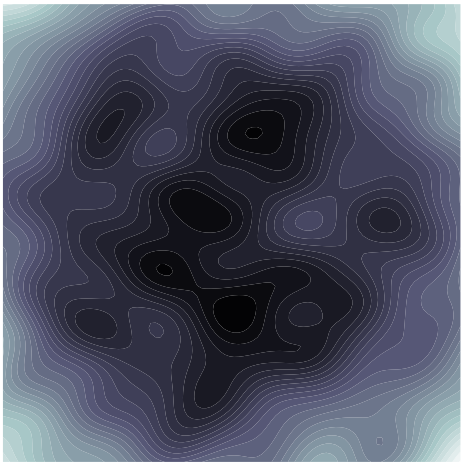
- Can now think about "goodness of configurations" directly.
- Often called *energy models* with $E_\theta(x) = -f_\theta(x)$.
- The partition function $\mathcal{Z}_\theta$ may be intractable.
- Typically requires Markov chain Monte Carlo to sample.

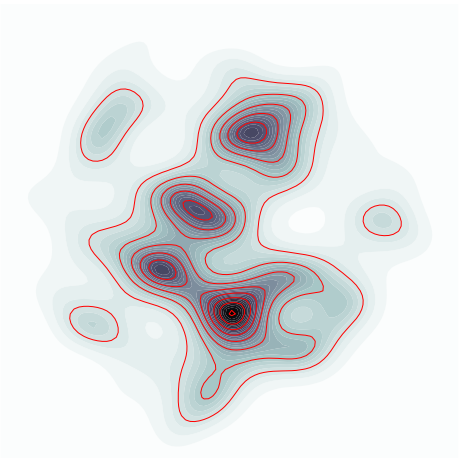


Recipe 3: Specify a log density directly

$f_\theta(x)$

$g_\theta(x) = \frac{1}{Z_\theta} \exp\{f_\theta(x)\}$





Recipe 3: Specify a log density directly

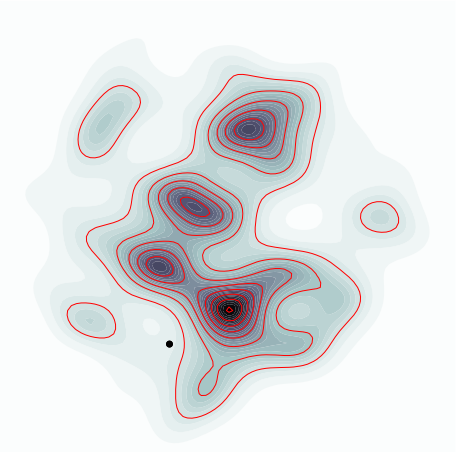
Markov chain Monte Carlo (MCMC):

- Random walk that converges to $g_\theta(x)$.
- Uses a stochastic operator $T(x' \leftarrow x)$.
- Ergodic and leave $g_\theta(x)$ invariant:

$$g_\theta(x) = \int g_\theta(x')\, T(x \leftarrow x')\, dx'$$

- Several common recipes:
  - Metropolis–Hastings
  - Gibbs sampling
  - Slice sampling
  - Hamiltonian Monte Carlo

$$g_\theta(x) = \tfrac{1}{Z_\theta} \exp\{f_\theta(x)\}$$



Recipe 3: Specify a log density directly

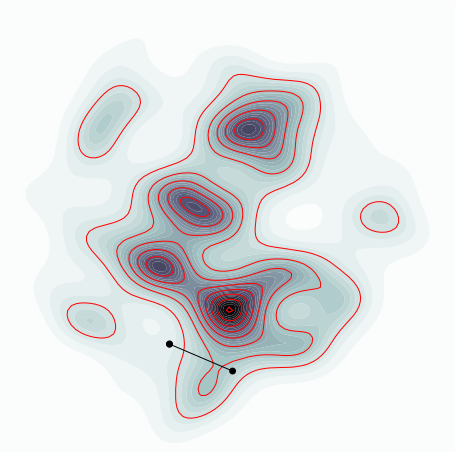
Markov chain Monte Carlo (MCMC):

- Random walk that converges to $g_\theta(x)$.
- Uses a stochastic operator $T(x' \leftarrow x)$.
- Ergodic and leave $g_\theta(x)$ invariant:

$$g_\theta(x) = \int g_\theta(x')\, T(x \leftarrow x')\, dx'$$

- Several common recipes:
  - Metropolis–Hastings
  - Gibbs sampling
  - Slice sampling
  - Hamiltonian Monte Carlo

$$g_\theta(x) = \tfrac{1}{Z_\theta} \exp\{f_\theta(x)\}$$



Recipe 3: Specify a log density directly

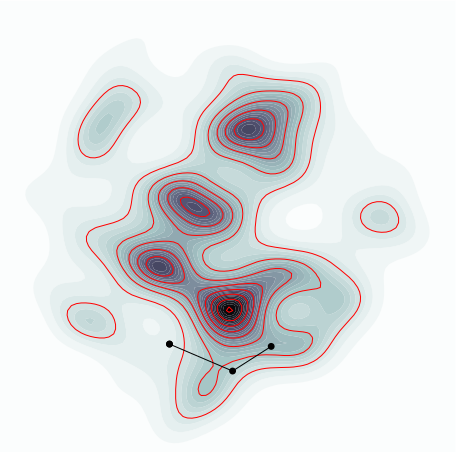
Markov chain Monte Carlo (MCMC):

- Random walk that converges to $g_\theta(x)$.
- Uses a stochastic operator $T(x' \leftarrow x)$.
- Ergodic and leave $g_\theta(x)$ invariant:

$$g_\theta(x) = \int g_\theta(x')\, T(x \leftarrow x')\, dx'$$

- Several common recipes:
  - Metropolis–Hastings
  - Gibbs sampling
  - Slice sampling
  - Hamiltonian Monte Carlo

$$g_\theta(x) = \tfrac{1}{Z_\theta} \exp\{f_\theta(x)\}$$



Recipe 3: Specify a log density directly

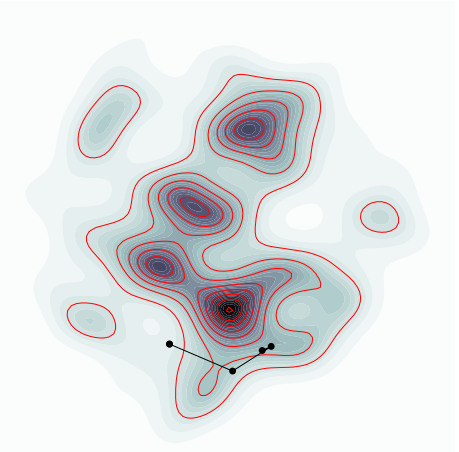
Markov chain Monte Carlo (MCMC):

- Random walk that converges to $g_\theta(x)$.
- Uses a stochastic operator $T(x' \leftarrow x)$.
- Ergodic and leave $g_\theta(x)$ invariant:

$$g_\theta(x) = \int g_\theta(x')\, T(x \leftarrow x')\, dx'$$

- Several common recipes:
  - Metropolis–Hastings
  - Gibbs sampling
  - Slice sampling
  - Hamiltonian Monte Carlo

$$g_\theta(x) = \tfrac{1}{Z_\theta} \exp\{f_\theta(x)\}$$



Recipe 3: Specify a log density directly

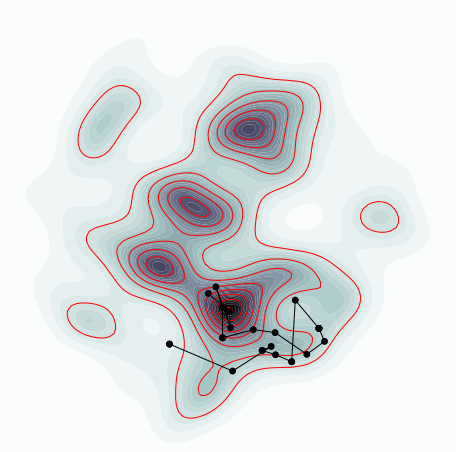
Markov chain Monte Carlo (MCMC):

- Random walk that converges to $g_\theta(x)$.
- Uses a stochastic operator $T(x' \leftarrow x)$.
- Ergodic and leave $g_\theta(x)$ invariant:

$$g_\theta(x) = \int g_\theta(x')\, T(x \leftarrow x')\, dx'$$

- Several common recipes:
  - Metropolis–Hastings
  - Gibbs sampling
  - Slice sampling
  - Hamiltonian Monte Carlo

$$g_\theta(x) = \tfrac{1}{Z_\theta} \exp\{f_\theta(x)\}$$



Recipe 3: Specify a log density directly

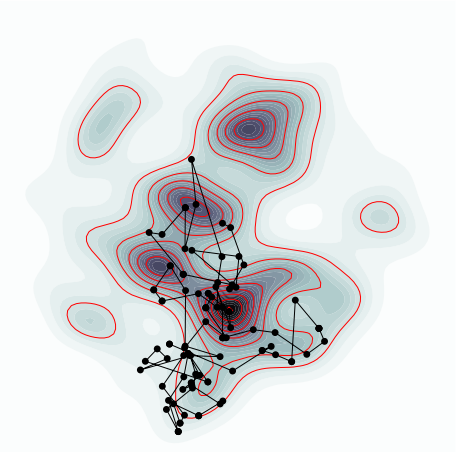
Markov chain Monte Carlo (MCMC):

- Random walk that converges to $g_\theta(x)$.
- Uses a stochastic operator $T(x' \leftarrow x)$.
- Ergodic and leave $g_\theta(x)$ invariant:

$$g_\theta(x) = \int g_\theta(x')\, T(x \leftarrow x')\, dx'$$

- Several common recipes:
  - Metropolis–Hastings
  - Gibbs sampling
  - Slice sampling
  - Hamiltonian Monte Carlo

$$g_\theta(x) = \tfrac{1}{Z_\theta} \exp\{f_\theta(x)\}$$



Recipe 3: Specify a log density directly

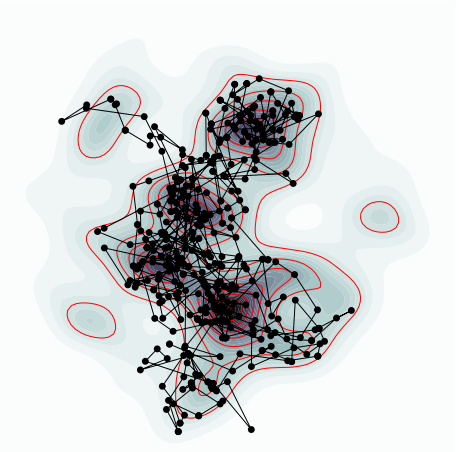
Markov chain Monte Carlo (MCMC):

- Random walk that converges to $g_\theta(x)$.
- Uses a stochastic operator $T(x' \leftarrow x)$.
- Ergodic and leave $g_\theta(x)$ invariant:

$$g_\theta(x) = \int g_\theta(x')\, T(x \leftarrow x')\, dx'$$

- Several common recipes:
  - Metropolis–Hastings
  - Gibbs sampling
  - Slice sampling
  - Hamiltonian Monte Carlo

$$g_\theta(x) = \tfrac{1}{Z_\theta} \exp\{f_\theta(x)\}$$

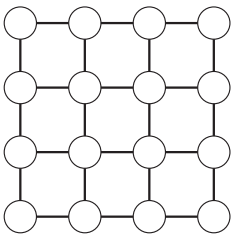


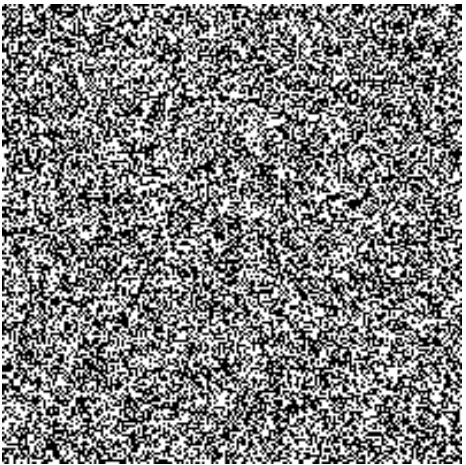
Recipe 3: Specify a log density directly

Example: Ising Model

- Classic model of ferromagnetism with binary "spins"
- Influential in computer vision
- Unary and pairwise potentials in energy:

$$E(x) = -f_\theta(x) \quad = -\sum_{ij} \theta_{ij} x_i x_j - \sum_i \theta_i x_i$$



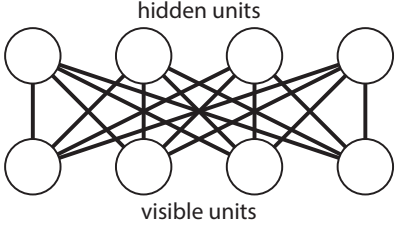


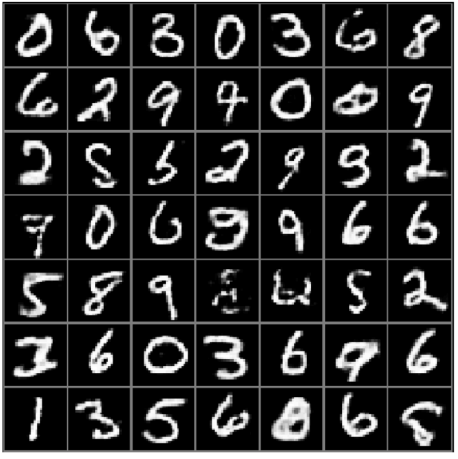
Credit: *Kai Zhang, Columbia*

Recipe 3: Specify a log density directly

Example: Restricted Boltzmann Machine (Freund and Haussler, 1992, Smolensky, 1986)

- Special case of the Ising model
- Bipartite: hidden and visible layers
- Fully connected between layers
- Typically trained with contrastive divergence



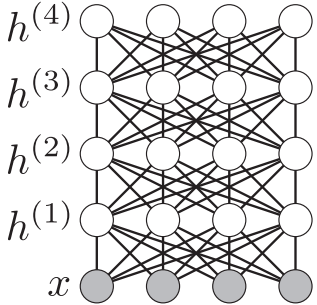


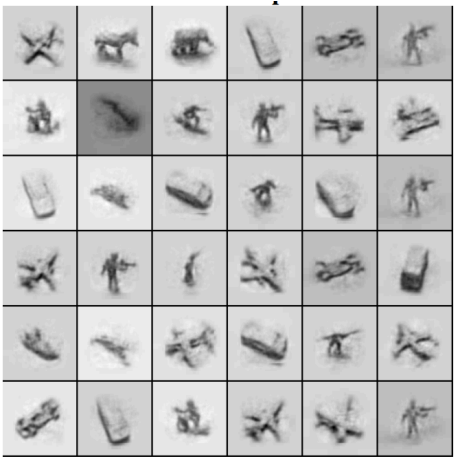
Credit: *Tieleman (2008)*

Recipe 3: Specify a log density directly

Example: Deep Boltzmann Machines (Salakhutdinov and Hinton, 2009)

- Special case of the Ising model
- k-partite: hidden and visible layers
- Fully connected between layers





Credit: *Salakhutdinov and Hinton (2009)*

Tutorial Outline

What is generative modeling?

Recipes for flexible generative models

Algorithms for learning generative models from data

Variational autoencoder

Combining graphical models and neural networks

Inductive principles for flexible generative models

We get N data $\{x_n\}_{n=1}^N$; how do we fit the parameters θ?

- Penalized maximum likelihood
- Computing a Bayesian posterior
- Score matching (Hyvärinen, 2005)
- Moment matching (e.g., Li et al. (2015))
- Maximum mean discrepancy (Dziugaite et al., 2015, Gretton et al., 2012)
- Pseudo-likelihood

MLE for invertible transformations

When $f_\theta(\cdot)$ is bijective, things are easy to reason about:

$$\ln P(\{x_n\}_{n=1}^N \mid \theta) = \sum_{n=1}^{N} \ln \pi(f_\theta^{-1}(x_n)) + \ln |\mathcal{J}[f_\theta^{-1}(x_n)]|$$

- Just use automatic differentiation to get gradients.
- Note: need the derivative of the Jacobian.
- The matrix $\mathcal{J}[f_\theta^{-1}(x_n)]$ may become nearly singular during training, causing numeric issues. See Rippel and Adams (2013) for a discussion.
- Real NVP (Dinh et al., 2016) parameterizes the matrix to have a Jacobian determinant that is easy to compute.

MLE for non-invertible transformations

$f_\theta(\cdot)$ **non-surjective:** some data have zero probability, i.e., infinite log loss
$f_\theta(\cdot)$ **non-injective:** data have multiple latent values

In general, you have to sum over the ways you could've gotten each x_n:

$$\ln P(\{x_n\}_{n=1}^N \mid \theta) = \sum_{n=1}^{N} \ln \int_{z:f_\theta(z)=x_n} \frac{\pi(z)}{|\mathcal{J}[f_\theta(z)]|} dz$$

Here we have to sum up all the ways we might've gotten each x_n.
Non-surjective $f_\theta(\cdot)$ means that the pre-image of x_n could be empty,
i.e., $\{z : f_\theta(z) = x_n\} = \emptyset$.

Statistical tests for non-invertible transformations

Sometimes you don't care about the density or the latent representation.
You just want fantasy data that has the same statistical properties as the real data.

Statistical tests for non-invertible transformations

Sometimes you don't care about the density or the latent representation.
You just want fantasy data that has the same statistical properties as the real data.

1. Cook up a function h that takes an x and produces a scalar.

Statistical tests for non-invertible transformations

Sometimes you don't care about the density or the latent representation.
You just want fantasy data that has the same statistical properties as the real data.

1. Cook up a function h that takes an x and produces a scalar.
2. Transform some real data with h and get the empirical distribution.

Statistical tests for non-invertible transformations

Sometimes you don't care about the density or the latent representation.
You just want fantasy data that has the same statistical properties as the real data.

1. Cook up a function h that takes an x and produces a scalar.
2. Transform some real data with h and get the empirical distribution.
3. Transform some fantasy data with h and get the empirical distribution.

Statistical tests for non-invertible transformations

Sometimes you don't care about the density or the latent representation.
You just want fantasy data that has the same statistical properties as the real data.

1. Cook up a function h that takes an x and produces a scalar.
2. Transform some real data with h and get the empirical distribution.
3. Transform some fantasy data with h and get the empirical distribution.
4. Use your favorite two-sample test to compare the distributions.

Statistical tests for non-invertible transformations

Sometimes you don't care about the density or the latent representation.
You just want fantasy data that has the same statistical properties as the real data.

1. Cook up a function h that takes an x and produces a scalar.
2. Transform some real data with h and get the empirical distribution.
3. Transform some fantasy data with h and get the empirical distribution.
4. Use your favorite two-sample test to compare the distributions.
5. Search for an $f_\theta(\cdot)$ that passes the test for many h in a big set $\mathcal{H}$.

Statistical tests for non-invertible transformations

Sometimes you don't care about the density or the latent representation.
You just want fantasy data that has the same statistical properties as the real data.

1. Cook up a function h that takes an x and produces a scalar.
2. Transform some real data with h and get the empirical distribution.
3. Transform some fantasy data with h and get the empirical distribution.
4. Use your favorite two-sample test to compare the distributions.
5. Search for an $f_\theta(\cdot)$ that passes the test for many h in a big set $\mathcal{H}$.

A nice kernel formalism for constructing tests and $\mathcal{H}$ is *maximum mean discrepancy* (Gretton et al., 2012). See also Dziugaite et al. (2015) and Huszar (2015).

Statistical tests for non-invertible transformations

Sometimes you don't care about the density or the latent representation.
You just want fantasy data that has the same statistical properties as the real data.

1. Cook up a function h that takes an x and produces a scalar.
2. Transform some real data with h and get the empirical distribution.
3. Transform some fantasy data with h and get the empirical distribution.
4. Use your favorite two-sample test to compare the distributions.
5. Search for an $f_\theta(\cdot)$ that passes the test for many h in a big set $\mathcal{H}$.

A nice kernel formalism for constructing tests and $\mathcal{H}$ is *maximum mean discrepancy* (Gretton et al., 2012). See also Dziugaite et al. (2015) and Huszar (2015).

You could also parameterize and learn the test with a *generative adversarial network* (Goodfellow et al., 2014). David Warde-Farley will talk about GANs next week.

MLE for latent variable models

Like the non-injective transformation case, the mixing case requires integrating over latent hypotheses:

$$\ln P(\{x_n\}_{n=1}^N \mid \theta) = \sum_{n=1}^N \ln \int P(x_n, z_n \mid \theta)\, dz_n = \sum_{n=1}^N \ln \int P(x_n \mid z_n, \theta)\, P(z_n \mid \theta)\, dz$$

Generally, three ways to do this kind of integral in ML:

- Addition – easy to do expectation maximization with discrete latent variables
- Quadrature – good rates in low dimensions, but bad in high dimensions
- Monte Carlo – approximate the integral with a sample mean
- Variational methods – approximate pieces with more tractable distributions

MLE with latent variables: expectation maximization

Initialize $\theta^{(0)}$ to a reasonable starting point, then iterate:

- **E-step** – Compute expected complete-data log likelihood under $\theta^{(t)}$:

$$Q(\theta \mid \theta^{(t)}) = \sum_{n=1}^{N} \mathbb{E}_{z_n \mid x_n, \theta^{(t)}} [\ln P(x_n, z_n \mid \theta)]$$

- **M-step** – Maximize this expected log likelihood with respect to θ:

$$\theta^{(t+1)} = \arg\max_{\theta} Q(\theta \mid \theta^{(t)})$$

That expectation may be just as hard as the marginal likelihood, however.

MLE for latent variable models: Monte Carlo EM

One approach to the integral is to use Monte Carlo. Recall:

$$\int \pi(z) f(z)\, dz = \mathbb{E}[f(z)] \approx \frac{1}{M} \sum_{m=1}^{M} f(z^{(m)}) \quad \text{where} \quad z^{(m)} \sim \pi$$

Initialize $\theta^{(0)}$ to a reasonable starting point, then iterate:

- **E-step** – Compute expected complete-data log likelihood under $\theta^{(t)}$, using M samples from the conditional on z_n:

$$Q(\theta \mid \theta^{(t)}) = \frac{1}{M} \sum_{n=1}^{N} \sum_{m=1}^{M} \ln P(x_n, z_n^{(m)} \mid \theta)$$

- **M-step** – Maximize this expected log likelihood with respect to θ:

$$\theta^{(t+1)} = \arg\max_{\theta} Q(\theta \mid \theta^{(t)})$$

MLE for latent variable models: Variational EM

Introduce a tractable (typically factored) distribution family on the $\{z_n\}_{n=1}^N$:

$$q_{\boldsymbol{\gamma}}(\{z_n\}_{n=1}^N) = \prod_{n=1}^N q_{\gamma_n}(z_n)$$

Jensen's inequality lets us lower bound the marginal likelihood:

$$\ln \int q_{\gamma_n}(z_n) \frac{P(x_n, z_n \mid \theta)}{q_{\gamma_n}(z_n)} \, dz_n \geq \int q_{\gamma_n}(z_n) \ln \frac{P(x_n, z_n \mid \theta)}{q_{\gamma_n}(z_n)} \, dz_n$$

Alternate between maximizing with respect to $\boldsymbol{\gamma}$ and θ.

If the $q_{\gamma_n}(z_n)$ family contains $P(z_n \mid x_n, \theta)$ then it's just regular EM.
If not, then it provides a coherent way to approximate the difficult expectation.
More on this later when we discuss variational autoencoders in detail.

MLE for energy models

"Energy models" specify the density directly via its log:

$$g_\theta(x) = \frac{1}{\mathcal{Z}_\theta} \exp\{f_\theta(x)\} \qquad \mathcal{Z}_\theta = \int \exp\{f_\theta(x)\}$$

We generally can't compute the partition function $\mathcal{Z}_\theta$:

$$\ln P(\{x_n\}_{n=1}^N \mid \theta) = \left[\sum_{n=1}^N f_\theta(x_n)\right] - N \ln \mathcal{Z}_\theta$$

You really do have to account for the partition function in learning.
$\mathcal{Z}_\theta$ prevents the model from assigning high probability everywhere!

MLE for energy models: contrastive divergence

$$\frac{\partial}{\partial\theta}\ln P(\{x_n\}_{n=1}^N \mid \theta) = \left[\sum_{n=1}^{N}\frac{\partial}{\partial\theta}f_\theta(x_n)\right] - N\frac{\partial}{\partial\theta}\ln\int \exp\{f_\theta(x)\}\,dx$$

MLE for energy models: contrastive divergence

$$\frac{\partial}{\partial\theta}\ln P(\{x_n\}_{n=1}^N \mid \theta) = \left[\sum_{n=1}^N \frac{\partial}{\partial\theta} f_\theta(x_n)\right] - N\frac{\partial}{\partial\theta}\ln \int \exp\{f_\theta(x)\}\,dx$$

$$= \left[\sum_{n=1}^N \frac{\partial}{\partial\theta} f_\theta(x_n)\right] - N\left(\int \exp\{f_\theta(x)\}\,dx\right)^{-1}\frac{\partial}{\partial\theta}\int \exp\{f_\theta(x)\}\,dx$$

MLE for energy models: contrastive divergence

$$\frac{\partial}{\partial\theta}\ln P(\{x_n\}_{n=1}^N \mid \theta) = \left[\sum_{n=1}^N \frac{\partial}{\partial\theta} f_\theta(x_n)\right] - N\frac{\partial}{\partial\theta}\ln \int \exp\{f_\theta(x)\}\,dx$$

$$= \left[\sum_{n=1}^N \frac{\partial}{\partial\theta} f_\theta(x_n)\right] - N\left(\int \exp\{f_\theta(x)\}\,dx\right)^{-1}\frac{\partial}{\partial\theta}\int \exp\{f_\theta(x)\}\,dx$$

$$= \left[\sum_{n=1}^N \frac{\partial}{\partial\theta} f_\theta(x_n)\right] - N\frac{1}{\mathcal{Z}_\theta}\int \frac{\partial}{\partial\theta}\exp\{f_\theta(x)\}\,dx$$

MLE for energy models: contrastive divergence

$$
\begin{aligned}
\frac{\partial}{\partial\theta}\ln P(\{x_n\}_{n=1}^N \mid \theta) &= \left[\sum_{n=1}^N \frac{\partial}{\partial\theta} f_\theta(x_n)\right] - N\frac{\partial}{\partial\theta}\ln \int \exp\{f_\theta(x)\}\,dx \\
&= \left[\sum_{n=1}^N \frac{\partial}{\partial\theta} f_\theta(x_n)\right] - N\left(\int \exp\{f_\theta(x)\}\,dx\right)^{-1}\frac{\partial}{\partial\theta}\int \exp\{f_\theta(x)\}\,dx \\
&= \left[\sum_{n=1}^N \frac{\partial}{\partial\theta} f_\theta(x_n)\right] - N\frac{1}{\mathcal{Z}_\theta}\int \frac{\partial}{\partial\theta}\exp\{f_\theta(x)\}\,dx \\
&= \left[\sum_{n=1}^N \frac{\partial}{\partial\theta} f_\theta(x_n)\right] - N\int \frac{1}{\mathcal{Z}_\theta}\exp\{f_\theta(x)\}\frac{\partial}{\partial\theta} f_\theta(x)\,dx
\end{aligned}
$$

MLE for energy models: contrastive divergence

$$
\begin{aligned}
\frac{\partial}{\partial\theta}\ln P(\{x_n\}_{n=1}^N \mid \theta) &= \left[\sum_{n=1}^N \frac{\partial}{\partial\theta} f_\theta(x_n)\right] - N\frac{\partial}{\partial\theta}\ln \int \exp\{f_\theta(x)\}\,dx \\
&= \left[\sum_{n=1}^N \frac{\partial}{\partial\theta} f_\theta(x_n)\right] - N\left(\int \exp\{f_\theta(x)\}\,dx\right)^{-1}\frac{\partial}{\partial\theta}\int \exp\{f_\theta(x)\}\,dx \\
&= \left[\sum_{n=1}^N \frac{\partial}{\partial\theta} f_\theta(x_n)\right] - N\frac{1}{\mathcal{Z}_\theta}\int \frac{\partial}{\partial\theta}\exp\{f_\theta(x)\}\,dx \\
&= \left[\sum_{n=1}^N \frac{\partial}{\partial\theta} f_\theta(x_n)\right] - N\int \frac{1}{\mathcal{Z}_\theta}\exp\{f_\theta(x)\}\frac{\partial}{\partial\theta} f_\theta(x)\,dx \\
&= N\left(\mathbb{E}_{\text{data}}\left[\frac{\partial}{\partial\theta} f_\theta(x)\right] - \mathbb{E}_{\text{model}}\left[\frac{\partial}{\partial\theta} f_\theta(x)\right]\right)
\end{aligned}
$$

MLE for energy models: contrastive divergence

Gradient is the difference between two expectations:

$$\frac{1}{N}\frac{\partial}{\partial\theta}\ln P(\{x_n\}_{n=1}^N \mid \theta) = \mathbb{E}_{\text{data}}\left[\frac{\partial}{\partial\theta} f_\theta(x)\right] - \mathbb{E}_{\text{model}}\left[\frac{\partial}{\partial\theta} f_\theta(x)\right]$$

MLE for energy models: contrastive divergence

Gradient is the difference between two expectations:

$$\frac{1}{N}\frac{\partial}{\partial\theta}\ln P(\{x_n\}_{n=1}^N \mid \theta) = \mathbb{E}_{\text{data}}\left[\frac{\partial}{\partial\theta} f_\theta(x)\right] - \mathbb{E}_{\text{model}}\left[\frac{\partial}{\partial\theta} f_\theta(x)\right]$$

- Use Monte Carlo for the second term by generating fantasy data?

MLE for energy models: contrastive divergence

Gradient is the difference between two expectations:

$$\frac{1}{N}\frac{\partial}{\partial\theta}\ln P(\{x_n\}_{n=1}^N \mid \theta) = \mathbb{E}_{\text{data}}\left[\frac{\partial}{\partial\theta} f_\theta(x)\right] - \mathbb{E}_{\text{model}}\left[\frac{\partial}{\partial\theta} f_\theta(x)\right]$$

- Use Monte Carlo for the second term by generating fantasy data?
- Bad news: generating data is hard, have to use Markov chain Monte Carlo.

MLE for energy models: contrastive divergence

Gradient is the difference between two expectations:

$$\frac{1}{N}\frac{\partial}{\partial\theta}\ln P(\{x_n\}_{n=1}^N \mid \theta) = \mathbb{E}_{\text{data}}\left[\frac{\partial}{\partial\theta} f_\theta(x)\right] - \mathbb{E}_{\text{model}}\left[\frac{\partial}{\partial\theta} f_\theta(x)\right]$$

- Use Monte Carlo for the second term by generating fantasy data?
- Bad news: generating data is hard, have to use Markov chain Monte Carlo.
- **Contrastive divergence** – start at one of the data and run K steps of MCMC (Hinton, 2002). For RBMs, good features, bad densities.

MLE for energy models: contrastive divergence

Gradient is the difference between two expectations:

$$\frac{1}{N}\frac{\partial}{\partial\theta}\ln P(\{x_n\}_{n=1}^N \mid \theta) = \mathbb{E}_{\text{data}}\left[\frac{\partial}{\partial\theta}f_\theta(x)\right] - \mathbb{E}_{\text{model}}\left[\frac{\partial}{\partial\theta}f_\theta(x)\right]$$

- Use Monte Carlo for the second term by generating fantasy data?
- Bad news: generating data is hard, have to use Markov chain Monte Carlo.
- **Contrastive divergence** – start at one of the data and run K steps of MCMC (Hinton, 2002). For RBMs, good features, bad densities.
- **Persistent contrastive divergence** – don't restart the Markov chain between updates (Tieleman, 2008), often does better.

Training a binary RBM with CD

- Binary data $\boldsymbol{x} \in \{0,1\}^D$
- Binary hidden units $\boldsymbol{h} \in \{0,1\}^J$
- Parameters: weight matrix $\boldsymbol{W} \in \mathbb{R}^{D \times J}$, biases $\boldsymbol{b}_{\text{vis}} \in \mathbb{R}^D$ and $\boldsymbol{b}_{\text{hid}} \in \mathbb{R}^J$
- Energy function:

$$E(\boldsymbol{x}, \boldsymbol{h}\,;\, \boldsymbol{W}, \boldsymbol{b}_{\text{vis}}, \boldsymbol{b}_{\text{hid}}) = -\boldsymbol{x}^\top \boldsymbol{W}\boldsymbol{h} - \boldsymbol{x}^\top \boldsymbol{b}_{\text{vis}} - \boldsymbol{h}^\top \boldsymbol{b}_{\text{hid}}$$

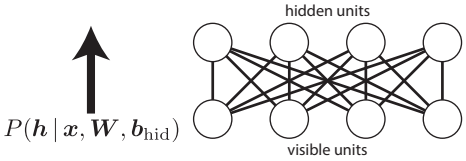
- Hidden given visible:

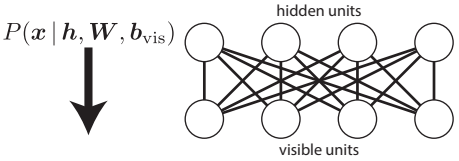
$$P(\boldsymbol{h} \,|\, \boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}_{\text{hid}}) = \prod_{j=1}^{J} \frac{1}{1 + \exp\{-\boldsymbol{W}^\top \boldsymbol{x} - \boldsymbol{b}_{\text{hid}}\}}$$

- Visible given hidden:

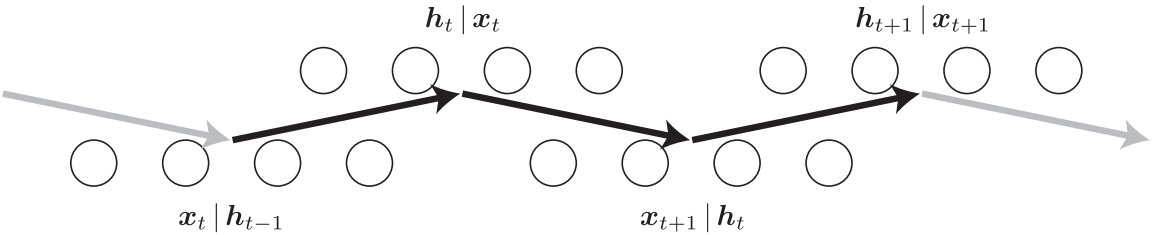
$$P(\boldsymbol{x} \,|\, \boldsymbol{h}, \boldsymbol{W}, \boldsymbol{b}_{\text{vis}}) = \prod_{d=1}^{D} \frac{1}{1 + \exp\{-\boldsymbol{W}\boldsymbol{h} - \boldsymbol{b}_{\text{vis}}\}}$$

Training a binary RBM with CD

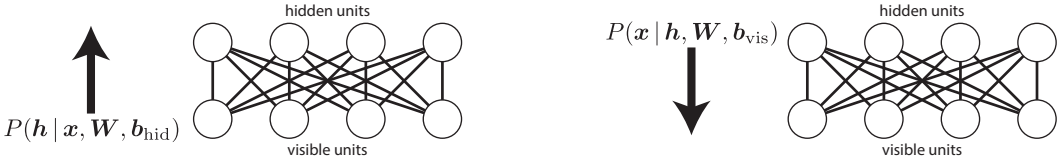




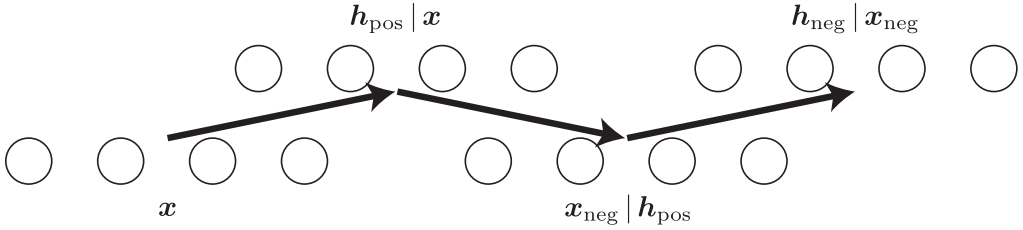
Bipartite structure of RBM makes Gibbs sampling easy.



Training a binary RBM with CD



Contrastive divergence: start at data and Gibbs sample K times.



Training a binary RBM with CD

Input: Parameters $\boldsymbol{W}$, $(\boldsymbol{b}_{\text{vis}}, \boldsymbol{b}_{\text{hid}})$; input $\boldsymbol{x} \in \{0,1\}^D$; learning rate $\alpha > 0$
Output: Updated parameters $\boldsymbol{W}'$, $\boldsymbol{b}'_{\text{vis}}$ $\boldsymbol{b}'_{\text{hid}}$
$\boldsymbol{h}_{\text{pos}} \sim \boldsymbol{h} \,|\, \boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}_{\text{hid}}$ ▷ Sample hiddens given visibles.
$\boldsymbol{h}_{\text{neg}} \leftarrow \boldsymbol{h}_{\text{pos}}$ ▷ Initialize negative hiddens.
for $t = 1 \ldots K$ **do**
    $\boldsymbol{x}_{\text{neg}} \leftarrow \boldsymbol{x} \,|\, \boldsymbol{h}_{\text{neg}}, \boldsymbol{W}, \boldsymbol{b}_{\text{vis}}$ ▷ Sample fantasy data.
    $\boldsymbol{h}_{\text{neg}} \leftarrow \boldsymbol{h} \,|\, \boldsymbol{x}_{\text{neg}}, \boldsymbol{W}, \boldsymbol{b}_{\text{hid}}$ ▷ Sample hiddens for fantasy data.
end for
$\boldsymbol{W}' \leftarrow \boldsymbol{W} + \alpha(\boldsymbol{x}\boldsymbol{h}_{\text{pos}}^{\mathsf{T}} - \boldsymbol{x}_{\text{neg}}\boldsymbol{h}_{\text{neg}}^{\mathsf{T}})$ ▷ Approximate stochastic gradient update.
$\boldsymbol{b}'_{\text{vis}} \leftarrow \boldsymbol{b}_{\text{vis}} + \alpha(\boldsymbol{x}_{\text{pos}} - \boldsymbol{x}_{\text{neg}})$
$\boldsymbol{b}'_{\text{hid}} \leftarrow \boldsymbol{b}_{\text{hid}} + \alpha(\boldsymbol{h}_{\text{pos}} - \boldsymbol{h}_{\text{neg}})$

Score matching for energy models

Hyvärinen (2005) proposed an alternative way to avoid the partition function.

Score function: gradient of the log likelihood **with respect to the data**.

$$\psi(x;\theta) = \frac{\partial}{\partial x} \ln P(x \mid \theta) = \frac{\partial}{\partial x} \left(f_\theta(x) - \ln \mathcal{Z}_\theta \right) = \frac{\partial}{\partial x} f_\theta(x)$$

Score matching for energy models

Hyvärinen (2005) proposed an alternative way to avoid the partition function.

Score function: gradient of the log likelihood **with respect to the data**.

$$\psi(x;\theta) = \frac{\partial}{\partial x} \ln P(x \mid \theta) = \frac{\partial}{\partial x} \left(f_\theta(x) - \ln \mathcal{Z}_\theta \right) = \frac{\partial}{\partial x} f_\theta(x)$$

Fitting a score function:

- Given observed data $\{x_n\}_{n=1}^N$, construct a density estimate $p_{\text{data}}(x)$.
- Denote the "empirical score function" of this density estimate as $\psi_{\text{data}}(x)$.
- Model and empirical score functions should be similar:

$$J(\theta) = \frac{1}{2} \int p_{\text{data}}(x) \, ||\psi(x;\theta) - \psi_{\text{data}}(x)||^2 \, dx$$

Score matching for energy models

Hyvärinen (2005) showed that this objective can be simplified:

$$
\begin{aligned}
J(\theta) &= \frac{1}{2} \int p_{\text{data}}(x) ||\boldsymbol{\psi}(x;\theta) - \boldsymbol{\psi}_{\text{data}}(x)||^2 \, dx \\
&= \int p_{\text{data}}(x) \left[\mathbf{1}^\top \boldsymbol{\psi}(x;\theta) + \frac{1}{2} ||\boldsymbol{\psi}(x;\theta)||^2 \right] dx + \text{const}
\end{aligned}
$$

Score matching for energy models

Hyvärinen (2005) showed that this objective can be simplified:

$$
\begin{aligned}
J(\theta) &= \frac{1}{2} \int p_{\text{data}}(x) ||\boldsymbol{\psi}(x;\theta) - \boldsymbol{\psi}_{\text{data}}(x)||^2 \, dx \\
&= \int p_{\text{data}}(x) \left[\mathbf{1}^\top \boldsymbol{\psi}(x;\theta) + \frac{1}{2} ||\boldsymbol{\psi}(x;\theta)||^2 \right] dx + \text{const}
\end{aligned}
$$

We don't actually need $\boldsymbol{\psi}_{\text{data}}(x)$ and can use the raw empirical $p_{\text{data}}(x)$:

$$
\tilde{J}(\theta) = \frac{1}{N} \sum_{n=1} \mathbf{1}^\top \boldsymbol{\psi}(x_n;\theta) + \frac{1}{2} ||\boldsymbol{\psi}(x_n;\theta)||^2
$$

If the model is identifiable, $\hat{\theta} = \arg\min_\theta \tilde{J}(\theta)$ is a consistent estimator.

Tutorial Outline

What is generative modeling?

Recipes for flexible generative models

Algorithms for learning generative models from data

Variational autoencoder

Combining graphical models and neural networks

A closer look at the variational autoencoder

Consider a latent variable model that combines Recipes 1 and 2:

Basic VAE Generative Model (Kingma and Welling, 2014)

Spherical Gaussian latent variable:

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

Transform with a neural network to parameterize another Gaussian:

$$x \,|\, z, \theta \sim \mathcal{N}(\boldsymbol{\mu}_\theta(z), \boldsymbol{\Sigma}_\theta(z))$$

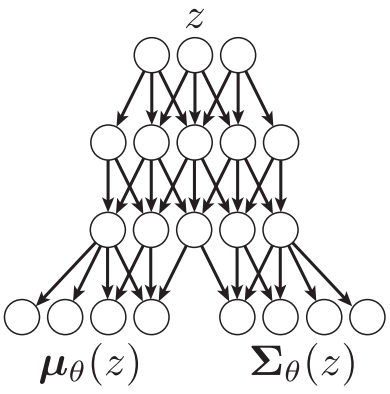
Given some data $\{x_n\}_{n=1}^N$, maximize the likelihood with respect to θ:

$$\theta^\star = \arg\max_\theta \sum_{n=1}^N \ln \int \mathcal{N}(x_n \,|\, \boldsymbol{\mu}_\theta(z_n), \boldsymbol{\Sigma}_\theta(z_n))\, \mathcal{N}(z_n \,|\, 0, \mathbf{I})\, dz_n$$

Variational autoencoder

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

$$x \,|\, z, \theta \sim \mathcal{N}(\boldsymbol{\mu}_\theta(z), \boldsymbol{\Sigma}_\theta(z))$$



Credit: *OpenAI blog post on generative models*

Learning the VAE model with mean-field

We want to solve this:

$$\theta^\star = \arg\max_\theta \sum_{n=1}^{N} \ln \int \mathcal{N}(x_n \mid \boldsymbol{\mu}_\theta(z_n), \boldsymbol{\Sigma}_\theta(z_n))\, \mathcal{N}(z_n \mid 0, \mathbf{I})\, dz_n$$

- Have to estimate the z_n associated with each x_n.
- Can't use vanilla EM because $P(z_n \mid x_n, \theta)$ is complicated.
- Approximate $P(z_n \mid x_n, \theta)$ with $\mathcal{N}(z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n)$.
- Compute the *evidence lower bound* using Jensen's inequality:

$$\ln \int \mathcal{N}(x_n \mid \boldsymbol{\mu}_\theta(z_n), \boldsymbol{\Sigma}_\theta(z_n))\, \mathcal{N}(z_n \mid 0, \mathbf{I})\, dz_n$$
$$\geq \int \mathcal{N}(z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n) \ln \frac{\mathcal{N}(x_n \mid \boldsymbol{\mu}_\theta(z_n), \boldsymbol{\Sigma}_\theta(z_n))\, \mathcal{N}(z_n \mid 0, \mathbf{I})}{\mathcal{N}(z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n)} dz_n$$

Maximize the VAE mean-field objective directly?

We could try to maximize this objective directly:

$$\mathcal{L}(\theta, \{\boldsymbol{m}_n, \boldsymbol{V}_n\}_{n=1}^N) = \sum_{n=1}^N \int \mathcal{N}(z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n) \ln \frac{\mathcal{N}(x_n \mid \boldsymbol{\mu}_\theta(z_n), \boldsymbol{\Sigma}_\theta(z_n))\, \mathcal{N}(z_n \mid 0, \mathbf{I})}{\mathcal{N}(z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n)} dz_n$$

$$= \sum_{n=1}^N \left[\int \mathcal{N}(z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n) \ln \mathcal{N}(x_n \mid \boldsymbol{\mu}_\theta(z_n), \boldsymbol{\Sigma}_\theta(z_n))\, dz_n \right.$$

$$\left. + \int \mathcal{N}(z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n) \ln \frac{\mathcal{N}(z_n \mid 0, \mathbf{I})}{\mathcal{N}(z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n)} dz_n \right]$$

$$= \sum_{n=1}^N \left(\mathbb{E}_{z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n} \left[\mathcal{N}(x_n \mid \boldsymbol{\mu}_\theta(z_n), \boldsymbol{\Sigma}_\theta(z_n)) \right] - \mathrm{KL}\left[\mathcal{N}(z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n) || \mathcal{N}(z_n \mid 0, \mathbf{I}) \right] \right)$$

Maximize the VAE mean-field objective directly?

We could try to maximize this objective directly:

$$\sum_{n=1}^{N} \underbrace{\mathbb{E}_{z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n}\left[\mathcal{N}(x_n \mid \boldsymbol{\mu}_\theta(z_n), \boldsymbol{\Sigma}_\theta(z_n))\right]}_{\text{expected complete-data log likelihood}} - \underbrace{\mathrm{KL}\left[\mathcal{N}(z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n) || \mathcal{N}(z_n \mid 0, \mathbf{I})\right]}_{\text{difference between approximation and prior (easy)}}$$

Annoying because

- The number of optimized dimensions scales with N.
- We have to perform an optimization to make an out-of-sample inference.
- Computing the expected complete data log likelihood looks hard.

Maximize the VAE mean-field objective directly?

Zooming in on the expected complete-data log likelihood:

$$\mathbb{E}_{z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n}\left[\mathcal{N}(x_n \mid \boldsymbol{\mu}_\theta(z_n), \boldsymbol{\Sigma}_\theta(z_n))\right] = \int \mathcal{N}(z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n) \ln \mathcal{N}(x_n \mid \boldsymbol{\mu}_\theta(z_n), \boldsymbol{\Sigma}_\theta(z_n))\, dz_n$$

Can we just draw $z_n^{(m)} \sim \mathcal{N}(z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n)$ and use Monte Carlo?

$$\mathbb{E}_{z_n \mid \boldsymbol{m}_n, \boldsymbol{V}_n}\left[\mathcal{N}(x_n \mid \boldsymbol{\mu}_\theta(z_n), \boldsymbol{\Sigma}_\theta(z_n))\right] \approx \frac{1}{M} \sum_{m=1}^{M} \ln \mathcal{N}(x_n \mid \boldsymbol{\mu}_\theta(z_n^{(m)}), \boldsymbol{\Sigma}_\theta(z_n^{(m)}))$$

- Gradient with respect to θ? No problem.
- Gradient with respect to $\boldsymbol{m}_n$ and $\boldsymbol{V}_n$? Where did they go?!?!?

Kingma and Welling (2014) suggested a clever trick.

The Reparameterization Trick

The reparameterization trick is a way to address the following general situation:

$$\nabla_\alpha \, \mathbb{E}_{z \sim \pi_\alpha}[\, f(z)\,] = \nabla_\alpha \int \pi_\alpha(z) f(z)\, dz$$

Here the parameter α governs the distribution under which the expectation is being taken. If we sample $z_m \sim \pi_\alpha$, we get something non-differentiable in α:

$$\nabla_\alpha \left[\frac{1}{M} \sum_{m=1}^{M} f(z_m) \right]$$

The Reparameterization Trick

Can simulate from many "standard" parametric distributions via differentiable parametric transformation of a fixed distribution.[3] Examples:

$$
\begin{aligned}
\text{univariate Gaussian:} \quad & w \sim \mathcal{N}(0,1) && \implies && aw + b \sim \mathcal{N}(b, a^2) \\
\text{multivariate Gaussian:} \quad & \boldsymbol{w} \sim \mathcal{N}(0, \mathbf{I}) && \implies && \boldsymbol{A}\boldsymbol{w} + \boldsymbol{b} \sim \mathcal{N}(\boldsymbol{b}, \boldsymbol{A}\boldsymbol{A}^{\mathsf{T}}) \\
\text{exponential:} \quad & w \sim \mathcal{U}(0,1) && \implies && -\ln(w)/\lambda \sim \text{Exp}(\lambda) \\
\text{gamma:} \quad & w \sim \text{Gamma}(k,1) && \implies && a\boldsymbol{w} \sim \text{Gamma}(k,a)
\end{aligned}
$$

Reparametrize the integral using the simple fixed distribution $\rho(w)$ and an α-parameterized transformation:

$$\nabla_\alpha \, \mathbb{E}_{z \sim \pi_\alpha}[\, f(z)\,] = \nabla_\alpha \int \pi_\alpha(z) f(z)\, dz = \nabla_\alpha \int \rho(w) f(t_\alpha(w))\, dw$$

[3] Essentially anything with a reasonable quantile function.

The Reparameterization Trick

Reparametrize the integral using the simple fixed distribution $\rho(w)$ and an α-parameterized transformation:

$$\nabla_\alpha \, \mathbb{E}_{z \sim \pi_\alpha}[f(z)] = \nabla_\alpha \int \pi_\alpha(z) f(z) \, dz = \nabla_\alpha \int \rho(w) f(t_\alpha(w)) \, dw$$

Draw $w_m \sim \rho(w)$ and now Monte Carlo plays nicely with differentiation:

$$\begin{aligned} \nabla_\alpha \int \rho(w) f(t_\alpha(w)) \, dw &\approx \nabla_\alpha \frac{1}{M} \sum_{m=1}^{M} f(t_\alpha(w_m)) \\ &\approx \frac{1}{M} \sum_{m=1}^{M} \nabla_\alpha f(t_\alpha(w_m)) \end{aligned}$$

Shakir Mohamed has a very nice blog post discussing this trick (Mohamed, 2015).

Reparameterization and the VAE

Draw a set of $\epsilon_n^{(m)} \sim \mathcal{N}(0, \mathbf{I})$ and parameterize via $\boldsymbol{W}_n$ such that $\boldsymbol{W}_n \boldsymbol{W}_n^{\mathsf{T}} = \boldsymbol{V}_n$:

$$\mathbb{E}_{z_n \,|\, \boldsymbol{m}_n, \boldsymbol{V}_n} \left[\mathcal{N}(x_n \,|\, \boldsymbol{\mu}_\theta(z_n), \boldsymbol{\Sigma}_\theta(z_n)) \right] = \int \mathcal{N}(z_n \,|\, \boldsymbol{m}_n, \boldsymbol{V}_n) \ln \mathcal{N}(x_n \,|\, \boldsymbol{\mu}_\theta(z_n), \boldsymbol{\Sigma}_\theta(z_n)) \, dz_n$$

$$= \int \mathcal{N}(\epsilon_n \,|\, 0, \mathbf{I}) \, \ln \mathcal{N}(x_n \,|\, \boldsymbol{\mu}_\theta(\boldsymbol{W}_n \boldsymbol{\epsilon}_n + \boldsymbol{m}_n), \boldsymbol{\Sigma}_\theta(\boldsymbol{W}_n \boldsymbol{\epsilon}_n + \boldsymbol{m}_n)) \, d\boldsymbol{\epsilon}_n$$

$$\approx \frac{1}{M} \sum_{m=1}^{M} \ln \mathcal{N}(x_n \,|\, \boldsymbol{\mu}_\theta(\boldsymbol{W}_n \boldsymbol{\epsilon}_n^{(m)} + \boldsymbol{m}_n), \boldsymbol{\Sigma}_\theta(\boldsymbol{W}_n \boldsymbol{\epsilon}_n^{(m)} + \boldsymbol{m}_n))$$

Now it is possible to differentiate with regard to $\boldsymbol{m}_n$ and $\boldsymbol{W}_n$.

Amortizing Inference in the VAE

Recall that there were several annoying things about mean-field VI in our model:

- The number of optimized dimensions scales with N.
- We have to perform an optimization to make an out-of-sample inference.
- Computing the expected complete data log likelihood looks hard.

Amortizing Inference in the VAE

Recall that there were several annoying things about mean-field VI in our model:

- The number of optimized dimensions scales with N.
- We have to perform an optimization to make an out-of-sample inference.
- ~~Computing the expected complete data log likelihood looks hard.~~

Amortizing Inference in the VAE

Recall that there were several annoying things about mean-field VI in our model:

- The number of optimized dimensions scales with N.
- We have to perform an optimization to make an out-of-sample inference.
- ~~Computing the expected complete data log likelihood looks hard.~~

Can we just look at a datum and guess its variational parameters?

Amortizing Inference in the VAE

Recall that there were several annoying things about mean-field VI in our model:

- The number of optimized dimensions scales with N.
- We have to perform an optimization to make an out-of-sample inference.
- ~~Computing the expected complete data log likelihood looks hard.~~

Can we just look at a datum and guess its variational parameters?

Anybody have any good function approximators lying around?

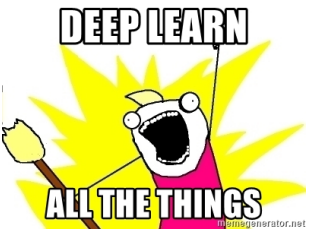
Amortizing Inference in the VAE

Recall that there were several annoying things about mean-field VI in our model:

- The number of optimized dimensions scales with N.
- We have to perform an optimization to make an out-of-sample inference.
- ~~Computing the expected complete data log likelihood looks hard.~~

Can we just look at a datum and guess its variational parameters?

Anybody have any good function approximators lying around?



Amortizing Inference in the VAE

Throw away all of the per-datum variational parameters $\{\boldsymbol{m}_n, \boldsymbol{V}_n\}_{n=1}^N$.

Replace them with parametric functions that see the input: $\boldsymbol{m}_\gamma(x)$ and $\boldsymbol{V}_\gamma(x)$.

Rederive the lower bound with γ instead of $\{\boldsymbol{m}_n, \boldsymbol{V}_n\}_{n=1}^N$:

$$\mathcal{L}(\theta, \gamma) = \sum_{n=1}^{N} \mathbb{E}_{z_n \mid x_n, \gamma} \left[\mathcal{N}(x_n \mid \boldsymbol{\mu}_\theta(z_n), \boldsymbol{\Sigma}_\theta(z_n))\right] - \mathrm{KL}\left[\mathcal{N}(z_n \mid \boldsymbol{m}_\gamma(x_n), \boldsymbol{\Sigma}_\gamma(x_n)) || \mathcal{N}(z_n \mid 0, \mathbf{I})\right]$$

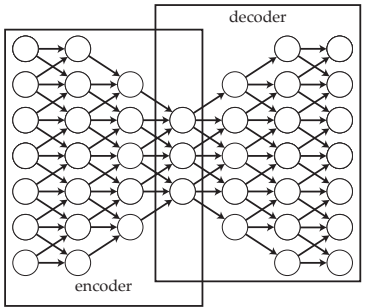
Can now do mini-batch stochastic optimization without local variables.

Amortized: pay up front and then use it cheaply. (Gershman and Goodman, 2014)

What does this have to do with autoencoders?

encoder = "recognition network" = amortized inference
takes data and maps it to (a distribution over) a latent representation

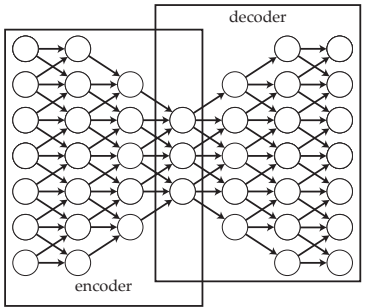
decoder = likelihood = generative model
takes latent representation and produces data

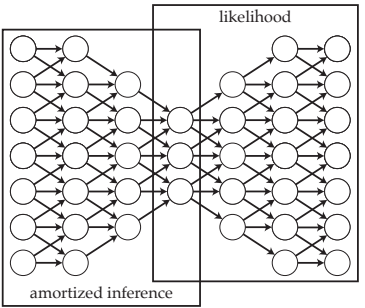


What does this have to do with autoencoders?

encoder = "recognition network" = amortized inference
takes data and maps it to (a distribution over) a latent representation

decoder = likelihood = generative model
takes latent representation and produces data





Importance Weighted Autoencoder (Burda et al., 2016)

$$\ln P(x \mid \theta) = \ln \int P(x, z \mid \theta)\, dz \quad = \ln \int q(z) \frac{P(x, z \mid \theta)}{q(z)}\, dz \geq \int q(z) \ln \frac{P(x, z \mid \theta)}{q(z)}\, dz$$

Rather than using a single z, compute the ELBO with multiple z:

$$\ln P(x \mid \theta) = \ln \int q(z^{(1)}) q(z^{(2)}) \left[\frac{P(x, z^{(1)} \mid \theta)}{2q(z^{(1)})} + \frac{P(x, z^{(2)} \mid \theta)}{2q(z^{(2)})} \right] dz^{(1)} dz^{(2)}$$

$$\geq \int q(z^{(1)}) q(z^{(2)}) \ln \left[\frac{P(x, z^{(1)} \mid \theta)}{2q(z^{(1)})} + \frac{P(x, z^{(2)} \mid \theta)}{2q(z^{(2)})} \right] dz^{(1)} dz^{(2)}$$

More generally, allow for K "importance samples":

$$\ln P(x \mid \theta) \leq \mathbb{E}_{z^{(1)}, \ldots, z^{(K)} \sim q(z)} \left[\ln \frac{1}{K} \sum_{k=1}^{K} \frac{P(x, z^{(k)} \mid \theta)}{q(z^{(k)})} \right]$$

All else being equal, bigger K leads to a tighter bound.

Tutorial Outline

What is generative modeling?

Recipes for flexible generative models

Algorithms for learning generative models from data

Variational autoencoder

Combining graphical models and neural networks

How to get more structure in a VAE?

Probabilistic graphical models:

- Powerful structured probabilistic modeling tools.
- A **complementary** technology to neural networks
  - Allows strong physical and subjective priors.
  - Can yield interpretable structure.
  - Often have fast inference procedures based on dynamic programming.
  - Imperative modeling style.
  - Represent uncertainty explicitly.
  - Well understood model selection criteria.

Opportunity for **semiparametric** models in machine learning:
Compact interpretable latent structure wrapped in “deep nonparametric goo”.

Motivation: unsupervised modeling of behavior

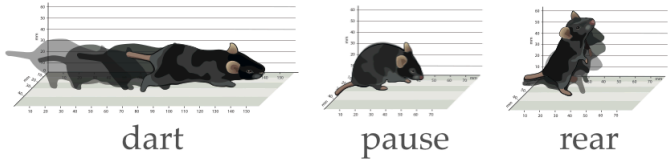
Motivation: unsupervised modeling of behavior

elevated “plus” maze

Motivation: unsupervised modeling of behavior

Motivation: unsupervised modeling of behavior

Motivation: discovering the language of behavior



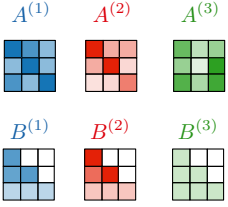
Wiltschko et al. (2015)

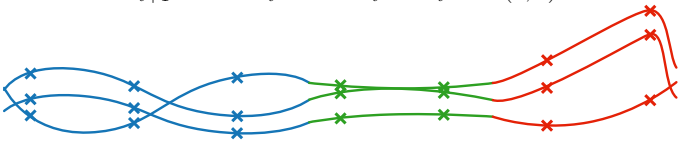
Mouse as switching linear dynamical system

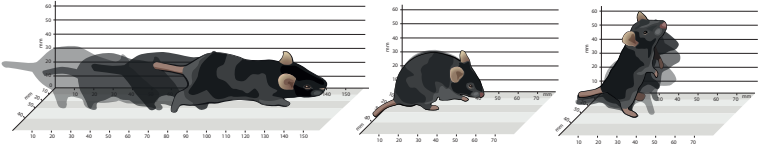
$$\pi = \begin{bmatrix} \pi^{(1)} \\ \pi^{(2)} \\ \pi^{(3)} \end{bmatrix}$$

$$z_{t+1} \sim \pi^{(z_t)}$$

$$x_{t+1} = A^{(z_t)} x_t + B^{(z_t)} u_t \qquad u_t \overset{\text{iid}}{\sim} \mathcal{N}(0, I)$$

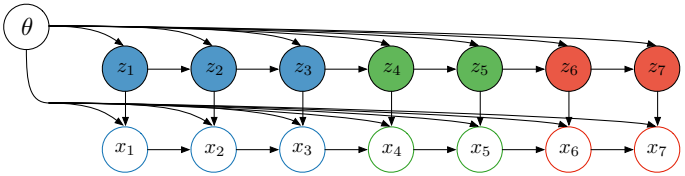






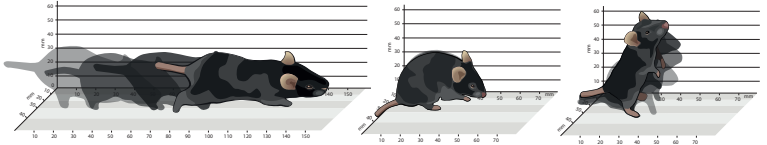
Mouse as switching linear dynamical system

$$\pi = \begin{bmatrix} — & \pi^{(1)} & — \\ — & \pi^{(2)} & — \\ — & \pi^{(3)} & — \end{bmatrix}$$

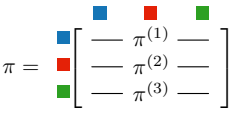


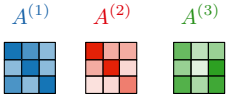
$A^{(1)}$ $A^{(2)}$ $A^{(3)}$

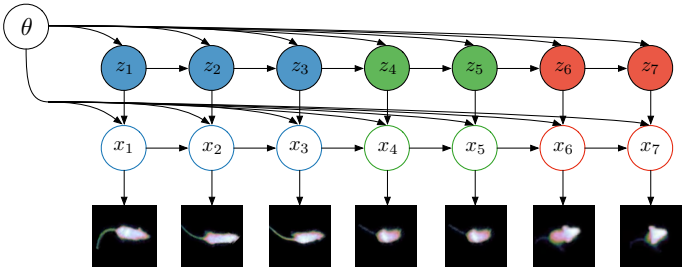
$B^{(1)}$ $B^{(2)}$ $B^{(3)}$

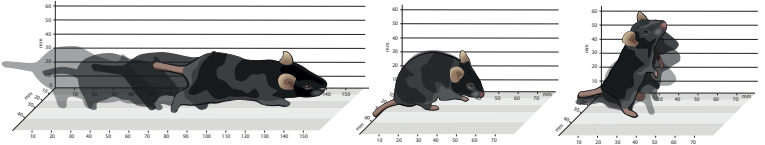


Mouse as switching linear dynamical system









Trading off richness and parsimony

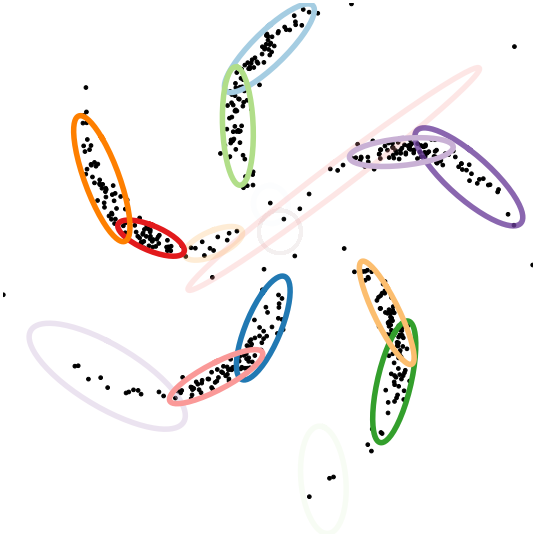
Simple data + simple model (linear regression)
= **simple hypotheses**

Complex data + complex model (deep neural net)
= **uninterpretable**

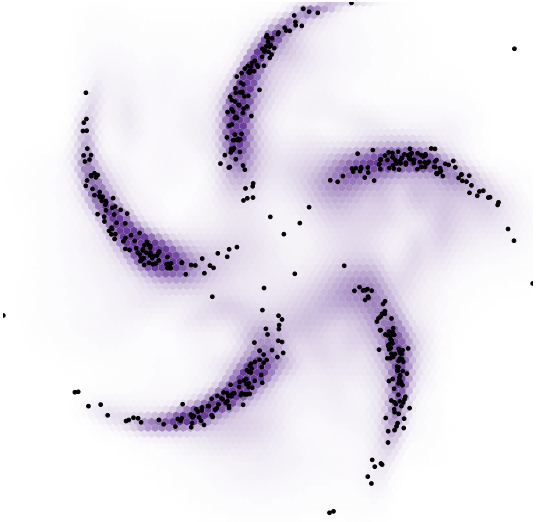
Complex data + structured model (semiparametric)
= **rich and interpretable**

Trading off richness and parsimony

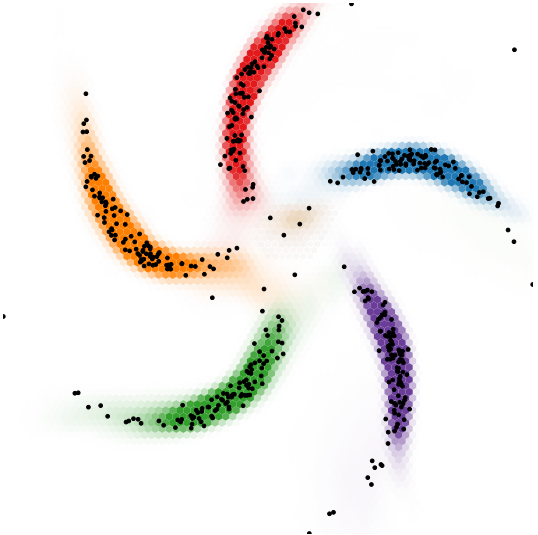
Trading off richness and parsimony



Trading off richness and parsimony



Trading off richness and parsimony



Manifold of mouse depth images

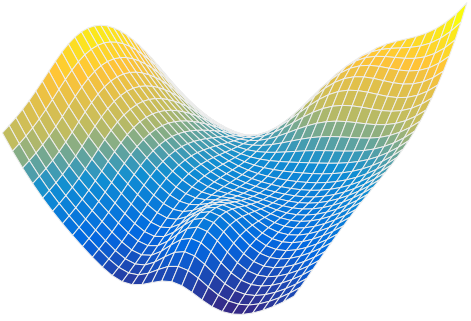
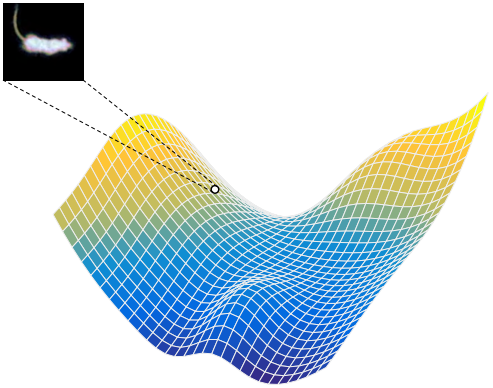


image
manifold

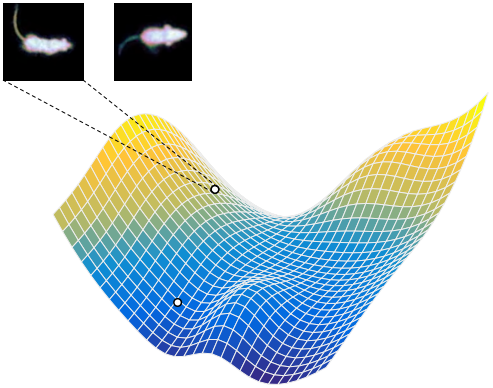
Manifold of mouse depth images



depth
video

image
manifold

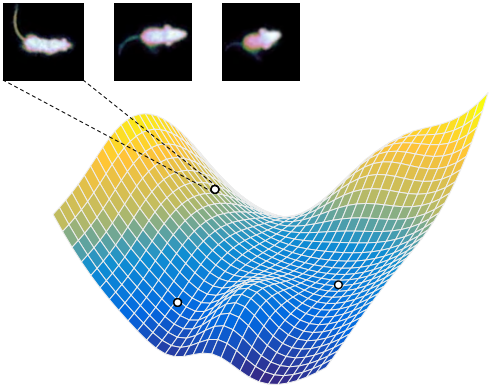
Manifold of mouse depth images



depth
video

image
manifold

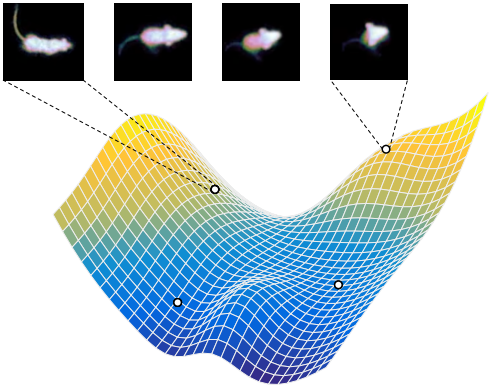
Manifold of mouse depth images



depth
video

image
manifold

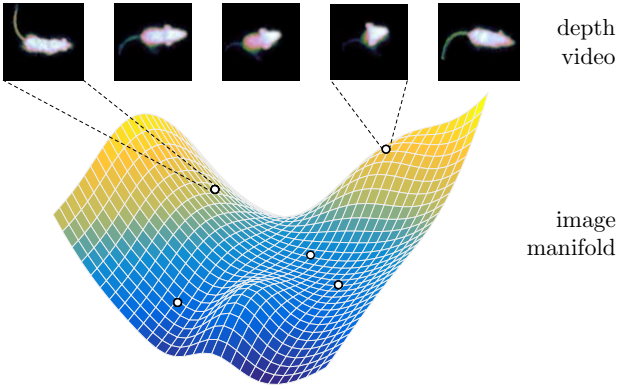
Manifold of mouse depth images



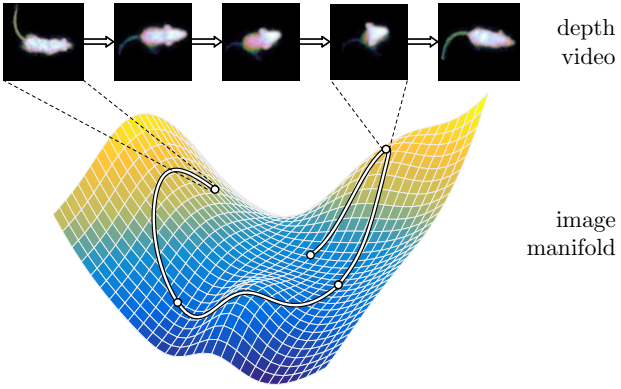
depth
video

image
manifold

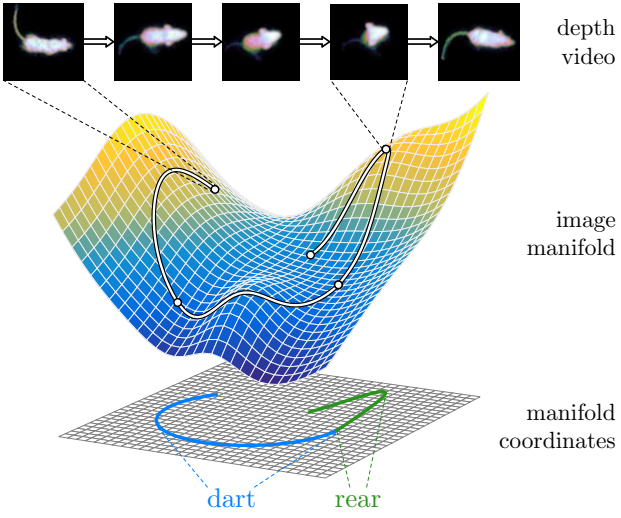
Manifold of mouse depth images



Manifold of mouse depth images

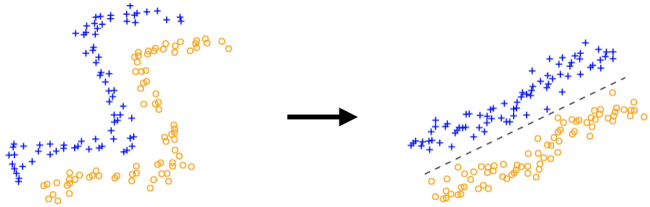


Manifold of mouse depth images

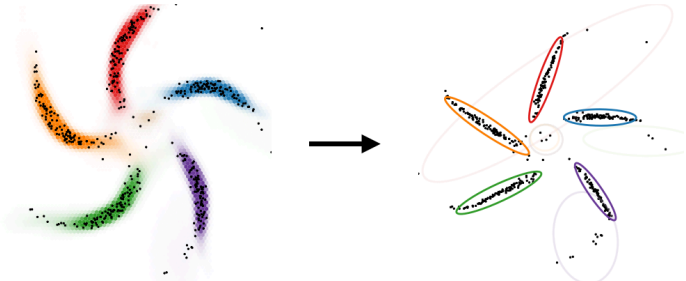


Big picture: learn basis functions that simplify

supervised learning
learn a basis so that linear classifiers work



unsupervised learning
learn a basis so that parsimonious density models work



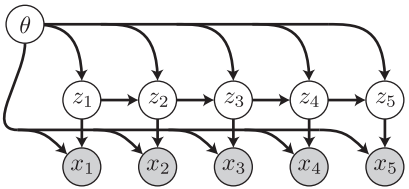
Stochastic variational inference

David Blei will talk about variational inference in much more detail.

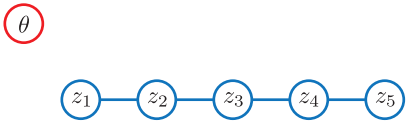
SVI from high altitude:

- Exponential families and conditional conjugacy lead to elegant stochastic optimization.
- Use same exponential family for variational approximation.
- Divide problem into *global* and *local* parameters.
- Determine optimal local parameters on a mini-batch and take a gradient step on the global parameters.
- Just computing expected sufficient statistics gives the natural gradient!
- Natural gradients use a metric that reflects the underlying probability model.

SVI in a linear dynamical system



$P(z \mid \theta)$ is linear dynamical system
$P(x \mid z, \theta)$ is linear Gaussian
$P(\theta)$ is a conjugate prior



$$q(\theta)q(z) \approx P(\theta, z \mid x)$$

$$\mathcal{L}(\eta_\theta, \eta_z) = \mathbb{E}_{q(\theta)q(z)}\left[\ln \frac{P(\theta, x, z)}{q(\theta)q(z)}\right]$$

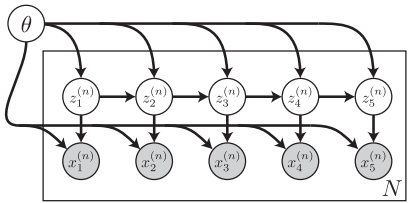
$$\eta_z^\star(\eta_\theta) = \arg\max_{\eta_z} \mathcal{L}(\eta_\theta, \eta_z)$$

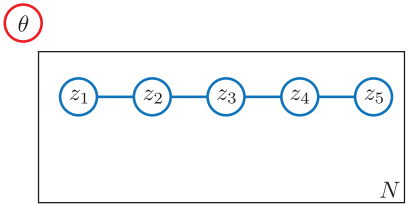
$$\mathcal{L}_{\text{SVI}}(\eta_\theta) = \mathcal{L}(\eta_\theta, \eta_z^\star(\eta_\theta))$$

Natural gradient SVI (Hoffman et al., 2013)

$$\tilde{\nabla}\mathcal{L}_{\text{SVI}}(\eta_\theta) = \eta_\theta^{\text{prior}} + \mathbb{E}_{q^\star(z)}(t_{x,z}(x, z), 1) - \eta_\theta$$

SVI in a linear dynamical system





$P(z \mid \theta)$ is linear dynamical system
$P(x \mid z, \theta)$ is linear Gaussian
$P(\theta)$ is a conjugate prior

$$q(\theta)q(z) \approx P(\theta, z \mid x)$$

$$\mathcal{L}(\eta_\theta, \eta_z) = \mathbb{E}_{q(\theta)q(z)}\left[\ln \frac{P(\theta, x, z)}{q(\theta)q(z)}\right]$$

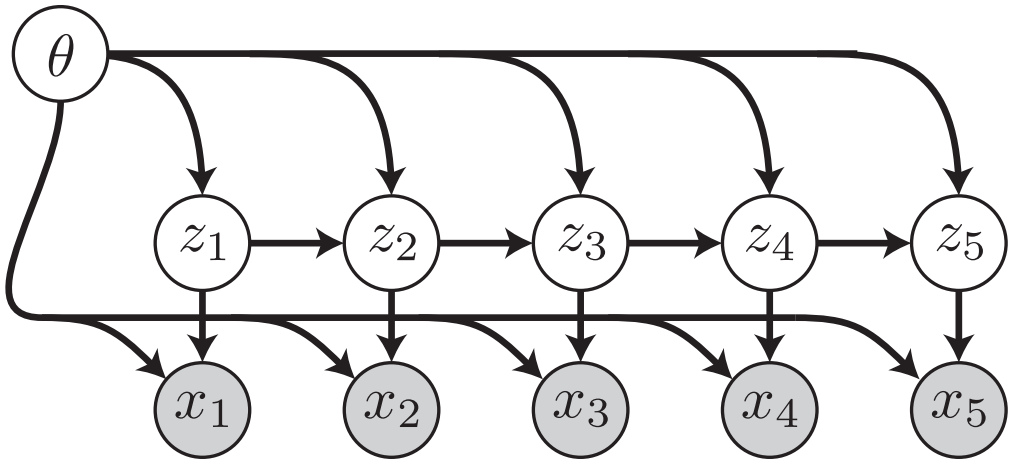
$$\eta_z^\star(\eta_\theta) = \arg\max_{\eta_z} \mathcal{L}(\eta_\theta, \eta_z)$$

$$\mathcal{L}_{\text{SVI}}(\eta_\theta) = \mathcal{L}(\eta_\theta, \eta_z^\star(\eta_\theta))$$

Natural gradient SVI (Hoffman et al., 2013)

$$\tilde{\nabla}\mathcal{L}_{\text{SVI}}(\eta_\theta) = \eta_\theta^{\text{prior}} + \sum_{n=1}^{N} \mathbb{E}_{q^\star(z_n)}(t_{x,z}(x_n, z_n), 1) - \eta_\theta$$

SVI in a linear dynamical system



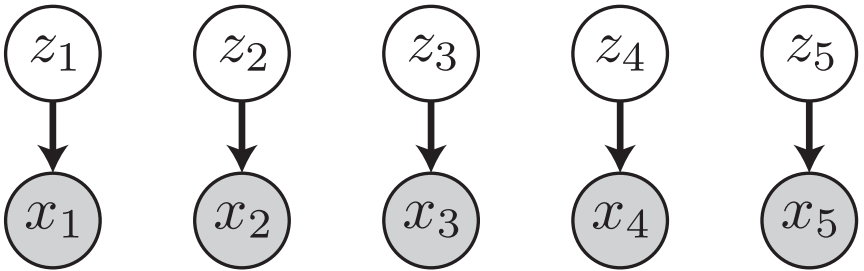
model

SVI in a linear dynamical system

observations

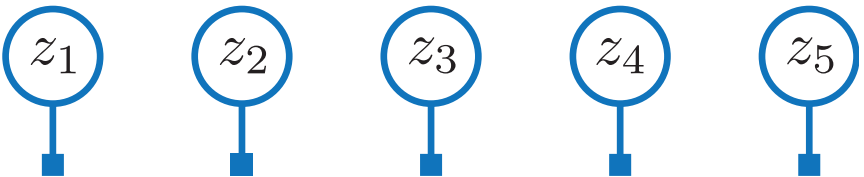
SVI in a linear dynamical system

likelihood



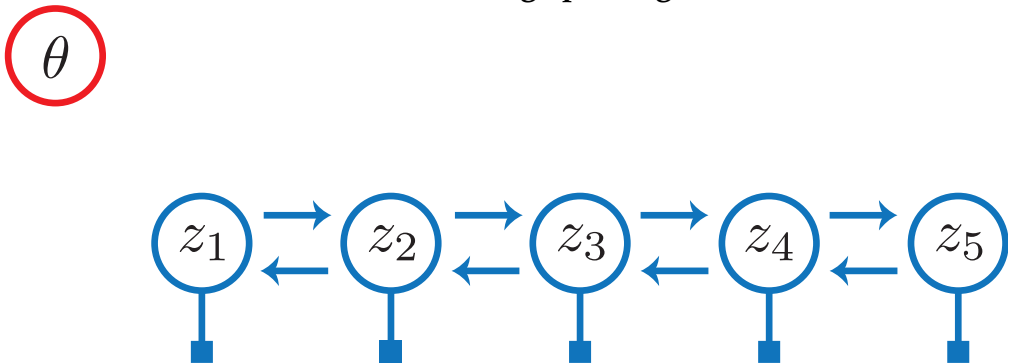
SVI in a linear dynamical system

evidence potentials



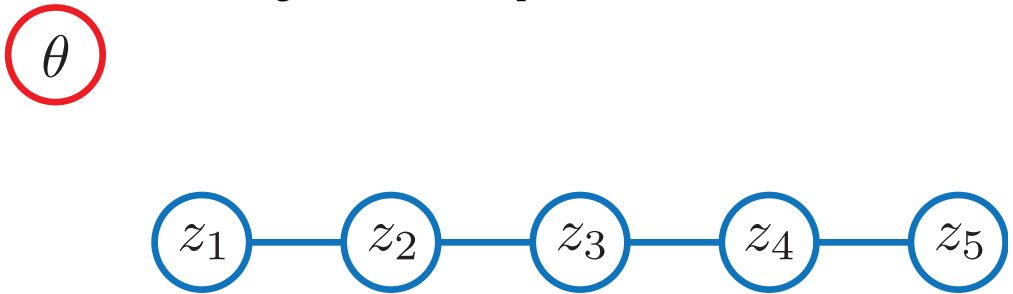
SVI in a linear dynamical system

fast message passing

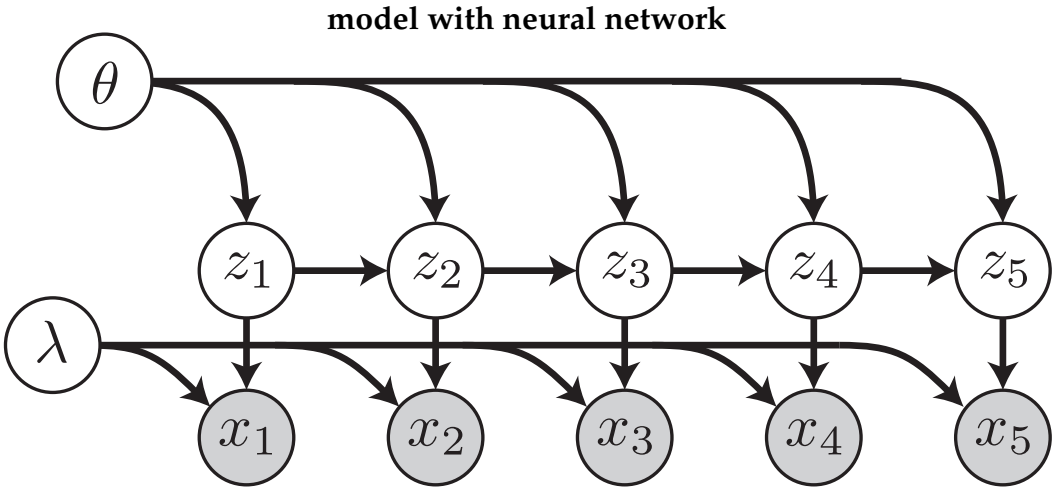


SVI in a linear dynamical system

natural gradient from expected sufficient statistics



Structured VAE (Johnson et al., 2016)

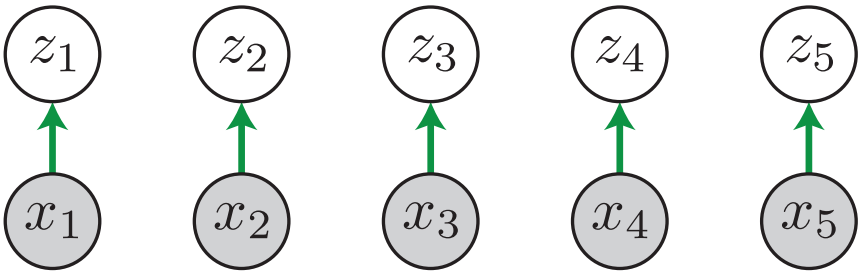


Structured VAE (Johnson et al., 2016)

observations

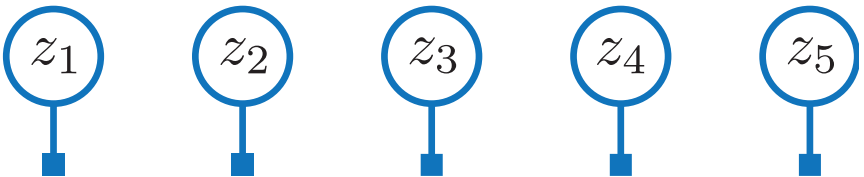
Structured VAE (Johnson et al., 2016)

recognition network



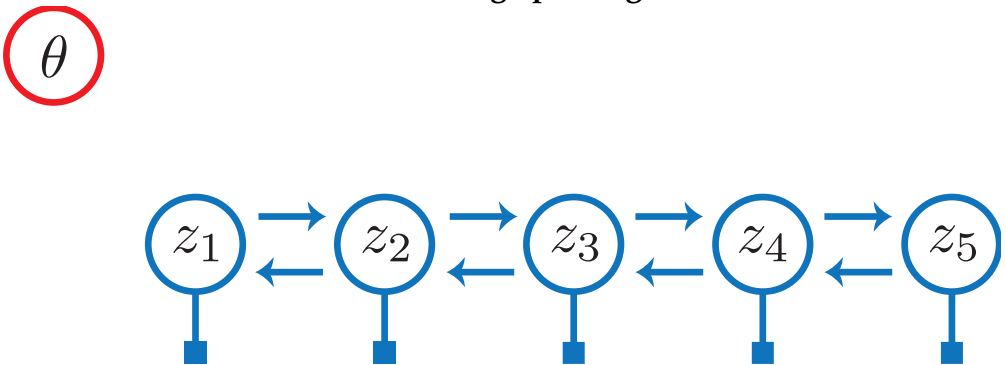
Structured VAE (Johnson et al., 2016)

evidence potentials



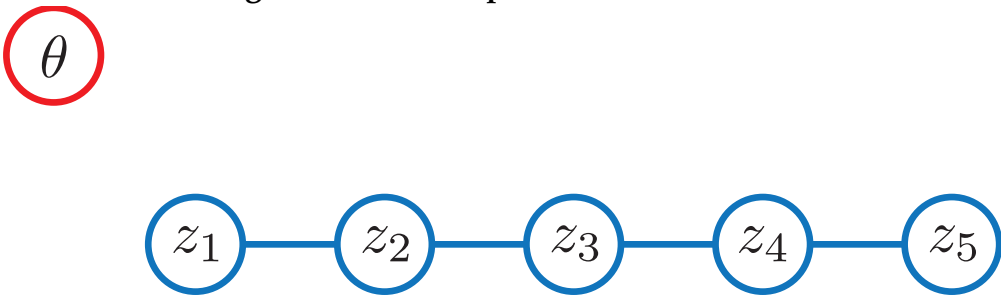
Structured VAE (Johnson et al., 2016)

fast message passing



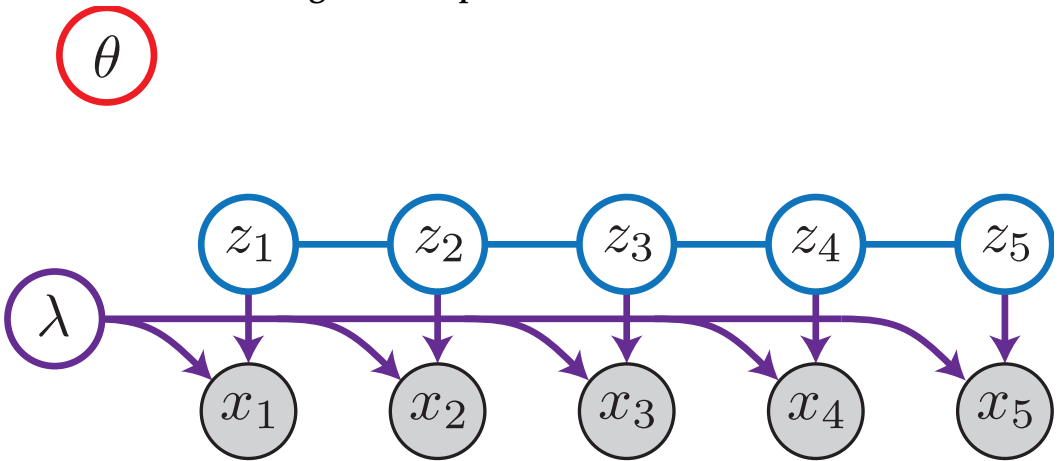
Structured VAE (Johnson et al., 2016)

natural gradient from expected sufficient statistics



Structured VAE (Johnson et al., 2016)

flat gradient updates for neural networks



SVAE: fitting a warped mixture

SVAE: finding behavioral syllables

SVAE: finding behavioral syllables

SVAE: finding behavioral syllables

Structured VAE (Johnson et al., 2016)

Natural gradient SVI:
- \- expensive for general obs.
- \+ optimal local factors
- \+ exploits graph structure
- \+ arbitrary inference queries
- \+ natural gradients

Variational autoencoder:
- \+ fast for general obs.
- \- suboptimal local factors
- \- limited inference queries
- \- no easy natural gradients
- \- gooey latent space

Structured VAE:
- \+ fast for general obs.
- \+ optimal conjugate factors
- \+ exploits graph structure
- \+ arbitrary inference queries
- \+ natural gradients on η_θ

Wrap-up

- Generative models allow us to ask many kinds of questions about data.
- Multiple recipes for rich parametric models.
- Lots of ways to do inference and learning, all with strengths and weaknesses.
- Power through composition and abstraction.
- Many things I did not cover:
  - Neurbeal autoregressive distribution estimation (Larochelle and Murray, 2011)
  - Denoising autoencoders as generative models (Bengio et al., 2013)
  - Deep exponential families (Ranganath et al., 2015)
  - Helmholtz machine (Dayan et al., 1995)
  - Deep energy models (Ngiam et al., 2011)
  - Sum-product networks (Poon and Domingos, 2011)
  - ...

References I

Adams, R., Wallach, H., and Ghahramani, Z. (2010). Learning the structure of deep sparse graphical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 1–8.

Bach, F. R. and Jordan, M. I. (2002). Kernel independent component analysis. *Journal of Machine Learning Research*, 3(Jul):1–48.

Bengio, Y., Yao, L., Alain, G., and Vincent, P. (2013). Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems*, pages 899–907.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Burda, Y., Grosse, R., and Salakhutdinov, R. (2016). Importance weighted autoencoders. In *International Conference on Learning Representations*.

References II

Burel, G. (1992). Blind separation of sources: A nonlinear neural algorithm. *Neural Networks*, 5(6):937–947.

Comon, P. (1994). Independent component analysis, a new concept? *Signal Processing*, 36(3):287–314.

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995). The Helmholtz machine. *Neural Computation*, 7(5):889–904.

DeMers, D. and Cottrell, G. W. (1993). Non-linear dimensionality reduction. In *Advances in Neural Information Processing Systems*, pages 580–587.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

Dziugaite, G. K., Roy, D. M., and Ghahramani, Z. (2015). Training generative neural networks via maximum mean discrepancy optimization. In *Conference on Uncertainty in Artificial Intelligence*.

References III

Fergus, R., Hogg, D. W., Oppenheimer, R., Brenner, D., and Pueyo, L. (2014). S4: A spatial-spectral model for speckle suppression. *The Astrophysical Journal*, 794(2):161.

Freund, Y. and Haussler, D. (1992). Unsupervised learning of distributions on binary vectors using two layer networks. In *Advances in Neural Information Processing Systems*, pages 912–919.

Frey, B. J. and Hinton, G. E. (1999). Variational learning in nonlinear Gaussian belief networks. *Neural Computation*, 11(1):193–213.

Gershman, S. and Goodman, N. (2014). Amortized inference in probabilistic reasoning. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 36.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. (2016). Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*.

References IV

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.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012). A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347.

Huszar, F. (2015). Another favourite machine learning paper: Adversarial networks vs kernel scoring rules. http://www.inference.vc/another-favourite-machine-learning-paper-adversarial-networks-vs-kernel-scoring-rules/.

Hyvärinen, A. (2005). Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(Apr):695–709.

References V

Hyvärinen, A. and Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural Networks*, 13(4-5):411–430.

Johnson, M., Duvenaud, D. K., Wiltschko, A., Adams, R. P., and Datta, S. R. (2016). Composing graphical models with neural networks for structured representations and fast inference. In *Advances in neural information processing systems*, pages 2946–2954.

Jutten, C. and Herault, J. (1991). Blind separation of sources, part i: An adaptive algorithm based on neuromimetic architecture. *Signal Processing*, 24(1):1–10.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational Bayes. In *International Conference on Learning Representations*.

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243.

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

References VI

Lawrence, N. (2005). Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *Journal of Machine Learning Research*, 6(Nov):1783–1816.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al. (2017). Photo-realistic single image super-resolution using a generative adversarial network. In *International Conference on Computer Vision and Pattern Recognition*.

Li, Y., Swersky, K., and Zemel, R. (2015). Generative moment matching networks. In *International Conference on Machine Learning*, pages 1718–1727.

Linderman, S. W., Johnson, M. J., Wilson, M. A., and Chen, Z. (2016). A Bayesian nonparametric approach for uncovering rat hippocampal population codes during spatial navigation. *Journal of Neuroscience Methods*, 263:36–47.

References VII

MacKay, D. J. (1995). Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 354(1):73–80.

Mescheder, L., Nowozin, S., and Geiger, A. (2017). Adversarial variational Bayes: Unifying variational autoencoders and generative adversarial networks. In *International Conference on Machine Learning*, volume 70.

Miller, A., Wu, A., Regier, J., McAuliffe, J., Lang, D., Prabhat, M., Schlegel, D., and Adams, R. P. (2015). A Gaussian process model of quasar spectral energy distributions. In *Advances in Neural Information Processing Systems*, pages 2494–2502.

Mohamed, S. (2015). Machine learning trick of the day (4): Reparameterisation tricks. http://blog.shakirm.com/2015/10/machine-learning-trick-of-the-day-4-reparameterisation-tricks/.

Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, 56(1):71–113.

References VIII

Ng, A. Y. and Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In *Advances in Neural Information Processing Systems*, pages 841–848.

Ngiam, J., Chen, Z., Koh, P. W., and Ng, A. Y. (2011). Learning deep energy models. In *International Conference on Machine Learning*, pages 1105–1112.

Pajunen, P., Hyvärinen, A., and Karhunen, J. (1996). Nonlinear blind source separation by self-organizing maps. In *International Conference on Neural Information Processing*.

Poon, H. and Domingos, P. (2011). Sum-product networks: A new deep architecture. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 689–690. IEEE.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Ranganath, R., Tang, L., Charlin, L., and Blei, D. (2015). Deep exponential families. In *Artificial Intelligence and Statistics*, pages 762–771.

References IX

Regier, J., Miller, A., McAuliffe, J., Adams, R., Hoffman, M., Lang, D., Schlegel, D., and Prabhat, M. (2015). Celeste: Variational inference for a generative model of astronomical images. In *International Conference on Machine Learning*, pages 2095–2103.

Rippel, O. and Adams, R. P. (2013). High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*.

Roweis, S. and Ghahramani, Z. (1999). A unifying review of linear Gaussian models. *Neural Computation*, 11(2):305–345.

Roweis, S. T. (1998). EM algorithms for PCA and SPCA. In *Advances in Neural Information Processing Systems*, pages 626–632.

Salakhutdinov, R. and Hinton, G. (2009). Deep boltzmann machines. In *Artificial Intelligence and Statistics*, pages 448–455.

Schölkopf, B., Smola, A., and Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319.

References X

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. Technical report, COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE.

Tieleman, T. (2008). Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071. ACM.

Tipping, M. E. and Bishop, C. M. (1999). Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622.

Vapnik, V. (1998). *Statistical learning theory. 1998*. Wiley, New York.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *Conference on Computer Vision and Pattern Recognition*, pages 3156–3164. IEEE.

References XI

Wiltschko, A. B., Johnson, M. J., Iurilli, G., Peterson, R. E., Katon, J. M., Pashkovski, S. L., Abraira, V. E., Adams, R. P., and Datta, S. R. (2015). Mapping sub-second structure in mouse behavior. *Neuron*, 88(6):1121–1135.

Wood, F. and Black, M. J. (2008). A nonparametric Bayesian alternative to spike sorting. *Journal of Neuroscience Methods*, 173(1):1–12.