Probabilistic Reasoning

Volodymyr Kuleshov

Cornell Tech

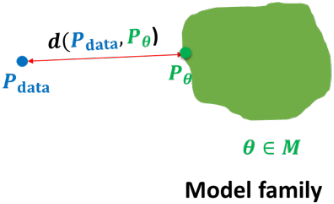
Lecture 13

Announcements

- Due date for Assignment 3 extended to March 20.
- Deadline for signing up for presentations will be **this Friday**.

Why Are Generative Models Useful?

$\mathbf{x}_i \sim P_{\text{data}}$
$i = 1, 2, \dots, n$



$d(P_{\text{data}}, P_\theta)$

P_{data}

P_θ

$\theta \in M$

Model family

We want to learn a distribution $p(x)$ with the following features:

- **Generation:** If we sample $x_{new} \sim p(x)$, x_{new} should look like the data distribution
- **Density estimation:** $p(x)$ should be high if x is from data distribution, and low otherwise
- **Unsupervised representation learning:** Learning hidden structure (features) underlying the data

These are practical use cases. Generative models are also an example of a *way of thinking* about machine learning and specifying inductive biases.

Lecture Outline

1. Probabilistic Reasoning
    - Motivational Example: Modeling Student Grades with Bayes Nets
    - Probabilistic Approach to Machine Learning, Box's Loop
    - In-Depth Example: Topic Modeling with Latent Dirichlet Allocation
2. Deep Generative Models & Probabilistic Modeling
    - Composing Probabilistic Models with Deep Neural Networks
    - Examples: GMMs, Deep Kalman Filters
3. Neural Networks & Probabilistic Inference
    - Black-Box Variational Inference
    - Implicit Variational Inference
4. Probabilistic Programming

Motivational Example: Modeling Student Grades

Suppose we want to build a model that infers the difficulty of a class given the students' grades and other data. We have two options:

1. Build a discriminative model $p(\text{Difficulty} \mid \text{Data})$ parameterized by a black-box model trained using supervised learning.
2. Use a structured generative model to encode our domain knowledge of how grades, letters, etc. are related to each other:

$$p(\text{Difficulty} \mid \text{Data}) = \frac{p(\text{Difficulty}, \text{Data})}{p(\text{Data})}$$

Recall: Bayesian Networks

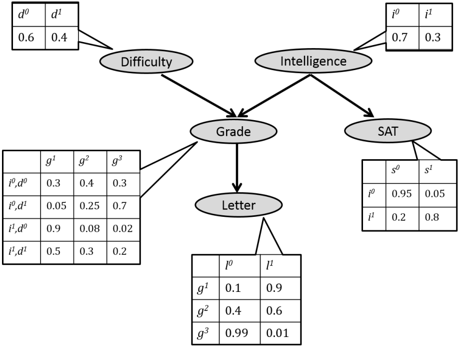
- A **Bayesian network** is specified by a *directed* **acyclic** graph $G = (V, E)$ with:
  1. One node $i \in V$ for each random variable X_i
  2. One conditional probability distribution (CPD) per node, $p(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)})$, specifying the variable's probability conditioned on its parents' values
- Defines a joint distribution:

$$p(x_1, \dots x_n) = \prod_{i \in V} p(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)})$$

- The graph encodes our knowledge of conditional independencies
- Graph provides a compact model representation: model size is exponential in $|\mathrm{Pa}(i)|$, not $|V|$

Modeling Student Grades with Bayesian Networks

Consider the following model of student performance in a class.



d^0	d^1
0.6	0.4

i^0	i^1
0.7	0.3

	g^1	g^2	g^3
i^0,d^0	0.3	0.4	0.3
i^0,d^1	0.05	0.25	0.7
i^1,d^0	0.9	0.08	0.02
i^1,d^1	0.5	0.3	0.2

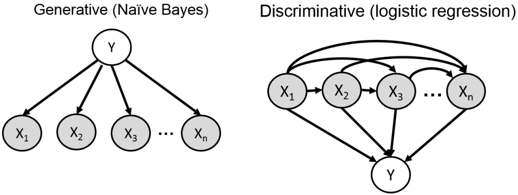
	s^0	s^1
i^0	0.95	0.05
i^1	0.2	0.8

	l^0	l^1
g^1	0.1	0.9
g^2	0.4	0.6
g^3	0.99	0.01

- It tells a generative story of the data and incorporates inductive biases derived from our prior knowledge.
- It yields predictions via Bayes' rule:

$$p(\text{Diff} \mid \text{Letter}, \text{Grade}, \text{SAT}) = \sum_{\text{Intelligence}} P(L|G)P(G|D,I)P(S|I)P(D)P(I)$$

Discriminative vs. Generative Models



Discriminative and generative models often specify the same (linear) model family. But they produce different results

- In general, discriminative models make fewer assumptions and are more accurate given enough data.
- But if the inductive bias of the generative model is correct, it will learn faster and with less data.

See Ng and Jordan (2002) for more details.

Probabilistic Reasoning

Generative models exemplify the *probabilistic approach* to machine learning.

- In the probabilistic approach, we specify a structured probabilistic model based on our understanding of the problem.
- This approach provides a principled way of specifying prior knowledge and inductive biases.
- It also provides a principled treatment of uncertainty – important for decision-making, human-in-the-loop AI, etc.
- It's a general *philosophy* for how to approach and solve ML problems.

Three Aspects of Probabilistic Reasoning

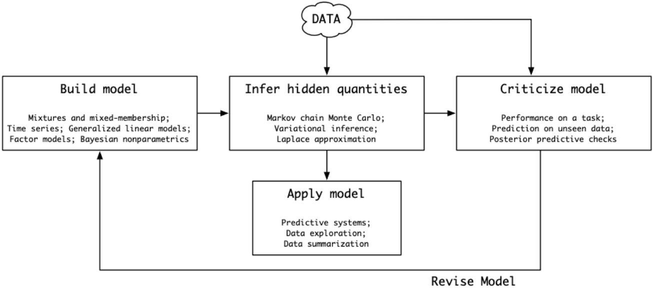
Probabilistic reasoning is an approach to machine learning in which we work with structured probabilistic models that encode our understanding of the problem.

There are three main questions in probabilistic reasoning:

- **Representation**: How do we specify a (possibly latent-variable) model $p(x, z)$?
- **Inference**: How do we interrogate the model (e.g., find features $p(z|x)$, generate from $p(x|z)$)?
- **Learning**: How do we learn $p(x, z)$ from data?

Box's Loop

Box's loop is an iterative process for building probabilistic models.

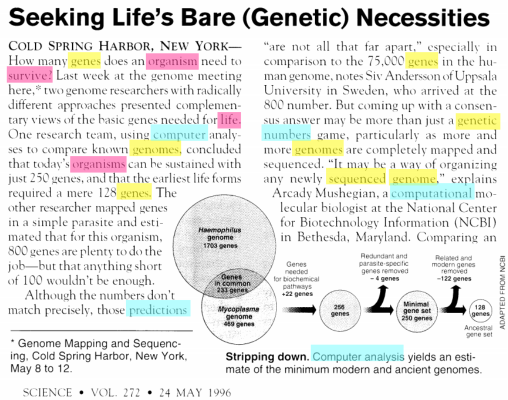


1. We formulate a model based on our knowledge of problem structure
2. We perform inference and learning and examine the model's results
3. We critique the model, comparing it against data. We go back to Step 1 and repeat until we're satisfied.

Figure by David Blei

Example: Topic Modeling

Consider the following scientific paper on computational genomics:



Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK—How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

Haemophilus genome 1703 genes
Genes in common 233 genes
Mycoplasma genome 469 genes
Genes needed for biochemical pathways +22 genes
256 genes
Redundant and parasite-specific genes removed − 4 genes
Minimal gene set 250 genes
Related and modern genes removed −122 genes
128 genes
Ancestral gene set

ADAPTED FROM NCBI

Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

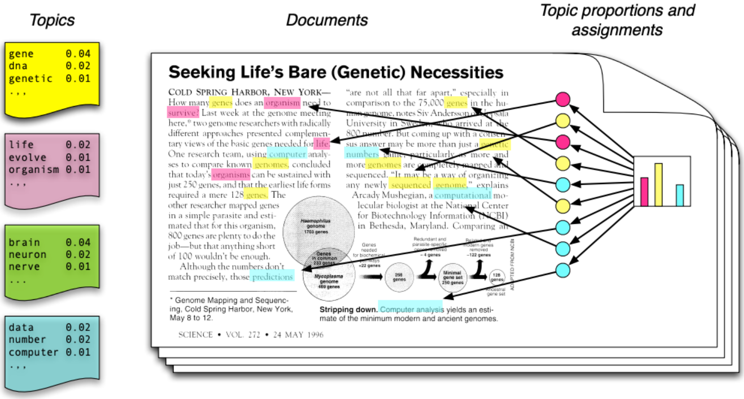
SCIENCE • VOL. 272 • 24 MAY 1996

Its contents describe at least three different topics: genetics (yellow), computers (blue), and biology (pink).

Materials by David Blei

Example: Topic Modeling

We want a model that will identify the topics discussed in the article.



Topics
Documents
Topic proportions and assignments
Haemophilus genome 1703 genes
Genes in common 233 genes
Mycoplasma genome 469 genes
Genes needed for biochemical pathways +22 genes
256 genes
Redundant and parasite-specific genes removed −4 genes
Minimal gene set 250 genes
Related and modern genes removed −122 genes
128 genes
Ancestral gene set
ADAPTED FROM NCBI

gene	0.04
dna	0.02
genetic	0.01
...	

life	0.02
evolve	0.01
organism	0.01
...	

brain	0.04
neuron	0.02
nerve	0.01
...	

data	0.02
number	0.02
computer	0.01
...	

Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK— How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

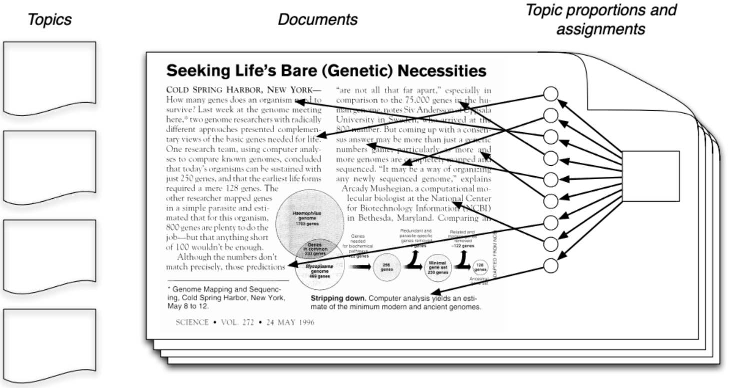
* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

SCIENCE • VOL. 272 • 24 MAY 1996

Materials by David Blei

Example: Topic Modeling

In practice these topics are not observed and we need to infer them


Topics
Documents
Topic proportions and assignments
Haemophilus genome 1703 genes
Genes in common 233 genes
Mycoplasma genome 469 genes
Genes needed for biochemical pathways +22 genes
256 genes
Redundant and parasite-specific genes removed -4 genes
Minimal gene set 250 genes
Related and modern genes removed -122 genes
128 genes
Ancestral gene set
ADAPTED FROM NCBI

Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK— How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

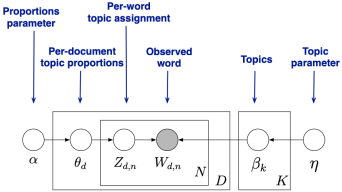
* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

SCIENCE • VOL. 272 • 24 MAY 1996

Materials by David Blei

Latent Dirichlet Allocation: Representation

Latent Dirichlet Allocation (LDA) defines the following probabilistic model



The model is defined by the following generative story. For each document d:

1. Draw a word distribution $\beta_k \sim \text{Dir}(\eta)$ for each topic k. Each $\beta_k \in [0,1]^N$ is a categorial distribution over a vocabulary of N words.
2. For each document d:
   1. Draw a set of topic proportions $\theta_n \sim \text{Dir}(\alpha)$, $\theta_n \in [0,1]^K$
   2. For each word n in the document:
      1. Draw a topic index $Z_{d,n} \sim \text{Cat}(\theta_d)$ in $1, 2, ..., K$
      2. Draw a word from the distribution for that topic $W_{d,n} \sim \text{Cat}(\beta_{Z_{d,n}})$

This defines a latent variable model: $p(W_{d,n}|Z_{d,n}, \beta_k)p(Z_{d,n}|\theta_d)p(\theta_d)p(\beta_k)$

Latent Dirichlet Allocation: Inference and Learning

How do we fit this model? At least two possible approaches:

- **Variational Inference:** Maximize ELBO to find the variational posterior $q(Z, \theta, \beta | W)$
- **MCMC:** Consider the true posterior $p(Z, \theta, \beta | W)$ as an unnormalized energy based model $\exp(-E)/Z$ with energy

  $$E = -\log p(Z, \theta, \beta, W)$$

  and an untractable $Z = p(W)$ and apply the sampling-based techniques for energy-based models that we saw.

We can also learn this model via variational inference or contrastive divergence with Gibbs sampling.

Lecture Outline

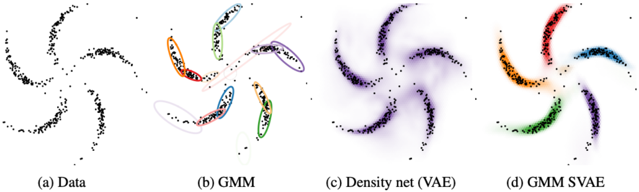
1. Probabilistic Reasoning
    - Motivational Example: Modeling Student Grades with Bayes Nets
    - Probabilistic Approach to Machine Learning, Box's Loop
    - In-Depth Example: Topic Modeling with Latent Dirichlet Allocation
2. Deep Generative Models & Probabilistic Modeling
    - Composing Probabilistic Models with Deep Neural Networks
    - Examples: GMMs, Deep Kalman Filters
3. Neural Networks & Probabilistic Inference
    - Black-Box Variational Inference
    - Implicit Variational Inference
4. Probabilistic Programming

Composing Probabilistic Models with Neural Networks

Probabilistic modeling and deep generative models are intimately connected.

1. Deep generative models are a notable example of the probabilistic approach (*previous section*).
2. Deep generative models can serve as composable building blocks within probabilistic models (*this section*).
3. Deep neural networks can accelerate and automate inference and learning in probabilistic models (*next section*).

Example: Deep Gaussian Mixture Models



(a) Data (b) GMM (c) Density net (VAE) (d) GMM SVAE

Consider fitting generative models to data shown in (a) (Johnson et al., 2016).

- **GMM (b):** Fits the data but uses too many components

$$z_i \sim \mathrm{Cat}(\pi) \qquad x_i|z_i, \{\mu_k, \Sigma_k\}_{k=1}^K \sim \mathcal{N}(\mu_{z_i}, \Sigma_{z_i})$$

- **VAE (c):** Diffuse probability; doesn't encode structure

$$z_i \sim \mathcal{N}(0, I) \qquad x_i|z_i, \theta \sim \mathcal{N}(\mu_\theta(z_i), \Sigma_\theta(z_i))$$

- **GMM+VAE (d):** Fits data with correct # of interpretable components.

$$z_i \sim \mathrm{Cat}(\pi) \quad u_i|z_i, \{\nu_k, T_k\}_{k=1}^K \sim \mathcal{N}(\nu_{z_i}, T_{z_i}) \quad x_i|u_i, \theta \sim \mathcal{N}(\mu_\theta(u_i), \Sigma_\theta(u_i))$$

Example: Learning Deep Gaussian Mixture Models

The GMM+VAE model finds interpretable components like a GMM and supports non-gaussian data for each component as a VAE.

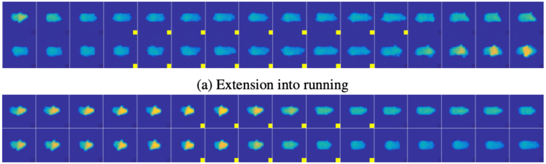
$$z_i \sim \mathrm{Cat}(\pi) \quad u_i | z_i, \{\nu_k, T_k\}_{k=1}^{K} \sim \mathcal{N}(\nu_{z_i}, T_{z_i}) \quad x_i | u_i, \theta \sim \mathcal{N}(\mu_\theta(u_i), \Sigma_\theta(u_i))$$

How do we perform learning and inference? Use variational inference:

1. For a fixed z_i we can fit $q(u_i|x_i)$ like a VAE
2. For a fixed u_i we can fit $q(z_i|u_i)$ like a GMM

Example: Deep Kalman Filtering

We extend GMM+VAE to time series (Johnson et al., 2016; Krishnan et al., 2016).


(a) Extension into running

(b) Fall from rear

- Suppose we observe a sequence of movie frames x_t, $t = 1, 2, ..., T$.
    - A video of a mouse running in a cage.
- Each x_t has a latent representation u_t and $x_t \sim \mathcal{N}(\mu_\theta(u_t), \Sigma_\theta(u_t))$
    - Latent feature representation of the mouse video frames
- The u_t have linear dynamics $u_{t+1} = Z_{z_t} u_t + B\epsilon$, $\epsilon \sim \mathcal{N}(0, I)$
    - Latent features evolve smoothly, like the frames
- The dynamics Z_{z_t} depend on discrete state z_t and $z_{t+1} \sim \text{Cat}(\pi_{z_t})$
    - Discrete, interpretable mouse states (e.g., running; see yellow dot)

Lecture Outline

1. Probabilistic Reasoning
   - Motivational Example: Modeling Student Grades with Bayes Nets
   - Probabilistic Approach to Machine Learning, Box's Loop
   - In-Depth Example: Topic Modeling with Latent Dirichlet Allocation
2. Deep Generative Models & Probabilistic Modeling
   - Composing Probabilistic Models with Deep Neural Networks
   - Examples: GMMs, Deep Kalman Filters
3. Neural Networks & Probabilistic Inference
   - Black-Box Variational Inference
   - Implicit Variational Inference
4. Probabilistic Programming

Composing Probabilistic Models with Neural Networks

Probabilistic modeling and deep generative models are intimately connected.

1. Deep generative models are a notable example of the probabilistic approach (*previous sections*).
2. Deep generative models can serve as composable building blocks within probabilistic models (*previous sections*).
3. Deep neural networks can accelerate and automate inference and learning in probabilistic models (*this section*).

Classical Approaches to Variational Inference Are Hard

Consider a probabilistic model over x, z defined as

$$\begin{aligned} z &\sim p(z) & \text{e.g. } z &\sim Cat(\pi), \\ x|z &\sim p(z|x) & \text{e.g. } x &\sim \mathcal{N}(\mu_z, \Sigma_z) \end{aligned}$$

- The classical (pre-2010) approach to learning this model is mean-field variational inference.
  - This works only when the model is well-behaved (e.g., conjugate priors).
  - The form of $p(z|x)$ needs to be known; $q(z|x)$ must have same form; this lets us manually derive optimization updates for q.
- This is challenging for several reasons
  - It only works for certain types of models p
  - It imposes constraints on q (e.g., often q needs to be fully factored)
  - Deriving the update equations for q is very laborious and error-prone
  - In practice, mean-field gets stuck in local optima

Black-Box Variational Inference (Ranganath et al., 2014)

Consider a probabilistic model over x, z defined as

$$z \sim p(z) \qquad\qquad x|z \sim p(z|x)$$

In black-box variational inference, we instead parameterize $q(z|x)$ with a neural net and optimize the ELBO using SGD.

- This can look like VAE; also applies to other models (e.g., discrete z)

Parameterizing $q(z|x)$ with a neural net has many advantages:

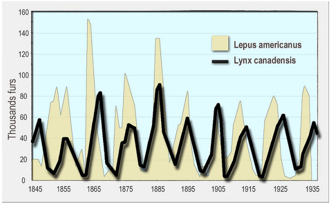
- **Flexiblity:** Works for any $p(x, z)$
- **Accuracy:** $q(z|x)$ is in a flexible class and can produce a tighter ELBO and approximate $p(z|x)$ better
- **Easy to deploy**: We can fit $q(z|x)$ by optimizing ELBO with SGD. We don't need to derive updates for q from p; hence we can perform VI with any p!
- **Easy to implement**: We can easily implement VI in Tensorflow/PyTorch – it's just SGD!
- **Speed:** Can leverage compute such as GPUs to optimize $q(z|x)$.

Implicit Models

Implicit models are ones that are defined by a (Bayesian) sampling process:

$$z \sim p(z) \qquad \theta \sim p(\theta) \qquad x = G_\theta(z)$$

We have a distribution over x, but no closed-form expression for $p(x, z, \theta)$.



Example: Lotka-Volterra (predator-prey) equations:

$$\frac{dx}{dt} = \alpha x - \beta y + \epsilon \qquad \frac{dy}{dt} = \delta xy - \gamma y + \epsilon$$

- x, y are the number of prey and predators, respectively; $\alpha, \beta, \delta, \gamma$ are parameters; we sample initial state x_0, y_0 from some prior; ϵ is Gaussian noise
- We can easily simulate (x, y)-trajectories but not compute their distribution

Implicit Variational Inference

How can we perform inference in implicit models? Treat it like a GAN!

Suppose want to learn $q(z|x)$ and $q(\theta)$. Let $q(x)$ denote the empirical distribution of data $\{x_i\}_{i=1}^n$. We can write the ELBO as:

$$\begin{aligned}\mathcal{L} &= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{q(\theta)q(z_i|x_i)}\left[\log p(x_i, z_i, \theta) - \log q(z_i|x_i) - \log q(\theta)\right] \\ &= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{q(\theta)q(z_i|x_i)}\left[\log \frac{p(x_i, z_i)}{q(z_i, x_i, \theta)} + \underbrace{\log q(x_i)}_{\text{const}}\right] \\ &= \mathbb{E}_{q(\theta)q(z_i, x_i)}\left[\log \frac{p(x_i, z_i, \theta)}{q(x_i, z_i, \theta)}\right] + \text{const}\end{aligned}$$

If we can estimate the log-ratio $\log \frac{p(x_i, z_i, \theta)}{q(x_i, z_i, \theta)}$, we can optimize the ELBO (Tran et al., 2017).

Implicit Variational Inference

How do we estimate the log-odds ratio $\log \frac{p(x_i, z_i, \theta)}{q(x_i, z_i, \theta)}$?
We train a discriminator $\sigma(r(x, z, \theta))$ to distinguish between real x_i, z_i, θ sampled from $p(x, z, \theta)$ and fake ones sampled from $q(x, z, \theta)$ (as in BiGAN), minimizing the cross-entropy

$$\mathcal{D} = \mathbb{E}_{p(x,z,\theta)} \left[\log \sigma(r(x, z, \theta))\right] + \mathbb{E}_{q(x,z,\theta)} \left[\log(1 - \sigma(r(x, z, \theta)))\right].$$

As we saw in our lectures on GANs, the optimal r^* is

$$r^*(x, z, \theta) = \log p(x, z, \theta) - \log q(x, z, \theta).$$

After we trained r^*, we learn q to optimize our estimate of the ELBO

$$\mathbb{E}_{q(x_i,z_i,\theta)} \left[r^*(x_i, z_i, \theta)\right].$$

The final estimate of the parameters is given by the approximate Bayesian posterior $q(\theta)$.

Lecture Outline

1. Probabilistic Reasoning
    - Motivational Example: Modeling Student Grades with Bayes Nets
    - Probabilistic Approach to Machine Learning, Box's Loop
    - In-Depth Example: Topic Modeling with Latent Dirichlet Allocation
2. Deep Generative Models & Probabilistic Modeling
    - Composing Probabilistic Models with Deep Neural Networks
    - Examples: GMMs, Deep Kalman Filters
3. Neural Networks & Probabilistic Inference
    - Black-Box Variational Inference
    - Implicit Variational Inference
4. Probabilistic Programming

Probabilistic Programming

In probabilistic programming, we specify probabilistic models using computer programs.

- The generative story of our model is specified by an arbitrary computer program
  - This is more general that standard math, e.g. can define control flows.
  - Can represent arbitrary computable data generation processes
- In practice, one uses domain-specific languages (PyMC3, Stan, Edward, Pyro)
  - Languages provide useful primitives (e.g., probability distributions)
  - They make it possible to automate inference for a given program
- Probabilistic programming holds a lot of promise for advancing ML
  - Analysts only need to specify the model
  - Inference engine allows instantly answering arbitrary inference queries
  - Very promising in science: can quickly experiment with many models

Probabilistic programming is still a very active research area.

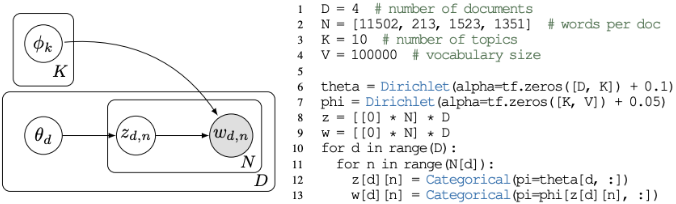
Deep Probabilistic Programming

Probabilistic programming also benefits for recent advances in deep learning. Modern deep probabilistic programming languages:

- Specify the sampling process via a deep neural network
- Use deep neural networks for accelerating inference (e.g., using Implicit VI)
- Use existing software and hardware to accelerate development speed and ease-of-use as well as execution and training speed

Edward (Tran et al., 2017)

Edward was an early deep probabilistic programming language built on top of Tensorflow.



```
D = 4  # number of documents
N = [11502, 213, 1523, 1351]  # words per doc
K = 10  # number of topics
V = 100000  # vocabulary size

theta = Dirichlet(alpha=tf.zeros([D, K]) + 0.1)
phi = Dirichlet(alpha=tf.zeros([K, V]) + 0.05)
z = [[0] * N] * D
w = [[0] * N] * D
for d in range(D):
  for n in range(N[d]):
    z[d][n] = Categorical(pi=theta[d, :])
    w[d][n] = Categorical(pi=phi[z[d][n], :])
```

Figure 11: Latent Dirichlet allocation (Blei et al., 2003).

- Programs as Tensorflow graphs with random variables at some nodes
- Inference engine takes another TF graph for q and automates VI
- Representation, inference, learning, sampling is built on top of TF and can arbitrarily mix neural network and classical components

Summary of the Lecture

- Generative models are a prominent example of *probabilistic reasoning*, a general approach to machine learning
  - Provides a principled way of specifying inductive biases
  - Three key tasks: representation, learning, and inference
  - Box's loop is a procedure for developing such models
- Classical probabilistic models can be composed with neural networks
  - We can manually encode the structure that we know (e.g., number of components)
  - We can let neural nets learn unknown structure (e.g., shape of components)
- Neural networks can also improve inference and learning
  - By using flexible neural networks for q, we can automate variational infernece
  - We can also learn more general models, such as implicit ones
- Probabilistic programming provide tools for developing probabilistic models