Lecture 3: Bayesian Modeling (Part 2)

Advanced Topics in Machine Learning

Dr. Tom Rainforth
January 24nd, 2020

rainforth@stats.ox.ac.uk
This lecture will focus on methods for constructing Bayesians models.

In particular, we will cover the following topics:

- Independence
- Graphical models
- Probabilistic programming
Independence
Events are **independent** if the occurrence of one event does not affect the probability of the other

- Equivalent definition for random variables
  - If $X$ and $Y$ are independent, we denote this as $X \perp \!\!\!\!\!\!\perp Y$
  - This is sometimes also known as **marginal independence**

If $X \perp \!\!\!\!\!\!\perp Y$, then $P(X = x) = P(X = x | Y = y)$ for all $x, y$.

- Independence is symmetric:

\[ X \perp \!\!\!\!\!\!\perp Y \implies P(Y = y) = P(Y = y | X = x) \]

- Through the product rule, we further have: if

\[ X \perp \!\!\!\!\!\!\perp Y \implies P(X, Y) = P(X)P(Y) \]
Examples of Independent Random Variables

- The outcomes of two flipped coins
- The speed of a car and whether a speed camera is malfunctioning
- We often assume independence between variables when constructing a model
  - Don’t forgot Bayesian probabilities are subjective beliefs
Conditional Independence

- Two events $A$ and $B$ are **conditionally independent** given event $C$ if $A$ and $B$ are independent given that $C$ occurred.
  - Again equivalent definition for random variables
  - If $X$ and $Y$ are conditionally independent given $Z$, we denote this as $X \perp \perp Y | Z$
- If $X \perp Y | Z$, then
  \[ P(X = x | Z = z) = P(X = x | Y = y, Z = z) \] for all possible $x$, $y$, and $z$.
  - This is again symmetric in $X$ and $Y$, but not $Z$
    \[ X \perp Y | Z \implies P(Y = y | Z = z) = P(Y = y | X = x, Z = z) \]
    \[ X \perp Y | Z \not\implies X \perp Z | Y \text{ or } Z \perp Y | X \]
- Conditional independence does not imply marginal independence or vice versa
  \[ X \perp Y | Z \not\implies X \perp Y, \quad X \perp Y \not\implies X \perp Y | Z \]
Conditional Independence Examples

- The amount of traffic and the number of umbrellas being used are not independent.
- But they are conditionally independent given it is raining.
- A speed camera being switched on and a car speeding are marginally independent.
- But they are not conditionally independent given the camera is triggered.
Why are Independence Relationships Important?

- (Conditional) Independence between variables is one of the most important **modeling assumptions** we can make when constructing Bayesian models.

- The **chain rule** is an extension of the product rule that allows us to break down any joint distribution into a product over conditional probability distributions:

\[
P(X_{1:N}) = P(X_1)P(X_2|X_1) \ldots P(X_N|X_1, X_2, \ldots, X_{N-1})
\]

\[
= P(X_1) \prod_{n=2}^{N} P(X_n|X_{1:n-1}) \tag{1}
\]

- Note we can use any ordering of the variables we want and the chain rule also applies densities, e.g.

\[
p(x_{1:N}) = p(x_N) \prod_{n=1}^{N-1} p(x_n|x_{n+1:N})
\]
Why are Independence Relationships Important? (2)

- Different orderings of the variables in the joint breakdown are known as **factorizations** of the joint.
- Independence assumptions allow us to make simplifications that show up in the factorizations.
- For example, the **Markov property**

\[ X_n \perp X_1:N \setminus X_n \mid X_{n-1}, X_{n+1} \]  (2)

allows us to factorize the joint as

\[ P(X_1:N) = P(X_1) \prod_{n=2}^{N} p(X_n \mid X_{n-1}) \]  (3)
Example: Flipping Lots of Coins

- We flip $N$ independent coins
- The joint space of possible outcomes is $2^N$ which becomes impractical to deal with for even moderate $N$
- Utilizing the known independence, we can reason about each outcome separately such that the output space is only $2 \times N$
We often want to assume datapoints are independent given a generative model
• Some factorizations allow for more simplification than others as they incorporate more information about independences
• Example: if $X_1 \perp \perp X_3 | X_2$ then the factorization
  \[ P(X_1, X_2, X_3) = P(X_1)P(X_2|X_1)P(X_3|X_2) \]
  is simpler than
  \[ P(X_1, X_2, X_3) = P(X_1)P(X_3|X_1)P(X_2|X_1, X_3) \]
• Critically though, we typically do not know all factorizations of a model when we construct it
  • For Bayesian problems we have direct access to $p(\theta)$ and $p(D|\theta)$ but not $p(D)$ and $p(\theta|D)$: computation is required to uncover them
  • We can thus think about Bayesian inference as finding unknown factorizations of a joint
Graphical Models
What is a Graphical Model?

- Generative models typically have many variables with a complex dependency structure.
- **Graphical models** are a ubiquitous method for representing and reasoning about generative models, with a particular focus on the dependency structure.
- They are formed by a number of connected nodes, where each node represents a random or observed variable in the model.
- Links between nodes represent dependencies: any two connected nodes have an explicit dependency, though unconnected nodes may still be dependent.
Graphical models can be separated into two distinct classes: **directed graphical models** and **undirected graphical models**.
Undirected Graphical Models

- Undirected graphical models, also known as Markov random fields, imply no ordering to variables and are used only to express conditional independences.
- They are useful for representing models that are difficult to express in a generative manner.
- Independence in undirected graphical models can be deduced through the global Markov property:
  - Two variables $A$ and $B$ are conditionally independent given a subset of other variables $C$ if there is no path between $A$ and $B$ that does not pass through $C$.
  - This implies that each variable is conditionally independent of all the other variables given its neighbors.
- We will not cover these in detail—see C M Bishop. *Pattern recognition and machine learning.* 2006, Section 8.3 instead.
Predicting whether it will rain at various locations does not have a natural generative model.

But the probability that it rains at one location is correlated to whether it rains at neighboring locations.
Bayesian Networks / DAGs

- We’ll focus on **directed acyclic graphical models** (DAGs), i.e. directed graphs containing no cycles or loops.
- DAGs, also known as **Bayesian networks**, define a factorization for a generative model.

\[
p(a, b, c, d, e, f) = p(a)p(b)p(c|a, b)p(d|c)p(f|c, d)p(e|f)
\]
To define a model using a DAG, we need to be able to define the probability of each variable given its parents (i.e., the nodes with arrows pointing towards our current node).

For any node in a DAG containing the variables $x_{1:n}$

$$p(x_n|x_{1:N}\setminus x_n) = p(x_n|\text{parents}(x_n))$$ (4)
D'YA LIKE DAGS?
Why Ya Should Like DAGs

- DAGs allows us to provide a piecewise explanation for the generative process
- They allow us to combine simple, local distributions into a larger, overall generative process
- This makes them a useful way to describe and construct Bayesian models

Image Credit: Pieter Abbeel
Imagine a medical diagnostic problem where we wish to predict if a patient has lung cancer.

Let $f$ denote unknown lifestyle and genetic factors of a hypothetical patient (e.g. whether they smoke)

We encode this distribution of these as $p(f)$

Our DAG just starts with a single node $f$
Example: Medical Diagnosis

- Given $f$, we can develop a model for the probability that a patient will develop lung cancer.
- We encode this through the conditional distribution $p(c|f)$ where $c = 1$ indicates cancer is present.
- We thus add a node $c$ to the DAG and an arrow from $f$ to $c$ to represent this conditional dependency.

\[
p(f)p(c|f)
\]
Example: Medical Diagnosis

- Given \( f \) and \( c \), we can predict what symptoms, \( s \), might be observed, e.g. a persistent cough.
- We encode this through the conditional distribution \( p(s|f, c) \) and add the new node \( s \) to the DAG.
- We now have the complete DAG which provides a factorization of our model:

\[
p(f)p(c|f)p(s|c, f)
\]
Observed Nodes

- The power DAGs becomes apparent once we start considering **observing** nodes.
- This equivalent to conditioning in a Bayesian model: we fix a node to take on a certain observed value.
- Observed nodes in a DAG are denoted through shading.
- We can use the DAG to reason about unobserved variables conditioned on the observed ones.
• The power DAGs becomes apparent once we start considering **observing** nodes

• This equivalent to conditioning in a Bayesian model: we fix a node to take on a certain observed value

• Observed nodes in a DAG are denoted through shading

• We can use the DAG to reason about unobserved variables conditioned on the observed ones

\[ p(c | d, e) \propto p(a)p(b)p(c | a, b)p(d | c)p(e | c) \]
For our medical diagnosis example we actually observe the symptoms. We can thus perform Bayesian inference to try and infer whether the patient has cancer. In general, posteriors are not available in closed form and DAGs provide a useful tool for providing helpful information to calculate them: see Bishop Chapter 8.
Explaning Away

- Sometimes conditioning can introduce dependencies we might not at first expect
- Consider a model for speed camera being triggered, $t$
- This depends on both whether the camera is malfunctioning, $m$, and whether the car is speeding, $s$, as per the DAG on the right
- $m$ and $s$ are clearly marginally independent
- But if $t$ is observed they become dependent

$p(m)p(s)p(t|m, s)$

\[ p(m)p(s)p(t|m, s) \]
Explaining Away

- This phenomenon is known as **explaining away**
- Here $m$ and $s$ provided alternative explanations for $t$
- For example, if the camera is triggered, then the car must be speeding or the camera must either be malfunctioning
- If the camera is not triggered, we cannot have both that the car is not speeding and the camera is working

$$p(m)p(s)p(t|m,s)$$
Alarm Triggering Example

Variables

- B: Burglary
- A: Alarm goes off
- M: Mary calls
- J: John calls
- E: Earthquake!

Example and image credit: Pieter Abbeel
Alarm Triggering Example (2)

Example and image credit: Pieter Abbeel
Optional homework: use this Bayes net to calculate:

\[ P (\text{John Calls}|\text{Burglary}) \] (5)
\[ P (\text{Burglary}|\text{John Calls}) \] (6)
\[ P (\text{Earthquake}|\text{Mary Calls, Burglary}) \] (7)

If you’re looking for a challenge (optional), try your hand at this homework assignment from Frank Wood:

http://www.robots.ox.ac.uk/~fwood/teaching/AIMS_CDT_ML_2016/homework/HW_1_sum_product/

Note this will require substantial extra reading (Bishop Chapter 8) and most likely a few hours to implement.
• How can we deduce the independence relationships from a DAG?

• d-separation gives us rules for doing this

• Consider three arbitrary, non-intersecting, subsets $A$, $B$, and $C$ of a DAG.

• $A$ and $B$ are conditionally independent given $C$ if there are no unblocked paths from $A$ to $B$ when $C$ is observed
  • $A$ is said to be d-separated from $B$ by $C$
• Paths do not need to be in the same directions as the arrows in the DAG—we just have to move between connected nodes
• A path between $A$ and $B$ is blocked if
  1. There is an unobserved node, $n$, in the path where both the arrows point towards $n$ and $n$ has no observed descendants
  2. Consecutive arrows in the path meet at an observed node and either one or both of them points away from the node.
• Note that only the first of these rules is necessary for establishing marginal independence
Is the path between $a$ and $b$ blocked?

Rule 1  There is an unobserved node, $n$, in the path where both the arrows point towards $n$ and $n$ has no observed descendants
Rule 2  Consecutive arrows in the path meet at an observed node and either one or both of them points away from the node.
Is the path between $a$ and $b$ blocked?

- **Unblocked**
- **Blocked: 2nd rule**
- **Unblocked**

**Rule 1** There is an unobserved node, $n$, in the path where both the arrows point towards $n$ and $n$ has no observed descendants.

**Rule 2** Consecutive arrows in the path meet at an observed node and either one or both of them points away from the node.
Is the path between $a$ and $b$ blocked?

Rule 1: There is an unobserved node, $n$, in the path where both the arrows point towards $n$ and $n$ has no observed descendants.

Rule 2: Consecutive arrows in the path meet at an observed node and either one or both of them points away from the node.
Is the path between $a$ and $b$ blocked?

Rule 1 There is an unobserved node, $n$, in the path where both the arrows point towards $n$ and $n$ has no observed descendants

Rule 2 Consecutive arrows in the path meet at an observed node and either one or both of them points away from the node.
We finish our introduction to graphical models by considering a common model where the dependency structure is very important: **Hidden Markov Models** (HMMs)

- Bishop has almost a whole chapter dedicated to these (Chapter 13)
As we see from its DAG, a HMM has $T$ latent variables $x_{1:T}$ and $T$ observations $y_{1:T}$. The joint distribution is as follows

$$p(x_{1:T}, y_{1:T}) = p(x_1)p(y_1|x_1) \prod_{t=2}^{T} p(x_t|x_{t-1})p(y_t|x_t), \quad (8)$$

where each $x_t$ is independent of $x_{1:t-2}$ and $y_{1:t-1}$ given $x_{t-1}$, and of $x_{t+2:T}$ and $y_{t+1:T}$ given $x_{t+1}$.

This Markov property means the model has no memory as information is passed forwards (or backwards) only through the value of the previous (or next) latent state:

Many dynamical systems obey the Markov property: HMMs are extensively used for a number of tasks involving sequential data, such as tracking and DNA sequencing.
Probabilistic Programming
Captchas Revisited

Security check

To proceed, please enter the security code below and click "Submit".

Enter security code

By clicking Submit I acknowledge the Terms and Conditions for use of the connectivity service(s)

Submit

Can't read the characters?

Refresh Image
We have explored making use of deep neural networks and the processing speed of deep neural networks for generative modeling. The technique presented here provides a framework within which importance sampling. The amortized inference technique parameterizes proposal distributions during sequential inference. At test time an amortized inference architecture transforms a problem given in the form of a probabilistic program into a trained neural network architecture that parameterizes proposal distributions during sequential inference. At test time an amortized inference architecture transforms a problem given in the form of a probabilistic program into a trained neural network architecture that parameterizes proposal distributions during sequential inference.

Table 1: Captcha recognition rates.

<table>
<thead>
<tr>
<th>Method</th>
<th>2011 Baidu</th>
<th>2013 Baidu</th>
<th>eBay</th>
<th>Yahoo reCaptcha</th>
<th>Wikipedia</th>
<th>Facebook</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our method</td>
<td>99.8%</td>
<td>99.9%</td>
<td>99.2%</td>
<td>98.4%</td>
<td>96.4%</td>
<td>93.6%</td>
</tr>
</tbody>
</table>

Starostenko et al. (2015) 91.5% 54.6%
Probabilistic Programs are Simulator–Based Models

Inference

Parameters

Program

Output

Scientific Simulation

Simulator

\[ p(y|x)p(x) \]

\[ p(x|y) \]

\[ p(x) \]

\[ p(y|x) \]

\[ y \]

\[ x \]

Statistics
Probabilistic Programs are Simulator-Based Models

- Probabilistic Programming
- Inference
- Parameters
- Program
- Observations
- Output

Scientific Simulation

Probabilistic Programming

Statistics

\[ p(y|x)p(x) \]

\[ p(x|y) \]

\[ p(y|x) \]
Example: Deep Sea Oil Pipes
Example: Deep Sea Oil Pipes (2)

\[ p(\theta) \]

\[ p(D|\theta) \]
Application: Scene Perception

Abstract

We present a method for controlling the output of procedural modeling programs using Sequential Monte Carlo (SMC). Previous probabilistic methods for controlling procedural models use Markov Chain Monte Carlo (MCMC), which receives control feedback only for completely-generated models. In contrast, SMC receives feedback incrementally on incomplete models, allowing it to reallocate computational resources and converge quickly. To handle the many possible sequentializations of a structured, recursive procedural modeling program, we develop and prove the correctness of a new SMC variant, Stochastically-Ordered Sequential Monte Carlo (SOSMC). We implement SOSMC for general-purpose programs using a new programming primitive: the stochastic future. Finally, we show that SOSMC reliably generates high-quality outputs for a variety of programs and control scoring functions. For small computational budgets, SOSMC's outputs often score nearly twice as high as those of MCMC or normal SMC.

CR Categories:

I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems;

Keywords:

Procedural Modeling, Directable Randomness, Probabilistic Programming, Sequential Monte Carlo

Introduction

Procedural modeling has long been used in computer graphics to generate varied, detailed content with minimal human effort. Procedural models for trees, buildings, cities, and decorative patterns enrich the virtual worlds of movies and games [Méch and Prusinkiewicz 1996; Müller et al. 2006; Wong et al. 1998]. Ambitious new projects aim to produce fully-procedural, galactic-scale environments for players to explore [Procedural Reality 2014; Hello Games 2014]. This expressive power comes at a cost: procedural models often use complex, recursive control logic, resulting in emergent behavior which is difficult to direct. As a result, technical artists often must tweak parameters and massage initial conditions to achieve a desired look. This time and effort may defeat the purpose of using procedural modeling in the first place.

Fortunately, recent years have seen advances in the use of probabilistic inference techniques to control procedural models [Talton et al. 2011; Stava et al. 2014; Yeh et al. 2012]. Viewing a procedural model as sampling from a probability distribution allows for the application of Bayesian inference techniques: the prior is the procedural model itself, and the likelihood is some high-level control expressed as a scoring function.

This previous work relies on Markov Chain Monte Carlo (MCMC), but other Bayesian posterior sampling algorithms are available: another popular choice is Sequential Monte Carlo (SMC). SMC uses a set of samples, or particles, to represent a distribution that changes over time as new evidence is observed. As the distribution changes, SMC shifts more particles (and thus more of its computational budget) to higher-probability regions of the state space. For probabilistic models that fit this pattern of ‘evidence arriving over time,’ such as modeling the location of a mobile robot, SMC is often the method of choice: the incremental evidence it receives provides feedback early and often, allowing it to converge quickly [Doucet et al. 2001]. In contrast, MCMC receives feedback only after running through the entire model.

Application: Guided Object Generation

Forward Sampling  
SOSMC-Controlled Sampling

What is a Probabilistic Programming System?

- Probabilistic programming systems (PPSs) allow Bayesian models to be represented in the form of a stochastic simulator and statements for conditioning on data.
- Informally, one can think of the generative model as the definition of a prior, the conditioning statements as the definition of a likelihood, and the output of the program as samples from a posterior distribution.
- Their core philosophy is to decouple model specification and inference.
- Inference is automated for any model the user might write.
Separating Models from Inference

Programming Language Representation / Abstraction Layer

Algorithm 1 Sequential Monte Carlo (all for $i = 1, \ldots, N$)

1. **Input:** data $y_1:T$, number of particles $N$, proposals $q_i$
2. $x_1^i \sim q_i(x_1)$
3. $u_1^i = \frac{f_i(x_1)}{q_i(x_1)}$
4. for $t = 2$ to $T$
5. \[ a_{t-1}^i \sim \text{Discrete}\left\{u_{t-1}^i\right\}_{i=1}^N \]
6. $x_t^i \sim q_i(x_t|x_{t-1}^i)$
7. Set $x_{t-1}^i = (x_{t-1}^i, x_t^i)$
8. $u_t^i = \frac{f_i(x_t)}{q_i(x_t|x_{t-1}^i)}$
9. end for

Algorithm 3 iP/MCMC sampler

1. **Input:** number of nodes $M$, conditional nodes $P$ and MCMC steps $N$, initial $x_1^0$.
2. for $r = 1$ to $R$
3. Workers $1:M$ run Algorithm 1 (SMC)
4. Workers $1:P$ run Algorithm 2 (CSMC), conditional on $x_{t-1}^r[r-1]$ respectively.
5. for $j = 1$ to $P$
6. Select a new conditional node by simulating $c_j$ according to (5).
7. Set new MCMC sample $x_j^r[r] \sim x_j^r$ by simulating $b_j$ according to (7).
8. end for
9. end for

Simulator = User’s Responsibility

Inference Engine(s) = Developer’s Responsibility
Using Information in the Code

\begin{align*}
\textbf{Inputs:} & \text{ Transition std-dev } \sigma, \text{ output shape } \\
& \text{ } \alpha, \text{ output rate } \beta, \text{ data } y_{1:T} \\
1: & \quad x_0 \leftarrow 0 \\
2: & \quad \text{tr-dist } \leftarrow \text{normal}(0, \sigma) \\
3: & \quad \text{obs-dist } \leftarrow \text{gamma}(\alpha, \beta) \\
4: & \quad \text{for } t = 1, \ldots, T \text{ do} \\
5: & \quad \quad x_t \leftarrow x_{t-1} + \text{sample}(\text{tr-dist}) \\
6: & \quad \quad \text{observe}(\text{obs-dist}, y_t - x_t) \\
7: & \quad \quad z_t \leftarrow \mathbb{I}(x_t > 4) \\
8: & \quad \text{end for} \\
9: & \quad \text{return } z_{1:T}
\end{align*}

\[ p(x_{1:T}, y_{1:T}|\sigma, \alpha, \beta) = \]
\[ \mathcal{N}(x_1; 0, \sigma^2) \text{ GAMMA}(y_1 - x_1; \alpha, \beta) \]
\[ \prod_{t=2}^{T} \mathcal{N}(x_t - x_{t-1}; 0, \sigma^2) \text{ GAMMA}(y_t - x_t; \alpha, \beta) \]
Worked Example: Bayesian Linear Regression

**Inputs:** Input points $u_1:N$

1. $m \leftarrow \text{sample (normal} (0,1))$
2. $c \leftarrow \text{sample (normal} (0,1))$
3. for $n = 1, \ldots, N$ do
4. \hspace{1em} $\mu_n \leftarrow mu_n + c$
5. \hspace{1em} $v_n \leftarrow \text{sample (normal} (\mu_n, 0.1))$
6. end for
7. return $m, c, v_{1:N}$
Worked Example: Bayesian Linear Regression

**Inputs:** Data $u_{1:N}, v_{1:N}$

1: $m \leftarrow \text{sample} \left( \text{normal} \ (0,1) \right)$
2: $c \leftarrow \text{sample} \left( \text{normal} \ (0,1) \right)$
3: for $n = 1, \ldots, N$ do
   4: \hspace{1em} $\mu_n \leftarrow mu_n + c$
   5: \hspace{1em} observe $\left( \text{normal} \ (\mu_n, 0.1), v_n \right)$
4: end for
5: return $m, c$

\[
\theta = \{m, c\} \quad \mathcal{D} = \{u_n, v_n\}_{n=1}^{N}
\]
Probabilistic programming is more of an umbrella term than an exactly defined approach.

There are two philosophies systems are built around:

- **Inference driven** PPSs start with a specific inference method(s) and build a language around making it as easy as possible to write models suitable for that method(s).
- **Model driven** PPSs start with language capable of defining any computable distribution and try and construct inference engines capable of working on any program.
  - These are sometimes based on Turing–complete languages, in which case they are known as **universal** PPSs.
Universal PPSs allow Stochastic Control Flow

**Inputs:** Event rate $\lambda$

1. $L \leftarrow \exp(-\lambda)$, $k \leftarrow 0$, $p \leftarrow 1$
2. **while** $p > L$ **do**
3. $u \leftarrow \text{sample (uniform (0,1))}$
4. $p \leftarrow pu$
5. **if** $p \leq L$ **then**; **break** **while**; **end if**
6. **observe** (bernoulli (0.2), 1)
7. $k \leftarrow k + 1$
8. **end while**
9. **observe** (bernoulli (0.99), $\mathbb{I}(k>3)$)
10. **return** $k$
Example PPSs

Probabilistic Programming
Tom Rainforth
Languages
Tu r i n g
Zhusuan
Pyro ProbTorch
Edward

Dynamic Support
Pyro Turing ProbTorch
webPPL Probabilistic C
Venture Anglican

Church Infer.NET
Factorie Zhusuan
Edward

LibBi
STAN
PyMC
JAGS

WinBUGS
Static
Support

BUGS
Deep PPSs

Edward

PROB
TORCH

PYRO

ZHUSUAN

1. \( W_0 = \text{Normal}(\text{mu}=\text{tf.zeros}([D, H]), \text{sigma}=\text{tf.ones}([D, H])) \)
2. \( W_1 = \text{Normal}(\text{mu}=\text{tf.zeros}([H, 1]), \text{sigma}=\text{tf.ones}([H, 1])) \)
3. \( b_0 = \text{Normal}(\text{mu}=\text{tf.zeros}(H), \text{sigma}=\text{tf.ones}(L)) \)
4. \( b_1 = \text{Normal}(\text{mu}=\text{tf.zeros}(1), \text{sigma}=\text{tf.ones}(1)) \)
5. \( x = \text{tf.placeholder}(\text{tf.float32}, [N, D]) \)
6. \( y = \text{Bernoulli}(\text{logits}=\text{tf.matmul}(\text{tf.nn.tanh}(\text{tf.matmul}(x, W_0) + b_0), W_1) + b_1) \)

---

Recap

- Conditional independences are a very important modeling assumption.
- Graphical models, and in particular DAGs, allow us to construct complex generative models by piecing together components of the model.
- Observing nodes in graph allow us to condition on observations and thus perform Bayesian reasoning.
- Probabilistic programming gives us a means of encoding Bayesian models as code.
- It further provides automated inference engines to calculate the posterior.

Next time: Bayesian inference—actually calculating posteriors.
Further Reading

- The lecture notes provide a more thorough and technical introduction to probabilistic programming

- Eric Xing’s course on Probabilistic Graphical Model: http://www.cs.cmu.edu/~epxing/Class/10708-14/lecture.html

- Pieter Abbeel and Dan Klein on Bayes Nets (Lectures 16 to 19 in the recommended list): http://ai.berkeley.edu/lecture_videos.html

- Bishop, *Pattern recognition and machine learning*, Chapters 8

- D Barber. *Bayesian reasoning and machine learning*. 2012, Chapters 2-4

- Video tutorial on probabilistic programming by Frank Wood: https://www.youtube.com/watch?v=Te7A5JEm5UI&t=500s

- Full conference of talks on probabilistic programming: https://www.youtube.com/channel/UCTFdB7aQY1ewBYwJJrpKp6Q

- Websites for individual PPSs (Pyro is a good place to start)